

# **ENCYCLOPEDIA OF COMPUTER SCIENCE AND TECHNOLOGY**

**2**

**EXECUTIVE EDITORS**

**Jack Belzer  
Albert G. Holzman  
Allen Kent**









---

---

**ENCYCLOPEDIA OF  
COMPUTER SCIENCE  
AND TECHNOLOGY**

---

---

**VOLUME 2**

# INTERNATIONAL EDITORIAL ADVISORY BOARD

- SHUHEI AIDA, Tokyo, Japan  
J. M. BENNETT, Sydney, Australia  
DOV CHEVION, Jerusalem, Israel  
LUIGI DADDA, Milan, Italy  
RUTH M. DAVIS, Washington, D.C.  
A. S. DOUGLAS, London, England  
LESLIE C. EDIE, New York, New York  
S. E. ELMAGHRABY,  
Raleigh, North Carolina  
A. P. ERSHOV, Novosibirsk, U.S.S.R.  
HAROLD FLEISHER,  
Poughkeepsie, New York  
BRUCE GILCHRIST,  
New York, New York  
V. M. GLUSHKOV, Kiev, U.S.S.R.  
C. C. GOTLIEB, Toronto, Canada  
EDWIN L. HARDER,  
Pittsburgh, Pennsylvania  
GRACE HOPPER, Washington, D.C.  
A. S. HOUSEHOLDER,  
Knoxville, Tennessee  
MANFRED KOCHEN,  
Ann Arbor, Michigan  
E. P. MILES, JR., Tallahassee, Florida  
JACK MINKER, College Park, Maryland  
DON MITTELMAN, Oberlin, Ohio  
W. J. POPPELBAUM, Urbana, Illinois  
A. ALAN B. PRITSKER,  
Lafayette, Indiana  
P. RABINOWITZ, Rehovot, Israel  
JEAN E. SAMMET,  
Cambridge, Massachusetts  
SVERRE SEM-SANDBERG,  
Stockholm, Sweden  
J. C. SIMON, Paris, France  
WILLIAM A. SMITH, JR.,  
Bethlehem, Pennsylvania  
T. W. SZE, Pittsburgh, Pennsylvania  
RICHARD I. TANAKA,  
Anaheim, California  
DANIEL TEICHROEW,  
Ann Arbor, Michigan  
ISMAIL B. TURKSEN, Toronto, Canada  
MURRAY TUROFF, Newark, New Jersey  
MICHAEL S. WATANABE,  
Honolulu, Hawaii

---

# ENCYCLOPEDIA OF COMPUTER SCIENCE AND TECHNOLOGY

---

EXECUTIVE EDITORS

*Jack Belzer   Albert G. Holzman   Allen Kent*

UNIVERSITY OF PITTSBURGH  
PITTSBURGH, PENNSYLVANIA

QA  
76.15  
E5  
V.2  
Science  
Ref.

VOLUME 2  
*AN/FSQ to Bal*

MARCEL DEKKER, INC. • NEW YORK

COPYRIGHT © 1975 by MARCEL DEKKER, INC.  
ALL RIGHTS RESERVED

Neither this book nor any part may be reproduced or  
transmitted in any form or by any means, electronic  
or mechanical, including photocopying, microfilming,  
and recording, or by any information storage and  
retrieval system, without permission in writing from  
the publisher.

MARCEL DEKKER, INC.  
270 Madison Avenue, New York, New York 10016

LIBRARY OF CONGRESS CATALOG CARD NUMBER: 74-29436  
ISBN: 0-8247-2252-3

Current printing (last digit):  
10 9 8 7 6 5 4 3 2 1

PRINTED IN THE UNITED STATES OF AMERICA

---

## CONTENTS OF VOLUME 2

---

CONTRIBUTORS TO VOLUME 2	v		
AN/FSQ-7 COMPUTER	Edward Wenzel	1	
ANALOG-DIGITAL CONVERSION		Barbara Stephenson	7
ANALOG SIGNALS AND ANALOG DATA PROCESSING		Walter J. Karplus	22
ANALYSIS OF VARIANCE		H. Ginsburg	52
AND-OR-NOT-NAND-NOR LOGIC		Marlin H. Mickle	75
APL TERMINAL SYSTEM		Harry Katzan, Jr.	93
APPROXIMATION METHODS		George D. Byrne	136
ARGONNE NATIONAL LABORATORY		M. K. Butler	141
ARITHMETIC OPERATIONS		Charles E. Donaghey, Jr.	180
ARPA NETWORK		Howard Frank	197
ARRAYS		George F. Badger, Jr.	217
ARTIFICIAL INTELLIGENCE		B. Chandrasekaran and M. C. Yovits	225
ARTIFICIAL LANGUAGES		Jean E. Sammet and R. Tabory	245
ASCII CODE		Sumner E. Mendelson	272
ASSOCIATION FOR COMPUTING MACHINERY (ACM)		Herbert S. Bright	278
ASSOCIATIVE MEMORIES AND PROCESSORS: A DESCRIPTION AND APPRAISAL		Jack Minker	283
ASTRONOMY		Frank Bartko	325
ATOMIC ENERGY COMMISSION (ENERGY RESEARCH AND DEVELOPMENT ADMINISTRATION)		Sidney Fernbach	347
AUDITION		Lawrence F. Feth	361
AUERBACH CORPORATIONS		Milton Strassberg	379
AUTOCORRELATION		J. R. Zelonka	384
AUTOMATA THEORY		Michael A. Arbib	388

## CONTENTS OF VOLUME 2

AUTOMATIC INDEXING: PROGRESS AND PROSPECTS	Bertrand C. Landry and James E. Rush	403
AUTOMATICALLY PROGRAMMED TOOLS— NUMERICALLY CONTROLLED MACHINE TOOLS	J. L. Goodrich and Robert D. Thrush	448
AUTOMATION	John M. Evans, Jr.	461
BABBAGE, CHARLES	B. Sendov	469
BIVALENT PROGRAMMING BY IMPLICIT ENUMERATION	Egon Balas	479

---

## CONTRIBUTORS TO VOLUME 2

---

MICHAEL A. ARBIB, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts: *Automata Theory*

GEORGE F. BADGER, JR., Computing Services Office, University of Illinois, Urbana, Illinois: *Arrays*

EGON BALAS, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, Pennsylvania: *Bivalent Programming by Implicit Enumeration*

FRANK BARTKO, Martin Marietta Corporation, Denver, Colorado: *Astromomy*

HERBERT S. BRIGHT, M.S.E.E., Computation Planning, Inc., Washington, D.C.: *Association for Computing Machinery (ACM)*

M. K. BUTLER, Applied Mathematics Division, Argonne National Laboratory, Argonne, Illinois: *Argonne National Laboratory*

GEORGE D. BYRNE, Department of Mathematics, University of Pittsburgh, Pittsburgh, Pennsylvania: *Approximation Methods*

B. CHANDRASEKARAN, Department of Computer and Information Science, The Ohio State University, Columbus, Ohio: *Artificial Intelligence*

CHARLES E. DONAGHEY, JR., Industrial Engineering Department, University of Houston, Houston, Texas: *Arithmetic Operations*

JOHN M. EVANS, JR., Institute for Computer Sciences and Technology, National Bureau of Standards, Gaithersburg, Maryland: *Automation*

SIDNEY FERNBACH, Lawrence Livermore Laboratory, Livermore, California: *Atomic Energy Commission (Energy Research and Development Administration)*

LAWRENCE L. FETH, Ph.D., Department of Audiology and Speech Sciences, Purdue University, West Lafayette, Indiana: *Audition*

HOWARD FRANK, Network Analysis Corporation of Glen Cove, New York: *ARPA Network*

H. GINSBURG, Bettis Atomic Power Laboratory, Westinghouse Electric Corporation, Pittsburgh, Pennsylvania: *Analysis of Variance*

J. L. GOODRICH, Instructor, Department of Industrial and Management Systems Engineering, College of Engineering, The Pennsylvania State University, University Park, Pennsylvania: *Automatically Programmed Tools—Numerically Controlled Machine Tools*

WALTER J. KARPLUS, Professor and Chairman, Computer Science Department, University of California, Los Angeles, California: *Analog Signals and Analog Data Processing*

HARRY KATZAN, JR., Computer Science Department, Pratt Institute, Brooklyn, New York: *APL Terminal System*

BERTRAND C. LANDRY, Martin Marietta Corporation, Denver Colorado, *Automatic Indexing: Progress and Prospects*

SUMNER E. MENDELSON, Falk Library, University of Pittsburgh, Pittsburgh, Pennsylvania: *ASCII Code*

MARLIN H. MICKLE, Department of Electrical Engineering, University of Pittsburgh, Pittsburgh, Pennsylvania: *AND-OR-NOT-NAND-NOR LOGIC*

JACK MINKER, Professor and Chairman, Department of Computer Science, University of Maryland, College Park, Maryland: *Associative Memories and Processors: A Description and Appraisal*

JAMES E. RUSH, Director, Research and Development, The Ohio College Library Center, Columbus, Ohio: *Automatic Indexing: Progress and Prospects*

JEAN E. SAMMET, Manager, Programming Language Technology, IBM Corporation, Federal Systems Division, Cambridge, Massachusetts: *Artificial Languages*

B. SENDOV, Professor, Bulgarian Academy of Science, Sofia, Bulgaria: *Babbage, Charles*

BARBARA STEPHENSON, Digital Equipment Corporation, Maynard, Massachusetts: *Analog-Digital Conversion*

MILTON STRASSBERG, Manager, Marketing Services, Auerbach Publishers, Inc., Philadelphia, Pennsylvania: *Auerbach Corporations*

R. TABORY, IBM Corporation, Mohansic, Yorktown Heights, New York: *Artificial Languages*

ROBERT D. THRUSH, Electro-Mechanical Division, Westinghouse Electric Corporation, Cheswick, Pennsylvania: *Automatically Programmed Tools—Numerically Controlled Machine Tools*

EDWARD WENZEL, University of Pittsburgh, Pittsburgh, Pennsylvania: *AN/FSQ-7 Computer*

M. C. YOVITS, Department of Computer and Information Science, The Ohio State University, Columbus, Ohio: *Artificial Intelligence*

J. R. ZELONKA, Department of Industrial Engineering, University of Pittsburgh, Pittsburgh, Pennsylvania: *Autocorrelation*



---

---

ENCYCLOPEDIA OF  
COMPUTER SCIENCE  
AND TECHNOLOGY

---

VOLUME 2



# **AN/FSQ-7 COMPUTER**

## **INTRODUCTION**

### **Systems Overview**

The AN/FSQ-7 digital computer was the data processing heart of SAGE (Semi-Automatic Ground Environment), a system of electronic air defense developed by the Rand Corporation of Santa Monica, California, for the United States Air Force. The SAGE system began as Project Rand shortly after World War II when the project's researchers were commissioned to study the economics and technology involved in developing a man-machine air defense system utilizing the then state-of-the-art electronics available. The research and development work was accomplished at Lincoln Laboratories of the Massachusetts Institute of Technology. System hardware and software design and implementation was a multicontract cooperative effort. The basic ideas of the system resulted from the efforts of Drs. G. E. Valley and J. W. Forrester of M.I.T. Many organizations contributed to the development of SAGE. The International Business Machine Corporation designed, manufactured, and installed the AN/FSQ-7 combat direction center and AN/FSQ-8 combat control computer installations. The Western Electric Company, Inc. provided management services and the design and construction of the direction center and combat center buildings. A subcontractor, Bell Telephone Laboratories, assisted in the performance of the Western Electric Company services. The Burroughs Corporation manufactured, installed, and provided logistic support for the AN/FST-2 coordinate data transmitting sets. The Systems Development Corporation (SDC), formed from the System Development Division of the Rand Corporation, assisted Lincoln Laboratories in the preparation of the software program system for both the direction center and combat center complexes. The SDC assumed full responsibility for the software systems in the late 1950s, a responsibility that they retained for the duration of the SAGE system. SDC also assumed the responsibility of development of training programs for each air defense station that would both exercise the system and keep the Air Force manning staffs sharp and improve training methods.

The SAGE system became operational in 1957. Various considerations including changing technology and continuing reassessment of the strategic value of SAGE resulted in its obsolescence in the mid-1960s. The final complement of direction centers fell short of the originally planned 32 stations. Original estimates of the cost of each installation was \$20 million. Undoubtedly the experience gained in computer technology by the participating contractors of SAGE has contributed significantly to the advancement of the state of the art of computer science.

### The SAGE System Complex

SAGE direction centers, each containing duplexed AN/FSQ-7 digital computers, were built, installed, and made operational at strategic sites around the perimeter of the United States. SAGE combat centers, each containing duplexed AN/FSQ-8 digital computers, were also installed. The combat centers were higher echelon divisional data gathering and summarization headquarters for two or more direction centers. Combat centers displayed, monitored, and condensed the air defense picture for their respective direction centers, transmitted tactical directives to these direction centers, and forwarded the divisional air defense status to the North American Air Defense Headquarters (NORAD) in Colorado Springs, Colorado.

### The SAGE System Mission

At the end of World War II, United States defense planners assessed that the immediate defense threat would be air attacks by intercontinental bombers that could deliver thermonuclear weapons to the United States. They concluded that the existing manual air defense system, including radar systems, automatic fire control devices, and communication links for ground-to-ground or ground-to-air communication, navigational systems, and both missiles and manned aircraft were only necessary components for a successful air defense system. Intelligent utilization of these components, however good, must be coupled with a system of detection and tracking of enemy targets at long ranges. More important, the intelligent commitment of these resources requires up-to-date knowledge of the total enemy threat and the success and status of weapons already committed. The researchers concluded that the manual air defense system could not adequately coordinate the use of our improving components against a growing enemy threat. The problem was inadequate continental data handling capability and inadequate facilities for communication filtering, storage, control, and display of the air defense situation. A system was required that would (1) maintain a complete current picture of the air and ground situations throughout North America; (2) control modern weapons rapidly and accurately; and (3) present filtered pictures of the air and weapons situations to the Air Force personnel manning the defense system.

SAGE, therefore, was developed to satisfy these requirements. SAGE utilized digital computers which were then considered large to process continent-wide air defense data. It was a real-time control system, a real-time communication system, and a real-time management information system.

## THE AN/FSQ-7 COMPUTER

### General Characteristics

The SAGE AN/FSQ-7 duplexed computer occupied the entire second floor of a SAGE

direction center. Seventy frames containing almost 60,000 vacuum tubes were required to handle all input-output data, perform air defense calculations, and store systems status data. The primary application of the AN/FSQ-7, air defense, required a reliability capability of a 24-hour operation. The duplex philosophy—that whenever individual equipment failure could cause a complete system failure that equipment would be duplicated—was adhered to; therefore the central computer, the drum system, and the magnetic tape units were duplexed.

### **System Organization**

The SAGE computer system consisted of the duplexed AN/FSQ-7 central computer, system status data stored on auxiliary magnetic drums, and the air defense computer programs consisting of approximately 100,000 computer instructions. Figure 1 shows a system organization chart. The central computer was buffered from all sector and console in-out equipment by magnetic drums. An exception was console keyboard inputs which used a 4096-bit core memory buffer. A real-time clock and four IBM 728 magnetic tape units used for input simulation and output summary recording compiled the basic computer configuration.

### **System Components**

#### *The AN/FSQ-7 Central Processing Unit*

The AN/FSQ-7 was a large-scale vacuum-tube general-purpose binary, parallel, single-address computer with a 32-bit word length and a memory cycle time of 6  $\mu$ sec. Its magnetic core memory contained 8192 words or 270,336 bits of storage arranged in two banks of 33 planes, each plane consisting of a  $64 \times 64$  core matrix. The 33rd plane was used for parity checking. The programs were written in a machine language instruction code with each instruction using one 32-bit word. The effective operating rate was about 75,000 instructions per second. Four index registers were available for address modification as was a 17-register test memory that could be used for system modifications at the operator console.

#### *Magnetic Drums*

Twelve magnetic drums, each with a capacity of 12,288 32-bit words, were available. They were used to store system control programs, system status data, and to buffer input and output data from external sources and the internal display system.

#### *System Display Consoles*

The major air defense functions required the use of over 100 display console stations, most of which included a keyboard input, a situation display, and a digital display

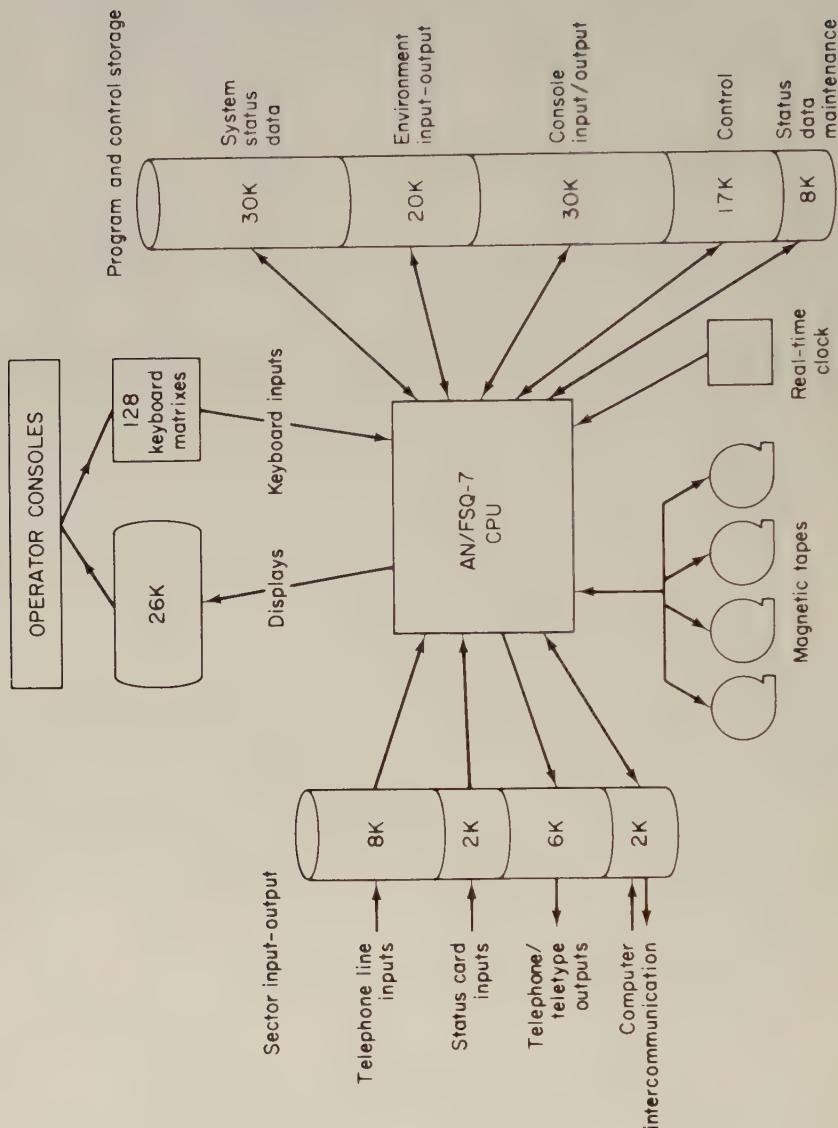


Fig. 1. AN/FSQ-7 logical organization.

scope. Each operator at a console had input and display facilities tailored to his responsibilities. Input information was inserted via the keyboard. Each console was provided with an input capacity to the computer of 25 to 100 bits of information at one time; the total keyboard input capacity for all consoles was over 4000 bits which were processed by the computer every 5 to 15 sec. A 19-in. Charactron cathode-ray tube displayed geographically oriented data covering the whole or part of the sector. On this air situation display scope the operator viewed different categories of tracks or radar data, geographical boundaries, predicted intercept points, or special displays generated by the computer. A light gun was available to the operator to notify the computer of data selection. Every 2.5 sec the computer would display up to 200 different types of displays requiring up to 20,000 characters, 18,000 points, and 5000 lines. Some of these displays were always present on the situation display scope. Others could be requested by the operator. Displays requiring the operator's attention or action would be forced to the operator's console display.

The operator's console usually was equipped with a 5-in. Typotron digital display tube which was used to present status or attention data such as weather conditions at an airbase, aircraft operational status, or "reason" data why the computer rejected an operator's actions. Sixty-three different characters were available on the Typotron scope. The computer display system displayed characters at the rate of 10,000 characters per 5-sec period to all the digital display scopes in the system.

## System Operation

### *Data Transfer*

With only an 8192-word core memory it is obvious that extensive data and program I/O transfer was required for the AN/FSQ-7 to operate a large-scale air defense program system. Under control of the central computer, data were transferred in variable length blocks between the magnetic drum and core memory. The CPU could transfer from 20 to 5000 word blocks of data per second. Maximum utilization of the computer was obtained by use of an in-out break feature. The in-out break feature allowed calculations in the CPU to continue during I/O operations; calculations were interrupted only for the one core memory cycle required to transfer a word between the core memory and the terminal device. More than 50% of real time was required for input-output searching, waiting, and transferring. The input-output buffering drums processed data independently of the CPU and thereby freed the CPU for more complex air defense processing. Separate READ-WRITE heads were provided for the buffering equipment and the CPU. The magnetic drums could therefore receive and transmit data while the CPU performed some other function of air defense. An example of this feature is in the method of receiving input data from voice bandwidth phone lines. A serial 1300 pulse/sec message was demodulated and stored in a shift register of appropriate length. When the complete message had been received, the message was shifted at a higher rate into a second shift register whose length was a multiple of 32 bits. This freed the first register to receive another message. When the first empty

register was located on the input buffer drum, parallel writing stored the word in  $10\mu\text{sec}$ . A relative timing indicator was also stored with the message, since the computer could not process the message for several seconds and time of receipt was critical. The CPU read this randomly stored message by periodically requesting block transfers of occupied drum slots only. Output messages were processed conversely. Another example of buffered processing is in the display system. The SAGE programs maintained ordered tables on the buffer display drum. This table was read and displayed by special-purpose equipment every 2.5 sec at the appropriate display console. The programs could change any part of the display at any time by rewriting only the appropriate words on the drum.

### *Real-Time Operation*

The real-time SAGE system programs operated in time periods of fixed length termed "frames." Each frame was several seconds in duration and was subdivided into subframes and semi-subframes. Component programs were operated at least once per frame and some operated by subframe. The buffer storage, system status data, and the computer programs were organized into blocks, each block consisting of from 25 to 4000 computer words. A sequence control program stored in core memory transferred appropriate program and data blocks into core memory, transferred control to the appropriate program, and then resumed control from the program to transfer appropriate data blocks (but not program blocks) back to the buffer drums. In order to fully utilize the in-out break feature, the operation of each program was closely coordinated with the sequence control program so that the proper programs and their respective operating environments were transferred at the proper times. This real-time synchronization resulted in program operation at regular intervals. During periods of light load, the sequence control program would buffer time until the real-time clock would indicate that the next frame or program cycle should begin. This feature simplified many of the timing problems associated with the control and input-output functions without system degradation.

### **System Reliability**

The primary application of the AN/FSQ-7, air defense, required a high reliability capability for 24-hour interrupted operation. For this reason, the system was duplexed. One AN/FSQ-7, the active computer, performed air defense, while the other was in "standby" status. The standby computer may have been in several modes, i.e., down for repairs, undergoing scheduled maintenance, or it may have been in use for program or system testing of special air defense programs. The reversal of active and standby AN/FSQ-7 computers was termed "switchover." Simplex devices which had been connected to the active computer were automatically transferred to the standby computer. Periodic transfer of critical system data was effected during each frame of real-time operation via an intercommunication drum between the duplexed computers. Because of the independent drum buffering system, this was possible even if the standby

computer was down for maintenance. Therefore the air defense data critical to system operation was made available to the standby computer in the event of an emergency failure of the active computer.

## BIBLIOGRAPHY

- Astrahan, M. M., B. Housman, J. F. Jacobs, R. P. Mayer, and W. H. Thomas, The logical design of the digital computer for the SAGE system, *IBM J. Res. Develop.* 1 (1957).
- Everett, R. R., C. A. Zrabet, and H. D. Benington, SAGE—A data processing system for air defense, in *Proceedings of the 1957 Eastern Joint Computer Conference*.
- Ogletree, W. A., H. W. Taylor, E. W. Veitch, and J. Wylen, AN/FST-2 radar-processing equipment for SAGE, in *Proceedings of the 1957 Eastern Joint Computer Conference*.
- Vance, P. R., L. G. Dooley, and C. E. Diss, Operation of the SAGE duplex computers, in *Proceedings of the 1957 Eastern Joint Computer Conference*.

Edward Wenzel

# ANALOG-DIGITAL CONVERSION

## ANALOG-DIGITAL CONVERSION

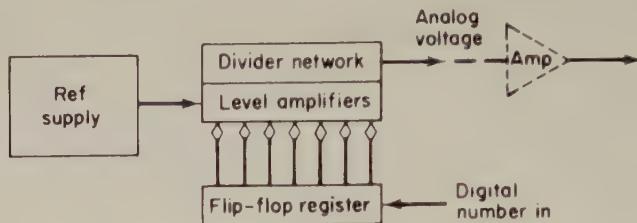
The mechanisms for converting between a discrete (digital) number (such as that held in a register of a digital computer) and a continuous analog signal (such as a voltage or current) is here described.

Conversion to analog is required in many applications such as process control and natural and physical sciences. For example, in medicine, an electrocardiogram is a graph of electrical potential voltage as a function of time as measured across various parts of the body. Similarly, a cathode-ray tube requires a voltage input whose value is proportional to the location of the point to be displayed.

## DIGITAL-TO-ANALOG CONVERSION

To convert a digital (discrete) number to a continuous analog voltage (or current), a resistive divider network is connected to a flip-flop register which holds the digital

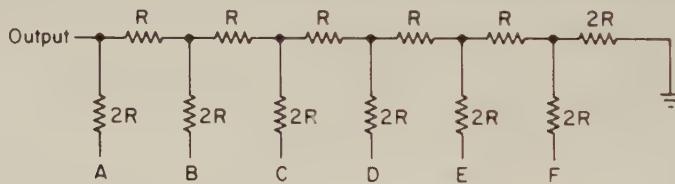
number (see Fig. 1). The divider network is weighted so that each bit of the register will contribute to the output voltage in proportion to its value.



**Fig. 1.** Digital-to-analog conversion.

The digital input signal determines the analog output voltage, since the divider network is usually a passive circuit. However, because digital voltage levels are not usually as precise as required in an analog system, level amplifiers are placed between the flip-flops and the inputs to the divider network. The amplifiers usually switch the divider network between ground and a precision reference voltage (e.g., -10 V). The output voltage range is thus between these two voltage levels. If the digital-to-analog converter is to drive a long cable or a heavy load, an operational amplifier or emitter follower is usually put on the output of the circuit to lower the output resistance.

The digital binary divider is a ladder network as shown in Fig. 2. The open circuit output voltage is one-half the voltage at input *A*, plus one-fourth the voltage at *B*,



**Fig. 2.** Binary ladder.

plus one-eighth the voltage at *E*, and so on. The input to each input mode of the ladder is thus either 0 or the reference voltage, depending on the input bit being 0 or 1. Thus the resulting open circuit output voltage is a properly weighted sum of the individual binary bits.

## ANALOG-TO-DIGITAL CONVERSION METHODS

The basis of analog-to-digital conversion is the comparator circuit and the digital-to-analog converter. The comparator compares an unknown input voltage with a second reference voltage and indicates which of the two is larger.

Fundamentally, analog-to-digital conversion is a guessing (or searching) process. The converter supplies a guess which is converted to an analog voltage, and this guess is compared against the unknown input. Depending on the accuracy of the guess, the search continues or terminates. Such a process is very similar to searching a symbol table for a particular value.

However, certain input values can only be approximated because of the discrete quantizing nature of the process. For example, an input of 1.75231 V can only be determined approximately as either 1.75 or 1.76 V in a 3-digit converter. Thus there is the representation problem that is fundamental to discrete (integer) mathematics.

### Simultaneous Method

Figure 3 shows how a simultaneous analog-to-digital converter can be built using several comparator circuits. Each comparator has a reference input signal. The

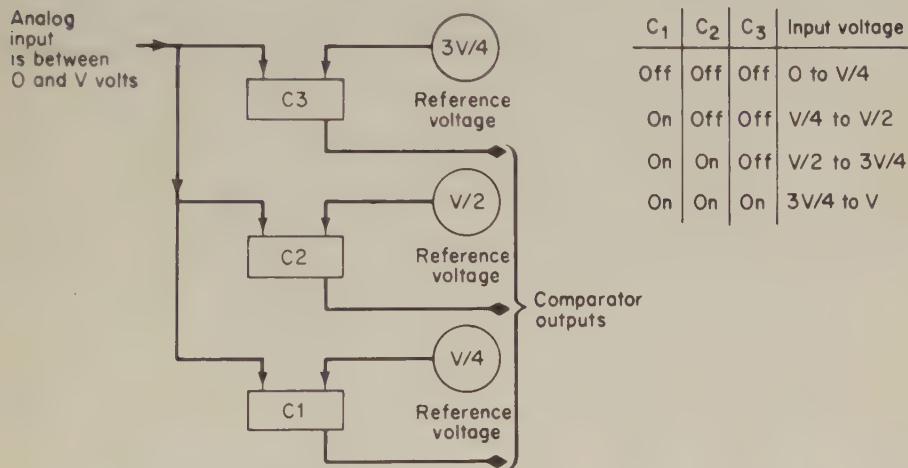


Fig. 3. Simultaneous analog-to-digital converter.

other input terminal of the comparators is driven by the unknown input analog signal, which is between 0 and  $V$  V. The comparator is called ON if the analog input is larger than the reference input. Then, if none of the comparators are on, the analog must be less than  $V/4$ . If C1 is on, and C2 and C3 are off, the input must be between  $V/4$  and  $V/2$ . Similarly, if C1 and C2 are on, and C3 is off the voltage is between  $V/2$  and  $3V/4$ ; and if all the comparators are on, the voltage is greater than  $3V/4$ .

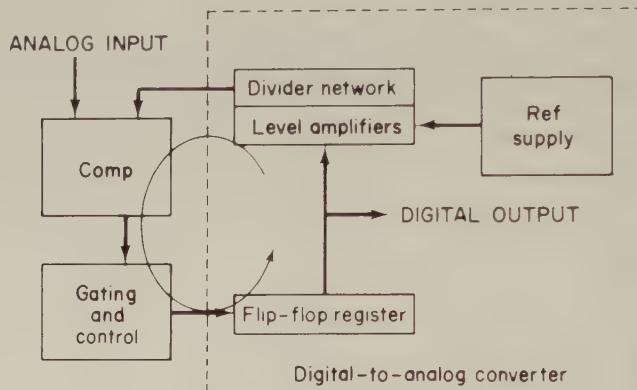
Here, the voltage range is divided into four parts, which can be encoded into two bits of information. Seven comparators would give 3 bits of information. Fifteen comparators would give 4 bits. In general,  $2^N - 1$  comparators will give  $N$  bits of information.

The simultaneous method is extremely fast for small resolution systems. For large

resolution systems (a large number of bits), this method requires so many comparators that it becomes unwieldy and prohibitively expensive.

### Feedback Methods

If the reference voltage were variable, only one comparator would be needed. Each of the possible reference voltages could be applied in turn to determine when the reference and the input were equal. But a digitally controlled variable reference is simply a digital-to-analog converter. Thus the generalized analog-to-digital converter shown in Fig. 4 is actually a closed-loop feedback system. The main components are



**Fig. 4.** Analog-to-digital converter incorporating a digital-to-analog converter.

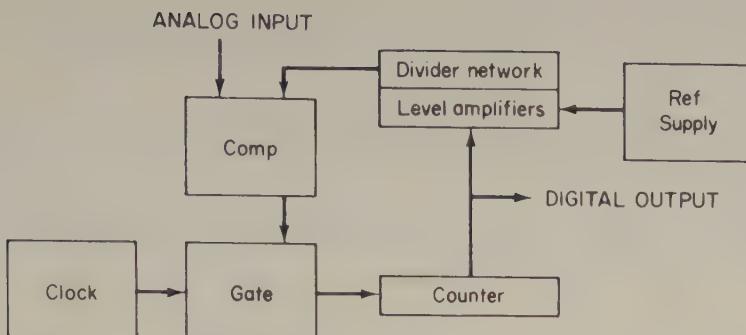
the same as the digital-to-analog converter plus the comparator and some control logic. With a digital number in the DAC (digital-to-analog converter) the comparator indicates whether the corresponding voltage is larger or smaller than the input. With this information, the digital number is modified and compared again. The following sections will describe various strategies for controlling the "search" for an encoded number.

### Counter (Linear Search) Method

Although numerous methods may be used for controlling the conversion, the simplest is to start at zero and count until the DAC output equals or exceeds the analog input.

Figure 5 shows a converter in which the DAC register is a counter and a pulse source has been added. The gate stops pulses from entering the counter when the comparator indicates that the conversion is complete.

The counter method is good for high resolution systems: as the number of bits is increased, very little additional circuitry is needed. Multiple inputs can easily be

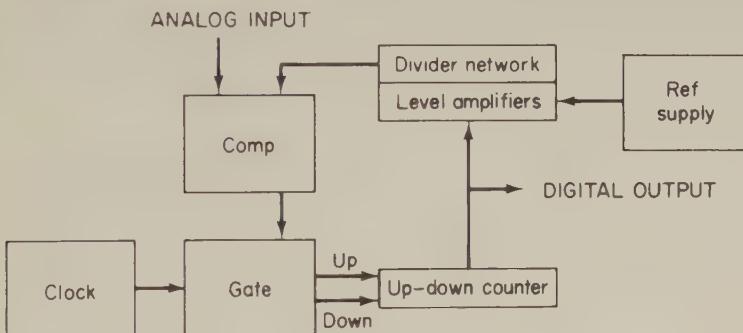


**Fig. 5.** Counter converter.

converted simultaneously by using multiple comparators and reading the counter at the appropriate times. However, conversion time increases rapidly with the number of bits, since an  $N$ -bit converter must allow time for  $2^N$  counts to accumulate. The average conversion time will, of course, be half this number.

#### Continuous Counter (Tracking) Method

A slight modification of the counter method is to replace the one directional counter with an up-down counter as in Fig. 6. In this case, once the proper digital



**Fig. 6.** Continuous converter.

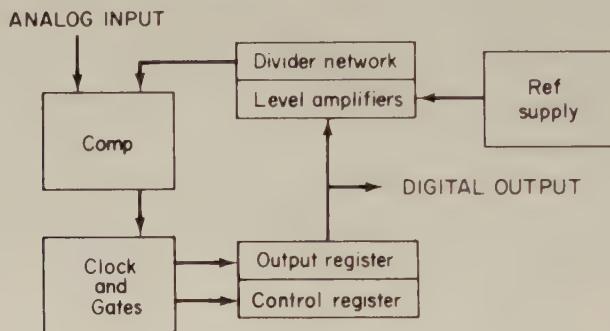
representation has been found, the converter can continuously follow the analog voltage, thus providing readout at an extremely rapid rate. This method, called continuous conversion, is particularly useful when a single channel of information is to be converted. The converter starts running, and the digital equivalents of the input voltage can be sampled at any time.

The continuous method is less effective for multiple inputs or for inputs that change faster than the converter can change. Each time the input makes a large

change, the converter may require as many as  $2^N$  steps to catch up. However, if a rapid rate of change is necessary, extra comparators may be added so that the up-down counter can count in units of 2, 3, 4, or more.

### Successive Approximation (Binary Search) Method

For higher speed conversion of many channels, the successive approximation converter (Fig. 7) is used. This method requires only one step per bit to convert any



**Fig. 7.** Successive approximation converter.

number. The successive approximation analog-to-digital converter operates by repeatedly dividing the voltage range in half as follows:

	111	111
110	110	110
	101	101
100	100	100
	011	011
010	010	010
	001	001
000	000	000

Thus the system first tries 100, or half scale. Next it tries either quarter scale (010) or three-fourths scale (110) depending on whether the first approximation was too large or too small. After three approximations, a 3-bit digital number is resolved.

Successive approximation is a little more elaborate than the previous methods since it requires a control register to gate pulses to the first bit, and so on. However, the additional cost is small and the converter handles all types of signals about equally fast, i.e., in  $\log_2$  (number of bits).

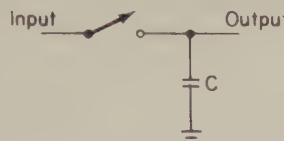
The successive approximation method is good for general use. It handles many continuous and discontinuous signals and large and small resolution conversions at a moderate speed and moderate cost.

## SAMPLE AND HOLD CIRCUIT

A sample and hold circuit holds an analog input value at a particular time for a longer time duration. In this way an analog input can be held constantly during the analog-digital conversion process.

The sample and hold circuit can be represented as shown in Fig. 8. When the switch is closed, the capacitor is charged to the value of the input signal; then it follows the input. When the switch is opened, the capacitor holds the same voltage that it had at the instant the switch was opened.

It is possible to build a sample and hold circuit just as shown in Fig. 8. Often the same circuit is used with a high gain amplifier to increase the driving current available into the capacitor or to isolate the capacitor from an external load on the output.



**Fig. 8.** Sample and hold.

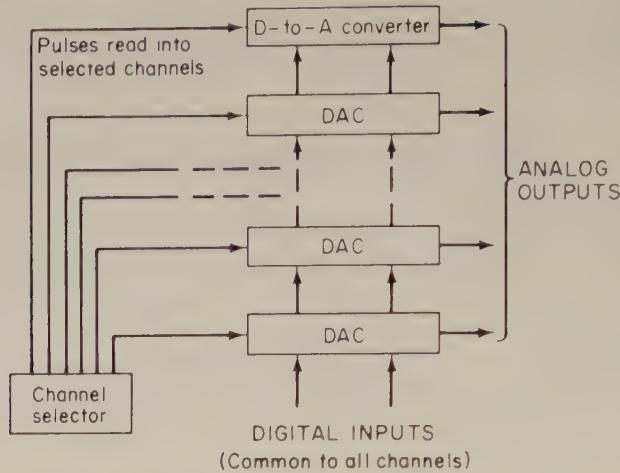
The *acquisition time* of a sample and hold is the time required for the capacitor to charge up to the value of the input signal after the switch is first shorted. The *aperture time* is the time required for the switch to change state and the uncertainty in the time that this change of state occurs. The *holding time* is the length of time the circuit can hold a charge without dropping more than a specified percentage of its initial value.

## MULTIPLEXING

Often it is desirable to multiplex a number of analog channels into a single digital channel or, conversely, a single digital channel into a number of analog channels. Multiplexing can take place in the digital part, the analog part, or in the conversion process.

### Digital-to-Analog Multiplexing

In digital-to-analog conversion, a common problem is to take digital information which is arriving sequentially from one device, such as a digital computer, and to distribute this information to a number of analog devices. Usually it is necessary to hold the information on the analog channel even when it is not being addressed from the digital device. There are two ways to multiplex. A separate digital-to-analog converter may be used for each channel as shown in Fig. 9.



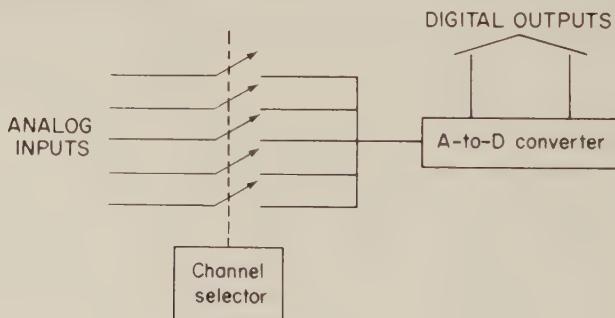
**Fig. 9.** Digital-to-analog systems.

In this case the storage device is the digital buffer associated with the converter. A single digital-to-analog converter may also be used, together with a set of multiplexing switches and a sample and hold circuit on each analog channel. The cost of the first method is slightly more than the cost of the second method, but it has the advantage that the information can be held on the analog channel for an indefinite period of time without deteriorating, whereas with the multiple sample and hold technique it is necessary to renew the signal on the sample and hold at periodic intervals.

Alternatively, a single digital-to-analog converter can be used with its output connected to multiple sample and hold circuits.

### Analog-to-Digital Multiplexing

In analog-to-digital conversion it is more common to multiplex the analog inputs. Here either relays or solid-state switches are used to connect the inputs to a common analog bus. This bus goes into a single analog-to-digital converter which is used for all channels (see Fig. 10). If simultaneous time samples from all channels are required,

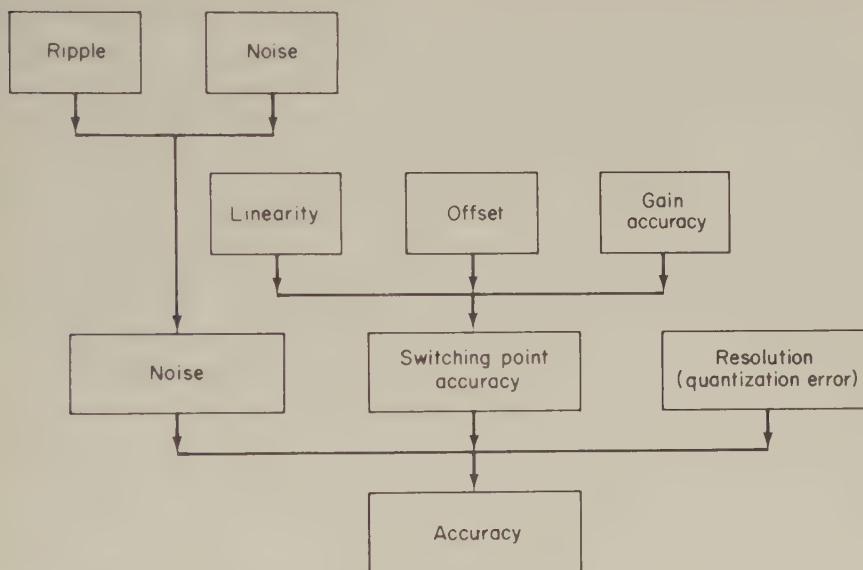


**Fig. 10.** Multiplexed analog-to-digital conversion system.

a sample and hold circuit can be used ahead of each multiplexer switch. In this way, all channels would be sampled simultaneously and then switched to the converter sequentially. The multiplex switches and sample and holds will introduce some error into the system. However, it is usually less expensive to go to higher quality sample and hold and multiplex circuits than to add extra converters.

## MEASURES OF CONVERTER PERFORMANCE ACCURACY

Since the end result of conversion is the representation of a given value in different terms, it is important to know how accurate the representation is. In systems where accuracy requirements are, say, in the order of 1%, an overall specification is usually



**Fig. 11.** Measures of cumulative error.

sufficient. In cases where the desired accuracy is 0.1% or greater, it is necessary to isolate the various sources of error; and since a converter is a hybrid device, both digital and analog sources must be taken into account.

In high accuracy systems particularly, accuracy figures given in the general specifications may not include isolated sources of error, e.g., noise. Thus, it is important to know the various types of errors, their causes, how they are measured and specified, and when they are important. Figure 11 shows a breakdown of various types of errors.

## DIGITAL ERROR SOURCES

When a continuous signal is quantized, there is an error which is equal in magnitude to the smallest quantum. For a linear converter, the smallest quantum is the least significant bit. In most converters the quantization error is centered so that it is equal to  $\pm \frac{1}{2}$  the least significant bit, written as  $\pm \frac{1}{2}$ LSB.

In a continuous converter or digital voltmeter, accuracy may not be as important as avoiding chatter. That is, if the input is right on the dividing line between two quantization states, the output should not oscillate. If hysteresis is introduced so that the quantization error is just under  $\pm \frac{1}{2}$ LSB, then oscillations will normally be avoided and the accuracy will not be greatly impaired.

The digital-to-analog converter reproduces exactly all the digital input information which it accepts. Hence digital error is not included in its accuracy specifications. However, if the input has more bits than the converter, there will be a quantization error in the readin process which should be taken into account. If desired, a  $\frac{1}{2}$ LSB offset can be built into the converter so that the readin will round off, rather than truncate, more precise digital information.

## ANALOG ERROR SOURCES

The dc accuracy of the converter (or switching point accuracy) depends on the offset, the gain calibration, and the linearity. Nonlinearities are due to the variation in gain (or common mode effect) in going from the smallest input to the largest input. Some of these will be long term because of the common mode effect of the comparator circuit in the analog-to-digital converter, for example. Some will be shorter term because of discontinuities in the divider network or insufficient settling time of the comparator. The offset and the gain can be adjusted in the calibration until their effects are essentially negligible.

The measurement of analog error in a digital-to-analog converter is easily made by putting in a digital number (the same word length as the converter) and observing the output. In an analog-to-digital converter the analog error is difficult to locate since the quantization error is always present. However, the point where the output oscillates approximately equally between two neighboring digital numbers is fairly well defined. This point, called the switching point, can be measured and compared with the theoretical value.

The ripple on the reference supply and other sources of noise are often measured separately since one or the other can sometimes be neglected in the final result. The two can be separated by measuring the ripple in the reference supply and subtracting it from the measured noise, or by running the input source in the converter from the same reference, thereby giving a direct measurement of all noise sources except the reference supply ripple. In a digital-to-analog converter the noise and ripple can be measured by observing the output with a scope. In an analog converter they can be

measured by observing the input range which causes the output to oscillate between two states.

## DIFFERENTIAL LINEARITY

Differential linearity is the variation in the size of the required voltage change that causes an analog-to-digital converter to go from one switching point to another. That is, it is the variation in the size of the states and is generally quoted as a percent of the size of the states. It is a part of the overall linearity discussed above, but deserves special mention because of its importance when an analog-to-digital converter is being used in histogram applications. For example, when plotting the number of inputs vs the digital state, if one of the states is twice as big as its neighbor, it will tend to accumulate twice as many counts. Naturally, a very misleading output results.

Differential linearity is one of the few accuracy characteristics which is affected by the conversion technique. The differential linearity tends to be best when the converter goes through all the states sequentially as in the counter-type converter. In an approximation converter, such as the successive approximation type, the large transients which result in going from, say, half scale to quarter scale require a long time to settle down, and any hysteresis in the comparator circuit causes relatively large variations in the state size. However, the differential linearity of an approximation converter can be improved by running it at very low speed. Differential linearity is also affected by variations in the divider networks (although they are relatively small). It can be avoided by using a ramp converter.

The shorter the converter word length, of course, the better the differential linearity will tend to be. However, this gain may well be compromised, since small resolution could result in the loss or the smoothing of sharp peaks in the histogram.

Techniques commonly used to overcome difficulties with differential linearity are changing the offset on the converter (or equivalently the bias on the input signal) and changing the word length of the converter. Switches can be mounted on the converter for this purpose, or the change can be made programmable so that the controlling device can make the change automatically.

## DISTRIBUTION OF ERROR

How much of the total error should be in the digital circuitry and how much in the analog portion? For converters in the range of up to 10 or 11 bits, the digital error generally accounts for about  $\frac{1}{3}$  to  $\frac{1}{2}$  of the total. Thus a typical 10-bit system would have a quantization error of  $\pm \frac{1}{2}$  LSB and an analog error of  $\pm 0.1\%$ .

If the accuracy requirement is low, the word length may be the major source of

error. The total error may then be treated simply as roundoff. If the accuracy requirements are stringent, it is desirable to minimize all sources of error, analog and digital. The digital error is quite simple to minimize by extending the number of bits within practical limits. A converter with an overall error of 0.1% and a word length of 20 bits would be unjustified, as for all intents and purposes the least significant bit could have been generated by a random number generator.

Requiring monotonicity is one way to assure that all the bits are meaningful. This means that all states must exist and they must be in the correct order. In terms of converter operation, as the number going into the digital-to-analog converter is increased, the output voltage must also increase; it should never dip back down at any point. Similarly, if the input voltage to an analog-to-digital converter is increased, the digital output should stay at the same value or increase and should not skip over any states.

The converter is most likely to lose monotonicity when switching between digital states such as 0111 and 1000. If the weighting of the bits is not quite correct, in a digital-to-analog converter the output might jump directly from 0110 to 1000.

## MEASURES OF SPEED

### Digital-to-Analog Conversion

The maximum conversion rate is theoretically limited only by the minimum time between readins to the converter flip-flops, and can easily be as high as 10 MHz. However, such a figure may be misleading. The desired ratio of settling time to nonsettling time usually determines the maximum usable conversion rate.

The settling time of a converter is measured from the time the digital readin is performed to the time when the analog output has settled to within the specified limits of accuracy. How the output approaches its final value depends on the output circuit, as discussed below.

The divider output will have high-frequency transients before it begins to settle. If the output is going to a low-frequency device, the transients can be ignored. In some applications it is more desirable to smooth the transition between states than to minimize the total time, in which case the oscillations can be damped with the capacitor or a low-pass filter.

If the output is from an amplifier circuit, the settling time will be determined by the maximum rate of change of the amplifier. Thus, the first readin may take longer to settle than subsequent readins, which usually do not change the converter by such a percentage of the full scale.

### Analog-to-Digital Conversion

Conversion time is measured from when a request is given to when a digital output

is available. In converters like the successive approximation type, where all conversions are completely independent, time must be allowed for completion of entire steps in the conversion process. In the continuous converter the conversion time is usually just the time required to synchronize the request and get the number.

The conversion rate is usually the inverse of the conversion time. In some systems an amplifier or comparator recovery time is required between conversions; thus the rate is lower. The conversion rate will also be slower if logical operations must be carried out between conversions. In some cases, such as the counter converter performing a number of simultaneous conversions or the synchronous sequential

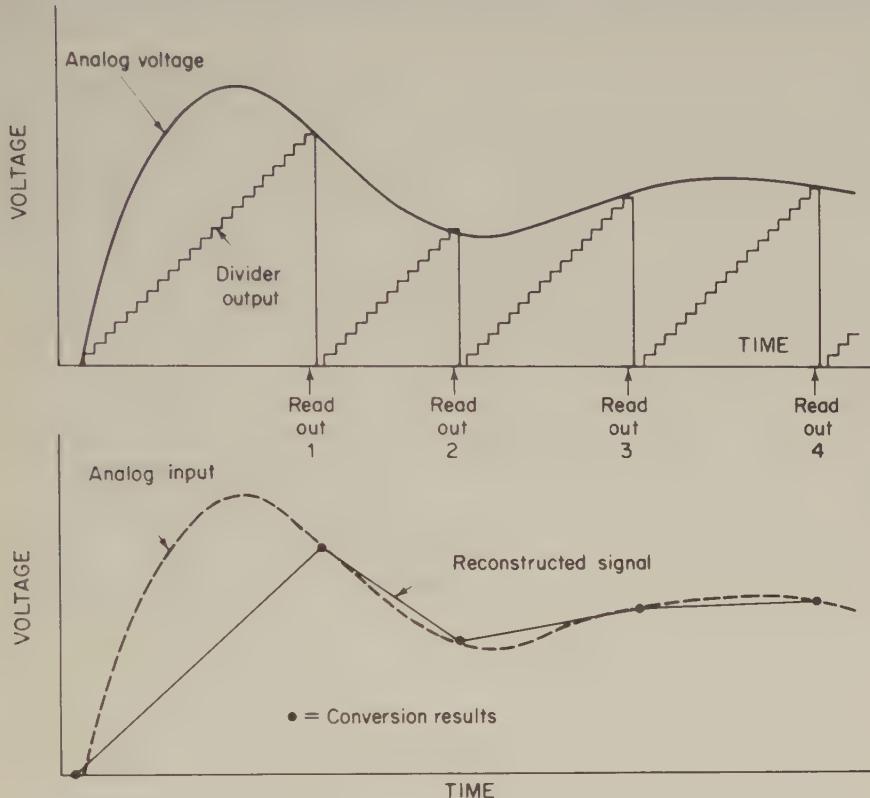


Fig. 12. Counter converter.

converter, the conversion rate is actually faster than the inverse of the conversion time.

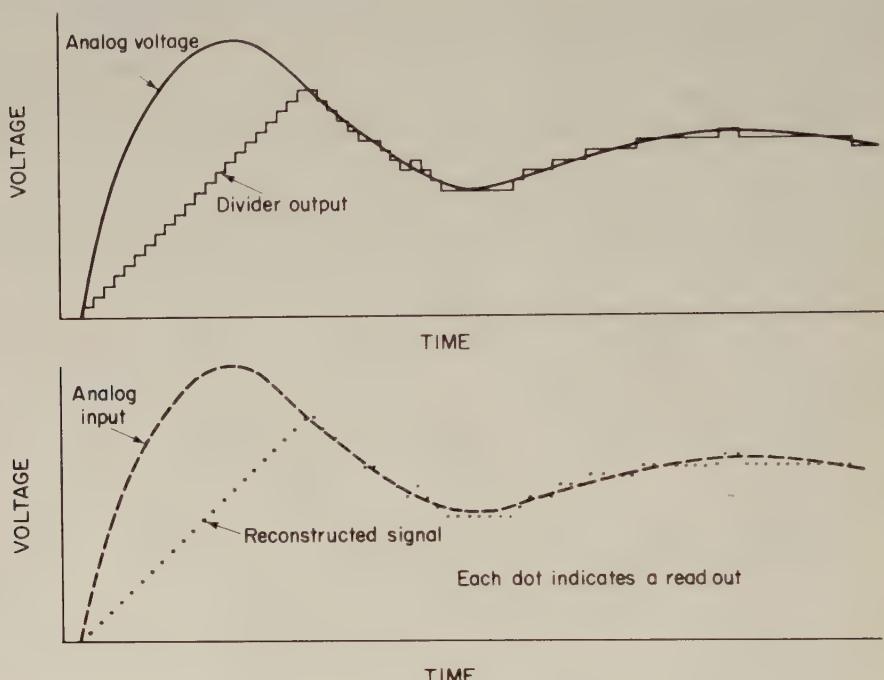
If the input signal is changing with respect to time, it is very important to know when the signal had the value given by the output. The uncertainty in this time measure is called the aperture time (sometimes also called window or sample time). The size of the aperture and the time when the aperture occurs vary, depending on the conversion method.

Figures 12 through 15 illustrate how the aperture varies with different conversion techniques. In each case the upper portion of the figure shows how the converter

arrives at an output. The lower portion of each figure shows how the input might be reconstructed from the digital data.

In the counter converter (Fig. 12) the aperture occurs at the end of the conversion. This is not constant with respect to the beginning of the conversion, but it may be calculated from the digital output.

For the continuous converter (Fig. 13) the aperture is the time for the last step. Here the assumption is made that the input signal does not change more than



**Fig. 13.** Continuous converter.

$\pm 1$ LSB between conversion steps. To meet this requirement, the maximum rate of change of the input voltage must not exceed the maximum rate of change of the converter. This is  $V_{\text{ref}}/2^N \Delta T$ , where  $V_{\text{ref}}$  is the full scale voltage,  $N$  is the number of bits, and  $\Delta T$  is the time per step. The maximum rate of change of the sine wave is  $2\pi V_p f$  or  $\pi V_{\text{pp}} f$ . Thus, if the converter is to follow the input, the maximum frequency components in the input must satisfy

$$\pi V_{\text{pp}} f = V_{\text{ref}}/2^N \Delta T$$

and if the peak-to-peak voltage is assumed equal to the converter reference, then the maximum frequency is

$$f = \frac{1}{2^N \Delta T \pi}$$

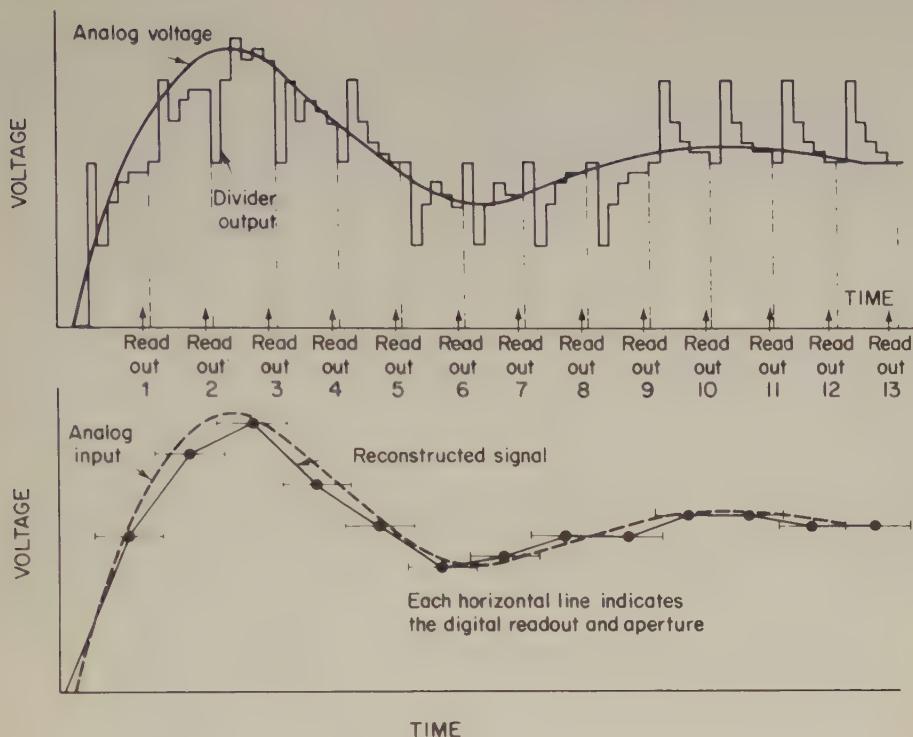


Fig. 14. Successive approximation converter.

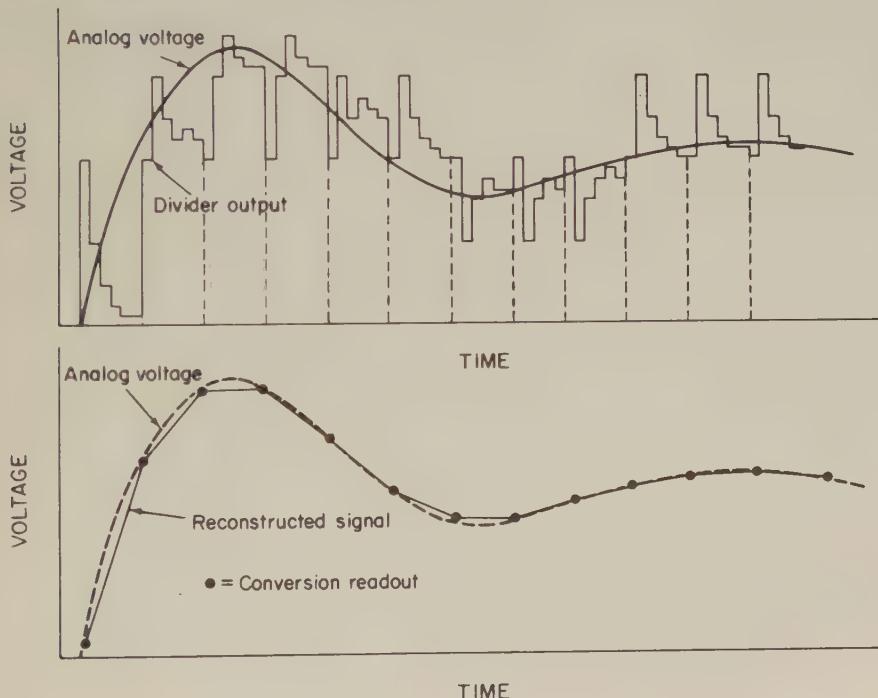


Fig. 15. Successive approximation converter with sample and hold.

For a successive approximation converter (Fig. 14) the digital output corresponds to some value the analog input had during the conversion. Thus the aperture is equal to the total conversion time. Aperture time of the successive approximation converter can be reduced by using a sample and hold circuit. The sample and hold is illustrated in Fig. 15.

*Barbara Stephenson*

## ANALOG SIGNALS AND ANALOG DATA PROCESSING

### THE NATURE OF ANALOG INFORMATION

The real world with which digital computers interact is largely analog in nature. The monitoring and control of physical processes, the measurement at remote locations of physical variables, the display of information using cathode-ray-tube oscilloscopes, the utilization of most common carrier communication networks—these are only some of the many areas in which digital computer systems must interface with data and signals in analog form. In order to study the interaction of digital computers and external systems, it is, therefore, essential to appreciate the nature of analog information and its differences from digital data. This in turn permits an understanding of the manner in which data can be processed and computations can be carried out using analog computational units.

Modern science and engineering are based upon a quantitative description of the physical universe. A variety of so-called physical variables is measured, and inferences are drawn from the results of these measurements. In this connection, it is necessary to first distinguish between independent and dependent variables. In most system analyses, time and space constitute the independent variables. That is, physical measurements are distinguished from each other and ordered according to the location in the time-space continuum at which the measurements were made. The measured quantities are the dependent variables, and they may be expressed as functions of time and/or space. Some familiar dependent variables include voltage, displacement, velocity, pressure, temperature, stress, and force. The measurement of these variables requires the selection of appropriate instruments and a decision as to the manner in which the measurements are to be recorded and utilized. There are two major ways

in which a physical variable is treated by measuring and data processing systems: analog and digital. These are defined as follows.

1. A dependent variable is said to be an *analog variable* if it can assume any value between two limits.
2. A variable is said to be a *digital variable* if its magnitude is limited or restricted to certain specified values or levels.

Although most readers will have encountered the above definitions in one form or another, their implication and real meaning is often misunderstood. As illustrated in Fig. 1, analog as well as digital variables can take a variety of forms. It should be recognized, first of all, that the distinction between analog and digital signals hinges on the dependent rather than on the independent variable. While both independent and dependent variables are continuous in form in Fig. 1(a), the independent variable is discrete in Fig. 1(b). Nonetheless, the transient shown in Fig. 1(b) is an analog variable because the amplitude of the pulses is not restricted to discrete levels. Furthermore, the definition of an analog variable does not preclude the existence of constant levels or of steps, as shown in Fig. 1(c) and (d), as long as the variable is not prohibited or constrained from assuming magnitudes in between the steps. For a signal to be considered analog, it is not necessary that it actually assume all values within its dynamic range; it is only necessary that there be a possibility of its assuming all these values.

Similarly, digital variables can be represented not only by constant levels, as in Fig. 1(e), but also by irregular transients such as those shown in Fig. 1(f) and (g). What makes them digital variables is that any departure from the fixed levels (not necessarily two levels) is ignored by the measuring device. To a large extent, therefore, the difference between analog and digital variables is "in the eyes of the beholder." It is usually not possible to examine an electrical transient with an oscilloscope or other measuring device to determine whether the electrical signal represents analog or digital information; that depends entirely upon how the transient is used or interpreted in a specific application.

Some data processed by computers fall naturally into the digital category. For example, bookkeeping, accounting, and other financial information is always quantized to the nearest cent or dollar. Similarly, data which is introduced into the computer using alphanumeric keyboards is necessarily digital in form. On the other hand, the variables arising in such physical systems as electric circuits and fields, mechanical devices, chemical processes, heat transfer and fluid flow systems, and biological systems all exist in nature in analog form. That is, there is no constraint upon these variables which forces them to assume discrete levels. The processing of such variables and signals can be carried out in three basic modes.

1. *All digital:* All variables are immediately upon sensing translated into digital form using an analog-digital converter. This involves the rounding-off of the variable magnitudes to the nearest available discrete level.

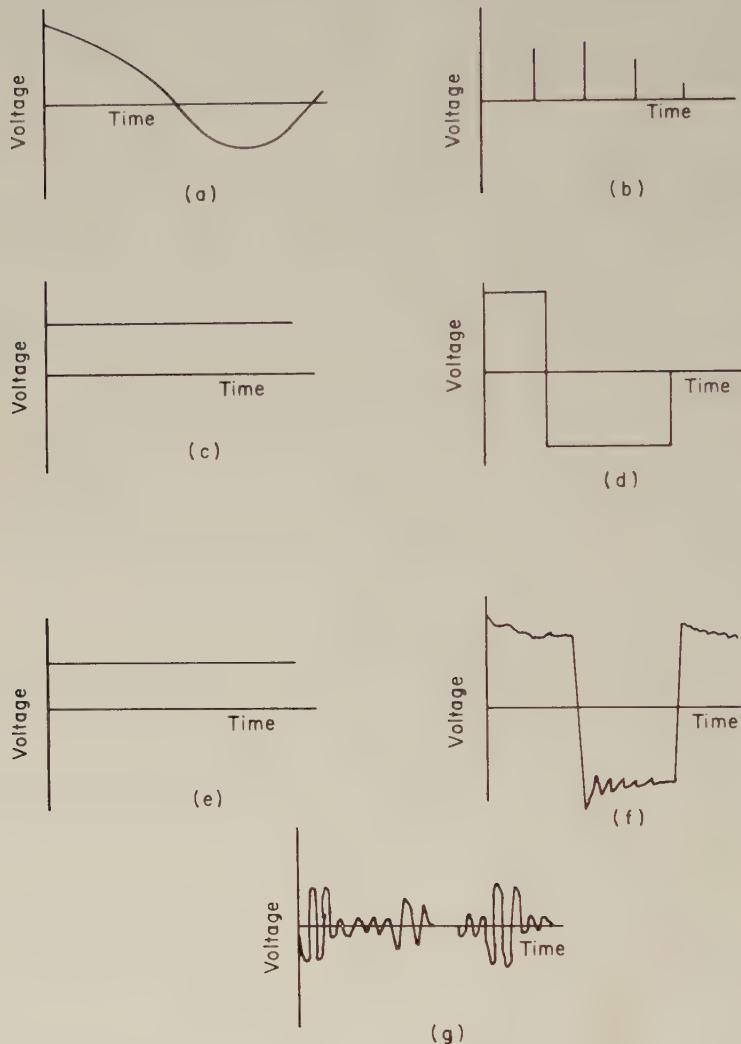


Fig. 1. (a)-(d) Analog signals. (e)-(g) Digital signals.

2. *All analog*: Analog computing units are supplied for the entire data processing operation, so that no analog-digital conversion is required.

3. *Part analog, part digital*: Some variables are translated into digital form for processing by a digital computer, while other variables are processed using analog computational units. This is known as *hybrid* computation.

The decision as to whether to process data in analog or digital form has far-reaching consequences on the organization of the computer system and its cost, upon the accuracy of the computations, and upon their speed. In order to place the discussion of analog signal processing in its proper perspective, these considerations are briefly summarized.

The carrying out of computations requires electronic circuitry to perform specified mathematical operations. A basic distinction between analog and digital data processing is that digital computations are usually performed sequentially or serially, while analog computations are performed simultaneously or in parallel. Digital data processing generally requires reference to data and programmed instructions stored in a memory unit. For technical reasons, there exists a bottleneck at the entrance to this memory, so that only one item of information can be read into or read out of the memory at any particular instant of time. Therefore, only one arithmetic operation can be performed at a time. This implies that data processing consists of a sequence of arithmetic operations. For example, if ten numbers are to be added, ten successive additions are performed. No additional equipment is needed if 100 additions are required instead.

By contrast, an analog processor does not require an expensive component such as a memory which must be time-shared among the various mathematical operations. Rather, a separate electronic unit or "black box" is supplied for each mathematical operation. If a computation requires ten additions, ten analog operational units must be provided and interconnected; all of these units operate simultaneously. If the number of required additions is increased to 100, the amount of electronic equipment necessary is multiplied by a factor of 10. The hardware structure and the cost of an analog data processing system is, therefore, determined by the types and numbers of specific mathematical operations which are to be performed. The structure of a digital processing system, on the other hand, includes standardized memory, control, and arithmetic units, and is more or less the same regardless of the types of computations that are to be performed.

The accuracy of a computation performed by a digital processor is determined by the word length or number of bits employed to represent data. For example, if two numbers are to be multiplied in a digital processing system in which numbers are represented by 12 binary digits, the result of the multiplication must be rounded up or down to the nearest least significant bit. There is, therefore, a chance of a roundoff error corresponding to one half of the least significant bit. At the expense of computing time, this roundoff error can be reduced by using double-precision arithmetic. In an analog processor, data are not discretized, and roundoff errors are therefore not incurred. Instead, the accuracy is limited and error is introduced by the nonideal functioning of the operational units used to carry out the computations. If two

variables are to be added, they are each applied as continuous voltages to an adder unit. The output voltage of the adder then corresponds to the sum of the two variables. The accuracy of this addition operation is limited by the quality (tolerance) of the electronic components making up the adder and by the precision with which the output voltage can be measured and recorded. In the performance of linear mathematical operations (such as addition, subtraction, and multiplication by a constant), errors of 0.01% are almost impossible to avoid; in the case of nonlinear operations, the best available electronic units are subject to errors of 0.1%. In this connection, it is important to recognize that the term "accuracy" has different meanings in the digital and the analog worlds. In digital computations, percentage error  $\varepsilon_D$  is usually defined in terms of the computed value  $C$  and the true value  $T$  as

$$\varepsilon_D = \frac{C - T}{T} \times 100\%$$

That is, error is specified as a percentage of the correct value. Such a definition of error is difficult to apply to analog data, since analog variables can pass smoothly through the zero level, as in the case for the function shown in Fig. 1(a). Note that the above equation "blows up" as  $T$  approaches zero. In other words, in the treatment of an analog variable such as that shown in Fig. 1(a), the above equation would predict infinite error or zero accuracy regardless of how excellent the analog processing unit may be. For this reason, analog equipment manufacturers and analog users have agreed to define the percentage error  $\varepsilon_A$  in analog systems as

$$\varepsilon_A = \frac{C - T}{D} \times 100\%$$

where  $D$  is the dynamic range, defined as the maximum excursion of the dependent variable. The two definitions for error are identical only at the extremity of the dynamic range, that is, for the maximum values of the dependent variable. At all other points, particularly near zero,  $\varepsilon_A$  is considerably smaller than  $\varepsilon_D$ .

The speed with which a sequential digital computation can be performed is determined by the complexity of the computation. The larger the number of arithmetic operations that must be performed, the longer the time required. One hundred additions require nearly ten times as much computing time as ten additions. By contrast, in analog data processing, the time required for computations is independent of problem complexity. One hundred additions require precisely the same time as ten additions; approximately ten times as much hardware is required, however. The speed with which a mathematical operation can be performed using an analog unit is determined by the characteristics of its electronic components as well as by the characteristics of the measuring or output devices. In most instances, analog data processors have a considerable speed advantage over competing digital processing equipment.

The cost and capability of analog processors is determined almost entirely by the quantity and quality of the operational units which are supplied to perform the various required mathematical operations. For this reason, most of the rest of this article is devoted to a discussion of the performance of mathematical operations

using analog devices. In most modern systems utilizing analog processing, only the operational units actually required for the specific task at hand are provided. These are interconnected in a semipermanent fashion as required by the specific application. Less frequently, so-called general-purpose analog processors are fashioned by assembling a variety of operational units and permitting the user to interconnect them as he requires. Such a system is termed an *electronic analog computer*. Since virtually all considerations involved in discussing the operation of analog computers also apply to the operation and use of analog operational units for special-purpose tasks, whereas the reverse is not true, the discussion in the following sections emphasizes general-purpose analog computations. The concepts developed are directly relevant to more limited analog computational tasks including process control, filtering, and analog communication. In the next section the organization of analog computers and analog signal processors is briefly discussed. The following sections will discuss techniques for performing linear mathematical operations using analog devices; nonlinear mathematical operations; the control of analog units; and the interconnection of operational units for the performance of complex mathematical tasks. For more detailed discussions of these topics, the reader is referred to the texts dealing with the design and operation of analog computers and analog processors [1-5].

## ORGANIZATION OF ANALOG COMPUTERS

The term "analog computation" has been used to describe a broad spectrum of computational methods and techniques. Conductive field analogs, such as electrolytic tanks, were widely used even before the turn of the century. In the 1920s and 1930s, the so-called Bush mechanical differential analyzer became an important engineering design tool. However, modern-day analog computation is a direct product of electronic advances during and after World War II. In particular the development of dc amplifiers, first in vacuum-tube form, later using solid-state devices, and finally in integrated circuit and LSI realizations, led to the application of analog techniques in a wide variety of industrial and scientific applications. At the same time the use of analog techniques for general-purpose computations narrowed to the treatment of systems of differential equations, since the analog computer was unable to compete with digital techniques in the treatment of problems requiring only the arithmetic operations of addition, subtraction, multiplication, and division. On the other hand, the operation of integration required for the solution of differential equations is relatively difficult to perform accurately using digital equipment, while it is a "natural" for analog devices.

Since differential equations underlie most engineering disciplines, the analog computer became an important engineering design tool. Because of their relatively specialized nature, analog computers are also known as *electronic differential analyzers*. The utility of analog computers was greatly increased in the 1960s by the interconnection of analog computers with on-line digital computers to form

hybrid computer systems. By the 1970s, competing digital techniques had advanced sufficiently to pose a significant challenge to analog computers even in the treatment of differential equations. This resulted in a further diminishing of the role of general-purpose analog computers, and it may eventually lead to their extinction. There is little challenge, however, to the role of analog operational units in special-purpose data processing and control applications.

In order to function efficiently and automatically, general-purpose analog computers must make provision for the performance of the following functions: input of numerical data and instructions; storage of data; performance of mathematical operations; control of data flow; output.

These five functions correspond closely to those which are performed in all digital computer systems. It is possible therefore to describe the organization of electronic differential analyzers in a manner paralleling step by step description of the operation of digital computer systems. At the same time it must be recognized that the continuous nature of the data in analog computers calls for electronic elements entirely different in design and operation from those employed in digital computers. Furthermore, the parallel nature of analog computation leads to the assignment of a far greater fraction of the electronic circuits within the analog computer to mathematical (arithmetic) operation than is the case in digital computers.

In digital computers it is the size and nature of the central memory which determine to a large extent the general utility and also the cost of the computer installation. In analog computers, on the other hand, it is the array of mathematical operational units which determines the characteristics of the overall computer system. For example, the solution of a single second-order differential equation requires at least two integrators and one adder. To solve a set of 50 simultaneous second-order differential equations then requires at least 100 integrators and 50 adding units, and it is the precision with which each of these units functions that determines the accuracy of the final solution.

The mathematical operational units which are available in most conventional analog computer facilities fall into two categories: linear and nonlinear units. The linear operational units include

1. Constant multipliers—units generating a transient voltage which is the product of the input voltage and a positive or negative constant.
2. Adders—units which generate transient voltages equal to the sum of two or more transient input voltages.
3. Integrators—units which generate the integral with respect to time of the sum of one or more transient input voltages, and which make provision for the application of specified initial conditions.

Inherent in the operation of integration is provision for short-term memory. It will be recalled that the integral of a function presented as a curve  $f(x)$  vs  $x$  is obtained by performing a cumulative summation of the area under the curve, starting with an initial value. In analog computers this cumulative sum is represented by the accumulated charge in a capacitor, which acts to keep a “running tally” of the integral.

Nonlinear operational units frequently found in computer installations include

1. Multipliers—devices for generating a transient voltage which is the product of two transient input voltages.
2. Resolvers—devices for generating transient voltages which are equal to the sine or the cosine of transient input voltages.
3. Arbitrary function generators—devices generating transient output voltages having adjustable and specified but nonanalytic relationships to the input voltages.

Large analog computer installations may include several hundred integrators and adders, 50–100 multipliers, 20 or more resolvers, and 50 or more function generators of various types. Having assembled the desired combination of mathematical operational units, it remains for the computer designer to provide means for their programming and control, as well as for the input and output of numerical data. As in the case of digital computers, the overall design of analog computer facilities has become sufficiently standardized over the years so that the general organization of virtually all available electronic differential analog analyzers can be represented by a single block diagram, such as the one shown in Fig. 2.

All commercial installations are designed around a patch bay located in a control console. Wires leading to the input and output terminals of all operational units are brought out to an array of several hundred or even thousands of patch tips. Removable problem boards, made of an insulating material, are machined to fit precisely over these patch tips in such a manner that a clearly identified hole in the problem board lies directly over each patch tip. The bulk of the programming and connecting of the computer can then be accomplished by means of electrical connections, termed patch cords, interconnecting the various holes in the patch board. Usually a considerable number of patch boards are available with each computer. Problems can be programmed on these boards and stored for subsequent experimental work while the computer is being employed to solve an entirely different problem. A considerable effort has been expended in optimizing the design of patch boards to facilitate their use. Even so, the programming of reasonably complex problems results in a veritable maze of plug-in wires, a factor which frequently leads to errors in programming and makes problem checking very difficult. To alleviate this situation, most manufacturers have introduced color-coded plug-in connectors and multicolored problem boards.

The patch board with its plug-in connections serves in the analog computer the function occupied by the following items in a digital computer

1. The deck of punched cards which contain the instruction program for a given problem.
2. The memory cells in which this program is stored prior to problem solution.
3. The control circuitry which channels the instructions and commands in the prescribed manner.

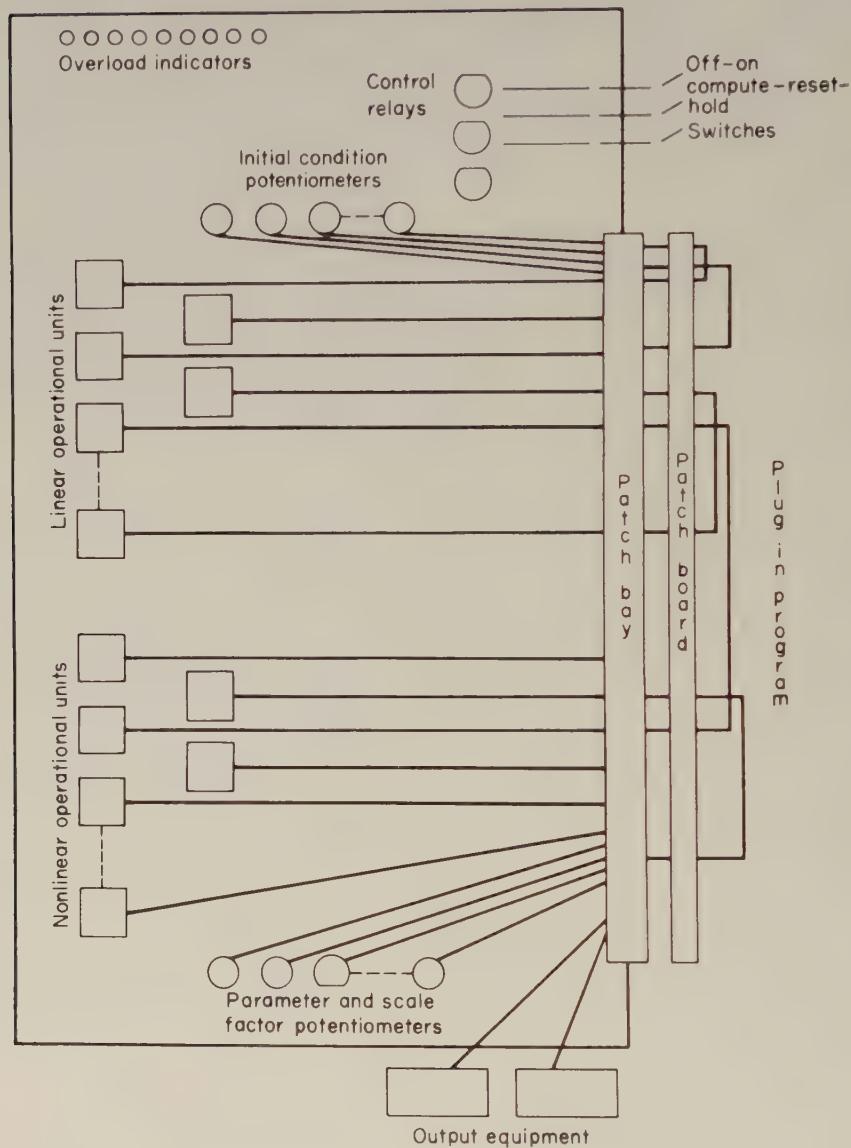


Fig. 2. General organization of an electronic analog computer.

The patch board therefore serves the combined function of long-time memory of instructions (but not of numerical data), and of control.

Numerical input information is memorized in analog computers with the aid of resistance potentiometers. This information includes particularly the initial conditions which must be furnished to permit the solution of ordinary differential equations, and the numerical values of all parameters. A separate potentiometer is provided for each numerical input quantity. Thus, unlike digital computers, analog computer memories for commands and for numerical data take entirely different forms.

In addition to the patch bay, which performs the program control function, the control console includes a variety of relays to facilitate such control tasks as start, stop, reset, hold, and a variety of others. More recently, several computer manufacturers have begun to provide a number of digital modules, such as counters and gates, to permit automatic modification of the computer program in successive repetitive solution cycles. These modules enable the computer to automatically readjust initial conditions, parameter values, and sometimes the network topology, on the basis of information generated in the course of a problem solution.

Input information may be applied to analog computers manually, but most modern computer facilities make available devices for the automatic setting of potentiometers and for the input of graphical data. Output devices found in virtually all facilities include digital voltmeters for the reading of steady-state or dc values and oscillographs or oscilloscopes for the plotting of transient output data. In addition, computer consoles are usually equipped with an array of neon lights, one for each operational unit, which glow whenever the corresponding unit exceeds its dynamic range in either the positive or negative direction. The overloading of electronic amplifiers in analog computers results in errors very similar to those encountered in digital computers operating in a fixed-point mode when a register is caused to overflow. To avoid such overloading in analog computers it is necessary to select suitable *scale factors* to relate the independent and dependent problem variables to the corresponding quantities in the analog computer. These scale factors are introduced into the computer as potentiometer settings. If scale factors are chosen properly, the full range, say  $-100$  to  $+100$  V, available in the analog computer is utilized without exceeding this range in any of the operational units.

The treatment of a system of linear or nonlinear ordinary differential equations on an analog computer then takes the following steps.

1. The specified equations are subjected to mathematical transformations to facilitate their programming with available equipment.
2. Scale factors are selected relating all dependent variables and their derivatives to voltages within the computer system, and relating the independent problem variable to the computer time variable.
3. A flow chart is prepared showing the manner in which the computer mathematical units must be interconnected in order to generate the problem solution. This step corresponds closely to that of preparing the flow chart for digital computer solutions.

4. A patch board is programmed by interconnecting holes in the board by means of electrical connectors. This step corresponds more or less to the coding of the digital computer.

5. A number of dials are set manually to correspond to specified numerical data including particularly the initial conditions and parameter values. This step corresponds to the "readin" of numerical data as the first step in a digital computer run.

6. The patch board is clamped to the patch bay so that the patch-board connections act to interconnect the mathematical units, the initial condition and parameter potentiometers, and the output equipment specified by the flow chart program.

7. The "start" button is pressed, causing all mathematical units to begin operation simultaneously. The solution is generated immediately and is displayed either on the face of an oscilloscope or as a graphical plot on a strip-chart or two-coordinate plotter.

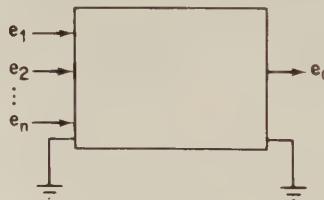
8. When sufficient data have been generated, the "stop" button is pressed and the computer comes to a halt.

9. The computer output is analyzed and translated back into the language of the original problem.

As in digital computers, the time required for the running of a problem is usually negligible compared to the time expended in programming and problem check-out. In the case of the analog computer, however, this difference is considerably more marked. Thus solutions of a specific set of differential equations rarely require more than several minutes regardless of the complexity of the problem.

## LINEAR OPERATIONAL UNITS

The purpose of the linear operational units, and indeed all operational units, in electronic differential analyzers is to accept one or more transient dc voltages as



**Fig. 3.** Operational unit.

inputs and to provide transient output voltages bearing some specified functional relationship to the input voltages. This is illustrated in Fig. 3. Many mathematical operations can be performed by combinations of passive circuit elements. For example, integration can be performed by means of a single resistor and a single capacitor. Such devices are capable of providing accurate results only in the absence

of load. Since the operation of analog computers requires the interconnecting and cascading of individual operational units, no-load operation is impossible. It therefore becomes necessary to introduce active elements in the form of operational amplifiers into the circuit. Since these amplifiers are at the heart of all modern electronic differential analyzer operational units, the characteristics of these electric circuits are discussed first in this section. This is followed by a survey of the more important linear operational units.

### The Operational Amplifier

In order to permit the performance of the desired mathematical operations with sufficient accuracy and precision, the electronic amplifiers used in the computing process must have the following general properties

1. High gain; generally in excess of 100,000.
2. Linearity over a wide region of operation; generally from  $-100$  to  $+100$  V at the output.
3. Flat frequency response; generally from direct current to several hundred cycles per second but sometimes up to several kilocycles.
4. Zero output voltage for zero input voltage.
5. Very high input impedance.
6. Reversal in polarity between the input and output of the amplifier.
7. Very low noise level.

The severity of the above requirements, particularly regarding linearity and frequency response, delayed the realization of practical general-purpose electronic analog computers until after World War II. In order to permit operation at dc it is necessary to employ direct coupling between successive amplifier stages so that no coupling capacitor can be used. This in turn makes it necessary to provide a positive as well as a negative high voltage power supply.

The main source of error attending the use of dc operational amplifiers is their tendency to *drift*. Prior to use as a computing element the amplifier must be carefully "balanced" by connecting the input terminal to ground and adjusting a bias within the amplifier to give zero output voltage. If this balance setting is left unchanged and the input terminal maintained at zero, the output terminal will be found to deviate or drift gradually away from zero. The causes for this phenomenon include variations in the power supply voltages as well as transient variations in the properties of the circuit components. Much effort has gone into the design of operational amplifiers with a minimum of drift errors. The most successful and widely used devices of this type employ so-called *chopper stabilization*. This method is based on the recognition that an ac amplifier is not subject to drift in the same manner as a dc amplifier. Accordingly a separate ac amplifier and a solid-state chopper are provided.

The conventional symbol used to denote an operational amplifier is shown in Fig. 4. As in most analog circuit diagrams the ground bus is omitted, it being understood that all voltages in the circuit diagram are referred to the computer ground

terminal. The output voltage is equal to the input voltage times a large negative constant. The manner in which operational amplifiers are used in analog computer circuits makes it unnecessary to specify the magnitude of this constant provided only that it be very large. To fashion analog computing circuits the operational amplifier

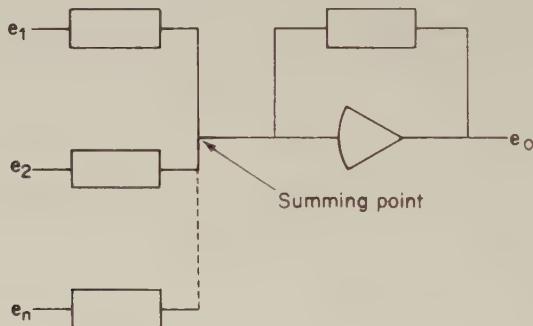


$$e_o = -A e_g$$

$$A \ggg 1$$

**Fig. 4.** Schematic symbol for an operational amplifier.

is generally connected as shown in Fig. 5. A circuit element, usually a resistor or a capacitor, is connected from the output terminal to the input terminal, referred to as the *summing point*. A number of other circuit elements, generally fixed resistors, are connected to the summing point, and the variable voltages constituting the input are connected at the other terminals of these input impedances. In employing a circuit of the type shown in Fig. 5 care must be taken that the specified voltage range of the



**Fig. 5.** General analog computing circuit.

amplifier not be exceeded. Most frequently operational amplifiers have a range from  $-100$  to  $+100$  V. Should an output voltage outside of this range be demanded in the course of a calculation, serious errors will be committed by the computer unit.

### Multiplication by a Constant

The basic component for the multiplication of variable voltages by a constant is the potentiometer. A potentiometer is a resistance element having terminal connections at each end and a sliding contact which is capable of traversing this element over its entire length, thereby permitting the variation in the resistance between the sliding contact and one of the fixed terminals. The principal characteristics which

determine the quality of such an element are the linearity and the resolution. Potentiometer linearity deviation is the maximum deviation from the best straight line which can be drawn through the points of a graph of resistance vs angular position of the sliding contact. This deviation is generally expressed as a percentage of total resistance. The resolution of a potentiometer refers to the accuracy with which any shaft setting can be obtained. Since most potentiometers are of the wire-wound type, a graph of resistance vs rotation has the shape of a staircase rather than a continuous curve. The number of steps, and hence the resolution of the element, is determined by the total number of turns of wire used in winding the potentiometer. Precision potentiometers with linearity tolerances as close as 0.01%, multiturn constructions for high resolution, and extremely low noise level are mass produced for analog applications.

The elementary multiplication circuit is shown in Fig. 6. The output voltage  $e_o$  is to be the product of the input voltage  $e_i$  and the angular position of the potentiometer

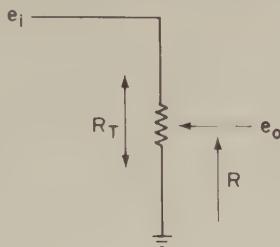


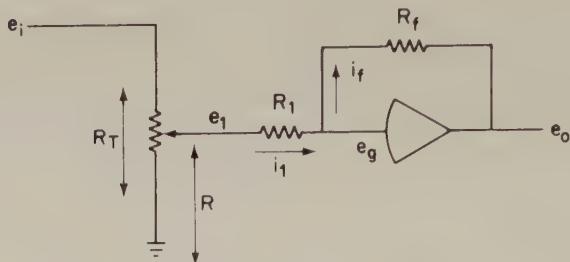
Fig. 6. Resistance potentiometer.

$R/R_T$ . As the result of the presence of the loading resistor  $R_L$  this operation is, however, subject to a *loading error* which is a function of the relative magnitudes of  $R_L$  and  $R_T$  as well as the angular position of the potentiometer. The deviation  $\Delta$  of  $e_o$  from the geometrical setting  $e_i(R/R_T)$  is

$$\Delta = \frac{R}{R_T} e_i \left[ 1 - \frac{1}{1 + \left( 1 - \frac{R}{R_T} \right) \frac{R}{R_L}} \right] \quad (1)$$

Evidently for accurate calculations it is desirable that  $R_L$  be as large as possible. Since it is often necessary to employ potentiometers to drive other analog devices having a relatively low input impedance, it is expedient to isolate the potentiometer from the load by means of an operational amplifier as shown in Fig. 7.

Provided that the operational amplifier itself draws a negligible current, i.e., its input impedance is sufficiently high, the current  $i_1$  in Fig. 7 is identical to the current  $i_f$ . In order to determine the relationship between the input and the output voltages of this circuit, use is made of Kirchhoff's current law. This law is based upon the recognition that no electrical charge can accumulate at a node point in an electrical circuit. Accordingly, the algebraic sum of all currents into and out of an electrical node



**Fig. 7.** Potentiometer in cascade with a sign changer.

must be equal to zero. Applying this law to the summing junction, identified by  $e_g$  in this circuit, yields

$$i_1 = i_f \quad (2a)$$

$$\frac{e_1 - e_g}{R_1} = \frac{e_g - e_0}{R_f} \quad (2b)$$

but the summing point voltage,  $e_g$ , is related to the output voltage by

$$e_g = -\frac{e_o}{A} \quad (3)$$

where  $A$  is a very large constant. The magnitude  $e_o$  is limited by the dynamic range of the computer, generally from  $-100$  to  $+100$  V. Provided that  $A$  is well in excess of 100,000, it is apparent that

$$e_g \simeq 0 \quad (4)$$

This is a fundamental relation in the application of operational circuits. The voltage at the summing point is always negligible in magnitude, provided only that the gain of the operational amplifier be high enough. The exact magnitude of this gain is unimportant and does not appear anywhere in the operational equations. Inserting Eq. (4) in Eq. (2) and solving for the output voltage yields

$$e_o = -\frac{R_f}{R_i} e_1 \quad (5)$$

If now  $R_i$  is large compared to the potentiometer resistance  $R_T$ , so that loading errors can be neglected, the input-output relationship of the circuit shown in Fig. 7 is

$$e_o = -\frac{R}{R_T} \cdot \frac{R_f}{R_i} e_i \quad (6)$$

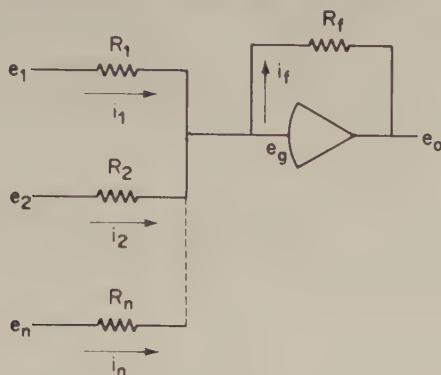
If the feedback resistor of the amplifier  $R_f$  and the input resistor  $R_i$  are equal in magnitude (generally of the order of  $1 \text{ M}\Omega$ ), the output voltage will be determined entirely by the setting of the potentiometer. Note however the sign change introduced by the operational amplifier. By making  $R_f$  larger than  $R_i$ , multiplication by negative constants greater than unity can readily be attained. If  $R = R_T$  and  $R_i = R_f$  the circuit acts as a sign changer.

In order to obviate any errors resulting from the loading of the potentiometer by the input resistor  $R_i$ , most modern electronic analog computers are equipped

with special potentiometer setting circuits which automatically compensate for the loading of the potentiometer.

### Addition–Subtraction

The addition of two or more variables is readily accomplished by means of the combination of an operational amplifier, one feedback resistor, and several input



**Fig. 8.** Analog summing circuit.

resistors, as shown in Fig. 8. Applying Kirchhoff's law to the summing junction in this circuit yields

$$i_1 + i_2 + \dots + i_n = i_f \quad (7)$$

or expressing the currents as voltage drops,

$$\frac{e_1 - e_g}{R_1} + \frac{e_2 - e_g}{R_2} + \dots + \frac{e_n - e_g}{R_n} = \frac{e_g - e_{\hat{o}}}{R_f} \quad (8)$$

Substituting Eq. (4) and solving for the output voltage,

$$e_{\hat{o}} = -R_f \left[ \frac{e_1}{R_1} + \frac{e_2}{R_2} + \dots + \frac{e_n}{R_n} \right] \quad (9)$$

If all the resistors have the same magnitudes, for example,  $1 \text{ M}\Omega$ , the output is the simple sum of all the input voltages times  $-1$ .

To subtract one variable voltage from another, the sign of one of the voltages is first reversed, using a sign changer, and is then added to the other voltage by means of an adding circuit.

### Integration–Differentiation

In order to apply the dc operational amplifier to the mathematical operation of integration with respect to time, the addition circuit is modified by replacing the feed-

back resistor by a capacitor, as shown in Fig. 9. Again applying Kirchhoff's node law to the summing junction,

$$i_1 + i_2 + \cdots + i_n = i_f \quad (10)$$

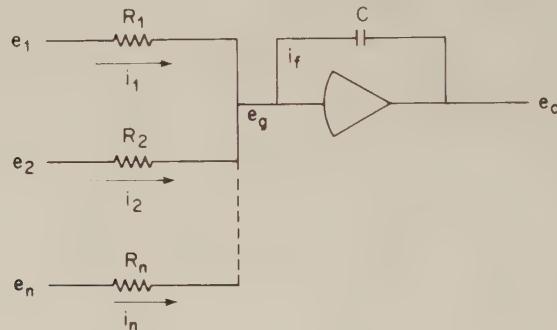
This equation becomes, upon insertion of the appropriate voltage-current relationships,

$$\frac{e_1 - e_g}{R_1} + \frac{e_2 - e_g}{R_2} + \cdots + \frac{e_n - e_g}{R_n} = C \frac{d(e_g - e_o)}{dt} \quad (11)$$

Recognizing that  $e_g = 0$  and solving for the output voltage yields

$$e_o = -\frac{1}{C} \int_0^t \left( \frac{e_1}{R_1} + \frac{e_2}{R_2} + \cdots + \frac{e_n}{R_n} \right) dt + k \quad (12)$$

The constant of integration  $k$ , the output voltage at time  $t = 0$ , must be applied to the integrator as a separate voltage excitation. Effectively, the feedback capacitor is



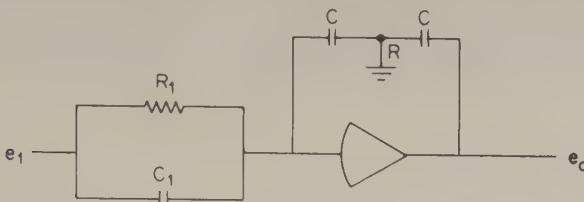
**Fig. 9.** Combined summation and integration.

charged to a voltage corresponding to the initial condition, and at time  $t = 0$  this excitation is abruptly removed from the circuit by means of a relay or a solid-state switch.

The operation of differentiation can be accomplished by employing an input capacitor and a feedback resistor. Differentiation circuits suffer from noise difficulties. While any noise, for example, electrical static or power line interference, at the input terminals of an integrator will tend to be smoothed out and removed in the process of integration, the opposite is true in differentiators. The derivative of a variable function is the rate of change or slope of the function. Any small abrupt variations in the input voltage are therefore greatly magnified in the process of differentiation and may actually tend to mask completely the desired derivative. For this reason every effort is made in employing analog computers to avoid the use of differentiators. In the solution of most differential equations, this is possible by a suitable rearrangement of the original equations.

### Complex Transfer Functions

The feedback and input elements of computing circuits utilizing operational amplifiers need not necessarily be limited to single resistors or capacitors. In general, both the input and feedback impedances may be four-terminal networks. As a simple example consider the operational circuit shown in Fig. 10. Application of Kirchhoff's



**Fig. 10.** Analog circuit with complex input and feedback networks.

law to the summing point node leads to an expression for the ratio of the input to the output voltage as

$$\frac{e_o(s)}{e_i(s)} = -\frac{z_f(s)}{z_i(s)} = -\frac{R_1(1 + 2sCR)(1 + R_1C_1s)}{sC} \quad (13)$$

where  $s$  is the Laplace transform parameter. The Laplace transform is a widely used tool for the analysis of systems characterized by ordinary differential equations. It serves in effect to transform a differential equation into an algebraic equation for easier manipulation and solution. Tables listing the transfer ratios of a wide variety of circuits of the type shown in Fig. 10 may be found in Refs. 2 and 3.

By using complex input and feedback networks in combination with one operational amplifier, it is possible to obtain functional relationships between input and output voltages which would otherwise require the use of a number of amplifiers. In addition to the economy in operational amplifiers, certain difficulties attending complex feedback loops can also be avoided by the use of complex transfer impedances. On the other hand, the utilization of complex networks may demand capacitors of odd magnitudes, not readily available. An additional problem is the application of appropriate initial conditions to the capacitors comprising the networks.

## NONLINEAR OPERATIONAL UNITS

The treatment of nonlinear differential equations requires the inclusion of nonlinear operational units within the analog computer system. These units generate transient output voltages bearing nonlinear functional relationships to the voltages applied at their input terminals, relationships which may be analytic or arbitrary in character. Multipliers generating the product of two input voltages are by far the most important

nonlinear units and are available in various design realizations. Biased diodes constitute another important tool for realizing nonlinear operational units, and are particularly useful for obtaining input-output relationships having sharp discontinuities or breaks. Trigonometric functions are generated using resolvers. The number and types of nonlinear functions which can be obtained in analog computers can be greatly increased by interconnecting the available nonlinear units with such linear devices as adders and integrators. In this section the operation of multipliers is first considered in some detail; this is followed by a description of the utilization of biased diodes for the realization of important nonlinear functions.

### Multiplication-Division

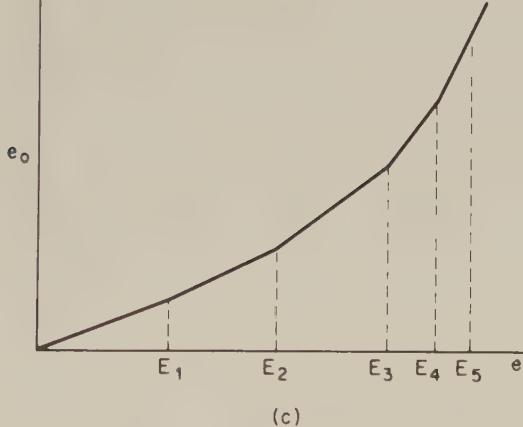
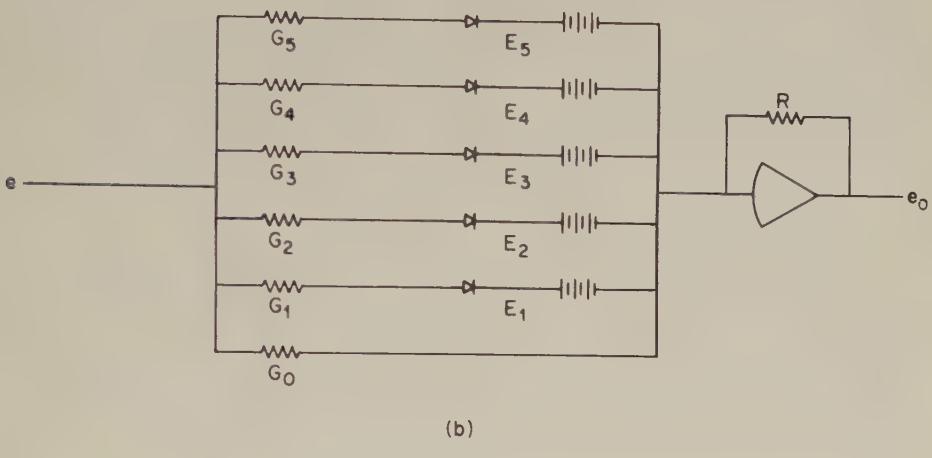
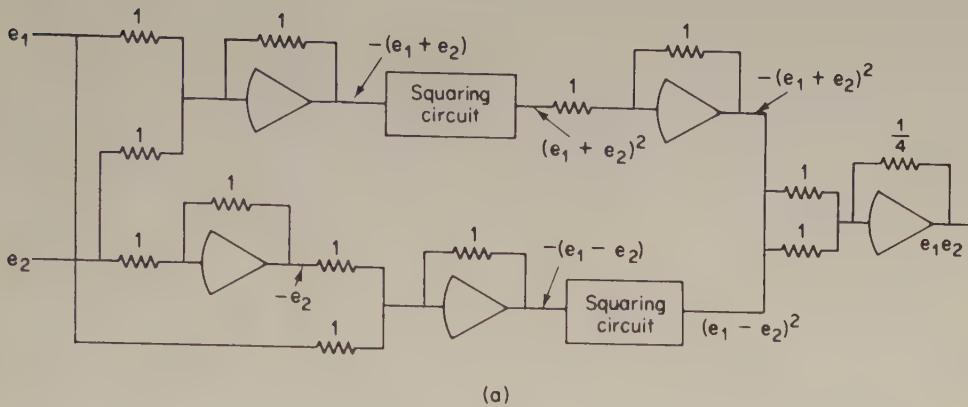
Frequently it is necessary in the treatment of engineering problems to multiply two quantities, both of which vary with time. Since the basic potentiometric multipliers previously described require the manual setting of the sliding arm of a potentiometer, this method can only be used to multiply a variable voltage by a constant. The multiplication of two variables requires additional equipment and technique. Although well over 25 different analog multiplication methods have been proposed and tried in past years, only a small number of these have achieved any considerable importance in general-purpose analog computation. In this section only the most important of these will be briefly reviewed.

The principle of operation of the most important class of electrical multipliers is based upon the equation

$$xy = \frac{1}{4}[(x + y)^2 - (x - y)^2] \quad (14)$$

Provided the squares of the sum and the difference of two variables can be generated, the product can be obtained by the simple combination of two such squaring circuits and standard adding circuits as shown in Fig. 11. A number of devices are available for the squaring operation. The most important of these involve the use of semiconductor diodes. An arrangement of diodes, bias supplies, and series conductances necessary to produce an output voltage approximately equal to the square of the input voltage is shown in Fig. 11(b). The bias voltages  $E_1$  to  $E_5$  are adjusted in such a way that as the input voltage is increased, successive diodes which are normally cut off are rendered conductive, thereby increasing the slope of the  $e_o/e_i$  curve, as shown in Fig. 11(c). A square law curve may be approximated to within  $\pm 2\%$  with three diode sections, to within  $\pm 1\%$  with five sections, and to within less than  $0.4\%$  with 15 diodes. To permit four quadrant multiplication it is necessary to employ two sets of diodes and bias supplies—one for positive and the other for negative inputs. It is possible to improve the operation of diode circuits by superimposing a small ac voltage upon the desired signal. This has the effect of rounding off the corners at the intersection of adjacent line segments, thereby obviating the effect of the sharp changes in the slope of the  $e_o/e_i$  curve.

Separate units to perform the operation of division are generally not supplied in

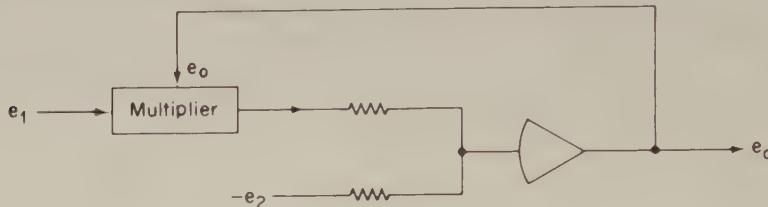


**Fig. 11.** (a) Analog function multiplier. (b) Squaring circuit. (c) Piecewise linear approximation to a square law curve.

analog computer systems. Rather, a technique termed implicit function generation is employed. Using a circuit of the type shown in Fig. 12, an equation of the type

$$e_2 + ke_1 e_o = 0 \quad (15)$$

is instrumented. The high gain amplifier is connected in a feedback loop in such a manner that it generates an output voltage which forces the output of the multiplier



**Fig. 12.** Implicit function generator used for division.

to be equal and opposite in sign to the input voltage  $e_2$ . The output voltage  $e_o$  is then proportional to the quotient  $e_2/e_1$ .

### Discontinuous Function

In many cases, particularly in the study of dynamics and control systems, it is necessary to employ computer units in which the output vs input voltage curve has sharp breaks or discontinuities. Biased-diode devices are by far the most widely used generators of such functions. One of the most important of the discontinuity devices is the *limiter*. Such a unit has a linear transfer characteristic over a specified range  $e_1$  to  $e_2$ . If the input voltage  $e_i$  exceeds  $e_2$  in the positive direction or  $e_1$  in the negative direction, the slope of the  $e_o/e_i$  curve undergoes an abrupt change. The yield point of a material, magnetic saturation, airplane control surface position, etc., introduce such limiting values. The circuit shown in Fig. 13(a) can be employed to generate the transfer function of Fig. 13(b). Here diode  $V_2$  is cut off until the output voltage  $e_o$  exceeds  $e_2$ . At that point, resistor  $R_3$  is effectively placed in parallel with  $R_2$  so that the voltage gain of the entire circuit is reduced, and the slope of the transfer function curve is abruptly changed. A similar effect occurs when  $e_o$  becomes more negative than  $e_1$ . Then  $R_2$  is placed in parallel with  $R_4$ , thereby limiting the output. If  $R_3$  and  $R_4$  are omitted from the circuit, the output voltage is prevented from exceeding  $e_2$  and  $e_1$ .

## CONTROL

As previously indicated, a major portion of the control function within an analog computer is vested in the patch board program. By means of electrical connections

linking various patch tips, the arithmetic units as well as the memory and the output units of the computer are interconnected as determined by the specific program. It is by "plugging" the patch board that the operator therefore effectively programs and controls the data flow within the computer. The organization of the patch boards

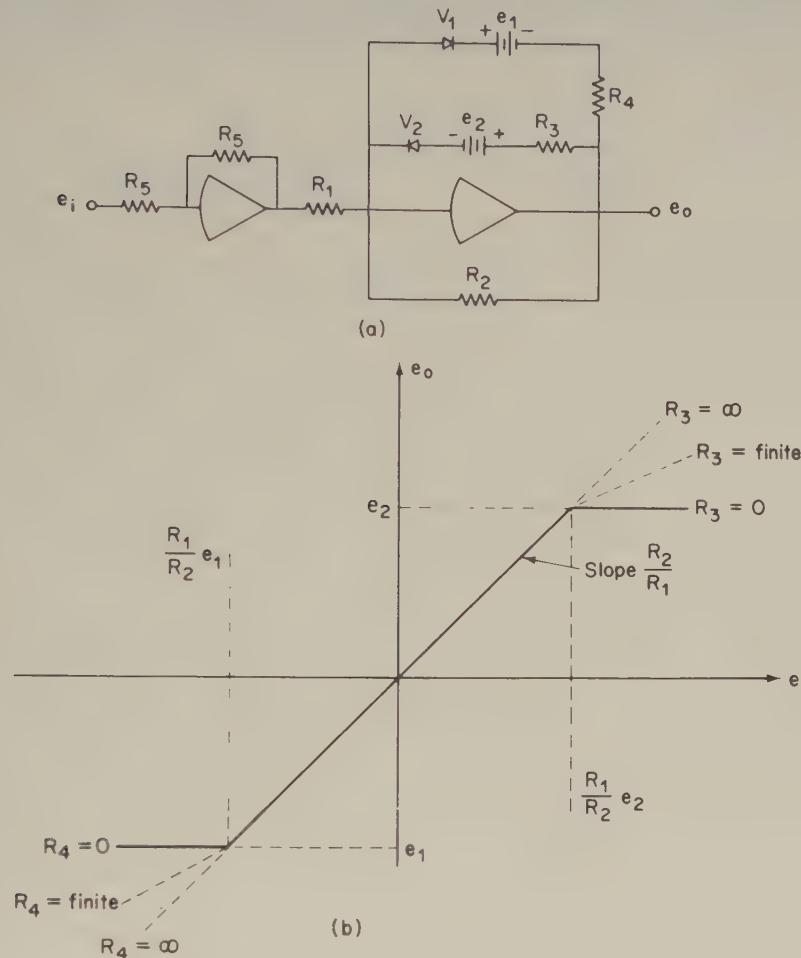


Fig. 13. (a) Analog limiter circuit. (b) Transfer characteristic.

supplied by various manufacturers of the electronic differential analyzers differ in detail, but all are arranged so as to facilitate the programming of nonlinear ordinary differential equations. Nonetheless, the effecting of the hundreds of electrical connections required for the treatment of complex systems of equations is a very time-consuming activity. For this reason there has been some effort in recent years to automate the patch board programming.

A second major control function is effected by means of one or more multiposition

switches provided on the control console. This switching system activates a large number of relays within the computer to permit the carrying out of such commands as "reset," "compute," and "hold."

In most computers at least two relays or solid-state switches are associated with each operational amplifier which participates in the operation of integration. A typical control circuit for such an amplifier is shown in Fig. 14. The relay coils

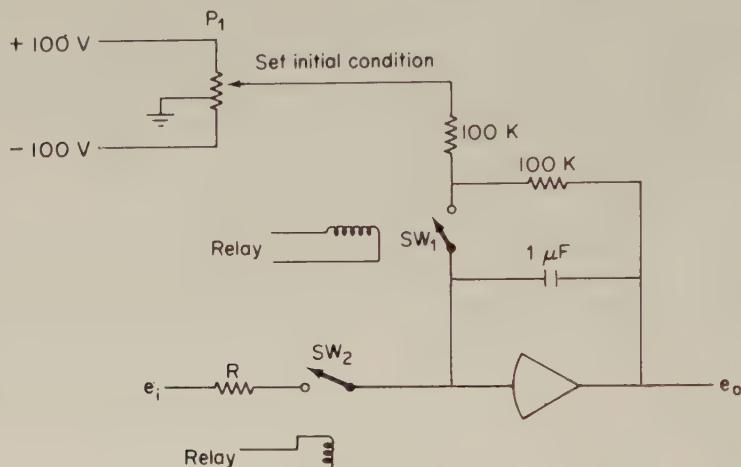


Fig. 14. Integrator with mode control switches.

activating switches  $SW_1$  and  $SW_2$  for each of the integrators are in parallel and are simultaneously activated by a switch on the computer console. Prior to a computer run, initial conditions must be applied to all integrators. This involves giving each feedback capacitor a charge corresponding to a specified initial condition. Accordingly, the computer is placed in the "reset" mode, and switch  $SW_1$  of each integrator is closed while switch  $SW_2$  of each integrator is opened. The capacitor,  $1.0 \mu\text{F}$  in this case, is therefore connected to a dc power supply. By suitable adjustment of potentiometer  $P_1$  the voltage at the output of the amplifier can be given any potential from  $-100$  to  $+100$  V.

When all initial conditions and other parameter values have been set to the desired magnitudes, the control switch is moved from the "reset" to the "compute" position. This causes switch  $SW_1$  to open and switch  $SW_2$  to close. The initial-condition power supply is now effectively out of the circuit, and the operational unit is ready to integrate any voltage applied at its input terminal,  $e_i$ . The output of the circuit of Fig. 14 is then

$$e_o = -\frac{1}{1 \times 10^{-6} R_i} \int_0^t e_i dt + IC \quad (16)$$

where  $IC$  is the initial condition applied by means of potentiometer  $P_1$ .

Occasionally in the course of solving a problem on an analog computer, it becomes desirable to interrupt briefly all dynamic processes, that is, to "freeze" the analog

computer at some point in the computing cycle. For this purpose a "hold" position is provided for the control switch. In the "hold" mode, switches SW<sub>1</sub> and SW<sub>2</sub> are both open, so that any charge stored in the feedback capacitor the instant the switch is opened is constrained to remain there. Voltages can be held in this way for several minutes without serious errors. This gives the computer operator time to change scale factors or make other necessary changes or adjustments in the computer system.

## PROGRAMMING

As in the application of digital computing techniques, the utilization of analog computers for the solution of engineering problems involves a sequence of distinct phases:

1. Reformulation of the given differential or algebraic equations into a form more suitable for computer solution.
2. Preparation of the computer program or flow chart.
3. Preparation of a detailed computer wiring diagram.
4. Preparation of the computer patch board and setting of all potentiometers.
5. Solution of the problem on the computer.

In this section, general considerations involved in the preparation of the computer flow chart are considered. As in digital computers, the flow chart can be considered to be the "interface" between the engineer with a problem to be solved and the computer operator, though frequently engineers find it practical to carry out the entire problem-solving procedure themselves. The computer flow chart indicates clearly the manner in which all necessary operational units are to be interconnected. As such, this diagram tells at a glance how much computing equipment is required and whether the available computing facility is adequate to solve a given problem. The very nature of the computer flow chart gives, in addition, important insights into the problem under study. Frequently it is possible, by examining the flow chart, to determine whether the system under analysis is stable or whether possibilities of instability exist over certain parameter ranges. Finally, the computer flow chart is of direct utility in setting up and wiring the computer system.

The construction of an effective computer flow chart implies a close familiarity with and conformance to conventional notation. Over the years, two sets of symbolism have been employed to represent analog computer units. In the first system, sometimes referred to as the "detailed notation," the input and feedback impedances of all operational units are shown on the diagram. The magnitude of each element is likewise indicated. Conventionally, resistor magnitudes are taken to be in megohms and capacitor magnitudes in microfarads unless otherwise specified. In the second system, termed the "compact notation," special symbols are employed to identify potentiometers, adders, and integrators without showing actual resistors or capaci-

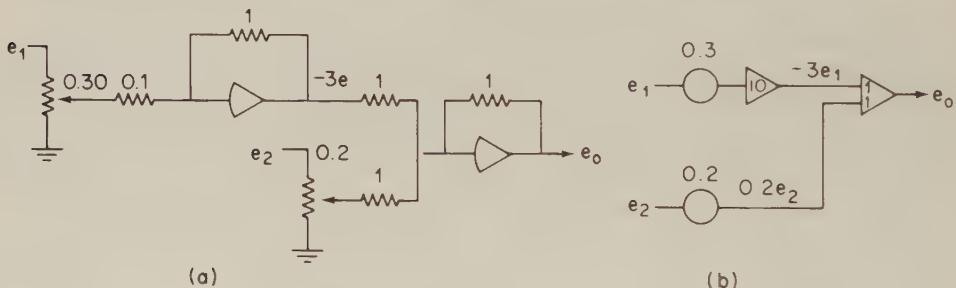
tors; rather, the feedback to input impedance ratio is indicated for each input terminal. In neither notation is the ground bus shown explicitly, it being understood that all voltages indicated in the diagram are referred to computer ground. Table I represents a summary of the more important symbols employed in both systems of notation. The symbol ( $t$ ) is included to emphasize that the independent variable is represented by time. In principle these elements are identical regardless whether they are employed in one-shot or repetitive differential analyzers. The chief difference lies in the fact that the repetitive analog computer components must have much wider bandwidths extending well beyond 10 kilocycles, and that integrators have much shorter time constants.

To construct a computer flow chart, indicating the solution of a specified equation, units of the type shown in Table I are interconnected in a suitable manner. This interconnection invariably involves the linking of input and output terminals of computer units. Frequently closed loops are formed in this manner. The general technique for constructing such a flow chart is illustrated below for several simple problems.

Consider first that it is desired to obtain a transient voltage  $e_o$  which is related to two other transient voltages  $e_1$  and  $e_2$  by

$$e_o = 3e_1 - 0.2e_2 \quad (17)$$

Figure 15(a) illustrates how this may be accomplished with the aid of a sign changer and an adding circuit. The same circuit is shown in compact notation in Fig. 15(b).



**Fig. 15.** (a) Analog circuit implementation of Eq. (17). (b) The same circuit using compact notation.

Note that the input and feedback resistors of the sign changers and amplifiers are assigned round values, such as 1.0, 0.5, and 0.1, and that all parameters (3.0 and 0.2 in this case) are introduced by means of potentiometers. This permits the use of a very limited number of fixed precision resistors, which may be housed in a temperature-controlled environment. In utilizing adders and integrators it is important not to overlook the phase shift, or multiplication by  $-1$ , inherent in these operations.

As a second example, consider the generation of an output voltage  $e_o$  which is related to three transient input voltages by the relationship

$$e_o = -2e_1 + \int_0^t (e_2 - e_3) dt \quad (18)$$

Computer Operation	Input $e_i$	Output $e_o$	Detailed Schematic	Compact Schematic
Constant multiplication	$f(t)$	$kf(t), k \leq 1$		
Addition	$f_i(t)$	$-\sum_{i=1}^n k_i f_i(t)$		
Integration	$f_i(t)$	$-\int_0^t \left( \sum_{i=1}^n k_i f_i(t) \right) dt + IC$		
Complex transfer functions	$f(s)$	$-\frac{Z_f(s)}{Z_i(s)} f(s)$		
Multiplication	$f(t), g(t)$	$kf(t) g(t)$		
Function generation	$g(t)$	$f[g(t)]$		
High-gain amplification	$f(t)$	$-gf(t)$		

**TABLE 1**  
Detail and Compact Schematic Symbols for Analog Computer Operations

where the initial condition on the integral is specified to be zero. Figure 16(a) and (b) represents suitable computer programs in the detailed and compact notations, respectively.

As an example of the programming of nonlinear operations consider the generation

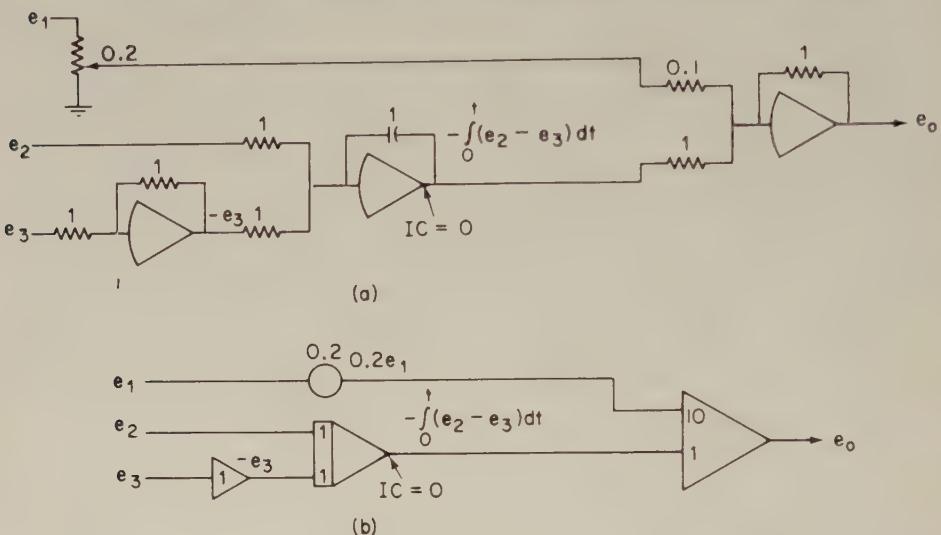


Fig. 16. (a) Analog implementation of Eq. (18). (b) Compact notation.

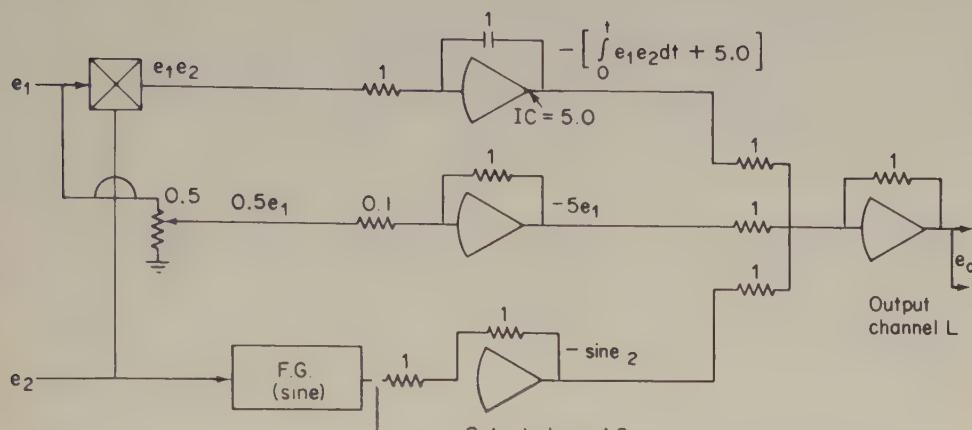
of an output voltage  $e_o$  which is related to transient input voltages  $e_1$  and  $e_2$  according to

$$e_o = \int_0^t e_1 e_2 dt + 5e_1 + \sin e_2 \quad (19)$$

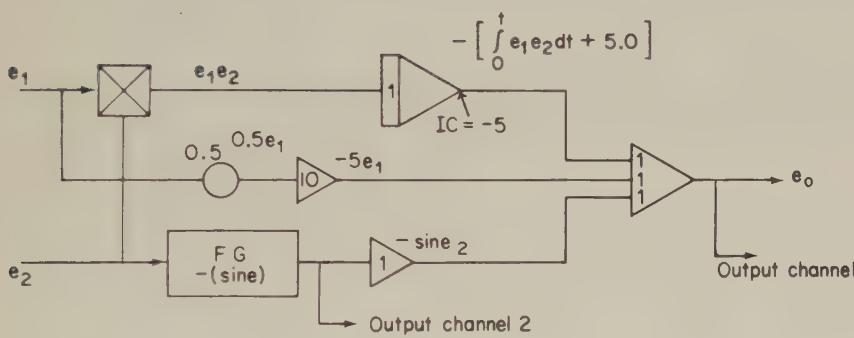
where the initial condition on the integral is specified to be 5.0. The computer schematic in the two alternative notations is shown in Fig. 17(a) and (b). Note that provision has been made in the circuit for readout of two transient wave shapes corresponding respectively to  $e_o$  and  $\sin e_2$ . These readouts could be generated simultaneously on parallel strip chart recorders or, in the case of repetitive analog computers, as two simultaneous traces on the screen of an oscilloscope. Note also that the initial condition at the integrator is set to -5.0, since the output of the integrator is indicated to be minus the integral specified in Eq. (19).

Frequently in the application of analog computers, closed loops are formed in the analog computer system. Consider, for example, the placing of a connector from the output terminal  $e_o$  to the input terminal  $e_3$  in Fig. 16(b). The modified diagram would then be that shown in Fig. 18. The input-output relation of the circuit can be derived by substituting  $e_o$  for  $e_3$  in Eq. (18) so that

$$e_o = -2e_1 + \int_0^t (e_2 - e_o) dt \quad (20)$$



(a)



(b)

Fig. 17. (a) Analog implementation of Eq. (19). (b) Compact equivalent.

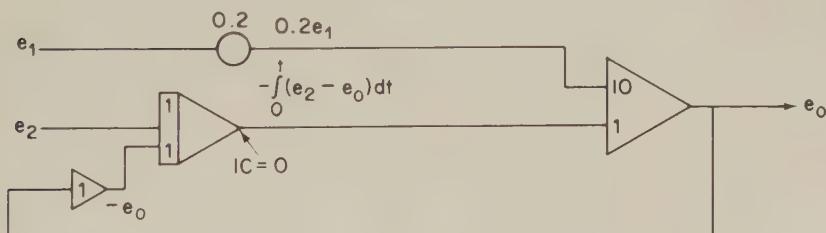


Fig. 18. Analog mechanization of Eq. (20)

Using Laplace transforms, Eq. (20) becomes

$$\bar{e}_\delta = -2\bar{e}_1 + \frac{1}{s} |\bar{e}_2 - \bar{e}_\delta| \quad (21)$$

where  $s$  is the complex frequency parameter. Solving for  $\bar{e}_\delta$ ,

$$\bar{e}_\delta = \frac{s\bar{e}_2 - 2\bar{e}_1}{s + 1} \quad (22)$$

In order that the computer systems shown in Figs. 15-18, and indeed all analog computer units, function in the specified manner, it is important to insure that the dynamic range of the computer units not be exceeded. For most computers this implies that the voltage at all points in the computer system must at all times remain in the  $-100$  to  $+100$  V range. If this range is exceeded, overload indicators light up and the calculation becomes effectively worthless. To obviate this possibility, suitable scale factors must be assigned to all dependent computer variables, i.e., voltages  $e_1$ ,  $e_2$ ,  $e_3$ , and  $e_o$  in the above examples. Occasionally, particularly in the treatment of nonlinear problems, the dynamic range of certain variables is very difficult to predict. In that case several trial computer runs must be performed before suitable scale factors can be determined.

## PROBLEM SETUP AND CHECKOUT

In digital computations the programming and the coding phases of problem preparation represent separate and distinct efforts. Even if compilers are employed, the translation of the flow chart into a useful machine program requires a knowledge of coding techniques and detailed familiarity with the specific digital computer to be used. In analog computations, on the other hand, the transition from the flow chart to the patch board program is quite direct. To be sure, the analog computer operator must become familiar with the organization of the patch board supplied with the specific computer to be used, but these boards are usually labeled and color-coded in such a way that the significance of each hole in the board is readily apparent. Program patching then requires only that each unit and connection shown in the flow chart be represented by corresponding connections on the patch board.

As the first step in problem setup, the flow chart prepared in the preceding phase of the preparation procedure is closely examined to determine whether the requirements of the program correspond to available equipment within the computer. Frequently it proves advantageous to rearrange the flow chart so as to optimize the utilization of available equipment. In general, the problem of synthesizing a computer circuit for the solution of a specified set of equations does not have a unique solution. It is therefore expedient to consider a number of alternative realizations and to select the one which appears most suitable in view of the equipment on hand. Once a final flow chart has been chosen, each of the operational units, amplifiers, poten-

tiometers, multipliers, etc., is assigned identifying numbers. These numbers establish the correspondences between the computing units shown in the flow chart and those existing in the electronic differential analyzer to be employed for solution and can be regarded as addresses. The problem board can now be wired in accordance with the flow chart. In complex problems, several hundred electrical connections may have to be made, and extreme care is necessary to avoid wiring error. Finally, specified numerical data, including particularly initial conditions, parameter magnitudes, and scale factors, are introduced as potentiometer settings.

Due to the likelihood of errors in the patch board program, the actual solutions of complex problems on a computer are always preceded by several so-called checkout runs. For this purpose a combination of parameters and initial conditions is selected which provides a solution that is already known exactly or at least approximately. In this way errors in programming or wiring can be detected. Frequently a static check run is also performed. For this purpose the feedback capacitors in all integrators are shunted by resistors, so that the transient phase of the problem solution is eliminated, and a steady-state response results. A digital voltmeter can then be employed to compare the voltages existing at various points in the computer circuit with those predicted from an analysis of the flow chart.

## REFERENCES

1. G. A. Korn and T. M. Korn, *Electronic Analog and Hybrid Computers*, McGraw-Hill, New York, 1964.
2. W. J. Karplus and W. W. Soroka, *Analog Methods: Computation and Simulation*, 2nd ed., McGraw-Hill, New York, 1959.
3. A. S. Jackson, *Analog Computation*, McGraw-Hill, New York, 1960.
4. R. Tomovic and W. J. Karplus, *High-Speed Analog Computers*, Wiley, New York, 1962; Dover Publications, New York, 1970.
5. G. E. Tobey, J. G. Graeme, and L. P. Huelsman, *Operational Amplifiers—Design and Application*, McGraw-Hill, New York, 1971.

*Walter J. Karplus*

# ANALYSIS OF VARIANCE

## INTRODUCTION

The phrase “analysis of variance” refers to a group of statistical procedures associated with the partitioning of the observed total variability in a set of data into “accounted for” and “unaccounted for” sources of variation. The specific type of decomposition or partition of the observed variation is dependent upon the assumed structure of the data, reflected by an underlying statistical model. The outstanding feature of the analysis of variance (ANOVA) technique is that it is a powerful and systematic method for statistical inference, i.e., it provides point or interval estimates of unknown parameters and/or tests of hypotheses about unknown parameters of the model.

The ANOVA was developed in the early 1920s by R. A. Fisher [1]. Early applications were primarily concerned with hypothesis testing in biological and agricultural experimental sciences. Over the years, the ANOVA has found wide usage in all areas of science and technology, and Fisher’s pioneering work has been extended to what is usually referred to as the general linear model.

## A SINGLE TREATMENT OR POPULATION

In order to introduce certain notions that will be used later on, we start with the trivial problem of a single treatment or population. Suppose an engineer is interested in determining the strength of a material which he purchases. For this hypothetical example, suppose he randomly selects four test specimens, and the results turn out to be 68, 80, 64, 66. (Questions concerning the number of test specimens and the manner of selecting them will be discussed later on.) The engineer views the “strength of the material” as a conceptual mean value of all possible test specimens of the material under consideration, of which he has but a random sample of size four. We postulate a simple population model

$$Y_i = \eta + \varepsilon_i \quad (1)$$

where  $Y_i$  represents the measured strength on the  $i$ th specimen,  $\eta$  is the mean strength of the population under consideration, and  $\varepsilon_i$  is the random error associated with the  $i$ th specimen. That is,  $\varepsilon_i$  is the discrepancy between  $Y_i$  and its expected value, or  $\varepsilon_i = Y_i - \eta$ . Since we have defined  $\eta$  as the mean (expected value) of all the  $Y_i$ , the mean of all the  $\varepsilon_i$  is zero. The variance of the  $\varepsilon_i$  is  $\sigma^2$ . Thus, we are stating that the

population of  $Y_i$  has a mean of  $\eta$  and variance of  $\sigma^2$  (or, standard deviation of  $\sigma$ ). We might further postulate that the  $e_i$  (or  $Y_i$ ) are normally distributed, and use the notation  $e_i \sim NID(0, \sigma^2)$  to denote that the errors are normally and independently distributed with a mean of zero and variance  $\sigma^2$ .

All of the above discussion refers to a way the engineer views the population of all possible strength measurements. If he knew the constants or parameters in the population model, he would not bother breaking the four test specimens. However, he can construct a corresponding sample model of the form

$$Y_i = m + e_i, \quad i = 1, 2, 3, 4 \quad (2)$$

where  $m$  is an estimate of  $\eta$  based upon his data. The method of estimation used in connection with the ANOVA is least squares. The least squares estimator of  $\eta$  is

$$m = \frac{1}{r} \sum_{i=1}^r Y_i$$

or the arithmetic average of the observed data. In this example  $m = \frac{1}{4}[68 + 80 + 64 + 66] = 69.5$ . In a corresponding manner, the residuals,  $e_i$ , equal  $Y_i - m$ . Thus, we view our data as

$$68 = 69.5 - 1.5$$

$$80 = 69.5 + 10.5$$

$$64 = 69.5 - 5.5$$

$$66 = 69.5 - 3.5$$

Note that the residuals sum or average to zero. We can estimate  $\sigma^2$  as

$$s^2 = \frac{1}{\nu} \sum_{i=1}^r e_i^2$$

where  $\nu$  represents the degrees of freedom ( $df$ )—in this case,  $r - 1$  or 3. For these data,  $s^2 = \frac{1}{3}[(-1.5)^2 + (10.5)^2 + (-5.5)^2 + (-3.5)^2] = 51.67$ . As a preliminary to the ANOVA technique, observe that

$$\sum_{i=1}^4 Y_i^2 = 19,476; \quad \sum_{i=1}^4 m^2 = 19,321; \quad \sum_{i=1}^4 e_i^2 = 155$$

and

$$\sum_{i=1}^4 Y_i^2 = \sum_{i=1}^4 m^2 + \sum_{i=1}^4 e_i^2$$

We now proceed to the problem of comparing  $v$  treatments or populations.

## COMPARING $v$ TREATMENTS OR POPULATIONS

Suppose the engineer is interested in comparing  $v$  materials in terms of their strength. For each of the  $v$  materials of interest, suppose he selects  $r$  test specimens at random. Furthermore, suppose that the order of testing the  $N = vr$  test specimens is determined by referring to a table of random numbers. This is referred to as a single factor experiment run in a completely randomized design, or by some authors, as a one-way classification. The engineer can envision the population model for this setup as

$$Y_{ij} = \eta_i + \varepsilon_{ij} \quad (3)$$

where  $Y_{ij}$  represents the measured strength of the  $j$ th test specimen of the  $i$ th material,  $\eta_i$  represents the mean strength of the  $i$ th material, and  $\varepsilon_{ij}$  is the random error associated with  $Y_{ij}$ . In comparative experiments, it is convenient to rewrite Eq. (3) as follows:

$$Y_{ij} = \eta + \tau_i + \varepsilon_{ij} \quad (4)$$

where  $\eta$  is the mean of the  $v$  materials, and  $\tau_i = \eta_i - \eta$ . Since the first moment about the centroid, or, the sum of deviations about a mean equals zero,

$$\sum_{i=1}^v \tau_i = 0$$

The standard assumption concerning the errors of this model is that the errors are normally and independently distributed with mean zero and common variance  $\sigma^2$ , or  $\varepsilon_{ij} \sim NID(0, \sigma^2)$ , i.e.,  $\sigma^2$  is constant or common to all  $v$  treatments.

Suppose the engineer is interested in assessing the strength of five commercially available materials, and randomly selects four test specimens from each material. Here,  $v = 5$ ,  $r = 4$ , so he will collect  $N = 20$  strength measurements. His corresponding sample model can be represented as

$$Y_{ij} = m + t_i + e_{ij} \quad (5)$$

where  $i = 1, 2, 3, 4, 5$  and  $j = 1, 2, 3, 4$ . Suppose Table 1 represents the data collected.

**TABLE 1**

Material	Strength measurements
A	61, 65, 60, 75
B	60, 66, 55, 70
C	64, 66, 68, 80
D	69, 80, 72, 80
E	83, 83, 70, 89

Using the method of least squares, our best estimate of  $\eta$  is

$$m = \frac{1}{N} \sum_{i=1}^5 \sum_{j=1}^4 Y_{ij} = \frac{1}{20} (1416) = 70.8$$

If we impose the reasonable side condition or constraint of

$$\sum_{i=1}^5 t_i = 0$$

the least squares estimators of  $\tau_i$  turn out to be  $t_i = \bar{Y}_{i\cdot} - m$  (the "dot" notation implies summation over the subscripts that the dots replace). That is, the estimate of the effect of the  $i$ th material is the difference between the average strength of the specimens obtained from the  $i$ th material and the grand average. With these data, the observed averages of the five materials were 65.25, 62.75, 69.50, 75.25, and 81.25, respectively. Since  $m = 70.8$ , then  $t_1 = -5.55$ ,  $t_2 = -8.05$ ,  $t_3 = -1.30$ ,  $t_4 = 4.45$ , and  $t_5 = 10.45$ . Thus, using the sample model (5), we can represent the first observation on material  $A$ , 61, as

$$61 = 70.8 - 5.55 - 4.25$$

In a similar fashion, we can represent the entire collection of observations as presented in Table 2.

TABLE 2

$Y_{ij}$	=	$m$	
(61 65 60 75)	=	(70.8 70.8 70.8 70.8)	
(60 66 55 70)	=	(70.8 70.8 70.8 70.8)	
(64 66 68 80)	=	(70.8 70.8 70.8 70.8)	
(69 80 72 80)	=	(70.8 70.8 70.8 70.8)	
(83 83 70 89)	=	(70.8 70.8 70.8 70.8)	
	+	$t_i$	
	+	(-5.55 -5.55 -5.55 -5.55)	
	+	(-8.05 -8.05 -8.05 -8.05)	
	+	(-1.30 -1.30 -1.30 -1.30)	
	+	(4.45 4.45 4.45 4.45)	
	+	(10.45 10.45 10.45 10.45)	
	+	$e_{ij}$	
	+	(-4.25 -0.25 -5.25 9.75)	
	+	(-2.75 3.25 -7.75 7.25)	
	+	(-5.50 -3.50 -1.50 10.50)	
	+	(-6.25 4.75 -3.25 4.75)	
	+	(1.75 1.75 -11.25 7.75)	

The sum of squares of each element in each array is as follows:

$$\sum_{i=1}^5 \sum_{j=1}^4 Y_{ij}^2 = 101,872$$

$$\sum_i \sum_j m^2 = Nm^2 = 100,252.8$$

$$\sum_i \sum_j t_i^2 = 4 \sum_i t_i^2 = 905.2$$

$$\sum_i \sum_j e_{ij}^2 = 714$$

At this point, we note that

$$\sum_i \sum_j Y_{ij}^2 = \sum_i \sum_j m^2 + \sum_i \sum_j t_i^2 + \sum_i \sum_j e_{ij}^2$$

We will return to these arrays in a moment.

If we substitute the least squares estimators in the sample model (5), we obtain

$$Y_{ij} = m + (\bar{Y}_{i\cdot} - m) + (Y_{ij} - \bar{Y}_{i\cdot}) \quad (6)$$

Note that Eq. (6) is an algebraic identity, and may be rewritten as

$$(Y_{ij} - m) = (\bar{Y}_{i\cdot} - m) + (Y_{ij} - \bar{Y}_{i\cdot}) \quad (7)$$

This identity states that the deviation of an observation from the grand average equals the sum of the deviation of its treatment average from the grand average and the deviation of the observation from its treatment average. If we square each observation and sum over all observations, we obtain

$$\sum_i \sum_j (Y_{ij} - m)^2 = \sum_i \sum_j (\bar{Y}_{i\cdot} - m)^2 + \sum_i \sum_j (Y_{ij} - \bar{Y}_{i\cdot})^2 \quad (8)$$

The cross-product term vanishes in Eq. (8). In ANOVA terminology the left-hand side of Eq. (8) is referred to as the total sum of squares (TSS). The first term on the right-hand side is called the between treatments sum of squares or the sum of squares due to treatments (SST). The last term in Eq. (8) is the within treatments sum of squares or the sum of squares due to error (SSE). For this sample problem, TSS = 1619.2, SST = 905.2, and SSE = 714. Some authors of texts and computer programs label  $\sum_i \sum_j Y_{ij}^2$  as the total sum of squares and separately list

$$\sum_i \sum_j m^2 \equiv Nm^2 \equiv \frac{1}{N} \left( \sum_i \sum_j Y_{ij} \right)^2$$

as the correction factor for the mean. That is, they outline the sums of squares on the basis of Eq. (6) rather than Eq. (7). One way of being sure which convention is used is to look at the listed degrees of freedom associated with the total sum of squares. The raw sum of squares of the  $N$  observations has  $N$  df and the correction factor for the mean has 1 df, i.e., we have used 1 df for estimating  $\eta$ . The sum of squares of deviations about the grand average, TSS, is based upon  $N - 1$  df.

The decomposition of the variability in the data is systematically listed in an ANOVA table. Table 3 represents the ANOVA table for the data on the five materials.

TABLE 3

Source of variation	df	Sum of squares	Mean square	Expected mean square	F
Between materials	4	905.2	226.3	$\sigma^2 + \sum_i \tau_i^2$	4.75
Error	15	714.0	47.6	$\sigma^2$	
Total	19	1619.2			

The first column lists the sources of variation in the experiment consistent with the underlying statistical model. The second column exhibits the df associated with each

sum of squares. That is, the total was based upon the deviations of the 20 observations about  $m$ , which was calculated from the data. Since the sum of deviations about an average is zero, TSS is based upon 19  $df$ . The 4  $df$  for materials represents the weighted comparison of the five material averages with the grand average  $m$  (recall that a side condition was imposed on the treatment effects). The  $df$  for error is the difference between 19 and 4, or 15. We can obtain an estimate of experimental error from the four observations on a given material, based on 3  $df$ . Since we are assuming that the true variance within a material is independent of or common to all five materials, we pool the sums of squares to obtain an overall estimate based upon 15  $df$ . The mean square is the ratio of the sum of squares to its corresponding  $df$ . This scaling places the variation on a common, per unit basis. The next column, the expected mean square, is generally omitted in canned computer programs. In fact, it is not terribly useful for this simple illustrative example, but it is indispensable for complex experimental programs. This column indicates what each mean square is estimating, based upon the underlying population model. Finally, the  $F$  represents a ratio of observed mean squares that is used in testing hypotheses.

Upon examining the ANOVA table, we observe that the error mean square, 47.6, is an unbiased estimate of  $\sigma^2$ . Suppose one wished to test the hypothesis that the five materials all have equivalent strength. If the true mean strengths of all five materials were equal, they would all equal  $\eta$ , and the  $\tau_i$  would be zero. We state this null hypothesis,  $H_0: \tau_1 = \dots = \tau_5 = 0$ , against the alternate hypothesis,  $H_1: \text{not all } \tau_i = 0$ . If the null hypothesis were true, note that the expectation of the materials mean square would be  $\sigma^2$ . Thus, to test this hypothesis, we form the ratio of the materials mean square to the error mean square, and in this example,  $F = 226.3/47.6 = 4.75$ . We compare this number with the  $100(1 - \alpha)$  percentile of the widely tabulated  $F$ -distribution based upon 4 and 15  $df$ . For example, if we set the error rate for a Type I error (rejecting the null hypothesis when it is indeed true) to 0.05, the critical value for the test is  $F_{0.95}(4, 15) = 3.06$ . Since  $4.75 > 3.06$ , we reject the null hypothesis, or state that the strengths of these five materials are statistically significantly different at the 5% level. We reserve the discussion of detailed comparisons among the materials to a later section.

Statistics texts rely heavily on algebraic identities in providing formulas for partitioning the sums of squares in the ANOVA. Unfortunately, the ones most commonly given are most appropriate for desk calculator ease of usage rather than electronic computer efficiency and precision. For the single factor experiment for comparing  $v$  selected treatments, each replicated  $r$  times in a completely randomized design, the formulas for the sums of squares are as follows:

$$\begin{aligned} \text{TSS} &= \sum_{i=1}^v \sum_{j=1}^r (Y_{ij} - m)^2 = \sum_i \sum_j Y_{ij}^2 - \frac{1}{vr} \left( \sum_i \sum_j Y_{ij} \right)^2 = \sum_i \sum_j Y_{ij}^2 - Nm^2 \\ \text{SST} &= r \sum_i t_i^2 \equiv r \sum_i (\bar{Y}_{i\cdot} - m)^2 \equiv \frac{1}{r} \sum_i T_i^2 - Nm^2 \end{aligned}$$

where  $T_i$  represents the total of the  $r$  observations on the  $i$ th treatment.

$$\text{SSE} = \sum_i \sum_j e_{ij}^2 \equiv \sum_i \sum_j (Y_{ij} - \bar{Y}_{i\cdot})^2 \equiv \text{TSS} - \text{SST}$$

For this setup, the ANOVA table would be as given in Table 4, where MST represents the between treatments mean square, and  $s_e^2$  denotes the within treatments or error mean square.

TABLE 4

Source of variation	<i>df</i>	Sum of squares	Mean square	Expected mean square	<i>F</i>
Between treatments	$v - 1$	SST	$SST/(v - 1) = MST$	$\sigma^2 + r \sum_i \tau_i^2 / (v - 1)$	$MST/s_e^2$
Error	$v(r - 1)$	SSE	$SSE/[v(r - 1)] = s_e^2$	$\sigma^2$	
Total	$vr - 1$	TSS			

MST represents the between treatments mean square, and  $s_e^2$  denotes the within treatments or error mean square.

To perform the standard calculations of the ANOVA and estimate the unknown parameters, we need only assume that the  $\epsilon_{ij}$  are uncorrelated and have common unknown variance  $\sigma^2$ . However, to test hypotheses and/or construct confidence intervals for the parameters, we further postulate the probability distribution of the errors. The standard reference distribution is the Gaussian or normal distribution, but normality is not a critical assumption for applications of this type of model. However, when data analysts resort to transformations in order to stabilize the variance for each treatment—e.g., the logarithm of strength might be appropriate for widely differing materials—the transformation usually also improves the approximation to the assumption of normality. Some general comments concerning the violation of standard assumptions of the ANOVA will be presented in a later section.

## COMPARING $v$ TREATMENTS IN A RANDOMIZED BLOCK DESIGN

In general, repeated observations taken close together are more homogeneous than corresponding observations taken over a greater separation of time or space. The process of dividing the experimental units into relatively homogeneous subsets and randomly assigning the  $v$  treatments to each subset is known as blocking. It is a noise-reducing procedure in that this group-to-group or block-to-block variation can be isolated and does not appear as part of the measure of experimental error. Hence, the comparisons among the  $v$  treatments may be made with greater precision. Some examples of blocks are time periods, batches of material, operators, test machines, and locations within a furnace or ingot.

Suppose that we are interested in comparing  $v$  materials but intend to use  $r$  testing

machines to obtain our data. We randomly assign a test specimen of each material to each test machine, and randomize the order of testing. Now if there is a bias among test machines, the bias will equally affect all  $v$  materials so that the comparisons among materials will be unaffected. Furthermore, any bias among test machines can be isolated from the measure of uncontrolled variation or experimental error, thus increasing the precision relative to unrestricted randomization over the  $r$  test machines.

An appropriate population model may be written as

$$Y_{ij} = \eta + \tau_i + \beta_j + \varepsilon_{ij} \quad (9)$$

$$i = 1, \dots, v; \quad j = 1, \dots, r; \quad \sum_{i=1}^v \tau_i = \sum_{j=1}^r \beta_j = 0; \quad \varepsilon_{ij} \sim NID(0, \sigma^2)$$

where  $Y_{ij}$  is the measured strength of the  $i$ th material on the  $j$ th testing machine;  $\eta$  is the true mean of the experiment;  $\tau_i$  is the true effect of the  $i$ th material, measured from  $\eta$ ;  $\beta_j$  is the true effect of the  $j$ th testing machine, measured from  $\eta$ ; and  $\varepsilon_{ij}$  is the true random error associated with  $Y_{ij}$ .

Consistent with the physical act of balancing the allocation of the  $v$  materials to each test machine, or, restricting the randomization, we introduce a set of  $r$  parameters into the model,  $\beta_j$ , to represent the relative biases among test machines.

The corresponding sample model would be written as

$$Y_{ij} = m + t_i + b_j + e_{ij} \quad (10)$$

where  $m$ ,  $t_i$ , and  $b_j$  are the corresponding least squares estimates, based upon the data, of the unknown parameters. Again, side conditions are placed upon the  $t_i$  and  $b_j$  such that  $\sum_i t_i = \sum_j b_j = 0$ . The least squares estimators turn out to be

$$m = \frac{1}{vr} \sum_i \sum_j Y_{ij}, \quad t_i = \bar{Y}_{i\cdot} - m, \quad \text{and} \quad b_j = \bar{Y}_{\cdot j} - m$$

Suppose that the previously used set of data were obtained from a randomized block design in which a specimen of each of five materials was randomly assigned to each of four testing machines. The rearranged data appear as in Table 5.

TABLE 5

Machine	$M_1$	$M_2$	$M_3$	$M_4$	Material total	$\bar{Y}_{i\cdot}$	$t_i$
$A$	61	65	60	75	261	65.25	-5.55
$B$	60	66	55	70	251	62.75	-8.05
$C$	64	66	68	80	278	69.50	-1.30
$D$	69	80	72	80	301	75.25	4.45
$E$	83	83	70	89	325	81.25	10.45
Machine total	337	360	325	394	1416		
$\bar{Y}_{\cdot j}$	67.4	72.0	65.0	78.8		70.8	
$b_j$	-3.4	1.2	-5.8	8.0			0

Since this is the same set of data as was previously used, note that  $m$  and  $t_i$  are identical to the previous case. With machines represented as columns in the above table, we can easily obtain least squares estimates of the block effects. Thus, we can represent the data set, consistent with the sample model, as in Table 6. Note that the

TABLE 6

$Y_{ij}$	=	$m$	+	$t_i$
$\begin{pmatrix} 61 & 65 & 60 & 75 \\ 60 & 66 & 55 & 70 \\ 64 & 66 & 68 & 80 \\ 69 & 80 & 72 & 80 \\ 83 & 83 & 70 & 89 \end{pmatrix}$	$=$	$\begin{pmatrix} 70.8 & 70.8 & 70.8 & 70.8 \\ 70.8 & 70.8 & 70.8 & 70.8 \\ 70.8 & 70.8 & 70.8 & 70.8 \\ 70.8 & 70.8 & 70.8 & 70.8 \\ 70.8 & 70.8 & 70.8 & 70.8 \end{pmatrix}$	$+$	$\begin{pmatrix} -5.55 & -5.55 & -5.55 & -5.55 \\ -8.05 & -8.05 & -8.05 & -8.05 \\ -1.30 & -1.30 & -1.30 & -1.30 \\ 4.45 & 4.45 & 4.45 & 4.45 \\ 10.45 & 10.45 & 10.45 & 10.45 \end{pmatrix}$
	$+$	$b_j$	$+$	$e_{ij}$
	$+$	$\begin{pmatrix} -3.4 & 1.2 & -5.8 & 8.0 \\ -3.4 & 1.2 & -5.8 & 8.0 \\ -3.4 & 1.2 & -5.8 & 8.0 \\ -3.4 & 1.2 & -5.8 & 8.0 \\ -3.4 & 1.2 & -5.8 & 8.0 \end{pmatrix}$	$+$	$\begin{pmatrix} -0.85 & -1.45 & 0.55 & 1.75 \\ 0.65 & 2.05 & -1.95 & -0.75 \\ -2.10 & -4.70 & 4.30 & 2.50 \\ -2.85 & 3.55 & 2.55 & -3.25 \\ 5.15 & 0.55 & -5.45 & -0.25 \end{pmatrix}$

inclusion of  $b_j$  caused a general decrease in the magnitudes of the residuals,  $e_{ij}$ , as compared to Table 2. This is because the former residuals have been reduced by the estimated block effects.

For randomized complete block designs with  $v$  treatments and  $r$  blocks, the sums of squares may be partitioned as follows:

$$\text{TSS} = \sum_{i=1}^v \sum_{j=1}^r (Y_{ij} - m)^2 \equiv \sum_i \sum_j Y_{ij}^2 - \frac{1}{vr} \left( \sum_i \sum_j Y_{ij} \right)^2 \equiv \sum_i \sum_j Y_{ij}^2 - Nm^2$$

Let  $T_i$  represent the total of the  $r$  observations on the  $i$ th treatment and  $B_j$  represent the sum of the  $v$  observations in the  $j$ th block.

$$\text{SST} = r \sum_i t_i^2 \equiv r \sum_i (\bar{Y}_{i\cdot} - m)^2 \equiv \frac{1}{r} \sum_i T_i^2 - Nm^2$$

$$\text{SSB} = v \sum_j b_j^2 \equiv v \sum_j (\bar{Y}_{\cdot j} - m)^2 \equiv \frac{1}{v} \sum_j B_j^2 - Nm^2$$

$$\text{SSE} = \sum_i \sum_j e_{ij}^2 \equiv \sum_i \sum_j (Y_{ij} - m - t_i - b_j)^2 \equiv \text{TSS} - \text{SST} - \text{SSB}$$

The ANOVA table is given as Table 7.

For the sample problem of five materials (treatments) and four test machines (blocks), we obtain the ANOVA table given as Table 8.

TABLE 7

Source of variation	<i>df</i>	Sum of squares	Mean square	Expected mean square	<i>F</i>
Between treatments	$v - 1$	SST	$SST/(v - 1)$	$\sigma^2 + r \sum_i \tau_i^2 / (v - 1)$	$MST/s_e^2$
Between blocks	$r - 1$	SSB	$SSB/(r - 1)$	$\sigma^2 + v \sum_j \beta_j^2 / (r - 1)$	$MSB/s_e^2$
Residual	$(v - 1)(r - 1)$	SSE	$s_e^2$	$\sigma^2$	
Total	$vr - 1$	TSS			

TABLE 8

Source of variation	<i>df</i>	Sum of squares	Mean square	Expected mean square	<i>F</i>
Materials	4	905.2	226.3	$\sigma^2 + \sum_i \tau_i^2$	16.89
Test machines	3	553.2	184.4	$\sigma^2 + 5 \sum_j \beta_j^2 / 3$	13.76
Residual	12	160.8	13.4	$\sigma^2$	
Total	19	1619.2			

Note that the residual mean square, an unbiased estimate of  $\sigma^2$ , is 13.4, considerably lower than the 47.6 of the previous analysis. Since this (coded) set of data was actually collected from a randomized block design, one might argue that the decrease in the magnitude was expected. However, the estimate of  $\sigma^2$  is based upon fewer *df* in a randomized block design than from a corresponding experiment run in a completely randomized design. Measures of the relative efficiency of a randomized block design are available (e.g., Refs. 2 and 3) which include both the reduction in the *df* as well as the magnitude of the estimate of the error variance.

In addition to the estimates of the unknown parameters which were obtained by the method of least squares, one can readily construct confidence intervals for the parameters or functions of the parameters (e.g.,  $\tau_i - \bar{\tau}_i$ ). If the experimenter were interested in testing the null hypothesis concerning the equivalent strength of the five materials, i.e.,  $H_0: \tau_1 = \dots = \tau_5 = 0$ , against the alternate hypothesis that not all  $\tau_i = 0$ , he would compare the ratio of the materials mean square to the residual mean square with the critical or tabulated value of the *F* statistic. That is,  $F = MST/s_e^2 = 226.3/13.4 = 16.89$ , which is greater than the  $100(1 - \alpha)$  percentile of  $F(4, 12)$ , for a reasonable  $\alpha$ . Thus, the observed difference among materials is greater than can be explained by chance alone, or, the differences are statistically significant. The justification for testing this hypothesis in this manner is based upon Cochran's theorem, and the rationale is illustrated in the expected mean squares column in the ANOVA table. If the null hypothesis were true (all  $\tau_i = 0$ ), the materials mean square is an unbiased estimate of  $\sigma^2$ . When the null hypothesis is false, the materials mean square is inflated by a nonnegative function of the  $\tau_i$ .

When randomized block designs were initially used in the biological sciences, hypotheses concerning block effects were rarely tested since the experimenter knew that different sections of land had different initial fertility, or that animals from the same litter were more alike than animals from different litters, etc. Thus, blocking obviously increased the precision of the experiment; if the experimenter showed any interest concerning  $\beta_j$ , it would only be in estimating their magnitudes. However, in many practical applications of these designs, blocks reflect a suspected important source of variation, and the experimenter may be genuinely interested in hypothesis testing. For example, the test machines under consideration in this example were all supplied by the same manufacturer and were periodically calibrated. Nevertheless, they are "modified" from time to time both for determining the strength of materials at elevated temperatures and for testing specimens of different geometries. Since the experimenter, on occasion, had observed poor agreement among replicate specimens from different test machines, he decided to take the precaution of treating test machines as blocks.

Upon examining the expected mean square for blocks in the ANOVA table, we observe that under the null hypothesis  $\beta_1 = \beta_2 = \beta_3 = \beta_4 = 0$ , the machine or block mean square is an unbiased estimate of  $\sigma^2$ . Thus we form the  $F$  ratio of  $MSB/s_e^2$  and refer it to the critical value of the Snedecor  $F$  distribution. For the sample problem,  $F = 184.4/13.4 = 13.76$ . If we select  $\alpha = 0.05$ , we observe that the tabular value of  $F_{0.95}(3, 12) = 3.49$ . Since the observed ratio exceeds the tabulated value (or the test statistic falls in the critical or rejection region), we reject the null hypothesis and state that the observed differences in test machines are statistically significant at the 5% level. Thus, we conclude that the estimates of  $\beta_j$  ( $-3.4, 1.2, -5.8, 8.0$ ) are not manifestations of random variation but are indicative of test machines bias.

In some experimental situations, an experimenter desires to control two types of environmental variation which impinge on the data-taking process by means of the technique of blocking. For example, he might view both operators and time periods as sources of variation which should be eliminated from residual variation. This would be reflected in an expanded statistical model, the execution of the experiment, and the resulting ANOVA. A classic illustration of a design for this situation is the Latin square design. Another type of blocking situation concerns the occasions where the natural block size is inadequate to contain each treatment under investigation. For example, suppose that run-to-run variation for a wear-testing machine is considered large, but only four test specimens can be handled in a single run, whereas we wish to compare five different materials. The ANOVA is more complicated for these incomplete block designs. However, this class of designs is also adequately covered in most of the texts on experimental design listed in the bibliography.

## MULTIFACTOR EXPERIMENTS

A common experimental problem is that of studying the effects and joint effects of several controlled variables or factors on some response variable. For example, a

chemical engineer may want to study the effects of pressure, temperature, and the concentration of an ingredient on process yield. Similarly, a metallurgist might study the effects of various alloying elements and processing variables on the corrosion resistance of an alloy. An approach to multifactor problems which has proved to be effective in many diverse fields of application is that of factorial experiments. A factorial experiment consists of jointly varying each of the factors over the levels of interest. That is, suppose the factor temperature is to be studied at three levels, the factor flow rate at two levels, and the factor type of catalyst at four levels; then a complete factorial experiment consists of the  $(3)(2)(4) = 24$  distinct treatment combinations. Any number of factors, each with any number of levels, can be handled by factorial experiments. However, since the number of runs for a single replication consists of the product of the levels for each factor, the cost may become prohibitively large. A balanced subset of all possible treatment combinations can often effectively be used to satisfy the objectives of a multifactor experiment. These balanced subsets are referred to in the literature as fractional factorial experiments or fractional replicates. Both quantitative factors, such as temperature, and qualitative factors, such as types of catalyst, can be handled in the general factorial structure. Furthermore, factorial experiments can be run in completely randomized designs, randomized block designs, incomplete block designs, etc.

Consider the case of three factors,  $A$ ,  $B$ , and  $C$ . Suppose that factor  $A$  is to be investigated at  $a$  levels, factor  $B$  at  $b$  levels, factor  $C$  at  $c$  levels, and  $r$  replications are to be run in a completely randomized design. Thus  $N = a \cdot b \cdot c \cdot r$ . A model for this experiment would be

$$Y_{ijkl} = \eta + \alpha_i + \beta_j + \gamma_k + \alpha\beta_{ij} + \alpha\gamma_{ik} + \beta\gamma_{jk} + \alpha\beta\gamma_{ijk} + \varepsilon_{ijkl} \quad (11)$$

where  $Y_{ijkl}$  is the observed response to the  $l$ th run in which factor  $A$  was held at its  $i$ th level, factor  $B$  at its  $j$ th level, and factor  $C$  at its  $k$ th level;  $\eta$  is the true mean for the specified conditions;  $\alpha_i$  is the true effect of the  $i$ th level of factor  $A$ , measured from  $\eta$ ;  $\alpha\beta_{ij}$  is the true interaction effect of the  $i$ th level of factor  $A$  and the  $j$ th level of the factor  $B$ , that is, if  $\eta_{ij}$  represents the true mean for runs with factor  $A$  held at its  $i$ th level and factor  $B$  at its  $j$ th level,  $\alpha\beta_{ij} = \eta_{ij} - (\eta + \alpha_i + \beta_j)$ , thus it is the nonadditive or synergistic effect of these two factors;  $\alpha\beta\gamma_{ijk}$  is the true interaction effect of the  $i$ th level of factor  $A$  and the  $j$ th level of factor  $B$  and the  $k$ th level of factor  $C$ ; this three-factor interaction, if it exists, represents a nonadditive joint effect of the three factors beyond the single or main effects of the factors and the two-factor interactions;  $\varepsilon_{ijkl}$  is the true error associated with  $Y_{ijkl}$ . In the above model, the effects sum to zero over their respective indices, and  $\varepsilon \sim NID(0, \sigma^2)$ .

The corresponding sample model would be

$$Y_{ijkl} = m + a_i + b_j + c_k + ab_{ij} + ac_{ik} + bc_{jk} + abc_{ijk} + e_{ijkl} \quad (12)$$

with side conditions imposed corresponding to the population model. The least squares estimators may be easily obtained. For example,  $a_i = \bar{Y}_{i...} - m$ ,  $ab_{ij} = \bar{Y}_{ij..} - m - a_i - b_j$ , etc.

The main advantages of factorial experiments are (1) efficiency, (2) detection and evaluation of interactions, and (3) a broader base for inductive inference. The efficiency arises in that all  $N$  observations are used to evaluate each of the effects. For

example, if factor  $A$  were evaluated at two levels, then  $N/2$  runs would be taken at each of these levels, while the combinations of the remaining factors would be equally represented, or balanced, within each set. It is as if the entire experiment were devoted solely to factor  $A$ . Furthermore, suppose that factor  $B$  were run at three levels. Then  $N/3$  observations would be obtained at each level, with the remaining factors balanced out. This characteristic of factorial experiments has been referred to as "hidden replication." The second feature, concerning interaction, is usually viewed by experimenters as the most important characteristic of factorial experiments. For example, a unit increase of element  $A$  might result in an average increase in strength of 2000 psi, and a unit increase in element  $B$  might result in an average increase in strength of 3000 psi. However, when both elements are simultaneously increased one unit, the average strength might be greater than or less than 5000 psi. There are many other ways of illustrating the phenomenon of interaction. The final advantage of factorial experiments is unfortunately overlooked by many experimenters, even though they readily recognize the other two listed advantages. If we view the set of treatment combinations or experimental conditions for a three-factor experiment as a three-dimensional factor space, we note that the gridlike spacing of treatment combinations spans the experimental region of interest. Thus, the conclusions drawn concerning these factors will tend to be more valid than an equal number of observations collected over a restricted part of the region of interest, as usually obtained through the vary-one-factor-at-a-time strategy.

The partitioning of the total variation through the ANOVA is as follows:

$$\text{TSS} = \sum_{i=1}^a \sum_{j=1}^b \sum_{k=1}^c \sum_{l=1}^r (Y_{ijkl} - \bar{Y})^2 \equiv \sum Y_{ijkl}^2 - \frac{1}{N} (\sum Y_{ijkl})^2 \equiv \sum Y_{ijkl}^2 - Nm^2$$

Let  $A_i$ ,  $B_j$ ,  $C_k$  represent the total of the responses to runs with factor  $A$  held at its  $i$ th level, factor  $B$  at its  $j$ th level, and factor  $C$  at its  $k$ th level, respectively. Similarly, let  $AB_{ij}$  represent the total of the responses to runs with both factor  $A$  at its  $i$ th level and factor  $B$  at its  $j$ th level, etc.

$$\text{SSA} = bcr \sum_i a_i^2 \equiv bcr \sum_i (\bar{Y}_{i..} - m)^2 \equiv \frac{1}{bcr} \sum_i A_i^2 - Nm^2$$

$$\text{SSB} = acr \sum_j b_j^2 \equiv acr \sum_j (\bar{Y}_{.j.} - m)^2 \equiv \frac{1}{acr} \sum_j B_j^2 - Nm^2$$

$$\text{SSC} = abr \sum_k c_k^2 \equiv abr \sum_k (\bar{Y}_{..k.} - m)^2 \equiv \frac{1}{abr} \sum_k C_k^2 - Nm^2$$

$$\begin{aligned} \text{SSAB} &= cr \sum_i \sum_j ab_{ij}^2 \equiv cr \sum_i \sum_j (\bar{Y}_{ij..} - \bar{Y}_{i..} - \bar{Y}_{.j.} + m)^2 \\ &\equiv \frac{1}{cr} \sum_i \sum_j AB_{ij}^2 - Nm^2 - \text{SSA} - \text{SSB} \end{aligned}$$

$$\begin{aligned} \text{SSAC} &= br \sum_i \sum_k ac_{ik}^2 \equiv br \sum_i \sum_k (\bar{Y}_{i.k.} - \bar{Y}_{i..} - \bar{Y}_{..k.} + m)^2 \\ &\equiv \frac{1}{br} \sum_i \sum_k AC_{ik}^2 - Nm^2 - \text{SSA} - \text{SSC} \end{aligned}$$

$$\begin{aligned}
 \text{SSBC} &= ar \sum_j \sum_k bc_{jk}^2 \equiv ar \sum_j \sum_k (\bar{Y}_{jk.} - \bar{Y}_{j..} - \bar{Y}_{..k.} + m)^2 \\
 &\equiv \frac{1}{ar} \sum_j \sum_k BC_{jk}^2 - Nm^2 - \text{SSA} - \text{SSC} \\
 \text{SSABC} &= r \sum_i \sum_j \sum_k abc_{ijk}^2 = r \sum_i \sum_j \sum_k (\bar{Y}_{ijk.} - \bar{Y}_{ij..} - \bar{Y}_{i..k.} - \bar{Y}_{..jk.} \\
 &\quad + \bar{Y}_{i..} + \bar{Y}_{j..} + \bar{Y}_{..k.} - m)^2 \\
 &\equiv \frac{1}{r} \sum_i \sum_j \sum_k ABC_{ijk}^2 - Nm^2 - \text{SSA} - \text{SSB} - \text{SSC} \\
 &\quad - \text{SSAB} - \text{SSAC} - \text{SSBC} \\
 \text{SSE} &= \Sigma e_{ijkl}^2 \equiv \Sigma (Y_{ijkl} - \bar{Y}_{ijkl})^2 \equiv \text{TSS} - \text{SSA} - \text{SSB} - \text{SSC} - \text{SSAB} \\
 &\quad - \text{SSAC} - \text{SSBC} - \text{SSABC}
 \end{aligned}$$

In the ANOVA table (Table 9), the mean squares—the sum of squares scaled by the corresponding  $df$ —are designated as MS.

TABLE 9

Source of variation	$df$	Sum of squares	Mean square	Expected mean square
$A$	$a - 1$	SSA	MSA	$\sigma^2 + bcr \sum_i \alpha_i^2 / (a - 1)$
$B$	$b - 1$	SSB	MSB	$\sigma^2 + acr \sum_j \beta_j^2 / (b - 1)$
$C$	$c - 1$	SSC	MSC	$\sigma^2 + abr \sum_k \gamma_k^2 / (c - 1)$
$AB$	$(a - 1)(b - 1)$	SSAB	MSAB	$\sigma^2 + cr \sum_i \sum_j \alpha \beta_{ij}^2 / (a - 1)(b - 1)$
$AC$	$(a - 1)(c - 1)$	SSAC	MSAC	$\sigma^2 + br \sum_i \sum_k \alpha \gamma_{ik}^2 / (a - 1)(c - 1)$
$BC$	$(b - 1)(c - 1)$	SSBC	MSBC	$\sigma^2 + ar \sum_j \sum_k \beta \gamma_{jk}^2 / (b - 1)(c - 1)$
$ABC$	$(a - 1)(b - 1)(c - 1)$	SSABC	MSABC	$\sigma^2 + r \sum_i \sum_j \sum_k \alpha \beta \gamma_{ijk}^2 / (a - 1)(b - 1)(c - 1)$
Error	$\frac{abc(r - 1)}{abcr - 1}$	SSE	$se^2$	$\sigma_e^2$
Total		TSS		

From the expected mean squares, it is clear how various hypotheses may be tested by means of the  $F$  distribution. When interactions turn out to be statistically significant, multiway tables of averages and graphs of these tables are useful in interpreting the nature of the interaction. For example, suppose that the  $AB$  interaction were statistically significant. Then, a table of the following structure is recommended:

$A$	$B$			$b$
	1	2	...	
1	$\bar{Y}_{11}$	$\bar{Y}_{12}$	...	$\bar{Y}_{1b}$
2	$\bar{Y}_{21}$	$\bar{Y}_{22}$	...	$\bar{Y}_{2b}$
.	.	.		.
.	.	.		.
$a$	$\bar{Y}_{a1}$	$\bar{Y}_{a2}$	...	$\bar{Y}_{ab}$
	$\bar{Y}_{..1}$	$\bar{Y}_{..2}$	...	$\bar{Y}_{..b}$
				$\bar{Y}_{..}$

If one graphs the body of the above two-way table with, say,  $A$  as the abscissa, the resulting  $b$  curves will illustrate the nature of the interaction. A relatively simple interaction is graphed in Fig. 1. The  $F$  test which was statistically significant informed us that the degree of nonparallelism of the curves is greater than can be explained by our measure of experimental error. One should keep in mind that the phenomenon of interaction is not an absolute, in the sense that it is dependent upon metrics and spacings used in the experiment. In fact, one aim of model reparameterization through transformations is to simplify the model, that is, to obtain an effective reparameterized model containing only low-order effects.

When all the factors are quantitative, setup costs are not too large, turnaround time for obtaining responses is not too long, and the experimental error is not too

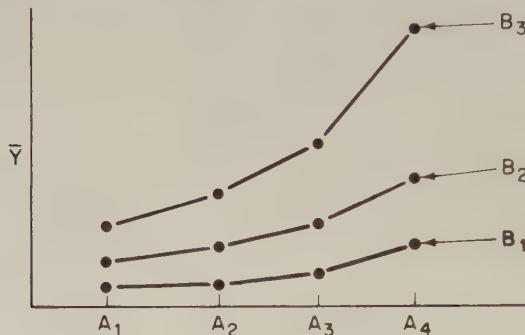


Figure 1.

large, an effective sequential procedure for experimentally determining optima of multifactor experiments is generally referred to as response surface methodology (RSM). The development of this work over the past two decades has been extensively documented in the literature by G. E. P. Box and his colleagues. See, for example, Refs. 4 and 5.

## MISSING PLOT TECHNIQUES

The concept of balance or orthogonality is deeply ingrained in the development of experimental designs because of the resulting characteristics of efficiency and power, and the ease of analysis and interpretation of such designs. From the point of view of the ANOVA, the relatively simple algebraic partitioning of the sums of squares into ascribable sources in an unambiguous manner for randomized block designs, Latin square designs, complete and fractional factorial experiments, etc., is based upon the inherent balance, symmetry, or orthogonality of the designs. Nevertheless, in the real world of experimentation, things occasionally go wrong, destroying the orthogonality of the design.

Of course, the resulting data could be handled through a general linear hypothesis,

or least squares, approach, but experimenters already familiar with the ANOVA technique for balanced designs prefer solutions within this framework which require only minor modifications, even if the resulting analysis is only a good approximation rather than an exact solution. In 1933, Frank Yates considered modifications to the ANOVA that are required with nonorthogonal data. Most of the texts in the bibliography discuss this topic under the title of Missing Plot Techniques. A review paper with an excellent bibliography is by Hoyle [6].

## FIXED, RANDOM, AND MIXED MODELS

The models corresponding to the designs described thus far contained unknown constants or parameters representing the effects of treatments or blocks. Only the  $\varepsilon$ , the errors, were random variables. For example, in the study of  $v$  materials, these materials were purposely selected by the experimenter because of their technical interest, and the inferences arising from the experiment are restricted to these specific  $v$  materials. Similarly, if the factor holding time were studied at 1, 2, and 4 hr, these three levels were purposely selected by the experimenter to reflect his range and spacing of interest. When the levels of a factor are purposely selected and the corresponding symbols in the model represent constants, this is known as a fixed effect. When all of the factors in an experiment have this property (only the  $\varepsilon$  are random variables), the model is known as a fixed-effects model, or, Model I.

There are many experimental situations in which the above characterization is inadequate. For example, the process engineer assessing the control of his process not only is concerned with the within-batch variation but also the batch-to-batch variation. His interest does not center on comparing  $v$  specific batches, but, from a random sample of  $r$  batches, he wishes to estimate the component of variance due to batches. If he were to repeat the study, his independent randomization would lead to a different set of  $v$  batches. The symbol in the model representing the  $i$ th batch effect is a random variable. Thus, for this illustrative example, suppose that he observes  $r$  experimental units on each of  $v$  randomly selected batches. We can write the model as

$$Y_{ij} = \eta + \beta_i + \varepsilon_{ij}, \quad i = 1, \dots, v, \quad j = 1, \dots, r \quad (13)$$

The  $\beta_i$  and the  $\varepsilon_{ij}$  are both random variables, and the differences among the  $Y_{ij}$  are due to both random variables impinging on the constant  $\eta$ . In addition to the assumption that  $\varepsilon_{ij} \sim \text{NID}(0, \sigma_\varepsilon^2)$ , we further assume that  $\beta_i \sim \text{NID}(0, \sigma_\beta^2)$ , where  $\sigma_\varepsilon^2$  represents the variance among responses from the same batch and  $\sigma_\beta^2$  represents the variance among the population of batch means. The model in (13) is called either a random-effects model, components of variance model, or Model II.

The arithmetic of the ANOVA for Model II is the same as for Model I. However, one's interest centers on the estimation or testing hypotheses concerning components of variance, and this is reflected in the expected mean squares (Table 10).

TABLE 10

Source of variation	df	Sum of squares	Mean square	Expected mean square
Between batches	$v - 1$	SSB	MSB	$\sigma_e^2 + r\sigma_\beta^2$
Within batches	$v(r - 1)$	SSE	$s_e^2$	$\sigma_e^2$
Total	$vr - 1$	TSS		

This suggests that we can test the null hypothesis  $\sigma_\beta^2 = 0$  against the alternative  $\sigma_\beta^2 > 0$  by comparing the ratio of  $\text{MSB}/s_e^2$  with  $F_{1-\alpha}[v-1, v(r-1)]$ . Although the arithmetic of the ANOVA is the same as for the initial Model I materials problem, one not only is testing different hypotheses through the  $F$  test, but even the formation of the  $F$  ratios is often different. The relevant guide is contained in the expected mean squares, and fortunately, many books (e.g., Refs. 7-9) contain rules of thumb for obtaining the expected mean squares, saving a great deal of messy algebra in complex models. In addition, the references should be consulted for constructing interval estimates for  $\eta$  and the components of variance. Point estimates of the components of variance may be easily obtained by equating the numerically calculated mean squares to their expectations, and solving the simultaneous linear equations.

Strictly speaking, the literature concerning random effects deals primarily with the abstraction that the population of levels is infinite. In practice, this means that the number of levels in the population is large relative to those randomly chosen for the experiment. This is a part of the classical problem of sampling without replacement and assessing the modification due to finite population correction factors. Bennett and Franklin [7], Brownlee [10], and Johnson and Leone [8], among others, deal with this problem concerning the ANOVA.

In some experimental problems, one or more of the factors may be fixed and one or more of the factors may be random. These are called mixed models, or Model III. However, the rules of thumb for obtaining expected mean squares cited before adequately handle the case of mixed models.

## HIERARCHICAL OR NESTED MODELS

Randomized block designs, Latin square designs, and factorial experiments are illustrations of cross-classified experiments. That is, in a randomized block design the designation treatment  $i$  refers to a specific treatment and carries the same meaning in whatever block the treatment appears. Similarly, in a factorial experiment, the  $i$ th level of factor  $A$  refers to a particular identifiable stimulus, regardless of the combinations of other factors.

In contrast to cross-classified experiments, there exists another experimental struc-

ture referred to as nested or hierarchical. Consider the two factors, shifts and operators, and suppose that five operators are selected on each of three shifts. Note that this involves 15 different operators, and operator 1 on shift 1 does not correspond to the operator 1 on shift 2. In this structure, the factor operator is nested within shifts.

Consider the following typical nested structure. A shipment of a large number of drums of granulated material is received from a supplier and it is to be checked to determine whether it conforms to the purchase specifications. The material must be examined on a sampling basis, so  $d$  drums are randomly selected for evaluation. From each of these  $d$  drums,  $s$  samples are randomly drawn, and  $a$  analyses are performed on each sample. The magnitudes of  $d$ ,  $s$ , and  $a$  are dependent upon testing costs, the magnitudes of the components of variance, and the degree of precision required.

The model for this problem may be expressed as

$$Y_{ijk} = \eta + \delta_i + \theta_{j|i} + \varepsilon_{k|ij} \quad (14)$$

where  $Y_{ijk}$  is the measured response on the  $k$ th analysis on the  $j$ th sample from the  $i$ th drum;  $\eta$  is the true shipment mean;  $\delta_i$  is the true random effect of the  $i$ th drum,

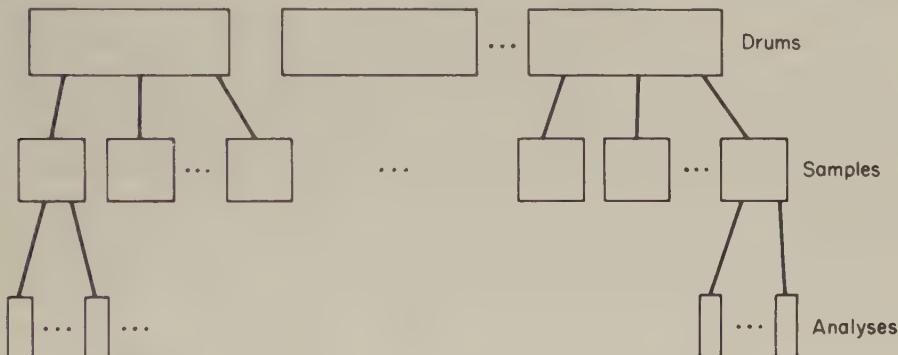


Figure 2.

assumed  $NID(0, \sigma_\delta^2)$ ;  $\theta_{j|i}$  is the true random effect of the  $j$ th sample in the  $i$ th drum, assumed  $NID(0, \sigma_\theta^2)$ ; and  $\varepsilon_{k|ij}$  is the true random effect of the  $k$ th analysis on the  $j$ th sample in the  $i$ th drum, assumed  $NID(0, \sigma_\varepsilon^2)$ . This problem has been defined as a Model II nested sampling scheme. The designation sample 1 from drum 1 has no correspondence to sample 1 of drum 2, but the sample numbering or labels are merely convenient accounting procedures. The variance component  $\sigma_\delta^2$  represents drum-to-drum variation,  $\sigma_\theta^2$  represents within-drum variation, and  $\sigma_\varepsilon^2$  represents the variance among analyses on a given sample. Graphically, the sampling plan may be represented as shown in Fig. 2.

The sums of squares calculations for the ANOVA are as follows:

$$TSS = \sum_{i=1}^d \sum_{j=1}^s \sum_{k=1}^a (Y_{ijk} - m)^2 \equiv \sum Y_{ijk}^2 - \frac{1}{N} (\sum Y_{ijk})^2 \equiv \sum Y_{ijk}^2 - Nm^2$$

Let  $D_i$  and  $S_{ij}$  represent the total of the responses from the  $i$ th drum and the  $j$ th sample from the  $i$ th drum, respectively.

$$\begin{aligned} \text{SSD} &= sa \sum_i (\bar{Y}_{i..} - m)^2 \equiv \frac{1}{sa} \sum_i D_i^2 - Nm^2 \\ \text{SSS} &= a \sum_i \sum_j (\bar{Y}_{ij.} - \bar{Y}_{i..})^2 = \frac{1}{a} \sum_i \sum_j S_{ij}^2 = \frac{1}{sa} \sum_i D_i^2 \\ \text{SSA} &= \sum_i \sum_j \sum_k (Y_{ijk} - \bar{Y}_{ij.})^2 \equiv \sum_i \sum_j \sum_k Y_{ijk}^2 - \frac{1}{a} \sum_i \sum_j S_{ij}^2 \end{aligned}$$

Note that  $\text{SSD} + \text{SSS} + \text{SSA} = \text{TSS}$ . The ANOVA table is given as Table 11.

TABLE 11

Source of variation	df	Sum of squares	Mean square	Expected mean square
Drums	$d - 1$	SSD	MSD	$\sigma_e^2 + a\sigma_\theta^2 + as\sigma_\delta^2$
Samples in drums	$d(s - 1)$	SSS	MSS	$\sigma_e^2 + a\sigma_\theta^2$
Analyses in samples	$ds(a - 1)$	SSA	MSA	$\sigma_\theta^2$
Total	$dsa - 1$	TSS		

The  $d - 1$  degrees of freedom for drums is self-evident. Within each drum  $s$  samples are taken. Thus each of the  $d$  drums offers an estimate of  $\sigma_\theta^2$  based upon  $s - 1$  df, and the pooled estimate has  $d(s - 1)$  df. Finally,  $a$  analyses are made, on each of the  $ds$  samples leading to an estimate of  $\sigma_\theta^2$  based upon  $ds(a - 1)$  df. For obtaining estimates of the components of variance, one could equate the observed mean squares with their expectations and solve the resulting simultaneous linear equations. For testing hypotheses, observe that  $H_0: \sigma_\theta^2 = 0$  would be tested by forming  $F = \text{MSS}/\text{MSA}$ , similar to that of the previous section, but  $H_0: \sigma_\delta^2 = 0$  would be tested by constructing  $F = \text{MSD}/\text{MSS}$ .

If a computer program were available for a cross-classification factorial experiment, but none was available for a nested structure, one could obtain the degrees of freedom and sums of squares for the above ANOVA through a simple pooling procedure. That is, in a cross-classification computer program, one would obtain the output of Table 12.

TABLE 12

Source of variation	df	Sum of squares
Drums	$d - 1$	SSD
Samples	$s - 1$	SSS
Analyses	$a - 1$	SSA
Drum $\times$ samples interaction	$(d - 1)(s - 1)$	SSDS
Drum $\times$ analyses interaction	$(d - 1)(a - 1)$	SSDA
Sample $\times$ analyses interaction	$(s - 1)(a - 1)$	SSSA
Drum $\times$ samples $\times$ analyses interaction	$(d - 1)(s - 1)(a - 1)$	SSDSA
Total	$dsa - 1$	TSS

For a nested structure, the first line in the ANOVA (drums) is correct. However, to obtain samples in drums, combine or pool the two lines labeled samples and drum  $\times$  samples interaction. Finally, to obtain the *df* and sum of squares for analyses in samples, pool the line in the above ANOVA labeled analyses with all interactions involving analyses. Of course, it is more convenient to have a special computer program for nested structures, but not many of the existing packages have one.

The sample problem represented a balanced completely hierarchical Model II, a not uncommon application. However, partially hierarchical models arise where some factors are nested and others are crossed. Further, both fixed and random factors may occur, and the sampling rates may not be uniform. See the references in the bibliography for handling the ANOVA under more complex conditions.

## VIOLATION OF UNDERLYING ASSUMPTIONS

Regardless of the particular structure of the underlying statistical model for the ANOVA, the strict validity of the resulting inferences are based upon the assumptions that random errors are normally and independently distributed, with common variance. Experienced data analysts examine the residuals, both through formal and graphical procedures, to obtain evidence of violation of these standard assumptions. Most experimenters who use ANOVA procedures would not mind if the resulting violations caused a change in the nominal error rate of, say, 5% to increase to 6%, but a gross degradation of the efficiency and power of ANOVA procedures cannot be tolerated.

Several studies have been made on the effects of violating the standard assumptions of the ANOVA in various specific ways. Some of these studies are described and referenced in the accompanying bibliography. In general, nonnormality has little effect on inferences concerning fixed effects, but inferences concerning random effects are more sensitive to nonnormality. Nonconstancy of variance, or heteroscedasticity, appears to have a small effect upon balanced or equally replicated Model I designs, whereas it becomes much more serious under unequal replication. Data analysts employ transformations of the data, e.g., logarithm, square root, inverse sine, to stabilize the variance. Weighting the observations with poorly estimated sample variances can be self-defeating. The physical act of randomization in assigning the treatments to the experimental units and in the order of execution of the experiment helps to achieve the assumption of uncorrelated random errors, for correlation of the errors can have a serious effect upon the standard inferences based upon assumed stochastic independence.

Although one should be pleased that certain aspects of the ANOVA technique are relatively insensitive to the violation of certain assumptions from the point of view of summarizing the data against a model, the examination of residuals for exposing unanticipated characteristics of the data is still a very important part of the data analysis and should be performed on every problem as standard analytical procedure.

## FURTHER SUBDIVISION OF SUMS OF SQUARES

The standard hypotheses tested in the ANOVA are very broad and are sometimes referred to as omnibus tests. That is, the null hypothesis tested for materials was that all of the  $v$  materials were of equivalent strength. A large  $F$  ratio, resulting in the rejection of the null hypothesis, merely leads the experimenter to the conclusion that not all of the  $v$  materials are of equivalent strength. This is hardly an appropriate stopping place for a sensible experimenter. He immediately wants to ascertain in what way the materials differ.

Two general situations arise which must be distinguished because of their differing probabilistic frames of reference. The first involves a set of comparisons among the materials which were technically interesting *before* the experimenter obtained the data, and the second involves either comparisons among the materials which were suggested by the data, or, making comparisons between all possible pairs of materials averages.

The first situation referred to above is described in the texts in the bibliography as the method of orthogonal contrasts, or, single degree of freedom comparisons. For example, suppose that these materials were obtained from two different vendors. Then, before the experimenter obtained the data, a technically sensible comparison would be between the materials supplied by one vendor and those supplied by the other. The sum of squares for this single degree of freedom contrast is a part of the materials sum of squares. In fact, the materials sum of squares, based upon  $v - 1$   $df$ , may be decomposed into  $v - 1$  separate orthogonal contrasts, if the experimenter should so desire. For quantitative factors, such as time, temperature, concentration, etc., the individual contrasts may sensibly correspond to the shape of the response function. The most common procedure is to use the coefficients for orthogonal polynomials, which are widely tabulated for equally spaced levels. Thus an experimenter takes data at three different temperatures because he expects or suspects a curved response. In this simple case he can decompose the temperature sum of squares, based upon two  $df$ , into a portion ascribable to a linear trend and a portion reflecting the quadratic trend. Formal tests of significance for these contrasts are embedded in the ANOVA setup.

The sum of squares for interaction effects can be similarly decomposed, as described in the texts listed in the bibliography. A total decomposition of all effects in an ANOVA into single degrees of freedom results in a multiple linear regression analysis.

The second type of detailed analysis, based upon either comparisons suggested by the data or all possible comparisons among the levels of qualitative factors, is dealt with under the topic of multiple comparisons. The names of Duncan, Dunnett, Scheffé, and Tukey are primarily involved with this problem. The reader is urged to pay careful attention to the distinction between comparisonwise and experimentwise error rates. Several of the texts listed in the bibliography describe multiple comparison procedures, but an outstanding one is by Miller [11].

## DETERMINATION OF NUMBER OF REPLICATIONS

The general problems of estimation of parameters and testing hypotheses can be handled with a minimal number of observations. However, 99% confidence intervals such as  $17 \pm 100$  are not very informative. Three general ways of decreasing the width of confidence intervals are (1) decreasing the confidence coefficient, (2) increasing the number of replications, and (3) decreasing the magnitude of experimental error through local control or blocking. In practical experiments, one must balance the gain in information (increased precision) against increased experimental costs. Several of the texts in the bibliography discuss the problem of the determination of sample size.

In the context of hypothesis testing, one attempts to design the experiment so that statistical significance becomes synonymous with practical significance. For example, in comparing two treatments, an experimenter may decide that a true difference between the treatment means of, say, 5 units would be important to detect. Based upon an estimate of experimental error, he can readily determine the number of replications required for rejecting the null hypothesis if the true difference in treatment means is indeed as much as 5 units. Tables and graphs are available for determining the required sample size for problems of this type. However, in comparing  $v$  treatments, the problem is more complex. Although the power function of the ANOVA test was tabulated by P. C. Tang in 1938, and several tables and charts have been published since then, it is difficult for the experimenter to specify differences of practical importance among the  $v$  means in terms of  $\sum_{i=1}^v \tau_i^2$ . Kastenbaum *et al.* [12] provided tables based upon the standardized range of the treatment means rather than the noncentrality parameter, and this has much more intuitive appeal to most experimenters. They also published a paper on randomized block designs [13].

## CONCLUDING REMARKS

An attempt has been made to expose the reader to an overview of the rationale of the ANOVA related to common experimental designs. It is hoped that the exposition will lead the reader to investigate the in-depth coverage to be found in the texts listed in the bibliography. One legitimate criticism of these texts, with the possible exception of Peng's book [14], is that they do not contain a computer-oriented approach to the subject. Based upon recent work appearing in the literature, the formation of a section on Statistical Computing of the American Statistical Association in 1971, and a few known texts still in draft form, this deficiency will be remedied within the next few years. Nevertheless, the experimentally oriented reader will continue to find the material in the bibliography of value.

Most of the computer manufacturers have ANOVA programs available. For example, user groups such as IBM SHARE, Control Data's VIM, and Digital Equipment's DECUS have several such programs. Commercially available libraries for statistical computing also contain ANOVA programs. In addition, the BMD Biomedical Computer Programs available from the University of California at Los Angeles are in wide usage.

During the next decade, one may expect that effective statistical computing systems will experience rapid growth. These systems will not only incorporate current research on random and mixed models, nonparametric statistics, and multivariate analysis of variance (MANOVA), but will readily incorporate the plotting of residuals in various ways as well as other graphical procedures.

## REFERENCES

1. R. A. Fisher, *The Design of Experiments*, 7th ed., Hafner, New York, 1960.
2. O. Kempthorne, *The Design and Analysis of Experiments*, Wiley, New York, 1952.
3. W. G. Cochran and G. M. Cox, *Experimental Designs*, 2nd ed., Wiley, New York, 1957.
4. R. H. Myers, *Response Surface Methodology*, Allyn and Bacon, Boston, 1971.
5. O. L. Davies, *The Design and Analysis of Industrial Experiments*, 2nd ed., Hafner, New York, 1963.
6. M. H. Hoyle, Spoilt data—An introduction and bibliography, *J. Roy. Statist. Soc., A*, **1134**, Pt. 3 (1971).
7. C. A. Bennett and N. L. Franklin, *Statistical Analysis in Chemistry and the Chemical Industry*, Wiley, New York, 1954.
8. N. L. Johnson and F. C. Leone, *Statistics and Experimental Design in the Physical and Engineering Sciences*, Vol. 2, Wiley, New York, 1964.
9. C. R. Hicks, *Fundamental Concepts in the Design of Experiments*, Holt, Rinehart and Winston, New York, 1964.
10. K. A. Brownlee, *Statistical Theory and Methodology in Science and Engineering*, 2nd ed., Wiley, New York, 1965.
11. R. G. Miller, *Simultaneous Statistical Inference*, McGraw-Hill, New York, 1966.
12. M. A. Kastenbaum et al., Sample size requirements: One-way analysis of variance, *Biometrika*, **57**, 421–430 (1970).
13. M. A. Kastenbaum et al., Sample size requirements: Randomized block designs, *Biometrika*, **57**, 573–577 (1970).
14. K. C. Peng, *The Design and Analysis of Scientific Experiments*, Addison-Wesley, Reading, Mass., 1967.

## BIBLIOGRAPHY

- Anderson, R. L., and T. A. Bancroft, *Statistical Theory in Research*, McGraw-Hill, New York, 1952.  
Bennett, C. A., and N. L. Franklin, *Statistical Analysis in Chemistry and the Chemical Industry*, Wiley, New York, 1954.  
Brownlee, K. A., *Statistical Theory and Methodology in Science and Engineering*, 2nd ed., Wiley, New York, 1965.  
Cochran, W. G., and G. M. Cox, *Experimental Designs*, 2nd ed., Wiley, New York, 1957.  
Cox, D. R., *Planning for Experiments*, Wiley, New York, 1958.  
Davies, O. L., *The Design and Analysis of Industrial Experiments*, 2nd ed., Hafner, New York, 1963.

- Draper, N., and H. Smith, *Applied Regression Analysis*, Wiley, New York, 1966.
- Duncan, A. J., *Quality Control and Industrial Statistics*, 3rd ed., Irwin, Homewood, Ill., 1965.
- Federer, W. T., *Experimental Design, Theory and Application*, Macmillan, New York, 1955.
- Fisher, R. A., *The Design of Experiments*, 7th ed., Hafner, New York, 1960.
- Graybill, F., *An Introduction to Linear Statistical Models*, McGraw-Hill, New York, 1961.
- Hald, A., *Statistical Theory with Engineering Applications*, Wiley, New York, 1952.
- Herzberg, A. M., and D. R. Cox, Recent work on the design of experiments: A bibliography and a review, *J. Roy. Statist. Soc., A* 132, Pt. 1, 29–67 (1969).
- Hicks, C. R., *Fundamental Concepts in the Design of Experiments*, Holt, Rinehart and Winston, New York, 1964.
- Huittson, A., *The Analysis of Variance*, Hafner, New York, 1966.
- Johnson, N. L., and F. C. Leone, *Statistics and Experimental Design in the Physical and Engineering Sciences*, Vol. 2, Wiley, New York, 1964.
- Kastenbaum, M. A., et al., Sample size requirements: One-way analysis of variance, *Biometrika*, 57, 421–430 (1970).
- Kastenbaum, M. A., et al., Sample size requirements: Randomized block designs, *Biometrika*, 57, 573–577 (1970).
- Kempthorne, O., *The Design and Analysis of Experiments*, Wiley, New York, 1952.
- Mandel, J., *The Statistical Analysis of Experimental Data*, Wiley-Interscience, New York, 1964.
- Mendenhall, W., *Introduction to Linear Models and the Design and Analysis of Experiments*, Wadsworth, Belmont, Calif., 1968.
- Miller, I., and J. E. Freund, *Probability and Statistics for Engineers*, Prentice-Hall, Englewood Cliffs, N.J., 1965.
- Miller, R. G., *Simultaneous Statistical Inference*, McGraw-Hill, New York, 1966.
- Myers, R. H., *Response Surface Methodology*, Allyn and Bacon, Boston, 1971.
- Natrella, M., *Experimental Statistics*, NBS Handbook 91, 1963.
- Peng, K. C., *The Design and Analysis of Scientific Experiments*, Addison-Wesley, Reading, Mass., 1967.
- Scheffé, H., *The Analysis of Variance*, Wiley, New York, 1959.
- Searle, S. R., *Linear Models*, Wiley, New York, 1971.
- Walsh, J. E., *Handbook on Nonparametric Statistics III: Analysis of Variance*, Van Nostrand, Princeton, N.J., 1968.
- Wine, R. L., *Statistics for Scientists and Engineers*, 2nd ed., Prentice-Hall, Englewood Cliffs, N.J., 1967.

*H. Ginsburg*

## AND-OR-NOT-NAND-NOR LOGIC

The present developments in logic and logic circuitry trace back at least as far as Aristotle who is credited with having made the statement: "Everything other than *A*

is not  $A$ ." While today this statement may seem to be trivial, it is very important to the present day mathematical developments which have given us such items as the digital computer and various complex digital control systems.

The use of logic considers a well-defined set of things where everything under consideration is categorized into two sets or collections; those things included in a particular set or collection named  $A$  having a particular attribute or attributes, and those things not included in the set  $A$  comprising a second set,  $A'$  or verbally not  $A$ . A set in the present discussion will be considered as any collection of elements or entities such that given any element  $x$  out of all elements under consideration, it is possible to determine if  $x$  is a member of  $A$  or a member of  $A'$ . Everything under consideration is categorized as either being  $A$  or  $A'$ , implying that  $x$  must be a member of one of the two sets but not both.

The collection of all items or elements to be categorized or considered is termed the *logical universe* or simply the *universe*. Thus the combined collection of all elements contained in  $A$  and all of the elements contained in  $A'$  makes up the logical universe. Neither  $A$  nor  $A'$  can contain any items which are common to both  $A$  and  $A'$ . The collection or set of elements each of which is contained in at least one of the two particular sets  $A$  and  $B$  is in itself a set and is said to be the *union* of  $A$  and  $B$  and written as  $A \cup B$ . Thus if 1 is used to identify the set of all elements in the universe, the set formed by the union of  $A$  and  $A'$  is 1,  $A \cup A' = 1$ .

The set of all elements, each of which is common to both of the two sets  $A$  and  $B$ , is said to be the *intersection* of  $A$  and  $B$  and written as  $A \cap B$ . Thus if 0 is used to identify the set consisting of no elements, the set formed by the intersection of  $A$  and  $A'$  is 0,  $A \cap A' = 0$ . For example, let the universe be the set of all people. Let  $A$  be the set of all males. Obviously from the definition of the universe and of  $A$ ,  $A'$  is the set of all people who are not males, i.e.,  $A'$  is the set of all females. The set of all people which in this case is the universe has been categorized into males and females.

Now anyone who is established as being a member of the set or class of things called people must either be a male or a female. The operation of union " $\cup$ " is correspondingly called an OR operation. If  $x$  is a member of the set of all people,  $x$  must be a member of  $A$  or a member of  $A'$ . Likewise the operation " $\cap$ " is termed an AND operation. In this case those things which are members of  $A$  and  $A'$  are nonexistent,  $A \cap A' = 0$ .

A universe may be subdivided into as many categories as desired. For example, in addition to sex, people can be categorized according to hair color. As a particular example, consider blonds. There are people who are blonds,  $B$ , and there are people who are not blonds,  $B'$ . With two categories, there are four possible sets in which all people must be included; blond males, males who are not blond, blond females, and females who are not blond. Thus to be a member of any one of the four sets, a person must satisfy two conditions. The universe is categorized into people who are blond and male,  $A \cap B$ ; people who are not blond and male,  $A \cap B'$ ; people who are blond and female,  $A' \cap B$ ; and people who are not blond and female,  $A' \cap B'$ . Additional material on the OR and AND operations will now be discussed.

The two operations,  $\cup$  and  $\cap$ , can be used with various sets in order to form *logical expressions* or simply *expressions*. The result of either operation with two sets

is itself a set, i.e.,  $(A \cup B)$  and  $(C \cap D)$  are each sets which can in turn be subjected to union or intersection, etc., e.g.,

$$(A \cup B) \cup (C \cap D) \quad \text{or} \quad (A \cup B) \cap (C \cap D)$$

Using the two operations, it is possible to establish identity expressions of sets which can be used to modify other expressions without changing their meaning. Below are listed examples of some possible identity expressions:

$$\begin{aligned}(A \cup B) \cup C &= A \cup (B \cup C) = A \cup B \cup C \\(A \cap B) \cap C &= A \cap (B \cap C) = A \cap B \cap C \\(A \cup B) \cap C &= (A \cap C) \cup (B \cap C) \\C \cap (A \cup B) &= (C \cap A) \cup (C \cap B)\end{aligned}$$

Using these identity expressions, it is now possible to rewrite the initial example expressions in alternate forms.

$$\begin{aligned}(A \cup B) \cup (C \cap D) &= A \cup (B \cup (C \cap D)) \\(A \cup B) \cap (C \cap D) &= (A \cap (C \cap D)) \cup (B \cap (C \cap D)) \\&= (A \cap C \cap D) \cup (B \cap C \cap D)\end{aligned}$$

The concepts of sets and their union and intersection can be conveniently illustrated by means of a Venn diagram [1] as shown in Fig. 1. The areas within the boundaries

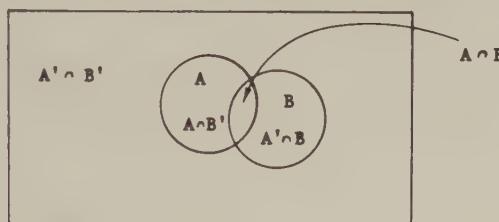


Fig. 1. Venn diagram.

are labeled on the diagram with the  $A$  and  $B$  sets of the previous discussion indicated by the area within the circles. Now considering the four distinct areas of Fig. 1, it is possible to describe any area or any class or set of areas making use of the union and intersection operations of sets. For example, the area within the circle labeled  $A$  can be alternately described as  $(A \cap B') \cup (A \cap B)$ . However, it is known by definition that this area is  $A$ . Thus the following identity expression is possible:

$$A = (A \cap B') \cup (A \cap B)$$

which can be rewritten to yield

$$A = A \cap (B' \cup B) = A \cap 1 = A$$

Thus it is possible to define a complete set of expressions for the various possible sets which can be obtained through the categorization of the quantities or elements contained in the logical universe.

## PROPOSITIONAL CALCULUS

Consider now the above sets and set expressions as representing propositions (combinations of words) which can be stated in the most general sense such that if  $A$  is itself a proposition, it must be either true or false. Thus a proposition in this sense is a combination of words, which of course makes sense, that must be either true or false.

Two propositions can be connected by means of a connective in the same manner as two sets were connected in the previous section by the union and intersection operations previously defined. The principle operations or connectives of propositional calculus are “ $\vee$ ” which is read as “or” and “ $\blacksquare$ ” which is read as “and.” A proposition can be negated as “ $\sim A$ .”

The connectives and negation for all possible combinations of two propositions  $A$  and  $B$  are shown in Table 1.

**TABLE 1**

$A$ and $B$	$A \cap B$	$A \blacksquare B$
$A$ and (not $B$ )	$A \cap B'$	$A \blacksquare \sim B$
(not $A$ ) and $B$	$A' \cap B$	$\sim A \blacksquare B$
(not $A$ ) and (not $B$ )	$A' \cap B'$	$\sim A \blacksquare \sim B$
$A$ or $B$	$A \cup B$	$A \vee B$
$A$ or (not $B$ )	$A \cup B'$	$A \vee \sim B$
(not $A$ ) or $B$	$A' \cup B$	$\sim A \vee B$
(not $A$ ) or (not $B$ )	$A' \cup B'$	$\sim A \vee \sim B$

Consider now the extension of the material on propositions to form statements or functions. The symbol “ $\supset$ ” is to be read “implies that” and provides a means of writing statements or functions composed of various propositions where any statements can be determined to be true or false depending on the truth or falsity of the individual propositions making up the statement. As an example, consider the following proposition of the previous section with  $A$  and  $B$  being two propositions.

$$A = (A \cap B') \cup (A \cap B)$$

The following statement can now be made:

$$A \supset (A \blacksquare \sim B) \vee (A \blacksquare B) \supset A \blacksquare (B \vee \sim B) \supset A$$

In other words stating that if either  $A$  is true and  $B$  is not true or  $A$  is true and  $B$  is true is equivalent to saying  $A$  is true and  $B$  or not  $B$  is true which is equivalent to stating that  $A$  is true.

Thus the above rather complex statement in terms of words, which is equivalent to the statement illustrated above, can be reduced to a simpler statement the validity of which can be more easily determined. Obviously it should be easier to deal with the symbolic statement in order to achieve any simplification.

The previous section dealt with the categorization and description of things which can be described making use of the concept of a set. The emphasis using propositions and statements illustrated in this section is on words, thoughts, ideas, etc., which

can be categorized in essentially the same manner as the sets discussed in the previous section with the concepts of one area having application in the other.

The categorization of the logical universe implied by the statement attributed to Aristotle opened a new branch of mathematics many years later; this is known as symbolic logic and was developed by the English mathematician George Boole [2]. The fundamental developments in this area are extremely interesting although they are beyond the scope of the present discussion. The categorization and the two connectives are, however, somewhat limited in their application to the everyday world.

The extension of the symbolic developments by Boole was in great part due to the Italian mathematician Giuseppe Peano [3]. As can be seen from the above limited discussion of propositions and statements, the vocabulary is an exciting part of what is known as basic deductive theory. It was Peano who extended this vocabulary providing the basis for later mathematical developments. The present day implementation of logic considered in this discussion has made use of only a few of Peano's developments. The future holds many exciting possibilities in new and different applications.

## TABLE OF COMBINATIONS

The *universe* or *universe of discourse* is the point of departure for the following discussion. Once a property, class, or proposition, to be identified by a variable which will be designated by, say,  $A$ , has been identified, it is possible to categorize all the elements of the universe as those for which  $A$  is true,  $A$ , and those for which  $A$  is not true,  $A'$ . It is assumed that the case is nontrivial, i.e.,  $A \neq 0$  and  $A' \neq 0$ . Thus all the elements of the universe are partitioned into two sets which can be identified by the two variables  $A$  and  $A'$ .

In turn, identification of another variable  $B$  allows still a further partition of all the elements of the universe into either  $B$  or  $B'$  giving rise to the possibilities  $AB$ ,  $AB'$ ,  $A'B$ , and  $A'B'$  for any element. The process can subsequently be repeated for  $C$ ,  $D$ , etc. Thus for four variables with 0 indicating a variable is false and 1 indicating a variable is true the following 16 possibilities exist.

$A\ B\ C\ D$	$A\ B\ C\ D$
0 0 0 0	1 0 0 0
0 0 0 1	1 0 0 1
0 0 1 0	1 0 1 0
0 0 1 1	1 0 1 1
0 1 0 0	1 1 0 0
0 1 0 1	1 1 0 1
0 1 1 0	1 1 1 0
0 1 1 1	1 1 1 1

For the  $n$  variable case, there is a total of  $2^n$  possible combinations within the universe. In the above example,  $n = 4$  giving  $2^4 = 16$ .

Consider now a proposition,  $P$ , which is true for some collection of combinations as shown below:

$A$	$B$	$C$	$D$	$P$
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

In the table,  $P$  is true ( $P = 1$ ) for the combinations  $A'BC'D'$ ,  $A'BC'D$ ,  $ABC'D'$ ,  $ABC'D$ .  $P$  is false ( $P = 0$ ) for all of the remaining combinations. If from any information it is known that  $P$  is true, then one of the four indicated combinations must be true. In order to identify which logical properties are indicated by the above table it can be said that if  $P$  is true then  $C$  is not true and  $B$  is true. In terms of the previously used symbols, the original statement

$$\begin{aligned} P = & (A' \cap B \cap C' \cap D') \cup (A' \cap B \cap C' \cap D) \\ & \cup (A \cap B \cap C' \cap D') \cup (A \cap B \cap C' \cap D) \end{aligned}$$

can be reduced to

$$P = (B \cap C')$$

The statement for  $P$  can be simplified from the one obtained from the original table of combinations. Thus it is possible to simplify the language of the statement of the conditions under which  $P$  is true. This simplification will have significance later when logic statements are implemented by physical elements.

## BOOLEAN ALGEBRA

In the nineteenth century, the mathematician George Boole developed an algebra, which has come to be known as Boolean algebra, that can be used in the manipulation and simplification of logical expressions.

The two operations fundamental to the developments of this section and the present utilization are the AND operation which is symbolized by a “.” and a type of OR operation termed an EXCLUSIVE-OR which is symbolized by a “ $\oplus$ ”. The  $\cdot$  of Boolean algebra is analogous to the  $\cap$  operation used previously. Consider the two given operations with the set  $\{0, 1\}$ , i.e.,  $\langle \{0, 1\}, \oplus, \cdot \rangle$  where the operations are defined by the following two tables:

$\oplus$	0	1	$\cdot$	0	1
	0	1		0	0
	1	0		1	0

The given formulation obeys what are termed the left and right distributive properties where for any  $x, y, z$  which can assume the values 0 or 1,

$$\begin{aligned}x \cdot (y + z) &= (x \cdot y) + (x \cdot z) \\(y + z) \cdot x &= (y \cdot x) + (z \cdot x)\end{aligned}$$

The set  $\{0, 1\}$  under the operations  $\oplus$  and  $\cdot$  is, in algebraic terms, a ring [4].

As an alternative, it is also possible using lattice theory to define a Boolean algebra as a complemented and distributed lattice [4]. In any event the set of elements composed of the operations  $\oplus$  and  $\cdot$  is called a Boolean algebra.

The use of the operation  $\oplus$  is useful from a mathematical standpoint in establishing the two operations and set as a ring. However, from a practical standpoint the operation defined by the following table is more useful.

$+$	0	1
0	0	1
1	1	1

The operation  $+$  is seen to be analogous to the operation previously discussed,  $\cup$ . The operation is called the INCLUSIVE OR or more simply the OR operation.

The newly defined OR operation also obeys the previously illustrated left and right distributive properties where for any  $x, y, z$  which can assume the values 0 or 1,

$$\begin{aligned}x \cdot (y + z) &= (x \cdot y) + (x \cdot z) \\(y + z) \cdot x &= (y \cdot x) + (z \cdot x)\end{aligned}$$

The development of logic using the  $+$ ,  $\cdot$  operations has been a result both of the properties of the operations and the ability to implement these operations with physical devices such as electronic circuits.

## IMPLEMENTATION

The implementation of logic using electric or electronic circuits will now be discussed making use of the two operations  $+$  and  $\cdot$  along with the set  $\{0, 1\}$ .

The logical universe is assumed to be given by a truth table where the number of variables in the universe is also assumed to be given. In addition, it is assumed that the output, i.e., the truth or falsity of each required output variable for each individual logical combination of the input variables, is also given.

Each combination of input variables from a truth table of an  $n$  variable universe,  $x_1 x_2 \cdots x_n$ , is termed a *minterm* where each variable may be in a primed  $x'_i$  or unprimed  $x_i$  state and termed a *literal*. According to the table of combinations, each minterm has a value of 0 or 1 according to whether the result represents a false or true condition. From the previous table,  $n = 4$ , and the minterm  $x_1' \cdot x_2 \cdot x_3' \cdot x_4$  is true or 1 and the minterm  $x_1' \cdot x_2 \cdot x_3 \cdot x_4$  is false or 0.

The collection of minterms for which the output is true can be written algebraically using the  $+$  and  $\cdot$  operations and is termed a *truth function*. For the table of combinations of the example given, the output is true if one or more of the four indicated minterms is true or, using the  $+$  operation, the output is true if

$$x'_1 \cdot x_2 \cdot x'_3 \cdot x'_4 + x'_1 \cdot x_2 \cdot x'_3 \cdot x_4 + x_1 \cdot x_2 \cdot x'_3 \cdot x'_4 + x_1 \cdot x_2 \cdot x'_3 \cdot x_4$$

is true which is written as the following truth function:

$$T = x'_1 \cdot x_2 \cdot x'_3 \cdot x'_4 + x'_1 \cdot x_2 \cdot x'_3 \cdot x_4 + x_1 \cdot x_2 \cdot x'_3 \cdot x'_4 + x_1 \cdot x_2 \cdot x'_3 \cdot x_4$$

Using the concepts of a minterm and a truth function it is possible to obtain mathematical expressions or functions which are equivalent to the original verbal problem statement, table of combinations, or truth table. Shannon [5] has shown that it is possible to establish a one-to-one correspondence between the binary valued (0, 1) and the contacts of a relay network [6].

Consider a wire, contact, or switch arrangement as shown in Fig. 2. Each switch  $A, B, C, D$  is shown in the open position. If any switch is closed, in the direction of

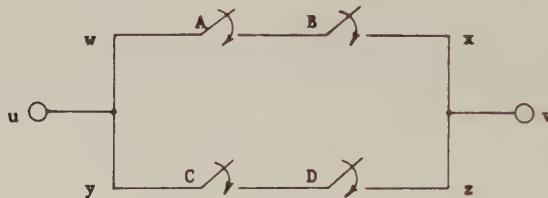


Fig. 2.

the arrow, it will complete a portion of the circuit. If both switches  $A$  and  $B$  are closed, the circuit will be complete between points  $w$  and  $x$ . If both switches  $C$  and  $D$  are closed, the circuit will be complete between points  $y$  and  $z$ . Thus the *series* connection of  $A$  and  $B$  is equivalent to the AND,  $\cdot$  operation. The open state of the switch corresponds to the 0 or false state of the variable  $A$ , i.e.,  $A = 0$ , the closed state of the switch to the 1 or true state of the variable  $A$ , i.e.,  $A = 1$ . The same rule applies to the other variables.

Now the circuit is complete in the above diagram if  $A$  and  $B$  are closed *or* if  $C$  and  $D$  are closed. In other words the *parallel* connection as shown in the diagram, with the circuit between the points  $w$  and  $x$  being in parallel with the circuit between the points  $y$  and  $z$  to form the circuit between the points  $u$  and  $v$ , is equivalent to the OR,  $+$  operation. The diagram is equivalent to the following truth function:

$$T = (A \cdot B) + (C \cdot D)$$

Proceeding in the same manner, it is possible to construct circuits for very complicated truth functions making use of the AND and the OR operations and the corresponding series-parallel implementation.

It is a relatively simple matter to use the circuit obtained through such an implementation in series with a battery and a light bulb or other electrical device in order to obtain an external indication such as a lit bulb when any configuration of

the input variables is such that the corresponding configuration given in the table of combinations indicates a true condition.

The logic circuit in Fig. 3 is any implementation of contacts or devices which are connected between two points  $u$  and  $v$  so as to obtain a completed (short) circuit or

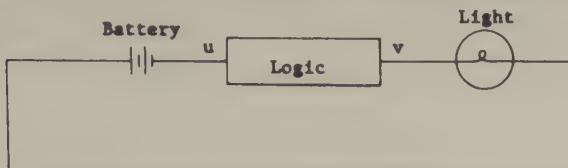


Fig. 3.

an uncompleted (open) circuit according to a prespecified table of combinations and truth function.

When any one of the true combinations of the input variables is met as indicated in the given table of combinations, the light will be lit. Otherwise the light will not be lit.

In the previous discussion, the NOT operation was not used. Consider the series-parallel logic circuit in Fig. 4. The given logic circuit can be shown to be the implementation of the truth function

$$T = (A \cdot B') + (C' \cdot D)$$

In other words, the circuit between points  $w$  and  $x$  is complete if  $A$  and  $B'$  are true, i.e.,  $A$  is true and  $B'$  is not true. Thus if  $A = 1$  and  $A' = 0$ , and  $B = 0$  and  $B' = 1$ ,

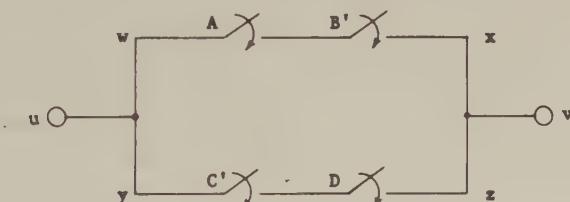


Fig. 4.

the series circuit between points  $w$  and  $x$  is complete. The same is true for the  $C'$ ,  $D$  case, i.e., if  $C$  is not true and  $D$  is true the series circuit between points  $y$  and  $z$  is complete.

Thus the implementation of every term of a specified table of combinations, including primed and unprimed literals, is possible using a series-parallel connection of logical devices (two-state). Normally, it is appropriate to name each device according to the input variable which it represents.

## PHYSICAL REALIZATION

Any logic function involving  $+$ ,  $\cdot$  logic can be realized (built) using the parallel-series methodology of the previous section. The actual physical implementation of

each variable of a truth function, a literal, represents an open-closed, 0-1 situation. A switch can be used for the two states in which case a person throws the switch to reflect the state of the variable. Thus one switch with this form of implementation is used for each literal of the given truth function.

The use of a switch thus implies a human interface between the indication of the state of the input variable and the state of the switch which is the implementation of the logic variable. Such an interface in many cases is extremely expensive and normally very slow.

Another possible means of implementation is an electromechanical device known as a relay [6]. An electrical signal is used to activate a part of the relay termed a coil, in order to "close the switch," in which case the state of the input variable is identified by the use of a voltage applied to the relay coil.

One possibility of implementing the input and output variables is a logical 0 signified by "no voltage" and a logical 1 signified by "a positive voltage" or "a negative voltage." Other possibilities of 0 and 1 identification are likewise possible. However, the important point here is that given the situation where all input variables are available in voltage form, the output can be automatically generated using a relay implementation. Thus as the input variables change, the corresponding output is automatically generated according to the truth function which is represented by the logic circuit. While relays once received a considerable amount of attention in the design of logic circuits and will always have a place in some implementation situations, they have for the most part been replaced by numerous types of electronic devices.

An electronic device which has been used for logic circuits is the vacuum tube [7]. The vacuum tube is a device which can be made to not conduct current (logical 1) or conduct current (logical 0) according to whether a voltage or no voltage is applied to one of its elements called a *grid*. Other such logical combinations are also possible. Thus the automatic nature of the logic circuit can be maintained using an electrical voltage as with the relay. The vacuum tube implementation results in a faster response logic circuit with less power required to operate the circuit than would be required with relay logic. The response is measured in terms of the time required to obtain the appropriate output for a change of the input combination. The increase in the speed of logic circuits using vacuum tubes in place of relays was significant. The first digital computer, ENIAC [8], completed at the University of Pennsylvania in 1945 used vacuum tube logic.

The transistor [9], invented in 1948, soon replaced the vacuum tube for logic circuit implementation. Once again an increase of speed was effected. The transistor is a solid-state device [9] which can be made to not conduct current (logical 1) or conduct current (logical 0) according to whether a voltage is applied to one of its elements called a *base*. Once again, other such combinations are also possible with various types of transistors and corresponding circuit configurations.

By this time in the discussion it should be obvious that the variety of methods of implementation is limited only by the state of the art of electronics and the ingenuity of the engineer. Other methods of implementing logic functions include diodes, tunnel diodes, magnetic cores, integrated circuits, and thin-film electronics. Appropriate references are included in the bibliography.

## COMBINATIONAL LOGIC CIRCUITS

The logic circuits discussed thus far are circuits in which the output, the logical output, at any given point in time is a function *only* of the input combinations which exist at the same point in time. Such circuits are termed *combinational logic circuits* or more simply *combinational logic*.

The realization of a combinational logic circuit begins with a table of combinations or truth table. Once an output has been specified for each input combination, a truth function is then obtained by "oring" together all those minterms corresponding to given combinations which have a logical 1 for an output. From the truth function which results, a circuit may be realized as discussed in the previous section.

Obviously, from the above discussion, as a circuit is realized there is a switch, vacuum tube, transistor, etc., for each literal of the truth function. Thus the cost of realization is essentially proportional to the number of literals appearing in the truth function which is to be realized. Therefore it is desirable from the standpoint of cost to minimize the appearance of literals in the truth function being realized.

## MINIMIZATION

One possible means of effecting a reduction in the cost of realization can be illustrated by considering the following logical relationship for any logical variable  $X$ .

$$X' + X = 1$$

Consider now the previous truth function:

$$T = x_1' \cdot x_2 \cdot x_3' \cdot x_4' + x_1' \cdot x_2 \cdot x_3' \cdot x_4 + x_1 \cdot x_2 \cdot x_3' \cdot x_4' + x_1 \cdot x_2 \cdot x_3' \cdot x_4$$

Applying the previously illustrated right distributive law to the above function in two instances as indicated below yields

$$T = x_1' \cdot x_2 \cdot x_3' \cdot (x_4' + x_4) + x_1 \cdot x_2 \cdot x_3' \cdot (x_4' + x_4)$$

Applying the relationship  $X' + X = 1$  and  $X \cdot 1 = 1 \cdot X = X$  yields

$$T = x_1' \cdot x_2 \cdot x_3' \cdot 1 + x_1 \cdot x_2 \cdot x_3' \cdot 1 = x_1' \cdot x_2 \cdot x_3' + x_1 \cdot x_2 \cdot x_3'$$

Now applying the left distributive law to the resulting function yields

$$T = (x_1' + x_1) \cdot x_2 \cdot x_3'$$

Once again applying  $X' + X = 1$  yields

$$T = 1 \cdot x_2 \cdot x_3' = x_2 \cdot x_3'$$

Thus a truth function with 16 literals taken directly from a table of combinations has been reduced, *minimized*, to a resulting truth function with two literals which is obviously cheaper to realize and is logically equivalent to the original truth function. The method of minimization illustrated here is but one of many which are possible. Other minimization techniques include the Karnaugh map [6] and the Quine-McClusky method [6].

For many years the search for minimization techniques which could handle larger

and larger numbers of input variables and require little effort to obtain a minimum function was quite significant and provided considerable interest for many researchers. However, recent developments in electronics have changed the emphasis of this now somewhat classical type of minimization.

The use of relays, vacuum tubes, and transistors involves three-dimensional circuits where it is possible to run insulated wires from any connection point to any other connection point. The three-dimensional circuit in this case refers to the space in which the actual components and associated wiring are placed to make up the physical realization of the logic circuit which has been designed. The three dimensions are obviously the spatial dimensions of the real world. In this context, a two-dimensional circuit is one which can be built and placed on a flat surface with no component, wire, etc., being above or below any other component, wire, etc. Otherwise, the circuit is three-dimensional. This description is obviously meant to be pictorial rather than rigorous. For the latter, the reader is referred to the work of Kuratowski [10].

For many years, no particular problem caused any concern over the two- or three-dimensional type circuits. However, newer developments in electronics make it possible to build a circuit out of elements which can be connected by conductors where both elements and conductors can be plated on a plane or two-dimensional surface. Two factors are particularly important here. First, the cost of any one device to implement a literal has correspondingly decreased by orders of magnitude with these newer technologies. Second, the cost of running a "wire" or conductor from any connection point to any other connection point is now a function of whether it must "crossover" another conductor. The use of a crossover thus causes a two-dimensional circuit to be three-dimensional. The cost of each crossover is now very high relative to the cost of a device. Consequently it is more important in many situations to minimize crossovers than it is to minimize the appearance of literals in the truth function.

It is important to note that any quantitative discussion at this point is difficult without dealing with particular technologies and components. It is possible to cite a particular example to illustrate the point. Certain logic circuits can be fabricated so cheaply that it is cheaper to double the number of logic elements than it is to incorporate one crossover. The illustration of a circuit requiring a crossover is deferred to the section entitled Logical Feedback and Memory. The minimization of crossovers is considerably more complicated than the minimization of literals, and the reader is referred to current literature in this area for additional discussion.

## NOT LOGIC

The implementation of a logic circuit using series and parallel connections involves AND-OR logic. In many instances it will be required to invert a logical variable which is the application of the NOT operation. Situations do exist where some or none of the

problem or input variables are available in their primed form. General problem implementation from a table of combinations requires the availability of inputs in both the primed and unprimed form. Thus it is required in many instances to generate the primed variables using a NOT circuit.

Logic circuits can be drawn diagrammatically for the previously described logic operations using the symbols shown in Fig. 5 for the AND-OR-NOT logic circuits.

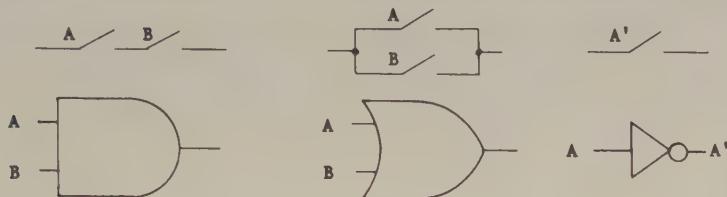


Fig. 5.

Thus the previous truth function used as an example and containing 16 literals can be drawn diagrammatically as shown in the logic diagram of Fig. 6 making use of the symbols in Fig. 5.

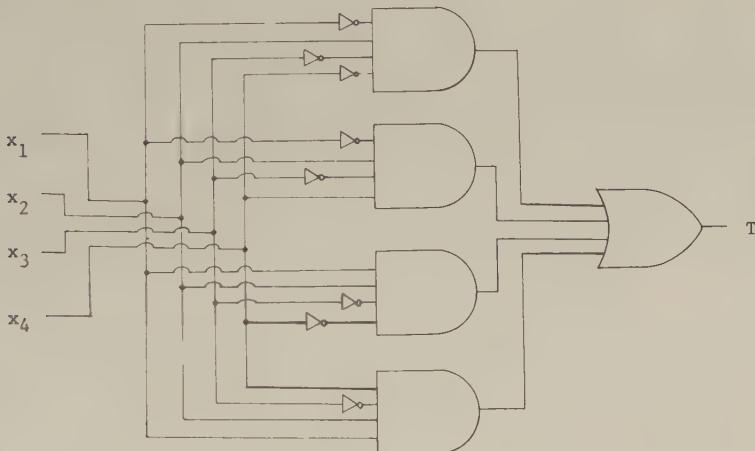


Fig. 6.

After simplifying, the truth function with two literals was obtained. The implementation of the reduced function yields the diagram in Fig. 7. Thus the output of both



Fig. 7.

of the above logic circuits is identical under all combinations of the input variables. The two circuits are logically equivalent.

## NAND-NOR LOGIC

In all of the above realizations three different logical functions are required: AND, OR, NOT. By making use of a combination of two of the given logical relationships and two particular types of electronic circuits, it is possible to realize any truth function using only one logic operation which results in a realization using only one type of electronic logic circuit or element.

The first logic operation and circuit to be considered is termed a NAND operation. The operation has a corresponding logic circuit with the symbol and table of combinations shown in Fig. 8. The function is simply an AND with a corresponding NOT applied to the result which gives the NOT AND or NAND operation.

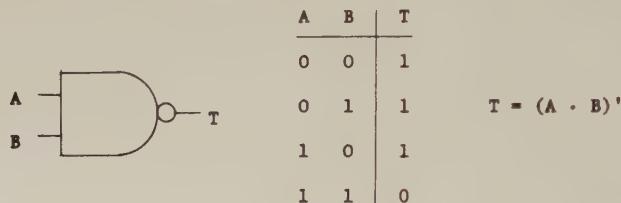


Fig. 8.

Consider any function of terms ORED together where each term is made up of literals which are in turn ANDED together. As a particular example without any loss to generality consider the following logic or truth function:

$$T = A' \cdot B \cdot C + A \cdot B' \cdot C + A \cdot B \cdot C'$$

The above function can now be manipulated into the following equivalent function using straightforward logical operations:

$$T = [(A' \cdot B \cdot C)' \cdot (A \cdot B' \cdot C)' \cdot (A \cdot B \cdot C')']'$$

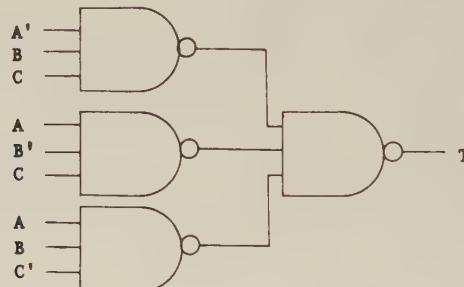


Fig. 9.

The procedure of manipulation should be obvious although the fact that the two functions are equivalent may not be quite so obvious. The resulting truth function is composed only of NAND elements and is realized as shown in the logic diagram in Fig. 9 where it is assumed that primed input variables are available.

The second logic operation and circuit to be considered is termed the NOR operation. The NOR operation has a corresponding logic circuit with the symbol and table of combinations shown in Fig. 10. The function is simply an OR with a corresponding

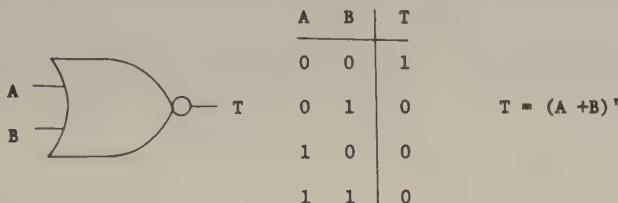


Fig. 10.

NOT applied to the result which gives the NOT OR or NOR. Once again consider the previous truth function example:

$$T = A' \cdot B \cdot C + A \cdot B' \cdot C + A \cdot B \cdot C'$$

The function can now be manipulated into the following equivalent functions again using straightforward logical operations:

$$T = (A + B' + C')' + (A' + B + C')' + (A' + B' + C)'$$

$$T = [(A + B' + C')' + (A' + B + C')' + (A' + B' + C)]'$$

Once again the procedure of the manipulation should be obvious although the fact that the two functions are equivalent may not be quite obvious. The above manipulation is not the only one possible and may not be the most desirable although it is the most convenient for the purpose of the present illustration. The resulting truth function is composed only of NOR elements and is realized as shown in Fig. 11, again assuming primed inputs are available.

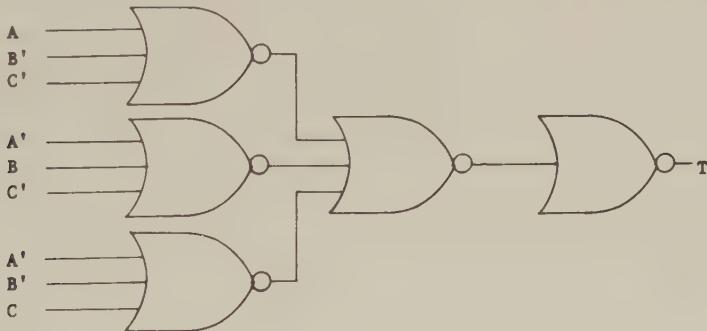


Fig. 11.

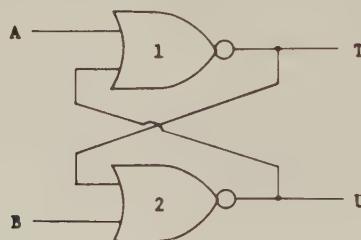
Thus the implementation or realization of logic functions as discussed here can be accomplished by the use of AND-OR-NOT-NAND-NOR logic. Other types of logical combinations may be grouped together (such as the previously discussed EXCLUSIVE-OR), but they are beyond the scope of the present discussion, and the interested reader is referred to the literature for additional material.

## LOGICAL FEEDBACK AND MEMORY

The previous discussions have dealt with logic and associated implementation where the output is strictly a function of the input. Given any logical combination, call it  $x$  as input with output  $T$ ; if the state of the input changes to  $y$  with output  $U$  and subsequently the input changes back to  $x$ , then the output will change to  $T$  for any choice of  $x$  and  $y$ . Such a circuit satisfying this condition is a *combinational* logic circuit. If the stated condition is not satisfied, then the circuit is a *sequential* logic circuit. The output is a function of the combination of inputs and also of the sequence in which they occur.

A sequential logic circuit has memory. It can remember, for example, that  $y$  existed by its output  $U$  even though the logical input combination  $y$  may change to another combination or state. The logical condition of the input variables at any given point in time will be termed the *input state*.

As an example of a logic circuit with memory, consider the circuit of Fig. 12 composed of two NOR logic blocks. The two NORs have been numbered 1 and 2 for



**Fig. 12.**

convenience. Note that the output of NOR1,  $T$ , has been "feedback" to the input of NOR2. The same is true of the output of NOR2,  $U$ , which has been "feedback" to the input of NOR1. Assume for the purpose of example that both  $A$  and  $B$  are logically 0 and that  $T$  is also 0. With both  $B$  and  $T$  logically 0,  $U$  is a logical 1, and with  $U$  a logical 1,  $T$  is a logical 0 as assumed.

Now let  $A$  remain a logical 0 and  $B$  change state to become a logical 1. From the truth table given for the NOR block,  $U$  will be a logical 0. Now with both  $A$  and  $U$  being a logical 0,  $T$  will change state and become a logical 1. Now let  $B$  return to the 0 state.  $T$  is now a logical 1 which causes  $U$  to remain a logical 0 and  $T$  a logical 1. Thus in a time sequence with the four-tuple giving the states of the indicated variables ( $A, B; T, U$ ), the input, output states are  $(0, 0; 0, 1)$ ,  $(0, 1; 1, 0)$ ,  $(0, 0; 1, 0)$ . Obviously with  $(0, 0; 0, 1)$  and  $(0, 0; 1, 0)$  the output(s) are not a combinational function of the input(s).

The above circuit has *memory* and is a sequential logic circuit. In particular, it is frequently termed a *flip-flop* or a *bistable*. The use of outputs as inputs as shown in the flip-flop is said to be *logical feedback* or simply *feedback*.

The reader will note that, for the illustrated flip-flop, if the circuit is laid out on a

two-dimensional surface, where the inputs and outputs are terminated on the boundaries, it is impossible to connect the feedback paths without crossing over at least one other logic line. Thus the circuit has one crossover. This situation was discussed in the previous material on minimization with respect to a problem which is of increasing concern, i.e., reducing the number of crossovers.

The use of flip-flops as memory elements provides a new dimension to the design of logic circuits, that being the design of sequential logic. The circuit with memory and the design using this memory can be more easily understood by considering the components of the circuit and the design in the following manner.

A set of input logic variables is assumed to be available where both the combinations of the input variables at any instant of time and the sequence of the different

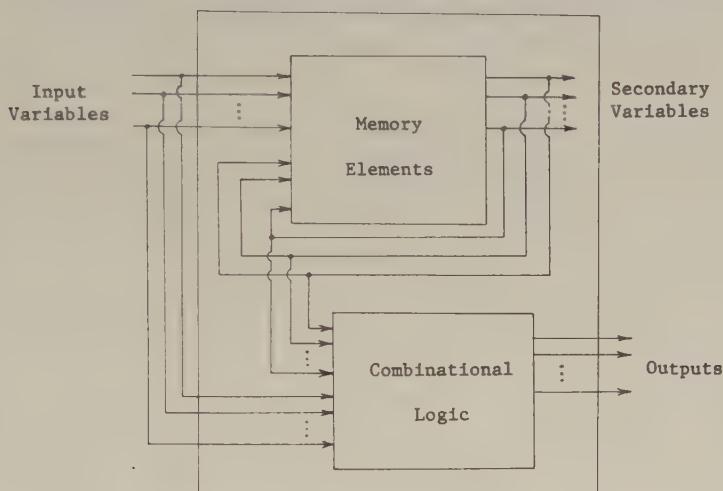


Fig. 13.

combinations both contain input information and are significant in determining the desired output.

The input variables in conjunction with other variables, to be discussed shortly, are used to generate logical information to be stored in the memory elements by directing the inputs to the appropriate terminals of the flip-flops. The pair of outputs of the memory element are the logical states of a variable and its complement which can be identified with the particular memory element, i.e., the given flip-flop. Thus if the flip-flop is designated  $A$ , one output is  $A$ , the other  $A'$ . These variables are termed *secondary variables* of the sequential circuit. Both input variables and secondary variables can be used to generate secondary variables. Thus the "other" variables mentioned at the beginning of this paragraph are these secondary variables.

The outputs of a sequential logic circuit are generated as logical combinations of the input variables and the secondary variables. The generation of the secondary variables is a sequential logic problem, and the generation of the outputs is a combinational logic problem.

The diagram of Fig. 13 is illustrative of the circuit which is to be designed. The

sequential logic circuits can be designed again making use of the AND-OR-NOT-NAND-NOR logic which has been discussed.

The design and utilization of sequential logic provides a point of departure for many interesting areas of study. The developments in all of these areas have been both numerous and rapid providing for exciting changes in the world around us. For that reason this discussion is only interrupted, not terminated.

## REFERENCES

1. J. Venn, *Symbolic Logic*, London, 1894.
2. G. Boole, *The Mathematical Analysis of Logic*, Cambridge, 1847; Oxford, Basil Blackwell, 1948.
3. J. R. Newman, *The World of Mathematics*, Vol. 1, Simon and Schuster, New York, 1956.
4. J. T. Moore, *Elements of Abstract Algebra*, Macmillan, New York, 1962.
5. C. E. Shannon, Symbolic analysis of relay and switching circuits, *Trans. AIEE* **57**, 713-723 (1938).
6. S. H. Caldwell, *Switching Circuits and Logical Design*, Wiley, New York, 1958.
7. Y. Chu, *Digital Computer Design Fundamentals*, McGraw-Hill, New York, 1962.
8. Funk and Wagnalls Standard Reference Encyclopedia, Vol. 7, p. 2315, New York, 1959.
9. M. V. Joyce and K. K. Clarke, *Transistor Circuit Analysis*, Addison-Wesley, Reading, Mass., 1961.
10. F. Harary, *Graph Theory*, Addison-Wesley, Reading, Mass., 1969.

## BIBLIOGRAPHY

- Ashenhurst, R. L., The decomposition of switching functions, *Ann. Comput. Lab., Harvard Univ.* **29**, 74-116 (1959).
- Boole, G., *An Investigation of the Laws of Thought*, London, 1854; Dover, New York, 1954.
- Curtis, H. A., *A New Approach to the Design of Switching Circuits*, Van Nostrand, Princeton, N.J., 1962.
- Dietmeyer, D. L., *Logic Design of Digital Systems*, Allyn and Bacon, Boston, 1971.
- Flores, I., *The Logic of Computer Arithmetic*, Prentice-Hall, Englewood Cliffs, N.J., 1963.
- Flores, I., *The Logic of Computer Arithmetic*, Prentice-Hall, Englewood Cliffs, N.J., 1963.
- Gill, A., *Introduction to the Theory of Finite-State Machines*, McGraw-Hill, New York, 1962.
- Hill, F. J., and G. R. Peterson, *Introduction to Switching Theory and Logical Design*, Wiley, New York, 1968.
- Hohn, F., *Applied Boolean Algebra*, Macmillan, New York, 1960.
- Hu, S. T., *Mathematical Theory of Switching Circuits and Automata*, University of California, Berkeley, 1968.
- Huffman, D. A., The synthesis of sequential switching circuits, *J. Franklin Inst.* **257** (3), 161-190; (4), 275-303 (1954).
- Huntington, E. V., Sets of independent postulates for the algebra of logic, *Trans. Amer. Math. Soc.* **5**, 288-309 (1904).
- Karnaugh, M., The map method for synthesis of combinational logic circuits, *Trans. AIEE*, Pt. I, **72**, 593-599 (1953).
- Keister, W., A. E. Ritchie, and S. H. Washburn, *The Design of Switching Circuits*, Van Nostrand, New York, 1951.
- Langer, S. K., *An Introduction to Symbolic Logic*, Dover, New York, 1953.
- Lawler, E. L., An approach to multilevel Boolean minimization, *J. Ass. Comput. Mach.* **11** (3), 283-295 (1964).

- McCluskey, E. J., Jr., Minimization of Boolean functions, *Bell Syst. Tech. J.* **35** (6), 1417–1444 (1956).
- McCluskey, E. J., Minimal sums for Boolean functions having many unspecified fundamental products, *Trans. AIEE, Pt. I* **81**, 387–392 (1962).
- McCluskey, E. J., Logical design theory of NOR gate networks with no complemented inputs, in *Switching Circuit Theory and Logical Design, Proceedings of the 4th Annual Symposium, Chicago, 1963*, pp. 105–116.
- McCluskey, E. J., *Introduction to the Theory of Switching Circuits*, McGraw-Hill, New York, 1965.
- Maley, G. A., and J. Earle, *The Logic Design of Transistor Digital Computers*, Prentice-Hall, Englewood Cliffs, N.J., 1963.
- Marcovitz, A. B., and J. H. Pugsley, *An Introduction to Switching System Design*, Wiley, New York, 1971.
- Marcus, M. P., *Switching Circuits for Engineers*, Prentice-Hall, Englewood Cliffs, N.J., 1967.
- Millman, J., and H. Taub, *Pulse, Digital and Switching Waveforms*, McGraw-Hill, New York, 1964.
- Moore, E. F., *Sequential Machines—Selected Papers*, Addison-Wesley, Reading, Mass., 1964.
- Nelson, R. J., Simplest normal truth functions, *J. Symbolic Logic* **20** (2), 105–108 (1955).
- O'Malley, J., *Introduction to the Digital Computer*, Holt, Rinehart, Winston, New York, 1972.
- Phister, M., *Logic Design of Digital Computers*, Wiley, New York, 1958.
- Quine, W. V., The problem of simplifying truth functions, *Amer. Math. Mon.* **59** (8), 521–531 (1952).
- Quine, W. V., A way to simplify truth functions, *Amer. Math. Mon.* **62** (9), 627–631 (1955).
- Richards, R. K., *Electronic Digital Components and Circuits*, Van Nostrand, New York, 1967.
- Sheng, C. L., *Introduction to Switching Logic*, Intext Educational Publishers, Scranton, Pa., 1972.
- Sobel, H. S., *Introduction to Digital Computer Design*, Addison-Wesley, Reading, Mass., 1970.
- Veitch, E. W., A chart method for simplifying truth functions, *Proc. Ass. Comput. Mach.*, 127–133 (May 1953).
- Werkmeister, W. H., *An Introduction to Critical Thinking*, Johnson, Lincoln, Nebr., 1958.
- Wickes, W. E., *Logic Design with Integrated Circuits*, Wiley, New York, 1968.

*Marlin H. Mickle*

## APL TERMINAL SYSTEM

### OVERVIEW OF THE APL SYSTEM

One of the most popular concepts in the short history of computer and information technology is the APL terminal system—commonly referred to as APL and frequently pronounced “apple.” APL, which is an acronym based on the title of the book, *A Programming Language*, authored by the distinguished computer scientist K. E. Iverson, combines two advanced technological concepts: (1) an advanced programming language, and (2) the time-sharing mode of system operation. Time sharing is useful without the APL language and the APL language would be useful without

time sharing. In combination, however, the two concepts provide the user with an extremely powerful computational facility. The APL system combines the simplicity of a desk calculator with the power of an automatic computer. APL is most useful for computer applications, large or small, to which a relatively fast response from the computer is needed.

### APL and Time Sharing

APL combines a recent technique called time sharing with Iverson's programming language to provide an interactive system that is appropriate for large and small problems and convenient for human use. *Time sharing* is a technique that allows several users at remote locations to concurrently use a computer via communications facilities. The system is accessed with a keyboard-type remote terminal device that is used for both input and output. The user types the information he wants sent to the computer and the computer responds, normally within seconds, with appropriate results. The discourse between the user and the computer is termed "interactive computing."

Clearly, the computer can only execute one program at a time. Thus, to provide the above service concurrently to several users, the computer is programmed to operate in a time-sharing mode. When time sharing, the central computer gives each user a short burst of computer time, called a *time slice*, on a periodic basis. The user is given the illusion of having the computer to himself, whereas in reality, it is switching rapidly between different users. Because of the high speed of modern computers, program control can be switched between users without an appreciable delay in a user's request for computation.

Most computer operating systems utilize a "supervisor" program to switch control of the computer between different user programs. The APL system includes a supervisor program of its own and as a result, can be run along with other non-APL type programs in a normal production environment.

### The APL Language

The APL language is based on the notation presented in Iverson's book, *A Programming Language*, mentioned previously. The language is based on the concise notation of mathematics and includes a multiplicity of primitive scalar functions that are extended on an element-by-element basis to *n*-dimensional arrays—that is, vectors, matrices, and arrays of higher dimension. The language also includes a set of functions defined on arrays, several statement types, and a variety of operational conventions. Since the APL language is designed to operate on arrays, much of the detail ordinarily associated with computer programming is subordinated to the APL system.

Although the APL language is based on well-defined mathematical notation, the use of APL is not limited to scientific and mathematical problems. APL has been

used for business applications, computer-assisted teaching, information storage and retrieval, text editing, logical analysis, and for simulation.

### Operation of the APL System

The APL system is designed as an interactive terminal facility. Normally, the user enters a statement and the APL system responds immediately, as in

2 + 2

4

This is termed the *execution mode*. The APL system indents the carriage of the user's terminal device six spaces and unlocks the keyboard so that information can be entered. The user types the information he wants sent to the computer and pushes the RETURN key. The keyboard is locked and the computer performs the required calculations and prints the computed results, if appropriate, beginning in the left-hand margin. The carriage is then moved up one line, indented six spaces, and unlocked for input. The system is then ready for the next interaction with the user. The following APL script demonstrates the above concepts:

```
A ← 5
B ← 2
1 + A × B
```

11

Programs are constructed in what is termed the *definition mode*. In this mode, statements are not executed immediately but are saved as part of a defined function. As an example, the following defined function gives descriptive statistics of a list of values

```
[1]      ∇ ← STAT LST
[2]      SUM ← +/LST
[3]      MAX ← ⌈/LST
[4]      MIN ← ⌊/LST
[5]      AV ← (+/LST) ÷ ρLST
[6]      MEDIAN ← 0.5 × +/(LST[ρLST])[0.5 + (0, 1) + LST]
[7]      R ← SUM, MAX, MIN, AV, MEDIAN
      ∇
```

It is applied to a simple list in the following script:

```
STAT -3 4 7 2 5
15    7   -3   3   4
```

Each of the constructs used in preceding examples is described in subsequent sections.

### Statements and Commands

Input to the APL system takes two principal forms: statements in the APL language and system commands. Statements in the APL language fall into three categories.

1. Specification statements, such as

$A \leftarrow 2 \times 3 + 4$

that assign a value to a variable

2. Branch statements, such as

$\rightarrow LOOP + 2$

that direct program control, in a defined function, to a specified statement number

3. Function definitions, such as

$\nabla R \leftarrow X PLUS Y$

[1]             $R \leftarrow X + Y$

[2]             $\nabla$

that are used to construct programs

In general, specification and branch statements permit expressions as arguments. Expressions can be composed of constants, variables, functions, and parentheses in the usual manner.

System commands are used to address the APL terminal system itself and provide operational functions outside of the APL language. Sample functions performed by system commands allow the user to sign on or off or to save, load, or copy a workspace. A system command starts with a right parenthesis followed by appropriate letters or digits. Sample system commands are given as follows:

)OFF  
)CLEAR  
)COPY ASPACE THETA

System commands augment the APL language by supplying operational facilities that are primarily implementation dependent.

### Functions

The APL system provides the user with an extensive set of arithmetic operations and mathematical functions that are collectively referred to as *primitive functions*. Sample primitive functions are given in Table 1.

TABLE 1

Primitive function	Meaning	Example
$A + B$	$A$ plus $B$	$7 \leftrightarrow 4 + 3$
$- B$	minus $B$	$-7 \leftrightarrow -7$
$A \times B$	$A$ times $B$	$-6 \leftrightarrow 2 \times -3$
$A \lceil B$	maximum of $A$ and $B$	$3 \leftrightarrow 2 \lceil 3$
$\lfloor B$	floor of $B$	$2 \leftrightarrow \lfloor 2.71828$
$A > B$	$A$ greater than $B$	$1 \leftrightarrow 2 > -1$
$A \vee B$	$A$ or $B$	$1 \leftrightarrow 0 \vee 1$
$! B$	$B$ factorial	$120 \leftrightarrow ! 5$
$  B$	absolute value of $B$	$1 \leftrightarrow   -1$

Primitive functions are extended on an element-by-element basis to arrays. For example,

			1	2	3 + 4	5	6
5	7	9					
			3 + 1	2	3		
4	5	6					
			!3	5	4		
6			120		24		

Array specifications are given in later sections.

Defined functions are used in a similar fashion to that of primitive functions. For example, consider the *PLUS* function defined and used as follows.

```

 $\nabla R \leftarrow A \text{ PLUS } B$ 
[1]    $R \leftarrow A + B$ 
[2]    $\nabla$ 
       $2 \text{ PLUS } 3$ 
5

```

The del symbol ( $\nabla$ ) denotes the start of a function and puts the APL system into the definition mode. The function header statement, i.e.,

$\nabla R \leftarrow A \text{ PLUS } B$

specifies that *PLUS* is the name of the function, that it has two arguments *A* and *B*, and that it returns an explicit result. In this case, the statement

$R \leftarrow A + B$

comprises the body of the function and the final del symbol closes the function definition and puts the APL system back into the execution mode. The *PLUS* function uses two arguments. Defined functions can use zero, one, or two arguments and are similar in this regard to standard mathematical operations. Arguments can be scalar values or arrays. The *STAT* function given previously uses one argument that is a one-dimensional array.

### The Workshop Concept and Program Libraries

Associated with each active APL terminal is a fixed-size block of storage in the computer called a *workspace*. The workspace is used to hold a user's working storage, control information, variables, and defined functions. By saving a workspace, the state of a terminal session, at a given point in time, can be preserved. Thus, a user can interrupt his work, store his workspace in a library, and deactivate his terminal. Later, he can connect to the system again, load his workspace, and continue his work. Most APL statements and system commands entered by a user are reflected in his active workspace. In addition to variables and defined functions, the workspace also includes control information on the execution of interrupted functions and system parameters that govern the manner in which the APL system responds to the user's

request for service. Sample system parameters are the line width (for printing), the indexing origin, and a "seed" value used for generating random numbers.

Each user that is permitted access to an APL system is assigned a library that can hold a given number of workspaces. A user "saves" workspaces in this library for later "loading." Sample system commands for workspace management are

```
)CLEAR
)SAVE DELPHI
```

and

```
)LOAD DELPHI
```

*CLEAR* loads a clear workspace for the user. *SAVE* stores the current active workspace in the user's library under the name *DELPHI*. The *LOAD* command replaces the active workspace with the workspace named *DELPHI* from the user's library.

The APL terminal system is convenient for human use because it is patterned after mathematical notation and the rules of the language are consistently applied. In reality, APL is a sophisticated system that includes a multiplicity of detailed concepts. In subsequent sections, detailed information on the APL language is given. Factors relating to a particular implementation of the language are generally excluded except for the most widely used system commands.

## APL LANGUAGE CHARACTERISTICS

The APL language is characterized by its typeface and keyboard arrangement, its statement structure, functions, data structures to which functions are applied, and the manner in which expressions are formed.

### CHARACTERS AND SYMBOLS

The APL language uses a special alphabet designed to facilitate the human factors aspects of man-computer interaction. The letters are capitalized italics and the digits

APL KEYBOARD



Fig. 1. APL keyboard arrangement.

are upright allowing similar characters to be readily distinguished. The APL alphabet and keyboard arrangement are given in Fig. 1 and the keyboard symbols are given in

Table 2. Table 3 lists the APL symbols for some frequently confused characters. The special symbols of the APL alphabets are included mainly as upper case characters and usually have a mnemonic association with their lower case counterparts. For example,  $\omega$  is over  $W$ ,  $\varepsilon$  is over  $E$ ,  $\rho$  (for rho) is over  $R$ ,  $*$  (for power) is over  $P$ ,  $\circ$  (circle symbol) is over  $O$ ,  $\alpha$  is over  $A$ , and  $'$  (for kwote) is over  $K$ .

**TABLE 2**  
APL Keyboard Symbols

Symbol	Name	Symbol	Name
$A \dots Z$	Letters	$\ $	Angle beams
$0 \dots 9$	Digits	$\_$	Underscore
$-$	Negative sign	$\nabla$	Del
$< \leq = \geq > \neq$	Comparison operators	$\Delta$	Delta
$\wedge \vee \sim$	Logical operators	$\circ$	Small circle
$+ - \times \div *$	Arithmetic operators	$'$	Quote
$?$	Question mark	$\square$	Quad
$\varepsilon$	Epsilon	$( )$	Parentheses
$\rho$	Rho	$[ ]$	Brackets
$\uparrow$	Up arrow	$\perp \top$	$\top$ beams
$\downarrow$	Down arrow	$ $	Vertical stroke
$\iota$	Iota	$;$	Semicolon
$\circ$	Circle symbol	$:$	Colon
$\rightarrow$	Branch arrow	$.$	Period
$\leftarrow$	Specification arrow	$,$	Comma
$\cap$	Cap	$/$	Solidus
		$\backslash$	Reverse solidus

**TABLE 3**  
APL Symbols for some  
Frequently Confused Characters

Digit	Letter
0	$O$
1	$I$
2	$Z$

A symbol of the APL language is one or more characters that is assigned a meaning that is not inherent in the constituent characters. Several symbols are well known:  $+$  for *plus*,  $-$  for *minus*,  $\times$  for *times*,  $\div$  for *divide*,  $\sim$  for *complement*, etc. In addition, APL permits composite symbols to be formed. A composite symbol is formed from two characters of the APL alphabet by backspacing and overstriking. For example,  $\circ$ , which is a function symbol for logarithm, is formed from the circle symbol ( $\circ$ ) and the asterisk (\*). Similarly, the factorial symbol (!) is formed from the quote symbol ('') and the decimal point (.). The meanings of other APL symbols are given when the respective function is introduced. Composite symbols are listed in Table 4.

In the APL language, a mathematical function is represented by a function symbol. Most function symbols are well known, such as + for plus,  $\times$  for times, and ! for factorial. In this respect, the APL notation is similar to mathematical notation. APL

**TABLE 4**  
APL Composite Symbols

Composite symbol	Used for	Formed with
!	Combination, factorial	' .
◎	Comment	○ °
▽	Grade down	▽
△	Grade up	△
⊤	I beam	T ⊥
*	Logarithm	○ *
⍲	NAND	∧ ~
⍲	NOR	∨ ~
⍲	Protected function	▽ ~
⌜	Quote-quad	□ '
⌞	Reversal, rotation	○
⌋	Transpose	○ \

notation, however, differs from mathematical notation in two important ways. First, every function has a symbol and that symbol must never be elided. Thus, implied multiplication, as in the mathematical expression

$$y = a + bx$$

must be made explicit in APL with the times symbol, that is,

$$Y \leftarrow A + B \times X$$

Second, the design of keyboard terminal devices restricts the input line to a linear sequence of characters. As a result, functions such as exponentiation that are usually denoted by a raised argument must be explicitly stated with appropriate function symbols. Thus, the mathematical expression

$$y = x^2$$

must be written in the APL language as

$$Y \leftarrow X * 2$$

[Note here that the asterisk (\*), which usually denotes multiplication in programming languages such as FORTRAN, COBOL, and PL/1, denotes the power function (exponentiation) in the APL language.]

The APL language deviates in another obvious way from other programming languages. The equal sign (=) that frequently denotes assignment, such as

$$A = 17$$

is replaced in the APL language with the left pointing arrow, i.e.,

$A \leftarrow 17$

The convention frees the equal sign for use as a test of strict equality.

### Data Types and Constant Values

The APL system is designed to accept and operate on data in two principal forms: numeric and character. A numeric datum is expressed in decimal form and is stored by the APL system as either a fixed-point or a floating-point number, whichever is most appropriate. A numeric constant is written in the usual fashion, that is, as a sequence of digits optionally preceded by a negative sign and possibly containing a decimal point. In addition, a numeric constant can be scaled by a power of ten by following the constant with the letter *E* followed by the power which must be expressed as either a positive or negative integer. Thus, the following are equivalent numeric constants:

123.45	and	0.12345E3
-30000	and	-.3E5
.00789	and	0.789E-2

The negative sign can only be used in the construction of numeric constants and should be distinguished from the minus sign used for negation and subtraction.

The logical truth values "true" and "false" are represented by the numeric values 1 and 0, respectively. Thus, the result of a relational or logical operation can be used as an operand in an arithmetic expression.

A constant vector is entered by typing elements of the one-dimensional array in order separated by one or more spaces. For example, the statement

$A \leftarrow 2 3 1$

assigns the vector with elements 2, 3, and 1 to the variable *A*. Computer storage for scalar and array data is handled dynamically in APL so the user need not be concerned with declaring the dimensions of an array. In fact, the size of an array may change during the course of computation.

A *character datum* is written as a series of characters enclosed in quote marks (e.g., 'TOM AND JERRY') and is called a *literal*. A single character is stored as a scalar value or an element of an array so that a literal composed of two or more characters is automatically stored as a one-dimensional array.

Matrices and arrays of higher dimension are covered in a later section.

### Names and Variables

A *name* is a sequence of the letters *A* through *Z*, the digits 0 through 9, or the character  $\Delta$ . A letter may additionally be underscored. The first character of a name must not be a digit and the beginning sequences  $S\Delta$  and  $T\Delta$  are not permitted because

they serve other functions in the language. In APL, function symbols, space characters, and punctuation characters serve as delimiters so that imbedded spaces are *not* permitted in names (or constants, as a matter of fact). Thus,

$A2B$	$ALL\Delta DONE$	$\Delta T$
$\underline{K25}$	$ABCD$	$ROOT\Delta OF\Delta EQUATION$

are all valid names. A name can be used to identify a scalar variable, an array, a statement (i.e., a statement label), a defined function, a workspace, or a group of names.

Theoretically, no limit on the length of a name exists. Most implementations of APL, however, impose a practical limit. In actual practice, concise names are frequently used because they occupy less space in a user's workspace.

A *variable* is a name assigned to a scalar datum or an array. A variable is not declared beforehand and is assigned attributes during the operation of assignment (or specification, as it is called in APL). Thus, a given name can identify a scalar, a numeric array, and a literal—all in the same program but obviously not at the same time.

### Specification

The specification function is denoted by the left-pointing arrow and is used to assign a value to a variable. Specification corresponds to the assignment statement or the replacement operation in other programming languages. For example, the statement

$LOAD \leftarrow 6.389$

specifies the value of the variable *LOAD* as 6.389 and causes that value to be stored in the active workspace. *LOAD* is classified as a scalar variable. Similarly,

$COT \leftarrow 6 \ 3 \ 1$

specifies that *COT* is a vector composed of the elements 6, 3, and 1; the length of *COT* is implicitly specified as 3.

The specification function is one of the criteria used by the APL system for deciding whether or not to display computed results on the user's terminal device. After the execution of a mathematical function or a series of mathematical functions, the APL system makes the following decision.

1. If the specification function is given as the last function to be executed, then the result is assigned to the specified variable and no printing takes place.
2. If the specification function is not given as the last function to be executed, then the system assumes that the user would like to see the result and displays it at his terminal.

Thus, a statement, such as

$A \leftarrow 3 * 2$

causes no output to be generated while the computation without the specification function, i.e.,

9  
3 \* 2

causes the computed result to be printed.

The APL language permits multiple specification, i.e.,

$A \leftarrow B \leftarrow 5$

which is the same as

$A \leftarrow 5$   
 $B \leftarrow 5$

and specification need not be the last function to be performed in an expression. This subject is discussed further under "expressions."

### Functions and Expressions

A function is characterized by the number of arguments it uses and the result it generates. The concept applies to both primitive functions and defined functions. A *monadic function* is written as

$\oplus A$

where  $\oplus$  stands for a function symbol and  $A$  is an operand; it is interpreted to mean that the function  $\oplus$  is applied to the argument  $A$ . A common monadic function is negation so that  $-W$  produces the value 5 when  $W = -5$ . A *dyadic function* is written  $A \oplus B$  where  $A$  and  $B$  are arguments and  $\oplus$  again stands for a function symbol; it is interpreted to mean that the function  $\oplus$  is applied to arguments  $A$  and  $B$ . For example, the expression  $3 \times P$  produces the value 6 when  $P = 2$ . The APL language includes the standard arithmetic, relational, and logical functions. These functions are primarily dyadic with the only monadic functions being negation ( $-$ ) and complement ( $\sim$ ). In addition, APL contains a multiplicity of other primitive functions, both monadic and dyadic, that correspond to operations most frequently performed in computation.

An *expression* is a combination of variables, constants, function symbols, and parentheses. Parentheses are used for grouping, and expressions within parentheses are executed before the operation of which they are a part. Because of the large number of functions in APL, it is impractical and confusing to establish a hierarchy among the various functions, and a strict right-to-left order for the evaluation of expressions is established. Thus, the expression  $2 \times 3 + 4$  evaluates to 14 while the expression  $(2 \times 3) + 4$  has the value 10.

The distinction between a monadic and dyadic function is easily determined. Given a symbol, such as the minus sign, that can represent either a monadic or dyadic function, the function is interpreted as monadic if the symbol to its immediate left is another function symbol. Otherwise, it is interpreted as a dyadic function. In the expression

$$A \times -B$$

for example, the symbol  $(-)$  is interpreted as the monadic negation function whereas in the expression

$$A \times B - C$$

the same symbol is executed as the dyadic minus function. The following script gives examples of APL expressions using commonly known functions.

```

14      2 × 3 + 4
9       3 * A ← 2
2       A
5       B ← 10
4       B - 7 - A
5       8 ÷ 4 ÷ 2
4       (8 ÷ 4) ÷ 2
1       W ← 4
7       -1 + A × W
0       25 × A > B

```

The right-to-left rule, which is a necessary convention, creates some unexpected results. In general, repeated subtraction is the difference of the sum of alternating arguments, e.g.,

$$A - B - C - D \leftrightarrow (A + C) - (B + D)$$

and repeated division is the quotient of the product of alternating arguments, i.e.,

$$A \div B \div C \div D \leftrightarrow (A \times C) \div (B \times D)$$

An important case demonstrated in the above APL script is that the specification function generates an explicit result in addition to performing a replacement operation. For example, in the expression

$$(B \leftarrow 3) * A \leftarrow 2$$

the variable *A* is assigned a value of 2, the variable *B* is assigned a value of 3, and the exponentiation function is executed using the result of both specification functions.

### Elementary Input and Output

Numeric input and output is denoted with the quad symbol ( $\square$ ). When the quad symbol appears to the left of the specification symbol, output is specified as shown in the following example.

$\square \leftarrow T \leftarrow 17$

17

The quad symbol may also be embedded in an expression to display intermediate results; that is,

$A \leftarrow 10 * \square \leftarrow 4 - 2$   
2

*A*

100

or more succinctly

$\square \leftarrow A \leftarrow 10 * \square \leftarrow 4 - 2$   
2  
100

The appearance of the quad symbol in a position that is not directly to the left of a specification symbol denotes numeric input as depicted in the following example.

$A \leftarrow 10 \times \square$   
 $\square:$   
2  
*A*  
20

The quad symbol  $\square$  is used anywhere in a statement that a constant or a variable is used. When the quad symbol is encountered, the APL system halts execution of the statement and makes an input request by typing the quad symbol followed by a colon. The carriage is moved up one line, indented six spaces, and unlocked. The user enters the input value and the execution of the statement continues as though the value entered were actually part of the statement. Input requests to the terminal are made as quad symbols are encountered during the right-to-left scan of a statement; this convention is depicted in the following example.

$B \leftarrow \square \div \square$   
 $\square:$   
2  
 $\square:$   
8  
*B*  
4

The input function is actually more general than implied. In response to the input quad symbol, the user can enter an expression that is evaluated at the point of reference; the execution of the statement containing the quad symbol is then continued as though the computed value of the expression were actually part of the statement. For example,

```
X ← 1
□ ← (A * 3) + (A ← □) * 2
□:
X + 1
```

12

The value of the expression in this example is equivalent to the value of the expression  $2^3 + 2^2$ .

The APL system also permits the input and output of character data. Since a character literal is most frequently stored as an array, this topic is covered in the section entitled "Vectors and Vector Functions."

## PRIMITIVE SCALAR FUNCTIONS

A *primitive function* is a function that is part of the APL language. A primitive *scalar* function is a primitive function that is defined on scalar values and extended on an element-by-element basis to arrays.

### Arithmetic Functions

The primitive scalar arithmetic functions in the APL language are given in Table 5. Each symbol has both a monadic and a dyadic interpretation and most symbols are related in some fashion to the functions they represent. All functions are defined on real numbers—regardless of the representation used in the computer. The key point is that the arithmetic function can be combined with other APL functions to construct compound expressions. The right-to-left rule for interpretation always applies, as in the following statement that rounds a value  $B$  to  $N$  places.

$$(10 * - N) \times [0.5 + B \times 10 * N]$$

### Comparison and Logical Functions

The primitive scalar comparison and logical functions are summarized in Table 6. All functions are dyadic except for the logical complement function, which is monadic. Using 1 and 0 for truth values has certain advantages as demonstrated in the APL statement

**TABLE 5**

Primitive Scalar Arithmetic Functions in APL

Function	Monadic or dyadic	Function symbol	Definition	Meaning	Example
$A + B$	D	+	$A$ plus $B$	$A + B$	$3 + 5 \leftrightarrow 8$
$+ B$	M	+	$B$ (identity)	$0 + B$	$+ 6 \leftrightarrow 6$
$A - B$	D	-	$A$ minus $B$	$A - B$	$5 - -3 \leftrightarrow 8$
$- B$	M	-	Minus $B$	$0 - B$	$- -3 \leftrightarrow 3$
$A \times B$	D	$\times$	$A$ times $B$	$A \times B$	$5 \times 3 \leftrightarrow 15$
$\times B$	M	$\times$	Sign of $B$	$(0 < B) - 0 > B$	$\times -3 \leftrightarrow -1$
$A \div B$	D	$\div$	$A$ divided by $B$	$A \div B$	$3 \div 2 \leftrightarrow 1.5$
$\div B$	M	$\div$	Reciprocal of $B$	$1 \div B$	$\div .5 \leftrightarrow 2$
$A * B$	D	*	$A$ power $B$	$A^B$	$3 * 2 \leftrightarrow 9$
$* B$	M	*	$e$ power $B$	$e^B$	$* B \leftrightarrow (2.71828...) * B$
$A \lceil B$	D	$\lceil$	Maximum of $A$ and $B$	$A$ if $A > B$	$3 \lceil 2 \leftrightarrow 3$
$\lceil B$	M	$\lceil$	Ceiling of $B$	$B + 1 \lceil -B$	$\lceil 3.14 \leftrightarrow 4$
$A \lfloor B$	D	$\lfloor$	Minimum of $A$ and $B$	$A$ , if $A \leq B$	$3 \lfloor 2 \leftrightarrow 2$
$\lfloor B$	M	$\lfloor$	Floor of $B$	$B - 1 \lfloor B$	$\lfloor -3.14 \leftrightarrow -3$
$A   B$	D		Residue of $B$ (mod $A$ )	$B - (\lfloor A) \times \lfloor B \div   A,$ if $A \neq 0$ $B$ , if $A = 0$ and $B \geq 0$ Error, if $A = 0$ and $B < 0$	$2   5 \leftrightarrow 1$
$  B$	M		Absolute value of $B$	$B \lceil -B$	$1.5   3.4 \leftrightarrow 0.4$

$$Y \leftarrow (((X > 0) \wedge X \leq 50) \times 3.51) + (X > 50) \times 134.138$$

that computes the step function

$$\begin{aligned} y &= 0, && \text{if } x \leq 0 \\ y &= 3.51, && \text{if } 0 < x \leq 50 \\ y &= 134.138, && \text{if } x > 50 \end{aligned}$$

### Mathematical Functions

Frequently used mathematical functions are included in APL as primitive scalar functions; they are summarized in Table 7. Clearly, each of the mathematical functions could be programmed using the other primitive functions; however, the use of primitive functions reduces the number of keystrokes to construct a program and increases precision and accuracy.

**TABLE 6**

Primitive Comparison and Logical Scalar Functions

Function	Monadic or dyadic	Function symbol	Definition	Meaning <sup>a</sup>	Example
$A < B$	D	<	$A$ less than $B$	1, if $(A - B) \leq -\text{fuzz} \times  B $ 0, otherwise	$2 < 3 \leftrightarrow 1$
$A \leq B$	D	$\leq$	$A$ less than or equal to $B$	1, if $(A - B) \leq \text{fuzz} \times  B $ 0, otherwise	$3 \leq 3 \leftrightarrow 1$
$A = B$	D	=	$A$ equal to $B$	1, if $( A - B ) \leq \text{fuzz} \times  B $ 0, otherwise	$3 = 4 \leftrightarrow 0$
$A \neq B$	D	$\neq$	$A$ not equal to $B$	1, if $( A - B ) > \text{fuzz} \times  B $ 0, otherwise	$2 \neq 3 \leftrightarrow 1$
$A \geq B$	D	$\geq$	$A$ greater than or equal to $B$	1, if $(A - B) > -\text{fuzz} \times  B $ 0, otherwise	$3 \geq 2 \leftrightarrow 1$
$A > B$	D	>	$A$ greater than $B$	1, if $( A - B ) > \text{fuzz} \times  B $ 0, otherwise	$2 > 2 \leftrightarrow 0$
$A \vee B$	D	$\vee$	$A$ or $B$	0, if $A = 0$ and $B = 0$ 1, otherwise	$1 \vee 1 \leftrightarrow 1$ $0 \vee 0 \leftrightarrow 0$
$A \wedge B$	D	$\wedge$	$A$ or $B$	1, if $A = 1$ and $B = 1$ 0, otherwise	$1 \wedge 1 \leftrightarrow 1$ $0 \wedge 0 \leftrightarrow 0$
$\sim B$	M	$\sim$	Not $B$	$1 \neq B$	$\sim 1 \leftrightarrow 0$ $\sim 0 \leftrightarrow 1$
$A \blacktriangleleft B$	D	$\blacktriangleleft$	$A$ NAND $B$	$\sim A \wedge B$	$1 \blacktriangleleft 1 \leftrightarrow 0$ $1 \blacktriangleleft 0 \leftrightarrow 1$ $0 \blacktriangleleft 0 \leftrightarrow 1$
$A \blacktriangleright B$	D	$\blacktriangleright$	$A$ NOR $B$	$\sim A \vee B$	$1 \blacktriangleright 1 \leftrightarrow 0$ $1 \blacktriangleright 0 \leftrightarrow 1$ $0 \blacktriangleright 0 \leftrightarrow 1$

<sup>a</sup> "Fuzz" is a small value in the neighborhood of  $1.0E^{-13}$ . It is used to compensate for performing decimal arithmetic on a binary machine. Definitions are based on numeric values. Character values may be tested only for equality and the result is true if the relationship holds.

## ORDER OF ARGUMENTS

Although functions in a compound expression are executed in a right-to-left sequence, the order of arguments established in mathematics is maintained in most cases. For example,

$A \div B$  means  $A$  divided by  $B$

and

$A * B$  means  $A$  power  $B$

In general, any function assumes as its rightmost argument the value of the entire expression to the right of its function symbol.

**TABLE 7**

## Primitive Scalar Mathematical Functions

Function	Monadic or dyadic	Function symbol	Definition	Meaning	Example
$A ! B$	D	!	Combinations of $B$ things taken $A$ at a time	$(! B) \div (! A) \times ! B - A$	$2 ! 5 \leftrightarrow 10$
$! B$	M	!	$B$ factorial or the gamma function of $(B - 1)$	$B \times (B - 1) \times (B - 2) \dots$ $2 \times 1$ , for $0 = 1   B$ Undefined for $(0 = 1   B) \wedge (-1 = \times B)$ , $\Gamma B + 1$ , otherwise	$! 4 \leftrightarrow 24$
? $B$	M	?	Random selection from the first $B$ positive integers		
$\otimes B$	M	$\otimes$	Natural logarithm of $B$	$\ln B$	$\otimes 2 \leftrightarrow 0.693147 \dots$
$A \otimes B$	D	$\otimes$	Common logarithm of $B$	$\log_A B$ or $(\otimes B) \div \otimes A$	$10 \otimes 2 \leftrightarrow 0.301029$
$\circ B$	M	$\circ$	Pi times $B$	$\pi \times B$	$\circ 1 \leftrightarrow 3.14159$
$A \circ B$	D	$\circ$	Circular functions	$A$ $A \circ B$ $-7$ arctanh $B$ $-6$ arccosh $B$ $-5$ arcsinh $B$ $-4$ $(-1 + B * 2) * .5$ $-3$ arctan $B$ $-1$ arcsin $B$ $0$ $(1 - B * 2) * .5$ $1$ sine $B$ $1$ cos $B$ $3$ tan $B$ $4$ $(1 + B * 2) * .5$ $5$ sinh $B$ $6$ cosh $B$ $7$ tanh $B$	

**VECTORS AND VECTOR FUNCTIONS**

The use of vectors (or one-dimensional arrays) is an integral part of the use of arrays in programming. A means of entering a constant vector was given earlier; that is,

$\square \leftarrow P \leftarrow -7 \ 3 \ 9 \ 6 \ 5$

-7    3    9    6    5

In this case, the dimension of  $P$  is saved automatically by the APL system and the user need not be concerned with it. Each element of an array is assigned an index that can be used to select that particular element. (For example, the third element of  $P$  has the value 9.)

### Extension of Scalar Functions to Vectors

Scalar functions are extended to arrays on an element-by-element basis. Thus,

			1	2	3	$\times$	4	5	6
4	10	18							
			1	2	3	$+$	20		
21	22	23							

The following forms are permitted (where  $\circ$  is a scalar function):

- vector  $\circ$  vector  $\rightarrow$  vector
- scalar  $\circ$  vector  $\rightarrow$  vector
- vector  $\circ$  scalar  $\rightarrow$  vector

If both arguments are vectors, then each must be the same size. If one argument is vector, then the scalar argument is extended to apply to all arguments of the vectors. For example,

1 2 3 \* 2

is equivalent to

1 2 3 \* 2 2 2

In many cases, however, the size of a vector is not known explicitly, although a function is available for computing it, and the first method is preferred.

All scalar functions, primitive or defined, are extended to arrays as demonstrated in the following example.

			!	1	2	3
1	2	6				
			-7	3	9	$\lceil$ 6 5 1
6	5	9				

### Indexing

Indexing (or subscripting as it is commonly called) is specified by enclosing the index in brackets following the array. That is, for example,

$P \leftarrow -7 3 9 6 5$   
 $P[4] + 2$

7                   (1 2 3 + 4 5 6)[2]

$P[2] \leftarrow 50$

$P$

-7 50 9 6 5

Indexing applies to the argument on its immediate left, regardless if it is an array variable, a constant vector, or an array expression in parentheses. An index may be any APL expression. In fact, an index may be an array and, in this case, an array of elements is selected rather than a single element. Thus,

```
P ← -7 3 9 6 5 1 4 3
P[2 4 5 7]
3 6 5 4
P[1 3] ← 10 20
P
10 3 20 6 5 1 4 3
```

Indexing can be regarded as a dyadic function; one argument is the array and the other is the index. The characteristics of the result are determined by the index.

APL allows zero- or one-origin indexing. One-origin indexing is assumed by default such that the indices of the elements of a vector with  $N$  elements are 1, 2, ...,  $N$ . The *ORIGIN* system command of the form

)ORIGIN 0

or

)ORIGIN 1

is used to change the index origin. Using zero-origin indexing, the indices of the elements of a vector with  $N$  elements are 0, 1, ...,  $N - 1$ . Zero-origin indexing applies to  $n$ -dimensional arrays as well as to vectors.

### Vector Generation

The monadic iota function, called the *index generator*, is written

$\iota N$

and generates a vector of length  $N$  that contains the positive integers 1 through  $N$  as elements. The expression  $\iota 0$  generates a null vector that prints as a blank line.

The *reshape function* uses the dyadic rho function symbol ( $\rho$ ) and takes the form

$M \rho N$

When  $M$  is a scalar value, a vector is generated. In general,  $M \rho N$  creates a vector of length  $M$  using argument  $N$ . If  $N$  is a scalar, it is repeated  $M$  times in the generated vector. If  $N$  is a vector and its length is less than  $M$ , then it is used cyclically. If the length of  $N$  is greater than  $M$ , then only the first  $M$  elements of  $N$  are used.

The use of the index generator and the reshape function are depicted in the following example.

```

      i3
1   2   3
1   2   3   4   5   6   7   8
      □ ← W ← i5
1   2   3   4   5
      □ ← V ← 4ρ -7  3  9  6  5
-7   3   9   6
      5ρ1
1   1   1   1   1
      7ρi2
1   2   1   2   1   2   1

```

### Size of a Vector

The use of the rho symbol ( $\rho$ ) as a monadic function gives the size of an array. Thus  $\rho V$  gives the number of elements in  $V$ . If  $N$  is a scalar value, then  $\rho N$  generates a null vector. For example,

```

V ← -7  3  9  6  5
ρV
5
ρi6
6
ρW ← 100ρ2
100

```

The monadic  $\rho$  function always generates a vector result. The number of elements in the result is equal to the number of dimensions of the argument.

### The Ravel Function and Catenation

The comma (,) used as a monadic function is a means of creating a vector from a scalar or an array of higher dimension. It is called the *ravel function*. Thus if  $\emptyset$  denotes a blank line, then

```

N ← 5
ρN
∅
V ← ,5
ρV
1

```

Applied to an array of higher dimension, the ravel function chains the elements by row to form a vector. Thus, if

$$A = \begin{pmatrix} -7 & 3 & 9 \\ 6 & 5 & 1 \\ 4 & 3 & 2 \end{pmatrix}$$

then  $,A$  generates the vector  $-7 \ 3 \ 9 \ 6 \ 5 \ 1 \ 4 \ 3 \ 2$ .

Two arguments are *catenated* by using the comma (,) as a dyadic function. Catenation chains scalars or vectors together in the usual fashion as demonstrated in the following example.

```

A ← 13 47
□ ← B ← A,13
13   47   1   2   3
                  (12),A
1     2     13   47
                  1,2,13
1     2     1     2   3   4   5

```

### Reduction

*Reduction* is the application of the scalar dyadic functions to the elements of the same array. Vector reduction is defined as

$$\oplus/V \leftrightarrow V[1] \oplus \dots \oplus V[\rho V]$$

where  $V$  is a vector and  $\oplus$  is a scalar dyadic function. Reduction is performed as though the equivalent expression were executed on a right-to-left basis. For example,

```

V ← 13
+/V
6
□ ← P ← ^/0 0 0 1
0
∨/0 0 0 1
1
>/6 2 8
1

```

The latter function is equivalent to  $6 > 2 > 8$  which uses the logical result 0 of the function  $2 > 8$  as a numeric value. Reduction is a powerful facility. As an example, the familiar “dot” product of two vectors  $U$  and  $V$  can be expressed in APL as  $+/U \times V$ .

### Reversal and Rotation

Reversal and rotation use the symbol  $\phi$  formed by overstriking the circle symbol ( $\circ$ ) and the vertical stroke ( $|$ ). The *reversal function* uses the monadic form of the function symbol and reverses the elements of its argument. For example,  $\phi\iota 3$  generates the vector  $3 \ 2 \ 1$ . The result of the reversal function applied to the vector  $V$  is  $V[1 + (\rho V) - \iota\rho V]$ .

The dyadic form of  $\phi$  is called *rotation* and is used to rotate a vector cyclically. If  $V$  is a vector and  $K$  is a scalar, then  $K \phi V$  rotates  $V$  to the left by  $(\rho V) | K$  elements. If  $K$  is positive, rotation is to the left; if  $K$  is negative, rotation is to the right. If  $V \leftarrow \iota 7$ , then  $3 \phi V$  is equal to the vector  $4 \ 5 \ 6 \ 7 \ 1 \ 2 \ 3$ . The result of the rotation function  $K \phi V$  is defined as  $V[1 + (\rho V) | -1 + K + \iota\rho V]$ . Reversal and rotation are demonstrated in the following script.

```

    □ ← A ← φ i7
7   6   5   4   3   2   1
      2 φ A
5   4   3   2   1   7   6
      -3 φ A
3   2   1   7   6   5   4

```

### Compression and Expansion

The *compression function* has the form

$$U/V$$

where  $U$  is a logical vector that has the same length as  $V$ , the vector being compressed. If either argument is a scalar, it is extended to apply to all elements of the other argument. The result of compression is generated as follows. The elements of  $V$  corresponding to a 1 in  $U$  are retained while the elements of  $V$  corresponding to a 0 in  $U$  are suppressed. For example,

```

    U ← 1   0   0   1
    □ ← R ← U/\4
1   4
          U/\' ABCD '
AD

```

If  $R$  is the result of  $U/V$ , then  $(\rho R) = +/U$ .

*Expansion* is the converse of compression and is written

$$U\setminus V$$

where  $U$  is a logical vector and  $(+/U) = \rho V$ .  $U\setminus V$  expands  $V$  by inserting padding

for elements that correspond to zero elements of  $U$ . If  $V$  is numeric, it is padded with zeros. If  $V$  is a character vector, it is padded with spaces. For example,

				$U \leftarrow 1 \ 0 \ 1 \ 0$
				$\square \leftarrow U \setminus AB \setminus$
$A$	$B$			$U \setminus U / t4$
1	0	3	0	

### Take and Drop Functions

The *take function* is written

$$T \uparrow V$$

where  $T$  is a scalar and  $V$  is a vector. If  $T$  is positive, the first  $T$  elements of  $V$  are selected. If  $T$  is negative, the last  $|T|$  elements of  $V$  are selected. The *drop function*, written as

$$T \downarrow V$$

is defined analogously. If  $T$  is positive, the first  $T$  elements of  $V$  are dropped and if  $T$  is negative, the last  $|T|$  elements of  $V$  are dropped. The use of both functions is demonstrated in the following example.

									$A \leftarrow -7 \ 3 \ 9 \ 6 \ 5 \ 1 \ 4 \ 3$
									$3 \uparrow A$
-7	3	9							$-2 \uparrow A$
4	3								$6 \downarrow A$
4	3								$-5 \downarrow A$
-7	3	9							

### Set Operations

The *index of function* is defined as

$$V \iota S$$

and gives the index of the leftmost occurrence of scalar  $S$  in vector  $V$ . The right argument is extended to arrays. Thus, if  $F \leftarrow -7 \ 3 \ 9 \ 6 \ 5$  and  $I \leftarrow 6$ , then  $F \iota I$  produces the value 4, which is the index of the value 6 in the vector  $F$ . If  $S$  is not found in  $V$ , then the function produces the result  $1 + \lceil / \rho V$ .

The *membership function* is written

$$A \epsilon B$$

and produces a logical result that is the same size as  $A$ . If an element of  $A$  is an

element of  $B$ , then the corresponding element of the result is 1; otherwise, the corresponding element in the result is zero.

The use of both functions is depicted in the following example.

	$V \leftarrow -7 \ 3 \ 9 \ 6 \ 5 \ 1 \ 4 \ 3$
	$\rho V$
8	
	$V u 4$
6	9    2    7
	$(\iota 4) e V$
1	0    1    1

### Random Generation

The *deal* function is written

$A ? B$

and generates a vector of  $A$  elements randomly from  $tB$  without replacement.  $A$  and  $B$  are both scalars. For example,

	3 ? 5
	5    1    2

### Grade Up and Down Functions

The *grade up* function, written as  $\Delta V$ , uses the monadic version of the symbol  $\Delta$  (formed by overstriking a  $\Delta$  and a vertical stroke |) and yields a vector of indices that would order the vector right argument in ascending order. Similarly, the *grade down* function, written

$\nabla V$

(where  $\nabla$  is the composite symbol formed from  $\nabla$  and |) produces the vector of indices that would order the vector right argument in descending order. For example,

	$V \leftarrow 9 \ -7 \ 3$
	$\Delta V$
2    3    1	
	$V[\Delta V]$
-7    3    9	
	$V[\nabla V]$
9    3    -7	

### Decode and Encode Functions

The *decode* function written

$R \perp A$

produces the base ten value of the coefficients  $A$  to the radix  $R$ . If  $R$  is a scalar, then it is extended to all elements of  $A$ . Otherwise,  $\rho R$  must equal  $\rho A$ . The *decode* function, written

$$R \top S$$

reverses the process and yields the vector of coefficients of the radix  $R$  that are equivalent to the scalar value  $S$ . The dimension of the result is the dimension of  $R$ . For example,

```
V ← 13
10 ⊥ V
123
      2 ⊥ 101
5
      24 60 60 ⊥ 2 1 3
7263
      2 2 2 ⊥ 5
1   0   1
      10 10 10 ⊥ 123
1   2   3
```

### Character Vectors

A *character vector* is treated as any other vector except that it is not in the domain of numeric functions. For example,

```
C ← 'TEA FOR TWO'
ρC
11
      C[5 6 7]
FOR
      A ← 'NEW'
      B ← 'YORK'
      A,B
NEWYORK
      A, ' ', B
NEW YORK
```

Character input and output is specified with the quote-quad symbol ( $\Box$ ) formed by overstriking the quad ( $\Box$ ) and the quote ('') symbols. For character input, the quote-quad function is executed in a manner similar to the quad function except that no prompting character is printed for the user and the carriage is not indented. In

response to the input quote-quad function, the user types his data without enclosing quote symbols, as shown in the following example.

```
POP ← █
TEA FOR TWO
    POP
TEA FOR TWO
    ρPOP
11
```

For output, the quote-quad symbol is placed to the left of the specification function in the usual fashion, that is,

```
D ← ' NEW '
█ ← E ← D, ' JERSEY '
```

*NEW JERSEY*

An array can contain all numeric values or all character values as elements. Therefore, an output request of the form

```
A ← 1 + B ← 2 + C ← 2
      A,B,C
5   4   2
```

forms a vector and then prints it. For output, arguments can be mixed by using the semicolon, as shown below.

```
VAL ← 50.43
'THE ANSWER IS' ; VAL
```

*THE ANSWER IS 50.43*

Most input and output functions are designed for use in defined functions which are covered in the next section.

## DEFINED FUNCTIONS

Defined functions are the means by which programs are written and user-defined functions are added to the language. Defined functions are stored in a user's workspace and can be moved from one workspace to another using system commands. Defined functions can utilize zero, one, or two arguments as follows:

Name	Number of arguments	Syntax
Niladic	0	f
Monadic	1	fB
Dyadic	2	AfB

A defined function can return an implicit result or an explicit result. Only functions that return an explicit result can be used in an expression. All primitive functions return an explicit result. The arguments and result of a function may be a scalar value or an array.

The system enters the definition mode when a del ( $\nabla$ ) is received from the user's terminal and leaves the definition mode when the next del is received.

### Syntax of Defined Functions

The function header determines the characteristics of a defined function. The six possible forms of the function header statement are

Arguments	Explicit result	Implicit result
0	$\nabla R \leftarrow FCN$	$\nabla FCN$
1	$\nabla R \leftarrow FCN Y$	$\nabla FCN Y$
2	$\nabla R \leftarrow X FCN Y$	$\nabla X FCN Y$

where  $FCN$  is the name of the defined function,  $X$  and  $Y$  are dummy arguments, and  $R$  is the dummy result. The following script depicts a dyadic function with an explicit result (*INTERS*), a monadic function with an explicit result (*SORT*) where both argument and result can be a vector, niladic functions with an explicit result (*SQR2* and *DEQUEUE*), and a monadic function with an implicit result (*QUEUE*):

```

       $\nabla R \leftarrow U \text{ INTERS } V$ 
[1]    $R \leftarrow (U \varepsilon V) / U$ 
[2]    $\nabla$ 
      (i5) INTERS 1 3
1     3
      'PAT' INTERS 'STOP'
PT
       $\nabla D \leftarrow SORT L$ 
[1]    $D \leftarrow L[\Delta L \leftarrow, L]$ 
[2]    $\nabla$ 
      SORT -7 3 9 6
-7   3   6   9
      SORT 5
5
       $\nabla R \leftarrow SQR2$ 
[1]    $R \leftarrow 1.41421$ 
[2]    $\nabla$ 
       $2 \times SQR2$ 
2.82842
       $\nabla Q \leftarrow Q, A$ 
[1]    $Q \leftarrow Q, A$ 
[2]    $\nabla$ 
       $\nabla R \leftarrow DEQUEUE$ 

```

```

[1]      R ← 1 ↑ Q
[2]      Q ← 1 ↓ Q
[3]      ∇
          Q ← i0
          QUEUE 45
          QUEUE 93
          QUEUE -5
          DEQUEUE
45
          QUEUE 123
          DEQUEUE
93

```

### Sequence and Control

Statements are executed sequentially in a defined function until control flows out of the function or a branch statement is encountered. A *branch statement* has the form

 $\rightarrow S$ 

where  $S$  is an expression. The APL system transfers program control to the statement numbered  $1 \uparrow S$ . (If  $S$  is a scalar, then  $1 \uparrow S$  is equivalent to  $S$ .) If  $S$  is equal to zero or is outside of the range of statement numbers in the function, then a normal exit is made from the function. If  $1 \uparrow S$  is null, then the next sequential statement is executed. If  $1 \uparrow S$  is the number of a statement in the function being executed, then program control is passed to that statement.

Each of the following branch statements is equivalent and causes a branch to statement numbered  $S$  if the logical value of the comparison expression  $X \text{r} Y$  is true:

$$\begin{aligned}\rightarrow(X \text{r} Y)/S \\ \rightarrow(X \text{r} Y)\rho S \\ \rightarrow S \times \iota X \text{r} Y\end{aligned}$$

Similarly, the following equivalent statements cause a branch to statement numbered  $S1$  if  $X \text{r} Y$  is true and to statement numbered  $S2$ , otherwise,

$$\begin{aligned}\rightarrow(S1,S2)[1 + X \text{r} Y] \\ \rightarrow((X \text{r} Y),\sim X \text{r} Y)/S1,S2\end{aligned}$$

The following problem demonstrates the concept of branching by producing a table of values.

```

VTABLE N
[1]      →(N > 0) × 2
[2]      I ← 0
[3]      I ← I + 1
[4]      I,(I * 2),(I * 3)

```

[5]  $\rightarrow(I < N)/3$

[6]  $\nabla$

TABLE 3

1 1 1

2 4 8

3 9 27

### Statement Labels and Local and Global Variables

A *statement label* is a scalar variable that has the value of the statement number to which it is assigned. A statement is given a label by preceding the body of the statement with a name and separating the two with a colon as follows:

HERE:  $A \leftarrow i10$

$\rightarrow$  HERE

Statement labels are local variables in APL and are useful when statement numbers are rearranged as a result of function editing.

All variables are *global* in APL unless specified to the contrary. A *local variable* is one that retains its value only within the function in which it is declared. A local variable dominates a global variable with the same name when the function in which it is declared is active. Local variables are declared in the function header; they are listed after the function prototype and must be preceded by a semicolon. For example,

$\nabla RES \leftarrow ABC\ W;I;J;K$

In this case,  $I$ ,  $J$ , and  $K$  are declared as local variables. The following program, which smooths a list of values, depicts the use of statement labels and local variables.

```

 $\nabla S \leftarrow SMOOTH\ X;I;L$ 
[1]  $\rightarrow((\rho X) > 3)/INIT$ 
[2]  $\rightarrow 0,\rho \square \leftarrow 'DATA\ ERROR'$ 
[3] INIT:  $I \leftarrow 1$ 
[4]  $L \leftarrow (\rho X) - 1$ 
[5]  $S \leftarrow i0$ 
[6] LOOP:  $I \leftarrow I + 1$ 
[7]  $\rightarrow(I > L)/0$ 
[8]  $S \leftarrow S, (X[I - 1] + X[I] + X[I + 1]) \div 3$ 
[9]  $\rightarrow LOOP$ 
[10]  $\nabla$ 
SMOOTH 1 3 5 7 9
3 5 7

```

### Recursive Functions

A recursive function is one that calls itself in the body of its definition. The factorial function, defined as  $n! = n \times (n - 1)!$  where  $0! = 1$ , is a common example. The factorial function, written in APL, is given as follows.

```

       $\nabla R \leftarrow FACT\ N$ 
[1]       $\rightarrow (N = 0)/4$ 
[2]       $R \leftarrow N \times FACT\ N - 1$ 
[3]       $\rightarrow 0$ 
[4]       $R \leftarrow 1$ 
[5]       $\nabla$ 
      FACT 4
24

```

(This program serves only as an example since APL includes a primitive function for computing the factorial.)

### Other Facilities

The APL terminal system contains a variety of other facilities for use with defined functions. Both function editing and line editing are permitted and program checkout is eased with a stop control and a trace control function. *Stop control* is initiated with a statement of the form

 $S\Delta FCN \leftarrow V$ 

where  $FCN$  is the function name and  $V$  is a vector of statement numbers. Execution of the function is halted *just before* the execution of each statement whose statement number is in  $V$ . *Trace control* is initiated with a statement of the form

 $T\Delta FCN \leftarrow V$ 

where, again,  $FCN$  is the function name and  $V$  is a vector of statement numbers. Execution of the function is traced by printing control information for each statement whose statement number is in  $V$ . Stop control and trace control are discontinued with statements of the form

 $S\Delta FCN \leftarrow \iota 0$ 

and

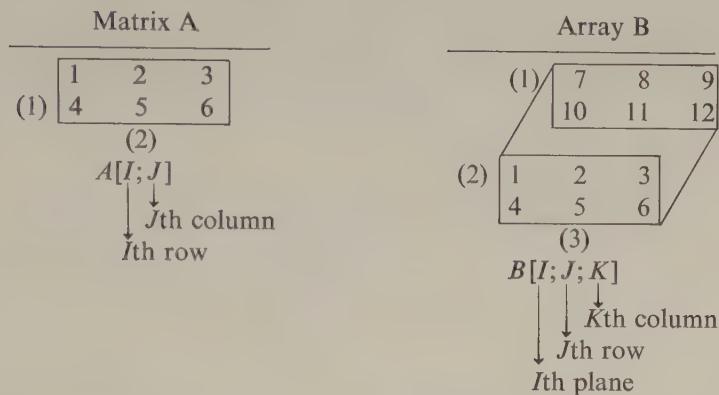
 $T\Delta FCN \leftarrow \iota 0$ 

respectively.

## MATRICES AND ARRAYS OF HIGHER DIMENSION

Functions defined previously on vectors are extended systematically to  $n$ -dimensional arrays. Several functions apply along a given coordinate of an array. Each coordinate

is numbered according to its subscript position, as follows (the number in parentheses denotes the coordinate number).



Arrays are synthesized on a right-to-left basis using vectors, matrices, three-dimensional arrays, etc.

Arrays are formed or raveled in row order, which is referred to as *index order* in APL. Array functions are introduced here and summarized in Table 8.

### Basic Array Functions

An array is generated with the dyadic reshape function of the form  $M\rho N$ , where  $M$  is a vector and  $N$  is a scalar or an array. The dimension of each coordinate of the generated array is given by the respective element of  $M$ . Thus,  $3\ 4\rho 1$  generates a matrix with 3 rows and 4 columns where each element is the value 1. As with vectors,  $N$  is used cyclically. Similarly,  $2\ 3\rho 1\ 2$  generates the matrix

$$\begin{array}{ccc} 1 & 2 & 1 \\ 2 & 1 & 2 \end{array}$$

The size of an array is given with the monadic rho function. Thus, if array  $A$  has 4 rows and 3 columns, then  $\rho A$  generates the vector  $4\ 3$ .  $\rho\rho A$  gives the number of coordinates in an array  $A$  and is referred to as its *rank*.

Indexing is performed with a list of the form

$$A[I; J; \dots]$$

where  $I; J; \dots$  denotes the elements to be selected along the respective coordinate. The semicolon is used as a separator since each index can be a scalar to an array or can be elided. If an index is elided, then the entire coordinate is selected. Thus if  $B \leftarrow 4\ 3\rho 12$ , that is,

$$B \leftrightarrow \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{pmatrix}$$

**TABLE 8**

Mixed Functions in APL

Name	Function	Definition or example*
Index		
generator	$\iota B$	$\iota 5 \leftrightarrow 1 2 3 4 5$
Index of	$A\iota B$	$T\iota 5 \leftrightarrow 3$
Size	$\rho B$	$\rho T \leftrightarrow 4$
Reshape	$A\rho B$	$\rho M \leftrightarrow 2 3$
Reversal	$\phi B$	$\phi T \leftrightarrow 7 5 3 2$
		$\phi M \leftrightarrow 3 2 1$
		6 5 4
		$\phi[1]N \leftrightarrow DEF$
		<i>ABC</i>
Rotation	$A\phi B$	$2\phi T \leftrightarrow 5 7 2 3$
		$-1 2\phi M \leftrightarrow 3 1 2$
		6 4 5
		$1 0 1\phi[1]N \leftrightarrow DBF$
		<i>AEC</i>
Ravel	$,B$	$,M \leftrightarrow 1 2 3 4 5 6$
Catenation	$A,B$	$T,2 \leftrightarrow 2 3 5 7 1 2$
		$M,M \leftrightarrow 1 2 3 1 2 3$
		4 5 6 4 5 6
		$N,[1]^{***} \leftrightarrow ABC$
		<i>DEF</i>
		***
Lamination	$A,[I]B$	If $0 < I < 1$ , then $R[1;;] \leftarrow A$ and $R[2;;] \leftarrow B$
		If $1 < I < 2$ , then $R[1;;] \leftarrow A$ and $R[2;;] \leftarrow B$ , etc.
Take	$A \uparrow B$	$2 \uparrow T \leftrightarrow 2 3$
		$-1 \uparrow T \leftrightarrow 7 2$
		$-2 \uparrow M \leftrightarrow 2 3$
		5 6
Drop	$A \downarrow B$	$1 \downarrow T \leftrightarrow 3 5 7$
Compression	$U/B$	$-2 \downarrow T \leftrightarrow 2 3$
		$-1 \downarrow M \leftrightarrow 2 3$
		4 6
Expansion	$U \backslash B$	$1 0 1 1 \backslash 3 \leftrightarrow 1 0 2 3$
		$1 0 1 \backslash [1]N \leftrightarrow ABC$
		<i>DEF</i>
Transposition	$\mathbb{Q}B$	$\mathbb{Q}M \leftrightarrow 1 4 2 1 \mathbb{Q}M \leftrightarrow 1 4$
		2 5 2 5
	$A \mathbb{Q}B$	$3 6 3 6$
		1 1 $\mathbb{Q}M \leftrightarrow 1 5$
Membership	$A\varepsilon B$	$(i4)\varepsilon T \leftrightarrow 0 1 1 0$
		$M\varepsilon T \leftrightarrow 0 1 1$
		0 1 0
Deal	$A?B$	Select $A$ elements of $B$ at random without replacement
Decode	$A \perp B$	$2 \perp 1 0 1 \leftrightarrow 5$
Encode	$A \top B$	$2 2 2 \top 5 \leftrightarrow 1 0 1$

*Notes.*

1. Arrays used in examples:

$$\begin{array}{ccccccc}
 T \leftrightarrow 2 & 3 & 5 & 7 & N \leftrightarrow ABC \\
 & & & & & & \\
 & & & & & & \\
 & & & & & & \\
 M \leftrightarrow 1 & 2 & 3 & & & & \\
 & 4 & 5 & 6 & & & 
 \end{array}$$

2. A mixed function is applied along the last coordinate of an array. If  $[S]$  appears after any of the function symbols, the relevant coordinate is determined by the scalar  $S$ .

3. Two extensions of the APL set of functions have not been covered: matrix division (that uses the domino symbol  $\top$ ) and the extension of decode and encode to apply to arrays. These functions are discussed separately in the section entitled Recent Advances in APL.

then  $B[3;2]$  is the element 8 and  $B[2\ 4;1\ 3]$  is the matrix

$$\begin{matrix} 4 & 6 \\ 10 & 12 \end{matrix}$$

Similarly,  $B[;2]$  is the vector 2 5 8 11 and  $B[3;]$  is the vector 7 8 9.

The *monadic ravel* function, written

$$,A$$

generates a vector of the elements of  $A$  taken in index order. Thus, if

$M \leftarrow 2\ 3\rho\ -7\ 3\ 9\ 6\ 5\ 1$ , then  $,M$  yields the vector -7 3 9 6 5 1.

If an array is used as the right argument in the reshape function, i.e.,

$$M\rho A$$

then  $A$  is raveled before the reshape function is performed. Thus if,

$M \leftarrow 2\ 3\rho\ -7\ 3\ 9\ 6\ 5\ 1$ , then  $3\ 2\rho M$  yields the matrix

$$\begin{matrix} -7 & 3 \\ 9 & 6 \\ 5 & 1 \end{matrix}$$

As with vectors, scalar functions are applied to arrays on an element-by-element basis. Thus if  $A \leftarrow 2\ 3\rho 6$  and  $B \leftarrow 2\ 3\rho 2$ , then  $A * B$  yields the matrix

$$\begin{matrix} 1 & 4 & 9 \\ 16 & 25 & 36 \end{matrix}$$

Similarly if  $N \leftarrow 3\ 2\rho\ -7\ 4\ 3\ -16\ -1\ 0$ , that is,

$$\begin{matrix} -7 & 4 \\ 3 & -16 \\ -1 & 0 \end{matrix}$$

then  $|N$  yields the matrix

$$\begin{matrix} 7 & 4 \\ 3 & 16 \\ 1 & 0 \end{matrix}$$

### Composite Functions

A *composite function* in APL is an extension of the primitive scalar dyadic function to arrays. Reduction, mentioned earlier, falls into this category. Two other composite functions are the inner product and outer product.

*Reduction* is applied along the  $I$ th coordinate of an array with an expression of the form

$$\oplus/[I]A$$

where  $\oplus$  is a primitive dyadic function and  $A$  is an array. If  $[I]$  is elided, the last coordinate is assumed. Thus if  $M \leftarrow 2\ 3\ \rho\!t\!6$ , then  $+/[1]M$  yields the vector  $5\ 7\ 9$  and  $+/[2]M$  generates the vector  $6\ 15$ . If  $N \leftarrow 2\ 2\ 3\ \rho\!t\!12$ , that is,

$$\begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{matrix}$$

$$\begin{matrix} 7 & 8 & 9 \\ 10 & 11 & 12 \end{matrix}$$

then  $\times/[1]N$  yields the matrix

$$\begin{matrix} 7 & 16 & 27 \\ 40 & 55 & 72 \end{matrix}$$

The *inner product* is written

$$Af.gB$$

where  $A$  and  $B$  are arrays and  $f$  and  $g$  are primitive scalar dyadic functions; the inner product function is defined as follows.

Type of arguments	Definition ( $R =$ result)
Vector	$R = f/AgB$
Vector and matrix	$R[I] = f/AgB[;I]$
Matrix and vector	$R[I] = f/A[I;]gB$
Matrix and matrix	$R[I;J] = f/A[I;]gB[;J]$

The ordinary matrix product is expressed as  $A +.\times B$ .

The *outer product* is written

$$A \circ .fB$$

where  $A$  and  $B$  are arrays and  $f$  is a primitive scalar dyadic function; the outer product function is defined as

Type of arguments	Definition ( $R =$ result)
Vector	$R[I;J] = A[I]fB[J]$
Vector and matrix	$R[I;J;K] = A[I]fB[J;K]$
Matrix and vector	$R[I;J;K] = A[I;J]fB[K]$
Matrix and matrix	$R[I;J;K;L] = A[I;J]fB[K;L]$

Outer product resembles the familiar Cartesian product. If  $A \leftarrow 1\ 2\ 3$  and  $B \leftarrow 6\ 8\ 10$ , then  $A \circ .+ B$  yields the matrix

$$\begin{matrix} 7 & 9 & 11 \\ 8 & 10 & 12 \\ 9 & 11 & 13 \end{matrix}$$

### Mixed Functions

Mixed functions are those functions that cannot be classed as primitive scalar functions or composite functions. Most of the basic array functions, such as the reshape, ravel, and the size functions fall into this category.

As with reduction, the reversal, rotation, compression, and expansion functions apply to  $n$ -dimensional arrays along a specified coordinate. These functions are not redefined but their application to arrays is demonstrated in Table 8, along with the other mixed functions. The monadic and dyadic transpose functions are covered here since they have not been mentioned previously.

The *monadic transpose* function, written

$\mathbb{Q}A$

interchanges the last two coordinates of array  $A$ . Thus, if  $M \leftarrow 2\ 3\rho 6$ , then  $\mathbb{Q}M$  generates the matrix

1	4
2	5
3	6

The *dyadic transpose* function is written

$V\mathbb{Q}A$

where  $V$  is a vector and  $A$  is an array.  $\rho V$  must equal  $\rho\rho A$  and the  $V[I]$ th coordinate of the result is the  $I$ th coordinate of  $A$ . Thus, if  $M \leftarrow 3\ 4\rho 12$ , then  $2\ 1\mathbb{Q}M$  yields the matrix

1	5	9
2	6	10
3	7	11
4	8	12

and if  $A \leftarrow 2\ 3\ 4\rho 24$ , then  $3\ 1\ 2\mathbb{Q}A$  yields

1	13
2	14
3	15
4	16

5	17
6	18
7	19
8	20

9	21
10	22
11	23
12	24

The functions presented here represent a level of the APL language that is equivalent to its original definition. Recent advances have been made and they are covered in an appropriate section.

## WORKSPACE MANAGEMENT

Although most system commands reflect the operating environment and thus the method of implementation, the concept and use of a workspace is an integral part of using the APL terminal system. System commands related to workspace management are covered briefly.

### Libraries

Two kinds of libraries exist in an APL system: public libraries and private user libraries. Public libraries are assigned an appropriate number and workspaces, functions, and variables can be retrieved from a public library by giving the library number and the name of the object to be retrieved.

Private libraries are assigned to users and each is identified by a user's identification number. When accessing his own library, a user need not give his own identification number.

A user can list the names of the workspaces in his library with the command

)LIB

and in a public library or another user's to which he is given access with the command

)LIB *number*

where "number" is the identification number of the library to be accessed.

### The Continue Workspace

Each user is given a workspace in his library named *CONTINUE*. If an unusual condition forces a terminal session to be discontinued, the active workspace is saved automatically in *CONTINUE*. When the user signs on the next time, the *CONTINUE* workspace is loaded automatically. Three commands allow the user to explicitly store a workspace in *CONTINUE*:

)SAVE *CONTINUE*  
)*CONTINUE*  
)*CONTINUE HOLD*

The )*CONTINUE HOLD* command saves the active workspace in *CONTINUE* and discontinues a terminal session without releasing a telephone line connection.

## Loading, Saving, and Dropping Workspaces

A workspace can be saved in a user's private library with the command

*)SAVE name*

or in public library *n* with the command

*)SAVE n name*

A saved workspace can be loaded as an active workspace with one of the following commands.

Library	Form of <i>LOAD</i> command
Active user's	<i>)LOAD name</i>
Public	<i>)LOAD n name</i>
Another user's	<i>)LOAD id name</i>

A workspace can be deleted from a private library with the command

*)DROP name*

and from a public library with the command

*)DROP n name*

In the latter case, a user can only drop a workspace from a public library if he owns it, that is, he saved it there.

## The *COPY* Command

The *COPY* command allows a user to copy an object (function, variable, group, or all functions and variables) from a saved workspace to the active workspace. The *COPY* command can assume the following forms.

*)COPY name*  
*)COPY name object*  
*)COPY n name*  
*)COPY n name object*  
*)COPY id name*  
*)COPY id name object*

When an object is not specified in the *COPY* commands, all functions, variables, and group are copied while control information is not.

## Group

A *group* is a collection of names that refer to functions, variables, or other groups. A group is formed with the *GROUP* command:

*)GROUP name list*

where “name” is the name of the group and “list” is a list of names separated by at least one space. Thus, if

)GROUP SETLIST SORT FLAG21 PERT

then

)COPY TESTSPACE SETLIST

is equivalent to

)COPY TESTSPACE SORT  
)COPY TESTSPACE FLAG21  
)COPY TESTSPACE PERT

### Other System Commands

Most implementations of APL provide additional system commands that generally fall into categories: terminal control, workspace control, inquiry, and communications. The concept of using system commands has permitted the form of the APL language to remain “pure” of operational considerations.

## RECENT ADVANCES IN APL

One of the design objectives of APL is that new language concepts do not invalidate old programs and require that the user “learn a new language.” This is possible because APL is based on rigorous mathematical notation, a few data types, and a multiplicity of primitive functions.

Several advances have been made to APL since the original concept: matrix division, catenation and lamination, the extension of encode and decode to arrays, adjustable “fuzz”, and data files. APL is lacking in the capability for handling data files and the material presented here on that subject is not a standard feature but is representative of several existing advances to the language in that area.

### Matrix Division

Matrix division is a primitive function that uses the domino symbol ( $\boxdot$ ), formed by overstriking the quad symbol ( $\square$ ) and the division sign ( $\div$ ). As a dyadic function, matrix division is written

$$X \leftarrow B \boxdot A$$

where the following conditions must hold:

$$\begin{aligned} 2 &\leftrightarrow \rho\rho A \\ (1 \uparrow \rho A) &\geq 1 \downarrow \rho A \\ \vee/1 \quad 2 &= \rho\rho B \end{aligned}$$

and

$$(1 \uparrow \rho A) \leftrightarrow 1 \uparrow \rho B$$

The result  $X$  of the function is the least squares solution to the system of simultaneous linear equations

$$(A +. \times X) = B$$

$X$  is computed such that the expression

$$R \leftarrow +/(B - A +. \times X) * 2$$

is minimized and  $\rho X \leftrightarrow N, 1 \downarrow \rho B$ , where  $\rho A \leftrightarrow M, N$ .

If  $A$  is a nonsingular square matrix, then  $R$  is zero and  $X$  is the solution to the set of equations. If  $M > N$ , then the system of equations is overdetermined and  $R$  is minimized.

As a monadic function of the form

$$X \leftarrow \boxdot A$$

the matrix divide function is equivalent to

$$X \leftarrow I \boxdot A$$

where  $I$  is an identity matrix of order  $1 \uparrow \rho A$ . If  $A$  is nonsingular, then  $X$  is the left inverse of  $A$ . If  $A$  is a nonsquare matrix, then the solution  $X$  minimizes the expression

$$+/(I - A +. \times X) * 2$$

As an example of dyadic matrix division, consider the set of equations

$$x_1 + x_2 + x_3 = 2$$

$$2x_1 + x_2 + 2x_3 = 3$$

$$x_1 + 3x_2 + x_3 = 4$$

If

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 2 & 1 & 2 \\ 1 & 3 & 1 \end{pmatrix}, \quad X = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}, \quad \text{and} \quad B = \begin{pmatrix} 2 \\ 3 \\ 4 \end{pmatrix}$$

then the system of equations can be expressed as

$$AX = B$$

or in APL notation as

$$(A +. \times X) = B$$

A solution  $X$  is computed as follows.

$$\begin{aligned} A &\leftarrow 3 \ 3 \rho 1 \ 1 \ 1 \ 2 \ 1 \ 2 \ 1 \ 3 \ 1 \\ B &\leftarrow 2 \ 3 \ 4 \\ \square &\leftarrow X \leftarrow B \boxdot A \end{aligned}$$

2      1      -1

### Catenation and Lamination

*Catenation* of arrays along the  $K$ th coordinate, written

$A,[K]B$

where  $A$  and  $B$  are arrays, is permitted for arrays of the same rank and dimension, except along the  $K$ th coordinate. If the arrays have different ranks, then the coordinates in common must be of the same size. For example,

```
C← 2 3ρt6
D← 2 3ρ1
C,[1]D
```

1	2	3			
4	5	6			
1	1	1			
1	1	1			
		$C,[2]D$			
1	2	3	1	1	1
4	5	6	1	1	1

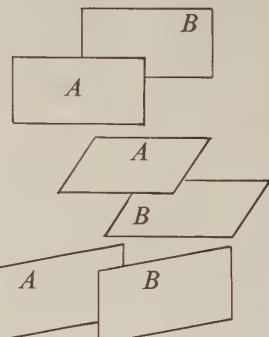
If  $[K]$  is elided, then the last coordinate is assumed.

*Lamination* joins two arguments along a new coordinate and is written as follows.

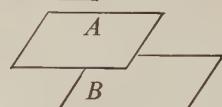
$A,[K]B$

The function inserts after the ( $\lceil / \rfloor [K]$ )th coordinate, a new coordinate with the range  $i2$ , are used to distinguish elements of  $A$  from elements of  $B$ . The process is depicted as follows.

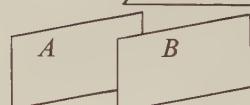
$A,[.5]B$                   as



$A,[1.5]B$                   as



$A,[2.5]B$                   as



For example,

```
A← 3 4ρt12
B← 3 4ρ1
□← C← A,[.5]B
```

1	2	3	4
5	6	7	8
9	10	11	12

1	1	1	1
1	1	1	1
1	1	1	1

$\rho C$

2	3	4
---	---	---

$\square \leftarrow D \leftarrow A,[1.5]B$

1	2	3	4
1	1	1	1

5	6	7	8
1	1	1	1

9	10	11	12
1	1	1	1

$\rho D$

3	2	4
---	---	---

Both catenation of arrays and lamination can be used with character arrays.

### Decode and Encode

The decode and encode functions are extended to apply to cases when the left and right arguments are arrays, other than the limited case presented previously. When the left argument is a vector (or an extended scalar) and the right argument is an array, written

$A \perp B$

then the result is the base value computation along the first coordinate of  $B$  using the radix vector  $A$ . If the left argument is a matrix, the computation is made using the rows of  $A$  as radix vectors. For example,

$B \leftarrow 2 \ 2 \rho \perp 4$

$10 \perp B$

13	24
8	14

$5 \perp B$

$C \leftarrow 2 \ 2 \rho 10 \ 10 \ 5 \ 5$

$C \perp B$

13	24
8	14

*Encode* is written

$A \top B$

When the left argument is a vector and the right argument is a vector, then each column of the result is the representation of the corresponding column of  $B$  using radix vector  $A$ . When the left argument is a matrix and the right argument is a vector, then the result  $R[;I;J]$  is the representation in base  $A[;I]$  of the component  $B[J]$ . For example,

	10	10	10	$\top$	13	24
0	0					
1	2					
3	4					
	5	5	5	$\top$	8	14
0	0					
1	2					
3	4					
	$A \leftarrow$	3	$2\rho$	10	5	
	$A \top$	13		24		
0	0					
0	0					
	1	2				
	2	4				
	3	4				
	3	4				

### Adjustable Fuzz

*Fuzz* is the very small value used to compensate for performing decimal arithmetic on a binary computer. If two values are within “fuzz” of each other, then they are regarded as being equal.

An APL function is defined to change the value of fuzz; it is written as follows.

*SETFUZZ N*

where  $0 \leq N \leq 31$ . The value of  $N$  denotes the number of rightmost bits to be disregarded when comparing two numeric values.

### Data Files

The use of data files is necessary for applications that work with more data than can be stored in a single workspace. The concept of a data file allows the user to store data on an external medium on a temporary or permanent basis. Most file

systems are proprietary extensions to APL and the following material is only representative of the various methods that exist.

A *file* is a collection of components, each of which can be scalar or array, of any type. A file is created with a function of the form

$$' \text{file name}' \quad FCREATE \quad \text{file-number}$$

and components (i.e., data items) are added to it with a function, such as

$$\text{value} \quad FAPPEND \quad \text{file-number}$$

Some file systems permit files to be constructed sequentially (as with a magnetic tape file) or directly by key (as with a file on a direct-access device). A file is read with a statement such as

$$\text{result} \leftarrow FREAD \quad \text{file-number, component-number}$$

The key point is that functions, such as *FCREATE*, *FAPPEND*, and *FREAD* are used as standard APL defined functions. The difference, however, is that the functions for use with data files are coded as assembler language subroutines and recognized as system functions by the APL interpreter.

A variety of other functions are provided as a part of a file subsystem. Typical features are: report formatting, a tab feature for printing, conversion and data editing facilities, capability for sharing files among users, and facilities for the input and output of large amounts of data. The reader is directed to the book by Gilman and Rose listed in the bibliography for information on a typical file subsystem.

## BIBLIOGRAPHY

- Falkoff, A. D., and K. E. Iverson, *APL 360 User's Manual*, IBM, T. J. Watson Res. Cen., Yorktown Heights, N.Y. (available as Form GH20-0683), 1968.
- Gilman, L., and A. J. Rose, *APL\360: An Interactive Approach*, Wiley, New York, 1970.
- Hellerman, H., *Digital Computer System Principles*, McGraw-Hill, New York, 1967.
- Iverson, K. E., *A Programming Language*, Wiley, New York, 1962.
- Iverson, K. E., *Elementary Functions: An Algorithmic Treatment*, Science Research Associates, Inc., Chicago, 1966.
- Katzan, H., *APL Programming and Computer Techniques*, Van Nostrand Reinhold, New York, 1970.
- Katzan, H., *APL User's Guide*, Van Nostrand Reinhold, New York, 1971.
- Pakin, S., *APL 360 Reference Manual*, 2nd ed., Science Research Associates, Inc., Chicago, 1972.

*Harry Katzan, Jr.*

# APPROXIMATION METHODS

The principles of approximation methods were certainly known to Archimedes in 250 B.C. [1]. He wanted to compute the area of a unit circle and to thereby find the value of  $\pi$ . Since he knew no direct way to do this he simply inscribed and circumscribed the circle with regular polygons. In doing so, upper and lower bounds for  $\pi$  were obtained, since the areas of the polygons were easily found. Archimedes introduced as many as 96 sides to find that

$$\frac{223}{71} < \pi < \frac{22}{7} \quad (1)$$

If one examines the upper half of the circle and represents that curve with  $f(x)$ ,  $0 \leq x \leq 1$ , then

$$\int_0^1 f(x)dx \doteq \int_0^1 \phi(x)dx = \sum_{j=1}^n W_j f(x_j) \quad (2)$$

where now  $0 \leq x_1 \leq x_2 < \dots < x_n \leq 1$  and  $\phi(x_j) = f(x_j)$ ,  $j = 1, 2, \dots, n$ . In Archimedes' work,  $\phi(x)$  represented the segment of the polygon. Since  $\phi(x_j) = f(x_j)$ , we would expect  $\phi(x)$  to approximate  $f(x)$  for  $x \neq x_j$  to some degree of accuracy and we would say that  $\phi(x)$  interpolates  $f(x)$  at the  $x_j$ ,  $j = 1, 2, \dots, n$ . As a consequence  $\sum_{j=1}^n W_j f(x_j)$  is called an integration or quadrature formula of interpolatory type. As an example, if  $\phi(0) = f(0)$ ,  $\phi(1/2) = f(1/2)$ , and  $\phi(1) = f(1)$ , with  $\phi(x)$  a quadratic function then

$$\int_0^1 f(x)dx = [(f(0) + 4f(1/2) + f(1))/3] + \{-f^4(\xi)/180\} \quad (3)$$

The term in braces denotes the error incurred if  $f(x)$  is sufficiently smooth and if all computations are exact. Also,  $\xi$  is some point between 0 and 1. The quadrature formula in Eq. (3) is the well-known Simpson's rule and can accommodate intervals other than  $0 \leq x \leq 1$ . Many other integration formulas have been developed and are discussed, e.g., in Refs. 2 and 3.

It is to be noted that an approximate method must include some kind of error estimate and must work for some fairly large class of problems.

Interpolation *per se* has been of considerable interest to mathematicians. Perhaps the greatest emphasis has been placed on finding  $p_{n-1}(x)$ , a polynomial of degree  $n - 1$  which would agree with a given function  $f(x)$  in some prescribed fashion. Lagrange interpolation is one of the more important types. Suppose that  $p_{n-1}(x_k) = f(x_k)$  for  $k = 1, 2, \dots, n$  and

$$l_k(x) = \frac{(x - x_1)(x - x_2) \cdots (x - x_{k-1})(x - x_{k+1}) \cdots (x - x_n)}{(x_k - x_1)(x_k - x_2) \cdots (x_k - x_{k-1})(x_k - x_{k+1}) \cdots (x_k - x_n)} \quad (4)$$

Then

$$p_{n-1}(x) = \left[ \sum_{k=1}^n l_k(x)f(x_k) \right] + \{f^{(n)}(\xi) \cdot (x - x_1)(x - x_2) \cdots (x - x_n)/n!\} \quad (5)$$

gives the Lagrange interpolant,  $p_{n-1}(x)$  to  $f(x)$  in brackets and the truncation error in braces for a sufficiently smooth function  $f(x)$ . Here,  $\xi$  is between  $x_1$  and  $x_n$ . It is worth noting that Runge has shown that if the  $x_j$  are evenly spaced the error can be unbounded. For a discussion, see Ref. 4.

Although it is not generally thought of as an interpolation result, Taylor's theorem involves the formula

$$\begin{aligned} f(x) = & \{f(a) + (x - a)f'(a) + [(x - a)^2/2!]f''(a) + \cdots \\ & + [(x - a)^{n-1}/(n - 1)!]f^{(n-1)}(a)\} \\ & + \{[(x - a)^n/n!]f^{(n)}(\xi)\} \end{aligned} \quad (6)$$

If the polynomial in the first set of braces is denoted by  $p_{n-1}(x)$ , then we find that  $f(a) = p_{n-1}(a)$  and  $f^{(j)}(a) = p_{n-1}^{(j)}(a)$  for  $j = 1, 2, \dots, n - 1$ . So long as  $x$  is close to  $a$ , it would be expected that  $p_n(x)$  could give a good approximation to  $f(x)$ . Taylor's theorem has been of great value in mathematics. For details of the theorem itself, see Ref. 5.

Notice that if  $n$  is prescribed, Lagrange interpolation requires no knowledge of the derivatives of  $f(x)$ , while Taylor's theorem requires knowledge of  $f(x)$  and its derivatives at just one point. Of course Hermite-type formulas require knowledge of  $f(x)$  and one or more derivatives at more than one point [6].

So far the emphasis has been on approximating  $f(x)$  by a polynomial. One reason for such emphasis is that polynomials are so easy to use in mathematics. Another is the Weierstrass approximation theorem. This theorem guarantees that for any function which is continuous on  $a \leq x \leq b$ , there is a polynomial of some degree  $p(x)$  such that for any given  $\varepsilon > 0$

$$|f(x) - p(x)| < \varepsilon \quad (7)$$

for any  $a \leq x \leq b$ . A similar result also holds for trigonometric polynomials of the form [7]

$$T(x) = \frac{1}{2}a_0 + a_1 \cos \pi x + b_1 \sin \pi x + \cdots + a_n \cos n\pi x + b_n \sin n\pi x \quad (8)$$

Although trigonometric polynomials have played a distinguished role in approximate calculations in mathematical physics, they do not seem well-suited to digital computers generally. As a consequence their use has become less popular in recent times even though trigonometric functions are useful in some instances. For work concerning trigonometric polynomials, see Refs. 8 and 9, and for some other methods, for approximating functions, see Refs. 7 and 10–12.

To a large extent, the role of polynomials is being taken over by the rapid development of piecewise polynomial functions or splines. If the interval  $a \leq x \leq b$  were partitioned so that  $a = x_1 < x_2 < \cdots < x_n = b$  then a piecewise polynomial function would be a polynomial of degree  $m$  for  $x_j \leq x \leq x_{j+1}$  and would have  $k$  continuous derivatives for  $x_1 \leq x < x_n$  with  $k \leq m - 1$ . For some piecewise polynomial functions, continuity conditions are imposed on their derivatives for

$-\infty < x < \infty$ . Of particular interest are functions  $p_j(x)$  which are defined for  $x_{j-1} < x < x_j$  by polynomials of degree  $2k - 1$  and which vanish elsewhere. Functions of this type are used to construct approximate solutions to differential equations and these solutions are  $2k - 2$  times continuously differentiable. Some references dealing with piecewise polynomial functions are Refs. 13–16.

Polynomial and piecewise polynomial interpolants can also be used to solve ordinary differential equations of the type

$$\frac{dy(x)}{dx} = f(x, y(x)), \quad y(a) = \eta \quad (9)$$

The solution to Eq. (9) can be written as

$$Y(x) = \eta + \int_a^x f(\xi, Y(\xi))d\xi, \quad a \leq x \leq b \quad (10)$$

If  $f(\xi, Y(\xi))$  is replaced by an interpolant then the formula

$$y_{n+k} - y_{n+p} = \sum_{i=0}^k \sum_{j=0}^m h^{j+1} b_{ij} f^{(j)}(x_{n+i}, y_{n+i}) \quad (11)$$

can be obtained. Here,  $0 \leq p \leq k - 1$  and the  $b_{ij}$  are constants determined by the interpolant and  $y_q$  is an approximation to  $Y(x_q) = Y(a + qh)$ . A formula such as Eq. (11) is called a linear multistep formula. In particular two such formulas are

$$y_{n+4} = y_{n+3} + h[55f(x_{n+3}, y_{n+3}) - 59f(x_{n+2}, y_{n+2}) + 37f(x_{n+1}, y_{n+1}) - 9f(x_n, y_n)]/24 \quad (12)$$

$$y_{n+3} = y_{n+2} + h[9f(x_{n+3}, y_{n+3}) + 19f(x_{n+2}, y_{n+2}) - 5f(x_{n+1}, y_{n+1}) + f(x_n, y_n)]/24 \quad (13)$$

Quite often, Eq. (12) is used to compute an initial estimate of  $Y(x_{n+k})$  and Eq. (13) is used to correct that estimate. Many other types of methods are available for use. For example, see Refs. 17–19 which are devoted to this topic and which treat the error analysis.

Notice that Eq. (10) is a special case of the problem of finding  $Y(x)$  such that

$$Y(x) = g(x) + \int_a^x k(x, \xi, Y(\xi))d\xi, \quad a \leq x \leq b \quad (14)$$

If  $x = x_n$  and the integral is replaced by a quadrature formula, then

$$y(x_n) = g(x_n) + \sum_{j=1}^n W_j k(x_n, x_j, y_j) \quad (15)$$

where  $y(x_n) \doteq Y(x_n)$ . One could use the integration formula

$$\int_{x_n}^{x_{n+1}} f(x)dx = h(f(x_n) + f(x_{n+1}))/2 + \{-h^3 f^{(3)}(\xi)/12\}, \quad x_n < \xi < x_{n+1} \quad (16)$$

to this end. As a consequence Eq. (15) amounts to a system of nonlinear algebraic equations which must be solved somehow. Approximate techniques for solving integral equations are given in Refs. 20 and 21.

Among the simplest methods for solving nonlinear equations is the scheme

$$y^{[n+1]} = F(y^{[n]}) \quad (17)$$

where the numbers in brackets denote the iteration number. If  $L < 1$  and

$$|F(y) - F(z)| < L |y - z| \quad (18)$$

then the iterations (17) yield that  $y$  for which

$$y = F(y) \quad (19)$$

Other techniques for solving systems of nonlinear equations are described in Refs. 22 and 23. Of course nonlinear equations arise in many ways and in a wide variety of disciplines. Systems of linear equations can arise if in Eq. (11) the integrand is replaced by  $k(x, \xi) Y(\xi)$ . They might also arise by approximating

$$\frac{\partial^2 u(x, y)}{\partial x^2} + \frac{\partial^2 u(x, y)}{\partial y^2} = 0, \quad 0 \leq x \leq 1, \quad 0 \leq y \leq 1 \quad (20)$$

by

$$u(x_{n+1}, y_n) - 2u(x_n, y_n) + u(x_{n-1}, y_n) \\ + u(x_n, y_{n+1}) - 2u(x_n, y_n) + u(x_n, y_{n-1}) = 0 \quad (21)$$

Here  $x_n = y_n = nh$  and  $NH = 1$ . If  $u(x, y)$  takes on specified values on the boundary of the square then a linear algebraic system of equations is obtained. If  $N > 1$ , Cramer's rule should be avoided like the plague and some approximate method or calculations should be used to solve the problem [24]. Among methods for solving linear algebraic systems are successive overrelaxation (SOR) and Gaussian elimination with partial pivoting. See Refs. 25–27 for other methods to solve linear systems. For techniques to solve partial differential equations, see Ref. 28.

In recent years, methods of approximation in an abstract setting have become increasingly popular. References along these lines include Refs. 16, 20, 23, and 29–31.

There are numerous approximate methods for solving problems which have not been treated here. However, the flavor of approximate methods is this. If it is impractical or impossible to provide an exact solution to a computational problem, which should be solved, either there is or there should be a corresponding approximation method for solving the problem. General books include Refs. 4, 6, 21, 32, and 33.

## REFERENCES

1. D. C. Joyce, Survey of extrapolation processes in numerical analysis, *SIAM Rev.* **13**, 435–490 (1971).
2. P. J. Davis and P. Rabinowitz, *Numerical Integration*, Blaisdell, Waltham, Mass., 1967.
3. V. I. Krylov, *Approximate Calculation of Integrals* (A. H. Stroud, transl.), Macmillan, New York, 1962.
4. E. Isaacson and H. B. Keller, *Analysis of Numerical Methods*, Wiley, New York, 1966.
5. T. M. Apostol, *Mathematical Analysis; A Modern Approach to Advanced Calculus*, Addison-Wesley, Reading, Mass., 1957.

6. F. B. Hildebrand, *Introduction to Numerical Analysis*, McGraw-Hill, New York, 1956.
7. J. R. Rice, *Approximation of Functions*, Vol. 1: *Linear Theory*, Addison-Wesley, Reading, Mass., 1964.
8. H. F. Davis, *Fourier Series and Orthogonal Functions*, Allyn and Bacon, Boston, 1963.
9. R. E. Edwards, *Fourier Series: A Modern Introduction*, Vol. 1, Holt, Rinehart and Winston, New York, 1967.
10. E. W. Cheney, *Introduction to Approximation Theory*, McGraw-Hill, New York, 1966.
11. G. G. Lorentz, *Approximation of Functions*, Holt, Rinehart and Winston, New York, 1966.
12. J. R. Rice, *Approximation of Functions*, Vol. 2: *Nonlinear and Multivariate Theory*, Addison-Wesley, Reading, Mass., 1969.
13. J. H. Ahlberg, E. N. Nilson, and J. L. Walsh, *The Theory of Splines and Their Applications*, Academic, New York, 1967.
14. T. N. E. Greville, ed., *Theory and Applications of Spline Functions*, Academic, New York, 1969.
15. I. J. Schoenberg, ed., *Approximations with Special Emphasis on Spline Functions*, Academic, New York, 1969.
16. R. S. Varga, *Functional Analysis and Approximation Theory in Numerical Analysis*, Society for Industrial and Applied Mathematics, Philadelphia, 1971.
17. F. Ceschino and J. Kuntzmann, *Numerical Solution of Initial Value Problems* (D. Boyanovitch, transl.), Prentice-Hall, Englewood Cliffs, N.J., 1966.
18. P. Henrici, *Discrete Variable Methods in Ordinary Differential Equations*, Wiley, New York, 1962.
19. L. Lapidus and J. H. Seinfeld, *Numerical Solution of Ordinary Differential Equations*, Academic, New York, 1971.
20. P. M. Anselone, *Collectively Compact Operator Approximation Theory and Applications to Integral Equations*, Prentice-Hall, Englewood Cliffs, N.J., 1971.
21. L. Fox, ed., *Numerical Solution of Ordinary and Partial Differential Equations*, Pergamon, Oxford, 1962.
22. J. M. Ortega and W. C. Rheinboldt, *Iterative Solution of Nonlinear Equations in Several Variables*, Academic, New York, 1970.
23. L. B. Rall, *Computational Solution of Nonlinear Operator Equations*, Wiley, New York, 1969.
24. G. E. Forsythe, Pitfalls in computation, or why a math book isn't enough, *Amer. Math. Mon.* **77**, 931–956 (1970).
25. V. N. Faddeeva, *Computational Methods of Linear Algebra* (C. D. Benster, transl.), Dover, New York, 1959.
26. R. S. Varga, *Matrix Iterative Analysis*, Prentice-Hall, Englewood Cliffs, N. J., 1962.
27. D. M. Young, *Iterative Solution of Large Linear Systems*, Academic, New York, 1971.
28. G. E. Forsythe and W. R. Wasow, *Finite-Difference Methods for Partial Differential Equations*, Wiley, New York, 1960.
29. P. L. Butzer and H. Berens, *Semi-Groups of Operators and Approximation*, Springer, New York, 1967.
30. L. Collatz, *Functional Analysis and Numerical Mathematics* (H. Oser, transl.), Academic, New York, 1966.
31. I. Singer, *Best Approximation in Normed Linear Spaces by Elements of Linear Subspaces*, Springer, New York, 1970.
32. B. Carnahan, H. A. Luther, and J. O. Wilkes, *Applied Numerical Methods*, Wiley, New York, 1969.
33. S. D. Conte, *Elementary Numerical Analysis: An Algorithmic Approach*, McGraw-Hill, New York, 1965.

# ARGONNE NATIONAL LABORATORY\*

## THE LABORATORY: A DESCRIPTION

As one of its first acts in 1946, the U.S. Atomic Energy Commission established the Argonne National Laboratory as a permanent national laboratory for atomic energy research and development. Its predecessor was the University of Chicago's World War II Metallurgical Laboratory where scientists achieved the world's first controlled nuclear chain reaction on December 2, 1942.

Argonne is located on a 3700-acre site in DuPage County, Illinois, 27 miles southwest of Chicago's Loop. Just a few miles away is the original Cook County Argonne Forest Preserve site where Metallurgical Laboratory experiments with graphite and heavy-water reactors were carried out and from which the laboratory's name is derived. Some of the laboratory's nuclear reactors and test installations are located at an auxiliary site at the National Reactor Testing Station, 50 miles west of Idaho Falls, Idaho. Argonne has approximately 4400 employees including 600 at the ANL-West location; about 40% of these are staff scientists and engineers.

From its establishment, the laboratory has been operated by The University of Chicago. Since 1966, this operation has been under the terms of a tripartite agreement involving the Atomic Energy Commission, the Argonne Universities Association (a corporation of 30 universities), and The University of Chicago.

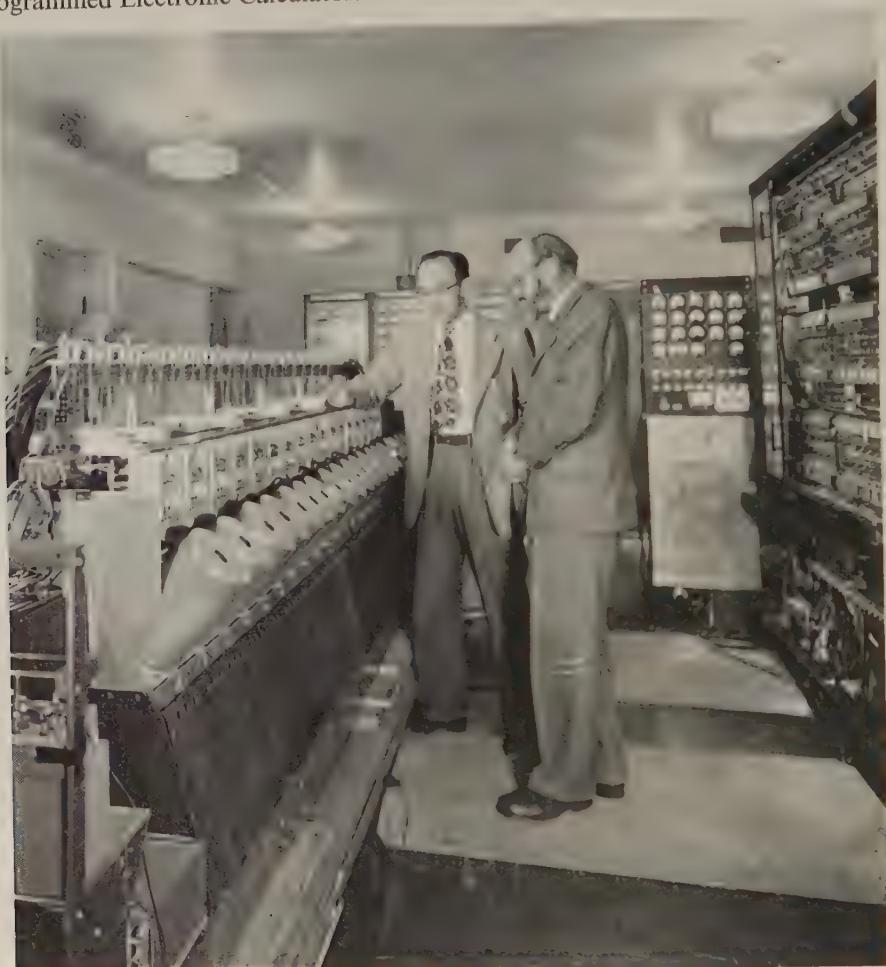
Funds are provided by the USAEC to conduct research and development in three major areas: reactor sciences, physical sciences, and life sciences. Currently, the reactor effort is devoted primarily to development of the liquid-metal-cooled fast breeder power reactor, LMFBR, while much of the basic research is concerned with the application of radiation as a tool in the physical and life sciences and the high energy physics program surrounding the operation of the laboratory's Zero Gradient Synchrotron (ZGS), one of the world's largest particle accelerators.

## CONSTRUCTION OF EARLY GENERAL-PURPOSE COMPUTERS [1, 2]

Occupation of the DuPage County site began in 1948, the same year in which the Atomic Energy Commission reported to Congress that progress was being made by von Neumann and colleagues at the Institute for Advanced Study, Princeton University, in the "design of special types of high-speed computing machines for the Commission." Their 1949 report supported the development and construction by IAS of an electronic computer and announced construction of computers already underway at

\* Work performed under the auspices of the U.S. Atomic Energy Commission.

both Argonne and Los Alamos based on this von Neumann prototype. By early 1950, the laboratory's Monthly Progress Reports contained descriptions of work in progress on this first Argonne digital computer. At this time the laboratory physics problems and reactor design computations were being solved by Herculean desk calculator efforts and the assistance of the newly acquired Reeves analog machine or IBM Card Programmed Electronic Calculator.



**Fig. 1.** J. C. Chu and D. A. Flanders at the Williams-tube memory unit of the AVIDAC, Argonne's Version of the Institute's Digital Automatic Computer.

### AVIDAC

In January 1953, Dr. D. A. Flanders, the Senior Mathematician at the laboratory who proposed building Argonne's first digital computer, and J. C. Chu, who supervised its engineering and construction, announced its completion. (Fig. 1). The machine was

christened AVIDAC, an acronym for Argonne's Version of the Institute's Digital Automatic Computer. This version differed from the prototype by including a greater variety of shift orders, overflow detection, and transfer of control on overflow.

The AVIDAC consisted of a binary fixed-point arithmetic unit, a control unit, a cathode-ray tube memory designed to store 1024 40-bit words, and paper tape input-output units. Modified Teletype and Western Union equipment was used off-line to prepare instruction and data tapes and to print the output tapes generated by the computer. Memory access time was 15  $\mu$ sec; additions were performed in 10  $\mu$ sec and multiplication required 1 msec.

Information was stored as electric charges on the inside face of the memory cathode-ray tubes, the presence of these charges detected by an amplifier, and this signal switched to the computer's arithmetic unit when requested, or alternatively used to regenerate the information in the memory. Regeneration was required about 50 times per second. This electrostatic memory proved the weakest link in the system and it was necessary to reduce the design 1024-word memory to 256 words to alleviate the "read-around" errors—changing of a 0 to a 1 due to electron refilling caused by repeated consultation of adjacent locations.

The machine contained 2500 electronic tubes, about 8000 resistors, and over three and a half miles of electrical wire. A bank of 355 storage battery cells, under constant charge from motor generators, provided the power for the computer which was constructed at a cost of \$250,000.

Later modifications to the AVIDAC system included the addition of Ferranti photoelectric paper tape readers and the high-speed Teletype paper tape punch. The machine served as a Laboratory computer until the latter part of 1957.

### ORACLE [3]

In October 1951, while the AVIDAC was being built, Argonne began the construction, with the assistance of Oak Ridge engineers, of a second IAS computer progeny for the Oak Ridge National Laboratory.

The time lag between the construction of the two machines was sufficient to incorporate improvements suggested by AVIDAC experience. In an attempt to alleviate the read-around problem, the Oak Ridge memory unit was composed of 80, rather than 40, cathode-ray tubes of a new type. These were packaged two per unit and designed to check one another, or if proved extremely reliable, to operate independently as a 2048-word memory. Three extra units were kept ready as spares to provide quick replacement in case a defect was detected in one of the operating units.

Dubbed ORACLE for Oak Ridge Automatic Computer, Logical Engine, this machine's 2048 40-bit word memory operated successfully and its arithmetic speed was improved to perform additions in 5  $\mu$ sec and multiplications in 500  $\mu$ sec.

The ORACLE contained 3500 electronic tubes, about 20,000 resistors, and approximately seven miles of electrical wire. It was built in 20 months at a cost of \$350,000.

The laboratory press release issued in the fall of 1953 immediately preceding its shipment to Tennessee proclaimed the ORACLE the "world's fastest high-speed general purpose digital computer . . . with the greatest capacity of any yet built."

Original plans called for Argonne development and construction of 4-million word magnetic tape auxiliary memory systems for use with the Argonne-produced computers. Each of these 42-track 2-in. wide tape systems consisted of four physical units. The first system completed was shipped to Oak Ridge at the end of June 1955, for use with the ORACLE.

### GEORGE [4, 5]

With the successful operation of the ORACLE memory at 2048 words, several proposals were advanced for improving the AVIDAC memory, a copy of the prototype Williams-tube design. A contract was ultimately signed with the International Telemeter Corporation for the construction of a 4096-word magnetic core memory. In early 1954, it was decided that rather than revise the AVIDAC to fit the core memory, it would be advantageous to build a new computer, GEORGE, to go with the new memory. The estimated completion date was set for mid-1955, and the auxiliary magnetic tape memory system intended for the AVIDAC was rescheduled for GEORGE. Due to delays in delivery of equipment, and the switch from a Laboratory research project era to a commercial one, with the concomitant fluctuation in key program and engineering personnel, GEORGE was not completed until late 1957.

The design of GEORGE was based on experience acquired with the AVIDAC and ORACLE and use of the Serial 4 UNIVAC at the AEC Computer Facility operated by New York University. This new computer, like the Institute-sired machines, had a 40-bit word, represented as ten hexadecimal characters, but while the AVIDAC and ORACLE held two single-address instructions per word, GEORGE held a single two-address (*A* and *B*) instruction per word. The instruction was divided into two independent parts—the order and the tag. Use of the order code and the *A*-address was essentially equivalent to the IAS single-address instruction. The *B*-address could be designated as the address of an index register, a preliminary addend, an operand store address, or a branch address. Up to four memory cycles could be evoked in the execution of a single instruction permitting preliminary setting of the arithmetic registers (PC), an arithmetic operation (OC), a store into memory (RC), and a jump, or branch (FC). PC, FC, and RC each required a 27- $\mu$ sec memory cycle, if evoked. OC varied from one memory cycle for addition, to 22 cycles or 594  $\mu$ sec for 39-step division. Half-precision, or 19-step, multiply and divide instructions were incorporated specifically for use in Monte Carlo calculations. Parity checking was employed in the input, output, and memory transfers and hardware overflow detection was provided. The console contained address recognition stop switches, breakpoint recognition (or sense) switches, and special-order buttons. If these special-order buttons were depressed, the order was executed and a unique symbol was recorded on the console typewriter. Other GEORGE equipment included an input keyboard, paper tape input-output facilities, the wide magnetic tape auxiliary memory, an off-line Electronic Associates Dataplotter, an Anelex line printer, a cathode-ray tube display device, 35-mm film recorder, and IBM magnetic tape input and output units.

### FLIP-GUS [6, 7]

The next step was the upgrading of GEORGE with floating-point arithmetic hardware. Here again, the acquisition of enhancement hardware brought forth a new computer designed to take advantage of the latest solid-state components.

One of the principal design objectives for the unit was the implementation of index-of-significance arithmetic. The computations were carried out in normalized floating-point binary arithmetic with a significance index computed for each result. Eighty-bit words were used with numbers represented in three parts—63 bits were allocated to the normalized fraction, 10 bits to its base 2 exponent, 6 to the associated index of significance with the remaining bit reserved as a word flag. FLIP, an acronym for floating indexed point, grew to contain in addition to its floating-point unit, an integer unit, and a command and control unit allowing it to take over the major portion of the arithmetic operations in this multiprocessor system. GEORGE, the slower processor, controlled the peripheral devices and performed other nonarithmetic, data conversions, formatting, and housekeeping functions. The processors communicated by means of sense and interrupt registers and shared two  $2-\mu\text{sec}$  4096 80-bit word core memory modules.

FLIP had a complex instruction format utilizing 20-bit instructions and addresses, described as control groups. As many as four such groups might be required for the 3-address floating-point instruction. The floating-point unit contained, in addition to the hardware required for the arithmetic and square root operations, four high-speed registers for temporary storage. Operands obtained from these registers were specified within the instruction allowing up to four instructions to be stored in a single FLIP word. The integer unit contained 16 high-speed index registers. The integer arithmetic instruction incremented or decremented these registers with the contents of specified control groups, counters, other index registers, or the exponent or index-of-significance portion of a word, and tested the result.

The command and control unit permitted the selection of an absolute or relative mode of addressing with or without indexing. Up to four index registers could be added in a single address calculation and multilevel indirect addressing was possible.

This multiprocessor system was termed GUS for GEORGE unified system. All communication to and from the memory modules was channelled through a central exchange designed to accommodate up to seven independent computers or devices, each with its own addressing facility. Prior to retirement in April 1966, GUS acquired two such additions—the Real Time Communicator, a link with an analog computer providing hybrid computing capability, and DAPHNIS, a Perceptron simulator.

During this entire period, 1949–1966, while Argonne was designing and building general-purpose computers, attempts were made to find commercially adequate equipment to meet the Laboratory's computational requirements. An analog facility was maintained on-site, and the early IBM CPC was followed by trips to the NYU UNIVAC facility to augment the ANL computer capabilities. In mid-1957, the Accounting Department acquired an IBM 650. This was followed by installation of an IBM 704 in the central facility that fall. In 1963, a CDC 3600 was purchased, and in July 1966, the 704 was replaced by an IBM 360 system model 50. The IBM 360 system was upgraded to a model 50/75 ASP system in mid-1967, and the CDC 3600 was moved out to make

room for the IBM 360/195 delivered in 1972. Plans call for the model 195 to carry the batch operation with the model 75 devoted primarily to TSO, the time-sharing facility for the Laboratory. As these commercial machines proved capable of handling the general-purpose computing requirements of the Laboratory's programs, research efforts in the development of general-purpose machines were phased out. The talents of the staff were directed toward the construction of special-purpose hardware and the mathematical methods and computational techniques for solving complex scientific and engineering problems using such hardware in conjunction with the available computer resources.

## DEVELOPMENT OF SPECIAL-PURPOSE HARDWARE

In the past decade much attention has been given the use of computers for the acquisition and analysis of experimental data, control of equipment, pattern recognition and image processing, and the enhancement or improvement of computer systems. Argonne's contributions in these areas include the following.

### On-line Data Acquisition and Analysis

#### *Van de Graaff Accelerators* [8-12]

PHYLIS, the physics on-line information system, was completed in 1964. It was an on-line data acquisition and analysis system used for processing data obtained in low-energy physics experiments at the 4.5-MeV Van de Graaff and 13-MeV tandem Van de Graaff accelerators while the experiments themselves were in process. The processor for the system was an ASI 2100 computer, with 2  $\mu$ sec memory cycle time, equipped with direct communication links to a Nuclear Data multichannel analyzer connected to the tandem, to a remote station at the other accelerator, and to the CDC 3600 computer at the central computing facility. The analyzer link allowed the ASI 2100 to read from the analog-to-digital converters and permitted bidirectional data transfers between the 2100 memory and the analyzer memory. The CDC 3600 link allowed the ASI 2100 to interrupt the larger computer to perform a calculation required by the experimenter, and beyond the capability of the 2100. Card reader, card punch, typewriter, and line printer equipment was located at both the remote station and the ASI 2100 site. In addition, a Calcomp plotter was attached to the 2100, and a typewriter and an oscilloscope display with light pen were provided the experimenter at the analyzer location.

System data processing programs handled such functions as data transfers, punching, printing, and displaying of selected data, and analysis programs included polarization calculations for neutron scattering experiments, reaction kinetics codes, and least squares fitting routines.

This system has evolved into the present Physics Division multicomputer, multi-experiment system which consists of on-line ASI computers at both the 15-MeV tandem

and 4-MeV Dynamitron (a 1968 replacement of the Van de Graaff) accelerators and the off-line ASI 2100. In addition, external core memories (98,304 21-bit words) for the ASI 210 at the tandem site and for the off-line ASI 2100 have been included in the system. The tandem external memory is operable either as a two-parameter analyzer (up to  $256 \times 256$  channels), or as up to eight one-parameter analyzers of 1024 channels each and has its own display unit. The ASI 2100 external memory is primarily used for data manipulation and program preparation. At the tandem there are computer-controlled scattering chambers and a computer-controlled  $\gamma$ -ray angular-correlation table.

### *Mössbauer-Effect Experiments [13, 14]*

A data collection system developed to gather data from up to four Mössbauer-effect experiments simultaneously was installed in 1968. This system uses an SCC 650 computer with 4096 12-bit words of core memory, display scope, paper tape punch, and typewriter. Each experiment is allocated a fixed portion of memory.

The Mössbauer effect is the absorption and emission of gamma rays by atomic nuclei. The gamma rays must have precisely the correct energy to be absorbed; this precision is obtained by Doppler shifting gamma rays of approximately that energy. Each experiment utilizes a radiation detector that moves to and from an energy source. Every cycle of the detector is divided into 400 time segments. Data collection consists of adding the number of detector pulses associated with a particular time segment to the accumulated total for that segment. Detection pulses are added in interface hardware rather than with the computer's arithmetic registers permitting rapid display and data collection when the computer is stopped.

The control program allows an experimenter through typewriter commands to initiate or stop any of the four experiments, to display a part or all of the data collected by any experiment at any of 15 different scale factors, and to perform binary-to-decimal conversion prior to punching paper tape or typing output.

### *EBR-II Data Acquisition System*

The EBR-II Data Acquisition System, DAS, became operational in 1971 in support of the operation of the reactor and experiments utilizing the reactor. The DAS is based on a SIGMA-5 computer with 32K memory. Up to 740 independent input signals can be handled simultaneously making it possible to perform the standard on-line data logging and reduction functions. It has also been used in conjunction with EBR-II control-rod calibration and with rod oscillator tests conducted to determine the reactor transfer function.

### **Computer Control of Equipment**

#### *An Automated Van de Graaff and Fast Neutron Laboratory [15-19]*

In the reactor development program, a third Van de Graaff originally served as a neutron source in experiments measuring neutron scattering properties of reactor

components and the probabilities of neutron capture by reactor material. Starting in 1964, an on-line computer system was used to control the major parameters of this 3.5-MeV machine while simultaneously performing the data collection and analysis functions.

The computer configuration consisted of two CDC 160As, each with 8192 12-bit words of core memory, coupled to an auxiliary 8K memory unit. Peripheral equipment included a typewriter, printer, graph plotter, and paper tape reader and punch. There was, in addition, a large collection of locally designed instruments—a display system, analog-to-digital and digital-to-analog converters, and scalers interfaced to either or both machines. The two computers communicated by means of the shared auxiliary memory.

In measurements of total cross sections (the sums of scattering and absorption probabilities) of fast neutrons of nearly the same energy, this system was completely automated. The computer varied the Van de Graaff energy step-by-step over a selected range, moving the sample material in and out, at the same time accumulating the particle counts. Then the cross section and its associated error were calculated from the counts and adjustments made for the next measurement.

Today, a special-purpose high-intensity tandem accelerator is employed in the determination of the microscopic nuclear data for the reactor development program. On-line computing systems were designed as a basic part of the facility to perform functions in the areas of data acquisition, data reduction, and feedback control in experiments extending to neutron energies of up to 20 MeV.

The computer configuration consists of one SEL 840 with 316K of 24-bit words for general-purpose data acquisition and control, the two CDC 160As which perform delegated data acquisition functions, a second SEL 840 used for off-line data reduction and analysis, and a third CDC 160A available for additional off-line data reduction, as required. A full complement of peripheral equipment is available to the system including line printers, card input and output, magnetic tape units, disk units, graphic display terminals, and console typewriters, in addition to the necessary specialized interfacing to and from experimental equipment and the main accelerator.

Complex experiments are carried out involving a diversity of on-line data acquisition, reduction, and analysis functions with emphasis on the manipulation of multidimensional data arrays. Simpler experiments (e.g., total neutron cross section measurements) include feedback control to the neutronic sensors and to the basic accelerator to optimize the experimental procedures. The determination of such items as statistical accuracies, accelerator energies, and experimental geometric configurations, has been automated. Supporting off-line operations encompass the complex data manipulation functions, Monte Carlo simulation, and nuclear model calculations.

#### *Neutron and X-ray Diffraction Equipment [14, 20, 21]*

By 1968, four ARCADE (for Argonne's Computer-Aided Diffraction Equipment) systems were in operation and a fifth was under construction (Fig. 2). The ARCADE system was designed to control the instruments used in neutron and X-ray diffraction studies, as well as to collect and analyze the experimental results. The two systems developed for

metallurgy experiments control a full-circle goniostat (crystal orienter), and a heavy-duty spectrometer and a cryo-orienter. The solid-state studies instrument controlled by an ARCADE system is a polarized neutron diffractometer; the chemical research instrument a



**Fig. 2.** One of the two ARCADE, Argonne's computer-aided diffraction equipment systems developed for metallurgy experiments in use at CP-5, the laboratory's research reactor.

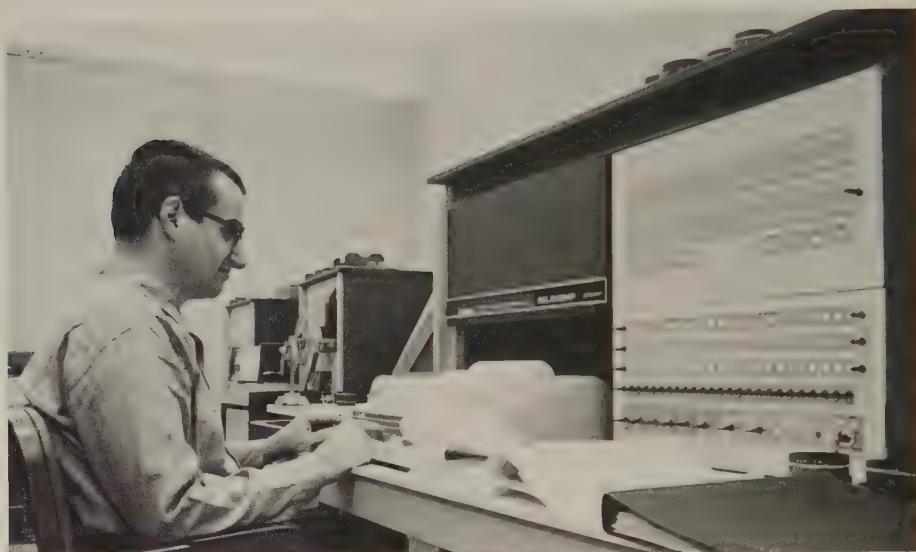
quarter-circle X-ray diffractometer. The fifth system controls a full-circle X-ray unit used for biological experiments.

All of the systems have as the control computer a 16-bit disk-oriented IBM 1130 equipped with paper tape input and output, typewriter, and printer. Using the typewriter, the experimenter supplies the information to control the motors and position the crystal. The computer then performs the crystal structure analysis, determines the intensity of the beam of neutrons or X-rays diffracted from the crystal, and decides what the next orientation will be.

#### *Automatic Reactor Control [22]*

Important behavior characteristics of large fast neutron power reactors are routinely studied in low-power critical experimental facilities constructed for this purpose. Argonne has such facilities at both the Illinois and Idaho sites. A DDP 24 computer acquired primarily for on-line data collection and analysis of such critical assemblies

was used in early studies of automatic reactor control (Fig. 3). In 1967 experiments the final fuel drawer of assembly ZPR 9 was inserted under computer control bringing the



**Fig. 3.** An on-line computer used for data collection and analysis at the Argonne ZPR-6 and ZPR-9 (zero-power reactor) critical facilities.

reactor through criticality, the point of achieving a self-sustaining chain reaction, and then leveling off to a predetermined power.

#### *A Computer-Controlled Microscope [23, 24]*

An optical microscope with computer control of specimen motion in three dimensions, via stepping motors, has been developed and used for automatic scanning of solid-state track recorders (SSTR). These SSTR, in which tracks of fission fragments are etched, have been employed in nuclear physics experiments designed to measure spontaneous fission half-lives, absolute fast neutron fission yields, and environmental neutron intensities. A major drawback is the necessity of visual or manual counting of the recorded fission tracks, an expensive, time-consuming, often imprecise activity.

At Argonne a DDP 24 computer with 8K core memory and fixed-head disk storage was programmed and interfaced with a Leitz Ortholux optical microscope on a movable stage for automatic scanning and analysis of SSTR. Since SSTR applications call for counting of the tracks contained within a given area of a specimen and that whole area cannot be scanned within a single field of view, it was decided that the best approach would be to scan by specimen motion only. This is done by use of a stage movable in two-dimensions by stepping motors coupled to micrometer screws. For automatic focusing the microscope tube is moved up or down relative to the stage in 3.2- $\mu$ sec steps by a third stepping motor driving a micrometer screw.

The automatic track scanner system, ATS, has been calibrated for Makrofol solid-state track recorders and absolute fission rate measurements have been completed. The running time is approximately 10 hr/cm<sup>2</sup> of scanned area, controlled primarily by stage motion. Track densities average 10,000 to 25,000/cm<sup>2</sup>. The benefits accrued by ATS are predominantly in the denser SSTR samples, the elimination of human bias, and the inherent capability of the computer system to handle the measuring and accumulating of track areas and associated distributions.

### Photographic Scanning and Measuring Systems

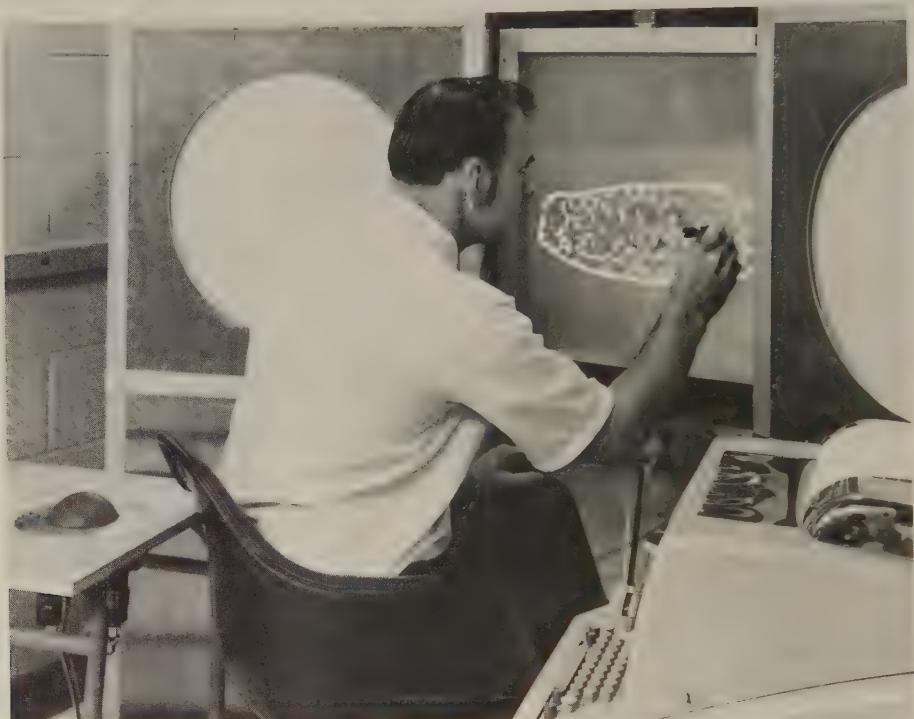
#### *General-Purpose Image Processing [25-28]*

Research on general-purpose film scanning and image processing systems began in the early sixties; the first such system, CHLOE, became operational in early 1963. This system was designed to digitize photographic information and perform calculations on the data obtained. Although influenced by its potential application to processing photographs from high-energy physics spark chamber experiments, the design was intended to be versatile and flexible. Because of this, CHLOE found wide use in a variety of unrelated applications whose only common characteristic was the choice of film as the data storage medium.

The two major system components were an ASI 210 computer with paper tape input and output, console typewriter, and auxiliary magnetic tape unit, and an optical scanner unit, designed and built at Argonne, controlled by the computer. The ASI 210 had an 8192 21-bit core memory with 2  $\mu$ sec cycle time. Each of the twin scanning stations consisted of a cathode-ray tube (CRT) light source which projected a spot of light onto the film and a photomultiplier to view the light transmitted through the film. The light spot was driven by a pair of counting registers, so as to appear sequentially at regular points over a designated area of the film frame. The extent of this area and the frequency of the spot's appearance were determined by the computer program. An array of as many as 4096 by 4096 spots could be selected over a single 1.25 by 1.25 in. area. When the photomultiplier detected a significant change in transmitted light, the coordinates of that point, together with the measured density level, were sent to the ASI 210 memory. The ASI program selected which of the eight density levels were to be used and the film condition, i.e., whether it was white on black, or black on white. In addition, the CRT light output could be set to any of 1024 values. A large display scope was provided to permit operator monitoring of scanning activity. Other features provided the ability to perform the scan in either of two patterns, and the ability to communicate directly with the central facility's CDC 3600 when its increased computing power or larger memory was needed.

CHLOE was retired in 1970 and construction started on the successor system, ALICE, which became operational that year. It consists of three major components—a DEC PDP 10 computer, the Controller, and the three scanning stations. The PDP 10 is a 36-bit word machine with 1  $\mu$ sec cycle 32,768 word core memory. It has floating-point and byte-manipulation instruction hardware lacking in the earlier CHLOE computer.

Peripheral equipment includes a model 33 Teletype console, card reader, printer, paper tape input and output, 3 DECTape units, and both a 7-track and a 9-track magnetic tape unit. The Controller is connected to the second memory port of each of the 16K memory modules. In operation the Controller's function is to specify to the scanning station a series of points ( $x-y$  Cartesian coordinate pairs) at which to measure the density level and to dictate the type of information (coordinates, densities, or both) to be recorded, the location in memory at which the data will be stored, the density levels to be used, and



**Fig. 4.** An operator using the GRAF/PEN at the optical display screen of the ALICE general-purpose image-processing system.

mode of operation for the scanning. Controller action is governed by program setting of 15 parameter registers.

The counting registers of CHLOE were replaced by adder circuits removing most of the earlier restrictions on size and direction of the interpoint and interline separation. A 131,072 by 131,072 spot raster is used with eight of the parameter registers set to define any parallelogram-shaped series of points. In addition, individual point, line, or area specifications defined by register settings can be chained to produce irregular scans. Density levels of 0 to 63 can be accommodated. Operator facilities provided include a monitor "slave" scope, a conventional CRT display, a 35-mm film recorder, a track ball, and an optical display screen with GRAF/PEN input facility (Fig. 4). The 36 PDP 10

console sense switches and program-controlled display lights are duplicated at the display scope station for operating convenience.

Scanning station 0 is for 16- or 35-mm sprocketed film. Scanning station 1 is limited to the 35-mm film applications, and station 2 accommodates a Leitz Orthoplan Research microscope.



**Fig. 5** A 12-foot bubble chamber experiment photograph being examined during measurement by the POLLY film scanning system.

#### *High-Energy Physics Film Scanning [27, 29, 30]*

Based on CHLOE experience, a series of POLLY film scanning systems has been developed for the automatic scanning and measuring on film produced in bubble chamber experiments (Fig. 5). The prototype POLLY machine used a PDP 7 computer

equipped with paper tape input and output units, a console typewriter, and a magnetic tape unit. It was constructed in 1966 to provide facilities for operator interaction to circumvent pattern recognition difficulties during the processing. Two CRT scanners were employed; one for the conventional (TV) flying-spot scan, the other for the more precise measuring of track information. The TV scan was displayed on a monitor screen and the operator could effect the track selection and measurement procedures by placing a marker on the screen or by input from the console typewriter.

POLLY II, developed in 1968, was a production model of this prototype with an improved monitor station and utilizing the larger Xerox Data Systems SIGMA-7 computer with 48K memory. The first automatically scanned bubble chamber experiment was performed using POLLY II during 1969. Continual refinement is aimed at increasing the ability of the POLLY system to automatically recognize and measure very complicated pictures. POLLY III, the latest development in the series, has been in production use since early 1972, measuring film from the 12-ft bubble chamber experiments.

#### *A Nuclear Emulsion Plate Scanner [14, 27, 31]*

In low-energy physics experiments glass plates, 2 by 10 in. in size, coated with a fine grain nuclear emulsion are exposed in broad-range magnetic spectrographs to record tracks of incident nuclear particles. The position and energy of the tracks provide information on the energy distribution of the incident particles. Tracks are typically 2- to 4- $\mu$  wide and approximately 100- $\mu$  long. PAULETTE, a computer-controlled scanning machine, was constructed to process these nuclear track plates. The system uses a PDP 9 computer to control a high resolution image-dissector tube scan and "count" the tracks detected about 50 times as fast as the average human scanner.

#### *A High-Energy Physics Film Measuring Table [32]*

In the mid-sixties, developmental work was carried out on a manual measuring table for high-energy physics film. The table, called LAURA, used a helium-neon continuous-wave laser as a collimated light source replacing the drive wires that conventionally connect the cross-hair indicator and shaft encoder. LAURA was interfaced to an ASI 210 computer to calculate the shaft angle and compute the position of the cross-hairs. It was intended that the single computer should support four such tables, computing coordinates and performing coordinate cross-checking. Although the angular encoder was inherently capable of determining the desired accuracy of the angle, temperature and low-amplitude noise affected repeatability to the extent that the project was discontinued as too costly for the accuracy achieved.

#### **Computer System Enhancement**

#### *Hybrid Computing Facilities [33]*

Since the original digital-analog Real Time Communicator (RTC) link was designed for the laboratory's GUS, interest in maintaining hybrid facilities for reactor kinetics

and feedback control problems has resulted in implementation of improved systems. The RTC consists of digital and analog control units and analog-to-digital (*A* to *D*) and *D* to *A* converters. Originally, it interconnected the digital computer GEORGE with a PACE Model 131-R analog unit. When GEORGE was dismantled, the RTC was used to link a PDP 7 digital computer with two PACE analog consoles. The PDP 7 had a 8192 18-bit word memory and paper tape I/O equipment. Late in 1965 two Reeves REAC 550



**Fig. 6.** The Argonne hybrid computing facility with the IBM 1130 digital computer in the foreground and the Reeves REAC 550 analog in the background.

analog consoles were added to the system, and in June 1968 an IBM 1130 computer with disk, card reader punch, and line printer facilities replaced the PDP 7 (Fig. 6). The digital and analog components can be used together as a hybrid system, or separately as stand-alone digital and analog machines.

#### *A Pattern Extractor Device [34]*

The last application run on the GEORGE machine was in conjunction with use of the special-purpose hardware device, DAPHNIS. DAPHNIS was implemented to provide a 700/l gain in speed over program simulation of processes involved in testing a variety of Perceptron models for recognition of events from nuclear emulsion track plates. The testing required generation of logical sums and products and counting the number of

"ones" in strings of binary data. These simple processes proved too time-consuming when simulated by a GEORGE program and it was decided to construct DAPHNIS as a hardware alternative. DAPHNIS operated on blocks of data eliminating the word at a time constraint imposed by the programmed simulation.

#### *A List Processing Machine [35]*

In 1964 a machine was designed at Argonne and built by Control Data Corporation implementing the IPL-V list-processing language. This machine utilized the host CDC 3600 system for memory, input-output facilities, assembling of orders, and programming of complex list-processing instructions. The IPL-V processor, known as Engine No. 2, was connected to the 3600 memory directly and to the 3600 arithmetic unit via the system's satellite coupler and a 24-bit data channel. IPL-V instructions could be executed in the Engine No. 2 circuitry, in the 3604 arithmetic unit, or as programmed subroutines. The intent was to supplement the arithmetic processing capability of the general-purpose computer by developing a more efficient processor capability for non-arithmetic requirements. Little use was made of this machine.

#### *Optoelectronic Information Storage*

During 1966 and 1967 a project to investigate the feasibility of holographic memories was carried out. Effort was devoted to utilizing the coherent light from a laser beam to make holograms, 2 mm on a side, containing a  $64 \times 64$ -bit array of information; these to be combined in a  $64 \times 64$  array on an emulsion-covered glass plate about 5 in. square, yielding information storage exceeding 16 million bits per plate. Since this plate preparation would require approximately 4 hr, use as READ-ONLY storage was predicated. Reading of the stored information was to be accomplished in about 4  $\mu$ sec by switching an illuminating laser beam to the hologram using a series of optoelectronic switches and refractive crystals. Once selected and illuminated, all 4096 bits are available.

#### *Communications Systems*

In 1969 the Laboratory put into operation the remote access data system, RADS, to allow remote entry of jobs for batch processing on the IBM 360 system (Fig. 7). Fourteen stations were developed, thirteen at the Argonne site and one at the National Accelerator Laboratory approximately 20 miles away; as many as 15 can be accommodated.

The basic terminal consists of a Varian 620I control computer with 4096 16-bit memory, a General Design 600 card-per-minute reader, a 300 line-per-minute POTTER printer, a 625 character-per-second Chalco paper-tape reader, with optionally a 240 character-per-second Teletype paper-tape punch, Calcomp plotter facility, or higher speed (1000 per minute) Mohawk Data card readers and printers. All on-site terminals use standard telephone lines with Argonne-developed modems operating at 5000 or 20,000 B. The NAL station uses a Western Electric 203A modem at 4800 B.

A 16,384 word Varian 620I serves as the message switching computer (MSC), directing the data between the remote station and the IBM 2701 Data Adapter Unit on the central system's IBM 360/50 multiplexor channel. Parallel 16-bit data transfer occurs between the MSC and the IBM system; bit-serial transfer occurs between the



**Fig. 7.** A RADS, remote access data system, terminal in the laboratory's Solid State Science Division.

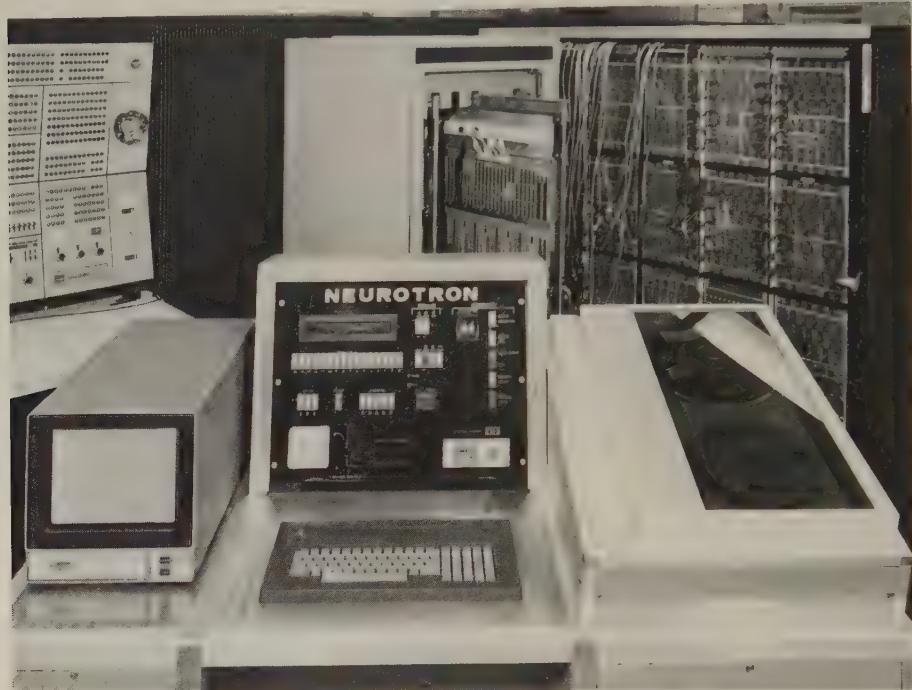
MSC and remote stations. The MSC memory accommodates the RADS control program, a diagnostic program, and 15 1600-byte data buffers.

The remote station's computer memory is allocated between the RADS control program and a 1600-word data area utilized as two 800-word (1600-byte) buffers. When an error is detected in transmission the entire buffer is retransmitted. Cards may be listed at the remote station or read as input to the 360 system; paper tape input can be input to the 360 also. The 360 system output data sets may be directed to either a station printer, Calcomp plotter, or paper tape punch.

#### *A Hardware Monitor System [36, 37]*

NEUROTRON, a hardware monitor system, designed and built around a minicomputer and random access memory (RAM) with associated arithmetic unit, was developed to dynamically collect, display, and record performance data obtained by probes attached

to host laboratory computer systems. It was designed to accommodate IBM 360 model 75 data rates (Fig. 8). The minicomputer is a Varian 620I which coordinates operation of the storage display, keyboard, and magnetic tape I/O facilities, algorithm selection



**Fig. 8.** The NEUROTRON Hardware Monitor System console keyboard, storage display, and magnetic tape facility showing probes attached to the host IBM/360 Model 75.

for the RAM, programmed logic, counters, and sequences. A 36 by 24 patch board programmer is included to facilitate selection and control of NEUROTRON elements.

The performance data are being used to identify program and system parameters which can be studied and evaluated to provide insight into modifications or changes required to optimize the performance of the host computer system. Studies have been completed analyzing the RADS performance and simulating the IBM cache memory organization using different replacement algorithms.

### Engineering Research

#### *The Braille Machine* [38, 39]

A Braille reader, about the size of a portable typewriter, has been developed and models are being built for testing with blind subjects. The machine, designed as a low-cost package, consists of a  $\frac{1}{2}$ -in. magnetic tape record and playback unit, a Braille media

belt and transport, and associated electronics. Information stored as properly formatted Braille characters on tape is played back and converted into embossed symbols on the reusable plastic belt. The blind user reads the information by touching the moving belt with his fingertips. Movement of the magnetic tape and the Braille belt is synchronized and controlled electronically. The user can vary the speed of the belt to match his reading speed. After the symbols move beyond the reader's fingers, the device "erases" the embossed symbols by depressing them and new material is embossed.

With a plug-in keyboard and the Braille machine, the user can record his own notes or letters onto magnetic tape for playback on a Braille machine.

Prototype models have been constructed and tested; 30 will be used in the field testing. A computer program to convert written information into Grade II Braille and to prepare the properly formatted audio tapes is being developed simultaneously.

### *Microcontrol, Microprogramming, and Processor Design*

Work during the past two years in the area of processor design for microprogrammable machines resulted in development of the Argonne Micro Processor, AMP. AMP was used as a tool for processor design research and in microcontrol studies and micro-programming language development efforts. The machine employs a minimally encoded microcontrol word, a flexible multiple-bus structure, high-speed local storage, several arithmetic/logic units, and completely asynchronous memory referencing.

### *Graphics Development [40, 41]*

Beginning in 1968 a study of computer-associated interactive graphic systems was undertaken. The Argonne Interactive Display system (AID), consisting of an IBM 1130 computer, a refresh disk, display controller and several raster scan (TV) displays, was constructed to provide computer-generated low-cost TV images in full color with no flicker.

In 1971 AID was dismantled and portions of the system incorporated into the MIRAGE microprogrammable interactive graphics system. MIRAGE consists of a modified PDP 7, the AMP microprocessor, the color TV displays, and associated peripherals including a 7-track tape unit and a hard-copy device. A virtual display processor has been defined on AMP which can accommodate a wide variety of display list formats. The processor has list-processing instructions for such things as automatic stack manipulation, character string generation, tracing, and masking, which augment the instruction repertoire of the PDP 7.

## COMPUTER SCIENCE RESEARCH

The construction of general-purpose digital computers and the development of special-purpose hardware has been accompanied by parallel and supporting efforts in

such areas as the construction of language processors, theorem proving and numerical methods research, and the development of installation program libraries and operating systems support.

### **Language Processors**

#### ***ANL Machine Support [7]***

All software support for the early ANL machines, AVIDAC, GEORGE, and FLIP-GUS, was developed in-house, including such items as floating-point arithmetic systems, relocatable loaders, subroutine libraries, limited operating systems, utility routines, and the hardware diagnostic programs required for design and maintenance.

A rudimentary operating system on the AVIDAC provided dynamic storage allocation of sorts, and input-output and error-handling facilities for the reactor design programs. Assemblers were written for both the GEORGE and FLIP computers. The ARGUS processor designed for GUS incorporated as major components FLARE, the FLIP assembler; a FORTRAN-like I/O statement processor; a microinstruction language processor; and a NELIAC-type algebraic language processor. The GUS operating system included GAR, the GEORGE assembler; the FLIP input-output package executed by GEORGE but called from FLIP via statements produced by the ARGUS I/O statement processor; the interrupt control programs for FLIP and GEORGE, and the OMNI data-handling package executed by GEORGE but invoked by GEORGE or FLIP to handle requests for transfer of data between core memory units and auxiliary storage devices such as the wide magnetic tape system.

#### ***A FORTRAN Preprocessor***

Not long after the IBM 704 was installed, the initial version of FORTRAN became available. By the time the machine's core memory had been increased to 32,768 words and the IBM 1401 had been acquired to replace off-line card-to-tape and tape-to-printer facilities, a large portion of the Laboratory's predominantly scientific and engineering computations were being programmed in FORTRAN. To more efficiently handle the ever-increasing FORTRAN load on the 704, a preprocessor was written for the 1401. This preprocessor, DDT, checked the programmer's FORTRAN source statements to detect and list programming errors. It was standard procedure to "check out" all FORTRAN compilations with DDT prior to entering them in the main processor's input queue.

#### ***COGENT, a Compiler and Generalized Translator [42]***

During the CDC 3600 era, the COGENT programming system was designed and implemented. The COGENT input language was tailored to the description of symbolic or linguistic manipulation algorithms. Although primarily intended for use as a compiler it was also applicable to problem areas such as algebraic manipulation, mechanical theorem-proving, and heuristic programming. The two major objectives of the system

were to combine the programming conciseness of a syntax-directed compiler system with the generality of a recursive list-processing program, and to produce a compiled program free of time-consuming interpretive operations.

The language consisted of productions, which defined the object language syntax, and generator definitions, which defined the list processing procedures, called generators. COGENT compiled its input language into 3600 assembly language.

### *General-Purpose Programming Language Research [43]*

GEDANKEN, a simple idealized programming language was defined at Argonne in 1968, based on two principles. One is a formalization of addressing or "naming"—a precise distinction between a value and an object which takes on values—and the second is a rule that any value permitted in some context in the language must be permitted in any other meaningful context.

EPS, or extensible programming system, is a general-purpose programming system being developed to enable a programmer working within the language to define new language elements. Plans call for the language to have an extended PL/1-like syntax but differ in the treatment of declarations. Programs written in EPS would be translated into PL/1 programs. An illustrative extensible language based on a PL/1 subset was designed and implemented in a limited manner on the 3600.

### *Processors for Other Machines*

A continuing development effort involves the use of the larger computer with its more complete peripheral equipment support to produce the machine-language code required for stripped-down control or one-of-a-kind special-purpose systems. The initial effort in this area was the construction of a CHLOE assembler, TCA, on the CDC 3600 system. The paper tape output of this assembler served as the program tape, complete with loader, for operation and control of the CHLOE ASI 210 computer, its peripheral equipment, and the attached scanner hardware. The language provided the user the symbolic facilities for assembling the ASI 210 instruction repertoire and macro-instruction capability for control and operation of the scanning equipment.

Later efforts include the construction of a PDP 7 assembler for the CDC 3600, an IBM 1130 assembler on the IBM 360/75 system, and the development of the current ARGOT meta-assembler for the IBM 360 system. ARGOT cross-assembler versions written in PL/1 are now operative for the CDC 160A, and PDP 7, PDP 9, and PDP 11 computers.

### *RESCUE Processors and Compiler*

Two general-purpose processors were implemented on the Laboratory's RESCUE time-sharing system which will be described later. These are DISCOM (direct interaction between scientist and computers), an interpretive computational language compiler

with a BASIC or JOSS-like language; and EDIT, a processor providing facilities to create, edit, and merge card-image files. In addition the RXCOM program was designed and implemented for the construction of RESCUE processor object modules.

DISCOM is a simple scientifically oriented language designed for the casual user, or as a substitute for slide rule or desk calculator computation. It combines the standard replacement, arithmetic and relational operator, input-output, transfer of control, and repetitive execution statements with array specification, function definition and subroutine capability, and includes a library of elementary mathematical function routines. It is an incremental compiler; DISCOM programs are constructed, edited, and executed a statement at a time.

The EDIT processor is generally used to build and maintain JCL (job control language), source program, and data input deck files for batch job execution.

RXCOM, a compiler for the RXPL language designed specifically for the construction of RESCUE processors, is an adaptation of the Stanford University XCOM-XPL system. RXPL contains, in addition to the general data manipulation XPL features, new statement types and built-in procedures required by the RESCUE environment such as console reading, prompting, and output, RESCUE file I/O, command scanning and attention handling. RESCUE processors have been written in RXPL to serve as editing preprocessors for special applications in biology, radiological physics, and numerical mathematics.

### **SPEAKEASY [44]**

A second scientific-user language developed at the Laboratory is known as SPEAKEASY. It was originally conceived to assist physicists working on shell-model physics computations. It provides the user with flexible data storage via incorporated dynamic storage allocation procedures, comprehensive function and subroutine library facilities for performing the more frequently needed mathematical operations on his one- or two-dimensional data arrays, and versatile graphical output capability.

The system was first designed for the CDC 3600 with the attached DD80 CRT output unit, but has been transferred to the IBM 360 system with either keyboard or card input and graphical or numerical output capability.

### *An Interactive Editor [45]*

EMILY, a syntax-directed interactive editor, was designed to assist PL/I users developing their programs at the IBM 2250 graphic display unit as part of a project investigating engineering principles for interactive systems. EMILY presents a portion of the program being developed on the display as a sequence of terminal and nonterminal symbols. For any selected nonterminal symbol, EMILY can then display a set of syntax-allowable replacements from which the programmer can select the desired one by application of the light pen. Identifiers, constants, and comments are entered from the console keyboard. The facility to move, replace, or copy structural units is available, and EMILY was constructed to accommodate additional languages.

### Theorem Proving [46–49]

Automated theorem proving is concerned with obtaining proofs of theorems from various fields of mathematics with the aid of a computer. It is hoped that many theorems whose proofs are tedious and time-consuming for humans to develop can be left to computers. Refutation procedures, whereby a purported theorem is assumed false and proof is obtained by contradiction, are employed in theorem proving by computer. Theorem proving research is the basis of many question-answering computer systems.

Work in this area has been carried out on the IBM 704, CDC 3600, and IBM 360 systems. Set-of-support strategies have been formulated and employed to make it easier for the computer to search for proof of a theorem by prohibiting the computer from considering certain paths of reasoning. This allows the machine system to prove theorems of greater depth. The current theorem proving system, based on the RW1 program, provides remote terminal facilities for testing combinations of strategies and inference rules to determine their effect upon proof search efficiency.

## Numerical Methods and Subroutine Library Development

### CDC 3600 [50–55]

When the laboratory shifted to commercial computers for its programmatic scientific and engineering computations, the numerical methods and subroutine development efforts shifted as well. When the CDC 3600 was delivered, a subroutine library accompanied the machine but there was no documentation. A project was immediately undertaken to test and document the manufacturer-supplied software, replacing with locally developed routines those which the testing indicated could be improved. In addition, plans were made to develop the necessary adjunct routines not provided by the manufacturer but essential to Laboratory programs. A standard subroutine library format was developed for documentation. Routines were developed, tested, and documented for the Bessel functions, complete elliptic integrals, the error function, gamma and incomplete gamma function, Coulomb wave and Fermi-Dirac functions.

A FORTRAN-compatible triple-precision package was prepared to assist in the subroutine development effort and associated numerical methods research. Other areas studied were numerical quadrature, solution of linear and differential equation systems, eigenvalue-eigenvector determination, and analysis and fitting of experimental data.

### IBM 360 [56–60]

The development, testing, and documentation of the necessary software tools for scientific problem solving continued with the acquisition of the IBM 360 system. The manufacturer-provided FORTRAN subroutine library routines were subjected to extensive testing for accuracy, timing, and error facilities, and the results of this testing documented. As later FORTRAN library and PL/I library releases were made available, these

routines too were tested and the results published. Other activity included the development of an efficient accurate method for exponentiation, minimax approximations for the exponential integral and the psi, or digamma, function, and subroutines for Dawson's integral, the Riemann zeta function, and the Coulomb wave function and phase shift.

### *An Eigensystem Package*

FORTRAN routines for numerical solution of the eigensystem problem, collectively called EISPACK, have been derived from the original ALGOL versions published by Wilkinson and Reinsch [87] in 1971. Thirty-four such routines have been prepared, tested, and documented, with versions for a number of machine systems made available. A control program, EISPAC, designed to assist the user in his application of the package, has been implemented on the IBM 360 system. This program is divided into a decision phase which interprets the user-supplied parameters to determine the appropriate eigensystem routines to be called, and an execution phase which calls the selected routines in order, transmitting the data required from one to the next. These phases communicate by means of a vector of logical values which determine the use of the individual eigensystem routines.

## **Operating Systems**

### **OOPS**

The first resident monitor, or operating system, employed at Argonne was the IBM 704 oops (Our OPerating System), installed in 1962 in an effort to alleviate the over-crowded conditions on the machine. oops was developed from an early version of the University of Michigan's Executive System with the basic requirement that it have minimum impact on the existing user-program environment. It was designed to reside completely on the 704 drum, if necessary. The core resident could be flushed as required when the user-job demanded full-core facility, and brought in by operator intervention in the event such full-core jobs aborted. The IBM 1401 served as a support processor, preparing the system input tapes and printing and punching oops output tapes. This tape-oriented system helped delay the saturation point for the IBM 704, and use of the system continued through fiscal 1966, almost three years after the CDC 3600 installation.

### ***SSP, SATCOPS and CDC 3600 Support [61]***

Much of the system software for the laboratory's CDC 3600 system was developed locally. The system, as purchased, included in addition to the 65,536 word 2-bank CDC 3600, four CDC 160s; two at the central facility and one each at two remote locations.

At the central location one 160A served as an off-line unit, the other as a satellite processor. A peripheral processor input-output package was developed for the off-line 160A to allow it to prepare the system input tapes for the CDC-supplied 3600 operating system, SCOPE, and print and punch SCOPE output. At the same time utility tasks such as card-to-printer, card-to-punch, card-to-tape, and tape-to-card, tape-to-printer, and tape-to-tape as well as tape testing could be handled. Up to four tasks could operate concurrently, and the unit handled the microwave communications between the central facility and the remote high-energy physics 160A station.

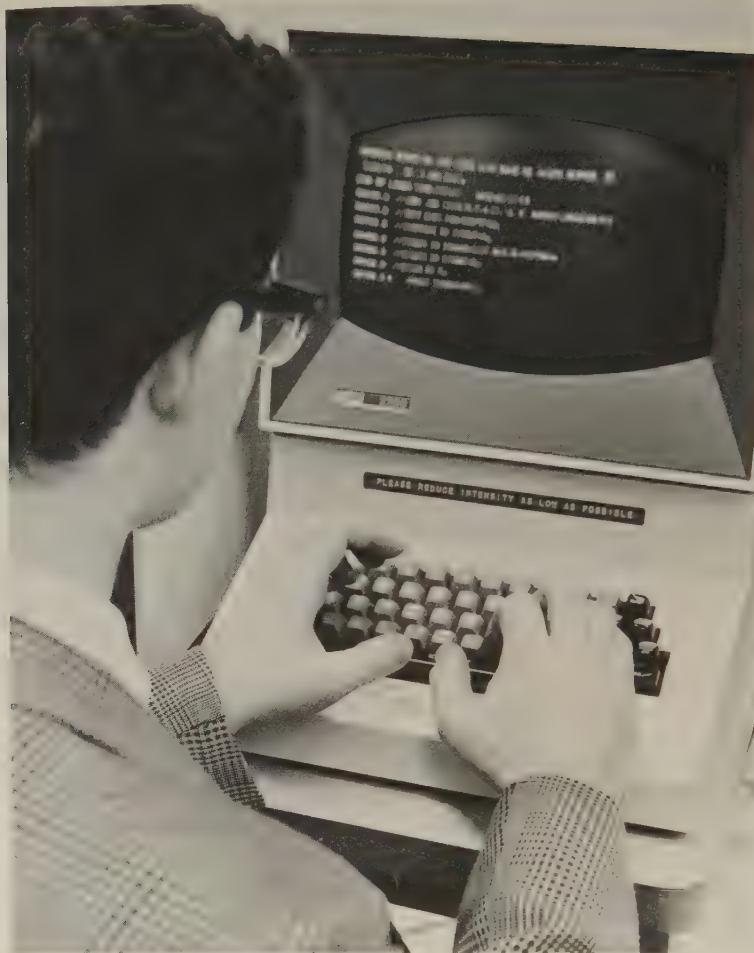
SATCOPS resided on the satellite 160A directly coupled to the CDC 3600. The system provided functions such as job scheduling and I/O device allocation, and controlled card reading and punching and printing for the combined machine system using the large disk file as a shared device. Jobs were scheduled for the main processor from three input queues—two on the disk file and one on tape. SATCOPS was capable of printing output from up to three different jobs while entering a fourth job from a card reader. The SATCOPS disk staging of jobs in input and output queues could be overridden by operator intervention or at user option.

Special software routines were also developed locally to permit programmer use of the large disk for scratch data storage and use of the CDC 3600 DD80A CRT display and film recorder unit.

### *RESCUE*

The conversational computing capability developed for the IBM 360 is called RESCUE, an acronym for RESensitive Computer User Environment, and designed to provide the user with interactive capability for remote job entry, language preprocessing, data storage and editing, and interpretive computation. RESCUE shares the model 50 with ASP, the attached support processor system, and half of the 2-million byte LCS (large capacity storage) is dedicated to RESCUE use. RESCUE programs execute from a 200K partition of the model 50's 512K core with system residence, a 2314 disk pack. A second 2314 and the data cells accommodate OS datasets used both by RESCUE and the model 75 batch system. Three types of terminals are supported; a limited number of IBM 2260 graphic display consoles over direct lines (Fig. 9), and teletypewriters and IBM 2741's on a dial-up basis, with up to 29 active at any time.

The system is made up of three major components—the control program, the subsystems or processors, and the user-system interface functions. The control program incorporates the time-sharing, program loading, and storage allocation functions, as well as the direct-access storage and terminal input-output drivers. The processors provided include DISCOM, and EDIT described previously and BATCH, a facility which allows the user to combine files built by the processors and submit them to the OS 360/75 as an OS job. In addition to the processors, RESCUE has service routines for transmitting messages to other terminals; allowing or inhibiting receipt of such messages; renaming, printing, or punching a RESCUE file; listing files, answering inquiries on the queue status of OS jobs, raising the priority of OS jobs, cancelling OS jobs, and providing assistance or explanation of system facilities or commands.



**Fig. 9.** An IBM 2260 Display Station being used with the RESCUE EDIT processor.

#### *Region Management of On-Line Direct-Access Devices*

A system has been implemented to assign ownership status to less than volume amounts of on-line direct-access devices. Ownership status guarantees reserved storage space to the owner and restricts allocation of such space to owner-assigned activities. Only the region owner may delete datasets stored within his region. Utilities have been written to provide information to owners on the amount of space available in an owned region and the activities using that region and to enumerate all owned datasets with their characteristics. The Computer Center can, with these facilities, charge for storage space allocated and monitor the amount and use of available on-line storage space. These usage statistics are made available to the region owner as well to assist him in making more effective use of this resource.

### *Computer Resource Accounting*

A comprehensive resource accounting system was implemented in 1972 which identifies resource usage and the cost to the user for each job processed and, in addition, provides monthly and fiscal year accounting summaries for particular activities and for administrative management.

### **System Performance and Monitoring**

Software monitoring and benchmarking efforts have been undertaken, along with the design and construction of the NEUROTRON hardware monitor. These have been carried out to assist in computer acquisition and configuration enhancement and to provide insight for system tuning to optimize system performance.

A Dynamic Status Recorder (DSR) system was developed and implemented within the ASP supervisor that controls the scheduling and peripheral I/O, whereby a magnetic tape containing elements defining the chronological sequence of events as they occur is written for analysis and study of the ASP system. Several samples have been recorded and analyzed.

In addition, a general analysis program has been written for use with the IBM 360 OS software monitor facility (SMF).

## **APPLICATIONS**

The direction and content of laboratory computing over the past two decades has been determined largely by the research and development program it supports. These fall predominantly in the major areas of USAEC funding mentioned in the first section, namely, reactor sciences, physical sciences, and the life and environmental sciences. Consequently, computer contributions in their most direct form, applications, are presented in this framework.

### **Reactor Sciences**

#### *The ARC (Argonne Reactor Computation) System [62, 63]*

The development of a comprehensive modular program system for reactor computation was begun at Argonne in 1965 to provide a single unified system to support algorithm and code development efforts along with standard reactor physics computational requirements. The ARC system consists of system modules which serve as an interface between the ARC system and the IBM 360 computer system environment, computational modules which are conventional program routines, the datapool containing the data organized in datasets, each generally containing a particular category of information,

and the standard "paths," or control programs, which establish the linkage of computational modules to solve a particular reactor problem. Computational modules exist to handle functions such as neutronics calculations, cross-section preparation, fuel cycle and depletion studies, adjunct computations, and safety or accident analysis calculations. Standard paths have been constructed to solve one- and two-dimensional diffusion and transport theory design problems, one- and two-dimensional diffusion theory and perturbation calculations, two-dimensional spatial synthesis studies, fuel cycle computations, etc. In addition, the ARC system allows the user to catalog datapool datasets and to link computational modules at his discretion in developing new applications or modules and testing algorithms.

### *Reactor Safety Studies and Accident Analysis [64-68]*

A number of large complex computer programs have been designed and implemented in support of LMFBR reactor safety studies. As these systems were developed several modules have been prepared and incorporated in the ARC system. During 1971 in safety studies associated with the Fast Flux Test Facility, FFTF, Argonne codes were used to analyze the effects of a complete failure of all FFTF coolant pumping capacity coupled with simultaneous failure of all safety shutdown equipment. The course of the postulated accident was followed in detail with the SAS2A transient analysis code treating the onset and early phases of the accident and the VENUS code determining the energy yield in the reactor disassembly. REXCO-H was then used to determine the pressure loadings on the primary containment vessel resulting from the disassembly.

### *Fuel-Element Performance [69, 70]*

During reactor operation fuel fission products build up pressures that can deform and rupture the cladding that isolates the fuel from the flowing coolant. The cladding material in a fast-reactor environment also swells at a rate dependent on the temperature and the amount of irradiation. Removal and replacement of deformed and ruptured fuel elements is a costly, time-consuming operation. The fuel-element performance and lifetime depend on such factors as cladding and fuel materials, fuel form (powder or pellet), cladding diameter and thickness, fuel density, spacing between fuel and cladding, power rating of the fuel element, coolant temperature, and fission-gas venting technique. Several computer programs have been developed in an effort to translate fundamental material-behavior models into analytical LMFBR fuel-element performance projections.

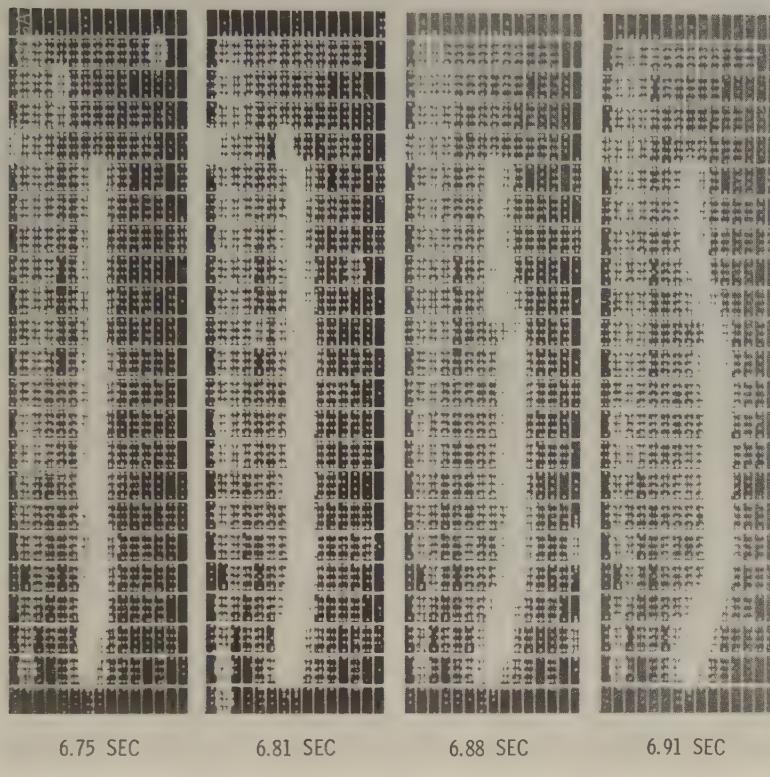
The BEMOD code simulates behavior of sodium-bonded metallic fuel such as that used in Argonne's Experimental Breeder Reactor II, EBR-II. It has been employed to predict EBR-II fuel performance at higher operating power and to identify changes necessary in the design of the metallic fuel elements to attain increased element lifetimes. SWELL and LIFE were written to simulate behavior of the gas-bonded uranium-plutonium oxide fuels being designed for LMFBR power plants. SWELL, intended for large parametric surveys such as power-plant optimization studies, is based on average operating

conditions. LIFE, and in particular LIFE-II, is based on a more rigorous analysis incorporating fuel-swelling and gas-release models.

### *The TREAT Hodoscope*

The fast neutron hodoscope at the transient reactor test facility, TREAT, in Idaho, records "pictures" of fuel elements undergoing destructive testing. Fast neutrons from

TRANSIENT NO. 1255



**Fig. 10.** Results from a fuel irradiation in TREAT recorded with the fast-neutron hodoscope. These are four images of the transient selected from a sequence of several thousand produced.

fissions in the element pass through a collimator and cast an image of the element upon an array of 334 fast neutron detectors (Fig. 10). Typical experiments are as short as a few hundred milliseconds or as long as 30 sec. The system records digital data onto film with a high-speed camera capable of providing time resolution of 1.2 msec. These photographic images are then scanned by the ALICE system and recorded as digital data on magnetic tape. Further PDP 10 processing restores the original count rate information

which is then either plotted as a function of experiment transient time on an SEL 840, or displayed on a PDP 11 terminal for study.

### Physical Sciences

#### *Accelerator Control, Magnetic Field Measurement, and Magnet Design [71-74]*

Automation of the control for the Zero Gradient Synchrotron, ZGS, has been under development since 1965. A CDC 924A computer operates on-line executing from



**Fig. 11.** The CDC 924A computer at the laboratory's Zero Gradient Synchrotron.

memory any of up to ten operating-mode programs. These operating-mode programs specify the values of some 300 tuning parameters which are transmitted via a hardware programmer as timing signals and level settings to the appropriate component apparatus (Fig. 11).

The accelerator can be operated in any of the programmed modes for a specified duration, or in different modes on successive pulses which occur at approximately 2.5-sec intervals. Sensing and monitoring equipment at the ZGS reads and records about 350 performance measurements on command from the control computer.

Besides controlling the operating mode of the ZGS and the monitoring of the ZGS operation, the on-line computer can be used for data analysis. Two feedback control experiments have been completed successfully. One measured the slope of the flattop magnetic field and adjusted the rectifier firing angles to minimize the slope. The other optimally adjusted the position of the shorting stub in the linac R-F system.

At the ZGS also, an SEL 810A computer with an 8K 16-bit word memory disk storage unit, card reader, line printer, paper tape I/O, graphic display unit, and Teletype console has been used on-line with a high-speed data acquisition system for steady-state or pulsed magnetic field measurements and off-line for designing superconducting coil, multipole magnets.

### *High-Energy Physics [75]*

Wire spark chambers have been coupled directly to small general-purpose computers at the ZGS to provide on-line data collection and analysis facilities together with operational monitoring of the experiment. Three ANL computers have been acquired for use with wire spark chamber and counter experiments. The first two, acquired in 1967 and 1968, are EMR 6050 machines each with 16K 24-bit words of memory while the third acquired in 1969 is a SIGMA-2 computer with a 16K 16-bit word memory. Other machines are provided by university groups; generally there are six on-line experiments of this type in operation simultaneously.

A typical experiment utilizing such equipment was a 1968 Argonne–University of Michigan study of the reaction between positive pi mesons and protons leading to production of positive sigma particles and  $K$  mesons. The wire chamber signals were read out through magnetostrictive delay lines and an electronic interface to the computer where each particle event was processed as it took place. Up to 20 events could be processed per ZGS pulse. The computer program was designed to provide equipment performance data such as chamber efficiency, position resolution, and sparks per gap, as well as the physics information on angular distributions, polarization of sigma particles, and the origin of events relative to the hydrogen target.

During 1971 an EMR 6050 computer was used with an array of wire spark chambers, serving as particle detectors, and a large wide-aperture magnet, for determining the momentum of the particles, in studies of meson production and neutral hyperon-associated production.

### *Chemistry [76–78]*

Theoretical chemists have relied on the large storage, high-speed computational power of the digital computer since IBM 704 days in their efforts to construct an adequate model of molecular structure. Early work on simple molecules such as calcium oxide, the alkali hydrides, and compounds containing two alkali metal atoms, such as NaK, was carried out on the CDC 3600 using the attached film recorder or an off-line graph plotter to present accurate representations of the electron densities based on the precise calculations of the molecular orbital model. Later efforts resulted in improvements to the model and its extension to larger chemical systems. During 1970 this

mathematical modeling capability was applied to the simulation of the lithium-hydrogen chemical reaction.

### *Physics*

Computer simulation techniques utilizing the computer system's computational strength in conjunction with its graphic output capability have also been applied in physics studies of magnetic material. In these studies visual displays of magnetic



**Fig. 12.** Computer simulation of a magnetic material—a physics application.

material on the CRT screen illustrate to the viewer how the magnetization pattern changes with time, and how it is affected by temperature and applied magnetic field variation (Fig. 12).

### **Life and Environmental Sciences**

#### *Biological Image Processing [79-85]*

Many of the biological computer applications fall in the area of image processing. The earliest of these was the project to automate the analysis of metaphase chromosome

images using the CHLOE hardware. In this application 35-mm film photographs of metaphase spreads made at the microscope were scanned and digitized by the CHLOE machine with the computer program setting the machine parameters, directing the scan operation, and determining shape characteristics before recording the partially analyzed data on magnetic tape for transfer to a larger computer and further processing. In the larger computer each chromosome was identified and associated with a point in 7-dimensional space, using the theory of algebraic invariants. A separation distance, or similarity measure, was calculated for all possible pair combinations and a machine karyotype prepared, displayed, and photographed (Fig. 13). An improved version of this CHLOE program is currently operable on the ALICE machine. In addition, a separate interactive chromosome classification system is presently under development.

A second project was aimed at precise densitometry of bone autoradiographs and microradiographs. In these studies the image-processing program computed the density spectrum of the plate and constructed density contour maps defining the dose distribution in the bone tissue. Parts of the chromosome program system were used in this application and in a later study of the relative mobility of malignant and nonmalignant cells by analysis of time-lapse photographs.

Projects counting nerve fibers in cross sections of nerve bundles and nerve tissue and counting corneal cells from dogs used in low-level radiation damage studies have been investigated also. The classification of radiographs of carcinomas is a potential application.

### *Simulating Zero Gravity*

Biologists in today's space age have become increasingly concerned with the effect of space on plant behavior. To study this effect it is necessary to duplicate the space environment in the laboratory for experimentally useful periods. The zero gravity condition cannot be duplicated. The effects of gravity and the earth's magnetic field and all other orientation-dependent forces can be removed, however, by continuously rotating the specimen so that equal time is spent in each part of its orientation space, a three-parameter manifold geometrically equivalent to the hypersurface of a hypersphere in four-dimensional space. This can now be done at Argonne with the use of the Argonne-built  $4\pi$  Compensator. This apparatus is controlled by a computer-produced magnetic tape which directs the movement of three gimbals. The tape was produced by a computer program developed to determine the set of uniformly spaced points on the hypersphere and specifies movement through cycles of 3600 orientations.

### *Atmospheric Research [86]*

In 1966 Argonne instituted a program of environmental research utilizing systems analysis methods in the construction of a model to aid in the prediction and planning for control of air-pollution incidents in the city of Chicago. As a part of this modeling effort a computerized information and computation system for environmental studies

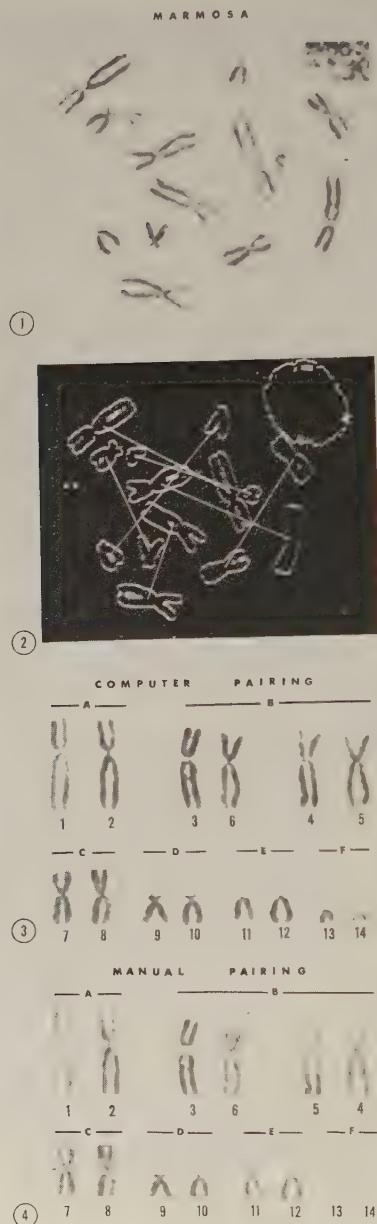
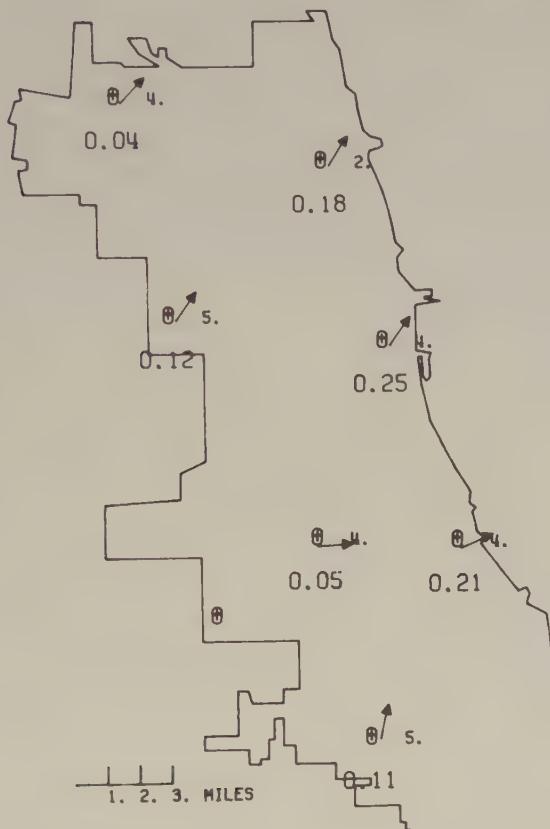


Fig. 13. A comparison of computer karyotyping (CHLOE) and manual karyotyping (cytogeneticist) of a marmosa chromosome spread.

was implemented (Fig. 14). This system, although general in concept, addresses the two basic requirements of environmental studies—the retrieval of data from a large file and the application of statistical analysis to the extracted data. The APICS system includes, in addition to the standard file maintenance facilities, printed or graphical



**Fig. 14.** Computer plot showing data recorded by Chicago's telemetered air quality monitoring stations. Wind direction and speed, in knots, and sulfur dioxide concentrations, in parts per million, of air are shown.

report-writing capability, a multivariate regression analysis package, and a stepwise discriminant analysis program.

#### ACKNOWLEDGMENT

Much of the material in this article has been taken from annual reports of the Argonne National Laboratory and from internal technical memoranda, AEC Computer Information Meeting activity reports, budget and summary reports of the laboratory's

Applied Mathematics Division. Additional information on many topics is available in the references cited. All references are to available documents as open-literature publications.

## REFERENCES

1. M. Butler, Computers and the nuclear industry: A historical background, *Nuclear News* **11** (4), 26-30 (1968).
2. M. K. Butler, Reactors and computers at Argonne: The application of computers to reactor problems over the past two decades, *Argonne Nat. Lab. Rev.* October 1969.
3. J. C. Chu, *Design of a computer—Oracle*, Argonne Nat. Lab. Rept. ANL-5368, Sept. 1954.
4. *ONR Digital Computer Newsletter* **8**, 1 (1956), in *Ass. Comput. Mach. J.* **3**, 44-47 (1956).
5. L. Kassel, GEORGE programming manual, Argonne Nat. Lab. Rept. ANL-5995, May 1959.
6. H. L. Gray and C. Harrison, Jr., Normalized floating-point arithmetic with an index of significance, *Proc. EJCC*, 1959, pp. 244-248.
7. W. F. Miller and R. Aschenbrenner, The GUS multicomputer system, *IEEE Trans. Electron. Comput.* **EC-12**, 671-676 (1963).
8. R. H. Vonderohe and D. S. Gemmell, A description of the PHYLIS on-line computing system, in *Automatic Acquisition and Reduction of Nuclear Data, Proceedings of an EANDC Conference, Karlsruhe*, 1964, pp. 156-162.
9. W. F. Miller, M. A. Fisher Keller, and K. Hillstrom, The PHYLIS executive program, in *Automatic Acquisition and Reduction of Nuclear Data, Proceedings of an EANDC Conference, Karlsruhe*, 1964, pp. 475-482.
10. D. S. Gemmell, Data-handling programs in use with the PHYLIS computing system, in *Automatic Acquisition and Reduction of Nuclear Data, Proceedings of an EANDC Conference, Karlsruhe*, 1964, pp. 483-490.
11. D. S. Gemmell, Experience with a multi-computer-multi-experiment system, in *Proceedings of the Skytop Conference on Computer Systems in Experimental Nuclear Physics*, CONF-690301, October 1969, pp. 37-51.
12. J. L. Yntema, Argonne computer-controlled scattering chamber, in *Proceedings of the Skytop Conference on Computer Systems in Experimental Nuclear Physics*, CONF-690301 October 1969, pp. 321-324.
13. R. H. Vonderohe and R. A. Aschenbrenner, *A Mossbauer-effect data-collection system*, Argonne Nat. Lab. Rept. ANL-7386, November 1967.
14. R. H. Vonderohe and R. A. Aschenbrenner, Computers and devices for specific data acquisition functions, in *Proceedings of the Skytop Conference on Computer Systems in Experimental Nuclear Physics*, CONF 690301, October 1969, pp. 380-390.
15. W. Miller, J. Meadows, A. Smith, and J. Whalen, On-line data acquisition with feedback to accelerator control—A status report, in *Automatic Acquisition and Reduction of Nuclear Data, Proceedings of an EANDC Conference, Karlsruhe*, 1964, pp. 222-230.
16. J. W. Meadows, J. F. Whalen, and A. B. Smith, Extensive experience with an on-line computer in a nuclear laboratory, in *Automatic Acquisition and Reduction of Nuclear Data, Proceedings of an EANDC Conference, Karlsruhe*, 1964, pp. 132-139.
17. J. F. Whalen, A computer-controlled display system for nuclear experiments, in *Automatic Acquisition and Reduction of Nuclear Data, Proceedings of an EANDC Conference, Karlsruhe*, 1964, pp. 399-405.
18. J. F. Whalen, A completely automated Van de Graaff accelerator, in *Use of Computers in Analysis of Experimental Data and the Control of Nuclear Facilities*, CONF-660527, May 1967, pp. 37-41, 1967.
19. J. F. Whalen, A fast neutron automated data acquisition system, in *Proceedings of the Skytop Conference on Computer Systems in Experimental Nuclear Physics*, CONF-690301, October 1969, pp. 77-89.

20. R. Aschenbrenner, Computer-controlled diffractometer equipment, in *Use of Computers in Analysis of Experimental Data and the Control of Nuclear Facilities*, CONF-660527, May 1967, pp. 67–76.
21. M. H. Mueller, L. Heaton, and L. Amiot, A computer controlled experiment, *Res./Develop.*, 34–37, August 1968.
22. C. E. Cohn, Automated data analyses and control for critical facilities, in *Use of Computers in Analysis of Experimental Data and the Control of Nuclear Facilities*, CONF-660527, May 1967, pp. 49–66.
23. C. E. Cohn and R. Gold, Computer-controlled microscope for automatic scanning of solid-state nuclear track recorders, *Rev. Sci. Instrum.* **43**, 12–17 (1972).
24. R. Gold and C. E. Cohn, Analysis of automatic fission track scanning in solid-state nuclear track recorders, *Rev. Sci. Instrum.* **43**, 18–28 (1972).
25. J. W. Butler, Automation of experimental science, *J. Data Management* **3**, 32–39 (1965).
26. R. Clark and W. F. Miller, Computer-based data analysis system, in *Methods of Computational Physics*, Vol. 5 (B. Alder, ed.), Academic, New York, 1966, pp. 47–99.
27. D. Hodges, Photographic scanning systems in the applied mathematics division, ANL, in *Use of Computers in Analysis of Experimental Data and the Control of Nuclear Facilities*, CONF-660527, May 1967, pp. 177–185.
28. J. W. Butler, J. R. Haasl, D. Hodges, B. Kroupa, C. B. Shelman, and R. H. Wehman, ALICE—A general-purpose image processing system, in *Proceedings of Information Colloquium on Polly and Polly-Like Devices*, CONF-720111, January 1972, pp. 149–159.
29. R. Barr *et al.*, POLLY I: An operator-assisted bubble chamber film measuring system, *Rev. Sci. Instrum.* **39**, 1556–1562 (1968).
30. W. W. M. Allison *et al.*, Automatic scanning and measurement of bubble chamber film on POLLY II, in *Proceedings of the International Conference on Data Handling Systems in High-Energy Physics*, CERN 70–21, 1970, pp. 325–353.
31. J. R. Erskine and R. H. Vonderohe, Automatic counting of tracks in nuclear emulsions, in *Proceedings of the Skypoint Conference on Computer Systems in Experimental Nuclear Physics*, CONF-690301, October 1969, pp. 434–444.
32. R. H. Vonderohe and J. H. Doede, Laser-activated ultraprecision ranging apparatus, in *Use of Computers in Analysis of Experimental Data and the Control of Nuclear Facilities*, CONF-660527, May 1967, pp. 73–76.
33. L. Bryant, L. Amiot, and R. Stein, A hybrid computer solution of the co-current flow heat exchanger Sturm-Liouville problem, in *Proc. Fall Joint Computer Conf.*, **29**, 759–769 (1966).
34. J. A. Gregory and G. R. Ringo, A variety of perception for recognition of nuclear events in *Physics Division Summary Report*, ANL-6767, July–August 1963, pp. 26–37.
35. D. Hodges, IPL-VC, *A computer system having the IPL-V instruction set*, Argonne Nat. Lab. Rept., ANL-6888, May 1964.
36. R. A. Aschenbrenner, L. Amiot, and N. K. Natarajan, The NEUROTRON monitor system, in *Proc. AFIPS Fall Joint Computer Conf.*, **39**, 31–37, (1971).
37. L. Amiot, N. K. Natarajan, and R. A. Aschenbrenner, Evaluating a remote batch processing system, *Computer* **5**, 24–29 (September/October 1972).
38. L. Leffler, A. P. Grunwald, and W. P. Lidinsky, The development of a computerized grade II Braille translation algorithm, *1971 WESCON Technical Papers*, 30/3, 1971.
39. A. P. Grunwald, A Braille-reading session machine, *Science* **154**, 144–146 (1966).
40. W. P. Lidinsky, MIRAGE, A microprogrammable interactive raster graphics equipment, in *Proceedings of the IEEE International Computer Society Conference*, Boston, September 22–24, 1971, pp. 15–16.
41. W. P. Lidinsky and J. A. Becker, A computer-driven full-color raster scan display system, *IEEE Trans. Broadcast Telev. Receivers* **BTR-18**, 26–31 (1972).
42. J. C. Reynolds, An introduction to the COGENT programming system, in *Proceedings of the Association for Computing Machinery 20th National Conference*, 1965, pp. 422–436.
43. J. C. Reynolds, GEDANKEN: A simple typeless language based on the principle of completeness and the reference concept, *Commun. Ass. Comput. Mach.* **13**, 308–319 (1970).

44. S. Cohen, The DELPHI-SPEAKEASY system I. Overall description, *Comp. Phys. Commun.* **2**, 1-10 (1971).
45. W. J. Hansen, *Creation of hierachic text with a computer display*, Argonne Nat. Lab. Rept. ANL-7818, June 1971.
46. L. Wos, G. Robinson, D. Carson, and L. Shalla, The concept of demodulation in theorem proving, *J. Ass. Comput. Mach.* **14**, 698-709 (1967).
47. L. Wos, G. Robinson, and D. Carson, The efficiency and completeness of the set of support strategy in theorem proving, *J. Ass. Comput. Mach.* **12**, 536-541 (1965).
48. L. Wos, D. Carson, and G. Robinson, The unit preference strategy in theorem proving, in *Proc. AFIPS Fall Joint Computer Conference*, **26**, 615-621 (1964).
49. J. A. Robinson, Theorem proving on the computer, *J. Ass. Comput. Mach.* **10**, 163-175 (1963).
50. W. J. Cody, Double precision square root for the CDC-3600, *Commun. Ass. Comput. Mach.* **7**, 715-718 (1964).
51. Y. Ikebe, A note on triple-precision floating-point arithmetic with 132-bit numbers, *Commun. Ass. Comput. Mach.* **8**, 175-179 (1965).
52. W. J. Cody, Chebyshev polynomial expansions of complete elliptic integrals, *Math. Comput.* **19**, 249-259 (1965).
53. W. J. Cody and H. J. Thacher, Jr., Rational Chebyshev approximations for Fermi-Dirac integrals of orders -1/2, 1/2, and 3/2, *Math. Comput.* **21**, 30-40 (1967).
54. A. J. Strecok, On the calculation of the inverse error function, *Math. Comput.* **22**, 144-158 (1968).
55. W. J. Cody, Jr., Complete elliptic integrals, in *Computer Approximations* (J. F. Hart *et al.*, eds.), Wiley, New York, 1968, Chap. 69.
56. N. A. Clark, W. J. Cody, K. E. Hillstrom, and E. Thieleker, *Performance statistics of the Fortran IV (H) library for the IBM System/360*, Argonne Nat. Lab. Rept. ANL-7321, May 1967.
57. K. E. Hillstrom, *Performance statistics for the Fortran IV (H) and PL/I (version 5) libraries in IBM OS/360 release 18*, Argonne Nat. Lab. Rept. ANL-7666, August 1970.
58. N. A. Clark and W. J. Cody, Self-contained exponentiation, in *Proc. AFIPS Fall Joint Comput. Conf.* **35**, 701-706 (1969).
59. W. J. Cody and H. C. Thacher, Chebyshev approximations for the exponential integral  $E_i(x)$ , *Math. Comput.* **23**, 289-303 (1969).
60. W. J. Cody, K. E. Hillstrom, and H. C. Thacher, Chebyshev approximation for the Riemann zeta function, *Math. Comput.* **25**, 537-547 (1971).
61. G. A. Robinson and R. F. Krupp, *A cathode-ray-tube plotting system for the Control Data 3600 Computer*, Argonne Nat. Lab. Rept. ANL-7121, December 1965.
62. B. J. Toppel, *The Argonne reactor computation system, ARC*, Argonne Nat. Lab. Rept. ANL-7332, November 1967.
63. H. Henryson, II, and B. J. Toppel, User experience with the ARC system, in *Proceedings of the Conference on New Developments in Reactor Mathematics and Applications*, CONF-710302, August 1971, pp. 478-493.
64. D. R. MacFarlane, ed., *SASIA, A Computer code for the analysis of fast reactor power and flow transients*, Argonne Nat. Lab. Rept. ANL-7607, August 1969.
65. F. E. Dunn, T. J. Heames, P. A. Pizzica, and G. Fischer, The SAS2A LMFBR accident analysis code, in *Proceedings of the Conference on New Developments in Reactor Mathematics and Applications*, CONF-710302, August 1971, pp. 120-127.
66. W. T. Sha and T. H. Hughes, *VENUS, A two-dimensional coupled neutronics-hydrodynamics fast reactor power excursion computer program*, Argonne Nat. Lab. Rept. ANL-7701, October 1970.
67. W. T. Sha *et al.*, Two-dimensional fast-reactor disassembly analysis with space-time kinetics, in *Proceedings of the Conference on New Developments in Reactor Mathematics and Applications*, CONF-710302, August 1971, pp. 128-151.
68. Y.-W. Chang, J. Gvildys, and S. H. Fistedis, *Two-dimensional hydrodynamics analysis for primary containment*, Argonne Nat. Lab. Rept. ANL-7498, November 1969.
69. V. Z. Jankus, *BEMOD, A code for the lifetime of metallic fuel elements*, Argonne Nat. Lab. Rept. ANL-7586, July 1969.

70. V. Z. Jankus and R. W. Weeks, *LIFE-I, A FORTRAN IV computer code for the prediction of fast reactor fuel-element behavior*, Argonne Nat. Lab. Rept. ANL-7736, September 1970.
71. R. L. Martin, Automated experimental facilities and progress toward automated control of the ZGS, in *Use of Computers in Analysis of Experimental Data and the Control of Nuclear Facilities*, CONF-660527, May 1967, pp. 77-83.
72. E. C. Berrill, R. D. George, and A. E. Algustyniak, Organization and use of a small-scale computer in on- and off-line magnetic data processing, in *Proceedings of the 3rd International Conference on Magnet Technology, Deutsches Elektronen-Synchrotron DESY, Hamburg*, 1972, pp. 1359-1372.
73. R. J. Lari, A. E. Algustyniak, M. M. Lieberg, and J. K. Wilhelm, An automated quadrupole magnet measuring system, in *Proceedings of the 3rd International Conference on Magnet Technology, Deutsches Elektronen-Synchrotron DESY, Hamburg*, 1972, pp. 1425-1438.
74. R. J. Lari, Small scale computer programs for magnet design, in *Proceedings of the 3rd International Conference on Magnet Technology, Deutsches Elektronen-Synchrotron DESY, Hamburg*, 1972, pp. 175-186.
75. D. R. Rust, On-line computer facilities at Argonne, in *Proceedings of the International Conference on Data Handling Systems in High-Energy Physics*, CERN 70-21, 1970, pp. 525-536.
76. A. C. Wahl, Molecular orbital densities: Pictorial studies, *Science* **151**, 961-967 (1966).
77. A. C. Wahl, P. J. Bertoncini, K. Kaiser, and R. H. Land, *BISON, A FORTRAN computer system for the calculation of analytic self-consistent-field wave functions, properties, and charge densities for diatomic molecules: Part 1, User's manual and general program description*, Argonne Nat. Lab. Rept. ANL-7271, January 1968.
78. G. Das and A. C. Wahl, *BISON-MC: A FORTRAN computing system for multiconfiguration self-consistent-field (MCSCF) calculations on atoms, diatoms, and polyatoms*, Argonne Nat. Lab. Rept. ANL-7955, July 1972.
79. J. W. Butler, A. Stroud, and M. K. Butler, Automatic classification of chromosomes, in *Proceedings of the Conference on Data Acquisition and Processing in Biology and Medicine*, Vol. 3, Pergamon, New York, 1964, pp. 261-275.
80. J. Butler, M. Butler, and A. Stroud, Automatic classification of chromosomes II, in *Proceedings of the Conference on Data Acquisition and Processing in Biology and Medicine*, Vol. 4, Pergamon, New York, 1965, pp. 47-57.
81. J. Butler, M. Butler, and A. Stroud, Automatic classification of chromosomes III, in *Proceedings of the Conference on Data Acquisition and Processing in Biology and Medicine*, Vol. 5, Pergamon, New York, 1968, pp. 21-38.
82. J. W. Butler, M. K. Butler, and B. Marzynska, Automatic processing of 1000 marmoset spreads, in *Proceedings of the Conference on Data Extraction and Processing of Optical Images in the Medical and Biological Sciences*, N.Y. Acad. of Sciences, New York, 1969, pp. 424-437.
83. E. Lloyd, J. H. Marshall, J. W. Butler, and R. E. Rowland, A computer program for automatic scanning of autoradiographs and microradiographs of bone sections, *Nature* **211**, 661-662 (1966).
84. E. Lloyd and D. Hodges, Quantitative characterization of bone: A computer analysis of micro-radiographs, *Clin. Orthoped.* **78**, 230-250 (1971).
85. G. Barski, J. W. Butler, and R. J. Thomas, Computer analysis of animal cell movement in vitro, *Exp. Cell Res.* **56**, 363-368 (1969).
86. C. Chamot et al., *APICS, A computerized air pollution data management system*, Argonne Nat. Lab. Rept. ANL/ES-CC-006, February 1970.
87. J. H. Wilkinson and C. Reinsch, in *Handbook for Automatic Computation*, Vol. II, Springer, New York, 1971.

M. K. Butler

## ARITHMETIC OPERATIONS

Man has always been fascinated by mathematics, but at the same time bored with the drudgery of computation. Thus, all through history we find records of man's attempts to devise computing tools to relieve this drudgery. The cavemen performed simple arithmetic by holding up fingers or using piles of pebbles. From this concept strings of beads as counting devices evolved and this led to the abacus, which was in use in China before 1000 B.C. It wasn't until the seventeenth century, however, that any mechanical devices with significant improvement over the abacus were invented. One reason for this long interval was the use of Roman numerals throughout Europe; Roman numerals increase the complexity of any mechanical computational device by many magnitudes. The gradual acceptance of the Arabic numeral system starting around A.D. 1200 provided a much simpler means of calculation.

In 1617, John Napier, a Scot, invented a device which became known as Napier's bones. Napier had earlier invented logarithms and his device was based on this concept. It was a mechanical arrangement of rods of bone that had numbers printed on them. When the rods were put into the proper combination it was possible to perform direct multiplication. Another similar invention based on logarithms was the slide rule developed by William Oughtred in 1621. The slide rule performs multiplication and division by adding and subtracting logarithms.

There still was a need for a device that could be used to rapidly add large groups of numbers. Such a device was invented by the Frenchman Blaise Pascal in 1642. His machine consisted of a row of wheels, each wheel having ten teeth, and each tooth representing a digit. The first wheel represented units, the second tens, and so on. Each wheel would display a single tooth marked with a digit in windows at the top of the machine. To initially enter a number into the machine the wheels would be turned until the windows displayed the desired digits. A number could be added to this number by moving each of the wheels the number of teeth corresponding to the positional digit. For example, if the number 623 was being added to some previously entered number, the units wheel would be moved three teeth, the tens wheel would be moved two teeth, and the hundreds wheel would be moved six teeth. The carry operation was automatically handled by the machine. Each time the units wheel turned past its nine tooth, it caused the tens wheel to advance one tooth. Whenever the tens wheel turned past its nine tooth it caused the hundreds wheel to advance one tooth, and so on. Subtraction was accomplished by turning the wheels in the opposite direction. The German Baron Von Liebnitz utilized Pascal's techniques and developed a machine in 1672 which could also multiply and divide. The multiplication and division was performed by sequences of addition and subtraction.

Other machines were developed from time to time for the next two centuries. For the most part they utilized Pascal's number wheel concept. None of these early machines received wide acceptance, not because the ideas were wrong, but because the technology of the time was not far enough advanced to produce large numbers of reliable machines. The first machine that was successfully marketed was designed by an American,

William S. Burroughs, in 1884. His machine was keyboard driven and had an important feature of recording the results of the calculations on paper. Electromechanical machines came into use about 1920.

All of these early calculating machines required that the operator intervene at each step of the computation. If ten numbers are being summed the operator enters the first number, he then enters the next number and the machine will form the sum of the first two numbers. The third number is entered by the operator and the machine forms a new sum. This process continues until the final sum is obtained. This is a basic difference between a calculating machine and a computer. With a computer the operator does not need to intervene between each problem step. The entire problem is introduced to the computer in one step, and the computer then proceeds with the calculations without any further intervention until a solution is reached.

Charles Babbage, an Englishman, recognized the improvements that could be made in calculating devices if the necessity of human intervention at each problem step could be removed. His work was most amazing, since he was designing and attempting construction of his machines in the early 1800s, half a century before reliable calculating machines were developed. His analytical engine had the same concepts found in present-day computers. The arithmetic section, or mill as Babbage named it, was comprised of decimal number wheels and gears and operated in much the same manner as Pascal's machine. The problem data were stored in columns of wheels which he called the store. A second type of store kept the problem statement encoded on punch cards. Whenever the mill completed one operation, it was to go to the store for the next instruction and data. Again, the technology of the day was not advanced enough to produce the machine, and it was labeled Babbage's folly by his critics. However, almost all of Babbage's ideas have found their way into modern computers.

The number wheel concept carried over into the arithmetic operations of the early digital computers which appeared in the 1940s. A ten-position stepping relay served the same purpose as the wheel. The relay was activated by electrical pulses instead of gears. Three pulses would cause the relay to move three positions. If it moved past the nine position, a single carry pulse would be generated and transmitted to the next relay.

The use of electronic devices, which have no moving parts, instead of relays increased the calculating speed of computers over 1000 times. Another improvement was the use of the binary number system for calculations instead of the decimal system.

There is no real advantage to the decimal, or base 10, number system that we use for our mathematics. The fact that man has ten fingers was the only reason that ten was selected as the base for our number system. Any arithmetic calculations that we perform in the decimal system can be done in any other number base system. A base 10 number system has ten digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. Any decimal number represents a sum of these digits multiplied by various multiples of ten. For example the number 7356 represents

$$\begin{array}{r}
 6 \times 10^0 = 6 \\
 5 \times 10^1 = 50 \\
 3 \times 10^2 = 300 \\
 7 \times 10^3 = 7000 \\
 \hline
 7356
 \end{array}$$

Similarly, digits to the right of the decimal point would represent by their positions, a product of the digit and ten raised to a negative power. For example, the number 3.279 represents

$$\begin{array}{r}
 3 \times 10^0 = 3.0 \\
 2 \times 10^{-1} = 0.2 \\
 7 \times 10^{-2} = 0.07 \\
 9 \times 10^{-3} = 0.009 \\
 \hline
 & 3.279
 \end{array}$$

The same notation applies to any number base. If we used the quinary, or base 5, system there would be five digits to consider: 0, 1, 2, 3, and 4. In counting, the sequence would be 0, 1, 2, 3, 4, 10, 11, 12, 13, 14, 20, 21, etc. The quinary number 243 represents a sum of various products of the digits times the base of the number system.

$$\begin{array}{r}
 3 \times 10^0 = 3 \\
 4 \times 10^1 = 40 \\
 2 \times 10^2 = 200 \\
 \hline
 & 243
 \end{array}$$

The number 10 used in the quinary system is equivalent to a 5 in the decimal system. Therefore, to convert the base 5 number 243 to a decimal number the calculations would be

$$\begin{array}{r}
 3 \times 5^0 = 3 \\
 4 \times 5^1 = 20 \\
 2 \times 5^2 = 50 \\
 \hline
 & 73
 \end{array}$$

The decimal equivalent to the quinary number 243 is 73. To avoid confusion when writing in various number systems, the number base used is often shown as a subscript to the right of the number, for example, 243<sub>5</sub> or 319.36<sub>10</sub>.

The binary or base 2, number system has only two digits: 0 and 1. The counting sequence is 0, 1, 10, 11, 100, 101, 110, 111, 1000, 1001, etc. The 10<sub>2</sub> in the binary system is equivalent to a 2<sub>10</sub> in the decimal system. Thus, the binary number 11010<sub>2</sub> is equivalent to 26<sub>10</sub>.

$$\begin{array}{r}
 0 \times 2^0 = 0 \\
 1 \times 2^1 = 2 \\
 0 \times 2^2 = 0 \\
 1 \times 2^3 = 8 \\
 1 \times 2^4 = 16 \\
 \hline
 & 26_{10}
 \end{array}$$

The binary number system is utilized very often in computers, since binary numbers are represented by a sequence of only 0's and 1's. This allows numbers to be represented by a series of bistable devices. A bistable (or two-state) device is one in which the device can be in only one of two possible states. The most frequently used example of a bistable device is the electric light bulb. The bulb can either be in the on or the off state. A number wheel in the early calculating machines, or a stepping relay in the first computers, would be examples of ten-state devices. At any time they can be in any one of ten possible states. The desire for more reliability led computer designers to the use of bistable devices. A slight change in the characteristics of a bistable device will usually not affect its performance. A light bulb as it grows older will give off less light, but it is still easy to tell if it is in the on or the off state. The bistable devices in computers may all be slightly different, but even with these slight differences it is still simple to tell the state of each device since there are only two possibilities. This would not be true for ten-state devices.

Many computers perform all of their arithmetic operations entirely in the binary system, and these machines are called binary computers. In order to better understand binary computers an investigation of binary arithmetic might be helpful. As stated previously, any arithmetic operation that is done in the decimal system can be performed in any other number system. The addition of binary digits occurs as shown below.

$$\begin{array}{cccc}
 0 & 0 & 1 & 1 \\
 +0 & +1 & +0 & +1 \\
 - & - & - & - \\
 0 & 1 & 1 & 10 \quad (\text{carry required})
 \end{array}$$

Since the digit  $2_{10}$  does not exist in the binary number system, this quantity is represented by the number  $10_2$ , and thus  $1 + 1 = 10$  in the binary system.

Subtraction of binary digits is as follows.

$$\begin{array}{cccc}
 0 & 1 & 1 & 0 \quad (\text{borrow required}) \\
 -0 & -0 & -1 & -1 \\
 - & - & - & - \\
 0 & 1 & 0 & 1
 \end{array}$$

Multiplication of binary digits.

$$1 \times 0 = 0$$

$$1 \times 1 = 1$$

Division of binary digits.

$$\begin{array}{ccc}
 \frac{1}{1} = 0 & \frac{0}{1} = 0 \\
 \frac{1}{0} \quad \text{undefined} & \frac{0}{0} \quad \text{undefined}
 \end{array}$$

Below are shown several simple arithmetic problems, with the problems solved in both the decimal and the binary number system.

Decimal	Binary
14	1110
$+12$	$+1100$
<hr/>	<hr/>
26	11010
25	11001
$-7$	$-111$
<hr/>	<hr/>
18	10010
13	1101
$\times 6$	$\times 110$
<hr/>	<hr/>
78	11010
1101	
<hr/>	
1001110	
6	110
$3\overline{)18}$	$11\overline{)10010}$
11	
<hr/>	
11	
<hr/>	
11	
<hr/>	
0	

For fractional numbers the binary system uses negative powers of 2. For example, the fractional binary number  $0.1011_2$  is equivalent to the decimal number  $0.6875_{10}$ .

$$\begin{aligned}
 1 \times 2^{-1} &= 0.5 \\
 0 \times 2^{-2} &= 0 \\
 1 \times 2^{-3} &= 0.125 \\
 1 \times 2^{-4} &= 0.0625 \\
 \\ 
 \hline
 & 0.6875
 \end{aligned}$$

Some fractional values which can be represented exactly in one number system, give non-terminating values in another. Below is shown an example of such a situation.

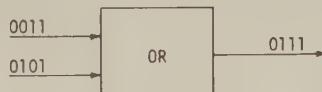
Decimal	Binary
$\frac{1}{5} = 0.20$	$\frac{1}{101} = 0.0011001\dots$
$5 \overline{)1.00}$	$101 \overline{)1.0000000}$
	101
	110
	101
	1000
	101
	110

Thus, the decimal number 0.2 cannot be represented exactly in the binary number system.

The word size of a computer dictates the largest number that can be stored in one memory cell. If a binary machine is said to have a word size of 32 bits, this means that each memory cell has 32 bistable devices available to store information. When storing numbers, one of the bistable devices (or bits as they are more often called) must be used to store the sign of the number. The usual convention is that the positive sign is represented by the 0 state, and the negative sign is represented by the 1 state. This leaves 31 bits to store the magnitude of the number. The largest integer that could be stored would be  $1111111111111111111111111_2$ , all 31 bits in the 1 state. The decimal equivalent to this number is  $2,147,483,647_{10}$ . If a computer has a 16-bit word length, the largest integer that can be stored would be  $11111111111111_2$ , 15 bits in the 1 state. The decimal equivalent to this number is  $32,767_{10}$ .

Computers of today use many ways to perform their arithmetic operations. It would be impossible to discuss all of the techniques that are employed. However, most of the techniques are variations and extensions of the basic concepts that will be presented.

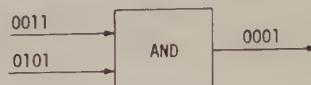
The heart of the arithmetic unit in a digital computer is the adder. The adder not only performs addition, but it is also used in the subtraction, multiplication, and division operations. An adder can be constructed from some rather simple electronic devices called gates. There are several types of gates, one of which is called an OR gate. Figure 1



**Fig. 1.** OR gate.

shows a schematic sketch of a two-input OR gate, with two sequences of binary digits as its input, and a single sequence as its output. The input and output of the gate would be a train of electrical pulses over time. The presence of a pulse would indicate a 1; the absence of a pulse in a time frame would indicate a 0. From the figure it can be seen that the output from an OR gate will be 1 whenever one or both of the inputs to the gate is 1. The output from the gate will be 0 whenever both of the inputs are 0.

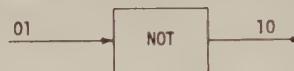
Another type of gate is the AND gate shown in Fig. 2. The output from a two-input AND gate is 1 whenever both of the inputs are 1 and a 0 whenever this is not true.



**Fig. 2.** AND gate.

The third type of gate to be considered is the inverter or NOT gate shown in Fig. 3. The NOT gate has a single input connection. Whenever the input to a NOT gate is 1 the output will be 0, and whenever the input is 0 the output will be 1.

A device for adding two binary digits must produce two output digits. One of the output digits will be the sum digit, and the other digit will be the carry. Table I shows all



**Fig. 3.** NOT gate.

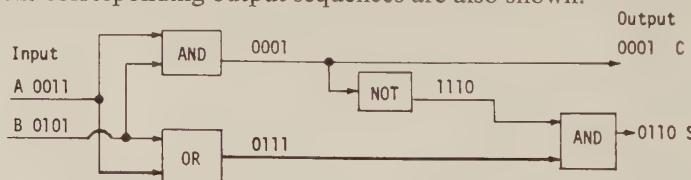
possible combinations of two input binary digits  $A$  and  $B$ , and the resulting digits which must be generated by the addition device in each case. For example, if  $A = 0$ , and  $B = 1$ , the sum digit  $S$  must be 1 and the carry digit  $C$  must be 0.

**TABLE 1**

Binary Addition Table

$A$	$B$	$S$	$C$
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Figure 4 shows how a two-input adder could be constructed using only the three types of gates previously discussed. Input sequences for the two input connections are shown, and the corresponding output sequences are also shown.



**Fig. 4.** Half-adder.

This type of adder is called a half-adder. Half-adders only have two input connections. In adding binary numbers, however, there are usually three digits which must be

considered. For example, consider the situation where two four-digit binary numbers are being added.

$$\begin{array}{r}
 b_4 & b_3 & b_2 & b_1 \\
 a_4 & a_3 & a_2 & a_1 \\
 \hline
 s_4 & s_3 & s_2 & s_1 \\
 c_4 & c_3 & c_2 & c_1
 \end{array}$$

Starting at the right only two digits,  $b_1$  and  $a_1$  need to be considered to form the first sum digit  $s_1$  and the first carry digit  $c_1$ . In the next position three digits must be considered to form the correct sum digit  $s_2$  and the correct carry digit  $c_2$ . These would be the digits  $b_2, a_2$ , and the carry digit  $c_1$  from the previous column. Each column after the rightmost column would involve the same situation. The two digits in the column and the carry digit from the previous column must be considered to form the sum and carry digits for that column. Therefore, a half-adder, with only two input connections, is not sufficient to perform binary addition. A full adder which has three input connections is required. A full adder can be constructed quite simply from two half-adders and an OR gate as shown in Fig. 5. The  $C'$  input is the carry from the previous column of addi-

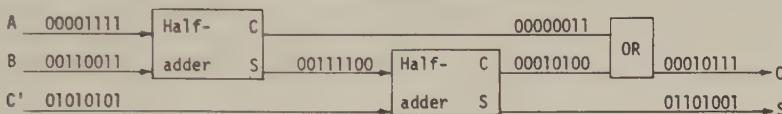


Fig. 5. Full adder.

tion. The three input streams show all the possible combinations of binary digits that can be obtained with three inputs.

Some computers perform their addition serially. This means that the two least significant digits of the two numbers to be added are first presented to the adder. The first sum digit and the first carry digit are generated and transmitted out of the adder. The pulses representing the next most significant digits enter into the adder one time frame later. The carry digit developed previously must be delayed one time frame so

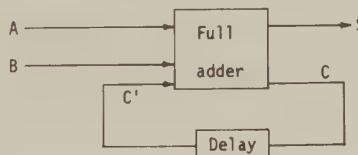


Fig. 6. Serial adder.

that it can enter the adder at the same time as these two digits. Figure 6 shows schematically how this operation takes place.

The sum of two binary numbers is placed in a register in the arithmetic section of the computer. The name given to this register is the accumulator. In serial addition the

least significant digit in the sum is initially placed in the leftmost position of the accumulator. One time frame later when the next most significant digit is formed the contents of the accumulator are shifted one place to the right and the next digit is placed in the leftmost position. This process of shifting and moving in the next digit at the left end of the accumulator continues until the entire sum is formed. An accumulator must be able to hold at least as many binary digits as a computer word.

A faster means of accomplishing binary addition is by use of parallel addition. With parallel addition each sum digit is formed in the same time frame. Essentially a parallel adder may be constructed by using a full adder for each digit position. A half-adder may be used for the least significant digit since a carry does need to be considered for this position. Figure 7 shows a parallel adder for adding two four-digit binary numbers.

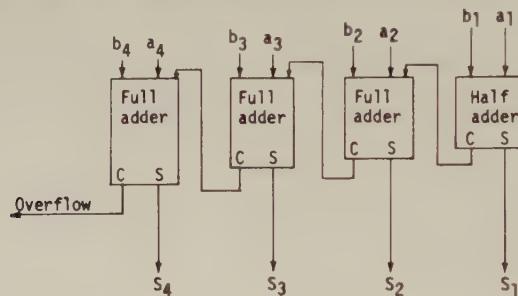


Fig. 7. Parallel adder.

The time frame that a full adder uses to form a sum and carry digit must be of sufficient length so that the two input pulses are still present when the carry pulse appears.

A binary subtractor can also be developed by an arrangement of the OR, AND, and NOT gates. However, a computer does not usually have a separate subtractor; instead it utilizes its adder for most of the subtraction operation. To see how the adder may be used to perform subtraction some simple problems using the decimal number system will be investigated. Suppose we have the subtraction problem

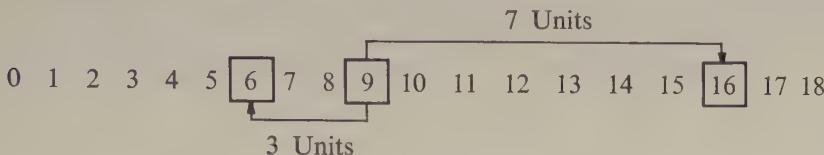
$$\begin{array}{r} 40325 \\ - 26296 \\ \hline \end{array}$$

To perform this subtraction we start with the rightmost column of digits and find it necessary to borrow in order to subtract 6 from 5. Moving to the next column another borrow is required, and this operation proceeds column by column.

$$\begin{array}{r} 3\ 21 \\ 40325 \\ - 26296 \\ \hline \\ 14029 \end{array}$$

This process of looking ahead in order to borrow causes subtraction to be more complex

than addition. However, if complementing is used the necessity of borrowing is avoided in subtraction. The principle of complementing can be demonstrated by the numbers shown below.



When subtracting 3 from 9 the answer 6 can be obtained by moving backwards from 9 for 3 units. The complement of a single-digit number can be obtained by subtracting the number from 10. For example, the complement of 3 is  $10 - 3 = 7$ . The complement of a two-digit number would be obtained by subtracting the number from 100, for three-digit numbers from 1000, etc. By adding the complement of the subtrahend to the minuend the correct answer is obtained if the high order digit is ignored. In the example,  $9 + 7 = 16$ , and the correct answer 6 is obtained when the high order digit is ignored. It can be seen that the high order digit will always be a 1, and if it is ignored the correct answer is obtained.

#### Problem

$$7 - 4$$

$$62 - 35$$

$$123 - 7$$

#### By complementing

$$7 + (10 - 4) = 10 + (7 - 4)$$

$$62 + (100 - 35) = 100 + (62 - 35)$$

$$123 + (1000 - 7) = 1000 + (123 - 7)$$

Note that the number of digits in the minuend determine the base to use when forming the complement of the subtrahend.

The question could be asked as to the advantage of complementing, since there is a great deal of borrowing involved to form the complement of the subtrahend, and borrowing is what is being avoided. The complements that have been discussed so far have been the 10's complement or the true complement. Another type of complement can be formed which does not involve any borrowing and this is called the 9's complement or the base-minus-one complement. To form the base-minus-one complement of a single-digit number the number is subtracted from 9, for a two-digit number it is subtracted from 99, for a three-digit number from 999, etc. Since each digit is subtracted from 9 there will never be any borrowing. When the base-minus-one complement of the subtrahend is added to the minuend the result will be one less than the result that is obtained when the true complement is added.

The principle of complementing, of course, holds true in any number system. To form the true complement of a three-digit binary number the number is subtracted from  $1000_2$ . The complement is added to the minuend, and the correct solution is obtained after the high order digit is dropped.

#### Binary subtraction

$$\begin{array}{r} 101 \\ - 011 \\ \hline 010 \end{array}$$

#### True complement solution

$$\begin{array}{r} 101 \\ + 101 \\ \hline 1010 \end{array}$$

To form the base-minus-one complement of a three-digit binary number the number is subtracted from  $111_2$ . Since binary numbers contain only 0's and 1's there will be no borrowing involved. A 1 must be added to the sum after the complement is added to the minuend to obtain the same result as when the true complement is added.

Binary subtraction                          Base-minus-one solution

101	101
+011	+100
<hr/>	<hr/>
010	1001
	+1
	<hr/>
	1010

The base-minus-one complement of a binary number can be formed very simply by just sending the number through a NOT gate. Since the base-minus-one complement is found by subtracting the number from a series of 1's, every digit that is a 1 in the number will have a 0 in the complement. Every digit that is 0 in the number becomes a 1 in the complement. Therefore to perform subtraction on a digital computer, the subtrahend is sent through a NOT gate, and the output from the gate and the minuend are sent into the adder. Adding the extra 1 can easily be done. In a parallel adder if a full adder is used for the rightmost digit, the extra 1 can be added in the same time frame. The high order digit will flow out of the accumulator, and the correct solution will be in the accumulator.

Binary multiplication on a digital computer can be achieved by a series of adds and shifts. To illustrate this operation, consider the following binary multiplication of two four-digit numbers.

$$\begin{array}{r}
 1011 \\
 \times 1101 \\
 \hline
 1011 \\
 10110 \\
 1011 \\
 \hline
 10001111
 \end{array}$$

Assume the operation is to be performed on a binary computer that has a 5-bit word size. The fifth bit stores the sign and will be ignored in this example. Three four-digit registers will be utilized to perform this multiplication, registers *A*, *B*, and *C*. The multiplicand ( $1011_2$ ) will initially be placed in the *A* register, the multiplier ( $1101_2$ ) will be placed in the *B* register, and register *C* will be set to all 0's. Thus, the registers will look as follows, as the machine begins the multiplication.

				<i>A</i>					
				1	0	1	1		
0	0	0	0	1	1	0	1		

If the rightmost digit of the *B* register contains a 1, the contents of the *A* register and the *C* register are fed into an adder and the resulting sum is placed in the *C* register. If the rightmost digit of the *B* register contains a 0 there is no addition and the *C* register remains unchanged. In the example, since the last digit of the *B* register is 1, the three registers will look as follows after this step.

<i>A</i>			
1	0	1	1
<i>C</i>			
1	0	1	1
<i>B</i>			
1	1	1	0

after addition

The next step is a shift, and the *C* and *B* registers are shifted one position to the right. The rightmost digit of the *C* register becomes the leftmost digit of the *B* register. The rightmost digit of the *B* register is no longer needed and it flows out of the *B* register. The *A* register remains unchanged throughout the entire multiplication operation, so it will no longer be shown in the example. After the shift the *C-B* registers look as shown below.

<i>C</i>				<i>B</i>			
0	1	0	1	1	1	1	0
<i>C</i>				<i>B</i>			

after 1st shift

The rightmost digit of the *B* register is again examined to determine if there should be an addition. Since it is a 0 there is no addition and the *C-B* registers are again shifted.

<i>C</i>				<i>B</i>			
0	0	1	0	1	1	1	1
<i>C</i>				<i>B</i>			

after 2nd shift

The rightmost digit of *B* is 1, so the contents of the *A* register is added to the contents of the *C* register and the sum placed in the *C* register. Another shift then takes place.

<i>C</i>				<i>B</i>			
1	1	0	1	1	1	1	1
<i>C</i>				<i>B</i>			
0	1	1	0	1	1	1	1

after addition

after 3rd shift

Another addition takes place, since the rightmost digit of *B* is 1, another shift takes place, and the final product is now found in the combined *C-B* registers.

<i>C</i>				<i>B</i>			
1	0	0	1	1	1	1	1
<i>C</i>				<i>B</i>			
1	0	0	0	1	1	1	1

after addition

after 4th shift

In the last cycle there was a 1 that overflowed when the *A* and the *C* registers were added. This 1 is retained and is shifted into the leftmost position of the *C* register when the combined *C-B* shift takes place. The number of add-shift cycles that takes place is of course dependent on the word size of the computer. After the multiplication, the high order digits will be in the *C* register, and the low order digits will be in the *B* register. Since a single word in memory is capable of holding only the contents of a single register, many computers have a multiply and round instruction. This instruction will shift the two register product to the right until the high order 1 in the *C* register appears as the leftmost digit in the *B* register. The final low order digit in the *B* register will be rounded depending on the last digit that was shifted out of the right side of the *B* register. In the example the product that would appear in the *B* register after the shifting and rounding would be 1001.

Division on a digital computer is accomplished by using subtraction, in much the same manner as multiplication is accomplished by using addition. As shown previously, subtraction as well as addition can be done with an adder with some modifications. Thus an adder is also used in the division operation. Consider the following binary division problem.

$$\begin{array}{r} 110.1 \\ 10 \overline{)1101.0} \\ 10 \\ \hline 10 \\ 10 \\ \hline 10 \\ 10 \\ \hline \end{array}$$

The same three registers used in multiplication will be used in division. The process starts with the dividend in the *C* register and the divisor in the *A* register. The quotient will be formed in the *B* register. During the initial loading of the *A* and *C* registers, the dividend and the divisor are both shifted to the left as many places as necessary until a 1 appears in the leftmost position of each register. The contents of the two registers are then compared, and if the dividend in the *C* register is less than the divisor in the *A* register, the *C* register is again shifted one more place to the left, so that the high order 1 in the dividend will be in the overflow position. In our example, the three registers would appear as shown below after the initial loading. Since the dividend is larger than the divisor the *C* register will not be shifted the extra position.

<i>A</i>	<i>B</i>			
<i>C</i>	<i>B</i>			
1	0	0	0	
1	1	0	1	0 0 0 0

During division the divisor in the *A* register is subtracted if possible from the dividend in the *C* register. If the subtraction takes place a 1 is placed in the rightmost position of the *B* register. If no subtraction is possible the rightmost position is left as zero. The combined *C-B* registers are then shifted one position to the left and the same process repeated. The division terminates when all the quotient digits in the *B* register have been determined. For the example the division would proceed as shown below. The divisor in the *A* register remains the same throughout the operation and will not be shown.

<i>C</i>					<i>B</i>			
0	0	1	0	1	0	0	0	1
<i>C</i>					<i>B</i>			
0	1	0	1	0	0	0	1	0
<i>C</i>					<i>B</i>			
0	0	0	1	0	0	0	1	1
<i>C</i>					<i>B</i>			
0	0	1	0	0	0	0	1	0
<i>C</i>					<i>B</i>			
0	1	0	0	0	1	1	0	0
<i>C</i>					<i>B</i>			
0	0	0	0	0	1	0	0	1

Since *C* was initially greater than *A*, subtraction takes place. First quotient digit is a 1.

*C-B* shifted. *C* greater than *A*, so subtraction will take place.

Second quotient digit is a 1.

*C-B* shifted. *C* less than *A*, so no subtraction will take place. Third quotient digit is 0.

*C-B* shifted. *C* equals *A*, so subtraction will take place.

4th quotient digit is 1. Remainder in *C*. End of operation.

All digital computers are not binary computers. A binary computer converts all arithmetic data into its equivalent binary form, and it performs all arithmetic operations in the binary mode. Some computers do not convert the data to binary. They retain the decimal identity of the numbers and their arithmetic operations are done in the decimal mode. These decimal machines still use bistable devices to store their data. A group of bistable devices is used to represent each decimal digit. In Table 2 are several codes that have been employed in decimal machines to store data with bistable devices. For

TABLE 2

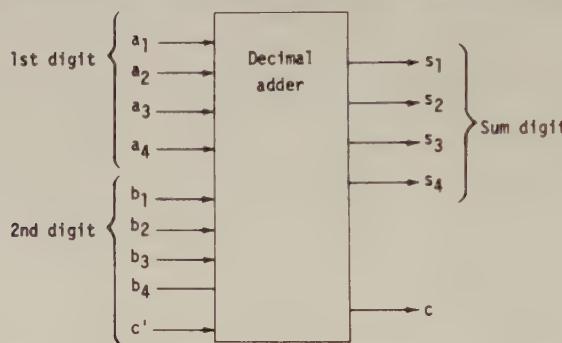
Digit	1-2-4-8	EXCESS-3	1-2-4-2*
0	0000	0011	0000
1	0001	0100	0001
2	0010	0101	0010
3	0011	0110	0011
4	0100	0111	1010
5	0101	1000	0101
6	0110	1001	1100
7	0111	1010	1101
8	1000	1011	1110
9	1001	1100	1111

example, if the number 16 were to be stored in a decimal machine that used the EXCESS-3 code, the eight bits involved would be

0100      1001

This has no relationship at all to the binary equivalent of  $16_{10}$  which is  $10000_2$ .

Binary coded decimal notation simplifies the conversion of decimal data to machine code, and thus decimal computers are usually more efficient in their input-output operations. However, the internal arithmetic operations are usually slower than those performed on a comparable binary machine. The arithmetic section of a BCD machine will have more gates and logic circuitry than a binary machine. For example, Fig. 8



**Fig. 8.** Decimal adder.

shows schematically a decimal adder for adding two decimal digits. Each digit is represented by a 4-bit code.

The decimal adder must accept 9 binary inputs, 4 for each digit, and a single carry bit. The output must have 4 bits for the sum digit and a single bit for the generated carry. A decimal adder can be constructed using full and half binary adders, with AND and OR gates. The design of the adder is of course dependent on the code that is used.

For many arithmetic operations, particularly in the scientific and engineering areas, the problem of keeping track of the decimal points would be extremely difficult, if not impossible, if programmers were required to do it. For example, consider the equation below that solves for the distance traveled  $s$ , during time  $t$  with an initial velocity  $v_1$  and acceleration  $a$ .

$$s = v_1 t + \frac{1}{2}at^2$$

A program to read values of  $v_1$ ,  $t$ , and  $a$ , perform the necessary calculations, and then print  $s$  can easily be written. However, most computers do not provide a position in a memory word to place a decimal point (or a binary point). Thus, if  $t$  is to be 21.376 sec, the number will be stored in a memory word as 21376. The programmer must be aware that there is an implied decimal point between the 1 and the 3. The result  $s$  will also have no decimal points. The computer will store as many significant figures as it can in a memory word, and the programmer will have to scale the output depending on the magnitude of the input data. Fortunately most computer systems of today handle the

problem of locating the decimal point without the assistance of the programmer, by the use of floating-point numbers.

Floating-point numbers are based on the notation that scientists and engineers frequently use when making computations. Any decimal number may be expressed as a decimal that is between 0.1 and 1.0 times some power of 10. For example, the number 207.1 may be expressed as  $0.2071 \times 10^3$ , the number 0.01562 may be expressed as  $0.1562 \times 10^{-1}$ . The decimal portion of the number is referred to as the mantissa and the exponent is called the characteristic. The mantissa can be brought into any desired range by shifting the decimal point right or left and decreasing or increasing the characteristic. If the decimal point is shifted one place to the right the characteristic is decreased by 1. If the decimal point is shifted one place to the left, the characteristic is increased by 1. The numbers shown below are all equivalent.

$$\begin{aligned}0.2071 &\times 10^3 \\2.071 &\times 10^2 \\0.02071 &\times 10^4\end{aligned}$$

In performing hand calculations, the mantissas are usually put into the range between 1 and 10. Then, when a series of multiplications and divisions are performed, the mantissa of the result can quickly be estimated, and the result characteristic can be found by adding characteristics during multiplications and subtracting them during divisions. The problem below illustrates these operations.

$$\frac{(2.0 \times 10^3)(3.0 \times 10^{-1})(4.0 \times 10^3)}{(4.0 \times 10^{-2})(8.0 \times 10^4)} = \frac{24}{32} \times 10^3 = 0.75 \times 10^3 = 750$$

The sum of the characteristics in the numerator is 5 and the sum of the characteristics in the denominator is 2. Since the numerator is divided by the denominator, the characteristic of the result is 3. This problem in conventional notation would be

$$\frac{(2000)(0.3)(4000)}{(0.04)(80000)}$$

A floating-point number in a computer is stored in a single memory cell. To illustrate the handling of floating-point numbers in a computer assume we are using a decimal machine that is capable of storing ten decimal digits and a sign in each word. The first two digits in a word will be used to store the characteristic of a floating-point number, and the remaining eight digits will be used for the mantissa. For example, the number  $0.5638 \times 10^2$  might be stored in a memory word as: +0 2|5 6 3 8 0 0 0 0. Since the characteristic can be negative, one might assume that the number  $0.734 \times 10^{-3}$  would be stored as: -0 3|7 3 4 0 0 0 0 0. If the sign position is used for the sign of the characteristic, this would not allow negative numbers to be stored. To circumvent this difficulty, the characteristics of floating-point numbers are not stored as shown above. A constant is first added to the characteristic and this resulting sum is then stored in the first two digit positions. In our example computer the constant will be 50. Thus the number  $0.5638 \times 10^2$  will be stored as: +5 2|5 6 3 8 0 0 0 0, and the number  $0.734 \times 10^{-3}$  will be stored as +4 7|7 3 4 0 0 0 0 0. Adding the constant 50 to the

characteristic allows the characteristic to be stored without involving the sign of the word.

When two floating-point numbers are added, the computer will first compare the characteristics of the numbers. If both characteristics are the same, the mantissas are added and the sum is given the same characteristic. If the characteristics are not the same, the mantissa of the smaller number is shifted to the right until its characteristic is equal to the characteristic of the larger number. Each position the mantissa is shifted increases the characteristic by 1. For example, consider the numbers  $0.5638 \times 10^2$  and  $0.734 \times 10^{-3}$ . These numbers as stored would be

$$\begin{array}{r} +5\ 2|5\ 6\ 3\ 8\ 0\ 0\ 0\ 0 \\ +4\ 7|7\ 3\ 4\ 0\ 0\ 0\ 0\ 0 \end{array}$$

If these two numbers were to be introduced to the arithmetic section for floating-point addition, the machine would shift the mantissa of the smaller number five positions to the right before addition would take place. In effect, the two numbers that would be presented to the floating point adder would be

$$\begin{array}{r} +5\ 2|5\ 6\ 3\ 8\ 0\ 0\ 0\ 0 \\ +5\ 2|0\ 0\ 0\ 0\ 0\ 7\ 3\ 4 \end{array}$$

The resulting sum that would be stored would be  $+5\ 2|5\ 6\ 3\ 8\ 0\ 7\ 3\ 4$ . The computer must look for overflow when adding the two mantissas. For example, in adding  $+5\ 2|7\ 3\ 6\ 4\ 0\ 0\ 0\ 0$  and  $+5\ 2|6\ 4\ 7\ 5\ 0\ 0\ 0\ 0$ , the sum of the two mantissas would result in a 1 going into the overflow position.

$$\begin{array}{r} +5\ 2|7\ 3\ 6\ 4\ 0\ 0\ 0\ 0 \\ +5\ 2|6\ 4\ 7\ 5\ 0\ 0\ 0\ 0 \end{array}$$

$$\underline{1\ 3\ 8\ 3\ 9\ 0\ 0\ 0\ 0}$$

Whenever overflow occurs the computer will shift the mantissa of the sum one position to the right and increase the characteristic of the sum by 1. The floating-point number that would be stored in the above example would be  $+5\ 3|1\ 3\ 8\ 3\ 9\ 0\ 0\ 0$ .

When subtracting floating-point numbers the computer will again shift the mantissa of the smaller number, if necessary, so that the characteristics of the numbers involved are equal. After the subtraction the machine will examine the result to see if the number is normalized. A floating-point number is unnormalized if there are any leading zeros in the mantissa. For example, the number  $+5\ 2|0\ 0\ 4\ 6\ 2\ 0\ 0\ 0$  is unnormalized. If a number is unnormalized, it is put in normalized form by shifting the mantissa to the left as many spaces as required and decreasing the characteristic. The same number in normalized form is  $+5\ 0|4\ 6\ 2\ 0\ 0\ 0\ 0\ 0$ . Floating-point numbers are always stored in normalized form on most computers. It allows for simpler logic design for the multiplication and division operations.

Multiplication and division of floating-point numbers do not require that the characteristics of the numbers be equal. On the hypothetical machine that we have been considering the characteristic of the product in multiplication can be found by adding

the two characteristics and subtracting 50. In division the characteristic of the divisor is subtracted from the characteristic of the dividend and 50 is added to obtain the characteristic of the quotient. Overflow is possible in multiplication and an unnormalized result can be obtained in division.

On most computers the floating-point operations are performed with a special set of registers. However, some machines do not have additional hardware for floating-point arithmetic. The floating-point operations on these machines are taken care of by special coding. Whenever a floating-point instruction is encountered, the operating system of the computer will call on this coding to execute the operation.

## BIBLIOGRAPHY

- Bartee, T. C., *Digital Computer Fundamentals*, McGraw-Hill, New York, 1960.  
Bernstein, J., *The Analytical Engine: Computers—Past, Present and Future*, Random House, New York, 1963.  
Davis, G. B., *An Introduction to Electronic Computers*, McGraw-Hill, New York, 1965.  
Flores, I. W., *The Logic of Computer Arithmetic*, Prentice-Hall, Englewood Cliffs, N.J., 1963.  
Goldstine, H. H., *The Computer From Pascal to Von Neumann*, Princeton University Press, Princeton, N.J., 1972.  
Hellerman, H., *Digital Computer System Principles*, McGraw-Hill, New York, 1967.  
Irwin, W. C., *Digital Computer Principles*, Van Nostrand, Princeton, N.J., 1960.  
McCormick, E. M., *Digital Computer Primer*, McGraw-Hill, New York, 1959.  
Murphy, J. S., *Basics of Digital Computers*, Vols. 1 and 2, Rider, New York, 1958.  
Richards, R. K., *Arithmetic Operations in Digital Computers*, Van Nostrand, Princeton, N.J., 1955.  
Siegel, P., *Understanding Digital Computers*, Wiley, New York, 1963.

Charles E. Donaghey, Jr.

## ARPA NETWORK

The ARPA Network (ARPANET), under development by the Advanced Research Projects Agency (ARPA) since 1968, stands as a significant advance both in computer systems and in data communications.

The ARPANET project is aimed at developing the concept of *resource sharing*—the linking together of a number of independent computer systems so that the user of any one system can freely make use of the resources of any system in the network, including computing power, specialized software, and specialized hardware. With such resource sharing, a program could call on the resources of other computers much as

it calls upon a subroutine in the computer in which it resides. With such a capability, many software and hardware duplications could be eliminated, communication between users with common interests would be greatly enhanced, and a computer community could be created with its resources available to all users.

Resource sharing represents a step beyond time sharing. It was ARPA, a research arm of the Department of Defense, which in the early 1960s funded much of the work which led to the development of time sharing, such as Project MAC at M.I.T.

Today, ARPANET links some 60 computers in 45 locations throughout the country to provide computer resource sharing among several thousand ARPA-related research workers.

The development of the network was undertaken by ARPA to overcome the inadequacies of the existing telephone system in meeting the stringent data communications requirements for resource sharing.

For ARPANET, ARPA researchers developed *packet switching*, a radically different communication technique. Packet switching, a form of store-and-forward message switching, is combined in ARPANET with advances in distributed network control and the use of high-speed leased telephone lines to achieve lower cost, higher speeds, greater reliability, and greater flexibility than heretofore has been realized in data communications systems. Virtually error-free communications is provided from almost any known interactive terminal type to any of a variety of computers, as well as between the computers themselves.

The advantages of ARPANET both for terminal-to-computer and computer-to-computer communications make the system one of the most significant recent contributions to the field of data communications.

Figure 1 shows a geographic map of ARPANET with user computers and terminals accessing the network through 45 minicomputers interconnected by some 40,000 miles of high-speed lines. Figure 2 is a logical map showing the ring-configuration multiple paths which interconnect the sites. Note the great variety of computers connected to the network.

Using ARPANET is a remarkable experience. Because the code and format differences between various types of terminals and computers are reduced to standard protocols throughout the network, one may sit at any terminal, call into any point in the network, and sign on any network computer such as the IBM 360/91 at UCLA, the H-645 Multics system at M.I.T., a PDP-10 at Stanford, and so on.

Not only does the ARPANET provide efficient sharing of computer resources such as programs and data bases among many users, but specialized hardware is being connected to the system for sharing by the user community, such as the experimental TX-2 computer at Lincoln Laboratory, and the ILLIAC IV and a trillion-bit laser memory scheduled for connection to the network in 1975. Load sharing is a good use of the system. For example, an East Coast university can send its overload to a West Coast university in the morning, and the reverse take place in the afternoon. Software specialization also becomes practical, with users in many locations accessing a particular specialized language or library on one machine.

In order to understand ARPANET and its implications, we first shall describe the manner in which the network functions and introduce the concept of the packet. We

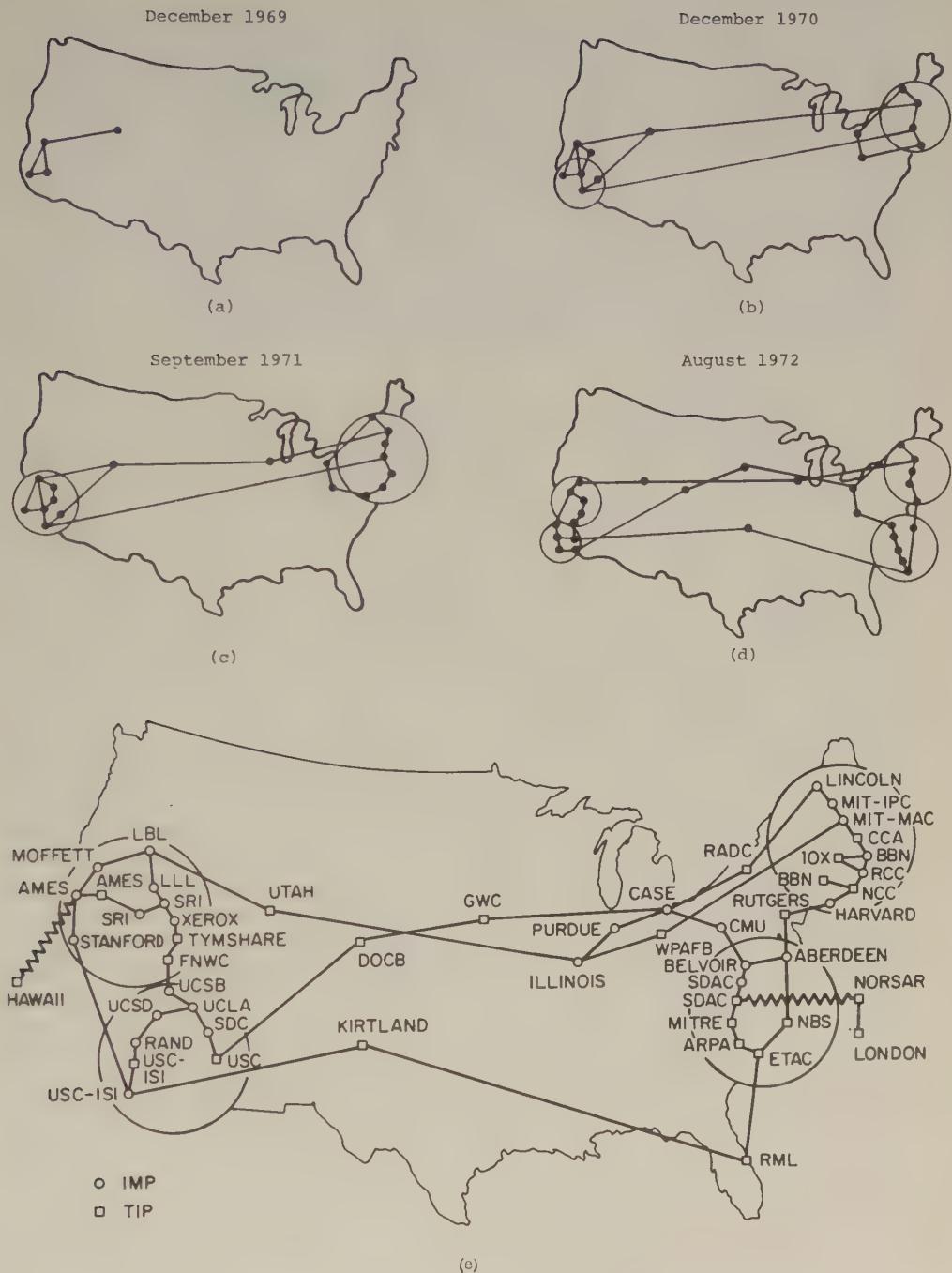
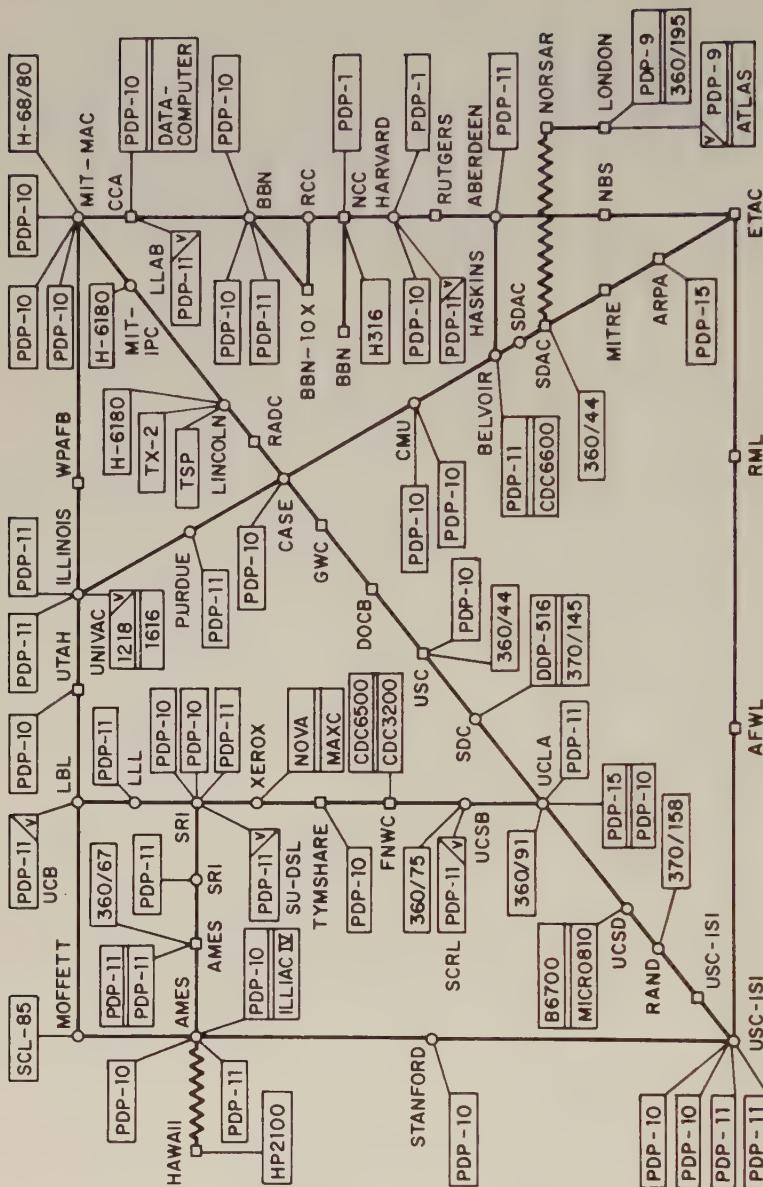


Fig. 1. Evolution of the ARPA network. (a) December 1969. (b) December 1970. (c) September 1971. (d) August 1972. (e) November 1974.



**Fig. 2.** ARPA network, logical map, November 1974.

then shall discuss the evolution of ARPANET, some of the technical problems encountered, and the economics of the network. Finally, we shall point out the special properties of distributed communication systems of this type, and the promise such networks appear to hold for the future.

## NETWORK FUNCTIONS

In the ARPANET, each using computer, which may be a time-sharing or a batch machine, is called a HOST. HOST computers and user terminals tie to the network through two types of minicomputers.

1. *Interface Message Processor (IMP)*. Each IMP interfaces up to three HOST computers to the network. In addition to providing the standard interface for each HOST computer, the IMP performs all communication functions—converting HOST messages into packets, routing, initiating, storing, and forwarding packets as they pass through the network acknowledging packet-receipt, controlling errors, preventing traffic congestion, etc.

2. *Terminal Interface Message Processor (TIP)*. In addition to performing all the functions of an IMP, the TIP provides for direct connection to the network of up to 63 user terminals or consoles. Serving as a simple HOST, a TIP converts the characteristics of diverse types of terminals to an ARPA network standard.

The IMPs and TIPs are connected by leased 50 kbit/sec or faster communication lines, and each handles its communications tasks completely independent of the HOST computers. The network operates under a distributed control scheme (which we discuss further later) under which each IMP and TIP makes its own decisions as to control of communications with its HOST and routing of message traffic through the networks. To send a message to another HOST, a HOST precedes the text of its message with an address and delivers it to its IMP. The IMP dynamically determines the best route, provides error control, and notifies the sender of its receipt. TIPs perform the same function for terminals.

When a message is ready for transmission, the originating IMP (or TIP) divides it into a set of one or more packets, each with appropriate header information. Each packet will independently make its way through the network to the destination IMP (or TIP), where the packets will be reassembled into the original message and then transferred to the destination HOST or terminal.

Packet switching differs from circuit switching by the ability to send a message from IMP to IMP, one link at a time. Each packet is stored at each intermediate IMP until a suitable communication line is available. Such a system is called a store-and-forward system. At no time need an entire communication path exist from sender to receiver.

In a circuit-switched network, the entire message path from sender to receiver is chosen in advance. Network resources are thus completely allocated before message

transmission begins. In the current telephone system, which employs circuit switching, a number of seconds are required for this allocation. In a packet-switching system, resources are dynamically allocated and conflicts between different users competing for the same communication facilities are resolved on a step-by-step basis. Thus a message may be partially transmitted through a system because spare capacity momentarily exists on a portion of the overall route the message will eventually follow. The message may then wait within the network until it can make further progress to its destination. The net result is that required network resources are utilized as soon as they become available, and thus more efficient utilization of these resources occurs.

Typical data transmission traffic consists of two types—messages corresponding to terminal-to-computer (teletypes, graphics consoles, etc.) and to computer-to-terminal conversations and long messages corresponding to computer-to-computer and tape-to-tape file transfers and other similar traffic. The packet switched system, which typically provides under 0.2 seconds average single packet delay, provides lower cost communications for each type of traffic. For example, a common characteristic of terminal input traffic is the large ratio of idle wait time to the actual time involved in transmitting data. Ratios of 100 to 1 are common. This means that if a standard communication line (either leased or dial-up) is established for such a conversation, the average utilization of that line will be about 100 times greater than the raw cost of moving the bits. For the leased line situation, the user would find himself paying for the line 24 hours a day, 7 days a week, while the dial-up user would have to either break the connection after each message input (clearly unfeasible because of the many seconds it presently takes to establish a connection) or utilize the communication path at the 1% efficiency level.

Consider a user who wishes to transmit a large block of data between two sites across the United States. With the conventional Bell telephone system, he would typically make such a transmission at night over either a dial-up or leased line path. The time taken for transmission would depend on the speed of the line. For example, a 10 million bit file transmitted at 2000 bit/sec over a dial-up line would require about two hours for transmission and would therefore cost at least as much as a two-hour long distance call (\$25 or more). With the ARPANET, the user could dial to a TIP located in his city (a local call) and then pay for his ARPANET usage at ARPANET costs (about 30¢/1000 packets of data). Thus the same file transfer would cost about \$3. The economic advantage in this case would derive from the packet-switched network's ability to share among a large community of users on a nearly instantaneous basis the broad-band communication lines (50 kbit/sec or higher) which have a substantially lower cost per bit.

## THE EVOLUTION OF ARPANET

Both the concept and implementation of ARPANET are due largely to the efforts of one individual, Dr. Lawrence Roberts, Director for Information Processing Techniques at ARPA. An early paper [1] coauthored by Dr. Roberts defined the

requirements and the basic approach to a communication system between computers and users which would allow the sharing of data, software, and computing resources.

Initial experiments were conducted between M.I.T.'s Lincoln Laboratory in Massachusetts and System Development Corporation in California. From these, Dr. Roberts concluded that an entirely new type of communication service would be required if an effective resource-sharing computing network were to be developed. Dr. Roberts thought that such a service should provide high reliability, low error rates, fast response time for interactive messages, high throughput for graphics users, file transfers, etc.; and in order to be useful, the cost of such communication should be no greater than about 20% of the computing costs of the systems connected to the network.

Early work by Baran and his associates at the Rand Corporation hinted that a distributed communications network might provide the answer. Dr. Roberts' selection of a distributed packet-switched communication system design to fulfill the stringent requirements for a resource-sharing network constitutes a major contribution to the field of computer science, and his judgment subsequently was confirmed by the successful development of ARPANET.

During the early phases of ARPANET development a typical node consisted of one or more large time-sharing computers connected to an IMP. As the network grew, an additional need to allow remote terminals direct access to the network was recognized, and a TIP was developed. The collection of HOSTs, IMPs, TIPs, and communications lines constitutes the packet-switched resource-sharing network.

ARPANET was initiated in 1969 as a network connecting three West Coast computers (University of California at Los Angeles and Santa Barbara, and Stanford Research Institute) to a computer in Salt Lake City (University of Utah). By February 1971, the network connected approximately 20 different computers at 15 facilities distributed across the United States. By early 1973 the network connected some 40 facilities with an average of several computers per facility. Figure 1 shows the evolution of the ARPANET from the basic four-IMP design in 1969 to a 1974 version of the network.

## ARPANET DESIGN

The development of ARPANET involved three principal activities.

1. The design of the IMP to act as store-and-forward packet switches.
2. The design of protocols for use in item sharing, batch processing, file transmission, and other data processing activities.
3. The system modeling, network analysis, and network design to predict network performance and to specify the capacity and location of each communication circuit.

The IMPs and TIPs were designed under a contract awarded by ARPA to Bolt Beranek and Newman Inc. (BBN) of Cambridge, Massachusetts, and were built to

operate independently of the exact network structure. The network structure in its various stages of evolution was specified by Network Analysis Corporation (NAC) of Glen Cove, New York, using models of network performance developed by NAC and by the University of California at Los Angeles (UCLA). A group of representatives of user universities and research centers, known as the Network Working Group, specified HOST-to-HOST and terminal protocols, and is in the process of developing higher level user protocols to facilitate effective use of the network.

A variety of performance requirements and system constraints were considered in the design of the network. Unfortunately, many of the key design objectives and user requirements had to be specified long in advance of actual network implementation. One of the significant advantages of the distributed design concept was the great flexibility of the system to accommodate uncertainties without degradation of performance.

## IMP AND TIP

### IMP Functions [2]

Since ARPANET is designed on the principle of packet switching, the network itself must deal with such problems as routing, buffering, synchronization, error control, and reliability. To insulate the various computer centers from these problems and to insulate the network from the problems of the computer centers, ARPA decided to place small identical processors (the IMPs) at each center.

The key IMP design question was the partition of responsibilities between IMPs and HOSTs to achieve reliable and efficient operation. The network was designed to function as a communication system and problems concerning the *use* of the network by the HOSTs (such as protocol development) were left to the HOSTs. To provide isolation between the IMPs and HOSTs, routing strategies must dynamically adapt to changes in the states of IMPs and circuits, and a flow control strategy must project the network against HOST malfunction and congestion due to IMP buffer limitations.

The IMP originally developed for ARPANET contains a 16-bit 1  $\mu$ sec computer (Honeywell DDP-516, later replaced by a Honeywell H-316) that can transmit from about 0.5 to 1 Mbit/sec of useful information.

The maximum HOST message size was constrained to be approximately 8000 bits. To obtain delays of a few tenths of a second for such messages and to lower the required IMP buffer storage, the IMP program partitions each message into one or more packets, each containing at most approximately 1000 bits. Each packet of a message is transmitted independently to the destination where the message is reassembled by the IMP before shipment to the destination HOST.

The IMPs provide error control and proper sequencing of messages. The IMP assigns a sequence number to each packet as it enters the network, and the destination IMP processes that sequence number. This processing is necessary since a duplicate

packet will arrive at the destination IMP if an acknowledgment is missed, thus causing a successfully received packet to be retransmitted.

Errors are caused primarily by noise on the communication circuits, and are handled by error detection and retransmission between each pair of IMPs along the transmission path. Failures in detecting errors are made to occur on the order of years apart by the use of a 24-bit cyclic check sum for each packet.

A conventional circuit switched network responds to a network overload by a busy signal. In a packet switched network, if packets were permitted to freely enter and leave the network, the network could become congested and cause message time delays to increase. Therefore, a major IMP responsibility that critically affects network performance is packet routing. Routing strategies for distributed networks require decisions to be made with only information available to an IMP. A simple example is for each IMP to *independently* route a packet along the IMP's current estimate of the "best" path to the destination. An IMP performs routing computations using information received from other IMPs and local information such as the alive/dead state of its lines. Normally, local information alone, such as the length of internal queues, is insufficient to provide an efficient routing strategy without assistance from the neighboring IMPs. Routing information cannot be propagated in sufficient time to characterize the instantaneous traffic flow accurately. Therefore, the routing algorithm does not focus on the movement of individual packets, but rather uses statistical information in the selection of routes.

The routing strategy originally used had each IMP select one output line per destination onto which to route packets. The line chosen was the one with the minimum estimated time delay to the destination. The selection was updated every half second using minimum time estimates from the neighboring IMPs and internal estimates of the delay to each of the neighbors. This approach works quite effectively with moderate levels of traffic. To handle heavy traffic flow, a more intricate scheme is being implemented which allows multiple paths to be efficiently used. This method separates the problem of defining routes onto which packets may be routed from the problem of selecting a route when a particular packet must be routed. By this technique it is possible to send packets down a path with the fewest IMPs and excess capacity, or when that path is filled, the one with the next fewest IMPs and excess capacity, etc.

### TIP Functions [3]

During the early phases of ARPANET development, nodes consisted of one or more large time-shared computer systems connected to an IMP. This arrangement provides a means for sharing resources between such interconnected computers, each site potentially acting both as a user and as a provider of resources. Thus the early ARPANET linked computers; these computers linked to terminal users outside of ARPANET via their own communications ports. However, since a user might be at a site either with no HOST computer or where the existing computer might not be a terminal-oriented time-sharing system, a need for direct terminal access to the network was recognized. A completely separate means for terminals to access the network directly seemed desirable and a type of IMP with a flexible terminal handling capability

—a terminal IMP, or TIP—was created. After considerable study it was decided that the TIP would function solely as a terminal handler, and that the computation load and storage should be in the HOSTs or in the terminal and not in the TIPs. This decision is in keeping with the philosophy that the ARPANET is a *communication* system and that noncommunication functions should be performed outside of the network.

Up to 63 terminals of widely diverse characteristics, either remote or local, may be connected to a given TIP and thereby access the network. In addition to supporting terminals, a TIP may serve as the network connection for a local HOST, in the same manner as an IMP. The TIP is built around a Honeywell H-316 computer with 20K core. It embodies a standard 16-port multiplexed memory channel with priority interrupts and includes a Teletype for debugging and program reloading. As in the standard IMP, interfaces are provided for connecting to high-speed modems (50 kbit, 230.4 kbit, etc.) as well as to HOSTs.

In order to accommodate a variety of devices, the TIP handles a wide range of data rates and character sizes in both synchronous and asynchronous modes. Data characters of length 5, 6, 7, or 8 bits are allowed. Since no character interpretation is done by software, any character set may be used. Given these characteristics, the TIP will connect to the great majority of terminal devices such as Teletypes, alphanumeric CRT units, and modems, as well as with suitable remote interface units, and to many peripheral devices. Either full or half duplex devices can be accommodated.

## NETWORK PROTOCOLS [4]

One of the more difficult problems in utilizing a network of diverse computers and operating systems is that of dealing with incompatible data streams. Computers and their language processors have many ways of representing data. To make use of different computers, it is necessary to develop a scheme for eliminating each incompatibility or produce standard representations for performing a set of functions. The method chosen for dealing with the bulk of these problems for ARPANET is via standard representations for performing various functions.

To support the practical use of intercomputer communications, sets of protocols that had to be constructed for: (1) remote use of interactive systems, (2) file transfer between differing and remote file systems, and (3) remote job entry. The lowest level software protocols in ARPANET involve message exchange between the IMPs. At these levels the contents of messages are unspecified. At higher levels, more and more is stated about the meaning of message contents and timing.

Three lower level software protocols support the user level communications interface.

1. The IMP-IMP protocol which provides for reliable communication among IMPs. This protocol handles transmission error detection and correction, flow control to avoid message congestion, and routing.

2. At the next higher level is the IMP-HOST protocol which provides for the passage of messages between HOSTs and IMPs to create virtual communication paths between HOSTs. With the IMP-HOST protocol, a HOST has operating rules which permit it to send messages to specified HOSTs on the ARPANET and to be informed of the dispensation of those messages. In particular, the IMP-HOST protocol constrains HOSTs to make good use of available communications capacity without denying such availability to other HOSTs.
3. The HOST-HOST protocol is the set of rules for HOSTs to construct and maintain communication between user jobs (processes) running on remote computers. One process requiring communications with another on some remote computer system makes requests on its local supervisor to act on its behalf to establish and maintain those communications under the HOST-HOST protocol. The major goal in the construction of these protocols was to provide user processes with a general set of useful communication primitives to isolate them from many of the details of operating systems and communications.

The major current ARPANET protocol activity involves the remote use of interactive systems via a Telecommunications Network (TELNET) protocol. A user at a terminal connected to his local HOST (the user HOST) controls a process in a remote HOST as if he were a local user of the remote HOST. His local HOST copies characters between his terminal and TELNET connections over ARPANET. The user TELNET must be able to distinguish between commands to be acted on locally and input intended for the remote HOST. In the remote HOST it is desirable that a process controlled over ARPANET behave as it would if controlled locally.

Since ARPANET is a distributed computer operating system, attempts were made to construct and use distributed file systems. Initially, several *ad hoc* file transfer mechanisms were developed to provide support. These mechanisms took two forms: use of the TELNET data paths for text file transfer and use of raw data stream communication between compatible systems. These *ad hoc* file transfer mechanisms require explicit and often elaborate user intervention and depend a great deal on the compatibility of the file systems involved. There is then an effort to construct a File Transfer Protocol (FTP) to allow exchange of structured sequential files among widely differing file systems with a minimum explicit user intervention. The FTP will support file operations including (1) list remote directly, (2) send local file, (3) retrieve remote file, (4) rename remote file, and (5) delete remote file. The protocol designers intend to regularize the protocol so that file transfer commands can be exchanged by consoleless file transfer jobs engaged in activities such as automatic back-up and dynamic file migration. The transfers envisioned will be accomplished with a Data Transfer Protocol (DTP) able both to preserve sequential file structure and to permit data flow between different file systems.

ARPANET is intended to give a wide community of users access to specialized facilities containing powerful machines for computer-intensive applications. The mode of operation most suited to these users has been batch remote job entry (RJE). Typically, a user will generate a "deck" for submission to a batch system. He expects to wait from minutes to hours for that deck to be processed, and then to receive his

output. A standard *RJE protocol* is being developed to provide for job submission to a number of facilities in the ARPANET. This protocol is being constructed using the TELNET and FTPs.

The development of ARPANET stimulated the growth of a new branch of computer sciences—intercomputer communications among dissimilar computers. An early major function of the network has been the use of the IMPs and protocols to allow different computers at the *same* facility to communicate. In fact, until April 1972, the intra-IMP traffic exceeded the inter-IMP traffic on the network. The development of protocols is an on-going activity that can be expected to continue for the life of the ARPANET. As protocols advance, use of the network will become easier, more users will find it convenient to perform their computation via the network and will thus be stimulated to make newer and more elaborate demands on the protocol designers.

## NETWORK STRUCTURE

The network structure (topology) design problem involves consideration of the following two questions.

1. Starting with a given state of the network topology, what communication line modifications are required to add or delete a set of IMPs or TIPs?
2. Starting with a given state of network topology, when and where should communication lines be added or deleted to account for long-term changes in network traffic?

If the locations of all network nodes are known in advance, it is clearly most efficient to design the topological structure as a single global effort. However, in ARPANET, as in most actual networks, the node locations are modified on numerous occasions. On each such occasion the topology might be completely reoptimized to determine a new set of circuit locations. In practice, there is a long lead time between the ordering and delivery of the high-speed lines used (9 months currently for a 50-kbit line) and therefore major topological modifications cannot be made without substantial difficulty. It is thus prudent to add or delete nodes with as little disturbance as possible to the basic network structure consistent with overall economic operations.

The major considerations in the topological design of a computer network such as ARPANET are

1. Cost
2. Throughput
3. Delay and response time
4. Network reliability

An efficient topological design provides a high throughput for a given cost. Although many measures of throughput are possible, a convenient one is the average amount of

traffic that a single IMP can send into the network when all other IMPs are transmitting according to a specified traffic pattern.

The original constraints on the topological design were the available common carrier circuits, the target cost and throughput, and the desired reliability. The network performance objectives on the initial ARPANET design were

1. Average packet delays under 0.2 sec.
2. Average total throughput capability of 10–15 kbits/sec for all HOSTs at an IMP.
3. Two-connected network topology; that is, at least two completely independent paths between each pair of IMPs.
4. Peak throughput capability of approximately 40 kbits/sec per pair of IMPs in an otherwise unloaded network.

While it is essential to consider each of these constraints, it often results that several are automatically satisfied for designs satisfying the remaining. Initially, this was the case for ARPANET. The requirement for 40 kbit/sec peak throughput and 0.2 sec average packet delay dictated the use of 50 kbit/sec communication links. The reliability requirement of two-connectivity forced the minimum value of network throughput to be in the range of 10–15 kbit/sec/IMP when 50 kbit/sec lines were used throughout the network. The time delay constraint was adequately met by using appropriate routing algorithms in a network whose maximum line loadings were slightly derated from 50 kbits/sec.

The original ARPANET reliability constraint of two-connectivity was adopted for simplicity to represent a reliability requirement involving a less well-defined concept concerning "availability of resources." Since failure and repair data for IMPs and lines were not known, this more sophisticated criterion could not be evaluated. Subsequent analysis showed that at the early stages of ARPANET evolution, the "availability of resources" was adequate if the network was two-connected. Thus the cost-throughput tradeoff was the overriding consideration, and the object of ARPANET structural optimization was to decrease the ratio of cost to throughput subject to an overall cost limitation. The technique used employed a complex network optimization program that utilizes heuristics, and routing and time-delay analysis algorithms, to generate low cost designs. Time-delay models were initially used to calculate the throughput corresponding to an average time delay of 0.2 sec and to estimate the packet rejection rate due to all buffers filling at an IMP. As experience with these models grew, the packet rejection rate was found to be negligible and the computation discontinued. The delay computation was subsequently first replaced by a heuristic calculation to speed the computation and later eliminated after it was found that time delays could be guaranteed to be acceptably low by preventing certain sets of links, called cutsets, from being saturated. However, it is becoming evident that as ARPANET increases in size, the reliability constraints are beginning to limit design choices. It may even happen that the cost-reliability tradeoff will replace the cost-throughput tradeoff as the basic design consideration. Because of this factor, we consider the reliability issue in more depth.

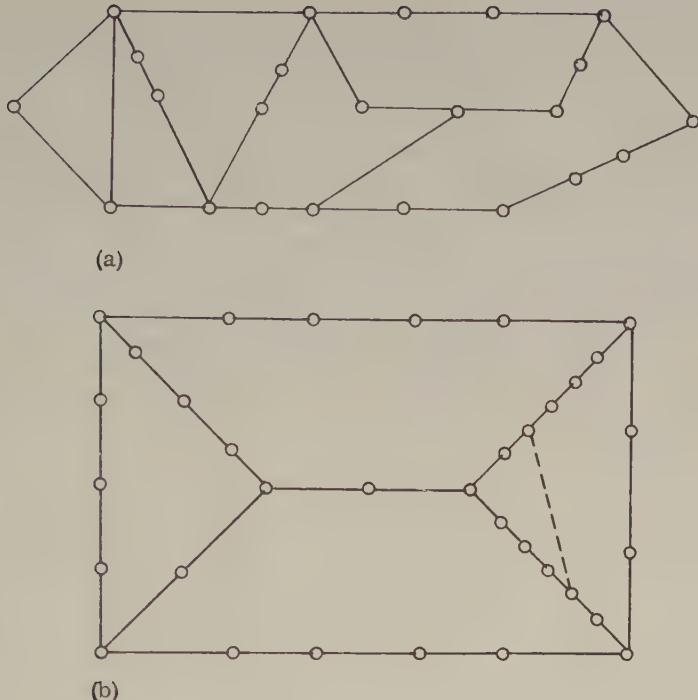
Reliability analysis for computer networks is concerned with the dependence of the

reliability of the network on the reliability of its nodes and links. *Element* reliability is easily defined as, for example, the fraction of time the element is operable. The proper measure of *network* reliability is not as clear and simple. Several possible measures are the number of elements which must be removed to disconnect the network; the probability that the network will be disconnected; the expected fraction of node pairs which can communicate through the network; and the expected throughput of the network subject to element failures. Many other measures can and have been suggested. Another class of measures arises when the nodes are not of equal importance as in centralized networks or hierachal networks. In a centralized network, one may be interested in the expected number of nodes which can communicate with the central node. More general criteria arise when different node pairs are weighted by their importance. For example, communication between ILLIAC IV and certain other nodes will be of high priority in the ARPANET.

Node failures can affect network reliability in two ways. First, if a node fails, clearly it cannot communicate with any other node in the network. Changing the network configuration has no effect on this component of network reliability. Another effect of node failures is that the failed nodes destroy some potential communication paths between other pairs of nodes. Link failures also affect network reliability in the second way.

As stated, the initial design procedure for ARPANET controlled reliability by insisting that there be at least 2 node-independent paths between every pair of nodes. Later computations proved that this implied almost perfect reliability in the following sense. Suppose each node in the network is inoperative a specified fraction of the time. Then a lower bound for the average number of pairs of nodes which cannot communicate is equal to the average number of node pairs not communicating in a network in which every pair of nodes is joined by a perfectly reliable link. In the actual network, no addition or redistribution links can reduce the average number of node pairs not communicating below this value. For small networks the existence of 2 node-disjoint paths between each pair of nodes invariably resulted in an expected number of node pairs not communicating very near the perfect lower bound. Thus the addition of more links for reliability purposes was not justified.

To fix these ideas and to give specific examples of the reliability characteristics of small networks, we consider two versions of ARPANET. The first is a 23-node network that has been thoroughly analyzed as a common measuring point or standard for various reliability analysis techniques. The second network is a medium size network of 33 nodes in which for the first time an additional link was added primarily for reliability reasons. The 23-node network is represented in Fig. 3(a). This design had a yearly line cost of \$847,000 (at government rates) for its 28 lines and an average throughput of 10 kbit/sec/IMP. Measured downtimes for both IMP and communication links in ARPANET have been on the order of 2%. With this value of reliability, it can be shown that on the average 4% of the pairs of nodes cannot communicate even if every pair of IMPs were connected by a perfect link. An analysis of the 23 IMP network shown in Fig. 3(a) yields that, on the average, 4.9% of the nodes cannot communicate. Thus we see that approximately 80% of the node pairs which cannot communicate can be ascribed purely to the fact that one of the nodes of the pair in



**Fig. 3.** ARPA networks for reliability example. (a) 23 IMP network.  
(b) 33 IMP network.

question has failed. Thus the improvement in reliability to be gained by changing the network configuration is minor.

Figure 3(b) depicts a 33-node ARPANET design. For this network the difference between the average fraction of node pairs not communicating (0.058) and the lower bound based on the perfect network topology containing all possible links is almost double the difference for the 23-node network. Improving the reliability by changing the network configuration becomes feasible. For example, a link from Fort Belvoir to Aberdeen (indicated by the dashed line) increased the cost by a little over 1% but increased the reliability by almost 10%. Thus even for a network with only 33 nodes, it is becoming necessary to consider reliability in more detail than the "two-connectivity" criterion.

## NETWORK PROPERTIES

ARPANET represents only one of the many possible implementations of a packet-switching network. However, distributed packet-switching communication networks like ARPANET have basic characteristics significantly different from circuit-switched

networks. While all of their properties have not yet been discovered, 4 years of analysis of this type of system have led to a number of important conclusions which significantly differ from the properties of other types of networks.

1. *Network costs are basically independent of the distances messages may travel.* This conclusion was reached after a series of traffic simulations and network design optimization in which traffic patterns were widely varied from ones in which communication use was uniformly distributed throughout the country. For any kind of distance-dependent traffic distribution studied, economic network designs which provided equal total traffic carrying capacity, equal reliability, and equal response times were found to have approximately equal cost. Examples of the nature of the results are shown in Figure 4 where the cost per megabit of transmitted data is plotted against the yearly network cost per node for three significantly different traffic patterns in a 40-node ARPA-like network [7].

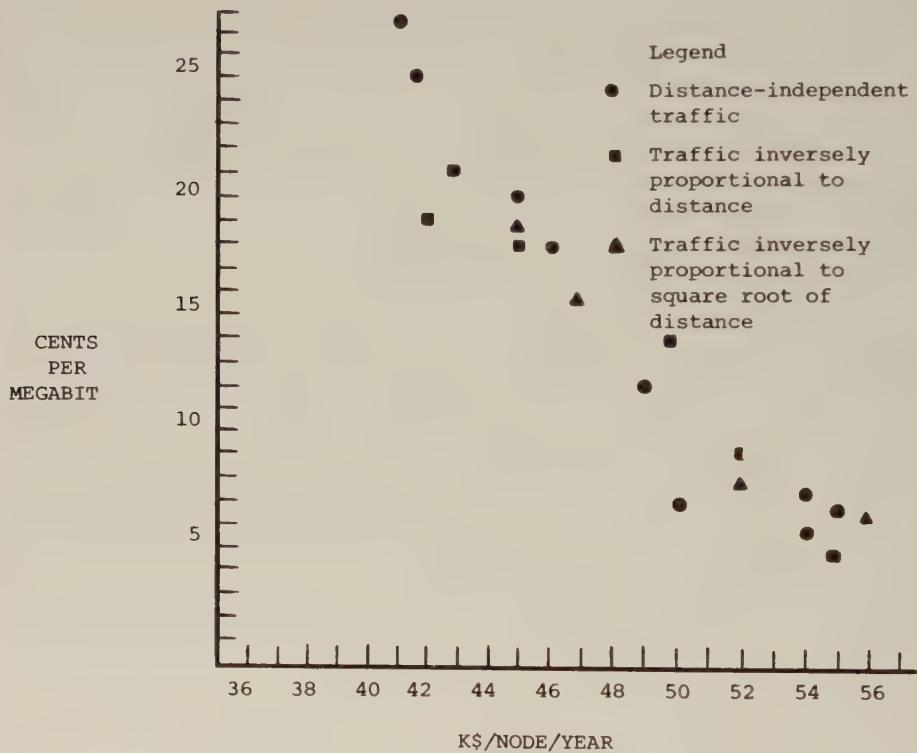


Fig. 4. Network costs as a function of capacity and traffic pattern.

2. *Network performance is relatively independent of traffic distributions as long as the total traffic input to the network does not change.* For example, if traffic is uniformly distributed between all pairs of nodes, network performance is approximately equal

to the performance which results if some nodes generate ten times as much traffic as others. The major performance factor is the total amount of traffic within the network. Thus network load can be simply measured by the total number of packets or bits within the network. An indication of this property is given in Table 1. Here we list average network throughput per IMP at 0.2 sec average delay for 100 totally randomly generated traffic patterns for an 18-node version of the ARPANET [5].

**TABLE I**  
18-Node Network with Random Loads<sup>a</sup>

Average throughput (bits/sec/node)	Number of traffic patterns
13,000	1
14,000	5
15,000	16
16,000	19
17,000	33
18,000	18
19,000	5
20,000	3

<sup>a</sup> Sample mean: 17 kbit/sec/node. Standard deviation: 1760 bit/sec/node.

3. *Network economics are independent of peak traffic rate* [1]. This property was derived through a design study performed to determine the cost effects of a good percentage of the facilities in the country joining ARPANET but not wishing to pay for 50-kbit line interconnections. It was supposed that they only required a peak rate which could be achieved by two 9.6 kbit line interconnections.<sup>1</sup> In a 2-connected network like ARPANET the effective data rate for long transmissions is about 80% of twice the basic line rate. Thus the capacity of a network using 50 kbit lines is about 80 kbytes, and that of a network with 9.6 kbit lines is about 15 kbytes. It was supposed that one-half of the nodes on the network were allowed to use 9.6 kbit/sec lines instead of 50 kbit/sec lines. The result was that, although the nodes using 9.6 kbit/sec lines received significantly lower peaks throughput, the cost of the network was approximately the same whether 9.6 kbit lines were used extensively or 50 kbit lines were used for all nodes. Analyses made with a random distribution of these factors generally showed that there is no significant variation in cost with varying distributions of peak rate requirements.

4. *Within a range of total traffic volume from 0 kbit/sec/node to approximately 20 kbit/sec/node, cost per node depends linearly on total traffic.* This factor is evidenced

<sup>1</sup> 9.6 kbit/sec lines were chosen since this speed is presently the fastest feasible data rate that can be achieved over a voice grade communication line.

by the curves in Fig. 5 which show annual ARPANET costs per node and costs per node for distributed networks connecting major United States population centers. The curves were the results of extensive design studies [5, 6].

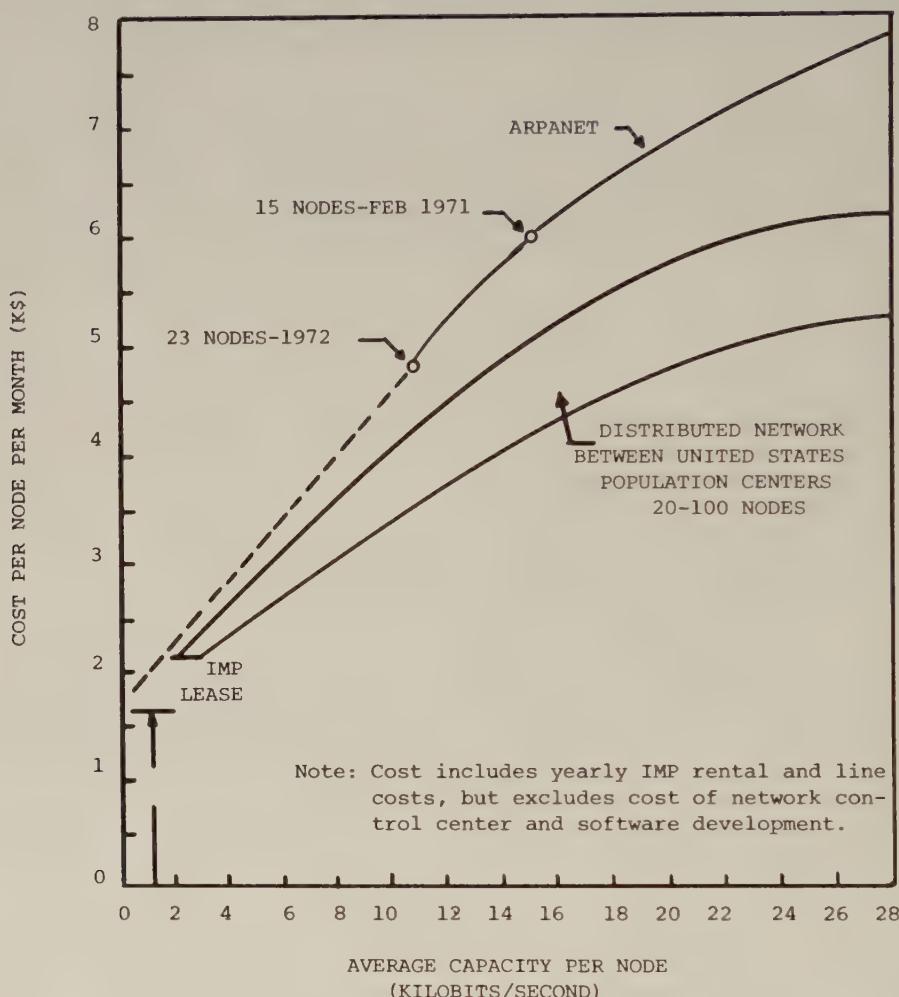


Fig. 5. ARPA network communication costs and cost region for 20-200 IMP networks connecting major population centers.

The above properties indicate that the economics of a packet-switched distributed network like ARPANET are significantly different from a circuit-switched network. One may charge for network use purely on a cost-per-bit or cost-per-packet basis plus a fixed charge to cover the cost of hardware and connection into the network. There are no other factors to consider.

## ARPANET IMPLICATIONS

When ARPANET was initiated in 1969, little was known about how well a distributed packet-switched network could perform. The capabilities of the IMP to be an effective network manager were unknown; the achievable levels of traffic and network configurations required to handle this traffic were not understood. The sensitivity of network's performance and the cost of providing a network which would yield high performance in spite of inaccurate projected requirements were also not known. The protocol problems to enable useful resource sharing over the network had not been addressed.

ARPANET research presently underway includes the expansion of ARPANET via satellites to Hawaii, Alaska, and other locations with a different kind of network extension which broadcasts packets from the satellite to all nodes; the addition of Remote Job Entry terminals which will allow users who have large computation requirements effective network access; the provision of specialized resources that no single facility could afford such as the ILLIAC IV parallel processor and a trillion-bit laser storage device; the development of a high-speed IMP for connection to T1 carrier lines; the investigation of low-speed remote host interconnection devices; and the investigation of packet transmission via radio terminals.

Today, many network design efforts using packet-switching technology are proceeding in Canada, Europe, and the United States. In the United States two organizations, Packet Communications, Inc., and Telenet Communications Corp., have been formed with the stated intention to provide a public distributed packet-switching network utility for computer resource sharing and data communications.

The packet-switched network resource of the future will offer dramatic services when compared with those available today.

1. Data communication costs will be directly related to volume of data transmitted. Charges will depend solely on the amount transmitted and will be independent of the distance between sender and receiver.
2. The user will automatically receive high reliability through the built-in redundancy of packet routings without additional cost.
3. Communications over the network will be virtually error free. This will eliminate the need for expensive error correction or detection equipment.
4. A user will be able to directly access the network with any commonly available terminal without having to develop special software.
5. The user will have available a large variety of computer and data base resources on a virtually instantaneous basis.

The packet-switched network will provide significant economies to both buyers and sellers of computer services. Time sharing and remote batch processing computer service companies will find the network a convenient and inexpensive vehicle to market their services and extend their sales regions. Owners of large under-utilized computers will be able to sell excess computer time to smaller users who cannot afford adequate local service. Organizations with a variety of different types of computer needs (e.g.,

"number crunching" and time sharing) will be able to buy specialized services from several vendors—rather than buy a single general-purpose machine which is a compromise between their user's needs. Eventually, we can expect to see reservation systems, credit card and point of sale terminals, inventory control, and a host of other new information management systems communicating efficiently and rapidly through the network.

The exact time schedule required for the development of such a national communications resource is difficult to predict. However, because of ARPANET, the concept of such a utility has evolved from science fiction to a foreseeable reality.

## REFERENCES

1. T. Marill and L. G. Roberts, Toward a cooperative network of time-shared computers, in *Proceedings of the Fall Joint Computer Conference*, 1966.
2. H. Frank, R. Kahn, and L. Kleinrock, Computer communication network design—Experience with theory and practice, in *Proceedings of the Spring Joint Computer Conference*, 1972.
3. S. Ornstein et al., The terminal IMP for the ARPA network, in *Proceedings of the Spring Joint Computer Conference*, 1972.
4. S. Crocker et al., Function oriented protocols for the ARPA network, in *Proceedings of the Spring Joint Computer Conference*, 1972.
5. H. Frank and W. Chou, Routing in computer networks, in *Networks* 1(2), 99–112 (1971).
6. H. Frank, I. T. Frisch, and W. Chou, Topological considerations in the design of the ARPA computer network, in *Proceedings of the Spring Joint Computer Conference*, 1970, pp. 581–587.
7. H. Frank and W. Chou, Throughput in computer-communication networks, in *International Report on the State of the Art of Computer Network*, Infotech, London, 1972.

## BIBLIOGRAPHY

- Baran, P., S. Boehm, and P. Smith, *On Distribution Communications* (series of 11 reports), Rand Corp., Santa Monica, Calif., 1964.
- Bhushan, A. K., et al., *The Data Transfer Protocol*, RFC #264, NIC #7812, November 1971.
- Bhushan, A. K., et al., *The File Transfer Protocol*, RFC #265, NIC #7813, November 1971.
- Carr, S., S. Crocker, and V. Cerf, Host-host communication protocol in the ARPA network, in *Proceedings of the Spring Joint Computer Conference*, 1970, pp. 589–597.
- Heart, F., R. Kahn, S. Ornstein, W. Crowther, and D. Walden, The interface message processor for the ARPA computer network, in *Proceedings of the Spring Joint Computer Conference*, 1970, pp. 551–567.
- Kahn, R. E., Terminal access to the ARPA computer network, in *Proceedings of the 3rd Courant Institute Symposium*, Prentice-Hall, Englewood Cliffs, N.J., 1970.
- Kahn, R. E., and W. R. Crowther, Flow control in a resource sharing computer network, in *Proceedings of the 2nd ACM-IEEE Symposium on Problems in the Optimization of Data Communication Systems*, Palo Alto, Calif., October 1971, pp. 108–116.
- Kleinrock, L., Models for computer networks, in *Proceedings of the International Conference on Communications*, 1969, pp. 21–21–16.
- Kleinrock, L., Analysis and simulation methods in computer network design, in *Proceedings of the Spring Joint Computer Conference*, 1970, pp. 569–579.
- Network Analysis Corporation, *4th Semiannual Technical Report for the Project, Analysis and Optimization of Store-and-Forward Computer Networks*, Defense Documentation Center, Alexandria, Va., December 1971.

- Network Analysis Corporation, *5th Semiannual Technical Report for the Project, Analysis and Optimization of Store-and-Forward Computer Networks*, Defense Documentation Center, Alexandria, Va., June 1972.
- Network Working Group, *The Host-Host Protocol for the ARPA Network*, NIC #7147, 1971. Available from the Network Information Center, Stanford Research Institute, Menlo Park, Calif.
- O'Sullivan, T., et al., *TELNET Protocol, ARPA Network Working Group Request for Comments, RFC #158*, NIC #6768, May 1971.
- Postel, J. B., *Official Initial Connection Protocol, ARPA Network Working Group Document 2*, NIC #7101, June 1971.
- Roberts, L. G., *Multiple Computer Networks and Inter-computer Communications*, presented at the Association for Computing Machinery Symposium on Operating Systems, Gatlinburg, Tenn., 1967.
- Roberts, L. G., Access control and file directories in computer networks, in *Proceedings of the IEEE International Convention*, March 1968.
- Roberts, L. G., Resource sharing networks, in *Proceedings of the IEEE International Convention*, March 1969.
- Roberts, L. G., A forward look, *SIGNAL* 25(12), 77-81 (1971).
- Roberts, L. G., Extensions of packet communication technology to a hand held personal terminal, in *Proceedings of the Spring Joint Computer Conference*, 1972.
- Roberts, L. G., and B. Wessler, Computer network development to achieve resource sharing, in *Proceedings of the Spring Joint Computer Conference*, 1970.
- Roberts, L. G., and B. Wessler, The ARPA computer network, in *Computer Communication Networks* (Abramson and Kuo, eds.), Prentice-Hall, Englewood Cliffs, N.J., 1973.
- Van Slyke, R., and H. Frank, Reliability of computer-communication networks, in *Proceedings of the 5th Conference on Applications of Simulation, New York*, December 1971.
- Van Slyke, R., and H. Frank, Network reliability analysis—I, *Networks* 1(3), (1972).

*Howard Frank*

## ARRAYS

A large proportion of the problems for which computers are used lends itself to representation as operations on matrices or as an array process. The algorithms used are representable both in terms of operations on matrices and where each individual component is specified in the algorithm.

An array, in the sense being used here, is an organized collection of entities whose organization lends itself to a subscripted representation. A matrix is a frequently encountered example.

The computer support for this type of problem comes in a number of forms which have developed over the last 15 to 20 years to aid the programmer, as well as to make possible processing of extremely large arrays or complex processes applied to arrays.

The problems related to the computer processing of arrays are still relatively early in their solution, although a considerable amount of development has gone on during the past few years. There is still wide-ranging debate over the appropriate technology for the successful processing of large array problems. One factor which stands out as a motivating force for solution is that, to some degree, the problems tackled by scientists are limited by the speed of the largest computer available. Although there are special computers, special languages, and other support for large array processing, there are many problems of current interest which the largest computers available will not solve in a reasonable length of time. Some of these problems will be mentioned in the discussion of array computers.

## LANGUAGE SUPPORT

The first step taken beyond the raw computer was in the late 1950s when higher level programming languages really began to be developed. One of the significant parts of these languages was the combination of declarations about the dimensionality and characteristics of an array as a structure, and the looping capability that was built into languages such as FORTRAN and ALGOL for processing the array.

The simple matrix multiplication outlined below is a FORTRAN expression of the method for solving the product of two matrices and storing the results in a third.

```
C      FORTRAN PROGRAM TO MULTIPLY TWO
C      MATRICES A AND B, STORING
C      RESULTS IN C
C
C      DIMENSION A (64,64), B (64,64), C (64,64)
C      ALLOCATE SPACE AND DECLARE SIZE
C
C      DO 1 I = 1, 64
C      DO 1 J = 1, 64
C      DO 1 K = 1, 64
1      C (I,J) = C (I,J) + A (I,K) * B (K,J)
C
C      ACCUMULATION OF SUM OF PRODUCTS
C      INTO C (I,J)
C
C      END
```

A comparable program in the BASIC language is:

```
100  DIMENSION A (64,64), B(64,64), C(64,64)
200  MATRIX C=A*B
300  END
```

The BASIC program statement following the FORTRAN example reflects the language designer's ability to notice that certain kinds of operations are done so commonly, and

are so well defined, that they should become part of the language. It also reflects a recognition of the data type "matrix" explicity. In the design of the BASIC language, arrays are treated as a normal language element (data type) and a large variety of operations on these arrays can be expressed directly without having to specify the complexities of the applicable algorithm. An example of much higher efficiency in specification is in matrix inversion. The statement **MATRIX C = INV (A)** is a legal BASIC statement. The meaning here is clear and again, the language designer has incorporated this and other operations into the possible statements which his language supports.

Unlike the multiplication example where there was no loss of power to the programmer by specifying the matrix operation in a very high level language, there is considerable loss of generality of the power available to the BASIC programmer in that he has specified that an inverse is to be taken, but not how, i.e., not what algorithm is to be applied.

Considerable attention has been directed to an area of mathematics which deals with specific algorithms for doing matrix operations, such as inversion and eigenvalue extraction, which are highly sensitive to the type of matrix being treated, the precision of the machine, and other variables. The BASIC programmer, in making simple expressions, is giving up the ability to treat unusual kinds of matrices with some assurance that their treatment will be correct.

This kind of trade-off is a constant consideration in language design between generality and ease of use. This is true both in language design and at the level of hardware support. Since generality and ease of programming ability has been found to be extremely useful, many languages now support arrays as a data type which can be talked about in language statements.

In parallel with these language developments, there were also major developments in algorithms for performing operations on matrices, algorithms using operations on matrices, and the arithmetic operations that were required to be performed for carrying out these algorithms. Operations in the last sense are computer operations, such as multiply or divide.

One of the characteristics of any operation on matrices is that somewhere during the algorithm there will be a long sequence of the same operations (e.g., multiply then add) performed with only the elements operated on changing. As an example, if we look at the matrix multiplication, there are a large number of multiply-add combinations.

It is patterns like this which motivated the other two major steps which have occurred in support of matrix calculation, special purpose hardware and computers specifically designed to work efficiently with the array computation.

## SPECIAL PURPOSE HARDWARE

If we look again at the matrix multiplication, we see that it is a repetitive process of multiply-add, multiply-add, etc. It would also be possible to write an equivalent

algorithm which did all of the multiplies for a given element of the resultant matrix followed by all of the adds.

One of the typical characteristics of computers is that to execute an operation such as a multiplication, it is necessary to make several accesses to memory during the computation. For example, a multiply instruction requires that the operation itself be picked from memory and passed to the execution portion of the computer, that the two elements to be multiplied be accessed, usually one being in a register and the other in memory, then multiplying and leaving the result in a register. A further access to memory is frequently made to store the result. These memory accesses are usually sequential and their time requirements are therefore additive.

Given this pattern of steps which take time during computation, we see that only one part of the computation really requires a computer/execution element, while the remainder is dependent on the ability of memory to provide information quickly.

Noting this, and noting that there are no conditional operations in the middle of a matrix multiply, such as testing results, we see it is possible to specify a machine as an adjunct to the computer which can do specialized operations very well, if the performance improvement is worth the machine cost.

Several such pieces of equipment have been designed in the past and put into use. Typically they operate somewhat more like a computer peripheral storage device than like a computational unit in that they release the central processor of the machine to go on to other tasks while they are performing their own.

The special purpose device which would be necessary to support the matrix multiplication would need to be able to take several arguments and carry out a set of calculations based on those arguments with a minimum interference with memory, and to perform the multiply or add function at the highest possible speed. The arguments that would need to be passed in this case would be, for the multiply, the fact that it was a multiply, the starting address of the two lists of elements to be multiplied, a place to put the result, and the length of the lists. Thus five arguments are adequate to specify the operation.

Note that in this case the number of memory accesses is reduced (no instructions need to be fetched), but by less than 1/2. However, the memory accesses do go on in parallel if the central processor is using memory for some other purpose. The central processor is released to other tasks and it is possible to make a machine capable of doing one particular operation very much faster than the average operation of the host machine.

The array processor spoken of above is an abstraction of the problem. There have been actual machines built for this purpose.

An array processor somewhat more general and powerful is the IBM 2938 Array Processor, an attachment to the System 360 models 44, 65, and 75. This unit attached to the input-output data path and was programmable as an I/O device. Quite extensive matrix operations, involving a sequence of elementary arithmetic steps, could be specified for execution in parallel with mainframe programs. The timing performance reported for this unit was up to 10 times mainframe speed.

In introducing a special class of hardware, we have introduced several notions which are totally foreign to the languages which are developed and are in use up to this point

in time, as well as possibly requiring modification in the algorithm or program itself.

In constructing an algorithm which utilizes both the central processor and its adjunct array processor, the user or programmer is required to understand what goes on in parallel or what potentially can go on in parallel, or to handle the special purpose device like an I/O device, waiting for completion of requests. The efficiency with which the program runs is dependent on the user's comprehension and organization of the program.

While it is possible to incorporate in higher level languages (or subroutine calls) features which will take the user's program and map it on to a more efficient sequence, language development in this direction has not yet gone very far. (The multitasking facility of PL/I may be an exception.) The auxiliary processor approach has not been very successful or extensively used. Perhaps the primary reason for this is that there have been alternative directions in the use of memory hierarchy which alleviate the problem of multiple accesses to memory and in building increasingly faster in-stream instruction processors for operations like multiply and divide. The net result of this is that the good features of the auxiliary processors have been absorbed into the design of the single processing stream computers.

What characterizes this approach is the notion of doing a single instruction at a time, but doing it very fast and with the minimum burden on the main machine. Minor modifications to this, such as having a computer that can do several instructions at a time, have been available through the 1960s and indeed, this is how the manufacturers have been able to get very high speed out of conventional technology, using a single instruction stream.

In the long run there appears to be an inherent limitation on how fast a computer can do any one thing. This leads to the suggestion of an alternative approach to making computers faster: Make them do several operations at the same time.

## ARRAY COMPUTERS

Because so much of what the computer does is the repetitive, basically identical operation involved in array calculations, one approach which has been tried with some success is the array computer. The array computer is designed to do the same operation on a number of different operand pairs at the same time, thus relieving the requirement that any single operation be exceedingly fast. Rather, the alternative approach is taken that the same operation will be performed relatively slowly, but within the capabilities of standard technology, with many repetitions going on at the same time.

The array computer is a particular organization of computer hardware aimed at solving a particular kind of problem in an economical and effective fashion, recognizing the requirements to stay within current technology for building components, and the ability to replicate these components in large numbers. Standard technology, at any point in time, is much cheaper than state-of-the-art technology.

The current state-of-the-art with regard to array computing is represented by the ILLIAC IV Computer, which is the result of a continuing project at the University

of Illinois, looking into special purpose computers for solving extremely large cases of matrix problems. A list of the problems cited in justifying the development of the ILLIAC were:

1. Matrix algebra (addition, multiplication, inversion)
2. Matrix eigenvalue calculation
3. Linear and integer programming
4. Eulerian hydrodynamic flow
5. Alternating direction implicit methods
6. Table lookup techniques
7. General circulation weather model
8. Digital filter design
9. Beam forming and convolution
10. Fourier analysis
11. Ordering, filing, and retrieving data
12. Genetics model

The design of the ILLIAC incorporates an array of small computers, each containing, or having access to, separate memories; the ability to process instructions; and the ability to pass information to some set of its neighboring processors. In addition to these "processing elements," there is, in the basic design, a master computer which schedules and controls the action of the processing elements.

An ILLIAC IV configuration would consist of some 64 processing elements, each with 4000 to 8000 words of main memory, and a collection of data paths adequate to transfer information to a few of its neighbors. Each processing element is a computer with extensive capabilities for performing arithmetic operations. In the configuration of 64 processing elements, it would be possible for each of the 64 to perform the same operation on the same relative components of a matrix. For instance, the  $i$ th processor could perform a multiplication of  $A_i \times B_i$ . Each of these operations would be performed concurrently and thus the time required would be the time for one to perform the operation. That results in a machine 64 times as fast, if all of the elements are in use.

Several assumptions are made in this best case analysis. For the machine to operate at its peak of efficiency, each processing element must have access within its own memory to the elements required for this particular computation. If it is necessary to transfer information between elements, overhead reduces the effectiveness.

If there is a strong pattern to this transfer of information, then the neighborhood network allows rapid transmission and controls the loss of efficiency. In a case of random transfer, or a worst case transfer, the results could be a major reduction in efficiency.

A further assumption that was made is that there is some optimum match between array size and machine size, in our example above 64, which is to be treated identically at the same time. If only 32 array elements were available for a 64 processor configuration, the machine could be reduced to half its normal efficiency, as it would if there were 65 elements. Reasonable matching between the number of processing elements and the problem at hand is thus seen as a critical item.

Let us refer back to our matrix multiplication example and see how it maps onto the

ILLIAC in this case, choosing a  $64 \times 64$  array to be multiplied by another  $64 \times 64$  array, with the results stored in a third array.

As an example of programming the ILLIAC type of computer, consider again our matrix multiplication:

$$C = A * B$$

We can expand the matrix C to show the 262,144 individual multiplications and adds that must occur.

$  \begin{aligned}  C_{1,1} &= a_{1,1} * b_{1,1} & 1 \\  &+ a_{1,2} * b_{2,1} & 2 \\  &+ a_{1,3} * b_{3,1} & 3 \\  &\vdots \\  &+ a_{1,64} * b_{64,1} & 64  \end{aligned}  $	$  \begin{aligned}  C_{1,2} &= a_{1,1} * b_{1,2} & 1 \\  &+ a_{1,2} * b_{2,2} & 2 \\  &+ a_{1,3} * b_{3,2} & 3 \\  &\vdots \\  &+ a_{1,64} * b_{64,2} & 64  \end{aligned}  $
⋮	
$  \begin{aligned}  C_{2,1} &= a_{2,1} * b_{1,1} \\  &+ a_{2,2} * b_{2,1} \\  &+ a_{2,3} * b_{3,1} \\  &\vdots \\  &+ a_{2,64} * b_{64,1}  \end{aligned}  $	$  \begin{aligned}  &\vdots \\  &\vdots  \end{aligned}  $

Consider now the starting arrangement of the 64 memories.

Memory 1	Memory $n$
$a_{1,1}$	$a_{n,1}$
$a_{1,2}$	$a_{n,2}$
$\vdots$	$\vdots$
$a_{1,64}$	$a_{n,64}$
$b_{1,1}$	$b_{1,n}$
$b_{2,1}$	$b_{2,n}$
$\vdots$	$\vdots$
$b_{64,1}$	$b_{64,n}$
$c_{1,1}$	$c_{n,1}$
$c_{1,2}$	$\vdots$
$\vdots$	$\vdots$
$c_{1,64}$	$c_{n,64}$
$\vdots$	$\vdots$

Given this initial arrangement, 64 sets of 64 multiply-add pairs of instructions are available without any rearrangement, one row by one column. They are numbered 1 to 64 and result in computing the diagonal element of C.

No further useful computation can be carried out at this point without a rearrangement of memory. However, a simple rearrangement of memory, shifting the column of  $B$  one memory to the left (i.e., to its nearest neighboring memory), results in the availability of data for computing another 64 elements. The elements are those directly above the diagonal.

This process can be repeated 64 times and all computation will be complete with a very efficient use of the 64 processors. All operations, both arithmetic and data transfer, use all available processors in parallel.

This verbal picture of the algorithm can be stated as follows:

1. Set up initial memory.
2. Multiply  $a_{ik}$  by  $b_{kj}$ —these elements are in memory  $i$ .
3. Add to temporary  $C_{ij}$ —this is also in memory  $i$ .
4. Repeat steps 2 and 3 for  $K = 1$  to 64.
5. Move a column of  $b$ .
6. Repeat until computation is complete.

In setting up to analyze this example, we can list multiplications that have to take place as well as the additions. Then, by finding some algorithm for mapping the elements into memories and mapping data transfers, we can construct a program and measure its efficiency in terms of parallel instructions and get an efficiency rating on the machine performing compared to a single stream, single memory machine.

Anyone who has done programming in an algorithmic language will note that there are strong relationships between the expression of this algorithm for doing the moving and processing and those which would be involved in writing within the present language, such as FORTRAN or ALGOL.

The algorithm and the data assignment, however, express a number of factors which the FORTRAN programmer did not need to take into account, and it encompasses an analysis which is very specific to the machine, unlike that of the classical programming languages.

As we go on to more complex algorithms, this analysis and its machine dependence become more and more critical. An obvious answer to this is to construct programming languages and implementation to these languages which accomplish a mapping of a standard algorithm onto a highly structured machine which is different than the machine which the programmer is used to; namely, from a nonparallel to a parallel machine. Techniques for analyzing programs in this were, and generating code for them are, partially developed, and the whole area of research into parallel programming algorithms has been worked on for a number of years.

While the ILLIAC represents the state-of-the-art at the present time, there are several things which raise doubt as to whether the array processor is in fact the direction that the industry will go.

We have noted that an ideal match between the configuration of the machine and the problem at hand yields outstandingly good results. On the other hand, a slight mismatch begins to degrade these results significantly. It has been argued that programs, in general, represent a very poor match for an array processor. Clearly, one of

the measures of performance is the degree of parallelism which can occur and will occur in executing a program to solve a particular problem.

Some parts of all programs are inherently nonparallel; for example, testing the results of an operation and then taking one of several branches in the program based on the results of that test. Parallelism breaks down at the point where a branch occurs.

In this case the simple design of a number of processors performing identical functions on the same relative elements will not work out well. Again, set-up prior to execution of a program is usually nonparallel. This would include determining memory layout and setting parameters.

With all of these comments, it is quite possible that the next generation which makes a significant breakthrough in array processing will take into account some of the branch-and-join research that has gone on in parallel processing.

A further area of problems, system reliability, has occurred with the ILLIAC. It may indicate difficulty in building a one-of-a-kind machine or some inherent reliability problem with the overall design philosophy.

The system reliability has been an extremely serious problem in bringing up the ILLIAC, but it is too early to tell whether this problem will be overcome and what its long-term impact will be. This is another area of concern to possible vendors of such large computers.

## BIBLIOGRAPHY

- Barnes, G. H., *et al.*, The ILLIAC IV computer, *IEEE Trans.* C17(8), 746-757 (August 1968).  
Kuck, D. J., ILLIAC IV software and application programming, *IEEE Trans.* C17(8), 758-770  
(August 1968).  
Murtha, J. C., Highly parallel information processing systems, in *Advances in Computers*, Vol. 7,  
Academic, New York, 1966, pp. 2-116.  
Ruggiero, J. F., and D. A. Coryell, An auxiliary processing system for array calculations, *IBM  
Sys. J.* 8(2), (1969)

*George F. Badger, Jr.*

# ARTIFICIAL INTELLIGENCE

## INTRODUCTION

Not too long ago the term "giant brain" was a commonplace synonym for the newly developed electronic digital computer with its awesome speed and ability to do

repetitive kinds of computation and data-processing far better than humans could. A hardy and growing branch of the computer community has been attempting to transform this descriptive term into a reality. Turing [1], and later Shannon [2] among others, speculated, respectively, on the general problem of designing "intelligent" machines and the specific problem of a chess-playing machine. A quarter century later the field has emerged from speculation to the hard task of writing programs to perform various tasks commonly conceded to require intelligence if done by a human. The body of activity goes variously under the names of "artificial intelligence," "machine intelligence," sometimes more modestly, "complex information processing," as well as numerous other terms more descriptive of the particular type of application.

The enterprise is also controversial in at least two distinct ways. It elicits hostility from those who believe that it may be possible to create a very intelligent machine but it is immoral to do so since they might "take over" or rob humans of their dignity. There are others, taking quite a different tack, who declare the enterprise of building intelligent machines, or simulating human brains in all their complexity, to be doomed to failure for various reasons. It must also be mentioned for the sake of the alert reader that we are proceeding on the assumption that if there exists a "machine" to do some task, there exists a computer program on a general purpose computer that is equivalent to that machine. (This particular assumption is a consequence of the so-called Turing's hypothesis which states essentially that corresponding to any mechanical procedure there exists an equivalent machine, called a Turing machine, and that every Turing machine can be simulated by a program on a general purpose digital computer provided with adequate memory.)

The philosophical issues in the problem of man-machine equivalence are varied and subtle and have been considered by many individuals. We refer the reader to anthologies by Anderson [3], Chandrasekaran and Reeker [4], Gunderson [5], and Dreyfus [6] for the dimensions of the problem. The moral implications of artificial intelligence essentially revolve around the following argument. If human intelligence can be equaled by a machine, there is apparently no reason why it cannot be surpassed. There exists a small minority which looks upon the prospect of a mechanical species superior to man, as man is superior to a cow, with either bovine acquiescence or with the excitement of a devout midwife during the birth of a heralded prophet. A variant of the moral position is that of Weizenbaum [7] who maintains that the metaphors used to view man control the direction man fashions for himself, and the metaphor of the machine which is at the root of artificial intelligence can be a most damaging one. The rapist, says Weizenbaum, will taunt the victim with "it is your dream, lady!"

In any event, it seems to us at least, that the day of a subversive HAL is further away than 2001, and in the meantime artificial intelligence is an exciting field self-charged with the great task of understanding the ways that the human mind works.

### Turing's Test

In many debates on artificial intelligence, an observer on the sidelines is often impressed with the fact that if only "intelligence" were precisely defined, much of the

debate would be needless. Alas, a satisfactory definition does not appear to be on the horizon. That leaves us with the problem of deciding whether a given machine or program is an intelligent one.

Some of the abilities of the human mind are manifested in pattern recognition, problem solving, language use, inductive inference, creative work as in writing poems, composing music, etc. Hence, naturally there have been programs which attempt to do one or more of these. Of course, not many humans are good at everything or even always good at the same thing. So one should not be guilty of the "superhuman human" fallacy in what we demand of the machine.

Turing [1] proposed a test in the form of an imitation game. There are three participants: the machine (A), a human (B), and an interrogator (C). The object of the game for the interrogator is to determine which of the other two is the human and which is the machine. It is the machine's objective to cause the interrogator to make the wrong identification. In order to concentrate on purely intellectual aspects, A and B are located in separate rooms, connected to C by means of a Teletype machine. The interrogator, C, questions both of them through the Teletype and receives answers also through the Teletype. The human would probably give truthful answers, but A is, naturally, constrained by no such limitation. If after, say, five minutes of questioning the interrogator does not have more than a 70% chance of making the right identification (the figures are Turing's), most people would agree that the machine has displayed intelligent behavior. Turing suggests that the question of whether computers can be programmed to pass such a test should replace the traditional question of whether computers can think. This, then, is an operational definition of intelligence.

On occasion, it is claimed that the machine has passed the Turing's test when its performance is indistinguishable from the human's in a specific task—say in very limited language use. This overlooks the fact that the difficulty of the task is a crucial aspect of Turing's test, the starting point of which has the question, "can machines think?" and not "can a machine do (a specific task) as well as a human?"

Of course, at the moment there exist no machines which can come anywhere near passing the test. Turing himself estimated the time when a machine could pass the test as nearly the end of the twentieth century.

In this article we outline some of the major approaches and programs that are available, so that the reader can estimate for himself the probable date of a machine passing the Turing test.

To impart a general feeling for the way that computers have been programmed to perform tasks intelligently, we briefly discuss game-playing programs. Later, we move to a general discussion of the various components of an intelligent program, discuss some work in cognitive simulation, and close with some typical state-of-the-art programs for various specific tasks. It is clearly not possible to give an exhaustive summary of the current work. However, we do feel that the activities chosen are generally typical.

## GAME-PLAYING PROGRAMS

Computer chess matches are common nowadays, even though, contrary to earlier optimistic predictions, the day when the world champion is a computer is still far away. But the computer *can* play a rather good game of checkers. Countless numbers of other games ranging from tic-tac-toe to Qubic, from poker to Stratego, have been programmed at various times. In order to illustrate the basic techniques used in programming computers to play games, Samuel's program [8] for checkers is briefly considered.

Many of the basic ideas were first set down by Shannon [2] in his discussion of a chess-playing program. Both chess and checkers are finite games, so in principle a technique that could be used is to consider all the possible moves, all the possible replies by the opponent, and so on until the entire game tree is generated. Each tip node must be one of the three: win, lose, or draw represented numerically by, say, 1, -1, and 0, respectively. The player then can work back recursively thus: a node which corresponds to the opponent's move is given the value which corresponds to the minimum of the values of the successors; a node which corresponds to the protagonist's move (possibly the computer's) is given the value which corresponds to the maximum of the values of the successor. The rationale is, of course, that the intelligent opponent will make that move which will be most beneficial to him and similarly for the protagonist. By this recursive evaluation procedure, all the alternatives for the protagonist at a certain board position can be evaluated and the move corresponding to the highest evaluation can be chosen. It is seen that this is the *best* possible move, whether made by a human or a computer. This method of evaluating back from tip nodes is called minimaxing.

However, neither chess nor checkers is such a trivial game. The reason is, of course, that the game tree, finite as it is, is enormous; the fastest computer will take centuries or more to compute all the alternatives. Thus one needs "intelligent" mechanisms to make the moves, rather than a blind search through all of the possibilities. Here is where Shannon's suggestions are important. Shannon suggested: consider all alternatives for a given depth, search all continuations to a fixed depth, evaluate each of the board positions to obtain a number indicating the "promise" of that position, minimax back to the successors of the node which corresponds to the state of the game, and thus choose the best move. The part of the above scheme which is specific to the game is the evaluation in which the knowledge and expertise of the programmer (or those borrowed from an expert) come into play.

There is a large number of variations of the above basic technique. For instance, instead of searching to a fixed depth, one might search until a so-called dead position is reached, i.e., the board position is stable in some sense; one could use more complicated back-up procedures; and so on. There are interesting techniques for computational efficiency, such as alpha-beta pruning procedures. Restrictions of space forbid a detailed examination of these and other very important techniques. We refer the reader to a text such as Slagle [9] for more details.

In checkers, an evaluation function may be of the following form [9]:  $6k + 4m + u$

where  $k$  is the king advantage,  $m$  is the (plain) man advantage, and  $u$  is the undenied mobility advantage. The undenied mobility of a player in a position is the number of moves he has such that the opponent can make no jumps in the successor positions. Thus this static evaluation function measures in some sense the "promise" of a board configuration. Instead of searching to a fixed depth, one might use some more involved termination criteria such as "stop searching if  $k$  levels have been reached and if the position is dead (i.e., no immediate jumps are available)."

In 1967 Samuel described a very powerful checkers program [8]. The termination criteria included game over, minimum depth, maximum depth, and dead position. There are many other heuristics controlling search. The features are man advantage, king advantage, mobility advantage, and a large number of others. Samuel formalized these features in his program, but they were obtained informally from checkers experts.

There are two aspects of "learning" incorporated into the program: generalization and rote learning. In generalization learning, the program improves its evaluation function (i.e., the weighting of features) by experimenting with it until its moves agree with the moves recommended by experts a large percentage of the times. The rote learning simply keeps in memory book moves (those recommended by experts) as well as positions which have already been evaluated by the program and which occur often in games, thus having an edge over the ordinary player and saving time.

## COMPONENTS OF AN INTELLIGENT SYSTEM

Minsky [10] in 1961 summarized progress at the end of the first decade of research in artificial intelligence. In his definitive review he identified the various elements of an intelligent program as: search, pattern recognition, learning, planning, and induction. This list is still a profitable one to examine. Minsky says:

A computer can do, in a sense, only what it is told to do. But even when we do not know exactly how to solve a certain problem, we may program a machine to *search* through some large space of solution attempts. Unfortunately, when we write a straightforward program for such a search, we usually find the resulting process to be enormously inefficient. With *pattern recognition* techniques, efficiency can be greatly improved by restricting the machine to use its methods only on the kind of attempts for which they are appropriate. And with *learning*, efficiency is further improved by directing search in accord with earlier experiences. By actually analyzing the situation, using . . . *planning* methods, the machine may obtain a really fundamental improvement by replacing the originally given search by a much smaller, more appropriate exploration. Finally, [by exploring] *induction*, we consider some rather more global concepts of how one might obtain intelligent behavior.

To Minsky's list we will add the additional element of *representation*.

The concept of heuristic programming, or more simply, a heuristic, can be conveniently illustrated in the framework of searching in a space of possible solutions. Consider the problem of programming a computer to solve the well-known 15 puzzle, which consists of 15 numbered, movable tiles set in a  $4 \times 4$  frame, thus leaving one

cell in the frame always empty. Any one of the tiles adjacent to the empty cell can be moved into the empty cell. Given the frame with tiles in an arbitrary arrangement, the puzzle consists of finding a sequence of moves such that the tiles are arranged in some specific (for example, increasing) order. An algorithmic solution to the problem especially appropriate to the computer, might consist of the machine generating from the initial configuration all the configurations that are obtainable from it. Each one of these configurations can thus be "expanded" into further configurations and so on. Of course, a simple check should be provided to make sure that configurations previously generated are not generated again. This generation procedure can be conveniently represented in the form of a finite tree. If the puzzle is solvable, then, as the tip node of the tree represents the initial configuration, one of the other nodes must represent the goal configuration. A path from the initial node to the "goal node" is then a solution. The whole procedure can be programmed in a computer.

But suppose the size of the tree were so large as to make impossible a complete search of all these paths in the form of generating all the nodes one by one and checking it to see if it is a goal node. Most interesting games like chess and checkers fall into this category, as mentioned earlier. Then for practical reasons one has to give up the algorithmic certainty of finding the solution and resort to other "intelligent" techniques. Humans, under these conditions, try out "plausible" moves rather than all the moves. Similarly, the program may be provided with a "heuristic" by which only a promising subset of the nodes is chosen for expansion. Note that heuristics are such that they may facilitate arriving at a solution, but there is no guarantee that a solution, even if it should exist, will be found.

For instance, a possible heuristic in the example just mentioned might be an estimate of how promising a node is in terms of whether it lies on a path from the initial to the goal node. Out of all available nodes, then, one might expand each time only that node with the best such estimated evaluation. For this puzzle an evaluation function that has been used with success is the number of tiles which differ from the goal configuration at each node; the assumption being that the smaller this number, the greater is the chance of reaching the goal node. Of course, it is not always true that the node with the best evaluation number lies in the start-to-goal path. Some of these nodes may even be dead ends. The point is, however, that the search is now considerably smaller. What has happened is that we have transmitted to the program some of our knowledge of how to go about finding a solution.

The outstanding feature of human problem solving is not only the knowledge we have of the objective facts of the world (which are easily programmed), but also the ability of bringing to bear various techniques for looking for a solution. Suitable heuristics embody this knowledge. Of course, it is not always necessary that the heuristics used in the machine be identical in every respect with the heuristics humans employ.

Once the concept of heuristics is introduced, then anything that helps to discover a solution, whether or not it is used by a human, is permissible. Later, when we discuss specific programs, we will see examples of heuristics which are in the form of procedures rather than numerical evaluation functions.

### Representations

One of the fundamental issues in artificial intelligence is the problem of representation of the problem-solution space so that search for a solution can begin. This is familiar to all of us in the form of the right and wrong ways to look at a problem. A classic example in the field is the following problem due to McCarthy [11]: Deform a  $2n \times 2n$  square board by removing one cell from each of two opposite corners. Suppose now we attempt to cover this board with  $1 \times 2$  tiles in such a way that the tiles neither overlap nor stick out over the edge of the board. Is such a covering possible? Readers who are unacquainted with the problem generally find it to be quite difficult. However, consider the following representation of this same problem. Think of the board as having alternate white and black cells (a checkerboard). Let the cells that were cut out be black. Similarly, let the tile be composed of one each of black and white cells. Given this representation, it is now simple to arrive at the solution. The mutilated board has two more white cells than black cells and any integral number of  $1 \times 2$  tiles must cover equal numbers of black and white cells. Hence, the impossibility of the suggested covering.

The above is a dramatic example of how often the right kind of representation can convert the problem from one of near-impossibility to near-triviality. Of course, most situations lie in the area in between. A good representation generally makes the search manageable, or creates conditions in which productive heuristics can be naturally introduced. In the above example almost all of the intelligence in the solution was used in arriving at the representation. Therefore an important question arises. How can a program be considered "intelligent" if the right representation is given to it by a programmer? A related question of interest is: How is one to program a machine to arrive at its own representation for a problem?

These are not trivial questions. We can only indicate some general considerations concerning the representation problem. First of all, most humans never arrive at the right representation for the above problem either. It is a safe bet that most people who know the solution to the above problem were given the representation by somebody else. Those who do come up with the given representation (there must have been at least one, obviously) probably arrived at the representation and the problem simultaneously when contemplating a checker board. Perhaps the right representation was built up from related problem situations which were part of the knowledge of the world that the specific originator possessed. However, almost nothing is known currently about credible models in which such representations may be established from more primitive ones to meet the demands of the problem. It is perhaps fair to say that progress in this area will come long after many other less difficult problems are solved.

Even with a suitable representation, the solution of a problem is generally not trivial, and substantial "intelligence" is called for in arriving at the solution in the form of right routes to take. Or in the framework of search, intelligence is required in determining the correct solution path in a tree or graph of possibilities. Here is where the various heuristics are helpful in controlling the size of the search.

## Search

We have introduced some of the concepts such as evaluation functions earlier. Nilsson [12] provides a comprehensive account of the mathematical aspects of the search problem. We briefly describe some of these here.

In artificial intelligence, search mechanisms can be employed in different formal structures. First of all, one might want to choose a set of parameters to extremize an objective criterion function. This is a problem familiar to workers in operations research and stochastic optimization, and an extensive literature is available on the subject. Wilde [13] provides a good introduction. The basic principle in the random search method is to choose some set of parameters and explore about that point in the parameter space to find the direction of steepest ascent (in the case of maximization) or descent (for minimization). Then change the parameters a certain distance along the direction of the gradient and repeat until there is no further improvement. There are two problems in this approach: the machine might be trapped at a local extremum rather than the global extremum. Also, what Minsky and Selfridge [14] call the "mesa phenomenon" might occur. This is a situation where a small change in the value of the parameters leads either to practically no change at all or to a large change in the value of the objective function. If the machine is in a flat region, then much aimless wandering could result.

The hill-climbing method may at best be only one of the methods an intelligent mechanism would employ, the various methods being arranged in hierarchies or recursive structures. Minsky [10] suggests that perhaps what may be hill-climbing in one level may have the appearance on a lower level of being a sudden jump of "insight."

A problem may often be formulated in an abstract state. For instance, in the 15-puzzle each possible configuration defines a possible state. The initial state and the goal state correspond, respectively, to the given and desired configurations. In order to change the states, we have operators which in this case are the allowed moves of the empty cell. Here the search is for a sequence of operators which would transform the initial state to the desired state, or, alternatively, the search may be viewed as that for the goal state since the generation of a state can be tagged with the operator to be used in generating that state. Various heuristic evaluation functions can be used to determine which among the possible states is worth arriving at, so that further applications of the operators promise to yield the goal node.

Another useful approach is that of problem reduction. This operates in the following way. Given a problem, various transformations may be applied to generate a collection of subproblems, hopefully all of lesser difficulty. Transformations may again be applied to the subproblems in the collection. This process may be repeated for any generated problem until the machine can trivially solve it or decides that it cannot be solved with the knowledge the machine has.

Suppose a problem  $P_i$  (which itself may be a subproblem for another larger problem) results on transformation in a set

$$\{P_{i_1}, \dots, P_{i_k}\}.$$

The transformations may be of two kinds. In one,  $P_i$  can be solved if all of the problems

$$\{P_{t_1}, \dots, P_{t_k}\}$$

can be solved. In a graphical representation, this would produce an AND-graph, where the node  $P_i$  has  $k$  branches emanating from it, the tips of which are labeled

$$P_{t_1}, \dots, P_{t_k},$$

and the  $P_i$  node has a marker identifying it as an AND-node. In the other kind of transformation,  $P_i$  can be solved if any *one* of the problems

$$\{P_{t_1}, \dots, P_{t_k}\}$$

can be solved. This produces an OR-graph, where the  $P_i$  node has a marker denoting it as an OR-node. Situations in between can be treated by combinations of AND-OR nodes, with increase in the number of levels, if necessary. The AND-OR graph generated in this manner thus has tip nodes which correspond either to problems which can be immediately solved, or to those which are not solvable by the machine. An AND-node is considered solved if *all* of its successor nodes are solved, and an OR-node is solved if at least one of its successor nodes is solved. With this recursive definition, it is possible to decide if the original problem is solved.

The search problem in this approach consists of finding an AND-OR graph to solve a problem from the myriad AND-OR graphs that can be generated. Again, various heuristic aids should be employed to cut down the search. There are further problems created by the possibility of having certain kinds of circularities in the graph which correspond to "circular reasoning." The text by Nilsson [12] discusses the intricacies in this approach.

Slagle's pioneering program SAINT [15], the task of which is to solve symbolic integration problems at the level of a student in freshman calculus, can be used to illustrate this problem-reduction approach.

The integration-by-parts rule, applied as a transformation, results in two subproblems and corresponds to an AND-node. Another AND-node is created by the decomposition rule which allows the substitution of a sum of integrals for an integral of sums of expressions. Other transformations create OR-nodes, and they represent various algebraic and trigonometric substitutions, division of numerator by denominator, completing the square, etc. They represent OR-nodes because given an expression, the application of each of the transformations results in another equivalent expression which is then examined for appropriateness for further transformations or labeling as solved or unsolvable nodes. Of course, various heuristics are necessary to decide which transformations are worth trying at any point in the attempt.

### Pattern Recognition

There are two specific and different senses in which we are interested in pattern recognition in the field of artificial intelligence. One of these is the sense in which one of the attributes that an intelligent system will have is to be able to solve problems

efficiently. In this sense, it is analogous to other attributes such as search, learning, and planning. The other interest in pattern recognition is that it describes the task of certain types of programs; e.g., a program to recognize handwritten characters, or a program to recognize fingerprints of individuals, or the recognition of particular features in a photograph. Many of the techniques used for realizing pattern recognition in these two senses may even be identical. Still, it is desirable to keep this distinction in mind.

Since Minsky wrote his review [10], there has been a general agreement given to the idea that pattern recognition is an important attribute of intelligent systems, but no systems yet exist in which this attribute has been deliberately designed as an integral part of a complex system. One of the reasons lies in the fact that pattern recognition as a term is rather loose in its meaning. Any conditional branching is, of course, a case of pattern recognition (although it is trivial). Heuristic tests to decide on the applicability of a specific operator in search situations can also be viewed as instances of pattern recognition.

In order for a machine to recognize patterns, the machine needs some sort of description of what the various patterns consist. The simplest approach is to begin by describing an object by a list of its properties. The problem of deciding which are the relevant properties is a similar problem, but one of lower complexity. Better still, a list of attribute-value pairs may be given for more detailed information. If the designer has definite ideas of what combination of properties constitutes the pattern he wants the machine to recognize, he can provide the machine with a recognition, or classification, logic. There exists an extensive body of material dealing with the methods by which to design this logic. In a geometric model, each object described by a set of attribute-value pairs, or simply a set of features, can be represented by a point in a suitable multidimensional space. Then different patterns, or classes, can be represented by some particular subspace of the multidimensional space. Each region corresponds to one of the pattern classes.

It is possible to use other subspaces, such as the linear subspaces suggested by Watanabe [16]. If one has available probabilistic knowledge concerning the distribution of feature values for each class, this knowledge can be used in a straightforward way to derive a decision logic to minimize the probability of misclassification or, if a loss function were used, to minimize the expected loss. For more details on the statistical classification techniques, see a recent text such as that by Duda and Hart [17].

A system such as the Perceptron, which was the subject of intensive research a decade ago, can be cast in the above geometrical model. On the basis of available examples of the patterns belonging to various classes, the system generally attempts to construct the best decision boundary in the form of a hyperplane to separate the known and given instances of the various pattern classes. The Perceptron algorithm is discussed later in this article.

In the case where there is partial availability of probabilistic information, various so-called learning techniques have been developed which can be viewed as making some sort of estimates of the distributions of features for various pattern classes.

So far we have discussed the pattern recognition problem as one of classifying an object as belonging to one of several prespecified classes. However, a more interesting

and more difficult problem is one of determining the existence of patterns in a mass of data. Of course, what is perceived as a pattern for one goal may not be so perceived for a different goal. Thus there are teleological considerations in pattern perception. Still, the general requirements for a system that will perceive patterns that might correspond to human perception, and in some cases might extend human capabilities, are the subject of much recent investigation. The problem is generally treated under the category of clustering algorithms, referring to their task of identifying clusters of points in a multidimensional space of features. The idea is that if the space of features is appropriate, then data points representing a pattern cluster together in that space. What constitutes a cluster is difficult to state precisely even in two dimensions where the human visual system works efficiently. In higher dimensional spaces the problem is very difficult. We refer the reader to Refs. 17 and 18 for further discussion of this problem.

The pattern recognition problem can be cast in a somewhat different model. One can view the existence of a specific pattern as being signaled not simply by the presence of features of certain values, but by specific interrelationship properties that govern how the various attributes are put together. This has been called the structural approach to pattern recognition. For instance, many of the optical character recognition machines for typed or printed characters have logics which first detect the presence of various elementary features (which itself is a pattern recognition problem, albeit of a lower level of complexity) such as strokes and loops, and, at the higher level in the decision logic, checks whether these features are joined in the appropriate way or in one of a number of appropriate ways. The decision logic itself is arrived at by experimentation on a large number of characters to obtain a structural description. Of course, these recognizers also incorporate various preprocessing and normalization routines, some of which may be statistical in nature. Further, they may also sometimes use context information to help in the recognition of characters.

A variant of the structural approach, at least in the theoretical sense, is the quest for a "grammar" of patterns. The basic idea is that in many situations there is an underlying basic, simple mechanism generating the elements of a possibly infinite pattern class. An example might be the line patterns observed in cloud chamber photographs. The formal system describing the rich variety of patterns of the traces of particles is governed by simple rules of particle physics. The rules expressing the generation of admissible cloud chamber traces can be considered a set of production rules in a grammar whose sentences are the traces. This insight is due to Narasimhan [19]. The usefulness of this idea is that a rather small, possibly recursive, description can be used to characterize a potentially infinite set of objects.

In this approach, variously called syntactic, linguistic, or grammatical, the process of recognition is a process of parsing. If a pattern can belong to one of many disjoint classes of patterns, each class characterized by a corresponding grammar, then a pattern of unknown classification belongs to that class for which there is a correct parsing of the pattern viewed as a sentence. The grammar written by Watt [20] to describe the generation of Nevada cattlebrands is one of the more interesting examples of the application of these morphological notions.

There are important differences between two-dimensional visual patterns and linear

strings which are the objects of study in traditional formal languages. The generalization of concatenation is not straightforward. There have been various attempts to come up with generalized two-dimensional grammars, but few of them have a naturalness which invites application to a variety of problems. The paper by Miller and Shaw [21] reviews some of the developments in the syntactic approach. For a discussion of the relationship between geometrical and structural models, see Ref. 22.

We have not said anything about an alphabet or a set of primitives for the generation of patterns. They are themselves best reviewed as patterns of lesser complexity. However, it is not always clear what should be the appropriate set of primitives for a class of patterns. If one has a good idea of the generating mechanism, it is generally possible to define an appropriate set of primitives. However, a significant portion of the work on syntactic pattern recognition is marred by a poor choice of primitives. For instance, some pattern description languages use lines oriented at different angles as a catchall set of primitives. It is true that all line drawings can be approximated by such lines. But such an approximation is generally cumbersome and difficult to work with. Such a description may be said to be as inappropriate as attempting to write an English grammar where the primitives are phonemes rather than words.

From the viewpoint of artificial intelligence, it would seem that pattern recognition approaches based on such formal systems are of limited value. Until a more universal theory of pattern recognition emerges, most successful applications of pattern recognition are likely to be based on heuristic concepts, tailored to specific applications. Guzman [23] is a good example of a dramatically successful pattern recognition program based entirely on heuristic concepts. In addition, the reader should be aware of rather successful systems that are in operation, such as the U.S. Post Office address reader which automatically sorts mail with addresses typed or printed in a wide variety of fonts and subject to significant noise.

### Learning and Inference

Like pattern recognition, the term "learning" has many meanings, from the simple to the profound. Systems where some parameters are adjusted as a function of past inputs and outputs can be considered learning systems. On the other hand, the processes involved in learning by humans—ranging from simple skills to complex concepts—are generally far more complex than this, even though it is undoubtedly true that for any learning phenomenon there does exist some abstract space in which some parameters have been updated. In this context we are looking for insights for building intelligent systems. Thus we will not discuss various reinforcement models of learning, which seem to us to be only of peripheral importance. Similarly, in the pattern recognition literature, considerable work has been reported on "learning" probability densities of the various classes. These learning algorithms are based on statistical theories of estimation and consist of various refinements for different *a priori* knowledge and structural assumptions.

One of the systems that attracted much attention in the early days of artificial intelligence is the previously mentioned Perceptron of Rosenblatt. There are numerous references and descriptions of this work. Reference 24 is typical. Rosenblatt attempted

to model, in some elementary aspects, the neural theories of learning advanced by Hebb who suggested that learning consists of the strengthening of certain neural pathways and the weakening of others as a function of experience. The notion of reinforcement played an important role in these theories. It was also the mechanism which resulted in the strengthening and weakening of pathways. Rosenblatt's model restricted itself to learning of patterns as represented in a two-dimensional retinal array of receptors.

Basically, patterns are first presented to the machine. The information from the receptors goes to a so-called associative unit which computes the presence or absence of various features which are themselves sets of retinal points, generally randomly chosen. The recognition unit weights the information about the features with a linear weighting function and the output is thresholded into classification signals. In the learning phase, if the classification is correct, no changes in the weights are made. If not correct, the weighting function is changed in an appropriate manner. This constitutes the reinforcement. If the paradigms from the two classes are linearly separable in the feature space induced by the associative units, then it can be shown—in fact, this is the content of the well-known Perceptron convergence theorem—that after a finite number of presentations of the patterns, the reinforcement algorithm will arrive at a decision logic separating the two classes correctly. There is also the property of generalization that is implicit in this model. That is, once the two classes have been separated, from then on paradigms of the two classes which were not in the training set will then be generally correctly classified as long as the training set of paradigms is representative of the two pattern classes.

One of the more successful uses of the concepts of learning in artificial intelligence is the previously described checkers-playing program of Samuel. Winston [25] describes a program whose learning occurs at a somewhat higher level. This program takes as input a two-dimensional scene which is an optical image of a group of three-dimensional geometrical objects. The first part of the program is the same as the program by Guzman [23] which decomposes the scene into three-dimensional objects. The second part of Winston's program learns structural descriptions. For instance, suppose we want to teach the program to recognize an arch. The trainer gives as input different examples of arches as well as different examples of nonarches, especially concentrating on near-misses. The program essentially analyzes each scene and computes a linked list where the links are various relations that hold between the objects. From various examples it computes the minimal description of an arch in terms of objects and their interrelationships. The importance of this program is that it emphasizes counter-examples and near-misses as important sources of successful learning. Further, the learning here is structural rather than updating of numerical parameters.

It is difficult to tell where structural learning leaves off and inference or, more specifically, inductive inference begins. The emphasis in inductive inference is to construct plausible models to explain an observed sequence of events and to predict the future behavior of that sequence. Almost all science does this. Hence the importance of inductive inference as a component of artificial intelligence systems.

Amarel [26], Simon [27], Kilburn et al. [28], and Hormann [29], among others, have done work on the problem of finding regularities in a body of data. These various

regularities are then used to enable prediction and to accommodate prior knowledge. Methods of prediction should describe regularities in a general way so that concepts or models are not restricted by linguistic constraints.

Solomonoff [30] has done very important theoretical work in this area. He views an induction problem as one of extrapolating a long sequence of symbols. Given a sequence  $S$ , one of his models considers a universal Turing machine which, when certain strings are presented to it as input, outputs  $S$  and continues. One can regard the input strings as "models" of  $S$  in the sense that they explain  $S$ . In order to extrapolate to the next symbol, one has to weigh the predictions made by the different input strings in a suitable way. One of the theories gives them a weight proportional to the "likelihood" of the input string that caused them. For a binary input string of length  $N$ , this weight will be about  $2^{-N}$ . This also satisfies the "Occam's razor"; i.e., the explanations that are simplest (shortest in this case) are weighted most heavily in obtaining the prediction. It can also be shown that as the length of the string  $S$  becomes large, it does not matter very much which specific universal Turing machine is used. In other words, predictions made by different machines will tend to become identical as  $S$  becomes larger. Solomonoff also proposed more complicated and plausible models of inductive inference. While theoretically very interesting, none of these models has yet been used in a practical way in any artificial intelligence systems.

In connection with learning systems, it is important to mention various experiments on concept learning. In research on human concept formation, the subject is given a set of objects that have been divided into two classes by some classification system unknown to him. Concept formation is said to have occurred when the unknown classification rule is discovered. For information on work in this area, see Hunt and Hovland [31].

### Planning and Problem Solving

Planning or problem solving constitute the heart of any intelligent system. The problem reduction approach discussed earlier is also closely related to the current discussion. In planning, the machine must be able to work with models of the task, the models being any of three kinds—analogous, semantic, and abstract.

So far, the most successful examples of problem-solving behavior are those exhibited by humans. Simulation of cognitive behavior is thus a field which is closely related to artificial intelligence, sometimes indistinguishable from it in subject matter. However, artificial intelligence research attempts to write programs to accomplish intelligence-requiring tasks without concern about whether the mechanisms employed have correspondences in the human cognitive system. On the other hand, the major objective of cognitive simulation is to obtain models of human thought processes.

In discussions of planning and problem solving, the ideas can be best illustrated by examples of programs that lie at the intersection of cognitive simulation and artificial intelligence.

The best known of the theorem-proving programs is the Logic Theorist (LT) program of Newell, Shaw and Simon [32]. The task is to prove theorems in the logical

system called propositional calculus. The specific system used was that of Russell-Whitehead which consists of five axioms and three rules of inference. There is an alphabet of symbols to denote logical propositions, connectives, etc. There are rules governing the formation of so-called well formed formulas (wff's). The axioms are wff's and the rules of inference describe the permissible transformations that can be performed on wff's to obtain further wff's. A wff is a theorem if it can be obtained after a finite number of applications of the rules of inference on axioms and, if necessary, other theorems.

The axioms and inference rules can be applied in a tremendous number of sequential permutations. The problem of selecting the particular sequence cannot therefore always be handled algorithmically. Even if it could be, humans do not prove theorems in any nontrivial domain in this manner.

In LT the main heuristic is "working backwards" from the theorem to be proved. For a proof to theorem T, the LT program searches among available theorems to find one from which T can be easily deduced, thus hopefully solving the problem. If it cannot, then it attempts to find one or more subproblems which are propositions from which T may be deduced. Now the task is to prove these propositions. Thus, structurally, it is a problem-reduction approach. Other heuristics are used to select promising subproblems, remove hopeless subproblems, and choose promising methods to attempt to prove the subproblems.

LT has a flavor of hierarchies of goals and subgoals, which became a major aspect of a later problem-solving system by Newell and Simon [33], named GPS for General Problem Solver. Ernst [34] has continued further extensive research on GPS. Although this particular line of research seems to have reached a kind of plateau, GPS-like models have become important in many aspects of psychological model building.

The major elements of GPS follow. Associated with a problem (which must be given to the program in a suitable representation) are a set of operators, so-called differences, difference orderings, and a table of connections. Operators are the actions available to the program. The solution of a problem must be in terms of a sequence of operators that must be applied to the initial configuration to produce the goal configuration. The set of differences specifies the ways in which a configuration can differ from the goal configuration. Difference ordering tells the program which differences are more crucial, more fundamental, or more difficult as the case may be. The table of connections specifies which operators are applicable to reduce which differences.

The philosophy behind GPS has been aptly described as a means-ends analysis. The program computes the differences between the initial and goal configurations, chooses the most important difference, and creates the subgoal of reducing that difference. Operators suggested by the table of connections are used to reduce the difference. There is a hierarchy of top goals and subgoals set up, and the simplest subgoal is solved by the application of suitable operators.

While GPS has been an important source of ideas for both artificial intelligence and psychology (see the book by Miller et al. [35] for the effect of information-processing models in psychological model making), it has not turned out as extensible as was originally hoped. It is fair to say that the term "General" is more a label than a descriptor.

Next we give an example of a program where the idea of a model contributes significantly to its success. The model is a semantic one and the program is the geometry theorem prover of Gelernter et al. [36]. In addition to using heuristics such as working backwards, generation of subproblems, etc., the program constructs a diagram much like a geometry student would. The advantage of this is that the plausibility of subproblems can be checked for the constructed diagram, and a large percentage of the subproblems can be eliminated on the basis of this semantic model. Thus, much like the case of the human, the diagram suggests worthwhile lines of attack while decreasing chances of attempting to prove inherently false propositions.

While we are discussing cognitive simulation, a pioneering program to solve the kind of problems that often constitute IQ tests should be mentioned. This is the Geometric Analogy program of Evans [37]. The typical question in such a program is: Figure A is to Figure B as Figure C is to which one of five possible answer figures? Here A, B, C, and the answer figures are various patterns formed from simple geometrical objects. To give a trivial example, Figure A might be a triangle inside a square, Figure B a triangle to the right of a square, Figure C a circle inside a triangle, and the answer figures are various combinations of objects, at least one of which should obviously be a circle to the right of a triangle. We can assume that the objects are normalized for size, etc. The first part of the program analyzes the descriptions and generates various relational descriptions holding among the parts of each figure. The second part first generates one or more rules which transform Figure A into Figure B. These rules are in the form of instructions to add, remove, move, or alter the objects and the properties. Then each rule, if possible after generalization, is applied to Figure C and the result is checked for a match with any of the answer figures. If more than one answer is found, an economy rule in the form of simplest transformation is invoked to find the "best" answer.

## PROGRAMS TO UNDERSTAND NATURAL LANGUAGE

In the fifties and early sixties, substantial effort was directed at automatic, high-quality computer or machine translation from one natural language to another. Whatever else these programs contributed to the understanding of language, they were generally considered to be failures at the task they set for themselves. It became gradually clear that the task of translation is exceedingly complicated and that substantial advances either in linguistics or in artificial intelligence techniques or both are required for the objective to be satisfactorily met.

On the other hand, programs to "understand" limited-discourse, limited-format natural language could be written and provided stepping stones for larger and more complex systems. We give a brief description of these systems which are mainly meant to answer questions about a data base. The organization of our discussion closely follows that of Winograd [38].

In the category of special format systems are such programs as **BASEBALL** [39]

which had a data base consisting of baseball results and treated questions as "specification lists" where the blanks were to be filled by retrieving data from the base. The program SADSAM [40] stored information about kinship structure in a network form and answered questions about simple relationships between people. The program of STUDENT [41] lies within the domain of discourse of algebra, and stores and retrieves linear equations to solve problems in that domain. The most striking of these systems is perhaps ELIZA [42] since it could, for a time at least, simulate the behavior of a casual, disinterested participant in a cocktail party conversation. It has a stored script of keywords which is changeable as the domain of discourse changes. The text is read and inspected for the presence of a keyword. If a keyword is found, the sentence is transformed according to rules associated with the keyword. If the input is "I'm depressed," the program's response would be, "I'm sorry to hear you are depressed"; of course, certain syntactical checks are made as part of the transformation. If no keyword is found, a noncommittal content-free remark may be printed out. The transformation of sentences are template based. If the input is "I am - - -," the response might be, "How long have you been - - -?", no matter what the blanks contain.

In the category of text-based systems, PROTOSYNTHEX [43] has an index which contains information about the places where each "content word" is found in the text. When a question is put to the system, the content words are extracted and the sentence in the text containing the most number of these words in common with the query is generated after checking for the appropriate syntactic relationships. Semantic network [44] traces the shortest path between two words, where the input is in the form of a pair of words rather than a question. The implicit question is the semantic relationship between those words.

In the so-called limited logic systems, immediate inference capabilities are present. Thus these systems can perform very modest deductions to answer simple questions, instead of being limited to a table look-up procedure for answering questions. Raphael's SIR [45] is such a program which can answer questions using simple logical relations.

In the general deductive system category, perhaps the most studied approach is that of using the Resolution Principle of first-order predicate calculus to make general inferences. In this approach, an example of which is QA3 [46], the data base is expressed as a set of first-order predicate calculus formulas. A question is then converted to an existentially quantified formula which is to be proved as a theorem given the data base formulas as axioms. If the theorem can be proved, then there exists a straightforward technique called the answer extraction process which generates an answer, the existence of which is proved by the theorem prover. Clearly, the heart of the approach is the theorem-proving section which gives it general inferential capability. The theorem-proving section uses the Resolution Principle which is a powerful rule of inference for the first-order predicate calculus. However, with all the available heuristics, the performance of resolution-based theorem provers on anything approaching realistic data bases has been unsatisfactory. Thus these question-answering systems have proved to be as weak as their weakest link, the theorem-proving section.

Winograd then identifies a class of systems called procedural deductive systems. These systems use procedural information ("how to do it" information) which is

expressed in ways which do not depend on the peculiarities and special structure of a particular program or subject of discussion.

Winograd's program to understand natural language incorporates some of the most promising recent ideas to come out of artificial intelligence research for dealing with language. Admittedly, the domain of discourse is limited in that it deals exclusively with a world containing blocks of different shapes and colors, such as blue pyramids and red cubes. However, the ideas suggested do have much promise of generalization.

One of the features of the system is that while syntactic and semantic routines are used, when a sentence is being processed these routines are used in an *interlaced* manner. In most previous systems the sentence would be first parsed by the syntactic routines, and then "understood" by the semantic routines. Here, on the other hand, the parser might call on semantic routines to help disambiguate, and the latter might call on the grammatical routines to help in understanding the sentence. The parser is based on a grammar called a systemic grammar where the syntactical units have been designed with an eye for "carriers of meaning." Another part of the program is called PLANNER which is a language used for representing and manipulating complex meanings. It is a deductive system used by the program at all stages of the analysis, both to direct the parsing process and to deduce facts about the world. Much of the knowledge of the system is in the form of PLANNER theorems. This knowledge, it must be emphasized, includes procedural knowledge in the form of ways one might prove or deduce certain things in that world. We refer the reader to Ref. 38 for a well-written description of the program, along with provocative discussions about the meaning of semantics in such systems.

## DENDRAL: A PRACTICAL APPLICATION

Much of the research described so far is theoretical and has no real claim to immediate practical consequence. To remedy this defect, we turn our attention to a program called DENDRAL [47] which attempts to discover molecular structures in organic chemistry from mass spectrometric data obtained when a sample is fragmented by a bombardment of electrons. The performance of the program is comparable to that of professionals in mass spectrometry for certain classes of organic molecules.

The program has four parts: the preliminary inference maker, the hypothesis generator, the predictor, and the evaluator. The preliminary inference maker interprets the data tentatively in terms of the presence or the absence of key functional groups. The hypothesis generator develops all the topologically possible isomers of the same empirical formula. Then, it searches through a tree of molecular structures, it screens out implausible ones and produces a list of molecular structures that are candidate hypotheses to explain the empirically obtained spectrum. The predictor computes what the theoretical mass-spectrum should be for each candidate hypothesis. The evaluator, by means of a heuristic hierarchy, matches the theoretically expected with the empirically obtained spectrum for each candidate hypothesis. A subset of candidates is selected and ordered, and the list is then printed out.

## CONCLUDING REMARKS

The literature on artificial intelligence is very extensive and of necessity we have had to be selective in our presentation. Our discussion has had as its objective to delineate for the reader some of the major issues in the field, and to give a flavor of current activity by describing some programs and outlining some of the main approaches that have proved promising. The reader who wants to pursue the matter should consult recent texts [9, 12], anthologies [48, 49], and proceedings of recent symposia such as the International Joint Conference on Artificial Intelligence and the Machine Intelligence Seminars at Edinburgh University.

We have discussed in what we believe are fairly representative terms the various features of artificial intelligence research and activity and have described, although sketchily, some of the better known and more typical or successful approaches in this field. We would hope that after reading this article the reader will be able to evaluate for himself the justification for building machines which exhibit intelligence. The reader will also be in a position to evaluate the extent to which current accomplishments vindicate the underlying assumptions in this field. Finally, it is likely that valuable insight into the operation of the human brain may result from study of the research on artificial intelligence.

A reasonably extensive (and we believe typical) although far from exhaustive bibliography is included as the references in this article. An exhaustive bibliography would include hundreds of references and would be misplaced in this review. The references chosen were those with which the authors are best acquainted, and we accept the fact that we may have omitted a number of very important and highly useful references.

## ACKNOWLEDGMENT

The work of B. Chandrasekaran in preparation of this article was supported by Grant AFOSR-72-2351 of the United States Air Force Office of Scientific Research.

## REFERENCES

1. A. M. Turing, Computing machinery and intelligence, *Mind* **59**, 433–460 (1950). Reprinted in *Computers and Thought* (E. A. Feigenbaum and J. Feldman, eds.), McGraw-Hill, New York, 1963.
2. C. E. Shannon, Programming a digital computer to play chess, *Phil. Mag.* **41**, 356–375 (1950).
3. A. R. Anderson (ed.), *Minds and Machines*, Prentice-Hall, Englewood Cliffs, N.J., 1964.
4. B. Chandrasekaran and L. Reeker, *Artificial Intelligence—A Case for Agnosticism*, to be published. Available as a tech report from The Ohio State University Computer and Information Science Research Center, Columbus, 1972.
5. K. Gunderson, *Mentality and Machines*, Doubleday Anchor, Garden City, N.Y., 1971.

6. H. Dreyfus, *What Computers Cannot Do*, Harper and Row, New York, 1971.
7. J. Weizenbaum, On the impact of the computer on society, *Science* **176**, 609–614 (1972).
8. A. Samuel, Some studies in machine intelligence using the game of checkers II, *IBM J. Res. Develop.* **11**, 6 (1967).
9. J. Slagle, *Artificial Intelligence*, McGraw-Hill, New York, 1971.
10. M. Minsky, Steps toward artificial intelligence, *Proc. IRE* **49**, 8–30 (1961). Reprinted in *Computers and Thought* (E. A. Feigenbaum and J. Feldman, eds.), McGraw-Hill, New York, 1963.
11. J. McCarthy, *A Tough Nut for Proof Procedures*, Stanford University AI memo no. 16, 1964.
12. N. Nilsson, *Problem Solving Methods in Artificial Intelligence*, McGraw-Hill, New York, 1971.
13. D. J. Wilde, *Optimum Seeking Methods*, Prentice-Hall, Englewood Cliffs, N.J., 1964.
14. M. Minsky and O. G. Selfridge, Learning in random nets, in *Proceedings of the 4th London Symposium on Information Theory* (C. Cherry, ed.), Academic, New York, 1961.
15. J. Slagle, A heuristic program that solves symbolic integration problems in freshman calculus, *J. Ass. Comput. Mach.* **10**, 507–520 (1963).
16. S. Watanabe, Karhunen-Loeve expansion and factor analysis, in *Transactions of the 4th Prague Conference*, 1967, pp. 635–660.
17. R. Duda and P. Hart, *Pattern Classification and Scene Analysis*, to be published.
18. G. H. Ball, Data analysis in the social sciences: What about the details? in *Proceedings of the Fall Joint Computer Conference*, Vol. 27, part I, AFIPS Press, Montvale, N.J., 1965.
19. R. Narasimhan, Syntax-directed interpretation of classes of events, *Commun. Ass. Comput. Mach.* **8**, 166–172 (1965).
20. W. C. Watt, *Morphology of the Nevada Cattlebrands and Their Blazons, Part One*, National Bureau of Standards Report 9050, Washington, D.C., 1966.
21. W. F. Miller and A. C. Shaw, Linguistic methods in picture processing—A survey, in *Proceedings of the Fall Joint Computer Conference* AFIPS Press, Montvale, N.J., 1968.
22. L. Kanal and B. Chandrasekaran, On linguistic, statistical and mixed models for pattern recognition, in *Frontiers of Pattern Recognition* (M. S. Watanabe, ed.), Academic, New York, 1972.
23. A. Guzman, Decomposition of a visual scene into three-dimensional bodies, in *Proceedings of the Fall Joint Computer Conference*, AFIPS Press, Montvale, N.J., 1968.
24. F. Rosenblatt, Perceptron experiments, *Proc. IRE* **48**, 301–309 (1960).
25. P. H. Winston, *Learning Structural Descriptions from Examples*, Massachusetts Institute of Technology, Project MAC-TR-76, 1970.
26. S. Amarel, An approach to automatic theory formation, in *Principles of Self-Organization* (H. Von Foerster and G. W. Zopf, eds.), Pergamon, New York, 1962.
27. H. Simon, Experiments with a heuristic compiler, *J. Ass. Comput. Mach.* **10**, 493–506 (1963).
28. T. Kilburn *et al.*, Experiments in machine learning and thinking, in *Information Processing*, UNESCO, Paris, 1960.
29. A. Hormann, Programs for machine learning, *Inform. Control*, Part I, **1962**, 347–367; Part II, **1964**, 55–57.
30. R. J. Solomonoff, A formal theory of inductive inference, *Inform. Control*, Part I, **1964**, 1–22; Part II, **1964**, 224–254.
31. E. B. Hunt and C. I. Hovland, Programming a model of human concept formation, in *Computers and Thought* (E. A. Feigenbaum and J. Feldman, eds.), McGraw-Hill, New York, 1963.
32. A. Newell, J. C. Shaw, and H. Simon, Empirical exploration with the logic theory machine, in *Computer and Thought* (E. A. Feigenbaum and J. Feldman, eds.), McGraw-Hill, New York, 1963.
33. A. Newell and H. Simon, GPS, A program that simulates human thought, in *Computers and Thought* (E. A. Feigenbaum and J. Feldman, eds.), McGraw-Hill, New York, 1963.
34. G. W. Ernst and A. Newell, *GPS: A Case Study in Generality & Problem Solving*, ACM Monograph, Academic, New York, 1967.
35. G. A. Miller, E. Galanter, and K. Pribram, *Plans and the Structure of Behavior*, Holt, New York, 1960.
36. H. Gelernter, J. R. Hansen, and D. W. Loveland, Empirical explorations of the geometry-theorem proving machine, in *Computers and Thought* (E. A. Feigenbaum and J. Feldman, eds.), McGraw-Hill, New York, 1963.

- 37 T. G. Evans, A program for the solution of geometric-analogy intelligence test questions, in *Semantic Information Processing* (M. Minsky, ed.), M.I.T. Press, Cambridge, Mass., 1968.
38. T. Winograd, Understanding natural language, *Cognitive Psychol.* 3, 1-191 (1972).
39. P. F. Green *et al.*, BASEBALL: An automatic question answerer, in *Computers and Thought* (E. A. Feigenbaum and J. Feldman, eds.), McGraw-Hill, New York, 1963.
40. R. K. Lindsay, Inferential memory as the basis of machines which understand natural language, in *Computers and Thought* (E. A. Feigenbaum and J. Feldman, eds.), McGraw-Hill, New York, 1963.
41. D. G. Bobrow, Natural language input for a computer problem solving system, in *Semantic Information Processing* (M. Minsky, ed.), M.I.T. Press, Cambridge, Mass., 1968.
42. J. Weizenbaum, ELIZA, *Commun. Ass. Comput. Mach. ACM* 9, 36-45 (1966).
43. R. F. Simmons *et al.*, An approach towards answering English questions from text, in *Proceedings of the Fall Joint Computer Conference*, Spartan, New York, 1968, pp. 441-456.
44. M. R. Quillian, Semantic memory, in *Semantic Information Processing* (M. Minsky, ed.), M.I.T. Press, Cambridge, Mass., 1968.
45. B. Raphael, SIR : Semantic information retrieval, in *Semantic Information Processing* (M. Minsky, ed.), M.I.T. Press, Cambridge, Mass., 1968.
46. C. Green and B. Raphael, The use of theorem proving techniques in question-answering systems, *Proceedings of the 23rd National Conference of Association for Computing Machinery*, Thompson, Washington, D.C., 1968.
47. B. Buchanan, G. Sutherland, and E. A. Feigenbaum, Heuristic Dendral: A Program for Generating Explanatory Hypotheses in Organic Chemistry, in *Machine Intelligence* (B. Meltzer, ed.), American Elsevier, New York, 1969.
48. M. Minsky (ed.), *Semantic Information Processing*, M.I.T. Press, Cambridge, Mass., 1968.
49. H. Simon and L. Siklossy, *Representation and Meaning*, Prentice-Hall, Englewood Cliffs, N.J., 1972.

*B. Chandrasekaran and M.C. Yovits*

## ARTIFICIAL LANGUAGES\*

### INTRODUCTION: WHAT IS AN ARTIFICIAL LANGUAGE?

From a traditional linguistic viewpoint the concept of artificial language is easy to define, by exclusion and by enumeration: All languages not on the list of natural languages are artificial. The list of natural languages, well known and established for a long time, contains English, Russian, Twi, French, Sanskrit, and many other languages, dialects, and so forth. In other words, a natural language develops historically, in an uncontrolled way, within a human community whose definition can

\* Most of this article first appeared in *Encyclopedia of Library and Information Science*, Vol. 1 (A. Kent and H. Lancour, eds.), Dekker, New York, 1968.

be ethnic, geographic, national, or purely linguistic. On the other hand, an artificial language appears as the planned construct of a single author (the "language designer") or a group of authors. It is created, in its entirety, at a given moment in time and subsequent changes to it take place in a controlled manner, through explicit decision by language designers.

The above-mentioned definition is unsatisfactory at one point: It implies the existence of a definition of language. Therefore the statement "an artificial language is a language not on the list of natural languages" is not complete or consistent. We thus add to the definition the following requirements: An artificial language must satisfy a scientific definition of language which will be given later.

Let us examine some questions arising from the definition, e.g., how far does the definition extend and are there any languages which may seem intuitively artificial and yet are not included in the artificial group by our definition? In this connection we introduce two diametrically opposed viewpoints: natural language imitation and formal notation.

Natural language imitation cannot be used as a defining (or rejecting) specification of artificial languages, since it does not distinguish Esperanto (or Volapuk, and some other languages) from restricted English, which might be used in connection with instructing computers. For the purposes of this encyclopedia, the distinction is essential; we want to exclude Esperanto from the artificial group, and we want to include restricted English. Consequently, even if Esperanto satisfies intuitively the notion of artificial language, it is ruled out on another basis: If it would be put to use, as proposed by its author, changes to the language would immediately start, as in any natural language, in a totally unplanned and uncontrolled way. So, after a short, initial, "artificial" period, the language would quickly become a full-fledged natural language. Esperanto and its sisters are thus ruled out by the stipulation "controlled language change." Restricted English, on the other hand, fully satisfies all the criteria of the definition; it is thus retained as an artificial language.

The definition removes traditional, widespread mathematical notation from the artificial group. Although it can be defined, scientifically, as a language, in discourse it occurs in the midst of English text. Furthermore—and this is the important point—it is not the planned construct of a single author or a research team. Mathematical notation develops similarly to natural language, but within a small professional community. Sporadically and historically, mathematicians add new features to the notation, continuing from earliest time to the present. Sometimes the new features gain universal acceptance, but this is far from being the case all the time. When a new feature *does* gain acceptance, this does not take place through the deliberate decision of some language-designing authority, but through practice and popularity of the notational innovation. Consequently to a computer user mathematical notation is inherited from the past, learned at school at a relatively young age, and consecrated by usage. To him mathematical notation is natural, unlike the programming languages he must learn from a special manual.

In restricted English most language designers include traditional mathematical notation as part of the natural language-imitating schema. Substantial segments of mathematical notation are also included in most artificial programming languages.

One can thus state that parts of the notation easily lend themselves to formalization and insertion into artificial languages. But this is true even for some subparts of natural language. Mathematical notation in its entirety, as a total concept, is not viewed as an artificial language.

After these exclusions the most significant language types included in the artificial group are languages used to instruct computers and languages used in meta-mathematics and formal logic. Languages used in library and information science, the subject matter of this encyclopedia, are mostly computer-instructing languages, due to the heavy penetration of computer usage in that area. Consequently information retrieval and other library languages are included in the group of computer-instructing artificial languages.

In the next two main sections of this article, both aspects of the definition of artificial language will be investigated. The first part of the definition requires that the artificial language be—above all—a language. This problem is the subject matter of the following main section. The second part of the definition emphasizes the planned and controlled life of an artificial language, from its birth throughout its existence. This problem, and most of its aspects, is considered through a detailed elaboration on programming (i.e., higher level computer-instructing) languages, the most substantial relevant artificial language group from the viewpoint of this encyclopedia.

## THEORETICAL ASPECTS OF ARTIFICIAL LANGUAGES

### Aspects of Metalinguistics

#### *Linguistics and Metalinguistics*

Until very recently no precise definition of language was available. Linguistics, historically a cover name for all sciences dealing with languages, primarily studied individual languages or reconstructed ancient, lost languages. Despite occasional insight by great philosophers into the “universal” nature of language (i.e., into the common element present in all languages), it is hard to speak of “general” linguistics—a commonly admitted term today—during earlier periods. The most we could observe then is a commonly accepted linguistic methodology.

During the first half of the twentieth century, linguistic methodology itself—previously a relatively uncontroversial subject—started to be questioned and examined. Furthermore, the first truly artificial languages appeared in the domain of formal logic, playing a metamathematical role, i.e., describing mathematics. The co-occurrence of these two phenomena, the problems related to them, and the remarkable development of formal logical tools all created an atmosphere where a precise, scientific language-defining framework appeared as a necessity and a possibility. Practical problems, mainly due to the appearance of computers, contributed to the atmosphere.

Metalinguistics was thus born under a double influence: (1) the progress of formal logic as a metamathematical system, by means of artificial languages based on logical formalism; (2) the need for a precise, "scientific" linguistic science, following the example of the development of physics. The composition of the term "metalinguistics" reveals its meaning: the study of what is "beyond" linguistics; in other words, the definition of an underlying framework within which linguistics of individual languages can then freely operate.

As of today the terms linguistics, metalinguistics, general linguistics, theoretical linguistics, and even mathematical linguistics are almost synonymous. The difference between natural and artificial languages is much less striking from a metalinguistic viewpoint than from other viewpoints.

### *History of Metalinguistics*

The period during which linguistic methodology gradually developed into something very close to metalinguistics started in the 1930s. During this period the American structuralist school of linguistics appeared on the scene. This school tried to ban semantics from linguistics; it held that meaning analysis is irrelevant from the linguist's viewpoint; the business of the linguist is to formulate laws and criteria concerning what a logician would call the "well-formedness" of sentence on a purely syntactic basis.

The main syntactic tool created by the structuralist school to achieve this goal is "immediate constituent analysis." The principal idea is the following: The basic language unit is the sentence. On the next lower level, sentences are composed of clauses, then clauses of phrases, phrases of words, words of stems and endings, and so forth. At each level a sequence of items is named by a unique descriptor, which is itself an item in some coherent sequence at the next higher level. The lower level items are the constituents of the higher level naming item; the constituents are "immediate," since they occur in the next lower level.

Constituent analysis tried thus to grasp the formal, structural aspects of sentence formation without any reference to meaning. It essentially failed in its task; what it accomplished was so-called string taxonomy. Various strings, composed of words and descriptors (e.g., noun phrase, verb, relative clause, and so forth) get classified; such a string belongs to the class of syntactic units whose name is the higher level item of which the string members are the constituents. The historical importance of constituent analysis is, however, quite significant.

During the same period, considerably sophisticated logical languages appeared, mainly through the work of Carnap [1] and his Vienna school of logic. As stated before, these languages were originally proposed as metamathematical systems, i.e., for the description of mathematics. Afterwards, various proposals by Carnap's followers offered them as metalinguistic tools for the description of natural language. Later developments have proved that languages generated by Carnap's logical syntax (as opposed to linguistic syntax) could be partially described by linguistic formalisms close in spirit to the linguists' immediate constituent analysis. At the time constituent analysis was invented, the formal tools necessary to specify mathematically constituent

analysis were not yet available, or at least they were not known by linguists. Supplied with such formal tools, immediate constituent analysis is a full-fledged metalinguistic theory.

The next major influence on the development of metalinguistics resulted from the invention of the stored program digital computer. Logicians defined artificial languages describing abstract models of computing machines. Restricting the original "Turing machine" model of computing automata, most of these languages have common metalinguistic properties, i.e., they all belong to the same family, the group of "regular" or "finite state" languages. On the other hand, programming language specialists prepared formalized grammars for the description of programming languages. The first and most well-known such formal grammar is the so-called "Backus Normal Form" [2] for the description of the programming language ALGOL. Subsequent developments produced an *a priori* not obvious result: grammars used by programming language designers turned out to be based on the same principle as the linguists' immediate constituent analysis; furthermore, the regular or finite state languages invented by logicians for the description of models of computing automata turned out to be a special, restricted case of formalized immediate constituent analysis.

It is obvious at this point that a unifying synthesis became necessary, grouping in a single theory all these results, approaches, and trends. The synthesis has been actually accomplished by Professor N. Chomsky of Massachusetts Institute of Technology. In the following subsection we are going to describe in some detail the elements of metalinguistic theory according to Chomskian principles [3].

### Theory of Syntax

#### *Notions of Language and Grammar*

Modern linguistic theory does not exclude semantics from linguistics, as the structuralist school attempted to do. It does not fall, however, into the other extreme of trying to interface linguistics with such disciplines as ethnology or sociology, which would require making the subject matter of these sciences the direct concern of the linguist. Some contemporary linguists hold that this is the way linguistics should go. Between the two extremes modern linguistics still gives priority to syntax, but its attitude is to discover how far syntax can go in laying the foundation of meaning interpretation. Somewhere, somehow, a borderline must exist beyond which meaning cannot be captured anymore by linguistic means; the factual, extralinguistic knowledge of language users intervenes at that point. Beyond that line meaning interpretation phenomena are of no direct concern to the theoretical linguist.

Modern linguistic theory also discards Carnap-type logical syntax (and the artificial logical languages they generate) for metalinguistic purposes. The most such meta-languages could accomplish is to map a tiny subpart of natural language onto formal logic. This subpart is the one closest to the reasoning capability expressable by natural language; it essentially comprises narrowly interpreted and disambiguated conjunctions, predicates, and so forth—in other words, that part of natural language which

historically motivated the creation of formal logic at all because of ambiguity and unprecisioness.

In the case of artificial programming languages, most language designers proceed in the way natural language linguists do, avoiding the identification of the language with structures determined by formal logic. The one exception in this respect is the philosophy of Professor J. McCarthy of Stanford University (formerly of Massachusetts Institute of Technology) in connection with the programming language LISP [4].

Modern linguistics does rely, however, on some other results of formal logic—not to identify subparts of languages with logical structures, but to specify and define the syntactic mechanism incorporated into metalinguistic systems. According to this view, the notion “language” is defined as follows: Given a vocabulary of words, form the set of all possible strings obtainable through concatenation (juxtaposition) of the words. A language is a distinguished subset of this set. Each string of words entering the subset is a sentence of the language. The problem is now how to distinguish this subset, i.e., what makes a string of words a sentence in the language.

A person, once he learns a language, can usually distinguish immediately sentences from mere strings of words. Thus an English speaker immediately recognizes that the string

“The program not is debugged”

is not a sentence, even though he may be capable of guessing its meaning. Similarly a person familiar with mathematical notation would recognize that there is something unusual and incorrect about the formula

$$I(a) = \int_b^c f(u,v) \, dx$$

In the latter case it is much harder to accomplish automatic error correction. The first task of metalinguistics is to replace the intuitive “decidability” property of human language users by something formal and precise.

The problem is to replace the infinite number of sentences, which comprise the language, by something finite and masterable and then to establish a connection between language and its finite representative. The fact that the number of sentences in a language is, at least theoretically, infinite is quite clear: The only way to limit this number is to limit sentence length. Now, given a sentence of some length, it can always be made longer by some insertion not violating the syntax rules.

The finite device representing a language is a grammar of that language. The grammar is composed of a finite number of rules. These rules are the formalized equivalent of the traditional notion of grammar rule. Rules can be executed; that is, they accomplish steps toward sentence formation. Certain rules can be executed only after other rules have been executed. A given sequence of rules either gets “blocked” during execution (i.e., at one point a rule cannot be executed) or leads to a syntactically correct—but perhaps meaningless—sentence. Both the blocking and successful sentence formation sequence are executed in a finite amount of time; furthermore, they are exclusive of each other; a sequence is either blocked or executed to the end. Such a grammar is called a “generative” grammar, since it generates sentences from the rules.

One can visualize such a grammar as a machine which, if one pushes a button, produces all syntactically correct sentences of the language, provided that we let the machine function until eternity.

The decidability problem can now at least be formulated. It appears as an augmented inverse of the above described generative procedure. Given a string, plus the grammar of the language, can one determine, within a determinable time (this restriction "augments" the inverse), whether a rule sequence exists generating that string? If yes, the string is a sentence; otherwise it is an arbitrary string.

In fact, linguists require more from a generative grammar than merely generating sentences. The grammar must produce, along with each sentence, a structural description of the sentence, exhibiting the syntactic relations among the sentence components. If the sentence is ambiguous, having several meanings, a structural description must be produced for each meaning. The above-outlined decidability procedure can then be extended. In addition to a yes/no answer concerning syntactic correctness, one can require that the procedure produce the structural description of the sentence, in case of a "yes" answer. If, in addition to time, the amount of space (computer memory, paper pads, and so forth) required by the procedure can also be computed in advance, the decidability procedure is called a recognition procedure. A recognition procedure either rejects the input string as unsyntactic or produces a structural description of it within computable time and space limits. These limits may depend, of course, on the length and complexity of the sentence.

It is obvious that there must be a connection between generative grammar and vocabulary or else the grammar could not generate sentences. These connections are grammar rules, like all the other rules. A rule may say, for example, that "boy" is a "noun." Another rule, not involving the vocabulary, may state that a "noun" is a "noun phrase" (not excluding, of course, other kinds of noun phrases). It appears thus that the grammar operates on two kinds of words: (1) the words of the vocabulary, or the "terminal" words; (2) words, like "noun," which finally do not enter the sentences formed but help to generate and describe the sentence; these words are called "nonterminal" or sometimes "metalinguistic descriptors."

The format of the grammar rules and of the structural descriptions determines the grammar class. The grammar class, or grammar type, is a basic metalinguistic notion. Languages can have more than one grammar in the same or different grammar classes.

A language is thus a decidable subset of the set of all strings "over" a given vocabulary. Furthermore, it must be the output of a generative grammar. This is the metalinguistic definition of language.

### *Hierarchy of Grammar Types*

The most important formal grammar types belong to an overall general class called "restricted rewriting systems." With some simplifications, a restricted rewriting system can be defined as a set of four elements:

1. The terminal vocabulary.
2. The nonterminal vocabulary.

3. A unique, distinguished element (or “axiom”) of the nonterminal vocabulary, usually denoted by  $S$  (abbreviating “sentence” or “statement”).
4. A set of rules or “productions.”

The general rule format in a restricted rewriting system is of the form

$$uAv \rightarrow w$$

where  $u$ ,  $v$ , and  $w$  are strings composed of elements of both the terminal and nonterminal vocabulary and  $A$  is an element of the nonterminal vocabulary. This kind of metalinguistics is an adaptation, for linguistic purposes, of procedures used in metamathematics to deduce theorems from axioms.

Further restrictions on the rule format establish the grammar classes belonging to the overall type of restricted rewriting system:

1. If there is at least one rule in a grammar in which either  $u$  or  $v$  or both are not omitted, the grammar is “context-sensitive.”
2. If in all rules  $u$  and  $v$  are omitted, the grammar is “context-free.”
3. If in all rules,  $w$  is of the form  $sX$  (in all rules) or  $Xs$  (in all rules) or  $s$ , where  $s$  is a string composed of terminal words and  $X$  is an element of the nonterminal vocabulary, then the grammar is “finite state.”

Sentence generation takes place in the following manner:

If the grammar is not “blocking,” there must be at least one rule of the form  $S \rightarrow w$ . This is the starting point of the generation from the axiom (or “initial symbol”)  $S$ .  $S$  is then replaced, or “rewritten,” as  $w$ . (The arrow in the rules should be read in this way.) Next, the nonterminals occurring in  $w$  are rewritten in an arbitrary order. In other words, segments of  $w$ , which satisfy the definition  $uAv$  in the left side of a rule, are replaced by  $uvw$ , where  $w$  is the right side of the newly applied rule. In the context-free, or in the finite state, case, there is no  $u$  or  $v$ ; these strings are “empty.” This procedure is continued until all elements of the current string (after successive rewritings) are terminal. This string is then a generated sentence.

The structural description, generated by the restricted rewriting system along with each sentence, is a tree. The top node of the tree is labeled by the initial symbol  $S$ . At each rule execution the elements of string  $w$  are appended as “successors,” in the order of their appearance in  $w$ , to the node labeled  $A$ , where  $A$  occupies—at that moment during the generation—a terminal position in the already generated part of the tree. At the end of the generation, words of the sentence label the terminal nodes of the tree. The other nodes are labeled by nonterminal vocabulary elements.

Restricted rewriting systems, especially the extremely popular context-free variant, are the formalized equivalent of the linguists’ immediate constituent analysis. A great number of grammar types, proposed by workers during the past decade and using different kinds of rule formats, turned out to be essentially equivalent to context-free grammars. Among these other systems we find the already mentioned Backus Normal Form, dependency, categorical, and some other kinds of grammars [5]. Finite state

grammars generate the languages used by logicians for the definition of abstract models of computing automata.

The three main types of restricted rewriting systems (finite state, context-free, context sensitive) form a hierarchy in the following sense: Starting with finite state, each type is a particular case of the next type, in the order just stated. The order thus established, based on rule format, also embodies an increasing order of generative power. This means that a higher level class (e.g., context-free is higher level than finite state) is inherently capable of generating more complex languages than the lower level type. This phenomenon can be best understood through recursivity. The rule  $uAv \rightarrow w$  is recursive if  $w$  contains  $A$ . In such a case the rule which rewrites  $A$  can be repeatedly executed until  $A$  disappears through some other rule application. Both finite state and context-free grammars can contain recursive rules, but only context-free grammars can "self-embed" the symbol  $A$ , since in finite state rules either the left side or the right side of  $A$  must be empty in  $w$ . As a consequence, context-free grammars are able to generate strings of the form  $aaaa \dots axbbb \dots b$ , where the number of  $a$ 's is equal to the number of  $b$ 's. Finite state grammars cannot guarantee that. The phenomenon is extremely important from a practical viewpoint (e.g., parentheses count in arithmetic expressions in programming languages). The grammar hierarchy is thus ordered according to increasing generative power.

Another important viewpoint is descriptive adequacy: Conceptually, how good are the generated structural descriptions for interpretative purposes? According to this viewpoint, too, higher level grammar class means higher descriptive adequacy. For example, finite state grammars generate binary (two-branching) trees, while context-free grammars are able to generate  $n$ -branching trees, where  $n$  is an arbitrary number. A language which could be generated by a finite state grammar could be given a context-free grammar, by the language designer or by the linguist, if higher descriptive adequacy of the generated trees is desired.

The hierarchy could be extended, beyond context-sensitive grammars, toward unrestricted rewriting systems wherein the rule format would be further loosened. A crucial metalinguistic problem in connection with rewriting systems is decidability. As of today this is an open question; all we know is that restricted rewriting systems are decidable and that a completely unrestricted rewriting system is not decidable; i.e., one cannot, in principle, impose time limitations on a recognition procedure.

At the lower end of the hierarchy, one could introduce finite languages beyond the finite state class. Such languages consist of a finite list of sentences, or sentence "frames," in which words can be inserted. Quite often, in practice, a finite language must be given a higher level grammar, because the list is much too long to grasp it for any purpose.

Another important type of formal grammars is the transformational class. These grammars operate on trees produced by severely restricted context-free grammars; the transformations execute tree processing operations under certain linguistic conditions. It is established that their generative power is at least equal to the power supplied by context-sensitive grammars and that they must be, in principle, decidable. Transformational grammars have been proposed by Chomsky [6] for the generation of natural language, but they may also penetrate the field of artificial languages.

### *Hierarchy of Computing Automata*

It was mentioned earlier that logicians use finite state artificial languages for the description of computing automata or, more precisely, of their abstract mathematical models. Let us, first, elaborate a little on this concept.

From an abstract viewpoint the input to a computing automaton is an "acceptable" string of symbols in the automaton's alphabet (i.e., the set of characters known to the automaton). The term "acceptable" means that the automaton knows what to do with the input string in some definable way. Since all such input strings are characterized by the attribute "acceptable," the set of all acceptable input strings forms a language. The automaton is then the equivalent of a decidability procedure—in the sense defined above—with varying degrees of sophistication: (1) the automaton merely replies by stating that the input string is or is not acceptable; (2) in the case of acceptance, the automaton produces a structural description of the input string; (3) definite, computable time and space limitations are respected by the automaton during the process. In the last case the automaton is sometimes called a recognition automaton. All these capabilities imply that the automaton has internalized the generative grammar of its input language; it is, in a way and partially, equivalent to this generative grammar. With these concepts the equivalence and connection between generative grammars and automata are established.

The most general computing automaton is called a Turing machine (after Turing, the British mathematician who first established such a model in the 1930s). The mentioned finite state automata have been defined by logicians to represent more realistically the behavior of a computer in a concrete situation. The input language of a finite state automaton is a finite state language, one of the members of the restricted rewriting system hierarchy. Other automata have been defined, and all these models form a hierarchy, from Turing machines "down," restricting more and more the computational power of the models. One of the most remarkable results of modern mathematical linguistics is the establishment of the equivalence of the two hierarchies: the one of rewriting systems and the one of computing automata. In Table 1 we show, for reference, the parallelism between the two hierarchies by merely giving the names of the equivalent automata types. The enumerated automata can act through suitable specifications as canonical recognition procedures for the corresponding grammar types, with the exception of Turing machines, which represent inherently undecidable languages.

TABLE I

Rewriting systems	Automata
Finite sentence list	Table look-up machine
Finite state	Finite state automaton
Context-free	Push-down store
Context-sensitive	Linear-bound automaton
Unrestricted rewriting system	Turing machine

## Computer Processing Aspects of Linguistics

### *Machine as a Language Processor*

An automaton must internalize the generative grammar of its input language; otherwise it could not perform recognition of an input string. The generative grammar is a mere set of rules with which a sentence generative process can be canonically associated. It is thus clear that the automaton must contain something more as part of its competence. This additional knowledge is the language processing capability of the automaton, which actually transcends a mere syntactic recognition capability in practical situations. The automaton must be able to interpret the structural descriptions to execute the instructions contained in the input text.

In its broad outline the situation is similar to human language usage. A human internalizes the rules of a natural language either by learning a foreign language or through a mysterious language acquisition process during early childhood for his mother tongue. His linguistic competence is then put into action in various linguistic performance situations, such as speaking, listening, understanding, translating, and formulating thoughts. The rules of performance are quite different from the internalized rules of competence. This performance/competence duality is a well-known psycholinguistic situation. A famous example will explain the difference.

Consider the language of ordinary multiplication, i.e., the arithmetic multiplication rules we all internalized in elementary school. With these rules any multiplication can be performed irrespective of the number of digits in the two factors. When we are faced with the problem of performing the multiplication mentally, in the case of two relatively long numbers, our performance is very limited. We need "auxiliary memory," i.e., a piece of paper, to remember partial results. The multiplication rules need not be changed or augmented; all we need is memory. The competence is there, but performance is restricted because of lack of space. Similarly an extremely long and complex English sentence (e.g., 1000 words and considerable embedding of relative clauses in each other) can be syntactically correct as formed through the rules of the generative grammars; nevertheless, a reader may have difficulties in figuring out its meaning. He would have to do what he does in the case of multiplication: take pencil and paper and establish partial structures. Again, the linguistic competence is in his mind, but his performance is limited.

Computing machines have, of course, severe performance requirements. The analogy between human and machine language perusal breaks down, however, in the semantic area.

A human language user employs, quasi-unconsciously, his entire factual and extralinguistic knowledge in every performance situation. Data accumulated in the past, his awareness of the context of the situation, contribute heavily to meaning penetration. One could say that natural language is built in a way which takes this situation into account. Language can be and is ambiguous but a human is able to disambiguate discourse. Discourse may contain references to other parts of the discourse. These references may vary from such down-to-earth devices as pronouns to

sophisticated, hidden allusions. The human user of natural language is able to grasp these references and substitute their referents.

Machines cannot process natural language because their built-in semantic system is radically different from a human's. Even the simplest pronoun may represent insurmountable difficulties. Therefore artificial languages for instructing computers must incorporate a semantic system which is compatible with the computer's understanding of things. This is the reason English must be severely restricted when used for instructing computers.

Thus language processing by machines has two main aspects: (1) syntactic recognition, as a preliminary for meaning interpretation; (2) automatic meaning interpretation.

### *Problems of Syntactic Recognition*

The recognition problem of syntactic structures is, by and large, solved for finite state and context-free languages with reasonable efficiency. The question is whether most artificial languages can be generated by context-free grammars. As it stands, the question is not well formed. In any language a context-free grammar can generate the constituent structure of the language. The question is how much of the structure is disregarded, which is nevertheless essential for meaning interpretation. For example, even conceptually very simple instructions of the programming language FORTRAN [7, 8] cannot be parsed by uniquely context-free means. Consider the specification

FORMAT 3Habc . . .

This command means that exactly three characters following the letter H (abbreviation for "Hollerith character") must be printed. The overall structure of this text can be described by context-free means: the word "FORMAT," followed by an integer, followed by the letter "H," followed by a character sequence. But a context-free grammar is incapable of actually parsing the statement, because it has no power to interpret the number before "H"; consequently the recognition procedure is unable to delimit and separate the sequence "abc" from possible other characters following them. To solve this problem, a counter must be incorporated in the recognizer, which interprets the number. Simple counters can be, in principle, viewed as push-down stores, but this push-down store would be an additional one, since the overall context-free recognizer already operates through a push-down store. Such a combination of two push-down stores is, in principle, not context-free.

This example shows that even such a simple concept as the value of a number, to be established from the digit string which makes up the number, is beyond context-free power. There are many other examples of context-free limitations. The overall syntactic recognition of even the simplest artificial languages is far from being a solved problem by metalinguistic means. In such situations language processors revert to traditional means; i.e., they utilize *ad hoc*, language-dependent procedures.

These considerations terminate the investigation and description of the notion of language as it is understood by modern theoretical linguistics. An artificial language must satisfy the definitions and criteria emerging from metalinguistics; furthermore,

from a pragmatic viewpoint, it must truly place itself in the opposing position with respect to natural languages. This means that the birth and life of an artificial language is planned and controlled through the deliberate decisions of language designers. The following main section will shed light on the definition and usage of a particular class of artificial languages through the discussion of the most relevant language group from the viewpoint of this encyclopedia, namely, the "higher level" or "programming" languages.

## PROGRAMMING LANGUAGES (= ARTIFICIAL LANGUAGES FOR COMPUTERS)<sup>1</sup>

Unfortunately there does not appear to be any universally accepted definition of a programming language, and it is therefore easier to provide characteristics rather than a specific definition. It is, of course, taken for granted that a programming language is some set of characters and rules for combining them by which the user can communicate with the computer to cause useful work to be done; this communication takes place through another program which is normally called a compiler, whose purpose it is to translate the user's program (called the source program) into machine code (called the object program) which can then be executed by the computer. This contrasts with the use of a language which is the same as, or very similar to, the direct language used by the computer, namely, an assembly language. Some people include assembly languages within the broad category of programming languages and use the term "higher level language" to exclude assembly languages. This article specifically discusses *only* higher level languages, and the phrase programming languages does *not* include assembly languages.

This section attempts to characterize programming languages, point out their advantages and disadvantages, provide some classifications of programming languages with proposed definitions, and discuss the major issues in programming languages. The last are subdivided into nontechnical and technical characteristics of programming languages. The next subsection discusses major classes of existing languages. The impossibility of giving more than a surface view of this field is evidenced by the existence of a book on the subject [9].

### Fundamental Concepts of Programming Languages

#### *Defining Characteristics of Programming Languages*

A programming language is a set of characters and rules for combining them which has the following four characteristics:

<sup>1</sup> The material in this section was basically published in articles by J. E. Sammet, "Fundamental Concepts of Programming Languages" and "Programming Languages: Current and Future Trends,"

1. It requires no knowledge of machine code by the user. In other words, the user need learn only the particular programming language and can use this quite independently of his (perhaps nonexistent) knowledge of any particular machine code. This does not mean that the user can completely ignore the actual computer. For example, he may wish to take advantage of certain machine facilities which are known to him and which provide more efficient programs, and in particular he obviously cannot use input/output equipment which does not exist on a particular computer configuration. However, the fundamental point is that he does not need to know the basic machine code for the given computer.

2. A programming language must have some significant amount of machine independence. This means that there must be some reasonable potential of having a source program run on two computers with different machine codes without completely rewriting the source program.

3. When a source program is translated to machine language, there is normally more than one machine instruction per executable unit created. For example, an executable unit in a programming language might be something of the form "A - B + C\*D" or "MOVE A TO B." Normally each of these executable units would be translated into more than one machine instruction.

4. A programming language normally employs a notation which is somewhat closer to the specific problem being solved than is normal machine code. Thus, for instance, the first example cited above might be translated into a sequence of instructions such as:

CLA	C
MPY	D
ADD	B
STO	A

which is clearly less understandable than the programming language form.

### *Advantages and Disadvantages of Programming Languages*

As with any item one cannot obtain something for nothing, and therefore there are both advantages and disadvantages to programming languages, where the alternative is some type of assembly language.

Let us consider the advantages first:

1. The primary advantage to a programming language is that it is easier to learn than a machine language. It must be emphasized that there is a relative aspect involved in this advantage. An extremely powerful programming language might be harder to learn than an assembly language on a computer with only a dozen instructions. However, given programming and assembly languages of approximately the same complexity in their relative classes, the programming language will be easier to learn.

This actually has two facets to it. The programming language may itself be extremely complex, but its ease of learning often comes because the notation is somewhat more related to the problem usage than is the machine code; furthermore, more attention can be paid to the language itself rather than to the idiosyncrasies of the physical hardware which are necessary when one deals in machine code.

2. A problem written in a programming language is generally easier to debug for two major reasons. First, there is actually less material which needs to be written because of the explosion factor indicated as the third characteristic of a programming language. Thus, in comparison with a program written in assembly language, the source program will be physically shorter. Since the number of errors is roughly proportional to the length of the program, obviously there will be fewer errors. A second reason for the program being easier to debug is that the notation itself is somewhat more natural and therefore more attention can be paid to the logic of the program with less worrying about details of the machine code.

3. A program coded in a programming language is generally easier to understand and transfer to someone other than the originator because of the notational advantages and relative conciseness already mentioned.

4. A fourth advantage accrues from the fact that the notation of a programming language automatically provides certain documentation because the notation is easier to understand and the logic is easier to follow.

5. Finally, the above advantages tend to accumulate into one general advantage which is that the total calendar time required for the problem solution is generally reduced significantly.

The disadvantages are as follows:

1. The primary disadvantage is that the advantages do not always exist in specific cases, and the person might be worse off than with a very simple assembly language as was indicated under the first advantage. Thus the programming language might be extremely difficult to learn, and unless proper attention is paid to the compiler and other facets of the overall system, the other advantages may not themselves accrue. Fortunately this seldom occurs.

2. A more specific disadvantage is that the additional process of compilation obviously requires machine time, and this may require more than the machine time saved from easier debugging.

3. The compiler might produce very inefficient object code. This would significantly affect production runs, i.e., programs which are run repeatedly and whose machine time requirements are increased significantly by any inefficiencies. (The counter argument to this, of course, is the fact that compilers nowadays generally produce code that is at least as good as the average programmer and there are only a limited number of really expert programmers who can write the most efficient machine code.)

4. The last disadvantage is that the program may be much harder to debug than an assembly language program if the user does not know machine code and the compiler does not provide the proper type of diagnostics and debugging tools. A user who must look at a memory dump in octal, which he does not understand, is going to have more

trouble than debugging an assembly program in which he understands what is happening.

### *Classifications of Programming Languages and Proposed Definitions*

As indicated earlier, it is very difficult to define a programming language. However, it is a little bit easier to propose definitions for classes of programming languages. The terms to be defined are the following: procedure oriented, nonprocedural, problem oriented, application oriented, special purpose, problem defining, problem describing, and problem solving. Note that some of these are overlapping and that a particular language may fall into more than one of these categories.

A procedure-oriented language is one in which the user specifies a set of executable operations which are to be performed in sequence; the key factor here is that these are definitely executable operations, and the sequencing is already specified by the user. The term "nonprocedural" has been bandied about for years without any attempt to define it. It seems clear now that a definition is not really possible, because "nonprocedural" is really a relative term in which decreasing numbers of specific steps need to be provided by the user as the state-of-the-art improves. Thus, before such languages as FORTRAN, the statement  $Y = A + B*C - F/G$  could be considered nonprocedural because it could not be written as one executable unit and translated by any system. In August 1972 the sentences "calculate the square root of the prime numbers from 7 to 91 and print in two columns" and "print all the salary checks" are nonprocedural because there is no compiler available that can accept these statements and translate them; the user must supply the specific steps required. As compilers are developed to cope with increasingly complex sentences, then the nature of the term changes. Thus what is considered nonprocedural today is definitely procedural tomorrow. One of the best examples of a currently available nonprocedural system is a report generator in which the individual supplies essentially the input and the output without any specific indication as to the procedures needed.

The term "problem oriented" has been used in many ways by different people, but it seems that the most effective use of this term is to encompass any language which is easier for solving a particular problem than machine code would be. Any current programming language illustrates this, and thus the term "problem oriented" is a kind of catchall. The term "specialized application" seems to apply best to a language which has facilities and/or notation which are useful primarily for a single specialized application area. The best illustrations of this are such things as APT for machine tool control and COGO for civil engineering applications. Notice that both of these are of course problem-oriented languages. On the other hand, FORTRAN and COBOL are problem oriented but for wider applications areas than APT or COGO. Here again, the term is somewhat relative because FORTRAN is suitable for applications involving numerical mathematics, whereas COBOL is obviously suited for business data processing and the overlap between these is relatively small. The wider the application area, the more general the language must be.

A "special purpose" language is one which is designed to satisfy a single objective. The objective might involve the application area or the ease of use for a particular application or might pertain to efficiency of the compiler or the object code.

A problem-defining language is one which literally defines the problem and may particularly define the desired input and output, but does *not* define the method of solution. Here, the best illustration is a report generator. A somewhat secondary example is sorting routines, except that the input to sort routines usually is not in the form of a language *per se*.

A much more general type of language classification is that referred to as "problem describing" in which the objective is described only in very general terms, e.g., "Calculate Payroll." All this does is describe in the most general way the problem which is to be solved; it gives no indication whatsoever as to how to solve it. We are an extremely long way from this.

Finally, a problem-solving language is one which specifies a complete solution to a problem. Like the term "nonprocedural," this is a relative term which changes as the state-of-the-art changes.

### Major Issues in Programming Languages

#### *Nontechnical (= Functional) Characteristics*

Most articles and talks about programming languages tend to throw in a great many concepts and characteristics without any very careful delineation of their relationships to one another. Thus when people try to decide which language to select for a given task, they often run into difficulty because of a lack of clear-cut factors against which to make their evaluation. In particular there is usually insufficient attention paid to the difference between nontechnical (which can also be called functional) and specific technical characteristics. While it is *not* the purpose of this section to tell readers how to select a particular programming language, it is hoped that some of the fundamental background material contained herein might be useful in such a situation. The remainder of this section describes the nontechnical characteristics, while the next section discusses the technical characteristics.

**Purpose.** There are three different types of purposes that must be determined in defining any programming language. The first and most important is the particular application area for which the language is designed. Thus it must be determined whether this is to handle such things as numerical scientific work, graphic displays, business data processing, or engineering design. The second purpose is to specify the type of language, where in this context a number of different things are meant. First, into which one or more of the classifications previously cited does the language fall? Second, what is the form of the language relative to succinct and/or formal notation versus naturalness? Is the language meant for direct input to a computer or primarily as a publication or reference language? Last, there must be specified the type of user the language is intended to assist. Generally speaking, programming languages tend to aid the application programmer who is not necessarily a full time professional programmer, or, in other words, most programming languages tend to help the individual who has a specific problem to solve rather than somebody whose major interest is in computer programming.

**Conversion and Compatibility.** There has probably been more talk and discussion about conversion and compatibility than almost any other aspect of programming languages. It would be impossible to discuss these topics in any great detail here, so the main thing is to point out some of the varying types of compatibility that exist. The first is compatibility across individual machines, i.e., how machine independent is a particular language? The second type of compatibility relates to the compiler; unfortunately, the same language is not always implemented the same way even for the same machine, and there exist cases in which compilers for the same language, on the same machine, do not produce the same results. A third type of compatibility is tied in with the problem of dialects, where major and minor changes and/or subsets and extensions are all made in the name of the same language.

With regard to ease of conversion, the first and easiest way of converting is based on compatibility, which hopefully exists. Two other ways are direct translation into another language and semitranslation whereby the user does some of the work.

**Standardization.** The purposes of standardization are fairly well known and do not need to be discussed in any detail. Basically, standardization of programming languages eases conversion problems, permits compatibility across machines, eases the training problem, and generally reduces economic costs from many points of view. The problems in standardization are a little less obvious. They actually fall into three classes: conceptual, technical, and procedural.

In the conceptual category falls the major difficulty of determining when it is time to establish a standard and how one establishes a standard and at the same time permits new developments to proceed. This type of conceptual problem is not unique to programming languages and exists in many fields.

The technical problems in standardizing programming languages are enormous, and it is indicative of the difficulty that in the first 5 years of existence of the sub-committee on standardizing programming languages within the American National Standards Institute (ANSI),<sup>2</sup> only one language (FORTRAN) was standardized. That was by far the best known and most widely available language but it still took a long time to develop the standard. By 1972 the only other language that had officially been promulgated as a standard was COBOL.

In essence it is very difficult to write down the definition of a language in unambiguous terms.

The third problem in standardization is the method of establishing standards and the procedures associated therewith. These are long and laborious but with the advantage of requiring a consensus so that when a standard actually does come into existence it is one which will really be obeyed, even though compliance with ANSI standards is completely voluntary.

**Types and Methods of Language Definition.** When one thinks of language definition, one almost invariably thinks of various technical ways in which languages can be defined. However, although it is often overlooked, the administrative framework within which a language is defined tends to have a significant effect on the language.

<sup>2</sup> ANSI was formerly known as USASI and prior to that as ASA.

For example, the question of who designed the language and in what organizational framework is of paramount importance. Unfortunately, language design is still an art and not a science, and different people often have very strong personal views which cannot be defended on any grounds other than "I (don't) like it." The organizational framework is also of major importance, because a language which is designed by an intercompany group will have many more compromises than that designed by a single organizational unit. However, those who oppose language design by committee are putting their heads in the sand, because there is no major fully implemented language which was actually fully designed by a single person. The moment that several people get involved, there is in fact a committee, even though it may be a single organizational unit with a person designated as being in charge. Thus the mere fact that a language is designed by an intercompany committee is not bad in and of itself. The more significant problem in a situation like that is that the people usually have other job assignments and thus their time is limited. Furthermore, the interconnection between the language designers, the implementors, and those who maintain the language (i.e., settle tricky points of interpretation and perhaps try to extend it) is a crucial part of the administrative framework. If the people who implement the language are not the ones who designed it, or conversely, there is always trouble. Similarly, if the people who maintain the language are not those who designed it, then there is difficulty arising from the fact that the newcomers always have (or think they have) better ways of doing things. Sometimes the language is redesigned each time a new group of people becomes involved and chaos results. The interconnection between the language maintainers and those who do the implementation plays a key role in the orderly development—or lack thereof—of a language.

There are a number of technical problems associated with language definition. First and foremost is that we do not yet know how to give a completely formal definition of a programming language. The three components of a language definition are (1) syntax which specifies the legal strings of characters (e.g., A + B, not + AB), (2) semantics which specifies the meaning (e.g., add, not subtract), and (3) pragmatics which specifies the interaction between the user and the system. We have fairly good ways of defining the syntax, and these can be used in a practical way in some instances. There are ways of formally defining the semantics but they are difficult to use in practice [10]. We have made no progress on the pragmatics except to recognize that it is a problem.

The problems posed in documenting language definitions are almost as severe as the technical definition problem itself, as can be seen in Ref. 11.

**Evaluation Based on Use.** Very often languages are evaluated on the basis of manuals and speculation and offhand opinion rather than on the basis of use. If somebody wishes to evaluate a language manual, that is all right, but he should not evaluate the language itself until it has undergone some usage. Part of what is involved in usage involves training, learning, writing programs, debugging them, and so forth, and each is fair game for objective evaluation. However, one must be very careful to distinguish between evaluating the language and evaluating the compilers which translate it. The language may be good and the compilers very bad; in cases like this it is almost always

the language which is blamed, although it simply may be a matter of ineffective compiler writing.

### *Technical Characteristics<sup>3</sup>*

The technical characteristics of a language are simply the sum total of what actually constitutes the language. When most people think of programming languages, they think only of the technical characteristics, but as has been shown above, this is too narrow a view. The main technical characteristics of a language are the form of the language, the structure of the program, the data types and units, the executable statement types, the declarations and non-executable statements, and the structure of the language and the compiler interaction.

**Form of Language.** The form of the language involves many facets. The first is the actual character set which is used, and the second is the ways in which the characters are combined to form names (e.g., TEMP), operators (PLUS, AND), executable commands (COMPUTE, READ), and nonexecutable declarations (DIMENSION, INTEGER). The third facet of the language form is the specific rules for forming identifiers and words. The rules for identifiers involve such things as the formation of data names and statement labels, the handling of subscripts and qualification, the use of reserved words, and the definition of literals.

The methods of combining words involve the significance of blanks, the use and meaning of punctuation, the presence or absence of noise words, and ways of combining operands and operators. Finally, the form of language obviously involves the physical input which is used, as well as the conceptual form (e.g., fixed format versus a string of characters, English-like versus highly symbolic). Although a language is really independent of the input medium, the language tends to be defined as a partially fixed format when punch cards are meant to be the prime input source. Similarly, if paper tape or direct typewriter input is involved, the language is usually designed as a string of characters.

**Structure of Program.** The structure of the program is based on the types of subunits that are permitted, their characteristics, and the ways of intermingling them. The types of subunits that are normally allowed are declarations (including data descriptions); executable units; loops; functions, subroutines, and procedures; the complete program; and the method of interacting with the operating system and the environment. The characteristics of subunits involve the methods of delimiting them, whether they are recursive and/or reentrant, the types of parameter passage required, what types of executable statements and declarations can be combined, and what other languages, including possible machine language, can be included.

**Data Types and Units.** There are a number of different types of data variables,

<sup>3</sup> Some of the terms used here may not be self-explanatory, and lack of space prevents adequate definition. It is hoped that the reader will get the general concepts even if a few specific words are not understood.

including the following: arithmetic, Boolean, alphanumeric, formal (= algebraic), strings, lists, vectors and arrays, hierachal. A key characteristic is what types of data units can be accessed by the commands in the language. Thus one needs to know what data units of the hardware, as well as what variable types, can be obtained through the use of a particular command. There are many different types of arithmetic which can be performed, e.g., fixed and floating point, rational arithmetic, multiple precision, and complex number. Along with all these different types of arithmetic and different types of data go various rules on modes such as rules on which data types can be combined, and conversion and precision rules for computation.

**Executable Statement Types.** Many people feel that a programming language is completely determined by the executable statements it contains. Certainly the executable commands play a very significant role, but they are not the only parts of a language. It is hoped that this rather narrow view will be dispelled as a result of preceding and following comments.

The major classes of executable statements are assignment statements, those for handling alphanumeric data, and those for sequence control and decision making. Almost all programming languages have all these in some form or other. An increasing number of languages include executable statements for handling various types of symbolic data, e.g., algebraic expression manipulation statements, and statements to handle lists, strings, and/or patterns. Finally, with the increasing complexity of hardware and operating systems, the programming language itself must include a number of statements to provide interaction among these. Specifically there are apt to be statements dealing with input/output, library references, debugging, storage allocation and segmentation, and finally statements which deal directly with the operating system and/or machine features.

**Declarations and Nonexecutable Statements.** Although the executable statements are obviously essential to accomplish anything, in general they cannot operate without knowing something about the data on which they are to act. For this reason most programming languages have either complicated or primitive declarations to cover such things as data and/or file descriptions, format descriptions for input/output data, and declarations about storage allocation.

**Structure of Language and Compiler Interaction.** Among the characteristics of programming languages are the abilities for self-modification of programs or self-extension of the language. Many languages contain directives to provide the compiler with useful information. Consideration must be given to the effect of the language design on compiler efficiency and to the inclusion of debugging aids and error checking. A key characteristic of more theoretical than practical interest is whether the language can be used to write a compiler for the language. Finally, an interesting characteristic of the language is whether or not it is useful outside of its primary application area.

### Major Classes of Existing Programming Languages

One of the most useful classifications of programming languages—particularly for people with specific problems to solve—is by application area, where in this context the phrase “application area” is used in its widest possible connotation. This section describes the most significant application areas, with illustrations of the major languages in each. Each language is described in Ref. 9, which also contains many more references. Complete lists of programming languages in the United States for 1971 and 1972 are shown in Refs. 12 and 13.

#### *Numerical Scientific*

The first application area from an historical, and probably even from a total usage, point of view is the general area of numerical scientific languages. The most widely used language in this area, is FORTRAN.

FORTRAN was initially developed by IBM as early as 1954. It took a while before the advantages of programming languages (as discussed earlier) were clear enough to cause the widespread use of the language. FORTRAN provided convenient facilities for solving numerical scientific problems, and a large library of subroutines was developed through cooperative user organizations such as SHARE. Through an evolutionary (but not always compatibility retaining) process, improvements were made in FORTRAN, culminating in FORTRAN IV. Virtually every computer manufacturer who wished to sell to the scientific market implemented their own version of FORTRAN. Thus FORTRAN became a *de facto* standard, although naturally there were problems in transferring FORTRAN programs from one computer to another, and so often some minor (or major) modifications had to be made. However, the significance and widespread use of FORTRAN made it the first language to become an official ANSI standard. Actually two standards were developed, corresponding roughly to what were FORTRAN II and FORTRAN IV; they attempted to reflect the consensus of FORTRAN usage, and the smaller is a proper subset of the larger. The official standards are given in Refs. 7 and 8, and numerous books have been written describing FORTRAN.

Within the general area of numerical scientific problems, FORTRAN is not the only major language. ALGOL 60, which was developed by an international committee composed of members from professional societies, plays a more significant role in Europe than in the United States. ALGOL differs significantly from FORTRAN in that the former is considered primarily a means of communicating algorithms and descriptions of computational processes, whereas the latter is designed for direct input into a computer. In particular, ALGOL contained no official input/output procedures until the development of the proposed ISO standard. Part of the reason for ALGOL’s difficulty in the United States is the way in which the first official report [2] was written; it gave joy to those interested in language design and methods of rigorous definition, but discouraged numerous people from even attempting to understand the language. ALGOL’s greatest impact has been through its indirect requirement for improved implementation techniques and most importantly through the Algorithm Section of the ACM *Communications*. Over 400 algorithms have appeared there, and a great many

of these have been certified by people who ran them on computers. (It is interesting to note that in many cases the people wishing to test them had no ALGOL compilers available, so they rewrote the algorithm in FORTRAN and tested it that way.) Many of these algorithms deal with more than just numerical problems, e.g., sorting and scanning. In spite of the much greater use of FORTRAN than ALGOL, it was not until 1966 that the Algorithm Section of the ACM *Communications* permitted algorithms to be submitted in FORTRAN.

The advent of time-sharing provided impetus for the development of languages designed to be used in such an environment. The first of these was JOSS which spawned numerous dialects. Then BASIC [14] was developed to be very easy to use and has become extremely popular. The efficient implementation of APL\360 provided a very powerful language, although its notation is very strange at first glance [15]. It has actually been used successfully in applications other than purely numerical calculations.

Several systems [16, 17] have been developed to try to overcome the difficulty of having to linearize mathematical notation to keypunch it. In each case special equipment was designed to permit—among other things—true subscripts and superscripts, which are the biggest handicap in reading programs written in FORTRAN or similar languages. These systems also attempted to provide a reasonably normal way of interspersing English words and phraseology in the midst of the mathematical expressions.

### ***Business Data Processing***

The second major area of current activity has been the general field of business data processing. Here, the only significant language is COBOL, whose initial specifications were developed by an intercompany committee under Department of Defense sponsorship in 1959. Work on extending the language has continued ever since. COBOL is an attempt to provide an English-like language suitable for expressing the solution to business data-processing problems. As such it provides elaborate mechanisms for defining files and individual record items and for handling input and output. Efficient procedures for dealing with mass storage devices and for handling tables were added several years after the original specifications, as were such facilities as sorting and a report generator.

COBOL is unique in that it was the first attempt to design a language which would handle complex files and still be reasonably machine independent. A number of installations have made successful use of this compatibility (which, of course, cannot be 100%) to minimize or lower their conversion costs.

Since the Government Services Administration (GSA) took the position that any manufacturer wishing to supply computers had to have a COBOL compiler (unless it could be clearly demonstrated that it was not needed, as in scientific installations), virtually every manufacturer has implemented the language. A standard was adopted in 1968 [18] and the Conference on Data Systems Languages (CODASYL) Programming Language Committee has also issued successive sets of specifications, for example Ref. 19.

### *List and String Processing*

In certain classes of problems (ranging from theorem proving to inventory control), it is necessary to deal with data whose size and amount varies constantly. To cope with that problem, the technique of "list processing" was developed as early as 1955. This technique involves storing with each piece of data a "pointer" to the next logical piece of data, which is generally *not* in the next sequential memory position. The first major list-processing languages were IPL-V and LISP. IPL-V provided a wide number of instructions for manipulating lists, essentially at the assembly language level. LISP was developed at M.I.T. around 1959 and introduced certain mathematical concepts and ways of expressing computational processes that are particularly suitable for problems in artificial intelligence and symbol manipulation.

Although both languages have been implemented on many different machines, no attempts at official standardization have been made because much of the control has remained with the original developers. The definitive reference for IPL-V is Ref. 20, and the best single source of material on LISP is Ref. 21.

For many types of problems (e.g., language translation), there is a definite need to manipulate strings of characters and change them from one form to another. The first significant language to do this was COMIT, developed at M.I.T. in 1958. COMIT provided the ability to automatically search a string of characters for a particular pattern and transform this pattern and/or the string into an entirely different form. The same concepts, and much of the same notation, were carried over into SNOBOL, developed at Bell Laboratories in 1964, but as with anything, time had indicated ways to make significant improvements over the original concepts. However, although generally SNOBOL is more efficient than COMIT within their intended domain, there are still some specific tasks which can be performed better with COMIT.

No attempts at standardizing either language have been made, nor has this been necessary, since control started with, and has been retained by, the original developers. COMIT and SNOBOL are described in Refs. 22 and 23, respectively.

### *Formal Algebraic Manipulation*

A later development was the use of computers to do formal algebraic manipulation. Languages such as FORTRAN have the facility for computing the expression  $(A - B) \times (A + B)$ , providing that numerical values are assigned to  $A$  and  $B$ . However, there are a vast number of problems in which there is a need to allow the computer to do lengthy and straightforward algebra, like that done in high school or in elementary calculus, e.g., obtaining  $A^2 - B^2$  from  $(A - B) \times (A + B)$  without numerical values. FORMAC, as an extension of FORTRAN IV, added the facility for formal algebraic manipulation—including differentiation—to an existing language. It was developed at IBM in 1964 (see Ref. 24). The same principle—but with significant improvements—was used to develop PL/I-FORMAC [25]. Other languages exist, but FORMAC appears to be the one receiving the most use in 1972.

### *Multipurpose Languages*

As time progressed and applications became more complex, it was realized that many problems required the numerical scientific facilities provided in languages like FORTRAN and ALGOL, but simultaneously needed the data-handling provisions of languages like COBOL. Therefore it became natural to develop languages which would attempt to handle both application areas simultaneously. The first language developed in response to this need was JOVIAL, developed initially around 1960 by System Development Corp. and continuously improved and maintained by them. JOVIAL is essentially an outgrowth of ALGOL 58 but added many of the data-handling facilities of COBOL. It has been widely used in command and control problems. It has been implemented on a number of machines, but conversion problems have been hampered significantly by the multitudinous versions and subsets and dialects of the language which exist. Some of the documents describing JOVIAL are listed as Refs. 28 and 29.

This same need to have a single language to handle both scientific and data-processing problems was recognized by IBM and SHARE, which formed a joint committee in 1963 to deal with this problem. Although originally intended as an extension of FORTRAN IV, it became clear to the language designers that they could not provide the desired facilities and still maintain compatibility with FORTRAN. Hence, they developed a language, eventually named PL/I, which combined the best features from FORTRAN, ALGOL, and COBOL as well as some facilities from other languages. PL/I can be considered the culmination of the union of scientific and data-processing facilities with other features included (see Ref. 30). ALGOL 68 [26] had similar objectives and was developed under the auspices of International Federation of Information Processing (IFIP), but is not widely used.

### **Specialized Application Areas**

The use of languages in specialized application areas has become an extremely important one, and the languages used for such purposes are about half of all those in use. This subject is discussed in some detail in Ref. 27.

In essence, the application areas referred to above have all been quite broad, and the languages could be used for more than a very specific and limited type of problem. However, there are languages which have been developed to deal with particular problems arising from a very limited discipline or industry. Key examples of these are APT for machine tool control [31], and COGO and STRUDL for engineering [32, 33].

Simulation languages have been developed to handle the class of problems which require some simulation of a large general system prior to actual design, e.g., traffic control and factory layout. Key languages in this area are GPSS [34] and SIMSCRIPT [35].

### **Use of English**

This subject is sufficiently controversial, even in its definition, to warrant some mention. When used here it definitely includes the use of mathematical or scientific notation wherever desired. Two extreme positions can be noticed in the literature: (1)

Putting English at the computer user's disposal may or should be the next significant step in instructing computers. The further development of formal notation is of minor importance to the adherents of this philosophy. (2) Use of English is not within foreseeable technology for various reasons; furthermore its use is not even desirable, since natural language lacks rigor, clarity, and other features which are mandatory in a computer-instructing language. Therefore emphasis should be put on the further development of formal notation.

Between these two extremes, let us just enumerate some viewpoints in favor of English: There exists certainly a class of computer users to whom use of English would be of great help. These are the highly trained (but nonprogramming) professionals in differing fields. Implementation of programming in English would make the computer (as are other kinds of business machines) accessible to these persons on a "self-service" basis. It is true that use of English (or any other natural language) in its full richness is not within computer and programming technology as of today. The concept of "restricted English," however, may help to solve the more general problem. The success of such an enterprise is by no means incompatible with further interesting developments in the area of artificial, formal notation. Some further comments are given in Ref. 36.

## REFERENCES

1. R. Carnap, *Introduction to Symbolic Logic and Its Applications*, Dover, New York, 1958.
2. P. Naur, ed., Report on the algorithmic language ALGOL 60, *Comm. ACM* 3(5), 299–314 (1960).
3. N. Chomsky and G. Miller, in *Handbook of Mathematical Psychology*, Vol. 2 (R. Luce, R. Bush, and E. Galanter, eds.), Wiley, New York, 1963, Chaps. 11–13.
4. J. McCarthy, A basis for a mathematical theory of computation, in *Computer Programming and Formal Systems* (P. Braffort and D. Hirschberg, eds.), North Holland, Amsterdam, 1963, pp. 33–70.
5. M. Gross, On the equivalence of models of language, in *Information Storage and Retrieval*, Vol. 2, Pergamon, New York.
6. N. Chomsky, *Aspects of the Theory of Syntax*, M.I.T. Press, Cambridge, Mass., 1965.
7. *American Standard FORTRAN*, ASA X3.9-1966, American Standards Association, New York, March 1966.
8. *American Standard Basic FORTRAN*, ASA X3.10-1966, American Standards Association, New York, March 1966.
9. J. E. Sammet, *Programming Languages: History and Fundamentals*, Prentice-Hall, Englewood Cliffs, N.J., 1969.
10. P. Lucas and K. Walk, On the formal description of PL/I, in *Annual Review of Automatic Programming*, Vol. 6, Part 3, Pergamon, New York, 1969, pp. 105–182.
11. V. H. Yngve and J. E. Sammet, eds., Toward better documentation of programming languages, *Commun. Ass. Comput. Mach.* 6(3), 76–92 (1963).
12. J. E. Sammet, Roster of programming languages 1971, *Comput. Automation* 20(6B), 6–13 (June 30, 1971).
13. J. E. Sammet, Roster of programming languages 1972, *Comput. Automation* 21(6B), 123–132 (August 30, 1972).
14. J. G. Kemeny and T. E. Kurtz, *BASIC Programming*, 2nd ed., Wiley, New York, 1971.
15. IBM, *APL\360 General Information Manual*, GH20-0850, IBM Corp., White Plains, N.Y.

16. M. Klerer and J. May, Further advances in two-dimensional input-output by typewriter terminals, in *Proceedings of the Fall Joint Computer Conference*, Vol. 31, 1967, pp. 675-687.
17. M. B. Wells, Recent improvements in MADCAP, *Commun. ACM* 6(11), 674-678 (1963).
18. *American National Standard COBOL*, ANSI X3.23-1968, New York, 1968.
19. *CODASYL COBOL Journal of Development* 1970, available from Canadian Government Specifications Board, Department of Supply and Services, 88 Metcalfe Street, Ottawa 4, Ontario, Canada.
20. A. Newell *et al.*, eds., *Information Processing Language-V Manual*, 2nd ed., Prentice-Hall, Englewood Cliffs, N.J., 1965.
21. E. C. Berkeley and D. G. Bobrow, eds., *The Programming Language LISP—Its Operation and Applications*, M.I.T. Press, Cambridge, Mass., 1966.
22. V. Yngve, *Computer Programming with COMIT II*, M.I.T. Press, Cambridge, Mass., 1972.
23. R. E. Griswold, J. F. Poage, and I. P. Polonsky, *The SNOBOL 4 Programming Language*, 2nd ed., Prentice-Hall, Englewood Cliffs, N.J., 1971.
24. *FORMAC (Operating and User's Preliminary Reference Manual)*, IBM, 7090R2IBM0016, August 1965.
25. IBM, *PL/I FORMAC Interpreter*, Contributed Program Library, #360D-03.3.004, IBM Corp., Program Information Department, Hawthorne, N.Y., October 1967.
26. A. van Wijngaarden (ed.) *et al.*, Report on the algorithmic language ALGOL 68, *Numerische Mathematik* 14 (1969). Copies can be purchased from the Association for Computing Machinery, New York.
27. J. E. Sammet, An overview of programming languages for specialized application areas, in *Proceedings of the Spring Joint Computer Conference*, Vol. 40, 1972, pp. 299-311.
28. M. H. Perstein, *The JOVIAL (J3) Grammar and Lexicon*, System Development Corp. Santa Monica, Calif., 1966.
29. *Standard Computer Programming Language for Air Force Command and Control Systems*, AFM 100-24, Department of the Air Force (June 1967).
30. *IBM System/360 PL/I Reference Manual*, Form C28-8201, IBM Corp., White Plains, N.Y.
31. S. A. Brown *et al.*, A description of the APT language, *Commun. ACM* 6(11), 649-658 (1963).
32. *Engineer's Guide to ICES COGO I*, R67-46, Department of Civil Engineering, M.I.T., Cambridge, Mass.
33. *ICES STRUDL-II, Engineering User's Manual*, Department of Civil Engineering, M.I.T., Cambridge, Mass., August 1967.
34. IBM, *General Purpose Simulation System V (User's Manual)*, SH20-0851, IBM Corp., White Plains, N.Y.
35. P. J. Kiviat, R. Vallanueva, and H. M. Markowitz, *The SIMSCRIPT II Programming Language*, Prentice-Hall, Englewood Cliffs, N.J., 1969.
36. M. I. Halpern, Foundations of the case for natural-language programming, in *Proceedings of the Fall Joint Computer Conference*, Vol. 29, 1966, pp. 639-649.

Jean E. Sammet and R. Tabory

# ASCII CODE

## INTRODUCTION

Just as people use words to communicate with each other, computers use codes, a code being a system of discrete representation of a set of symbols and functions. During the decade of the 1950s, there was a proliferation of codes developed by computer manufacturers. By the early 1960s there were over two dozen codes just in the United States. Not only did most manufacturers develop unique codes for their machines, but at least one, IBM, had different codes for its different series of computers. The problem was complicated further if a particular machine was used for both scientific and commercial purposes, since accountants desired such characters as %, &, and #, whereas engineers needed (,), +, and =. Unhappily, the type wheels of much of the equipment were limited to 48 characters, which was not enough to satisfy all users. There was also confusion concerning the codes required for the punched cards that machines used. One machine had two codes for the minus sign, the first to be used only in source program cards and the second only for object program output.

The proliferation of the various codes was clearly a disadvantage since it prevented different machines from communicating with one another without going through a costly translation process. By the early 1960s it was obvious to leaders in the field that a solution must be found.

## HISTORICAL DEVELOPMENT OF THE ASCII CODE

Amid the rapid growth that was characteristic of the computer industry during the decade of the 1950s, problems of incompatibility were more often solved through expediency than by a well thought out long-term solution. As time passed, the amount of coded data in files grew larger as did the estimated cost of conversion to a new code. It was imperative that a standard code be established.

The idea of a standard code was not new. As early as 1958, independent groups had been trying to reach agreement. Major efforts had been made by: the Electronics Industries Association, the Department of Defense, the British Standards Institution, IBM, and SHARE (an organization of users of IBM equipment). However, success was not forthcoming. It was in this atmosphere of disagreement among diverse interests that the American Standards Association assigned the ASA Subcommittee X3.2 on Coded Character Sets and Data Formats to seek a solution.

The subcommittee recognized that inasmuch as none of the existing codes was suitable as a standard, since they neither met the requirements of the time nor offered provisions for future requirements, a considerable departure from previous codes

would be essential in the new design. It was only after a careful review of past work in the field and a comprehensive program of original research and code design was completed that a 7-bit coded character set was developed.

On June 17, 1963 the *U.S.A. Standard Code for Information Interchange* (USASCII or more commonly just ASCII and pronounced "ask'-ee") was approved as an American Standard by the American Standard Association. The code included in its choice of graphics the digits, a single case of alphabetic letters A through Z, and those punctuation, mathematical, and business characters considered most useful. The set also included the characters commonly encountered in programming languages. In particular, the COBOL graphics were included, but it was not practicable to include all of the ALGOL graphics.

Twenty-eight of the 128 character positions were left unassigned. Twenty-six of these would be used in a later version for lower case alphabetic letters so that both cases would be available.

Although the standard was designed primarily as a code for information interchange among information processing systems, communications systems, and associated equipment, it could also be used for internal use in information processing equipment since many of the criteria used in establishing the set were processor-oriented, and this has turned out to be the case.

As an internal computer code, it was believed that ASCII would offer programmers many advantages. Bemer [1], who served as an alternate member of ASA Subcommittee X3.2, has cited several of these:

1. Manipulation of graphics by classes. Since all characters of a certain class (such as letters, digits, etc.) are grouped contiguously, they may be classified with very few instructions. In working with strings, it may be useful to create a corresponding class string in parallel for syntax analysis.
2. Fewer instructions in scans, due to regularity and unique codes.
3. Faster and cheaper sorting, when the collating sequence is identical to the binary sequence of the codes for the graphics.
4. Reduction in the number of routines required to be programmed, particularly for satellite equipment.
5. Fewer tables for mixed codes in communications, particularly those controlled by store-and-forward message switching systems.
6. Clarity of printed output, particularly the reproduction of the source program in the printed record of processing.
7. A tendency for keyboards to be identical with typing communications equipment. Thus, hard copy can be available immediately as a record of the program being keypunched.

Since 1963, the code has become widely used throughout the United States and Canada. On March 11, 1968, USA Standard X3.4-1967, which is a revision of the original code, was approved as a Federal Information Processing Standard by the President of the United States, Lyndon B. Johnson. USAS X3.4-1968 [2] (Fig. 1), which is a revision of the 1967 code, was approved on October 10, 1968 by the United States Standards Institute.

## 2. Standard Code

The diagram illustrates the mapping of three most significant bits (b<sub>7</sub>, b<sub>6</sub>, b<sub>5</sub>) to row and column indices. A triangle labeled B, I, T, S maps these bits to a row index (0 to 15) and a column index (0 to 7). The row index is determined by b<sub>7</sub> and b<sub>6</sub>, while b<sub>5</sub> is used for b<sub>4</sub>. The column index is determined by b<sub>4</sub>, b<sub>3</sub>, b<sub>2</sub>, and b<sub>1</sub>.

b <sub>7</sub>	b <sub>6</sub>	b <sub>5</sub>	b <sub>4</sub>	b <sub>3</sub>	b <sub>2</sub>	b <sub>1</sub>	COLUMN ROW	0 <sub>00</sub>	0 <sub>01</sub>	0 <sub>10</sub>	0 <sub>11</sub>	1 <sub>00</sub>	1 <sub>01</sub>	1 <sub>10</sub>	1 <sub>11</sub>
B	I	T	S	b <sub>4</sub>	b <sub>3</sub>	b <sub>2</sub>	b <sub>1</sub>	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	NUL	DLE	SP	0	@	P	'	P
0	0	0	1	1			1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2			2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3			3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4			4	EOT	DC4	\$	4	D	T	d	t
0	1	0	1	5			5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6			6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7			7	BEL	ETB	-	7	G	W	g	w
1	0	0	0	8			8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9			9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10			10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11			11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12			12	FF	FS	,	<	L	\	l	:
1	1	0	1	13			13	CR	GS	-	=	M	]	m	}
1	1	1	0	14			14	SO	RS	.	>	N	^	n	.
1	1	1	1	15			15	SI	US	/	?	0	_	o	DEL

Fig. 1. U.S.A. Standard Code for Information Interchange (U.S.A. Standard X3.4-1968), U.S.A. Standards Institute, New York, 1968.

## DESIGN CONSIDERATIONS FOR THE CODED CHARACTER SET

The success and widespread acceptance of the ASCII codes are due in large part to the perceptiveness of the subcommittee in determining the design considerations for the coded character set. It is clearly evident that the diverse interests of those who made up the committee were fully represented.

There were four main areas of consideration in the design of a standard coded character set intended for the interchange of information among information processing systems and associated equipment. These were: (1) set size, (2) set structure, (3) character selection (both graphic and control), and (4) character placement of the code.

Among the many factors affecting decisions in these four areas were: the need for an adequate number of graphic symbols, device controls, format effectors, communication controls, and information separators; desire for a nonambiguous code; physical requirements of media and facilities; error control considerations; special interpretation of the all zeros and all ones characters; ease in the identification of classes of characters; data manipulation requirements; collating conventions (logical and historical); keyboard conventions (logical and historical); other set sizes; international considerations; programming languages; and existing coded character sets.

## THE DESIGN CONSIDERATION OF SET SIZE

Inasmuch as a 7-bit (binary-digit) set was the minimum size that would meet the requirements for graphics and controls in applications involving general information interchange, it was incumbent on the committee to decide on that number. An eighth bit can be used as a parity bit.

## THE DESIGN CONSIDERATION OF SET STRUCTURE

In discussing the set structure it is convenient to divide the set into 8 columns and 16 rows. It was considered essential to have a dense subset which contained only graphics. For ease of identification, this graphic subset was placed in six contiguous columns. The first two columns were chosen for the controls for the following reasons: (1) the character NUL by its definition has the location 0/0 in the code table, NUL is broadly considered a control character; and (2) the representations in Column 7 were felt to be most susceptible to simulation by a particular class of transmission error—one which occurs during an idle condition on asynchronous systems. To permit the considerations of graphic subset structures to be satisfied, the two columns of controls had to be adjacent. Finally, the structure should be so designed as to enable the easy identification of classes of graphics and controls.

## THE DESIGN CONSIDERATION OF CHARACTER SELECTION—CHOICE OF GRAPHICS

Included in the set are the numerals 0 through 9, upper and lower cases of the alphabetic letters A through Z, and those punctuation, mathematical, and business symbols considered most useful. The set also includes a number of characters commonly

encountered in programming languages. In particular, all the COBOL and FORTRAN graphics are included.

In order to permit the representation of languages other than English, one diacritical (or accent) mark has been included, and provision has been made for the use of five punctuation symbols alternately as diacritical marks.

## THE DESIGN CONSIDERATION OF GRAPHIC SUBSET STRUCTURE

The basic structure of the dense graphic subset was influenced by logical collating considerations, the requirements of simply related 6-bit sets, and the needs of typewriter-like devices. For information processing it is desirable that the characters be arranged in such a way as to minimize both the operating time and the hardware components required for ordering and sequencing operations. This requires that the relative order of characters, within classes, be such that a simple comparison of the binary codes will result in information being ordered in a desired sequence.

Conventional usage requires that the SP (space) be ahead of any other symbol in a collatable set. This permits a name such as "JOHNS" to collate ahead of a name such as "JOHNSON." The requirement that punctuation symbols such as *comma* also collate ahead of the alphabet ("JOHNS, A" should also collate before "JOHNSON") establishes the special symbol locations, including SP, in the first column of the graphic set.

To simplify the design of typewriter-like devices, it is desirable that there be only a common 1-bit difference between characters to be paired on keytops. This, together with the requirements for a contiguous alphabet and the collating requirements outlined above, resulted in the placement of the alphabet in the last four columns of the graphic subset and the placement of the numerals in the second column of the graphic subset.

It is expected that devices having the capability of printing only 64 graphic symbols will continue to be important. It may be desirable to arrange these devices to print one symbol for the bit pattern of both upper and lower case of a given alphabetic letter. To facilitate this, there should be a single bit difference between the upper and lower case representations of any given letter. Combined with the requirement that a given case of the alphabet be contiguous, this dictated the assignment of the alphabet, as shown in Columns 4 through 7.

To minimize ambiguity caused by the use of a 64-graphic device as described above, it is desirable, to the degree possible, that each character in Column 6 or 7 differ little in significance from the corresponding character in Column 4 or 5. In certain cases this was not possible. The assignment of *reverse slant* and *vertical line*, the *brackets* and *braces*, and *circumflex* and *overline* were made with a view to the same considerations.

The resultant structure of "specials" (S), "digits" (D), and "alphabetics" (A) does

not conform to the most prevalent collating convention (S-A-D) because of other more demanding code requirements.

The need for a simple transformation from the set sequence to the prevalent collating convention was recognized, and it dictated the placement of some of the "specials" within the set. Specifically, those special symbols, viz., *ampersand* (&), *asterisk* (\*), *comma* (,), *hyphen* (-), *period* (.), and *slant* (/), which are most often used as identifiers for ordering information and which normally collate ahead of both the alphabet and the numerals, were not placed in the column containing the numbers, so that the entire numeric column could be rotated via a relatively simple transformation to a position higher than the alphabet. The sequence of the aforementioned "specials" was also established to the extent practical to conform to the prevalent collating convention.

The need for a useful 4-bit numeric subset also played a role in the placement of characters. Such a 4-bit subset, including the digits and the symbols *asterisk*, *plus* (+), *comma*, *hyphen*, *period*, and *slant*, can easily be derived from the code.

Considerations of other domestic code sets, including the Department of Defense former standard 8-bit data transmission code ("Fieldata," 1961), as well as international requirements, played an important role in deliberations that resulted in the code. The selection and grouping of the symbols *dollar sign* (\$), *per cent* (%), *ampersand* (&), *apostrophe* ('), *less than* (<), *equals* (=), and *greater than* (>) facilitate contraction to either a business or scientific 6-bit subset. The position of these symbols, and of the symbols *comma*, *hyphen*, *period*, and *slant*, facilitates achievement of commonly accepted pairings on a keyboard. This historic pairing of *question mark* and *slant* is preserved; and the *less than* and *greater than* symbols, which have comparatively low usage, are paired with *period* and *comma* so that in dual-case keyboard devices where it is desired to have *period* and *comma* in both cases, the *less than* and *greater than* symbols are the ones displaced. Provision was made for the accommodation of alphabetics containing more than 26 letters and for 6-bit contraction by location of low-usage characters in the area following the alphabet. In addition, the requirement for the digits 10 and 11 used in sterling monetary areas was considered in the placement of the *asterisk*, *plus*, *semicolon*, and *colon*, so that the 10 and 11 could be substituted for the *semicolon* and *colon*.

## THE DESIGN CONSIDERATION OF CONTROL SUBSET CONTENT AND STRUCTURE

The control characters included in the set are those required for the control of terminal devices, input and output devices, format, or communication transmission and switching on a general enough basis to justify inclusion in a standard set.

Many control characters may be considered to fall into the following categories:

1. Communication controls
2. Format effectors
3. Device controls
4. Information separators

To the extent practical, controls of each category were grouped in the code table. The structure chosen also facilitates the contraction of the set to a logically related 6-bit set.

The information separators (FS, GS, RS, US) identify boundaries of various elements of information, but differ from punctuation in that they are primarily intended to be machine sensible. They were arranged in accordance with an expected hierarchical use, and the lower end of the hierarchy is contiguous in binary order with SP (space) which is sometimes used as a machine-sensible separator. Subject to this hierarchy, the exact nature of their use within data is not specified.

The character SYN (Synchronous Idle) was located so that its binary pattern in serial transmission was unambiguous as to character framing, and also to optimize certain other aspects of its communication usage.

ACK (Acknowledge) and NAK (Negative Acknowledge) were located so as to gain the maximum practical protection against mutation of one into the other by transmission errors.

The function "New Line" (NL) was associated with LF (rather than with CR or with a separate character) to provide the most useful combinations of functions through the use of only two character positions, and to allow the use of a common end-of-line format for both printers having separate CR-LF functions and those having a combined (i.e., NL) function. This sequence would be CR-LF, producing the same result on printers of both classes, and would be useful during conversion of a system from one method of operation to the other.

## REFERENCES

1. R. W. Bemer, The American Standard Code for Information Interchange. Part Two, *Datamation* 9, 39-44 (September 1963).
2. U.S.A. *Standard Code for Information Interchange*, U.S.A. Standards Institute, New York, 1968.

*Sumner E. Mendelson*

# ASSOCIATION FOR COMPUTING MACHINERY (ACM)

Founded in 1947 as the society of the computing community, the Association for Computing Machinery (ACM) is dedicated to the development of information process-

ing as a discipline, and to the responsible use of computers in an increasing diversity of applications.

The purposes of the Association are [1]:

To advance the sciences and arts of information processing including, but not restricted to, the study, design, development, construction, and application of modern machinery, computing techniques and appropriate languages for general information processing, for scientific computation, for the recognition, storage, retrieval, and processing of data of all kinds, and for the automatic control and simulation of processes.

To promote the free interchange of information about the sciences and arts of information processing both among specialists and among the public in the best scientific and professional tradition.

To develop and maintain the integrity and competence of individuals engaged in the practices of the sciences and arts of information processing.

The ACM has been, in many ways, coterminous with the development and use of electronic stored-program digital computers.

The Association was formed by founding workers [2] of the discipline of automatic computation, early in the development [3, 4] of the EDSAC, the EDVAC, and the IAS machines, in anticipation of the need for communication between people interested in computing.

Its membership has grown in size (29,000 in 1974) to contain a significant fraction of the full-time computing specialists in the United States and elsewhere. It has grown in breadth to include, in addition to the founding mathematicians, engineers, and physical scientists, technical and administrative workers in almost all fields of human endeavor where large amounts of information can usefully be processed.

Its formal refereed serial publications include the quarterly archival *Journal*, the monthly technical *Communications*, the critical review journal *Computing Reviews*, and the survey/tutorial *Surveys*; refereed nonserials include the *Monographs* volumes and *Conference Proceedings*.

Its conferences include those directed to the interests of all people interested in computing (the ACM Annual Conferences and participation in the National Computer Conferences<sup>1</sup>); symposia in single technical areas and directed toward both developers and users of computing technology; an annual academic-community Computer Science Conference; and technical workshops for advanced workers in specialized problem areas.

Its membership structure has organized into subsets divided along both geographical and special-interest axes, with regular chapters in 89 locations inside the United States and 9 elsewhere, with 121 student chapters in educational institutions, and with 28 Special Interest Groups and Committees.

One hundred and eighty-six educational institutions and 18 industrial firms and other organizations are now (1974) ACM institutional members.

The Association maintains communication and/or direct working relationships with a broad range of external organizations including (to name a few) the National Research Council, the American Association for the Advancement of Science, the Data Processing Management Association, and the X3 Sectional Committee on Computers and Information Processing of the American National Standards Institute. It also has cooperative joint membership and/or exchange-subscription arrangements with several other societies including, e.g., the (United States) Institute of Electrical and Electronic Engineers Computer Society (IEEE-CS) and the British Computer Society. ACM is one

<sup>1</sup> Until 1973, called Joint Computer Conferences.

of the founding member societies of the American Federation of Information Processing Societies (AFIPS), which is the United States member of the International Federation for Information Processing.

The ACM's publications and other activities, as well as its governing body (the council), early reflected its membership and leadership concentration on research and development, with substantial participation by members of the academic and government research laboratory communities. The *Journal*, after a short period of carrying descriptive articles on early equipment and methods, developed into a scholarly research publication with many papers on fundamental developments in nonnumeric methods and theory (e.g., computability and finite automata) and in numerical mathematics. The *Communications*, which had started in 1958 as an informal quick-publication and news medium, soon became largely a refereed technical serial publication.

Another unique ACM service originated in February 1960 with publication in the *ACM Communications* of the *Algorithms* section. Initially expressed only in ALGOL 60 language, and later in FORTRAN language as an option, basic computational procedures are published in forms which permit broad application. This section publishes *Certifications* (results of independent tests on) and also *Remarks* (consisting of expository comments regarding) particular previously published *Algorithms*. The Association makes publicly available a continually updated publication consisting of *Collected Algorithms*, including *Certifications* and *Remarks* previously published, in loose-leaf binder form for use as a working handbook. In 1975 this work will be published in a new ACM serial, *Transactions on Mathematical Software* (TOMS).

Also in 1960, initially bound with *Communications* but separate as of its second year, *ACM Computing Reviews* appeared as the only critical review journal in the computing field. *ACM Computing Reviews* quickly gained recognition and remains unique in the field. In 1969 a survey and tutorial journal, *Computing Surveys*, was initiated and it, too, rapidly became known in the computing community.

The *ACM Monographs* series was initiated in 1961 to provide a publication vehicle for scholarly works of permanent significance. In 1964, full texts of technical submitted papers at the ACM Annual Conference were published in a *Conference Proceedings*. Every year since then, with the exception of 1970 when the conference was a unique series of informal colloquies with consumers of computing (and the *Proceedings* was edited transcripts of those colloquies), this publication has been produced for each ACM Annual Conference.

Membership and corresponding Association interests broadened considerably during the mid-1960s to include more applications-oriented topics. The serial publications reflected this change by including a growing amount of content directed toward problem solution rather than basic methodology. The same broadening of membership interest also affected ACM conference planning and content. The annual conferences of the Association each year included more and more application-oriented sessions, as did the specialized symposia and workshops. Participation by the (voluntary) Special Interest Groups (SIG) and Special Interest Committees (SIC) has increased steadily in the planning and leadership of major conference activities as well as in technical contributions by SIG and SIC members.

As the first quarter-century of ACM existence ended, growing interest was evident

in professional activities of the membership. After meeting resistance during the 1960s, an ACM Code of Ethics gained membership acceptance. As of 1974, members of the Association were showing increasing interest in the possibility of establishing an enforcement procedure for the code. Sparked by some legislative activities in individual states of the United States toward programmer licensing, membership interest has been growing in the development of standards of professional competence.

During the late 1960s, a group of educators within the ACM membership developed an array of curricula for education in computer science at various levels from the undergraduate to the doctoral degree. This work has had broad impact, particularly at the graduate education level, in universities in the United States. Similarly, extensive work was done during this same period in developing criteria for accreditation of training schools; this work, also, substantially affected computer-related teaching in the United States.

## GOVERNMENT OF THE ACM

The Association is governed by a council elected (by custom, usually in contested elections) by secret ballot. The council consists of twenty-five persons:

President, vice president, secretary, elected by direct vote of all regular members.

6 Members-at-large, also elected by direct vote of all regular members.

12 Regional representatives, each elected by direct vote of all regular members in a region.

Chairman of the publications board and treasurer are elected by the council.

Chairman of the SIG/SIC board is elected by the chairmen of SIGs and SICs.

The (immediate) past president.

The council usually meets three times per year and can act by mail ballot. An Executive Committee, consisting of the president, vice president, and secretary, meets more often to conduct the detailed management business of the association.

### ACM Committee Structure

In 1971 the ACM committee structure was reorganized under six boards, each headed by a chairman reporting to the vice president.

The ACM's Standing Committees (established in the bylaws) and *Ad Hoc* Committees (established by president's decision) are appointed by and serve at the pleasure of the president. They constitute the appointive structure of the Association, carrying out the editorial, advisory, and operational functions as supported by the headquarters professional staff.

Special Interest Groups are voluntary-participation organizations composed of workers in particular fields of computing. Each SIG is led by officers elected by its membership or appointed by the president—the choice is theirs; is supported in part

by membership dues; most publish serial newsletters with substantive technical content (in general, not refereed); may hold specialized conferences either independently or in conjunction with other organizations inside or outside ACM, drawing participants from geographic areas which may be as limited as a single metropolitan complex or as broad as the international computing community; and may have local SIGs associated with chapters. Their quick communication of new ideas within their specialist membership contributes to progress in the field.

Special Interest Committees are similar in function to SIGs. SIC officers are appointed by the president, commonly in response to a petition by concerned ACM members to initiate the SIC in a new area of technical interest. The normal lifetime of a SIC is 1 year, by the end of which it is expected to convert to SIG status or to disband.

All active SIGs and SICs report to the SIG/SIC Board, which is headed by a chairman elected by the chairmen of SIGs and SICs, and who is a member of council.

The editorial functions of the ACM report to the Publications Board via the Editorial Committee. The Publications Board is also responsible for business management of all of the major publications of the ACM. Both technical and management aspects of major conferences of the ACM are handled by the Conferences and Symposia Committee, under the Management Board. Geographical subsets of the ACM, consisting of Regular and Student Chapters, report through various committees to the Members and Chapters Board. Seven committees report to the External Activities Board.

ACM's delegation to the AFIPS Board of Directors, consisting of the current President plus two other Directors elected by the council, reports to the council.

## REFERENCES

1. From the masthead statement, *Commun. ACM* 18(3), (March 1975).
2. The Association was founded by F. L. Alt, E. C. Berkeley, R. V. D. Campbell, J. H. Curtiss, H. E. Goheen, J. W. Mauchly, T. K. Sharpless, R. Taylor, and C. B. Tompkins. Interestingly enough, the most widely known computing pioneer, John von Neumann, declined to take part in ACM because he felt that (as expressed in his letter of September 15, 1947, to Mr. Berkeley) "... [although] such an association [would be] highly desirable ultimately, . . . the general situation [had] not matured sufficiently." The basic historical reference for this article has been F. L. Alt, Fifteen years of ACM, *Commun. ACM* 5 (6), 300-307 (June 1962).
3. L. Revens, The first twenty-five years, ACM 1947-1972, *Commun. ACM* 15(7), 485-490 (July 1972).
4. S. Rosen, Electronic computers: A historical survey, *Comput. Surveys* 1(1), 7-36 (March 1969).

Herbert S. Bright

# ASSOCIATIVE MEMORIES AND PROCESSORS: A DESCRIPTION AND APPRAISAL

## INTRODUCTION

Associative memory systems have been under active investigation since 1955 when, as reported by Slade [1], the late Dudley Buck proposed the first cryotron associative memory which he called a recognition unit. Work in this field received its impetus from the efforts of Slade and McMahon [2] who described a cryogenic catalog memory cell constructed on the same principle as suggested by Buck. Associative memories have been referred to variously as catalog memories [2], associative memories [3], parallel search memories [4], content-addressable memories [5, 6], and data-addressed memories [7, 8].

As noted by Slade [1]:

The earliest associative memories were probably table look-up memories which used magnetic cores in a conventional manner. These memories contained information ordered on a certain field and a word could be retrieved by a partitioning search requiring  $\log_2 N$  memory cycles (where  $N$  is the number of words in the memory). This type of memory is, of course, limited in its size and usefulness.

The partitioning search referred to by Slade is better known as a binary search operation. The value  $\log_2 N$  is achieved when  $N$  is of the form  $2^n$ .

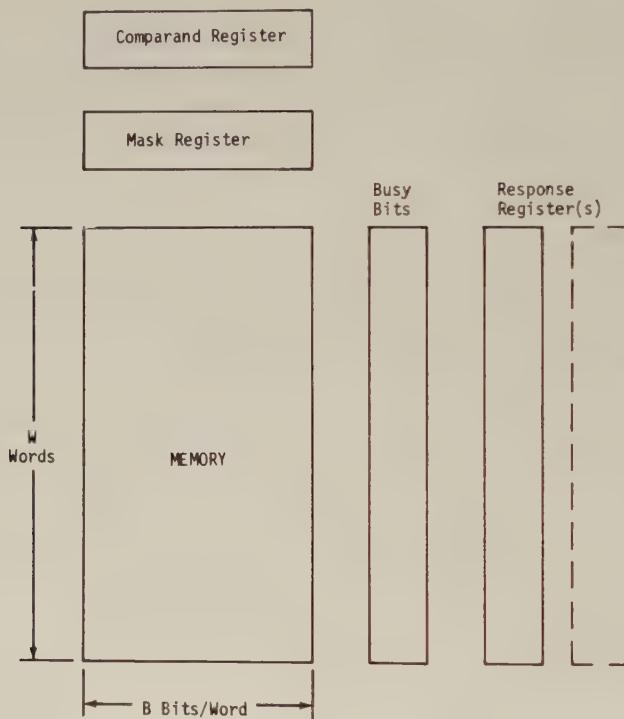
The International Federation of Information Processing Society Glossary defines an associative store as "a *store* whose *registers* are not identified by their name or position but by their content" [9]. In explaining the definition by example, the glossary further notes that the retrieval of any one item in such a store would be accomplished by searching all registers in parallel to retrieve the relevant data by a content search with but a single operation. An associative memory consists of a number of associative stores. In this article, associative memories and processors are described from different aspects.

This article is subdivided into several major sections. In the following section we describe the various components that comprise an associative processor. The next section describes the manner in which the associative capability is achieved for associative memories. The major technologies used to construct associative memories: magnetics, cryoelectronics, and semiconductors are described in the following section together with the manner in which the technologies have been applied in associative memory implementations. A related topic to associative memories, that of parallel processors, is covered in the same section. In the next section the interface organizations that have been achieved between associative memories and conventional computers are covered. An associative processor that represents the state-of-the-art in

1971 is described in the following section. Algorithms and software for associative memories are discussed in the next section as are numerous proposed areas of application for associative memories. The final section summarizes the state of associative technology in 1971.

## ASSOCIATIVE PROCESSOR COMPONENTS

An associative processor consists of an associative memory and additional hardware to permit manipulation of the data in the memory store.



**Fig. 1.** Basic structure of an associative memory processor.

Typical elements of an associative processor are shown in Fig. 1 and are as follows [10]:

1. *The associative memory array* which provides the data storage itself.
2. *The comparand-register* which contains the data to be compared against the contents of the memory array for searches; may provide a shifting register for some input/output operations; and can play an intermediate role in the transfer of data between

the associative memory array and storage units of a general purpose computer depending upon the configuration in which the processor is employed.

3. *The mask register* which is used to contain data specifying those portions of the word in the comparand-register that pertain to the search operation.
4. *The resolver* which is used to determine the location of response bits in the response store.
5. *The search logic* which causes the search commands received by the memory to be executed properly. Search operations are accomplished in a bit-serial, word-parallel fashion starting at the most significant bit (that is, the hardware operates on the same bit position of all words in parallel and upon completing the operation moves to the next bit position); or if distributive logic is used, in a word-parallel, bit-parallel fashion (i.e., operations on words are performed in parallel, and all bits in each word are operated upon in parallel).
6. *The response store (register)* which receives vectors indicating which data satisfy a given search criterion and which can execute logical operations, such as shifting and Boolean operations on these vectors. Each bit in a response store corresponds to a word in the associative array.

An associative memory developed for the National Aeronautics and Space Administration in 1971 by Honeywell Information System contains the six elements described above (see below in the subsection titled Associative Memories and Semiconductor Technology). Associative memories available in 1971 had, in addition to a nondestructive read operation (i.e., the ability to read a memory word without destroying its contents), the following capabilities:

1. Multiwrite into a bit slice [i.e., the ability to write into the same bit position of all selected (or desired) words].
2. Multiarithmetic operations between fields of a word or words [i.e., the ability to add two specified fields in the same word (or words) in parallel for all selected (or desired) words in the memory].
3. Word addressing capability (i.e., the ability to access a specific word in the associative array).

Typical search logic includes some or all of the following operations: (1) search on equality, (2) search on inequality, (3) search on maximum, (4) search on minimum, (5) search on greater than, (6) search on less than, and (7) search between limits. Other search operations may also be provided.

Results of the search operations reside in the response store. There is typically more than one response store (or a single response store with several bits per word). Multiple response stores are required for complex search operations and also to store the results between search operations. (In some search operations, such as a search on maximum, it is necessary to make several passes over the entire memory. The result of each pass must be saved for later operations.) An ability exists to perform logical operations between response store bits and to end-around (or circular) shift the response store in either direction between adjacent words under program control.

The importance of the associative memory lies not only in the ability to search by content, but to do so in parallel over an entire memory within approximately the same time as required to access a single memory element in a random access oriented memory. Because of this capability, numerous speculative articles have been written

specifying various fields of application where such memories would be useful. The articles were speculative primarily because associative memories and associative processors were not available for actual experimentation. The excellent survey article on associative memories by Hanlon [11] and the overview and comprehensive bibliography on associative memory technology by Minker [12] list numerous such applications. Some of these applications are discussed below in the section titled Software and Associative Memories.

Until 1968, associative memory arrays of more than a few words were laboratory feasibility models typified by a few words and a number of bits per word. For example, Newhouse and Fruin [7, 8] claimed the feasibility of a 300,000 bit memory based on experiments with a three-word module consisting of 81 crossed-film cryotrons. The IBM 360/67 [13-15] contains an eight-word associative memory. In 1968 Goodyear Aerospace Corp. delivered a 2048 word by 49-bit associative processor to Rome Air Development Center [16] and announced, in 1971, the STARAN S associative processor [17], the first such processor commercially available. The STARAN S processor is described in detail below in the section titled A State-of-the-Art of Associative Processors in 1971—The STARAN S.

Early work in associative memories gave rise to the hope that large associative memory arrays of the order of  $10^{10}$  bits would be available during the latter part of the 1960s [18, 19]. These expectations were not met. Furthermore, there is no evidence available in early 1972 that indicates that associative memory arrays will be available with a capacity of  $10^{10}$  bits within the 1970s. Because of the inability of the technology to achieve large associative memories, attention has been focused on the problem of developing clever organizations of computer systems that contain one or more small associative memory arrays. The special purpose Parallel Element Processing Ensemble (PEPE) system [20, 21] (also see Ref. 22), under development in the late 1960s and early 1970s for processing missile tracks in an antiballistic missile application, is such a configuration. Various ways in which to organize an associative processor in a computer system are described below in the section titled Associative Memory Organizations in Computer Systems.

## ASSOCIATIVE MEMORY OPERATIONS

Computer memories have been organized conventionally as sets of words, each word comprising a fixed number of binary digits (bits). Associated with each word is an address which corresponds in some regular manner with the physical location in memory at which the word value is stored. Access to a word in memory requires the designation of an address in the form of a binary number. The address value is translated by appropriate decoding logic circuits to provide the memory-access control signals.

The memory hardware technology required for developing an associative array is no different from that used for the development of word-addressed memories. The difference lies in the amount of logic associated with a word or a memory cell. The

logic depends upon whether a word-parallel, bit-serial, or a word-parallel, bit-parallel approach is used to achieve the associative capability. Hence, the built-in logic applied to the memory achieves the associative capability. The operations of a word-parallel, bit-serial, and word-parallel, bit-parallel associative memories are described in the following subsections.

### Operation of a Word-Parallel, Bit-Serial Associative Memory

An associative memory that operates in a bit-serial, word-parallel fashion may be described as follows with reference to Fig. 2 (based on the Goodyear Associative Processor, GAP) [23].

The associative array is loaded by transferring  $n$ -bit words into the data circuits, then transferring them to the comparand register which, along with the mask register, controls the data entry into the memory array through the bit circuits. The data word is loaded into a specific array location specified by the address register, even though the instructions being executed may have been given by a load instruction of data words into those locations responding to a previous search operation. In the latter case the address register would have been loaded sequentially from the response register scanner. (The connections for this transfer are omitted from Fig. 2.)

A data word to be searched on is placed in the comparand register along with a mask word which is stored in the mask register. Assuming that the associative memory is implemented in a computer and driven by a program stored in the main memory, instructions are sent to the associative memory by external function signals (EXF) and are synchronized by synchronizing circuits (SYNC) and interpreted by the control section.

During execution of a search instruction, the results of each interrogation are sensed by the sense logic and stored in the response register. There may be a need for multiple response registers to save partial results during the search instruction. Associated with each word is match logic hardware (which, in the case of the GAP, was in the form of sense amplifiers and flip-flops). To reduce the amount of logic required, the memory may be segmented into several parts, and the logic shared by the several parts. In this case a search over the entire memory must be performed one segment at a time with results stored (as, for example, having multiple responders and storing the results of the search in a responder). One reason for sharing the logic is to economize, while a second reason is to decrease the power requirements needed to drive large memories.

The control logic shown in Fig. 2 contains the instructions required to search and store data in the associative memory array. Some bits of the memory may be used as tag bits (for example, a bit may be set aside as a "busy bit" to mark those locations currently storing data). The busy bit is set to a "1" or a "0" when data are stored in that location under program control. In performing search operations and other instructions, the control logic implements various algorithms. A typical algorithm that might be implemented is a search on equality of the comparand register as specified by the mask register. This algorithm is as follows:

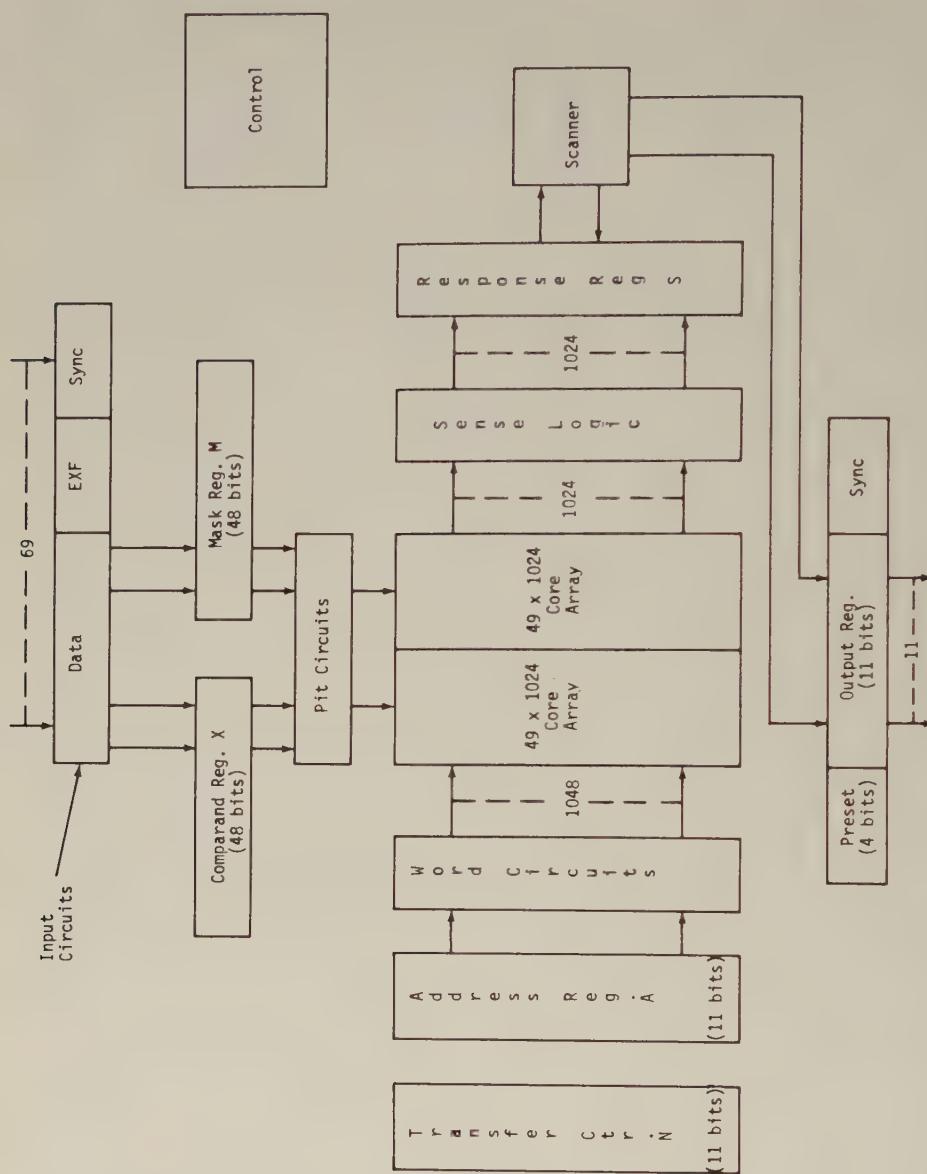


Fig. 2. Logic block diagram for a small associative processor.

1. Place the comparand in the comparand register.
2. Place mask in the mask register.
3. Set the response register to all ones.
4. Set  $j$  to zero ( $j \leftarrow 0$ , where  $j$  represents the  $j$ th bit position).
5. if  $j$  is less than the number of bits in the word, continue; otherwise exit.
6. Increment  $j$  by one ( $j \leftarrow j + 1$ ).
7. If the  $j$ th bit of the mask register is zero, go to 5; otherwise continue.
8. For all  $k$ , intersect the  $k$ th bit of the response register with the result of intersecting the  $j$ th bit of the comparand with the  $j$ th bit of the  $k$ th word and place the result in the  $k$ th bit of the response register, go to 5.

Wherever the response store contains a "1" in a bit position, the corresponding word satisfies the search criterion. Readout of the memory is achieved by means of the scanner and the output register. Addresses of responders to searches are successively determined by the scanner and may be placed in an output register if so required.

### **Operation of a Word-Parallel, Bit-Parallel Associative Memory**

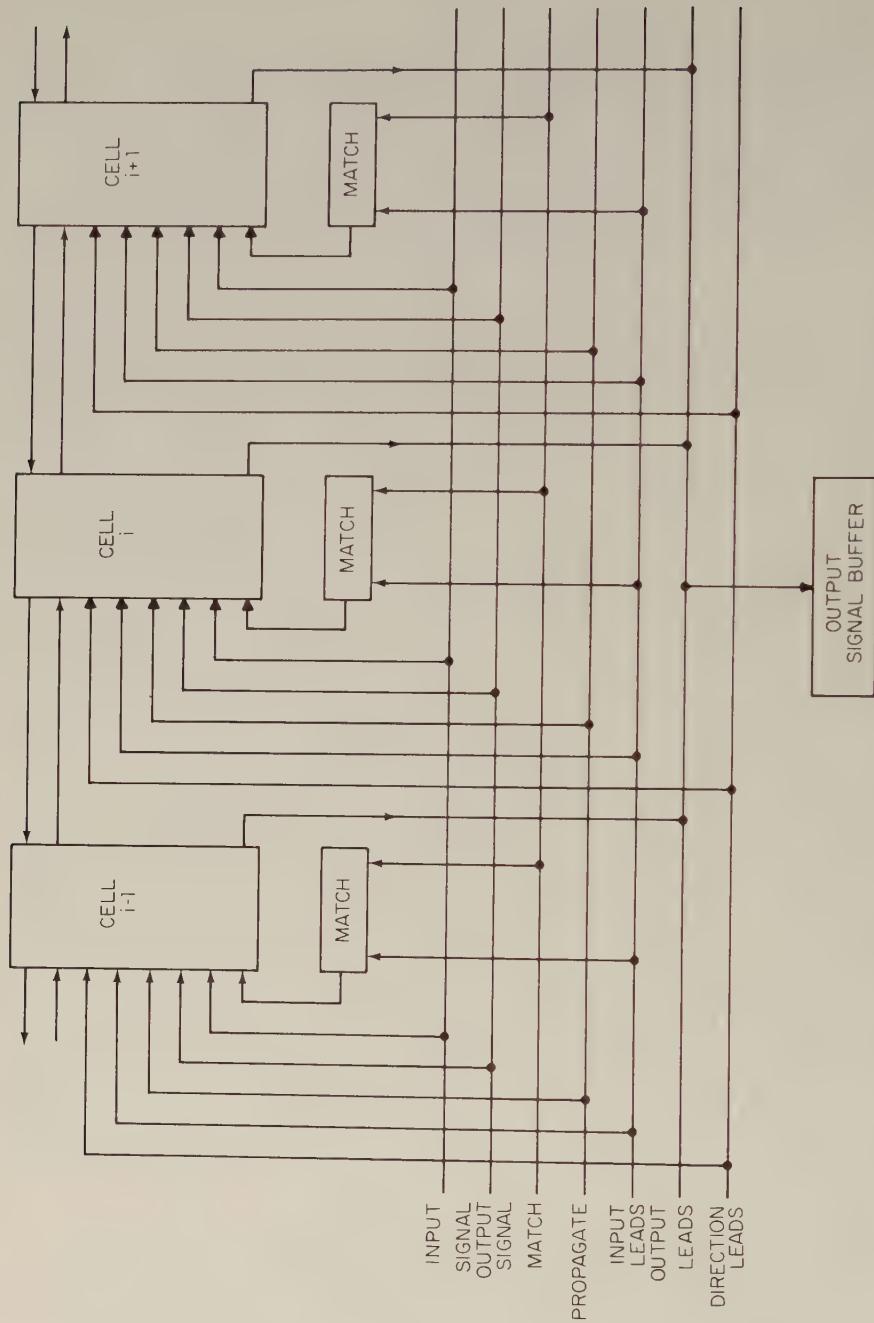
A true associative memory, in the sense of completely simultaneous interrogation of all bits in all words of a block of memory, requires that each elementary memory cell be furnished with a logic operation capability—essentially that of performing an exclusive-or between the associative cell content and the corresponding test bit value. A mismatch at the bit cell level produces a sense pulse; if this sense pulse is fed to a sense line common to all (tag) bits of a given word, then an automatic or-ing (inclusive) of mismatches of the tag bits results. Word flagging for a match word, therefore, corresponds to the absence of a mismatch signal on the word sense line.

Because of the added complexity of logic incorporated into each memory bit and the relatively high power required to simultaneously interrogate all selected tag bits of all words, true associative memories have been relatively expensive. Consequently, they have been developed and applied only for special cases where a small memory with associative characteristics is desired.

The concept of an associative memory based on distributive logic was developed by Lee [24] in 1962. The distributed logic organization (Fig. 3) described by Lee is constructed from cells that can store one character, connected together into a linear strip. Each cell is either active or inactive and uses a common control with four types of control leads: input, output, match, and propagate. When an input control signal is received and a cell is active, the contents of the input line are stored in the cell, and similarly on the output. The operations that take place in a cell are described by Murtha [25] as follows:

When a match signal appears, the cell compares its contents with the contents of the input line. If a match is obtained, the cell sends a signal to a neighboring cell causing the neighbor to become active. When a cell is active and a propagate command is received, a signal is sent to a neighboring cell which then becomes active, the original cell becoming inactive. The direction of propagation is controlled by two control leads: R and L.

Name fields are defined by use of a special character A, which appears at the beginning of the field. Retrieval is performed as follows: First a match is performed on the special character A thus setting all cells which begin a field to an active state. Then a



**Fig. 3.** Distributive logic intercommunicating cells memory organization.

match is performed on each character of the search key, one at a time. As the comparison continues, the initial activity propagates down the field as long as a continuous match is obtained. At the end of this process, only those name fields whose content is the same as the search key have an active cell next to their right-most cell.

The corresponding parameter field is stored to the right of the name field. All parameter fields start with the special character B. All cells with a B and which are preceded by a matching key have been set active by the previous search process. This active cell forms the basis for the output of the parameter field. Each of the characters is retrieved (fetched) in sequence until the special character A is met. The original activity in the B cell propagates through the parameter field as a tag for the retrieved. A similar process can be used to retrieve the name (from the name field) of the item stored in the parameter field.

## ASSOCIATIVE MEMORY TECHNOLOGY

An associative memory may be constructed using many different technologies. In the following we describe the technologies of magnetics, cryoelectrics, and semiconductors that have been used for constructing associative memories in the 1960s and early 1970s. Other technologies used are noted briefly. Each of the three technologies mentioned is first described. Following each description is a discussion of the use of the particular technology to construct associative memories.

### Magnetic Technology

The high-speed large-capacity memory market has been dominated by the ferrite core array memory since the early 1950s. Factors contributing to this dominance are:

1. Favorable physical characteristics inherent in magnetic phenomena.
2. Coincident-current selection, reducing selection electronics.
3. Steady evolutionary improvement in ferrite technology.
4. Cost reductions resulting from accumulated experience, increased automation, and large volume production.

In a broad review of computer memory technology up to 1969, Rajchman [26] reflects this dominance by devoting a significant portion of his text to ferrite core and related magnetic technologies.

Ferrite cores have an essentially square hysteresis loop with well-defined switching thresholds between two stable states. Both states are nonvolatile, requiring zero maintenance power. Absence of eddy-current dissipation in switching the low-conductivity ferrite material contributes to fast switching which is achieved by a coincident-current technique. If two current loops thread a core and are pulsed with currents at controlled levels, a pulse in only one loop will not be sufficient to cause switching, while simultaneous pulsing of both loops will reliably cause switching.

Ferrite memory array organizations based upon coincident-current switching arranges cores in a rectangular matrix for each bit plane. All cores in each row share

a common drive wire and switching drive network; likewise, all cores in each column share a common drive. When one selected row drive and one selected column drive are pulsed simultaneously (using properly chosen current amplitudes), only one core in the matrix receives sufficient linked current to be induced to switch (assuming the initial core state was opposite to the state induced by the combined drive pulse). The selected core accessed is the one at the intersection of the selected row and selected column. The switched core output is sensed as a pulse conducted on a common sense wire threading all the cores in a matrix and terminating in a single sense amplifier. Memory readout is "destructive" in that a core is driven to a standard state (say, zero); if an output pulse is sensed, the core state was a one, otherwise a zero. A one state is reestablished by reversing the polarity of the coincident-current pulses.

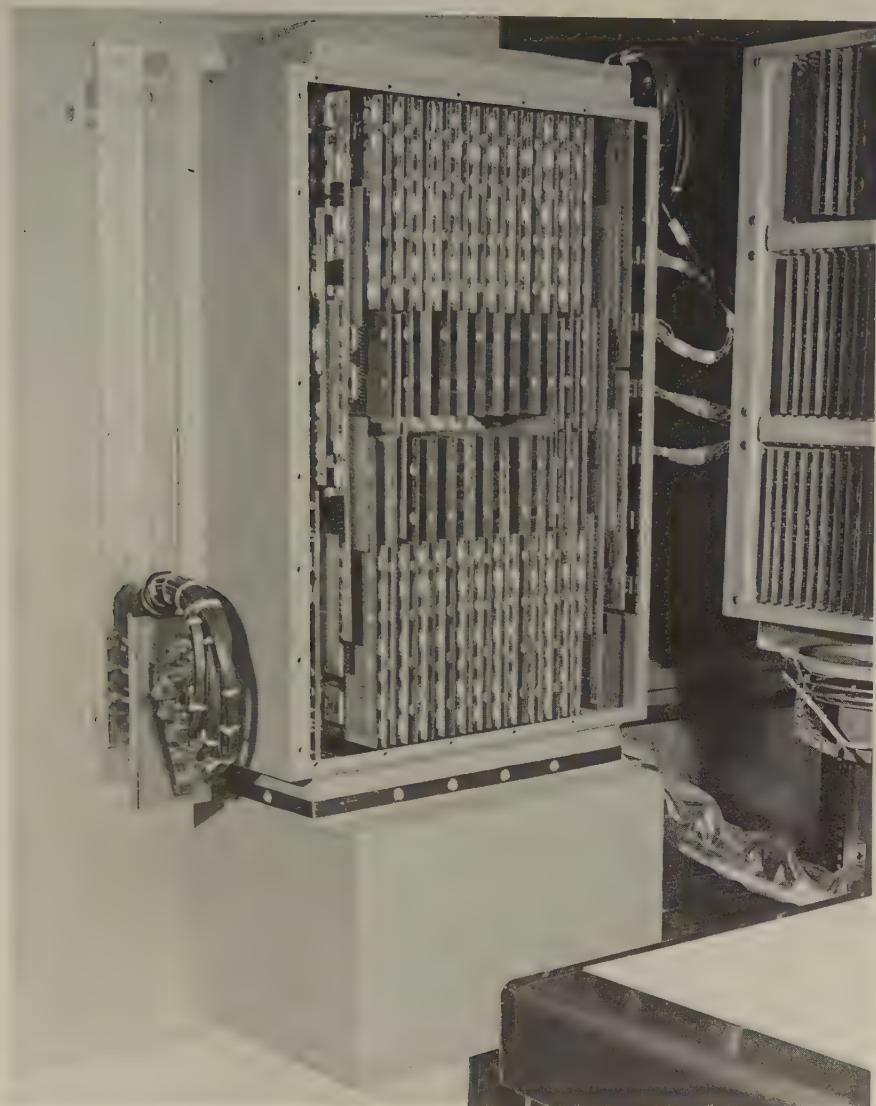
Several variations of core array organization, differing in details of selection logic as well as in arrangement and numbers of conductors threading the cores, are employed in memory systems available in 1971 [26].

A complete ferrite memory unit employs a multiplicity of core matrices in a "stack"; the number of matrices or planes in the stack corresponds to the number of bits per memory word. All bits of a selected word are read out in parallel through separate sense amplifiers, one for each plane. Corresponding row or column drive lines in all the planes share a common drive circuit. Consequently, for a stack of  $p$  planes, each of dimension  $m$  rows by  $n$  columns, a total of  $m + n$  drive circuits (each of drive power proportional to  $p$ ) and  $p$  sense circuits and inhibit circuits suffice to access and restore words in a core memory unit of capacity  $mnp$  bits.

Practical core matrix size is limited by core uniformity. A high degree of uniformity is achieved in the 1970s technology by careful control of core composition and fabrication, together with individual testing and selection of cores by high-speed automated testing devices. The resultant practicable size of core arrays leads to favorable economics for ferrite technology in that the proportionate amount of drive and sense electronics per bit becomes relatively small.

Access time performance of ferrite memories is improved by reducing core size; ferrite composition is also a significant factor in switching speed. The tiny cores now utilized (outer diameter 0.012 in., inner diameter 0.007 in.) are probably near the limit of practicality; smaller cores would be extremely fragile and would require threading wires of such small diameter that electrical resistivity would be too large. Only marginal improvement in performance can be expected over the typical read-write cycle of the order of 750 nsec for high-performance computer main memory.

A magnetic memory technology related to that of ferrite cores employs conductive wires plated with a suitable metallic magnetic material. These memories are known as plated-wire memories. With plated wires organized as rows of a matrix, and unplated conductors superimposed as columns, the magnetic plating in the immediate vicinity of each row-column intersection acts essentially like a core element. Thus a direct analog exists between ferrite and plated-wire memories. However, plated-wire memories can be fabricated so as to be extremely compact, rugged, and reliable under a wide range of environmental conditions. Consequently, these memories have seen wide application in militarized equipment. A plated-wire associative array module is shown in Fig. 4.



**Fig. 4.** Plated-wire associative array module (STARAN prototype #2):  
256 words  $\times$  256 bits; 256 processing elements included in module shown.

The economics of plated-wire technology would seem to be favorable because plating is a batch process and the assembly of a plated-wire matrix is more amenable to automation than is the tedious threading of wires through very small cores. However, attaining good uniformity in plating requires exacting process control; in practice, the number of bits per plated wire element is limited by nonuniformities in plating along an element and the number of plated wire rows is limited by variations between

switching characteristics of different elements. The relative economics have favored plated-wire memories for specialized applications only. Access time performance of plated wires is generally better than that for core memories; a factor better than two is achievable. Moreover, the plated-wire memory can utilize a nondestructive readout technique with a read cycle as short as 100 nsec.

Another magnetic memory technology that showed early promise used thin magnetic films deposited on a suitable base; conductor patterns are superimposed. Small element size and the batch nature of the fabrication process favor high performance and low cost. However, problems of material uniformity and production repeatability have led to relative abandonment of thin-film memories. Similar material process problems have likewise militated against monolithic ferrite array memory-exploitation. Rajchman [26] gives an excellent account of these integrated magnetic memory technologies.

### Associative Memories and Magnetic Technology

Because magnetic devices appeared to promise earlier realization of associative memories than other technologies (e.g., cryoelectric memory technology), many such devices have been developed. Consideration has been given to magnetic cores by Kiseda *et al.* [3] and McDermid and Petersen [27], to transfluxors by Lussier and Schneider [28], multibiax cores by Capobianco *et al.* [29], and to multiaperture logic elements by Crowther and Raffel [30] among several magnetic techniques. Due to the high power dissipation and high cost associated with the magnetic approach, the memories are necessarily small and have limited extendability in that they cannot store large data files. A magnetic film approach, which permits an associative memory using nondestructive readout, was reported in 1962 by Joseph and Kaplan [31]. Kaplan [32] estimated that with bi-core thin-film elements, a 10,000 word associative memory is feasible. The Goodyear Aerospace Corp. [16, 33] was funded by Rome Air Development Center (RADC) [AF 30 (602-3549)] to construct a 2,048 word associative processor, the Goodyear Associative Processor (GAP). The GAP, delivered to RADC in 1968, is constructed using what has been termed BILOC magnetic elements to hold each bit. The memory consists of two 1024 word memories of 48 bits per word that share common logic. The processor includes a mask register, a comparand register, and three response stores. Green *et al.* [23] have analyzed the cost of magnetic memory elements and alternative logic for associative processors such as the GAP. These estimates include the relative costs for the basic associative memory array, the logic required to drive the memory, and the response stores required to manipulate the output of a series of search requests submitted to the processor. The cost of the three response stores is approximately 45% of the cost of the GAP; the cost of the basic memory array and electronics is approximately 45%; and the cost of the comparand and mask registers, gates, and controls is 10%. The three response stores are important for efficient search operations and hence it is not possible to decrease the cost of the GAP by cutting down on the response stores from three to one since the processing capability would be decreased significantly.

Other magnetic devices have been used for constructing associative memory arrays.

A memory organization using tunnel diodes was described in 1963 by Fuller [5], an approach using laminated ferrites was noted by Wolff [34] and Crowther and Raffel [30], and solenoid arrays were described by Pick [35].

The STARAN PW associative processor [36] developed by Goodyear Aerospace Corp. and used in 1971 in an air traffic control application for the Federal Aviation Administration in Knoxville, Tennessee was constructed using plated-wire technology. The STARAN PW associative processor was used as an experiment, for a short period of time, in an operational mode at Knoxville. No quantitative data were available to assess the use of the processor at the time that this article was written. The FAA was gathering data and was expected to publish a report on the processor. All reports seemed to indicate that the experiment was successful.

### Cryoelectric Technology

#### *Basic Cryoelectric Technology*

Another memory technology which saw early promise and a potential for relatively inexpensive realization of large memory arrays is that based upon superconductivity. At extremely low temperatures, certain materials (super-conductors) have zero electrical resistance and permit persistent current loops to be established; the presence or absence of a persistent current can then be assigned as the zero or the one state of a binary memory cell. State switching can be achieved based on the phenomenon whereby the critical temperature for superconductivity depends upon an imposed magnetic field. A suitably arranged current following adjacent to a superconductive loop can, by its induced magnetic field, change the critical temperature above that of the element environment, thus causing any persistent current in the loop to collapse.

Practical realization of superconductive memories has not developed, in large part because of material process problems of uniformity and reproducibility in the production of large arrays. The requirement for ultralow temperature refrigeration has been a secondary hindrance. Nonetheless, Rajchman [26] has forecast a promising future for superconductive memories of  $10^7$  to  $10^9$  bits capability with microsecond access to randomly-addressed words.

#### *Cryoelectrics and Associative Technology*

Work in cryoelectrics was perhaps the first area investigated for associative memories. Early investigators such as Slade and McMahon [2], Slade [37], Seeber [38-41], Seeber and Lindquist [42, 43], Newhouse and Fruin [7, 8], Davies [44, 45], and Mann and Rogers [46] discussed various implementations of cryoelectric memories. The latter reported upon experimental work in which small arrays of cells were utilized, and described a bit logic for "between limits" retrieval. As more complicated operations are specified, the number of cryotrons per cell become large.

The cryogenic associative memory cell developed by Newhouse and Fruin (also

see the section above titled Associative Processor Components) is shown in Fig. 5. As noted by Murtha [25]:

The associative memory system constructed with these cells would have several modes of operation:

1. Comparison—the contents of all words in the memory can be compared simultaneously. This comparison can be made on any subset of bits of each word; any bits not pertinent to a particular comparison are masked out.
2. Read—the read operation must always be preceded by a comparison operation. If the comparison finds a matching word, the read operation places this word in an input/output register.
3. Write—this operation also must be preceded by a comparison operation. Any subset of any word can be written by appropriate masking.
4. Comparison bypass—this operation is used (in the case of multiple matching words) to exclude the first matching word after it has been dealt with.

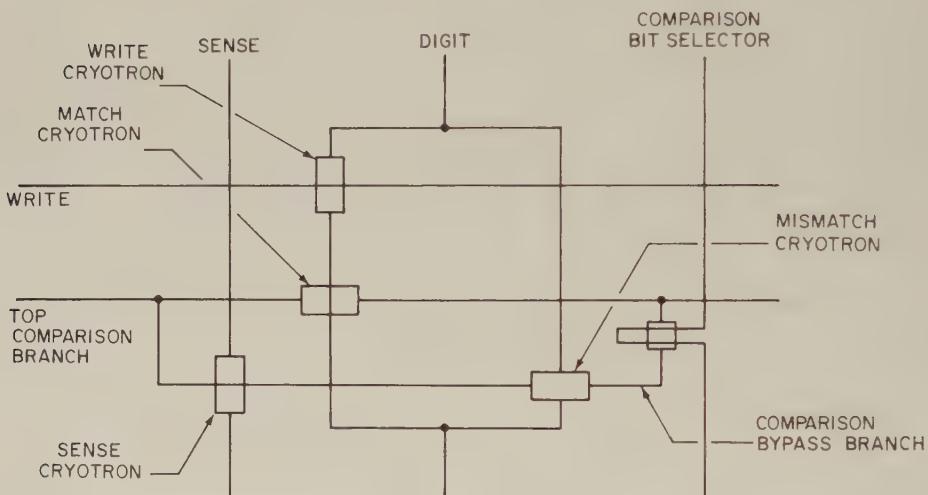


Fig. 5. Cryogenic associative memory cell.

Although technological developments in evaporation and insulation of films promised the realization of larger arrays than those envisioned by Newhouse and Fruin, no such developments have taken place. Ahrons and Burns [47, 48] described the development stages in applying superconductive devices and reviewed superconductive memories. Burns [49, 50] was developing a large superconductive random access memory in 1965 using batch fabrication techniques based on the RCA superconductive continuous sheet memory [51]. The approach showed promise for random access memories in the range of  $10^7$  bits and potentiality for a  $10^9$ -bit memory. However, no such memory had been reported at the time this article was written.

## Semiconductor Technology

### *Basic Semiconductor Technology*

The memory technology which saw the most dramatic growth in the late 1960s and early 1970s was that employing integrated semiconductor devices. The first impact of semiconductor memories was in the application of small capacity, quick access memories such as for "scratch-pad" storage, buffer storage, and general registers in third generation computer systems. For small capacity memories, core arrays lose their cost advantage because the smaller the array, the higher the ratio of address decoding and selection as well as sensing electronics to the number of core elements. Consequently, once the technology of integrated semiconductors became reliable, there was a natural market in which this technology could be cost competitive by virtue of production economies, at the same time offering greatly enhanced speed performance relative to ferrite core technology.

The significance of semiconductors as an emerging technology is indicated by Rajchman [26]. Koehler [52] provides an appraisal of semiconductor versus core technology in 1971. The trend was toward larger and larger semiconductor device arrays, increasingly cost competitive with core arrays for main computer memory applications. In particular, the metal-oxide-semiconductor (MOS) technology was most widely utilized because of its greater compactness, lower power consumption, and simpler fabrication as compared to integrated bipolar transistor technology. So-called current mode logic (CML) technology appeared to be significant and was expected to be used heavily.

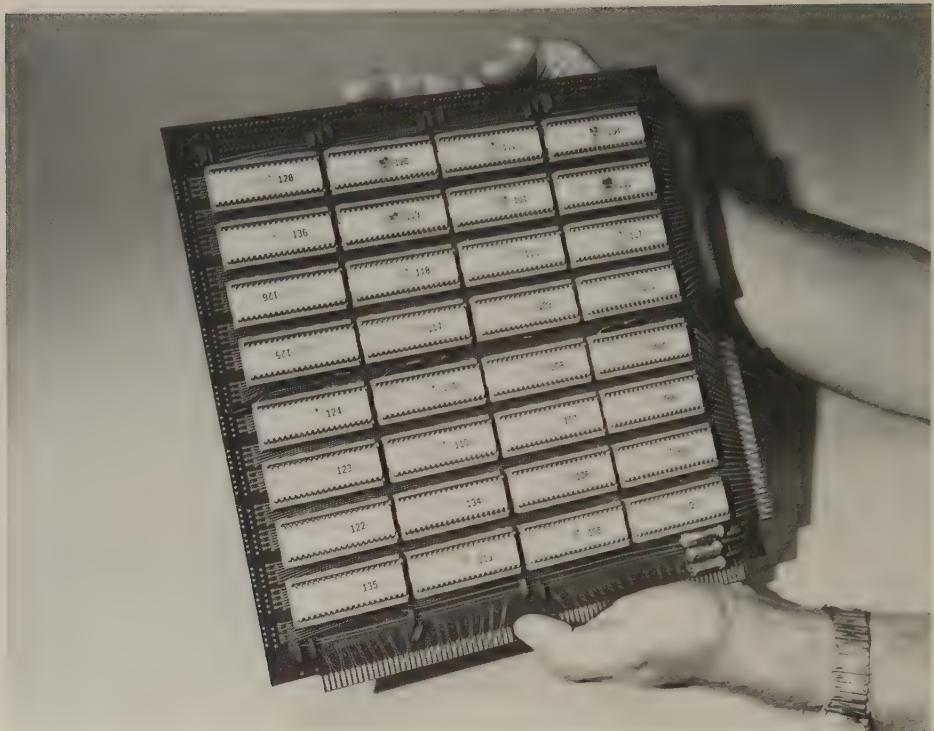
### *Associative Memories and Semiconductor Technology*

Semiconductor associative memories are very promising as a consequence of developments that have taken place in the integrated circuits and with the advent of the MOS field-effect technology. It already started to make an impact in 1971. The Goodyear Aerospace Corp. moved away from magnetic techniques with their STARAN S associative processor (see the section below titled A State-of-the-Art of Associative Processors in 1971) which is based on semiconductor MOS technology.

Silicon-on-saphire, complementary MOS circuits have also been developed for associative memories as reported by Burns and Scott [53]. Igarashi and Yaita [54] reported on an integrated MOS transistor associative memory with a 100 nsec cycle time.

The Honeywell Information System (HIS) associative memory (see Berg and Thurber [55], Berg and Johnson [56], Wald [57, 58], and Cochrane *et al.* [22]) is based on the integrated circuit MOS chips developed by the Texas Instruments Corp. The HIS memory was one of the first associative memories built that used MOS technology. Figure 6 shows the HIS memory plane.

The HIS memory was intended for use in a multiprocessor space-borne computer configuration to perform scheduling and executive functions. The memory represents the state-of-the-art in associative memory technology in 1971. It is estimated to cost



**Fig. 6.** Honeywell Information System MOS memory plane. The memory plane shown is a typical MOS memory plane. Each plane, a four-layer printed circuit card, contains 64 words of 64 bits in length.

\$150,000 for the memory and its associated logic on a custom-built basis. Figure 7 shows the entire HIS associative memory.

The Honeywell Information System's MOS associative cell is capable of storing a single bit of information and of performing the exclusive-or-logic function. Each cell consists of:

1. Ten transistors.
  2. Two pins in the bit direction and one pin in the word direction.
  3. Less than 100  $\mu$ W of standby power.
  4. Less than 750  $\mu$ W of operative power.
  5. High speed (approximately 1  $\mu$ sec).

The associative array of 256 words of 64 bits consists of a number of chips. Each chip consists of 218 bits. A photomicrograph of a chip is shown in Fig. 8. The organization of a chip is shown in Fig. 9. A chip is  $0.13 \times 0.11$  in., and although intended to achieve a speed of 300 nsec cycle time and 200 nsec access time, it did not meet this objective and operated at approximately 1  $\mu$ sec.

A typical associative memory organization is shown in Fig. 10. Although the largest memory that Honeywell had built was 256 words by 64 bits, the system was designed for an array of 512 words by 104 bits.



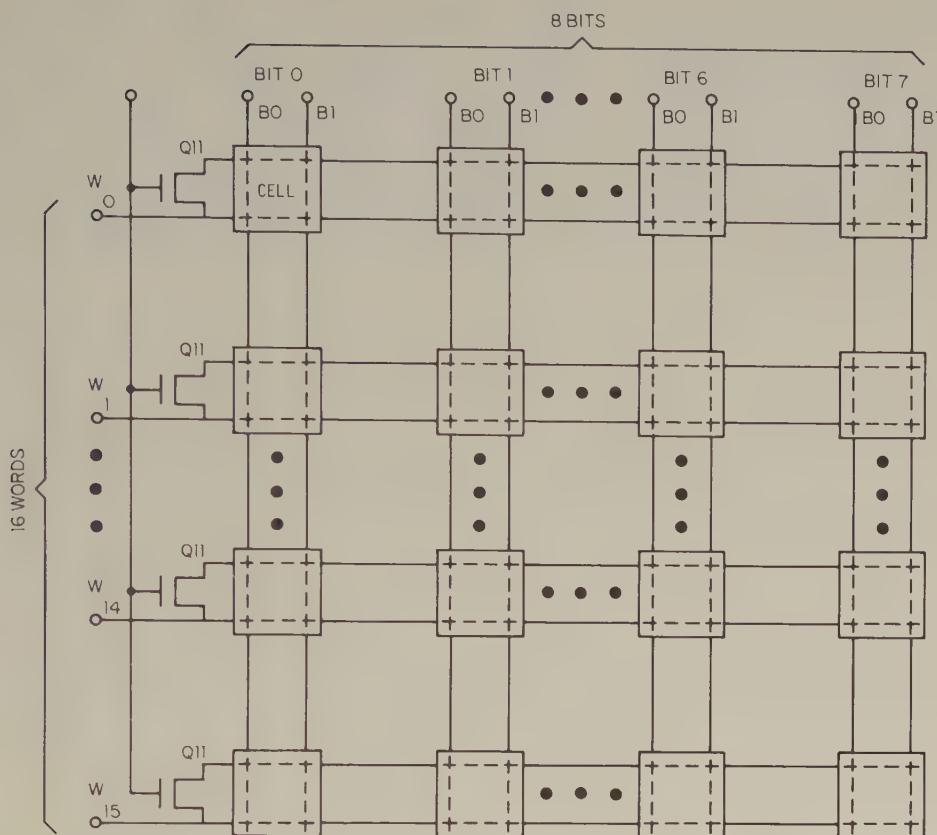
**Fig. 7.** Honeywell Information System associative memory. This is a photograph of the entire associative memory with the memory planes in the rear (in the bottom one-third), and therefore they cannot be seen. The lower third of the rack comprises the basic system with the word logic and registers on the bottom, the word MOS/TTL interface next, and the control unit at the top of this section. The control plane, which allows complete demonstration or self-test of the unit, is next above with the power supplies on top.

The associative cell is based on distributive logic at the bit position, and hence a parallel search may be performed on all bit positions at one time, rather than in a bit-serial fashion. The associative cell has the following capabilities:

1. Parallel write and read operation.
2. Two dimensional write and read.
3. Nondestructive read.
4. Parallel maskable equality search.



**Fig. 8.** A photomicrograph of the basic MOS associative memory array which is organized as 16 words, each of 8 bits. The 128 memory cells, each comprised of ten MOS transistors, require a total of 1296 transistors. This equivalent of several hundred gates is contained in a chip approximately  $0.11 \times 0.13$  in.

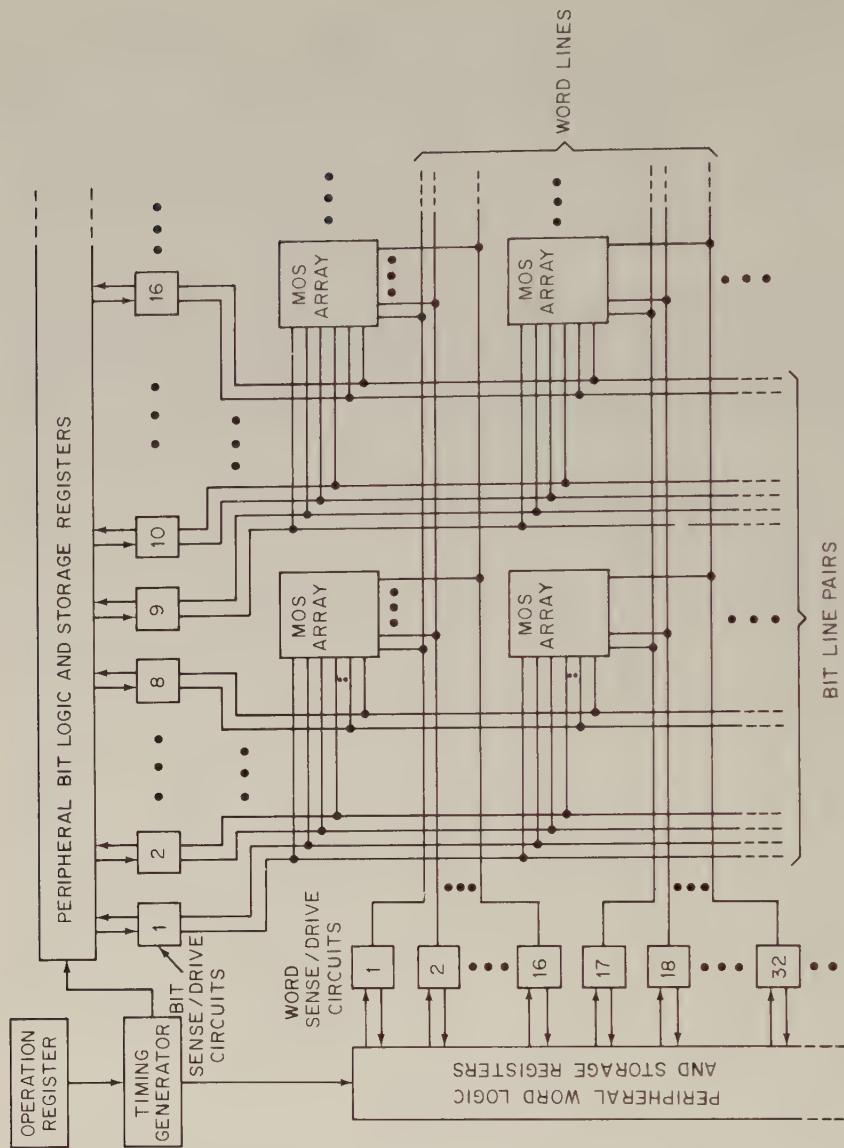


**Fig. 9.** Organization of an MOS chip.

The entire response store can be written to any bit slice. The memory has no arithmetic capability.

### Other Associative Memory Technologies

Holography promises to be an interesting technology with impressive improvements in storage density possible. A hologram is a recording of the interference pattern formed when light from an object illuminated by the coherent light of a laser is mixed with unmodulated light from the same laser. In digital holographic recording the hologram of a mosaic of light valves is formed on an area of  $1 \text{ mm}^2$  or less. Because surface imperfections affect only the signal-to-noise ratio of the entire image, and not specific bits, and because a stationary medium is used (eliminating errors due to vibration, eccentricity, etc.), the full theoretical resolution of the medium can be approached. Holographic storage devices are still undergoing research and are not yet imminent.



**Fig. 10.** Typical MOS associative memory organization.

Storage of  $10^{10}$  bits in pages of  $10^5$  bits, accessible in microseconds, appears to be feasible [22].

Sakaguchi *et al.* [59] report a preliminary experiment of a holographic associative memory system that used a thin hologram of 10 words by 10 interrogation bits by 20 information bits. The experiment was carried out and confirmed high signal-to-noise ratio in the interrogation and readout operations, associative translation function between different information, and high reliability in the memory operation.

Another technology that has been used for associative memories is glass delay lines. Rux [60-62] has used this technology for an associative memory.

### Parallel and Associative Processors

A number of developments concerning associative processors and a closely related subject, parallel processors, are discussed in this section. It is not possible to describe parallel processor technology in detail in this article; however, some of the relevant work in parallel processors is noted.

#### *Associative Processors*

One of the earliest associative processor designs was specified by Estrin [63, 64] who developed the concept of a fixed-plus a variable memory. It was shown that the processor would be useful for a number of application areas.

A number of other designers have considered associative processors. One development in this regard is the Associative Store Processor (ASP), described by Savitt, Love, and Troop [65-71]. The ASP machine organization was to provide the parallel search facilities of an associative memory plus intercell communication. The dominant element in the ASP machine organization was to be the "context-addressed memory." This memory was to store both data and programs and was intended to provide the ability to identify, in parallel, unknown items by specifying the context of relations in which the unknowns appeared. A relatively small READ-ONLY memory was to be employed to store a microprogram for executing ASP instructions. However, the ASP machine organization never reached a hardware implementation state. An interpreter was implemented for the IBM 360 family of computers to simulate a portion of the ASP system (as reported by Savitt *et al.* [70, 71]).

Kisylia [72] proposed an associative memory processor with distributed logic. His work was based upon the organization proposed by Lee and Paull [73]. The modification made by Kisylia was in the construction of the memory cell and in the various modes of local communication each cell enjoyed with its neighbors. Processing was to take place simultaneously on three distinct levels in the machine. The hardware organization has not been implemented. Others who have worked on distributive logic associative memories are Crane and Githens [74], Lee [75, 76], and Lipovski [77, 78].

Knapp [79] described work sponsored by the Rome Air Development Center

(RADC) in associative memory technology. In 1967 RADC was sponsoring the following hardware associative memory developments: an Associative List Selector (ALS) that implemented a fast hash-addressing scheme was under development by Good-year Aerospace Corp. [80]; a 120-bit 2,350 cryotron associative processor plane was being fabricated by Texas Instrument [81-83]; a breadboard model of an associative memory that used ferroelectric and photoconductor elements had been developed by the Marquardt Corp. [84-86]; and techniques for fabricating distributive logic and memory networks using monolithic chips, wafers, and individual miniature component elements were under development by the Westinghouse Aerospace Division [87].

All of the above hardware organizations permitted both the reading and the writing of data from and to an associative memory. READ-ONLY or fixed or semipermanent stores have also been used as content-addressable memories. Goldberg and Green [88] reported upon a fixed content-addressable memory that involved the use of permanent wiring of information in an array of linear cores. Lewin *et al.* [89, 90] developed a fixed content-addressable memory where electronic punched cards were used as the storage medium. Lewin [91] has written a thorough survey of READ ONLY memories for both content-addressable and random-access memories. READ ONLY memories are useful when one is storing encyclopedic information not subject to change. However, they are not useful when the data base is subject to change.

### *Parallel Processors*

The underlying philosophy in parallel processors involves the exploitation of orderliness, parallelism, and repetition inherent in processing tasks. Two basic approaches are represented.

As noted by Cochrane, Minker, Sable, and Roberts [22]:

One approach is to apply a sequence of operations to a parallel stream of operands. Once an instruction is fetched and decoded, it may be broadcast to several parallel processing elements each having a memory and registers for its own operands. This is called *array processing* and is exemplified by an ILLIAC IV array, with its control unit and 256 processing elements. The merit of this approach is the tremendous gain in effective execution rate possible whenever this data parallelism or simultaneity can be achieved in the problem. The drawbacks are the problems of economic use of all array elements, and the data movement necessary to set up the computation.

The other approach is that called *vector processing*, using the pipeline concept. In effect it is an assembly line for the operands undergoing the execution of an instruction. An instruction, such as a floating addition which is a natural functional primitive for the analyst and programmer, is not so to the hardware. The arithmetic unit may view it as several steps, e.g., (1) exponent subtract, (2) align, (3) add, and (4) normalize. Gain is achieved by implementing each step as a separate hardware stage, and routing the operands of an operation from stage to stage. In this way, separate resources may contribute simultaneously to the progress of several instruction steps. Some gain is achieved with a random sequence of instructions if the hardware partitioning into primitives is chosen for statistically well-balanced loading. Ideal operation occurs in the repetition of an instruction on well-ordered data, since the "pipe" is kept full, with each stage usefully occupied on every cycle on behalf of a separate instance of execution.

of the instruction. In the example cited above, four floating additions could proceed in the time that just one would require in conventional machines. One advantage of the pipeline approach is that programming and data management do not become the problems they promise to be in array processing. A drawback is that the gain achievable by this method is limited by the fact that these hardware primitives cannot be broken down further to achieve more parallelism, while the parallelism realizable by array processor architecture is conceptually unlimited.

These advances in processor architecture are largely attempts to exploit properties of scientific computation which lend themselves to enhancement by hardware organizations. Mathematical computation tends toward high competition of similar operations on a single datum (e.g., polynomial evaluation), or extrapolation of coordinates for many objects, etc. Regularity, orderliness, and pattern are the rule both for the method and the data, whereas in other applications of data processing these properties are present to a much lesser degree, and are often difficult to represent.

Even in the most advanced conventional architectures, such as the CDC 7600 or the IBM 360/195, the scratch pad memory fully realizes its intended use only in the execution of tight loops, prevalent only in mathematical applications.

The ILLIAC IV [92-96], which is an outgrowth of the Solomon Computer [97-100], provides extensive parallel processing capability. The ILLIAC IV was nearing completion in early 1972 and was implemented for the University of Illinois by the Burroughs Corporation. Daniel Slotnick was a primary designer in both the Solomon and ILLIAC IV computers. As noted in a Burroughs Corporation Manual [92], "The nucleus of the system is the ILLIAC IV array, a matrix of 256 identical processing units, configured into four identical quadrants, each having 64 processing units under the direction of a common control unit. These array elements perform the computational tasks for the system." Each of the 256 processing elements has 2048 words of 64 bit memory. Associative processing is accomplished by performing search operations when the elements of the search field are in the same relative position in each processor. The cycle time is 240 nsec. The processing elements also have a 32-bit mode where each quadrant would be considered as 128 parallel operating units. The processing elements can communicate data to and from neighboring processing elements by means of routing instructions.

The complete ILLIAC IV system for the University of Illinois includes a B6500 computer to perform input/output operations and compilation, and an operating system to control programs. A  $10^9$  bit, head-per-track disk file with a 40 msec rotation speed provides an effective transfer rate of  $10^9$  bits/sec. The  $10^9$  bits are comprised of a dual system, each with  $5 \times 10^8$  bits capacity. Westlund [101] developed a timing simulator for the ILLIAC IV. The simulator, written in ALGOL, is implemented on the Burroughs B5500. Kuck [102] has described the ILLIAC IV software and application programming effort.

A book edited by Hobbs *et al.* [103] contains a number of articles that describe developments in parallel and associative processor technology as of 1969. Some of the processors noted were the Sanders Orthogonal Processor [104] that incorporates aspects of both parallel and associative processing where the orthogonal memory has both so-called "horizontal" and "vertical" access modes. The Control Data Corp.'s STAR computer is another such configuration. The STAR configuration consists of a central processor and up to 12 peripheral processors. Main memory could contain up

to 1 million 64-bit words. The central processor, containing a pipeline arithmetic unit that permits operations upon a string of operands from memory, and stores corresponding string results, achieves a computational gain whenever the same operation may be performed upon many of the elements of such a string. The Texas Instrument Advanced Scientific Computer (ASC) is another example of a vector processor that uses the pipeline concept for high-speed operations on streams of well-ordered data [22, 105].

Hobbs and Theis [168] provide an interesting comparison and categorization of parallel processor types. Four categories are recognized: multicomputers and multiprocessors, associative processors, parallel networks of array processors, and functional machines.

Murtha [25] wrote a comprehensive and thorough article on highly parallel processing systems in which he also described associative memories and associative processors. Both hardware and software technical details of a large number of parallel processors were covered up to 1966.

## ASSOCIATIVE MEMORY ORGANIZATIONS IN COMPUTER SYSTEMS

In assessing associative memories and processors it is important to consider them in the context of the computer system in which they will reside and for a particular problem rather than as an abstract entity. There are several configurations that have been considered for interfacing an associative memory in a system.

Some of the interface organizations are as follows:

1. *Direct Memory Access (DMA).* A direct memory access (DMA) channel to the host computer memory allows fast interchange of data between systems in the common memory. It permits the associative control memory to be accessible to the host computer. The STARAN S associative processor permits such an organization (see the section below titled A State-of-the-Art of Associative Processors). The Goodyear Associative Processor (GAP) was integrated with the CDC 1604B in 1968 through a DMA channel. A thorough investigation of the GAP configuration in Green *et al.* [23, 106] showed that the "multi-processing of GAP in this configuration had major shortcomings. The overhead needed to set up, control, coordinate and synchronize GAP is very high and is not warranted."
2. *Integrated With Core Memory.* A portion of core memory is replaced by associative memory. This configuration, investigated by Green *et al.* [23, 106], permitted data to be transferred from a peripheral device directly to the associative memory. It also permitted both associative and nonassociative operations, and the ability to move data rapidly between random access core and associative memory.
3. *Buffered Input/Output (BIO).* A buffered input/output channel provides a slower means of exchanging data than does the DMA. In addition to being able to transfer blocks of data between the associative and host memories, different types of peripherals may be tied into the associative control memory through a BIO. The STARAN S processor permits a buffered input/output.

4. *Parallel Input/Output (PIO).* In a parallel input/output interface mode, each associative array of  $m$  words can have up to  $m$  inputs and  $m$  outputs fed into the array, which may be accomplished by loading the response store for the array. This allows any device to communicate directly with the associative array. In particular, the outputs from a multihread disk or drum may be so interconnected with the processor. Hence data stored in  $m$  tracks in a disk sector may be transferred directly to the associative array. The STARAN S was to be integrated with a disk in this manner in 1972.

Although the above are the dominant interface options, special purpose uses of associative memories have integrated the memory in other ways. Bird, Cass, and Fuller [107, 108] used an associative memory, with little logic, as a special purpose device to act as a filter between disk and the central processor. The device was intended to speed the retrieval of fixed-formatted data from disk. Green *et al.* [23, 106] proposed that the associative memory be used to coordinate, control, and optimize search operations employing peripheral devices and thereby to assist the overall computation process by decreasing the imbalance between memory speeds. Requests for I/O would be sent to the associative memory whose function would be to issue commands for the accessing of data or peripheral store. Hence the associative memory would take over the optimization function of the Executive Control system in the input/output area.

## A STATE-OF-THE-ART OF ASSOCIATIVE PROCESSORS IN 1971—THE STARAN S

The STARAN S associative processor [17] is described in this section as it is representative of the state-of-the-art in 1971.

### Introduction

The Goodyear Aerospace Corp.'s STARAN S associative processor was announced as commercially available in 1971. The "S" denotes that the associative memory is constructed from solid-state elements.

The STARAN S processors were in production during 1971 with the first unit scheduled to be available by the end of March 1972. The cost of a basic STARAN S system with two associative arrays is approximately \$500,000. Goodyear has recommended that two associative arrays be purchased as a minimum. If the associative processor is to be interfaced with a conventional processor, an interface hardware device will be required. The cost of the interface will depend upon the central processor employed.

### Description of the STARAN S Organization

#### *Architecture*

The architecture of the basic STARAN S organization is shown in Fig. 11. The elements of the associative processor are:

1. Associative arrays that consist of modules of 256 words  $\times$  256 bits. Up to 32 associative arrays can be included in a STARAN S configuration.
2. Associative processor (AP) control that performs data manipulation within associative arrays.
3. Associative processor control memory that stores AP control instructions and acts as a buffer between AP control and other system elements.
4. Program pager that moves program segments into high-speed memories.
5. Sequential controller and memory that performs maintenance, controls peripherals, and provides operator interface.
6. External function logic that transfers control information among STARAN elements.

Of these, the associative arrays and the AP control are the two most important elements of the processor. A brief description for the above six elements follows.

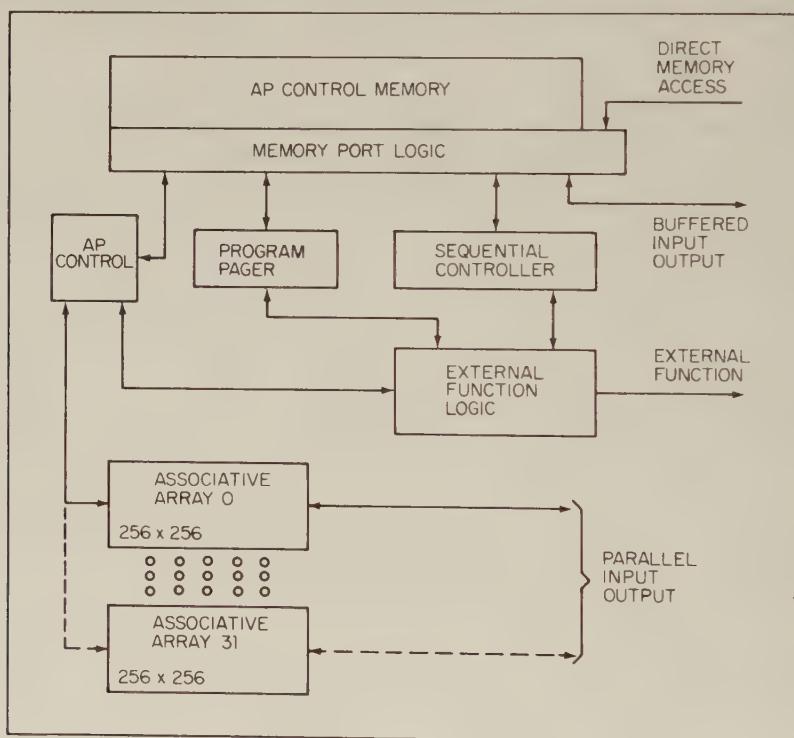


Fig. 11. STARAN functional flow.

**Associative Arrays.** The associative array consists of modules of memory elements. Each array consists of a 256 word  $\times$  256 bits memory unit with a response store of 3 bits per word and 256 words. The response store is activated and used to store the results of search operations. The modularity of the design prevents communication between memory elements. Thus a shift of the response store down by one bit results in an end-around of the last bit into the same module.

Either a bit slice or a word may be accessed. Arrays can be selected and operated in parallel or one at a time. The memory has a parallel bit write capability and a parallel arithmetic capability. An interesting feature is the ability to write or to read a word or a bit slice to and from response store.

**Associative Processor Control.** The AP control can perform parallel searches on data in one or more associative arrays. The AP control unit sends the same message to each array that is being accessed. The basic system called for one AP control unit. In 1971 Goodyear was designing a configuration with two AP control units. Each control unit is to be able to operate on the arrays simultaneously to effect a multi-processor configuration. Thus one AP control unit might be writing while the other AP control unit might be reading. Responsibility to effectively take advantage of this multi-processor feature is placed with the programmer.

**AP Control Memory.** The comparand register for a basic instruction is 32 bits. Thus, since the memory contains 256 bits in each word, to interrogate the entire memory array requires that a sequence of eight operations be performed across the entire memory.

The comparand register is not masked. To account for this important associative feature of being able to access any one bit or arbitrary sequence of bits, there is a restricted capability of having field pointers together with field length counters. There is a maximum of three field pointers and two field length counters available. The field pointers contain bit or word slice addresses that are to be operated upon; the field length counters define field sizes.

The AP control memory is a sequential memory separate from the associative memory. The three page memories shown in Fig. 11 use solid-state elements and have cycle times of less than 200 nsec. The high-speed data buffer also uses solid-state elements and permits rapid loading from external devices. The bulk core memory uses nonvolatile core storage with a cycle time of 1  $\mu$ sec. A page of data or program is transferred from the bulk core memory to one of the two pagers in 512  $\mu$ sec. It is possible to load one of the two pagers while executing the other pager.

**Program Pager.** The program pager loads words from the bulk memory, high-speed data buffer, or DMA channel into any one of the three page memories. Operation of the program pager is under program control.

**Sequential Controller.** The sequential controller is effectively a conventional computer system and consists of 8172 words of conventional memory, a keyboard printer,

a perforated tape reader/punch unit, and interface logic to connect the sequential controller to other STARAN S elements.

The controller provides capabilities for assembling and debugging STARAN S programs, the ability to initially load the AP control memory, and to communicate with the operator. Thus the STARAN S is not dependent upon a host processor to assemble and debug programs.

**External Function Logic.** The external function (EXF) logic facilitates coordination between the different elements of STARAN S for special functions and simplifies housekeeping, maintenance, and test functions.

By issuing external function codes to the EXF logic, elements of STARAN S can control and interrogate the status of other elements. Function codes can be transmitted to EXF logic by AP control, the program pager, sequential controller, and the host computer.

A summary of the standard configuration and hardware options for the STARAN S is given in Table 1.

**TABLE 1**  
Hardware Summary

Standard configuration	Options
1 Associative array	Up to 31 additional
1 AP control <sup>a</sup>	
15 Interrupts	Additional interrupts
1 AP control memory <sup>a</sup>	
3 512-word page memories	3 1024-word page memories
1 512-word BSBD	1 1024-word BSBD
1 16,384-word bulk core	1 32,768-word bulk core 1 block of 30,720 DMA, external- memory, address words
1 Program pager	
1 Sequential controller 8192-word core memory	Up to 20,480-word core memory
1 Keyboard printer	Additional peripherals
1 Perforated tape reader/punch	
8 Interrupts	Additional interrupts
1 External function logic section	Interface options Direct memory access (DMA) Buffered input/output (BIO) External function (EXF) Parallel input/output (PIO)

<sup>a</sup> Dual control system option available.

### *Interface Options*

Four types of interface options are possible to integrate computer systems and other external devices with the STARAN S. These are: (1) direct memory access (DMA), (2) buffered input/output (BIO), (3) external function (EXF), and (4) parallel input/output (PIO). (See the section above titled *Associative Memory Organizations in Computer Systems*.)

A multihead disk with heads operating simultaneously was to be connected to STARAN S via the PIO. When requesting data from disk, STARAN S sends the disk one or more external functions specifying a starting sector address, the number of sectors, and the direction of transfer. The disk system may interrupt STARAN S when the disk reaches the requested sector to initiate transfer over PIO lines.

The disk was to be interconnected with the STARAN S through the response store. All 256 tracks in a disk sector were to be read in parallel into the response store and transferred from response store to the appropriate bit slice in the associative array. On output from the associative array, a bit slice was to be transferred to the response store and then to the disk. Goodyear had purchased a disk and was integrating the disk with the AP at the time of this article. Demonstrations were scheduled in 1972. Goodyear believed that a tie-in with a disk precluded the need of a large fully associative memory of the order of  $10^{10}$  bits.

AP control instructions that actually read or wrote the PIO are synchronized to the disk so that STARAN S timing can be slaved to the disk timing during transfer.

### *STARAN S Instructions*

STARAN S is a general purpose computer capable of performing search, arithmetic, logic, and input/output operations on many sets of data simultaneously. To compare a bit slice against the argument register takes 0.15  $\mu$ sec for less than, greater than, or exact match with the comparand. An exact match requires 4.8  $\mu$ sec ( $0.15 \times 32$ ) for a 32-bit match. It takes a longer time to match on the full 256 bits in a word. To perform the addition between bits in different fields in the same word requires 85  $\mu$ sec. The capability exists to perform additions between fields in different words. The time to do so takes longer than between fields of the same word and depends upon the distance between the words.

The instruction set consists of some 100 instructions that are of four major types.

1. *Word addressing.* The reference address field of the instruction is (or was) used to address any word in either the high-speed data buffer (HSDB) memory, the direct memory access (DMA), or the bulk core memory.
2. *Immediate addressing.* The reference address field of the instruction word itself contains an operand value.
3. *Data pointer addressing.* The data pointer (DP) register contains the address of any word in either the HSDB memory, the DMA memory, or the bulk core memory.
4. *Array addressing.* The associative array can be referenced by bit-column (bit-slice) number or by word number. The array elements (bit-slice or word) may be referenced via:
  - (a) *Immediate addressing*—The instruction word itself contains an operand value.

- (b) Field pointer addressing—Any of the three field pointer registers contains the address of the array element.

The STARAN S represents an advance in associative processor technology. The use of integrated circuit technology for the memory array, the response store, and logic permit faster speeds than were available on earlier associative processors, and at less cost.

The STARAN S represents engineering compromises. The modularity achieved presents problems for the programmer, or increased cost to fix. In particular, modularizing the array memories to consist of 256 words does not permit communication between response stores since a response store shift in all response stores at once results in an end-around shift within each response store within each array. To achieve a response store operation that ignores array boundaries requires that the programmer devise a code to effect the operation.

## SOFTWARE AND ASSOCIATIVE MEMORIES

Associative memory hardware of interest to programmers became available in approximately 1968 and only in very small quantities. It is therefore not surprising that software for associative memories lags significantly behind hardware developments. In this section, algorithms for associative memories are discussed, as are the simulation of associative memories and areas of potential application.

### Algorithms and Software for Associative Memories

Falkoff [4], using Iverson's notation [109], has developed numerous algorithms for associative memories. He described the underlying structure of parallel-search memories. Algorithms for searches based on equality, maximum, minimum, greater than, less than, nearest to, between limits, and ordering are provided. Estrin and Fuller [110, 111] have also described algorithms for associative memories. They further note that content-addressable memories are more flexible than list structures since, within a list structure, data are ordered with respect to predecessors and successors of elements on a list. However, in a content-addressable memory, data sets are unordered and immediately accessible without following chains in a list. A number of individuals have described algorithms for associative memories. Rogers and Wolinsky [112] have shown how the algorithms may be implemented with cryogenic technology; Rogers [113] has described algorithms for complex search operations; Scheff [114] has described programming techniques for content-addressable memories; and the Goodyear Aerospace Corp. [115] developed notes on various algorithms for associative memories.

A major problem for retrieving data from an associative memory was that of reading out the responses of  $m$  words that all match the interrogation bit configuration. This problem was solved by Lewin [116] in an elegant fashion. He devised an algorithm, referred to as the Lewin algorithm, for performing the complete readout of  $m$  matching

words in  $2m - 1$  cycles. The algorithm requires that two sense outputs be implemented in the hardware for each digit of the word. The Lewin algorithm has been implemented in the associative memory portion of the special purpose Parallel Element Processing Ensemble (PEPE) hardware [22]. The importance of the Lewin algorithm is that the time to extract the multiple matches is proportional to the number of multiple matches, and not to the size of the associative memory store. Wolinsky [117] developed a new proof that the Lewin algorithm is correct and requires  $2m - 1$  cycles for extracting the multiple matches.

Little work exists in the area of general software tools for associative processors. Goodyear [17] developed a basic assembly program called APPLE (Associative Processor Procedure Language) to permit programmers to write in an assembly language for the STARAN S processor. APPLE is one of the first assembly programs written for and assembled by an associative processor. The program was operational in 1971. The Control Data Corporation developed an assembly program that provided the capability to assemble programs for the Goodyear Associative Processor. The assembly program, termed CODAP [16] was implemented on the CDC 1604B, the host computer for the GAP. The assembly program was available in 1968 and was used at Rome Air Development Center. Considerations have also been given to using the Goodyear Associative Processor to construct an associative compiler. Peters [118] concluded that the input/output for memory loads using the GAP was excessive so that the use of a small associative memory for compilation was not effective.

Dugan *et al.* [10] noted that several macro-type instructions are required for dealing with several classes of data in an associative memory. They described a routine called SPO that saves all response stores and status (busy) bits and "deactivates" that associative memory segment not defined by SPO. Other macros noted were NGT (to find the value of a data word from a list that is next greater than the comparand), AND (to find the conjunction of two lists of data words), and an executive routine DISPATCHER, that handled the problem of data loads to the associative memory.

The ease with which one can program an associative memory is subject to question. Dugan *et al.* [10] noted that programming for associative memories was "difficult to learn because associative configurations are conceptually different from other configurations." Green *et al.* [23, 106] expanded upon the implications of associative processing for programmers as reported by Dugan. Goodyear in their advertising literature on STARAN S claimed that programming for associative processors is easier than for conventional processing. The issue can be resolved only with the availability of associative memories.

Dugan also noted that the ability to retain and to manipulate tag bits in a word and to perform indirect addressing or index modifications of associative instructions for associative memory locations are important hardware features required by software personnel.

### Simulations of Associative Memories

Because of the unavailability of associative memories, several investigators simulated the functions of such memories by software techniques. Feldman [119], using a

hash-code to access entries in a table, simulates the exact match capabilities of an associative memory. The data accessed is organized in a ring data structure. His approach is very useful for systems that do not require the full capabilities of an associative memory (i.e., the ability to interrogate the entire memory on any combination of bits within a word, rather than on only an exact match). Rovner [120] modified the hash-code concept to organize the data where it exceeds the capacity of core. In this modification, a two-level data store consisting of core and drum is required. Rovner developed a scheme for a paged software-simulated associative memory. Again, only the exact match is considered. Hilbing [121] also considered a paged associative memory system. Feldman and Rovner [122, 123] developed an associative language called LEAP to facilitate programming their simulated associative memory. LEAP is an extension of ALGOL that includes associations, sets, and a number of auxiliary constructs. LEAP is a family of languages; each language adds a different set of features to the ALGOL base. Forms of LEAP contained matrix operations, property sets, and on-line graphics. Brotherton and Gall [126] described an Associative List Selector (ALS) hardware device with capabilities similar to those of the associative memory that Rovner and Feldman developed through software simulation. The ALS is a hardware implementation of hash addressing to give a fast search capability on equality.

Findler [127] developed a language termed Associative Memory Parallel Processor, AMPPL-II, that is effectively a software implementation of the Goodyear Associative Processor (GAP). Several problems in artificial intelligence have been investigated by Findler [128, 129] using the language.

### Applications of Associative Memories

Associative memories and processors have intrigued numerous investigators. In this section, several applications proposed for use with associative memories are noted. Seeber [38, 39] and also Seeber and Lindquist [42] proposed using the memory to perform sorting operations. Seeber [41] also proposed using the memory for symbol manipulation in general. Unger [130], McCormick and Divilbiss [131], and Yang [132] all proposed that the associative memory be used for pattern recognition; applications to picture processing were proposed by Fuller and Bird [133]; Aoki and Estrin [134] proposed the use of the memory for dynamic programming formulation of control optimization; Bussell [135] proposed the use of such memories for solving partial differential equations; Estrin and Viswanathan [136, 137] used the memory to compute eigen-values and eigenvectors; Katz [138] and Gilmore [139] used the memory for matrix operations; Joseph and Kaplan [31] proposed that the memory be used in radar-track correlation problems (which is the application for which the machine PEPE is to be used); and Thurber [140] and Thurber and Berg [141] suggested the memory for air traffic control, and Goodyear used an associative processor for an air traffic control problem as noted in the section above titled Associative Memories and Magnetic Technology. Bird [142] applied the memory to sonar signal processing; Crane [143] proposed that the memory be used for finding paths in graphs; Minker and Shindle [144] applied an associative memory to problems of substructure searching such as in chemical compounds and to other graph searching problems; Orlando and

Berra [145] used the memory for solving network flow problems; Cheydeur [146, 147] proposed a system termed DIMENSION to be used for document retrieval applications; Goldberg and Green [148] proposed the memory be used for large data files; Hayes [149] suggested that the machine be used for machine translation; and Savitt *et al.* [67, 68] devised the ASP associative processor for relational data searches. Ash and Sibley [150, 151] devised a software system called TRAMP for associative searches required in question-answering systems, and Green *et al.* [106] applied a small associative memory to data retrieval problems and found no advantage for the memory because of the high input/output rate of data to and from associative memory. Numerous areas of application of associative memories were proposed in the early 1960s by Estrin and Fuller [110, 111] and by Griffith [152]. Wolinsky [153] described principles and applications of associative memories.

With the exception of a very limited number of areas, as noted by Hanlon [11], “ . . . the superiority of the content-addressable memory or associative processor is only implied, not proved.” Minker [12] noted that, for a number of applications developed since Hanlon completed his survey, small associative memories did not appear particularly advantageous. Applications of this type include: formatted file problems [106–108], dictionary look-up for translation efforts [154], automatic abstracting problems [155, 156], and for use with compilers [118]. In both Dugan [10] and Bird *et al.* [107] a specific associative processor integrated with a second generation computer environment was postulated. Machine coding was generated in each instance. Although some advantage could be shown for an associative memory configuration, the cost differential and actual time saved would not make the approach worth the effort. Other areas proposed for associative processors since the Minker [12] article include data management problems [157], theorem-proving [158], and computer graphics [159, 160].

When originally conceived, associative memories were thought of as devices to permit the easy manipulation of large masses of data. More recently they have been thought of as devices to aid in the control functions associated with a data processor. In 1965 Chu [161] proposed an associative memory to assist in dynamic storage allocation problems. A small associative memory can be used to keep a map of memory and of program locations to assist in dynamic storage allocation. The IBM 360/67 [14, 15] contains a small, eight-word associative memory which is used for paging operations to translate logical addresses to physical addresses. The idea was first used with the Atlas computer [162]. Green *et al.* [23, 106] noted that a small associative memory might be used to store queued requests for input/output devices such as disks and drums so that the requests could be retrieved in a sequence that would minimize access time or latency time. Prywess [163] made a similar observation. Gunderson *et al.* [164] studied the use of associative memories for other control functions within a multiprocessor computer organization.

In assessing the value of an associative processor for an application, a cost effectiveness evaluation is required. The testing and comparing of isolated test problems should be performed only in the context of a total application. One can generally devise *ad hoc* techniques to improve the processing of an isolated problem on any device. The true test, however, is the effect of performing the processing in the context of the total

system, and the overall cost effectiveness of the approach. Studies of such a nature that use existing associative processors are required to determine the effectiveness of an associative processor for a specific application.

## SUMMARY OF ASSOCIATIVE TECHNOLOGY AS OF 1971

An investigation of associative technology as it exists as of 1971 indicates that plated-wire and large-scale integrated circuit technology made moderate sized associative memories feasible. The plated-wire technology should provide capacities of from 10,000 to 10-million bits at speeds of a fraction of a microsecond. Integrated semiconductor techniques provide speeds on the order of nanoseconds but of smaller capacity. Superconductive memories with capacities of from 10-million to  $10^9$  bits will have speeds of approximately 1  $\mu$ sec. All three technologies are likely to be used in future hardware implementations based on the size and speed of the memory required.

Plated-wire and large-scale integrated circuit associative memories have been constructed. Plated-wire, because of its reliability and ability to be made secure, is likely to find use in military-type applications. Semiconductor MOS technology is significant because of cost reduction and the ability to achieve distributive logic.

Superconductive memories remain in the research stage, and hence associative memories of the order of  $10^9$  bits are not foreseeable within the near future. There appears to be no evidence that the major manufacturers or any of the organizations previously active in associative memory technology are attempting to develop large superconductive associative memories.

Associative memory building blocks are available from a number of sources—Honeywell Information Systems, Goodyear Aerospace Corp., Texas Instruments, and Hughes Aircraft Corp.

Associative memory hardware technology for moderate-sized memories matured during the first two years of the 1970s. The developments at Goodyear and at Honeywell provide this evidence. No comparable progress was made with software for associative processing. With the availability of associative memories and processors, substantive rather than speculative studies can be accomplished.

Because large associative memories are not expected to be available within the foreseeable future, clever techniques are being developed to use moderate-sized memories effectively. Integrating a disk or drum with an associative processor is a particularly useful approach. Although many organizations believe that such an integration will alleviate the need for large associative memories ( $10^9$  bits), technical supporting evidence has not yet been developed to substantiate the claim. Technical problems remain in achieving an effective interface between an associative memory and a disk system. Software implications, such as data organizations for disk to make the approach effective and programs to control the flow of data and subsets of data, remain to be investigated.

Small associative memories have been used by major United States manufacturers (IBM, GE, CDC, Burroughs) and one in France (The Compagnie Internationale pour l'Informatique) for executive control programs in commercial machines [165, 166]. The memories are not available for use outside of the control program. It would appear that small associative scratch-pad memories can be used to advantage by programmers.

Speculative articles continued to be written that describe the utility of associative processors. Candidate problems for which an associative memory may be useful are those exhibiting highly parallel processing needs. However, there has been no demonstration that such memories will be effective for large-scale real-time applications.

Although a number of organizations have developed associative hardware, only Goodyear has made their STARAN S associative processor available commercially. It is significant that IBM, the major computer manufacturer, to all appearances is not pursuing associative memory and associative processor technology actively. Efforts at IBM appear to be restricted to the use of small associative memories with executive control systems and for implementing logic functions [167].

The future of associative memories and associative processors continues to be promising. However, it remains unclear as to when promise will lead to reality. Substantive articles rather than speculations are required. If associative processors become more widely available, as would appear to be the case, programmers will be able to determine their true effectiveness. One is likely to see future computer systems with a variety of memory types available—random access, associative memory, disk and drum, and tape. Programmers will have to devise clever executive routines to take advantage of all such memories, and to manage the allocation and use of the memories.

### ACKNOWLEDGMENTS

The author would like to express his appreciation to Mr. Wayne Brubaker of the Goodyear Aerospace Corp. who provided the photographs of the Goodyear equipment for the article; to Mr. Dale Gunderson and Mr. L. Wald of Honeywell Information Systems who provided photographs of the Honeywell associative memory. He would also like to thank Mr. A. E. Roberts of the Auerbach Corp. for his assistance with some of the material in the article, and Dr. Chan Park of the University of Maryland for his comments concerning the article.

### REFERENCES

1. A. E. Slade, A discussion of associative memories from a device point of view, in *American Documentation Institute 27th Annual Meeting*, 1964.
2. A. E. Slade and H. O. McMahon, The cryotron catalog memory system, in *Proceedings of the 1957 Eastern Joint Computer Conference*, Vol. 10, 1957, pp. 115-120.
3. J. R. Kiseda, H. E. Peterson, W. E. Seelbach, and M. Teig. A magnetic associative memory, *IBM J. Res. Develop.* **5** 106-121 (April 1961).
4. A. D. Falkoff, Algorithms for parallel search memories, *J. Assoc. Comput. Mach.* **9**(4), 488-511 (October 1962).

5. R. H. Fuller, *Content-Addressable Memory Systems*, UCLA Rept. No. 63-25, Contract No. NR-233(52), June 1963.
6. R. H. Fuller, Content-Addressable Memory Systems, Ph.D. Dissertation, UCLA, 1963.
7. V. L. Newhouse and R. E. Fruin, A cryogenic data addressed memory, in *Proceedings of the American Federation of Information Processing Societies, 1962 Summer Joint Computer Conference*, Vol. 21, 1962, pp. 89-100.
8. V. L. Newhouse and R. E. Fruin, Data addressed memory using thin-film cryotrons, *Electronics*, pp. 31-36 (May 1962).
9. Joint Technical Committee on Terminology, *IFIP-ICC Vocabulary of Information Processing*, North Holland, Amsterdam, 1966.
10. J. A. Dugan, R. S. Green, J. Minker, and W. E. Shindle, A study of the utility of a hybrid associative memory processor, in *Proceedings of the Association for Computing Machinery 21st National Conference*, 1966, pp. 347-360.
11. A. G. Hanlon, Content-addressable and associative memory systems—A survey, *IEEE Trans. Electron. Comput.*, pp. 509-521 (August 1966).
12. J. Minker, An overview of associative memory or content-addressable memory systems and a KWIC index to the literature: 1956-1970, *Comput. Rev.* 12(10), 453-504 (October 1971); also University of Maryland, TR-157.
13. H. Lorin, *Parallelism in Hardware and Software: Real and Apparent Concurrency*, Prentice-Hall, Englewood Cliffs, N.J., 1972, p. 508.
14. International Business Machines Corp., *IBM System 360 Model 67, Functional Characteristics*, Form A27-2719.
15. International Business Machines Corp. *IBM System 360 Time-Sharing System Concepts and Facilities*, Form C28-2003.
16. D. Brotherton and S. Domchick, *Preliminary Programming Manual For RADC 2048 Word Associative Memory*, Goodyear Aerospace Corp., January 1966, GER-12318.
17. Goodyear Aerospace Corp. *STARAN—A New Way of Thinking*, Document No. 8484 A, October 1971.
18. T. Maguire, Superconductive computers—Commonplace in ten years? *Electronics* 34, 45-51 (November 1961).
19. J. A. Rajchman, Computer memories—A survey of the state-of-the-art, *Proc. IEEE* 49, 104-127 (January 1961).
20. J. A. Githens, An associative, highly-parallel computer for radar data processing, in *Parallel Processor Systems, Technologies, and Applications* (L. C. Hobbs, D. J. Theis, J. Trimble, H. Titus, and I. Highberg, eds.), Spartan (1970), Chap. 3, pp. 71-86.
21. J. A. Githens, A fully parallel computer for radar data processing, *NAECON '70 Record*, pp. 290-297 (May 1970).
22. J. Cochrane, J. Minker, J. D. Sable, and A. E. Roberts, *Analysis of Computer Technology*, Final Rept. AUER-1920-100-TR-1, Auerbach Corp., March 1, 1972.
23. R. S. Green, J. Minker, and W. E. Shindle, *Analysis of Small Associative Memories for Data Storage and Retrieval Systems. Volume II, Technical Discussion*, Final Rept. AD-489 661, Auerbach Corp., July 1966.
24. C. Y. Lee, Intercommunicating cells, basis for a distributed logic computer, in *Proceedings of the American Federation of Information Processing Societies, 1962 Fall Joint Computer Conference*, Vol. 22, 1962, pp. 130-136.
25. J. C. Murtha, Highly parallel information processing systems, in *Advances In Computers* (Alt and Rubinoff, eds.), Academic, New York, 1966.
26. J. A. Rajchman, Computer memories, in *Computers and Their Role in the Physical Sciences* (S. Fernbach and A. Taub, eds.), Gordon and Breach, New York, 1970, Chap. 5.
27. W. I. McDermid and J. E. Petersen, A magnetic associative memory system, *IBM J. Res. Develop.* 1, 59-62 (January 1961).
28. R. R. Lussier and R. P. Schneider, All magnetic content-addressed memory, *Electron. Ind.*, pp. 92-98 (March 1963).
29. J. A. Capobianco, J. E. McAteer, and R. L. Kippel, Associative memory system implementation

- and characteristics, in *Proceedings of the American Federation of Information Processing Societies, 1964 Fall Joint Computer Conference*, Vol. 26, 1964, pp. 27-29.
30. T. S. Crowther and J. T. Raffel, A proposal for an associative memory using magnetic films, *IEEE Trans. EC-13* 5, 611 (October 1964).
  31. E. C. Joseph and A. Kaplan, Target track correlation with a search memory, in *Proceedings of the 6th National MIL-E-CON*, June 1962, pp. 255-261.
  32. A. Kaplan, A search memory subsystem for a general purpose computer, in *Proceedings of the American Federation of Information Processing Societies, 1963 Fall Joint Computer Conference*, Vol. 24, 1963, pp. 193-200.
  33. R. G. Gall, A hardware integrated general purpose computer search memory, in *Proceedings of the American Federation of Information Processing Societies, 1964 Fall Joint Computer Conference*, Vol. 26, 1964, pp. 159-173.
  34. M. F. Wolff, What's new in computer memories, *Electronics*, pp. 35-39 (November 1963).
  35. G. G. Pick, A semipermanent memory utilizing correlation addressing, in *Proceedings of the American Federation of Information Processing Societies, 1964 Fall Joint Computer Conference*, Vol. 26, 1964, pp. 107-121.
  36. Goodyear Aerospace Corp. *The Associative Processor—A New Computer Resource*, Report GER-14087, 21, February 1969.
  37. A. E. Slade, The woven cryotron memory, in *Proceedings of the International Symposium on Theory of Switchings*, Harvard Univ. Press, Cambridge, Mass., 1959, pp. 326-333.
  38. R. R. Seeber, Associative self-sorting memory, in *Proceedings of the 1960 Eastern Joint Computer Conference*, 1960, pp. 179-188.
  39. R. R. Seeber, Cryogenic associative memory, in *National Conference of Association for Computing Machinery*, August 1960.
  40. R. R. Seeber, *Associative Self-Sorting Memory Revised*, IBM Data Systems, TR-00, 756, November 1960.
  41. R. R. Seeber, Symbol manipulation with an associative memory, in *Proceedings of the 16th National Association for Computing Machinery Meeting*, September 1961.
  42. R. R. Seeber and A. B. Lindquist, Associative memory with ordered retrieval, *IBM J. Res. Develop.* 6(1), 126-136 (January 1962).
  43. R. R. Seeber and A. B. Lindquist, Associative logic for highly parallel systems, in *Proceedings of the American Federation of Information Processing Societies, 1963 Fall Joint Computer Conference*, Vol. 24, 1963, pp. 489-493.
  44. P. Davies, Design for an associative computer, in *Proceedings of the IEEE Pacific Computer Conference*, March 1963, pp. 109-117.
  45. P. Davies, Associative processors, in *IEEE Symposium on Search Memory*, May 1964.
  46. H. T. Mann and J. Rogers, A cryogenic "between limits" associative memory, in *Proceedings of the IRE National Aerospace Electronic Conference*, May 1962, pp. 359-362.
  47. R. W. Ahrons and L. L. Burns, *Cryogenic Associative Memory Techniques*, RCA, Final Rept., May 1964, Contract No. NR387900 AD-448 504.
  48. R. W. Ahrons and L. L. Burns Jr., Superconductive memories, *Comput. Design* 1, 12-19 (January 1964).
  49. L. L. Burns, *Cryolectric Random Access Memory, Phase 3*, RCA, Final Rept., November 1965, AF 30(602)-3090 AD-624 606.
  50. L. L. Burns, J. Y. Avins, L. S. Cosentino, L. Dworsky, and J. Feder, *Cryolectric Random Access Memory, Phase 3*, Vol. II, RCA Labs., Final Rept., June 1966, AF 30(602)-3090 AD-488 666.
  51. L. L. Burns, G. W. Leck, G. A. Alphonse, and R. W. Katz, Continuous sheet superconducting memory, in *Proceedings of the Symposium on Supereconducting Techniques*, 1960, pp. 167-185.
  52. H. F. Koehler, An impartial look at semiconductors, *Datamation*, 15, 42-46 (July 1971).
  53. J. R. Burns and J. H. Scott, Silicon-on-sapphire complementary MOS circuits for high-speed associative memory, in *Proceedings of the American Federation of Information Processing Societies, 1969 Fall Joint Computer Conference*, 1969, pp. 469-477.
  54. R. Igarashi and T. Yaita, An integrated MOS transistor associative memory system with 100

- nanosecond cycle time, in *Proceedings of the American Federation of Information Processing Societies, 1967 Fall Joint Computer Conference*, Vol. 22, 1967, pp. 499-506.
- 55. R. O. Berg and K. L. Thurber, A hardware executive control for the advanced avionic digital computer system, *NAECON '71 Record*, pp. 206-213 (May 1971).
  - 56. R. O. Berg and M. D. Johnson, An associative memory for executive control in an advanced avionic computer system, in *Proceedings of the IEEE Computer Group Conference*, June 1970, pp. 336-342.
  - 57. L. D. Wald, An associative memory using large-scale integration, *NAECON '70 Record*, pp. 277-281 (May 1970).
  - 58. L. D. Wald, MOS associative memories, *Electron. Eng.*, pp. 54-56 (August 1970).
  - 59. M. Sakaguchi, N. Nishida, and T. Nemoto, A new associative memory system utilizing holography, *IEEE Trans. Comput.* C-19(12), 1174-1181 (1970).
  - 60. P. T. Rux, *Evaluation of Three Content-Addressable Memory Systems Using Glass Delay Lines*, Oregon State Univ., July 1967, AD-660 792.
  - 61. P. T. Rux, *Design and Evaluation of a Glass Delay Line Content-Addressable Memory System*, Oregon State Univ., February 1968, AD-671 910.
  - 62. P. T. Rux, A glass delay line content-addressed memory system, *IEEE Trans. Comput.* C-18(6), 512-520 (1969).
  - 63. G. Estrin, Organization of computer systems—The fixed-plus-variable structure computer, in *Proceedings of the Western Joint Computer Conference*, 1960, pp. 33-37.
  - 64. G. Estrin, *Variable Structure Computer System*, ONR Report ACR-97, Information Systems Summaries, July 1964, p. 48.
  - 65. D. A. Savitt, H. H. Love, and R. E. Troop, *Associative Storing Processor Study*, Hughes Aircraft Co., FR-66-11-75, March 1966.
  - 66. D. A. Savitt, H. H. Love, and R. E. Troop, *Associative Storing Processor Study*, Hughes Aircraft Co., Interim Rept., AD-488 538, June 1966.
  - 67. D. A. Savitt, H. H. Love, and R. E. Troop, *Associative Storing Processor*, Vol. I, Hughes Aircraft Co., AF 30(602)-3669, AD-818 529, June 1967.
  - 68. D. A. Savitt, H. H. Love, and R. E. Troop, *Associative Storing Processor*, Vol. II, Hughes Aircraft Co., Final Rept., AD-818 530, June 1967.
  - 69. D. A. Savitt, H. H. Love, and R. E. Troop, ASP: A new concept in language and machine organization, in *Proceedings of the American Federation of Information Processing Societies, 1967 Spring Joint Computer Conference*, Vol. 31, 1967, pp. 87-102.
  - 70. D. A. Savitt, H. H. Love, R. E. Troop, and R. A. Rutman, *Association Storing Processor Interpretive Program—Program Logic Manual*, Final Rept. Phase No. 1, Hughes Aircraft Co., FR68-11-558, 1968.
  - 71. D. A. Savitt, H. H. Love, R. E. Troop, and R. A. Rutman, *ASP User's Manual Association Storing Processor Interpreter Program*, Hughes Aircraft Co., Fullerton, Calif., 1968.
  - 72. A. P. Kisylia, *An Association Processor For Information Retrieval*, Illinois Univ., Rept. No. R-390, August 1968.
  - 73. C. Lee and M. Paull, A content-addressable distributed logic memory with application to information retrieval, *Proc. IEEE* 15(6), 924-932 (1963).
  - 74. B. A. Crane and J. A. Githens, Bulk processing in distributed logic memory, *IEEE Trans. EC-14* 2, 140-152 (April 1965).
  - 75. C. Lee, Content-addressable and distributed logic memories, in *Applied Automata Theory* (J. T. Tou, ed.), Academic, New York, 1968.
  - 76. C. Lee, Synthesis of a cellular computer, in *Applied Automata Theory* (J. T. Tou, ed.), Academic, New York, 1968.
  - 77. G. J. Lipovski, *The Architecture of a Large Distributed Logic Associative Processor*, Coordinated Science Laboratory R-424, July 1969.
  - 78. G. J. Lipovski, The architecture of a large associative processor, in *Proceedings of the Fall Joint Computer Conference*, 1970, 1970, pp. 385-396.
  - 79. M. A. Knapp, RADC programs in associative processing, in *ONR/RADC Seminar on Associative Processing*, 1967.

80. R. G. Gall and D. E. Brotherton, *Associative List Selector*, Goodyear Aerospace Corp., AD-802 993, October 1966.
81. J. P. Pritchard, Jr., *Fabrication and Testing of Cryogenic Associative Processor Planes*, Texas Instruments, Inc., Final Rept., AD-618 491, May 1965.
82. J. P. Pritchard, Jr., New developments in cryogenic devices, *Int. Electron.* **11** 26-29 (January 1966).
83. J. P. Pritchard, Jr., *Fabrication and Testing of 5000 Word Cryogenic Associative Processor*, Texas Instruments F. T. Final Rept. 100CT66, RADC-TR-66-775, AD-811 983, February 1967.
84. R. W. Haas and E. H. Blevis, *Associative Tag Memory*, Marquardt Corp., AD-620 915, July 1965, p. 92.
85. R. Haas, E. Blevis, S. Requa, and I. Hanlet, *Element Development For Advanced Associative Memories*, Marquardt Corp., AF-30(602)-3709, AD-488 453, June 1966.
86. R. W. Haas and J. M. Hanlet, *Element Development For Advanced Associative Memories*, Marquardt Corp., Final Rept. AD-825 274, December 1967.
87. R. Trepp, *Fabrication Techniques for Batch Fabrication of Distributed Logic Networks*, RADC-TR-66-182, June 1966.
88. J. Goldberg and M. W. Green, *Multiple Instantaneous Response File*, Stanford Research Institute, RADC-TR-61-233, AD-266 169, August 1961.
89. M. H. Lewin, H. R. Bellitz, and J. A. Rajchman, Fixed associative memory using evaporated organic diode arrays, in *Proceedings of the American Federation of Information Processing Societies, 1963 Fall Joint Computer Conference*, Vol. 24, 1963, pp. 101-106.
90. M. H. Lewin, H. R. Bellitz, and J. Guerraccini, Fixed-resistor-card memory, *IEEE Trans. EC-14* **3**, 428-434 (June 1965).
91. M. H. Lewin, A survey of read-only memories, in *Proceedings of the American Federation of Information Processing Societies, 1965 Fall Joint Computer Conference*, Vol. 28, 1965, pp. 775-787.
92. Burroughs Corp., *ILLIAC-IV: Systems Characteristics and Programming Manual*, 2280-68-469, March 1968.
93. Burroughs Corp., *ILLIAC-IV: Systems Characteristics and Programming Manual*, Doc. No. 66000A, June 1969.
94. Illinois University, Department of Computer Science, *ILLIAC-IV*, Quarterly Rept. AF 30(602)-4144, AD-665 916, December 1967.
95. D. L. Slotnick, A parallel computing approach to digital simulation, in *Proceedings of the IBM Scientific Computing Symposium*, 1967.
96. D. L. Slotnick, Unconventional systems, in *Proceedings of the American Federation of Information Processing Societies, 1967 Spring Joint Computer Conference*, Vol. 31, 1967, pp. 477-481.
97. D. L. Slotnick, W. C. Borck, and R. C. McReynolds, The Solomon computer—A preliminary report, in *Proceedings of the 1962 Workshop on Computer Organization*, 1962, pp. 66-92.
98. D. L. Slotnick, W. C. Borck, and R. C. McReynolds, The Solomon computer, in *Proceedings of the American Federation of Information Processing Societies, 1962 Fall Joint Computer Conference*, Vol. 22, 1962, pp. 97-107.
99. A. B. Carroll, J. G. Gregory, W. H. Leonard, and D. L. Slotnick, The Solomon 2 computing system, in *Proceedings of the International Federation for Information Processing Congress*, Vol. 2, 1965, pp. 319-320.
100. J. Gregory and R. McReynolds, The Solomon computer, *IEEE Trans. EC-12* **5**, 774-781 (December 1963).
101. G. A. Westlund, *A Timing Simulator of ILLIAC-IV*, University of Illinois, Department of Computer Science, File No. 775, September 1968.
102. D. J. Kuck, ILLIAC-IV software and application programming, *IEEE Trans. Comput. C-17* **8**, 758-770 (August 1968).
103. L. C. Hobbs, D. J. Theis, J. Trimble, H. Titus, and I. Highberg, *Parallel Processor Systems, Technologies, and Applications*, Spartan, 1970.
104. W. Shooman, Orthogonal processing, in *Parallel Processor Systems, Technologies, and*

- Applications* (L. C. Hobbs, D. J. Theis, J. Trimble, H. Titus, and I. Highberg, eds.), Spartan, 1970, Chap. 15, pp. 297-308.
105. Harold S. Stone, A pipeline pushdown stack computer, in *Parallel Processor Systems, Technologies, and Applications* (L. C. Hobbs, D. J. Theis, J. Trimble, H. Titus, and I. Highberg, eds.), Spartan, 1970, Chap. 12, pp. 235-249.
  106. R. S. Green, J. Minker, and W. E. Shindle, *Analysis of Small Associative Memories for Data Storage and Retrieval Systems, Vol. 1, Management Report*, Auerbach Corp., Final Rept. AD-489 660, July 1966.
  107. R. M. Bird, J. L. Cass, and R. H. Fuller, *Study of Associative Processing Techniques*, RADC-TR-66-209, Vol. 1, AD-800 387, September 1966.
  108. R. M. Bird, J. L. Cass, and R. H. Fuller, *Study of Associative Processing Techniques*, RADC-TR-66-209, Vol. 2, AD-376 572, September 1966.
  109. K. E. Iverson, *A Programming Language*, Wiley, New York, 1962.
  110. G. Estrin and R. H. Fuller, Algorithms for content-addressable memory organization, in *Proceedings of the IEEE Pacific Computer Conference*, 1963, pp. 118-130.
  111. G. Estrin and R. H. Fuller, Some applications for content-addressable memories, in *Proceedings of the American Federation of Information Processing Societies, 1963 Fall Joint Computer Conference*, Vol. 24, 1963, pp. 495-508.
  112. J. L. Rogers and A. Wolinsky, *Associative Memory Algorithms and Their Cryogenic Implementation*, Space Technology Labs. Inc., Rept. No. 8670 6007RU000, 1963.
  113. J. L. Rogers, Algorithms for complex searches, in *IEEE Symposium On Search Memory*, May 1964.
  114. B. H. Scheff, Content-addressable programming techniques, *Electron. Prog.*, pp. 31-36 (Spring/Summer 1966).
  115. Goodyear Aerospace Corp., *Collection of Notes on Associative Memory*, Rep. GER-10587, October 1962.
  116. M. H. Lewin, Retrieval of ordered lists from a content-addressed memory, *RCA Rev.* **23**, 1215-1229 (June 1962).
  117. A. Wolinsky, A simple proof of Lewin's ordered-retrieval theorem for associative memories, *Commun. ACM* **11**(7), 488-490 (1968).
  118. C. Peters, *Associative Memory Compiler Techniques Study*, Informatics, Inc., Final Rept. AD-824 213, November 1967.
  119. J. A. Feldman, *Aspects of Associative Processing*, Lincoln Labs., M.I.T., AD-614 634, April 1965.
  120. P. D. Rovner, An Investigation into Paging a Software-Simulated Associative Memory System, S.M. Thesis, University of California at Berkeley, 1966.
  121. F. J. Hilbing, The Analysis of Strategies for Paging a Large Associative Data Structure, Ph.D. Dissertation, Stanford University, 1968.
  122. J. A. Feldman and P. D. Rovner, *An Algol-Based Associative Language*, Stanford University, Rept. No. AI-MEMO-66, AD-675 037, August 1968.
  123. J. A. Feldman and P. D. Rovner, An Algol-based associative language, *Commun. ACM* **12**, 439-449 (August 1969).
  124. P. D. Rovner and J. A. Feldman, *An Associative Processing System for Conventional Digital Computers*, Lincoln Labs., M.I.T., AD-655 810, April 1967.
  125. P. D. Rovner and J. A. Feldman, The leap language and data structure, in *International Federation for Information Processing*, Edinburgh, Scotland, 1968, pp. C73-77.
  126. D. E. Brotherton and R. G. Gall, *Associated List Selector*, Goodyear Aerospace Corp., Interim Technical Rept., June 1965-February 1966.
  127. N. V. Findler, On a computer language which simulates associative memory and parallel processing, *Cybernetica* **10**(4), 229-254 (1967).
  128. N. V. Findler and W. R. McKinzie, On a new tool in artificial intelligence research: An associative memory, parallel language, AMPPL-II, in *Proceedings of the International Joint Conference on Artificial Intelligence*, 1969.
  129. N. V. Findler, J. Pfaltz, and H. Bernstein, *Four High Level Extensions of Fortran IV: SLIP, AMPPL-II, TREETRAN, and SIMBOLANG*, Spartan, 1972.

130. S. H. Unger, Pattern detection and recognition, *Proc. IEEE* **47**, 1737-1752 (October 1959).
131. B. H. McCormick and J. L. Divilbiss, *Tentative Logical Realization of a Pattern Recognition Computer*, Rept. No. 4031, Digital Computer Lab., University of Illinois, 1961.
132. C. Yang, *Pattern Recognition by an Associative Memory*, Northwestern University, 1966, unpublished paper.
133. R. H. Fuller and R. M. Bird, An associative parallel processor with application to picture processing, in *Proceedings of the American Federation of Information Processing Societies, 1965 Fall Joint Computer Conference*, Vol. 28, 1965, pp. 105-116.
134. M. Aoki and G. Estrin, *The Fixed-Plus-Variable Computer System in Dynamic Programming Formulation of Control System Optimization Problems*, Part I, UCLA Report No., May 1961, pp. 60-66.
135. B. Bussell, *Properties of a Variable Structure Computer System in the Solution of Parabolic Partial Differential Equations*, Ph.D. Dissertation, UCLA, August 1962.
136. G. Estrin and C. R. Viswanathan, Organization of a "fixed-plus-variable" structure computer for computation of eigenvalues and eigenvectors of real symmetric matrices, *J. ACM* **9**(1), 41-60 (1962).
137. G. Estrin and C. R. Viswanathan, Correction and addendum, *J. ACM* **9**(4), 522 (October 1962).
138. Jesse H. Katz, Matrix computations on an associative processor, in *Parallel Processor Systems, Technologies, and Applications* (L. C. Hobbs, D. J. Theis, J. Trimble, H. Titus, and I. Highberg, eds.), Spartan, 1970, Chap. 6, pp. 131-149.
139. P. I. Gilmore, *Matrix Computations On An Associative Processor*. GER-152660, Goodyear Aerospace Corp., June 1971, p. 9.
140. K. J. Thurber, An associative processor for air traffic control, *Proceedings of the 1971 Spring Joint Computer Conference*, 1971, pp. 49-59.
141. K. J. Thurber and R. O. Berg, Applications of associative processors, *Comput. Design*, pp. 103-110 (November 1971).
142. Richard M. Bird, An associative memory parallel deltic realization for active sonar signal processing, in *Parallel Processor Systems, Technologies, and Applications* (L. C. Hobbs, D. J. Theis, J. Trimble, H. Titus, and I. Highberg, eds.), Spartan, 1970, Chap. 5, pp. 107-129.
143. B. A. Crane, Path finding with associative memory, *IEEE Trans. Comput.*, pp. 691-693 (July 1968).
144. J. Minker and W. E. Shindle, *Associative Memory Investigations: Substructure Searching and Data Organization*, Tech. Note 1374-TR-500-1, AF 30(602)-4309, AD-679 227, May 1968.
145. P. Orlando and B. Berra, *Associative Processors in the Solution of Network Problems*, presented at the 39th National Operations Research Society Meeting, May 5-7, 1971, Dallas, Texas.
146. B. F. Cheydleur, SHIEF: A realizable form of associative memory, *Amer. Doc.* **14**(1), 56-67 (1963).
147. B. F. Cheydleur, Dimensioning in an associative memory, in *Vistas in Information Handling*, Vol. 1 (P. W. Howerton and D. C. Weeks, eds.), Spartan, 1963, pp. 55-77.
148. J. Goldberg and M. W. Green, Large files for information retrieval based on simultaneous interrogation of all items, in *Large Capacity Memory Techniques for Computing Systems* (M. C. Yovits, ed.), Macmillan, New York, 1962, pp. 63-77.
149. J. P. Hayes, *A Content-Addressable Memory With Applications to Machine Translation*, University of Illinois Computer Laboratory, Report 227, June 1967.
150. W. Ash and E. Sibley, *A Relational Memory With An Associative Base*, Technical Report 5, University of Michigan, June 1967.
151. W. L. Ash and E. H. Sibley, TRAMP: An interpretive associative processor with deductive capabilities, in *Proceedings of the Association for Computing Machinery 23rd National Conference*, 1968, pp. 143-156.
152. J. E. Griffith, Techniques for advanced information processing systems, in *1st Congress on the Information Systems Sciences*, 1962.

153. A. Wolinsky, Unified interval classification and unified 3-classification for associative memories, *IEEE Trans. Comput.* **C-18**, 899-911 (October 1969).
154. F. T. Baker, W. E. Triest, C. H. Forbes, N. Jacobs, and J. Schenken, *Advanced Computer Organization*, IBM Final Rept. AF 30(602)-3573, AD-484 444, May 1966.
155. R. G. Gall, *Hybrid Associative Computer Study*, Vol. 1, *Basic Report*, Goodyear Aerospace Corp., Final Rept. AD-489 929, July 1966.
156. R. G. Gall, *Hybrid Associative Computer Study*, Vol. 2, *Appendices*, Goodyear Aerospace Corp., AD-489 930, 1966.
157. C. R. DeFiore, N. J. Stillman, and P. B. Berra, Associative techniques in the solution of data management problems, in *Proceedings of the Association for Computing Machinery 1971*, 1971, pp. 28-36.
158. R. B. Stillman, Computation Logic: The Subsumption and Unification Computations, Ph.D. Dissertation, Syracuse University, January 1972.
159. N. J. Stillman, A Feasibility Study of the Applicability of a Hardware Associative Memory to Computer Graphics, Ph.D. Dissertation, Syracuse University, February 1972.
160. N. J. Stillman, C. R. DeFiore, and P. B. Berra, Associative processing of line drawings, in *Proceedings of the Spring Joint Computer Conference*, 1971, Vol. 38, 1971, pp. 557-562.
161. Y. Chu, Application of content-addressed memory for dynamic storage allocation, *RCA Rev.*, pp. 140-152 (March 1965).
162. J. Fotheringham, Dynamic storage allocation in the atlas computer, *Commun. ACM* **8**, 435-436 (October 1961).
163. N. S. Prywess, Man-computer problem solving with multilist, *Proc. IEEE* **54**, 1788-1801 (1966).
164. D. C. Gunderson, J. P. Francis, and W. L. Heimerdinger, *Associative Techniques for Control Functions In A Multiprocessor*, Honeywell, Inc., Rept. No. 12029, RADC TR-66-573, December 1966.
165. M. H. Cannel et al., *Concepts and Applications of Computerized Associative Processing, Including an Associative Processing Bibliography*, MITRE Corp., ESD-TR-70-379, AD-879 281, December 1970.
166. D. Aspinall, D. J. Kinniment, and D. B. G. Edwards, Associative memory in large computer systems, in *International Federation for Information Processing Congress*, 1968.
167. M. Flinders, P. L. Gardner, R. J. Llewelyn, and J. F. Minshall, Functional memory as a general purpose systems technology, in *Proceedings of the 1970 IEEE International Computer Group Conference*, 1970.
168. L. C. Hobbs and D. J. Theis, Survey of parallel processor approaches and techniques, in *Parallel Processor Systems, Technologies, and Applications* (L. C. Hobbs, D. J. Theis, J. Trimble, H. Titus, and I. Highberg, eds.), Spartan, 1970, Chap. 1, pp. 3-20.

Jack Minker

# ASTRONOMY

## INTRODUCTION

The many spectacular advances and discoveries seen by astronomers within the last decade owe much to the availability and application of powerful computational facilities. Some branches of astronomy have been completely revolutionized with the availability of computers and computer techniques. In fact, their use has become so commonplace and widespread today that it has become virtually impossible to isolate specific roles and applications.

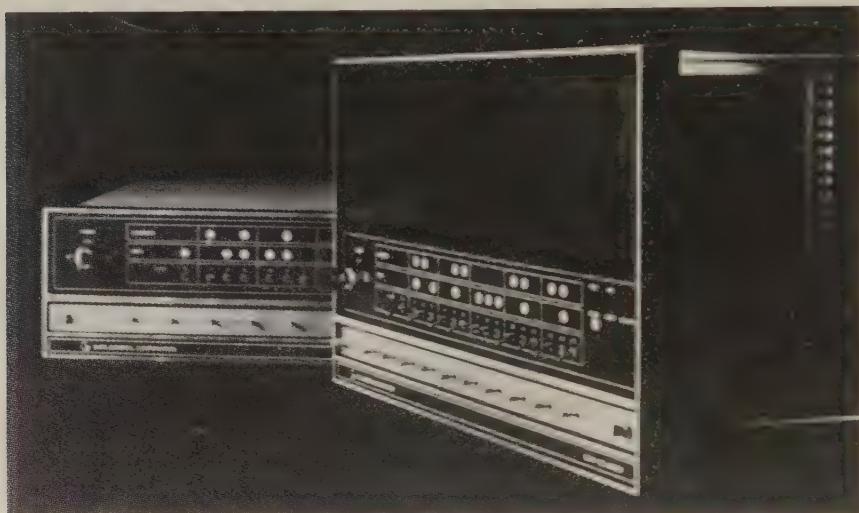
Previous discussions and reviews of computer applications in astronomy are relatively dated and may be found in Davis [1] and Wrubel [2]. More recent discussions emphasizing astrophysical studies can be found in Thomas [3]. Other discussions of broader scientific applications are presented by Fernbach and Taub [4], Lovett [5], and Rice [6].

When computers first became available to astronomers, initial applications consisted primarily of simple and routine reductions of raw observational data. However, as newer and faster computers appeared and increasingly larger storage capacities became available, astronomers implemented increasingly complex computational programs. Although these initial calculational exercises were limited in scope, they demonstrated the potential for more difficult and extensive exercises with still more powerful computers. Many current astrophysical calculations have contrived to place ever-increasing demands for even higher speed and greater storage capabilities. This cyclic trend does not appear to be relaxing in the foreseeable future.

Concurrently, astronomers have extended and pressed the use of small and medium capability computers. State-of-the-art developments, particularly in the smaller but very sophisticated minicomputers such as shown in Fig. 1, have made these systems extremely attractive for a broad range of astronomical studies. Recently, applications have been made to on-line, real-time data reduction and processing at the telescope. Additionally, a widespread trend at automation of many routine observational operations have been instituted in astronomical observatories. In many observatories, computers have become thoroughly blended and integrated with other astronomical instruments to the extent that it has become an indispensable tool for both the observational and theoretical astronomer. Their continued development and application promises even greater potential for advancing astronomical research.

In the present discussion the emphasis is placed on some representative examples of investigations in astronomy which demonstrate the breadth, depth, and scope of applications of computers. In these areas, computers have had their greatest impact on furthering progress and understanding in astronomical research. These examples do not constitute a comprehensive survey of all such applications. Furthermore, no special emphasis is placed on the particular results derived in these examples. The discussion is primarily qualitative in content. No mention of the relative merits or disadvantages

of various computers is provided, nor are there considerations given to the merits of various programming languages, systems programs, or specific numerical methods for computation. However, adequate coverage of the many references which are *typical* of the applications in astronomy today is provided, and interested readers are encouraged to examine the references to find further necessary details in these sources. In this manner, we hope to capture the essence and flavor of current computer applications in astronomy without incorporating excessive detail here. Readers interested in specific computer programs for analysis or automated instrumental operation may find explicit details in the *Communications of the Association for Computing Machinery*, and in the *NASA Computer Bibliography*, for example.



**Fig. 1.** Nova minicomputer which has found widespread use in many laboratories and observatories. Courtesy Data General.

## SOME REPRESENTATIVE CONTEMPORARY APPLICATIONS

### Astrophysical and Stellar Studies

Astrophysicists were quick to recognize and employ computers for extensive and detailed numerical models of stellar structure and evolution. From the mid-1950s to the present, the use of ever increasingly powerful computers has largely revolutionized the astronomer's understanding of stellar structure and evolution.

Modern theoretical astrophysics depends rather heavily on model building for its progress and advancement. The behavior of complex systems ranging from very hot, high-density plasmas, to neutron stars, to large aggregates of stars in clusters, and to

the large-scale cosmological structure of the universe can be studied with the aid of digital computers. Computers have become an indispensable tool for the astrophysicist, allowing him to perform numerical experiments on the properties of matter under conditions too extreme to achieve in the laboratory.

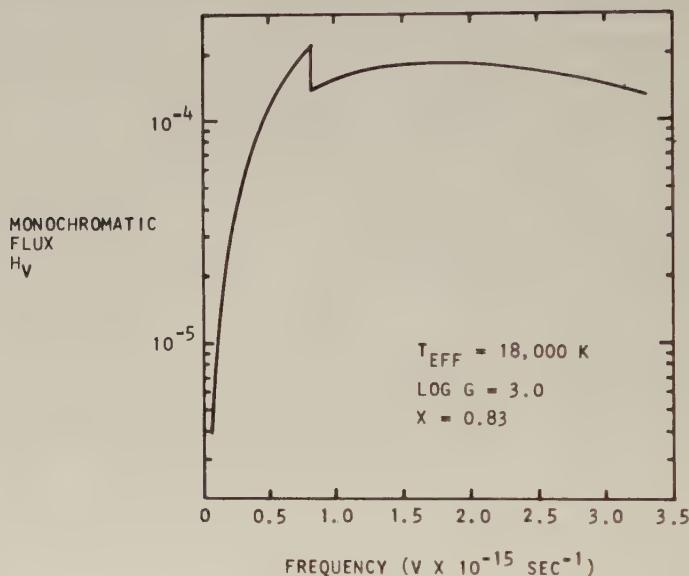
In such studies the astrophysicist attempts to model as realistically as possible the system of radiation and matter and its interaction. As he builds more physically and geometrically real properties into such systems, removing as many assumptions and restrictive conditions as he possibly can, he rapidly encounters the limits of performance and capability of contemporary computers. Thus, in part his progress in understanding the behavior of matter under normal and extreme astrophysical conditions depends on the advancement of the state-of-the-art in computer technology. Such a situation is likely to exist for some time: improved modeling requiring more powerful computers. In some instances, however, limitations in progress have been imposed by limitations in the fundamental knowledge of the physics of matter under certain conditions. The study of neutron stars, their formation, structure, and evolution depend on equations of state for nuclear matter which are not fully understood. Thus the computer, in conjunction with basic astronomical observations, has permitted the astrophysicist to approach a new horizon in his fundamental knowledge.

Many specific areas of astrophysics have been studied by means of computer modeling. To date, a remarkable theoretical understanding has been achieved with the assistance of the computer on the characteristics of stellar atmospheres [7], on stellar structure and evolution [8], and on stellar pulsation [9]. Calculations and numerical experiments have also been made in the study of the dynamics of spiral galaxies [10-13], in the study of the origin and distribution of element abundances [14], and in star formation and early stellar evolution [15]. More recently, numerical computations have made possible astrophysical advances in problems concerned with explosive nucleosynthesis in determining heavy element abundances [16], accurate tests of solar interior models by virtue of predictions and measurements of solar neutrino fluxes [17], and the effects of nonlocal-thermodynamic equilibrium studies on stellar spectra and energy distributions [18]. For purposes of this discussion, some specific examples of published results in stellar astrophysics dealing with stellar atmospheres, stellar structure and evolution, and the dynamics and evolution of galaxies will be emphasized.

In the study of stellar atmospheres, the astronomer is required to interpret photometric and spectroscopic data to provide basic data on temperatures, electron densities, surface gravity, and chemical composition. One must have available detailed numerical models of these outer stellar layers which depend largely on solutions of nonlinear, integro-differential equations of radiative transfer. Extensive use is also made of basic atomic physics data, some of which are obtained empirically and some acquired by independent calculations, also performed with the aid of digital computers.

In dealing with the structure of stellar atmospheres, two fundamental equations need to be treated. The first describes the energy transport or transfer based on one or more of several potential mechanisms such as radiative and/or convective transfer. The appropriate expression determines the distribution of temperature within the atmosphere considered. The second equation describes the mechanical balance, usually in

terms of hydrostatic equilibrium, and results in the distribution of pressure within the atmosphere. If the chemical composition is known or assumed, the equation of state and the excitation ionization conditions may be computed. If, in addition, an effective temperature and surface gravity are given, the theory permits the calculation of model atmospheres in terms of their pressure and temperature distributions. With these models, additional calculations of observable quantities such as the emergent energy distribution, spectral line profiles, and photometric indices can be effected. (See Figs. 2 and 3.) A series of calculations like this, resulting in a grid of models, may then be used for comparison with observational results to infer the physical properties of stellar atmospheres which best fit the data. In some instances this technique can be exploited to the point where the basic physical assumptions and ground rules may themselves be examined in order to explore any possible defects in the theory.

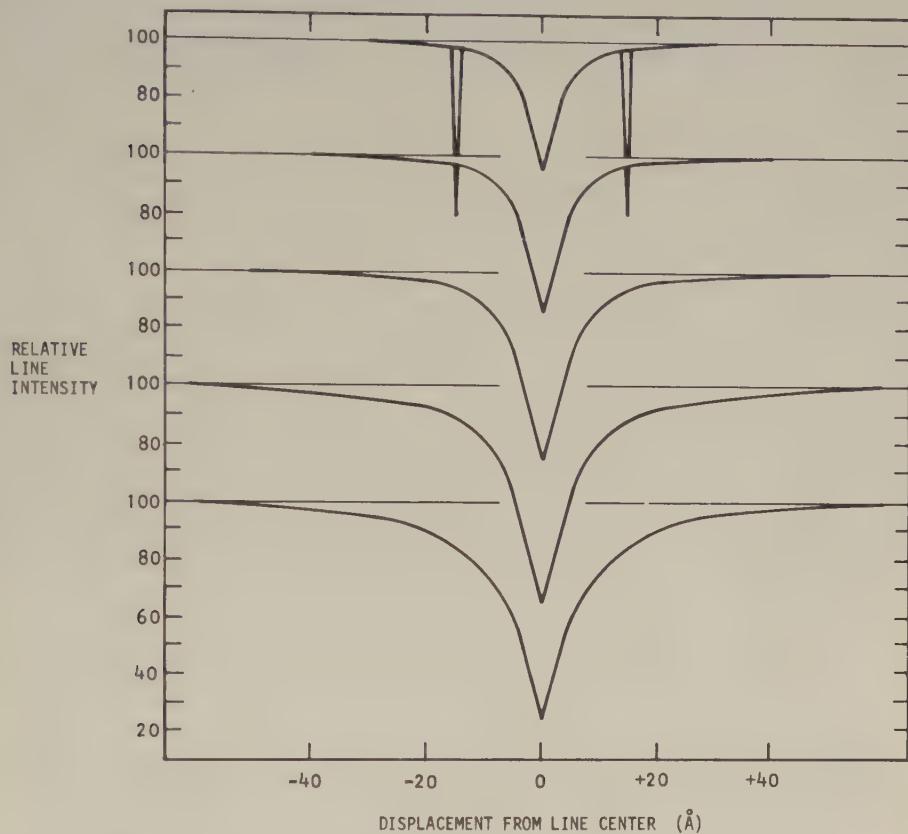


**Fig. 2.** Emergent energy distribution calculated from a hot stellar atmospheric model. Adapted from Strom and Avrett [20].

A description of an early specific program for use on a large digital computer is given by Gingerich [19]. The program is used to calculate a model atmosphere with a prescribed distribution of temperature and opacity, and provides an iterative means for improvement of the temperature distribution to achieve a desired flux.

Many exhaustive and elaborate models of stellar atmospheres have been calculated and published in the past decade. A recent monograph by Mihalas [7] discusses the basic theoretical framework, emphasizing calculations dealing with the difficult problem of non-LTE (local thermodynamic equilibrium) effects. Additional detailed computer programs are discussed by Mihalas [21], who also describes the required numerical techniques. Several examples of relatively recent tabulations of numerical

models may be found in Strom and Averett [20], Mihalas [21], and Parsons [22]. Other aspects of the problems associated with understanding stellar atmospheres, particularly with regard to calculations of spectrum line profiles, may be found in Underhill [23] and Mihalas and Auer [18]. Other calculations based on results of such models have extended their use for the interpretation of a wide variety of observational data. Indeed, the value of such grids of models lies largely in such applications. (See Bell [24], Hill and Hutchings [25, 26], and Fäy [27], for example.)



**Fig. 3.** Some early computed line profiles of H $\delta$  calculated for several model atmospheres in LTE. Lines of Si(IV) at  $-13$  and  $+15$  Å are included in the upper models. Adapted from Underhill [122]. See Mihalas and Auer [18] for some recent examples which include the effects of non-LTE.

Much of the progress in this area of stellar astrophysics has been made possible by the numerical techniques developed to handle the difficult radiative transfer calculations that are needed. Some of the investigations dealing with such procedures concerning radiative transfer may be found in Wendroff [28], Grant [29], Rybicki [30], and Ramnath [31], for example. See Modali *et al.* [32] for a discussion of an interesting and difficult transfer problem. The computational analysis necessary for the construc-

tion of numerical models of stellar atmospheres is again treated in Carrier and Avrett [33], Mihalas [21], Hummer and Rybicki [34], and Gingerich [19], for example. Similar discussions for models of planetary atmospheres are given in Bellman *et al.* [35, 36], while an example of some numerical models are given in Bartko and Hanel [37].

In studies of stellar structure and evolution, basic computational methods are discussed by Henyey *et al.* [38, 39], Kippenhahn *et al.* [40], Sears and Brownlee [41], Reiz and Petersen [42], and May and White [43], for example. The actual construction of stellar models has been achieved by numerical integration of four nonlinear, coupled differential equations. These equations represent the conditions for mechanical equilibrium, continuity, thermal equilibrium, and energy transport in a quasi-stationary state. These expressions are supplemented by three additional constitutive equations describing the energy generation rates, the opacity, and the equation of state, all of which depend on an assumed chemical composition. Numerical solutions are obtained after adopting suitable integration procedures and applying an appropriate set of boundary conditions. Henyey and his associates have developed the required numerical techniques which are almost universally used today.

Large grids of stellar models have been computed by Iben starting with his early results published in Iben and Ehrlman [44] and Iben [45], and followed by a long and continuing series in the *Astrophysical Journal* (see Iben [8]). Other examples are discussed in Hayashi [46], Kelsall and Stromgren [47], and Demarque and Geisler [48], for example. Table 1 and Fig. 4 show some typical results of stellar models and stellar evolutionary tracks, respectively. It is in the area of stellar evolution and age determinations that computers have had their greatest impact in furthering basic astronomical understanding. And the use of computers in probing the advanced, complex stages of stellar evolution offer even greater hope for filling in the remaining gaps in the astronomer's complete understanding of stellar evolution from birth to death.

**Table 1**

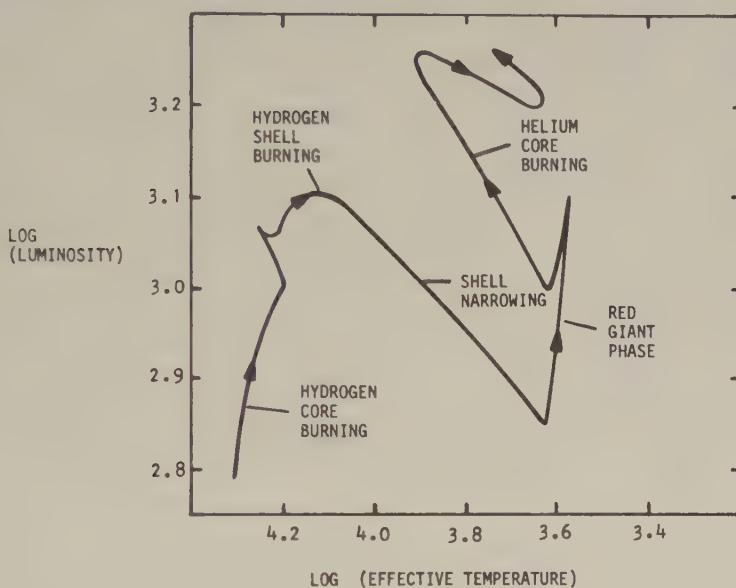
Tabulation of a 15-Solar Mass Stellar Model Developed from Numerical Computations with a Large Computer<sup>a</sup>

Age( $10^6$ years)	$M_{bol}$	$\log(T_{eff})$	$\log(R/R_\odot)$	$\log(L/L_\odot)$	$\log(T_c)$	$\log(\rho_c)$
0.0	-6.21	4.494	0.6997	4.3271	7.5093	0.7109
2.0	-6.32	4.492	0.7271	4.3731	7.5159	0.7092
4.0	-6.44	4.487	0.7606	4.4203	7.5239	0.7114
5.0	-6.51	4.483	0.7829	4.4486	7.5290	0.7154
7.0	-6.66	4.470	0.8377	4.5088	7.5417	0.7329
9.0	-6.83	4.445	0.9228	4.5759	7.5624	0.7775
9.7	-6.90	4.429	0.9683	4.6052	7.5759	0.8137
10.2	-6.96	4.417	1.0039	4.6280	7.5916	0.8591
10.5	-7.00	4.409	1.0279	4.6420	7.6084	0.9105

<sup>a</sup> Calculations such as these are routinely produced and are fundamental in understanding stellar structure; adapted from Riez and Petersen [42].

Recently, several broad attacks on the dynamics and evolution of stellar clusters

and spiral galaxies have been attempted. A number of investigations by Hohl and Hockney [49], Hohl [10, 11], and Miller [50], on the numerical simulation of a galaxy has been reported in the past several years. Additional related studies concerning the formation and evolution of spiral arms in galaxies are available in Barricelli *et al.* [51], Miller *et al.* [13], and Hohl [12]. These computer models consist of numerous point stars (about 100,000) used to simulate the evolution of an isolated disk-shaped galaxy over many revolutions. Computational techniques have been discussed by Cuperman *et al.* [52], Hénon [53], Hohl [11], and Miller *et al.* [13], and are based on direct summation or Fourier transform methods for calculating the gravitational potential in these studies. Figure 5 illustrates some of Hohl's results obtained on a CDC 6600 system indicating the dynamical instability of spiral arms in such stellar systems.

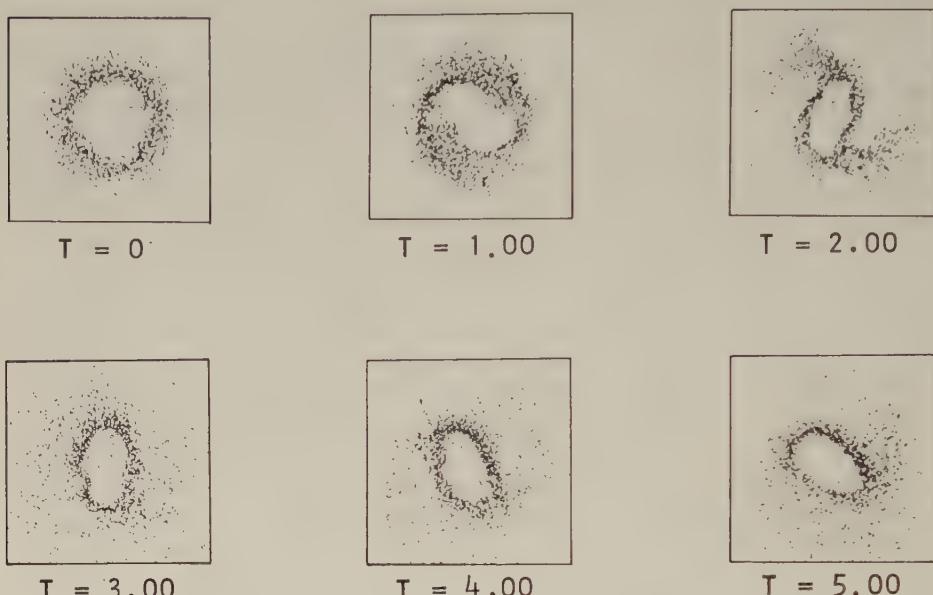


**Fig. 4.** A schematic, postmain sequence evolutionary track for a 5-solar mass star in the HR diagram. The detailed excursions and lifetimes typical of these calculations consume considerable computer time. Adapted from Iben [8].

In addition to these basic numerical studies, some beautiful results have been obtained by the Toomre's, who have studied the tidal interaction of multiple galaxies. Their analyses indicate that the formation of bridges of matter between close galaxies and the existence of long tails are due to brief but violent gravitational tidal interactions. See the report by Toomre and Toomre [54] for an informative and entertaining discussion, and for many illustrative examples.

In other astrophysical areas, fruitful computational studies have permitted the use of powerful interferometric techniques for planetary spectroscopy. Discussions are given in Cooley and Tukey [55], Connes [56], and Singleton [57]. Some other examples

of computations useful for analysis of certain stellar observations can be found in Proctor and Linnell [58], Bartko [59], and Kinman [60]. Further discussions of general astrophysical applications of computers may be found in Thomas [3].



**Fig. 5.** A two-dimensional plot of the spatial evolution of an initially uniformly rotating cold disk of stars. The indicated times are multiples of the rotation period. Note the brief appearance of spiral arms at  $t = 2$  and their subsequent rapid dissolution. Adapted from Hohl [49].

Future progress in stellar astrophysics through computational analysis by means of large and powerful computers will likely be found in topics concerned with hydrodynamical effects on the structure of stellar atmospheres and in the development and testing of various theories of stellar convection and its role in stellar mass loss problems, particularly for the complex, advanced stages of stellar evolution.

### Dynamical Astronomy and Astrometric Studies

Dynamical astronomy has undergone a renaissance, brought about largely by the use of computers to solve some long standing classical problems numerically. The orbital motions of the planets have been computed to far greater accuracy, particularly with reference to perturbations arising from other planetary bodies. In addition, sophisticated analyses of satellite orbital problems have become possible with numerical studies on large computers. The study of  $N$ -body problems has also seen significant progress, with several numerical experiments in the construction of models of star clusters and galaxies carried out successfully. Large configurations of stars have been

studied with respect to their stability and dynamical evolution. (See the preceding section.)

In solar system studies, the masses of the outer planets have been determined with greater accuracy using associated satellites and asteroids in some cases. Such calculations have been reported by Duncombe *et al.* [61], Klepczinski *et al.* [62–64], and Seidelmann [65]. Numerical procedures needed to effect such calculations are given in Jeffreys [66], LeSchack and Sconzo [67], and Walter [68], for example. The discussion by Jeffreys [66], in particular, is of great interest to those astronomers with special interests in celestial mechanics. A concise, authoritative summary of computer-related requirements for most problems in celestial mechanics, and a résumé of some of the recent calculations which have been attempted, is provided by Jeffreys.

Several other examples of solar system-related orbital studies include Everhart [69], Janiczek *et al.* [70], Muller [71], and Seidelmann [72]. Additional relevant applications may be found in Volume 73 of the *Astronomical Journal* and recent numbers of *Celestial Mechanics*. (Consult the article by Seidelmann, *Celestial Mechanics*.)

Turning to astrometric studies, computer techniques have permitted the development of considerably more sophisticated analyses of astrometric data. The plate overlap technique developed by Eichorn and discussed by Googe [73] requires extensive computational analysis. Similarly, classical reduction procedures necessary in treating large bodies of astrometric data can be handled more readily and in a fashion which allows numerous systematic errors and corrections to be determined routinely and simultaneously, thus allowing more accurate results. Small perturbations can be easily sorted in this fashion, allowing the inference of third body effects.

In other applications, computers have made possible the efficient determination of very high angular resolution measurements by means of intercontinental Very Long Baseline Interferometry with radio astronomical techniques. Angular resolutions of 0.001 arc sec have been achieved, but the analysis requires the application of many difficult corrections, efficiently and largely achieved by computer techniques.

Finally, as indicated in the discussion of automation, astrometric data have been more effectively and more accurately acquired as a direct result of current technology of automation, ranging from the automation of fundamental transit circle observations [74] to the use of sophisticated measuring engines [75] adapted from nuclear science studies.

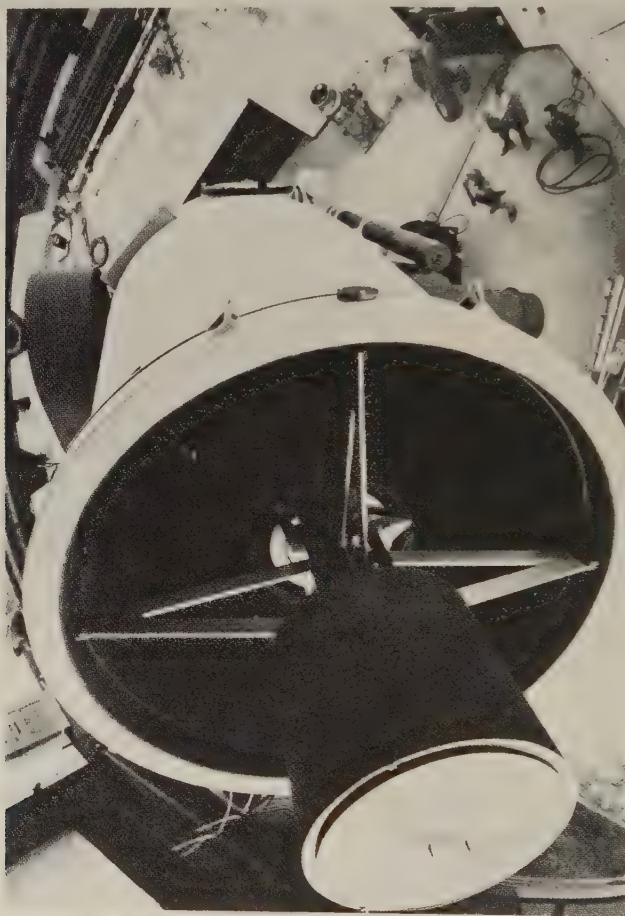
Future progress will likely continue in the area of basic improvements of fundamental solar system data regarding planetary orbits and elements. Studies of cometary and asteroidal orbits and perturbations will also likely increase. As indicated by Duncombe *et al.* [76], some areas of dynamical astronomy that are in need of further computational development include the study of planetary perturbations on the lunar orbit, the development of a more precise description of terrestrial precession and nutation, and the study of the general perturbations of all major planets in their simultaneous orbit determination.

### Observatory Automation and Related Studies

Extensive use is being made of computers in automating many astronomical instruments, the actual observational acquisition of data and its subsequent reduction, and

in supervising or in monitoring the real-time status of an observatory and all of its instruments.

Within the last decade many astronomical observatories have been automated to various degrees. Automated telescopes exist at the Kitt Peak National Observatory [77, 78], the National Radio Astronomy Observatory, the U.S. Naval Observatory [74], and most recently at the Massachusetts Institute of Technology [79]. Many smaller facilities have also been automated in part. (See the discussion by Maran [80] for some earlier examples.) Recently a great deal of effort has been expended on the automation of very large telescopes and entire observatories by means of a system of minicomputers supervised with a large computing system. Two examples of large ground-based telescopes relying heavily on computer control are shown in Figs. 6 and 7. Many varied



**Fig. 6.** A large, ground-based optical telescope which is controlled and operated by an IBM 1800 computer system. The computer points and guides the telescope, and rotates the dome. The effects of atmospheric and instrumental distortion are automatically taken into account. Courtesy IBM.

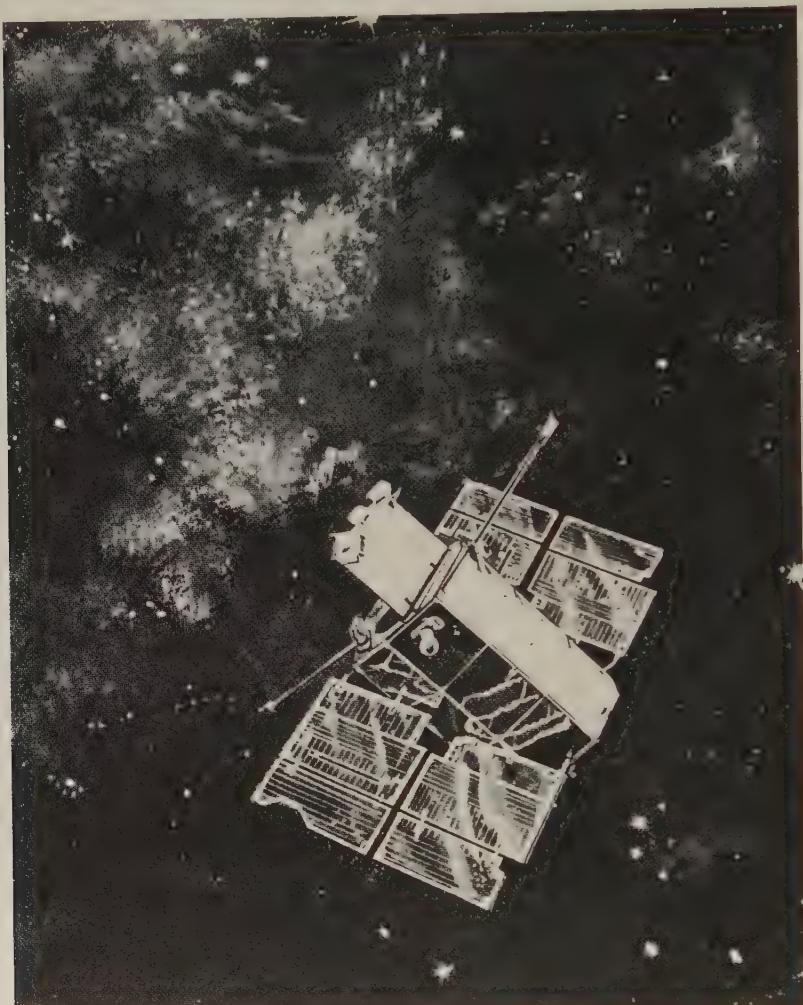
examples are discussed in the 1971 International Astronomical Union Colloquium on *Automation in Optical Astrophysics*. Other similar discussions may be found in the February 1972 issue of the *Publications of the Astronomical Society of the Pacific*. In most of these, the automation has resulted in systematically optimizing observing programs, has made possible some real-time data analysis, and has significantly increased efficiency in telescope operations and data acquisition.



**Fig. 7.** A large, ground-based radio telescope, relying heavily on an IBM 1130 computer for data handling, control, and processing. Courtesy IBM.

An important recent application relying heavily on computer usage is found in the Orbiting Astronomical Observatory program. The Orbiting Astronomical Observatory (OAO) is shown in Fig. 8. Computers are incorporated in the entire system of data handling, transmission and storage, both in space and on the ground. The remote operation of the OAO in space requires rapid computational effort in planning and assembling a detailed observational program. Observations are subject to numerous constraints involving available sky coverage, spacecraft thermal control, and suppression and minimization of stray, scattered solar radiation. The observations are effected

by a sequence of commands which are generated on the ground by computer. After the selection of appropriate program stars, an IBM 360/65 system is used at the ground control center to assemble the required sequence of commands necessary for executing the desired observations. These are then transmitted by a ground control station to the



**Fig. 8.** Orbiting Astronomical Observatory (OAO). Much of the control and operations of this satellite are executed by on-board computers. Additionally, large, ground-based computer systems at Goddard Space Flight Center's mission control center are responsible for handling and processing of OAO data. Courtesy NASA.

spacecraft, where the commands are stored in the spacecraft computer. The commands are issued at the designated times according to the spacecraft clock, and the resulting

data are stored in the memory. Upon passage over an appropriate ground station, the data may be dumped via telemetry where it is stored for later processing and analysis by still another computing system. In this particular application, many computers have become an integral portion of the entire space observatory system. The reader may refer to Code *et al.* [81] for a more complete discussion, while the report by Heacox [82] illustrates many of the programming problems associated with this system. Other space-related applications appearing in the current literature include Hanel *et al.* [83], Reeves *et al.* [84], Rindfleisch *et al.* [85], Shufelt [86], and Stabler and Creveling [87].

Some relevant discussions concerning automation of various ground-based instruments are given in Johnson and Mitchell [88], Bahng [89], Mankin *et al.* [90], Wells [91], McNall *et al.* [92], and Winnberg [93]. Representative applications involving computer use in data reduction and processing may be found in Sharpless [94], Herbison-Evans and Lomb [95, 96], Wills [97], Rindfleisch *et al.* [85], Bonsack [98], and Shufelt [86]. Discussions involving automated astronomical measuring machines are given in Stoy [99], Strand [75], and Luyten [100]. An example is shown in Fig. 9.

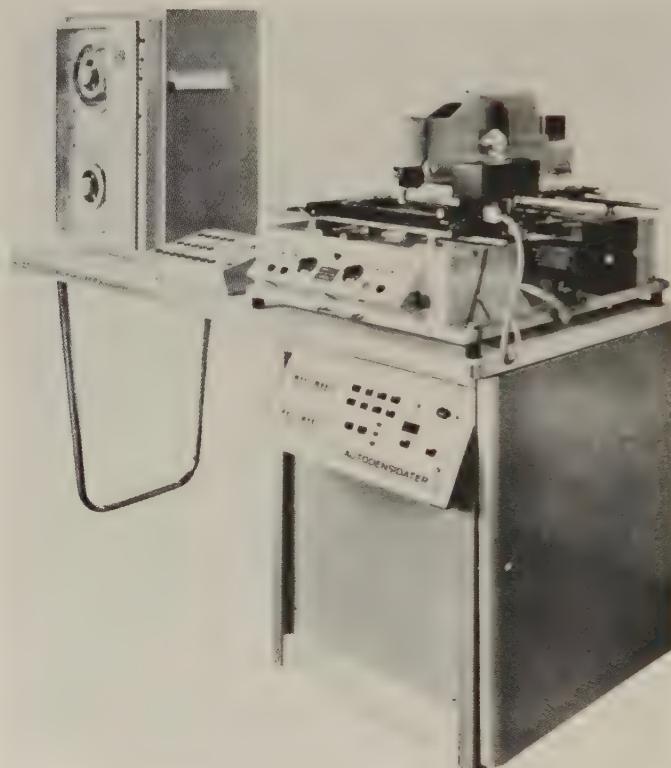
Important studies reporting the general planning and problems in observatory automation and data systems, with a variety of telescopes and instruments, are given in Albrecht *et al.* [101], Carpenter [102], Clark [103], Dennison [104], Hameen-Anttila [105], Reddish [106, 107], McCord *et al.* [79], Heart and Mathiasen [108], Maran [77, 78], and Nather [109]. Other related aspects are covered in Clayton *et al.* [110] and Trumbo [111], and some programming examples may be found in Grutzner [112], and Erickson *et al.* [113]. Figure 10 illustrates a block diagram which typifies some current automated observatories resulting from these studies. In such cases the computers become integral, functional, component instruments, residing as the mechanical observatory directory.

### Instructional Uses and Historical Studies

Other applications of computers in astronomy have been concerned with instructional uses in educational programs; with the study of historical problems relating to "astro-archeology" and the simulation of planetary orbits and positions many centuries ago; and with the preparation of large astronomical data banks to serve as fundamental reference sources of information. In the first case, only limited applications exist and the potential for future demonstrations of fundamental astronomical principles and concepts by computer techniques is very great. Astronomers working in these areas are few in number and progress is likely to be slow. Nevertheless, the value of programmed educational techniques should not be underestimated, and careful planning of appropriate demonstrations by computational means is sure to be rewarding, particularly for the student of astronomy. Samples of some recent investigations which deal with the use of computers in astronomically related instructional roles are found in Messina [114], Galley [115], and Blum and Bork [116].

Several intriguing discussions concerning the astronomical capabilities and facilities of early man, centuries before recorded history, have been presented by Hawkins [117, 118] and Hoyle [119]. These studies have been concerned with searching past

astronomical records, events, and achievements, and attempting to demonstrate the existence of sophisticated ancient astronomical observatories. In the discussion by Hawkins [117], a number of suggestions have been put forth on the astronomical significance of the alignments and configurations of many features found at Stonehenge. For example, Hawkins has suggested that this facility could be regarded as a neolithic computer used for predicting lunar phases and eclipse occurrences by early inhabitants. In conjunction with archeological data for this site, it has been inferred that early inhabitants of Stonehenge (about 2000 B.C.) were able to conduct rather sophisticated astronomical observations. Much of the analysis required, which permitted these inferences by Hawkins on Stonehenge in particular and other similar locations, is based on extensive computational techniques.

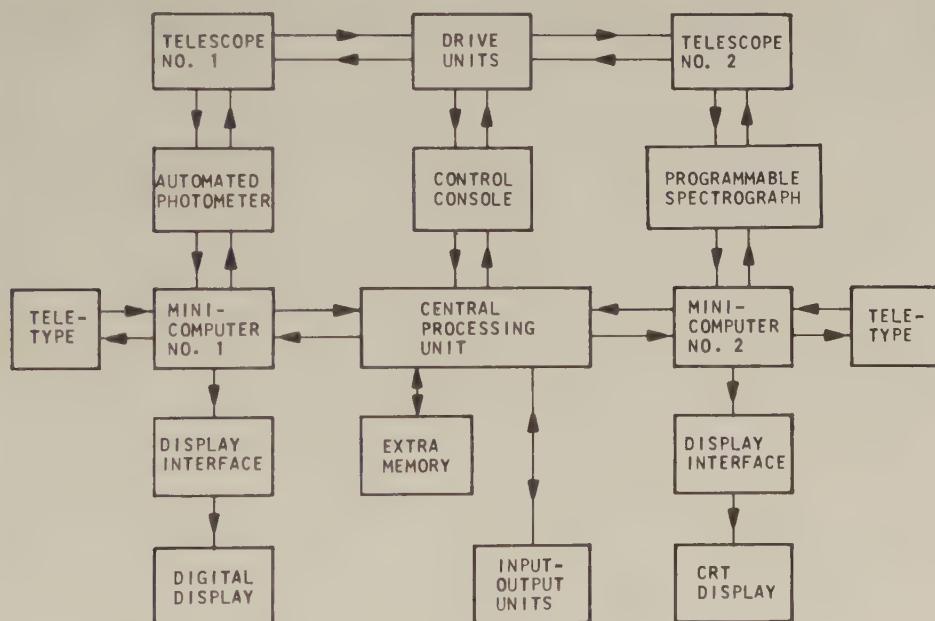


**Fig. 9.** An automated densitometer. Such devices are useful in measuring and analyzing stellar spectra. Many software options are available to allow direct communications with a computer in order to reduce and process the data for further study. Courtesy Joyce-Loebel Co.

Computers have also proved to be useful tools in the study of certain historical developments in astronomy. Many tables of planetary, solar, and lunar positions have been tabulated for periods of time ranging from a few thousand years B.C. to the

present, and well beyond in some cases. The results of these calculations have proven useful in the interpretation of historical documents and certain classical astronomical calculations. In standard practice a particular computation may be programmed for the computer and repeated as often as desired. By adjusting various parameters, one may explore and attempt to discover the underlying principles used in the construction of many key and intriguing ancient historical documents and tables having an apparent astronomical basis. Gingerich has demonstrated the utility of such calculations and of computer-based orbital computations in simulating planetary positions and configurations centuries ago. Gingerich's discussion concerning Kepler's analysis of planetary orbits, particularly the Martian orbit, is especially instructive. See Gingerich [120, 121] for a fuller discussion of these applications.

Finally, numerous steps in compiling large quantities of basic astronomical data on computer-compatible media have been attempted. Specialized catalogs of basic astronomical data, consisting, for example, of photometric data, radial velocities, proper motions, and positions, have been assembled on magnetic tapes and disks. Other types of data banks of a more universal nature have been collected which permit one to present specific queries to the computer and obtain rapid evaluations. The efforts of Hynek and his associates illustrate the value and potential of such collections of data. Special libraries of cataloged data will be of particular value in fully automated observatories, where observational programs can be carefully and efficiently planned. Such collections allow one to realize the convenience of computer-generated lists of astronomical sources appropriate for various observations at any given time.



**Fig. 10.** Schematic block diagram of typical automated observatory.

## SOME FUTURE TRENDS AND SPECULATIONS

Technological developments in computer capabilities have been so rapid that anticipated future generations of computers are difficult to envision. Few areas of technology have kept pace with the rapid improvement and advances in computer technology. And fewer areas, still, have had such a major impact on the manner in which astronomy is presently conducted.

It is perhaps pure folly to conjecture on possible future applications of computer and computational techniques in astronomy and astrophysics, since such applications depend largely on unforeseen technological breakthroughs in both computer hardware and software. Future discoveries and developments in astronomy will likely tax the capability and capacity of the then contemporary state-of-the-art facilities and spur still further development. As a result, extrapolations of present studies and application of computers to the future are speculative and likely to be erroneous. Nevertheless, certain trends seem to be reasonably well-defined and we speculate on some possibilities, particularly those for the near term future (see Fig. 11). With these qualifications in mind, the following areas of astronomical study will likely benefit from more extensive application of computer techniques:

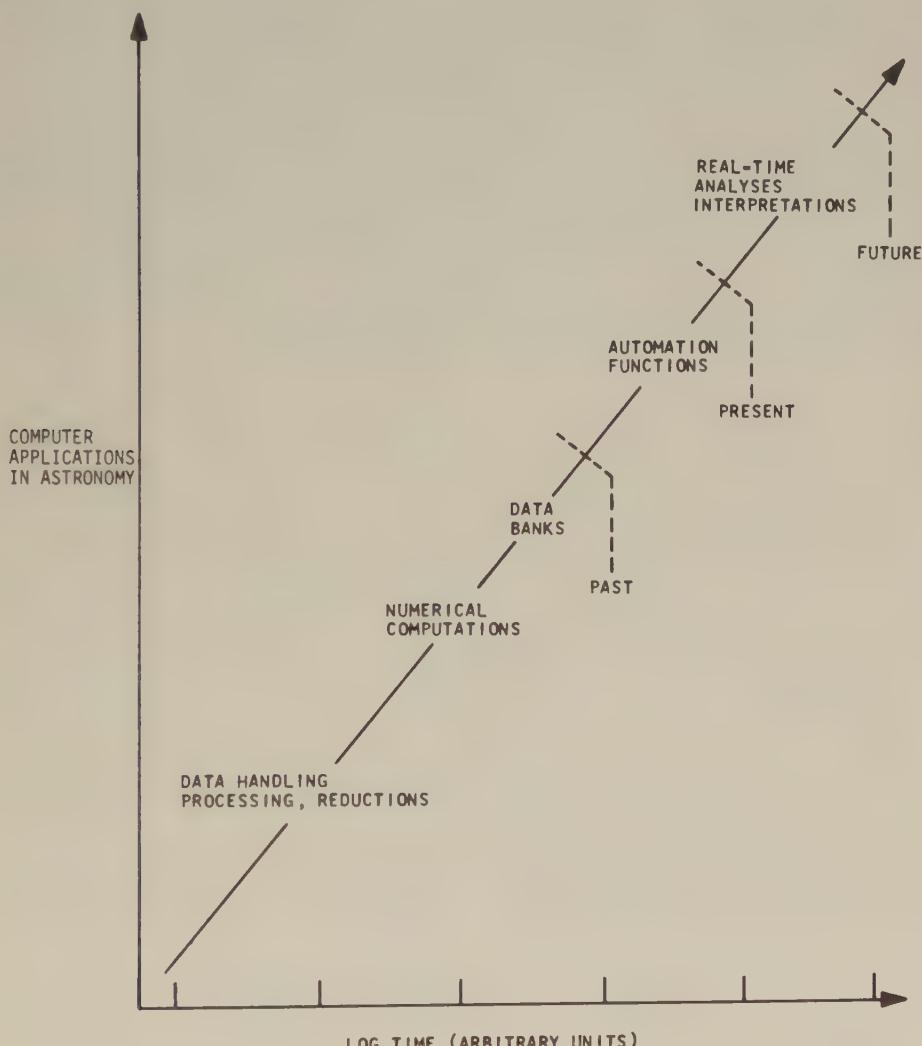
1. Numerical computations requiring virtually unlimited storage capacity will probably be routinely handled. In such applications the astronomer user may be required to expend large sums of effort and time in developing the computer programs necessary to handle the numerical computations. In such cases the development of more extensive and elaborate modular-type programming languages, capable of handling extensive blocks of calculations automatically, will be required to ease such burdens.

2. Mini- and midisized computers dedicated to specific operational functions in the astronomical observatory ought to be routine. In fact, one can readily suppose that dedicated observatory systems, comprised of a master central processing device controlling a number of smaller capacity slave units which may be associated with specific instruments on telescopes, should be available. In fact, such systems will be sufficiently capable and versatile to automatically plan, program, and execute complete observational programs more efficiently and effectively than presently available under manual control and supervision. Such systems would necessarily be integral with the entire observatory, and be subject to only routine servicing and reprogramming in order to fit the observational needs of the astronomical staff.

3. Many catalogs of astronomical data currently are available at several national centers on magnetic tape and cards. These data range from complete photometric catalogs to compilation of sundry astronomical information of wide application and interest. It is probably safe to assert that future observatories will each house extensive collections of data, formerly contained in library documents, on magnetic tape, disks, or cards. These data are invaluable in the planning and preparation of extensive, long-term observational programs basic to the advancement of research programs in astronomy.

4. In certain observations it may be possible to attempt real-time analyses and pos-

sible interpretation of data on the observational line at the observatory. In fact, it is presently possible, on a limited basis, to collect spectroscopic data in real time, and shortly thereafter, with modest processing of the data, obtain spectral plots suitable for analysis, interpretation, and strategic planning for further observations. It is not difficult to imagine an on-line processor memory with numerous theoretical models stored, with their corresponding observational results available for direct comparison with the real-time data, on which observational decisions could be made directly and promptly.



**Fig. 11.** Schematic summary of astronomical studies performed by computers.

5. Lastly, but not necessarily completely, one can predict the increased use of computers for purely instructional purposes. Many astronomical calculations and principles, both classical and astrophysical, lend themselves more readily to the clarity of computer demonstrations and practice. Several such attempts have been successfully demonstrated, and as a greater number of less expensive computers become available, one can imagine fairly complete courses in both general and specialized astronomy to be available, with demonstratable lessons, exercises, and problems. While it is true that good, readable textbooks will always find frequent use, many of the topics normally treated in standard, required classes will almost certainly be routinely and efficiently handled with the aid of the computer. While the computer is an indispensable tool for the observational and theoretical astronomer, it appears likely that it will be of considerable service to the student of astronomy as well.

The above-mentioned future uses of computers in astronomical work are to a large extent projections of current uses and depend to a great extent on the future resources that will be available. A blend of astronomical research and its contemporary computational needs, the capabilities of future generations of computer systems, and the imagination of the future astronomer users will likely determine what the actual uses of computers in astronomy will be. There is little doubt, however, that decades and centuries hence, an extensive array of exciting problems will confront the astronomer and challenge the capabilities of his state-of-the-art computers.

## REFERENCES

1. M. S. Davis, The role of computers in astronomy, in *Application of Digital Computers*, 1963, Ginon, pp. 85-96.
2. M. A. Wrubel, The electronic computer as an astronomical instrument, *Vistas Astron.* **3**, 107 (1960).
3. H. C. Thomas, Computer programs in astrophysics, *Comput. Phys. Commun.* **3**, 151 (1972).
4. S. Fernbach and A. M. Taub, *Computers and Their Role in the Physical Sciences*, Gordon and Breach, New York, 1970.
5. L. L. Lovett, Over 2100 applications of computers and data processing, *Comput. Automation* **20**, 20 (June 30, 1971).
6. J. R. Rice, On the present and future of scientific computation, *Commun. ACM* **15**, 637 (July 1972).
7. D. Mihalas, *Stellar Atmospheres*, Freeman, San Francisco, 1970.
8. I. Iben, Stellar evolution within and off the main sequence, *Ann. Rev. Astron. Astrophys.* **5**, 571 (1967).
9. R. F. Christy, Computational methods in stellar pulsation, in *Methods in Computational Physics*, Vol. 7, Academic, New York, 1967, p. 191.
10. F. Hohl, Dynamics of plane stellar systems, *Astrophys. Space Sci.* **14**, 91 (1971).
11. F. Hohl, Integration methods where force is obtained from the smoothed gravitational field, *Astrophys. Space Sci.* **14**, 168 (1971).
12. F. Hohl, Numerical experiments with a disk of stars, *Astrophys. J.* **168**, 343 (1971).
13. R. H. Miller, K. H. Prendergast, and W. J. Quirk, Numerical experiments on spiral structure, *Astrophys. J.* **161**, 903 (1970).
14. R. V. Wagoner, W. A. Fowler, and F. Hoyle, On the synthesis of elements at very high temperatures, *Astrophys. J.* **148**, 3 (1967).
15. R. B. Larson, Processes in collapsing interstellar clouds, *Ann. Rev. Astron. Astrophys.* **11**, 219 (1973).

16. R. V. Wagoner, Big bang nucleosynthesis revisited, *Astrophys. J.* **179**, 343 (1972).
17. J. N. Bahcall and R. L. Sears, Solar neutrinos, *Ann. Rev. Astron. Astrophys.* **10**, 25 (1972).
18. D. Mihalas, and L. H. Auer, Non-LTE model atmospheres. V. Multi-line hydrogen-helium models for O and early B stars, *Astrophys. J.* **162**, 1161 (1970).
19. O. Gingerich, Studies in non-gray stellar atmospheres. I. A. Basic computer program, *Astrophys. J.* **576** (1963).
20. S. E. Strom and E. H. Avrett, The temperature structure of early type model stellar atmospheres, II. A grid of models, *Astrophys. J. Suppl.* **12**(103), (1965).
21. D. Mihalas, The calculation of model stellar atmospheres, in *Methods in Computational Physics*, Vol. 7, Academic, N.Y., 1967, p. 1.
22. S. B. Parsons, Model atmospheres for yellow supergiants, *Astrophys. J. Suppl.* **18**(159), (1969).
23. A. B. Underhill, *The Early Type Stars*, Gordon and Breach, New York, 1966.
24. R. A. Bell, The application of synthetic spectra to color system transformations, II. The photographic and UBV systems, *M.N.R.A.S.*, **159**(4), 387 (1972).
25. G. Hill and J. B. Hutchings, The synthesis of close-binary light curves, I. The reflection effect and distortion in algol, *Astrophys. J.* **162**, 276 (1970).
26. J. B. Hutchings and G. Hill, The synthesis of close-binary light curves, II. Double distortion and the system AS Eridani, Lambda Tauri, and RS Vulpeculae, *Astrophys. J.* **166**, 373 (1971).
27. T. D. Fäy, Computed and observed cyanide-radical spectra of three N stars in the infrared, *Astrophys. J.* **168**, 99 (1972).
28. B. Wendroff, A difference scheme for radiative transfer, *J. Comput. Phys.*, **4**, 211 (1969).
29. I. P. Grant, Numerical approximations in radiative transfer, *Astrophys. J.* **125**, 417 (1962).
30. G. Rybicki, A modified Feautrier method, *J. Quant. Spectr. Radiative Transfer* **11**, 647 (1971).
31. R. V. Ramnath, On a class of nonlinear differential equations in astrophysics, *J. Math. Anal. Appl.* **35**, 27 (1971).
32. S. R. Modali, J. C. Brandt, S. O. Kastner, Monte Carlo treatment of Lyman-alpha radiation in a plane-parallel atmosphere, *Astrophys. J.* **175**, 265 (1972).
33. G. F. Carrier and E. H. Avrett, A non-gray radiative transfer problem, *Astrophys. J.* **134**, 469 (1961).
34. D. G. Hummer and G. Rybicki, Computational methods for non-LTE line-transfer problems, in *Methods in Computational Physics*, Vol. 7, Academic, New York, 1967, p. 53.
35. R. Bellman, H. Kagiwada, and R. Kalaba, Invariant imbedding and radiative transfer in spherical shells, *J. Comput. Phys.* **1**, 245 (1966).
36. R. Bellman, H. Kagiwada, R. Kalaba, and S. Ueno, A computational approach to Chandrasekhar's planetary problem, *J. Franklin Inst.* **282**, 330 (1966).
37. F. Bartko and R. H. Hanel, Non-gray equilibrium temperature distributions above the clouds of Venus, *Astrophys. J.* **151**, 365 (1968).
38. L. G. Henyey, J. E. Forbes and N. L. Gould, A new method for the automatic computation of stellar evolution, *Astrophys. J.* **139**, 308 (1964).
39. L. G. Henyey and R. D. Levee, Methods of the automatic computation of stellar evolution, in *Methods in Computational Physics*, Vol. 4, Academic, New York, 1964.
40. R. Kippenhahn, A. Weigert, and E. Hofreiter, Methods for calculating stellar evolution, in *Methods in Computational Physics*, Vol. 7, Academic, New York, 1967, p. 129.
41. R. L. Sears and R. R. Brownlee, Stellar evolution and age determination, in *Stellar Structure* (L. H. Aller and D. B. McLaughlin, eds.), University of Chicago Press, Chicago, 1965, Chap. 11, p. 575.
42. A. Reiz and J. O. Petersen, Calculations of main-sequence stellar models, in *Stellar Evolution* (R. F. Stein and A. G. W. Cameron, eds.), Plenum, New York, 1966, p. 221.
43. M. M. May and R. H. White, Stellar dynamics and gravitational collapse, in *Methods in Computational Physics*, Vol. 7, Academic, New York, 1967, p. 219.
44. I. Iben and R. Ehrlman, The internal structure of middle main sequence stars, *Astrophys. J.* **135**, 770 (1962).
45. I. Iben, Stellar evolution, I. The approach to the main sequence, *Astrophys. J.* **141**, 993 (1965).

46. C. Hayashi, Advanced stages of stellar evolution, in *Stellar Evolution* (R. F. Stein and A. G. W. Cameron, eds.), Plenum, New York, 1966, p. 253.
47. T. Kelsall and B. Stromgren, Calibration of the HR diagram in terms of age and mass for main sequence B and A stars, *Vistas Astron.* **8**, 159 (1968).
48. P. Demarque and J. E. Geisler, Models for red giant stars, I, *Astrophys. J.* **137**, 1102 (1963).
49. F. Hohl and R. W. Hockney, A computer model of disks of stars, *J. Comput. Phys.* **4**, 306 (1969).
50. R. H. Miller, Numerical experiments in collisionless systems, *Astrophys. Space Sci.* **14**, 73 (1971).
51. N. A. Barricelli, O. Havnes, J. Hemphil, and E. Bolviken, A computer study of galactic spiral arms, *Astrophys. Lett.* **12**, 37 (1972).
52. S. Cuperman, A. Harten, and M. Lecar, A phase space boundary integration of the Vlasov equation for collisionless one-dimensional stellar systems, *Astrophys. Space Sci.* **13**, 411 (1971).
53. M. Henon, Monte Carlo models of star clusters, *Astrophys. Space Sci.* **13**, 284 (1971).
54. A. Toomre and J. Toomre, Galactic bridges and tails, *Astrophys. J.* **178**, 623 (1972).
55. J. W. Cooley and J. W. Tukey, An algorithm for the machine calculation of complex Fourier series, *Math. Comput.* **19**, 297 (April 1965).
56. J. Connes, Computing problems in Fourier spectroscopy, in *Aspen Conference on Fourier Spectroscopy*, 1970, p. 83.
57. R. C. Singleton, On computing the fast Fourier transform, *Commun. ACM* **10**, 647 (October 1967).
58. D. D. Proctor and A. P. Linnell, Computer solution of eclipsing-binary light curves by the method of differential corrections, *Astrophys. J. Suppl.* **24**(211), (1972).
59. F. Bartko, Computer Program for the Calculation of the Orbital Elements of Single-Lined Spectroscopic Binaries, M.S. Thesis (unpublished), University of Pittsburgh, 1960.
60. T. D. Kinman, An RR Lyrae star survey with the Lick 20 inch astrograph, II. The calculation of RR Lyrae periods on electronic computer, *Astrophys. J. Suppl.* **11**(100), (1965).
61. R. L. Duncombe, W. J. Klepczynski, and P. K. Seidelmann, Orbit of Neptune and the Mass of Pluto *Astron. J.* **73**, 830 (1968).
62. W. J. Klepczynski, The mass of Jupiter and the motion of four minor planets, *Astron. J.* **74**, 774 (1969).
63. W. J. Klepczynski, P. K. Seidelmann, and R. L. Duncombe, The masses of Saturn and Uranus, *Astron. J.* **75**, 739 (1970).
64. W. J. Klepczynski, P. M. Janiczek, and A. D. Frada, The mass of Jupiter from motion of (76) Freia, *Astron. J.* **76**, 939 (1971).
65. P. K. Seidelmann, R. L. Duncombe, and W. J. Klepczynski, The mass of Neptune and the orbit of Uranus, *Astron. J.* **74**, 776 (1969).
66. W. H. Jeffreys, Automated algebraic manipulation in celestial mechanics, *Commun. ACM* **14**(8), 538 (1971).
67. A. R. LeShack and P. Sconzo, FORMAC language and its application to celestial mechanics —The use of electronic computers for analytical developments in celestial mechanics, a colloquium held by Commission 7 of the IAU in Prague, 28–29 August 1967, *Astron. J.* **73**, 217 (1968).
68. H. G. Walter, Lamé functions of the first kind generated by computer, *Celestial Mech.* **4**, 15 (1971).
69. E. Everhart, Horseshoe and Trojan orbits associated with Jupiter and Saturn, *Astron. J.* **78**, 316 (1973).
70. P. M. Janiczek, P. K. Seidelmann, and R. L. Duncombe, Resonances and encounters in the inner solar system, *Astron. J.* **77**, 764 (1972).
71. P. M. Muller, A user's experiment with sophisticated least-squares software in the discovery of the lunar mass concentrations (mascons), *Math. Software*, p. 57 (1971).
72. P. K. Seidelmann, A dynamical search for a transplutonian planet, *Astron. J.* **76**, 740 (1971).
73. W. Googe, The mathematical implementation of the overlap plate reduction technique, *Astron. J.* **72**, 623 (1967).
74. B. L. Klock, Instrumentation program for an improved meridian circle at the U.S. Naval Observatory, *Astron. J.* **71**, 860 (1966).
75. K. A. Strand, New automatic measuring machine, *Astron. J.* **71**, 873 (1966).

76. R. L. Duncombe, P. K. Seidelmann, and W. J. Kleczynski, Dynamical astronomy of the solar system, *Ann. Rev. Astron. Astrophys.* **11**, 135 (1973).
77. S. P. Maran, Automation and remote control of astronomical telescopes, in *Stellar Astronomy* (H. Y. Chiu, R. L. Warasila, and J. L. Remo, eds.), Gordon and Breach, New York, 1968, Chap. IX-1.
78. S. P. Maran, Instrumentation for automatic telescopes, in *Stellar Astronomy* (H. Y. Chiu, R. L. Warasila, and J. L. Remo, eds.), Gordon and Breach, New York, 1968, Chap. IX-2.
79. T. B. McCord, G. Snellen, and S. Paovola, The MIT automated astrophysical observatory, *Publ. Astron. Soc. Pacific* **84**, 220 (February 1972).
80. S. P. Maran, Telescopes and automation, *Science* **158**, 867 (November 1967).
81. A. D. Code, T. E. Houck, J. F. McNall, R. C. Bless, and C. F. Lillie, Ultraviolet photometry from the orbiting astronomical observatory, I. Instrumentation and operation, *Astrophys. J.* **161**, 377 (1970).
82. H. C. Heacox, Computer Programming for the Orbital Astronomical Observatory, M.S. Thesis, Department of Astronomy, University of Wisconsin, 1970.
83. R. A. Hanel, B. Schlachman, E. Brechan, R. Bywaters, F. Chapman, A. Rhodes, D. Rodgers, and D. Vanous, Mariner 9 Michelson interferometer, *App. Opt.* **11**, 2625 (1972).
84. E. M. Reeves, M. C. E. Huber, G. L. Withbrow, and R. W. Noyes, Real time control of the observing program of an orbiting solar observatory, in *I.A.U. Symposium on New Techniques in Space Astronomy*, D. Reidel, Pordrecht, Holland, 1971, p. 336.
85. T. C. Rindfleisch, J. A. Dunne, H. J. Fresler, W. D. Stromberg, and R. M. Ruiz, Digital processing of the Mariner 6 and 7 pictures, *J. Geophys. Res.* **76**, 394 (1971).
86. W. E. Shufelt, Real-time pictures of Mars by Mariner and by computer, *Comput. Automation* **21**, 7 (June 1972).
87. E. P. Stabler and C. J. Creveling, Spacecraft computers for scientific information systems, *Proc. Inst. Elec. Electron. Eng.*, p. 1734 (December 1969).
88. H. C. Johnson and R. I. Mitchell, A completely digitized multicolor photometer, *Commun. Lunar Planetary Lab.* **1**, 73 (1962).
89. J. D. R. Bahng, A computer interfaced astronomical spectrophotometer, *Decuscope* **9**(5), 2 (1970).
90. W. G. Mankin, R. M. MacQueen, and R. H. Lee, Computer-controlled telescope and interferometer for eclipse observations, *Aspen Conference on Fourier Spectroscopy*, 1970, p. 267.
91. D. C. Wells, The computer-controlled spectrometers at McDonald Observatory, *Publ. Astron. Soc. Pacific* **84**, 203 (February 1972).
92. J. F. McNall, D. E. Michalski, and T. L. Miedoner, The automation of the Wisconsin Echelle spectrograph, *Publ. Astron. Soc. Pacific* **84**, 176 (February 1972).
93. A. Winnberg, A computer-run radio telescope-spectrograph, *Sky and Telescope*, p. 275 (November 1972).
94. S. Sharpless, An application of an electronic computer to photoelectric reductions, in *Astronomical Techniques* (W. Hiltner, ed.), University of Chicago Press, Chicago, 1962, Chap. 9, p. 209.
95. D. Heribson-Evans and N. R. Lomb, Analysis of a variable spectroscopic double star (computer program), *Comput. Phys. Commun.* **2**(6), 368 (1971).
96. D. Heribson-Evans and N. R. Lomb, Derivation of the orbit of a double star from observations made with an intensity interferometer (computer program), *Comput. Phys. Commun.* **2**(2), 59 (1971).
97. R. D. Wills, Application of the Monte Carlo method to the calibration of detectors for gamma-ray astronomy, *Comput. Phys. Commun.* **4**, 51 (1972).
98. W. K. Bonsack, Experiments with digital processing of stellar spectrograms, *Publ. Astron. Soc. Pacific* **83**, 602 (1971).
99. R. H. Stoy, The initial performance of the galaxy machine, in *I.A.U. Colloquium on Proper Motions*, 1970, p. 48.
100. W. J. Luyten, Performance of an automated computerized plate scanner, *Proc. Nat. Acad. Sci.* **68**(3), 513 (March 1971).

101. R. Albrecht, P. Boyce, and J. Chastain, The Lowell Observatory data system, *Publ. Astron. Soc. Pacific* **83** (1971).
102. G. J. Carpenter, Automated methods in optical astronomy, *IERE*, p. 327 (1970).
103. B. G. Clark, Information processing systems in radio astronomy and astronomy, *Ann. Rev. Astron. Astrophys.* **8**, 115 (1970).
104. E. W. Dennison, Recent computer installations at the Hale Observatories, in *Proceedings of the 11th Colloquium of the I.A.U., Automation in Optical Astrophysics*, August 1970, p. 8.
105. J. Hameen-Anttila, An inexpensive astronomical control computer system, *Publ. Astron. Soc. Pacific* **84**, 185 (1972).
106. V. C. Reddish, Twin 16 inch photometric reflectors at Edinburgh, *Sky and Telescope*, p. 124 (September 1966).
107. V. C. Reddish, Instrumentation in optical astronomy, *Nature, Phys. Sci.* **229**, 37 (January 11 1971).
108. F. E. Heart and A. A. Mathiasen, Computer control of the Haystack antenna, *Proc. Inst. Elec. Electron. Eng.*, p. 1742 (December 1966).
109. R. E. Nather, An astronomical interface for a small computer, *Rev. Sci. Instr.* **43**(7), 1012 (July 1972).
110. R. J. Clayton, B. A. Delazi, and R. Elia-Shaoul, Evolution breeds mini-computers that can take on its big brothers, *Electronics* **44**, 62 (1972).
111. D. E. Trumbo, Telcom versatile real-time instrument control system, *Publ. Astron. Soc. Pacific* **83**, 608 (1971).
112. R. W. Grutzner, *INTAP: Interplanetary Analysis Program*, NASA-CR-109777, 1970.
113. J. D. Erickson, P. Tanner, and C. Fujii, *An Advanced Automatic Telescope Computer Control Program*, University of Michigan, Rept. No. 1386-27-T, 1970.
114. R. J. Messina, Instructional uses of the computer: Laboratory exercise on eclipsing binaries, *Amer. J. Phys.* **40**(5), 760 (1972).
115. S. W. Galley, Computer-animated film of hydrogen-wave functions, *Amer. J. Phys.* **35**, 984 (1970).
116. R. Blum and A. M. Bork, Computers in physics curriculum, *Amer. J. Phys.* **38**, 959 (1970).
117. G. S. Hawkins, Stonehenge: A neolithic computer, *Nature* **202**, 1258 (1964).
118. G. S. Hawkins, Astro-archeology, *Vistas Astron.* **10**, 45 (1968).
119. F. Hoyle, Stonehenge—An eclipse predictor, *Nature* **211**, 454 (1966).
120. O. Gingerich, The computer versus Kepler, *Amer. Sci.* **52**, 218 (1964).
121. O. Gingerich, Applications of high speed computers to the history of astronomy, *Vistas Astron.* **9**, 229 (1968).
122. A. B. Underhill, A program for computing the theoretical spectrum from a model atmosphere, *Publ. Dominion Astrophys. Obs.* **11**(24), 467 (1962).

Frank Bartko

# ATOMIC ENERGY COMMISSION (ENERGY RESEARCH AND DEVELOPMENT ADMINISTRATION)

## HISTORICAL DEVELOPMENT

### Pre-Atomic Energy Commission

The Atomic Energy Commission (AEC) officially assumed responsibility for the nation's atomic energy program in January 1947, in accordance with the Atomic Energy Act of 1946. Prior to this time, atomic energy matters were handled by the United States Army. As is well known now, the Manhattan Project of the early 1940s was successful in bringing off the world's first successful nuclear reactors and nuclear explosives. A number of the present day national laboratories or facilities supported by AEC were created in an effort to speed and foster development in these areas. Oak Ridge, Argonne, Hanford, and Los Alamos are now well known for their efforts in the nuclear energy field. When the commission took over the reins, it also assumed responsibility for many of these sites and set up new ones such as Knolls Atomic Power Laboratory at Schenectady to carry on the work in the energy field.

During this very same period of time, the electronic digital computer was going through its birth pains. Because of its need for firing tables, the Ballistic Research Laboratory of the Aberdeen Proving Ground in Maryland was supporting the development of analog computers during the early years of World War II. Through the fortunate involvement of certain key people in the effort, by 1943 a proposal to build an electronic high-speed computer was made and accepted. It was dedicated in February 1946. This computer, the ENIAC (Electronic Numerical Integrator and Computer) was to provide the U.S. Government (including the AEC) with valuable computational results, but even more important, it was to stimulate the further development of electronic computers in a big way.

During the war period, many of our most talented scientists were involved in both the energy and computer developments. Notably among them was John von Neumann who, knowing the demands the scientists could put on high-speed computers and being highly motivated in promoting computer design, was able to link the two and watch both grow. At this point in time, industry did not seem interested in putting huge sums of money into computer development. Von Neumann succeeded in inaugurating a project to advance the state-of-the-art in computing design. With the assistance of the government, a project was started (1946-1947) to construct a fully automatic, general purpose, high-speed computer at the Institute for Advanced Study at Princeton University. This became the prototype for many systems to be funded by the AEC for nuclear energy research. The Princeton project was supported in part by the AEC, but also by the Office of Naval Research and the Office of Air Research.

During the war the need for computation within the atomic bomb development program was great. Through people like von Neumann and other scientists who were already aware of the existence of mechanical calculators, a number of IBM electro-mechanical machines were acquired and put to work. The calculations were tedious and long, but as can be gathered from the successful conclusion of the Manhattan Project, were also highly successful. The "large"-scale computers of the 1940s were relatively unavailable until after the war, arriving on the scene at the same time as the AEC.

Two major problem types arose during the design and construction of both nuclear reactors and nuclear explosives that required extensive computation. These were the neutron transport problem and the implosion problem.

The behavior and distribution of neutrons in a reactor core is the fundamental information one must have to design reactors. The mathematical theory to determine such behavior exists, but analytic solutions are not possible in the needed configurations. Approximations may be made to put the mathematical equations into appropriate form to make solutions feasible. Even with approximations, the geometry is often so complex that the then available desk calculators were not adequate to solve the problems in a reasonable finite amount of time. The physicists were ingenious enough to simplify some of the calculations, but the computer soon became an essential ingredient to make realistic calculations.

The implosion method of assembling nuclear devices early became important. Several calculational problems were involved. First, the equation of state data for uranium and plutonium at very high pressures (where experimental data did not exist and theoretical calculation had never been done) was needed, and second, the calculations of shock waves through the various materials making up the bomb were very difficult. Techniques worked out by Metropolis, Teller, Feynman, and Courant, among others, made possible some of the computational processes using the only machines available at that time. The scientists performed some calculations by machine, others by hand—there was nothing automatic about the procedures. The first implosion calculation took 3 months to perform and perhaps no more than a dozen such calculations were completed before war's end. (Such calculations today take seconds.)

### **Early Computers (1947–1952) Used by Atomic Energy Commission Facilities**

The first of the larger machines available for use by the AEC were Mark I, ENIAC, IBMs SSEC, SEAC, and, of course, the early IBM electromechanical 602A, 604, and later the CPC.

#### ***Mark I***

The Harvard Mark I (Automatic Sequence Controller Calculator) was completed in 1944 by IBM and presented to Harvard University. Input and output was accomplished through the use of punched cards. Output could also be printed on an electric typewriter. Execution times for add and multiply were 0.3 and 6.0 sec, respectively.

These parameters were for many years used as criteria to evaluate computer performance for AEC computer systems. It was a decimal machine and had a word length of 24 decimal digits. Although the construction of this computer was not supported by AEC, some money was made available in 1950 for extending its operation to a 12-hr day. The problems run on the computer were selected by a committee of Harvard scientists. Among them was the problem of computing nuclear forces between protons and neutrons, a problem of major interest to the AEC nuclear scientists.

### **ENIAC**

ENIAC was ready for use in 1947. It also was a decimal machine, with a 10-digit word length. Add and multiply times were 200  $\mu$ sec and 2.8 msec, respectively. Input and output were provided on punched cards. In 1950 when the fusion bomb (popularly called the H-bomb) was being designed, it was essential that a high-speed computer be made available for the difficult problem of determining whether a fusion reaction, once started, could continue. ENIAC was the only machine then available that could cope with the problem. The computation was performed with difficulty, with rather unsatisfactory results. Even today such problems severely put to task our most capable computers.

### **SSEC**

SSEC (Selective Sequence Electronic Calculator) was the first attempt by IBM at building a large-scale electronic computer. It was completed in 1948 and dismantled in 1952. It was one hundred times as fast as the Mark I and had some new interesting features such as conditional transfers. The AEC used it for a period of 6 months on one large calculation.

### **SEAC**

SEAC (Standards Eastern Automatic Computer) was built at the National Bureau of Standards. It was put into operation in 1950 and heavily used by AEC in many of its weapons calculations for several years. It had a small internal memory (mercury delay line and Williams electrostatic tube) and used magnetic tape for external memory. It was capable of performing adds and multiplies in 0.2 and 2.4 msec, respectively (using electrostatic memory).

### *Electromechanical Computers*

The IBM electromechanical equipment received much use in many of the AEC laboratories from the time they became available until replaced by the first generation electronic equipment. The most advanced of these was the CPC (Card-Programmed Calculator) introduced in 1949. It was used in nuclear reactor and weapon development as well as in theoretical nuclear physics research. It had a very small internal vacuum tube store, was capable of a 480  $\mu$ sec add time and 2.4–24 msec multiply time. Problems were done in a man-machine loop, the punched cards results of one run being fed into the machine manually as the input for the next run.

*AEC Computer Construction Program*

The major efforts of the AEC to achieve fully automatic general purpose computers involved an in-house construction program. As mentioned earlier, AEC supported in part the development of the so-called IAS (Institute for Advanced Study) computer at Princeton. A number of modified versions of this prototype were scheduled for construction at several laboratories and universities under sponsorship of the AEC. The justification was that "the AEC laboratories will have at their disposal the most modern means of scientific computation." Argonne National Laboratory (ANL), Los Alamos Scientific Laboratory (LASL), and the University of Illinois were the first institutions to get under way in this effort. ANL built AVIDAC (Argonne's version of the IAS Digital Automatic Computer) for its own use and ORACLE (Oak Ridge Automatic Computer and Logical Engine) in collaboration with personnel at Oak Ridge for use at Oak Ridge National Laboratory (ORNL). Los Alamos produced MANIAC I (Mathematical Analyzer, Numerical Integrator, and Computer) for its own computational purposes. The University of Illinois also built two systems, the ILLIAC (Illinois Automatic Computer) for research use at the university and ORDVAC (Ordnance Discrete Variable Automatic Computer) to be used at Aberdeen Proving Ground. The characteristics of these computer systems are shown in Table 1.

**First Generation Commercial Computers (1952-1960)**

In 1952 the AEC purchased two of the first commercially produced, general purpose, fully automatic digital computers, Univac I. Although started by Eckert and Mauchly in the late 1940s under contract with the Department of Commerce for installation at the Census Bureau, the Univac was not completed until the early 1950s. By that time Eckert and Mauchly's company had been acquired by Remington Rand (now Sperry Rand). The two AEC-acquired machines were installed at Livermore and New York University in 1953. To further the development of fusion weapons, the AEC had set up a second weapons laboratory (the first being LASL) at Livermore to be operated by the University of California. It became known as the University of California Radiation Laboratory, but has had several name changes, the most recent being to Lawrence Livermore Laboratory. The Univac I system was dedicated to weapons computations. The system installed at NYU was placed at the Courant Institute for use on a variety of AEC projects as well as to further mathematical research in the solution of complex partial differential equation by the mathematicians at the institute. The AEC specifically stated that 10% of its time was to be used for basic research. To review the AEC computer requirements and apportion time on this computer, an AEC computer council chaired by Edward Teller was set up in January 1954. Project Matterhorn at Princeton was at this time involved in fusion research and became a heavy user of the NYU computer. Computer time was so scarce during the early 1950s that much time was bought for weapons development at other Univac sites, such as those at the Office of the Air Comptroller, USAF, and Army Map Service in Washington, D.C.

**Table 1**  
Characteristics of Computers Constructed by the Atomic Energy Commission, 1947-1952

Computer	Use began	Number base	Word length (bits)	Memory size (words)	Memory method	Intermediate storage	Add time (fixed, $\mu$ sec)	Multiply time (fixed, $\mu$ sec)	Input-output method	Vacuum tube count
IAS	1/52	2	40	1024	Electro-static	Magnetic drum	62	710	IBM cards	2300
MANIAC-1	3/52	2	40	1024	Electro-static	Magnetic drum	60	950	Printer; paper tape	2500
ORDVAC	3/52	2	40	1024	Electro-static	(None)	72	730	IBM cards paper tape	2700
ILLIAC	9/52	2	40	1024	Electro-static	(None)	72	730	Teletype, paper tape	2774
AVIDAC	3/53	2	40	1024	Electro-static	Magnetic tape	30	510	Teletype	2800
ORACLE	6/53	2	40	1024	Electro-static	Magnetic tape	38	510	Teletype	3500

It was learned rather early in the game that Univac I had too small a memory and was too slow to cope with the formidable problems encountered in both the transport and shock hydro-dynamics problems. Because of its serial memory, using mercury delay lines, the Univac was an order of magnitude slower than the IAS type of computer under construction at that time. Fortunately, both IBM and Remington Rand (having acquired the Engineering Research Associates) were building computer systems with parallel readout electrostatic memories at the same time. IBM's system, the 701, was ready for delivery by April 1953; the ERA 1103 by the beginning of 1954. These were very competitive machines, but because of its earlier availability of the IBM 701, AEC acquired two (out of the total of 19 built), one to be located at Los Alamos and the other at Livermore.

At this time the weapons development program was heavily committed to the design of fusion bombs. The complexities of the calculations were such that only with the most capable computer systems could many of the problems be successfully attacked. For this reason the AEC supported both LASL and LLL in their attempts to stay at the forefront of computer development. MANIAC I, Univac I, and the IBM 701s were operated around the clock, but more capability had to be obtained.

In 1954, IBM came up with its response to AEC's interest in more advanced systems. The 704 was proposed. It was faster than the 701 by a factor of at least 2, had both greater capacity and more reliable memory (magnetic cores instead of electrostatic tubes), built in floating point, and a CRT output device. Although this represented a great step forward in design, it actually fell far short of providing the needed speeds to perform the weapons calculations in reasonable times. To help encourage state-of-the-art computer development, the AEC solicited proposals for more capability. The main contenders for this development were Remington Rand and IBM. In actual fact, both were selected, first Remington in 1955 to build the LARC (Livermore Advanced Research Computer) for LLL and later, in 1956, IBM to build the STRETCH (Stretching the State-of-the-Art) for LASL. These computers were the first of the second generation machines to be put under contract. The LARC had to use some vacuum tubes for power; the STRETCH was completely transistorized. Characteristics of these machines are described in Table 2. Even though these computers were not delivered until 1960, they are compared with the workhorse of the late 1950s, the IBM 704. From the manufacturer's point of view, both systems were failures because only limited numbers were marketed—two by Remington Rand and eight by IBM. For the laboratories they were huge successes, even though delivered quite late. By the early 1960s other second generation systems were being produced, but with lesser performance than LARC or STRETCH. In addition to the LARC, LLL also acquired a STRETCH to help support its heavy computing requirements.

During the last half of the 1950 decade, the use of computers really accelerated within the AEC facilities. The basic physics research program required high-speed computers to process data acquired from bubble chambers; nuclear reactor development was in high gear; and, as mentioned previously, the weapon laboratories were pushing the state-of-the-art in computer development to achieve their goals. Starting in 1956, both LASL and LLL acquired multiple IBM 704 systems, the Naval Reactor Branch of the AEC supported similar systems at both Knolls Atomic Power Labora-

**Table 2**  
Characteristic of Commercially-Produced Computers within the Atomic Energy Commission, 1952-1960

Computer	Use began	Number base	Word length (bits)	Memory size (words)	Memory method	Add time (floating, $\mu$ sec)	Multiply time (floating, $\mu$ sec)
RR UNIVAC-1	1953	10	12 digits	1,000	Mercury delay tubes	525 <sup>a</sup>	2150 <sup>a</sup>
IBM 701	1953	2	36	2,048	Electrostatic	60 <sup>a</sup>	456 <sup>a</sup>
IBM 704	1956	2	36	32,768	Magnetic cores	84	204
IBM 709	1958	2	36	32,768	Magnetic cores	77	170
RR LARC	1960	10	12 digits	30,000	Magnetic cores	4	8
IBM 7030	1960	2	64 (plus 8 parity bits)	98,304	Magnetic cores	1.4	2.7
IBM 7090	1960	2	36	32,768	Magnetic cores	14	24

<sup>a</sup> Fixed-point time.

tory and the Westinghouse Bettis Atomic Power Division for shipboard reactor development. ANL and Oak Ridge each purchased IBM 704s for reactor work as well as for handling production facilities. For basic research, similar systems were installed at the University of California Radiation Laboratory at Berkeley (now called the Lawrence Berkeley Laboratory), New York University, and Midwest Universities Research Association (Madison, Wisconsin). By 1959, 19 IBM 70Xs were installed in various AEC laboratories. Several of these were for business data processing.

Development of STRETCH led to the introduction of the IBM 7090. Some of the modules designed for the larger system were used in constructing the IBM 7090 and provided a great improvement in performance ( $\times 6$ ) and reliability over the IBM 704 and 709 (the 709 had some minor improvements over the 704). This was IBM's real commercial entry into the second generation of computers, the transistorized set. Other manufacturers were in various stages of design of transistorized machines. For example, Control Data Corporation, which was organized in 1957, was building a CDC 1604; Philco Corporation was building a Transac S-2000. Both of these turned out to be of interest to AEC facilities in the early 1960s.

Some laboratories were still building computers. It was thought that substantial costs savings could be made in a construction program, but also, and more important, that better performance and much needed specialized features were possible. Los Alamos started its MANIAC II, and ANL its GEORGE computer.

### Atomic Energy Commission Computers after 1960

The 1960s saw the huge growth in second generation systems as well as the announcement and introduction of the integrated circuit systems, the third generation. Most of the facilities with vacuum tube machines traded them up to transistorized systems. These were very cost effective in that one obtained approximately 6 times the performance for roughly the same dollars. Most AEC systems at that time were leased, hence the switch could be made rather easily. Between 1960 and 1965, Control Data not only sold many 1604s but also introduced the 3600 which some of the laboratories acquired. IBM introduced an upgrade to the 7090, calling it a 7094. Again, most 7090 installations took advantage of the improvements. Univac introduced its second generation system, the 1107, and Philco made some improvements in the 2000 system. Several of the laboratories broke away from the patterns of buying mostly from IBM and acquired the Philco systems (KAPL and Bettis). These systems, in their final upgrade, were in the same class as STRETCH, at least a binary order of magnitude better on performance than those others mentioned above for the number-crunching activities of some of the AEC laboratories.

The marketing failures of LARC and STRETCH soured Univac (Remington Rand was now the Univac Division of Sperry Rand) and IBM on the idea of staying in the forefront of supercomputer development. Control Data Corporation (and for a time it seemed that Philco might also step into that role) took up the challenge and designed the CDC 6600. Stimulated by an AEC contract, CDC produced the first one for LLL, delivering it in 1964. Strangely, although most manufacturers were talking third generation and IBM was announcing its third generation 360 system at this time, the

6600 was a completely discrete component second generation. In performance, however, it was advanced beyond anything announced at the time of delivery. Not only was its CPU performance outstanding for the weapons type of jobs, but also the 128K memory made large jobs easier to cope with.

During the last five years of the 1960s the AEC facilities converted to third generation machines. Several selected IBM; two (Stanford Linear Accelerator Center and Oak Ridge) selected the top of the IBM line, 360/91s. These actually were introduced several years after the CDC 6600 and were somewhat superior in performance. IBM had limited production to approximately 15 systems, and the 360/91 was not formally announced as part of the 360 family. Several other large scale 360/65s or 75s were added. The larger number of facilities switched to the CDC 6600 at this time. It was very heavily used for nuclear reactor and weapons development as well as for high-energy physics bubble chamber data processing.

During these years the AEC Division of Research was still supporting computer construction. MANIAC III was built at the University of Chicago, MERLIN at Brookhaven National Laboratory, ILLIAC II at the University of Illinois, and the Rice Computer at Rice University. MANIAC II at Los Alamos was undergoing continual upgrading. Again these were constructed because of special features and cost effectiveness of in-house design and construction. For example, MANIAC III was the first computer to incorporate significant digit arithmetic.

In the later half of the decade, another factor of approximately five (over CDC 6600) in computer power became available through the introduction of first the CDC 7600 and later the IBM 360/195. These are still (1975) being added to AEC's stable of systems, normally those with 6600s switching to 7600s and those with IBM equipment switching to the more powerful IBM equipment.

By the mid-sixties the feasibility of constructing parallel processors (to gain significant factors of speed improvement) had been demonstrated through the SOLOMON model. AEC, primarily through the interests of LLL, tried unsuccessfully to have such a computer constructed to test its capability on large-scale problems. (The concept was finally adopted for the construction of ILLIAC IV under ARPA funding.) The AEC interest in this computer led some of the major computer manufacturers to think of greater parallelism in computers. After a solicitation of proposals from these manufacturers, a contract was awarded to CDC to construct a STAR (STring ARay) computer system for delivery at LLL. This computer provided its parallelism through a "pipeline" structure allowing for the streaming of operands through the arithmetic unit at high speeds. The first two STAR computer systems were delivered to LLL late in 1974, several years late.

Virtually all of the computers within AEC facilities are acquired through local negotiations and contracts. In 1972, in an attempt to obtain better pricing, the AEC decided upon a multiple acquisition for seven facility systems. The awards were split between IBM and CDC, involving 360/195s, 360/165s, CDC 7600, and CDC STARS. In the upgrading process, some of the lesser computing systems are being moved among the laboratories to satisfy other needs within the AEC family. The total number of large systems installed within AEC facilities is not growing as rapidly as in the early days of computing, but upgrading to some powerful systems continues. The demands

for greater capability are ever present; the rate at which the manufacturers introduce improved systems tends to determine the rate of acquisition. Of course, at some of the facilities, multiple systems exist primarily because the capability of single systems is not growing fast enough.

Obviously, the stress within AEC has been for the high-speed, large-capacity system. Many other systems have made their appearance at AEC-supported facilities, usually to support experimental data gathering and control. In the chemistry laboratory, in numerical control of machine tools, in high-energy physics scanning devices, in engineering measuring devices, in particle acceleration control systems, etc., one finds these smaller systems as highly cost effective components. Even in computer networks the small (in this case the mini-computer) has found its place. As seen in Fig. 3, the growth of small computers in the AEC has been tremendous over the last 10 years.

The computerized management of AEC activities at the various facilities is also a rapidly growing phenomenon. Each facility has its own means of providing data processing, either on central facility machines or smaller stand-alone systems. AEC headquarters itself has started its computerization process and has now a Management Information and Telecommunication Systems Division. Within this division, which runs a large 370/165 computer system, a highly sophisticated management system is currently being implemented.

### Special Projects Related to Computers

#### *High-Energy Physics*

During the late 1950s and early 1960s, bubble chamber data was starting to accumulate in large volume. This consisted of particle track photographs taken in stereo on 105-mm film. The information on these tracks had to be scanned and digitized for processing by computer. Many devices to accomplish this task were designed and integrated into computer systems to provide as complete automation as possible. One of the computer efforts supported by AEC was the construction of ILLIAC III at the University of Illinois. It was believed that a special purpose pattern recognition computer could greatly enhance performance in analyzing bubble chamber data. This system was never actually used for such purposes.

#### *Computer Peripherals*

Because of the unique needs of the AEC laboratories for high-speed computation and the large volumes of data to be processed, great pressures were exerted on manufacturers of computer equipment to provide larger and larger memory systems. Univac LARC had a capability of handling up to 100,000 words of core, and IBM STRETCH could support up to 256K words. These memories were so expensive that no attempt was made to equip the systems with the full complement of core, but the pattern was set. The CDC 6600 standard memory became 128K words. Most manufacturers now provide large central memories as well as extended core or bulk storage which can be added on.

Drum and disk drives also were pushed to higher performance and greater storage capacity by AEC requirements. The first trillion bit file, the photo digital storage device was built by IBM for the Lawrence Livermore Laboratory. The second went into operation in support of high energy physics research program at the Lawrence Berkeley Laboratory.

Need for other peripheral devices such as on-line display systems and terminals also had some effect on the production of such devices, but to a much lesser extent.

### *Networks*

Working with several large computers at each site led to the notion of internal networks. LLL in 1964 started its OCTOPUS network which was designed to tie its large facilities together to share common data base files. This has grown over the years to become one of the largest networks in the country. Other laboratories, such as Brookhaven National Laboratory, have on-line real time needs which it is satisfying through the netting of its major systems with other data collecting systems (BROOKNET). Many of the AEC laboratories are currently in various stages of tying systems into networks. There is no doubt that new ideas will arise and that these will be passed on throughout the computer community.

### *Software*

Each of the major sites has contributed significantly to software development. CDC 6600/7600 time-sharing systems were developed at LLL and NYU; language developments, such as those at LASL (MADCAP), NYU (SETL), and Argonne (SPEAKEASY), are going on and some are gaining popularity; operating systems with innovations are being worked on at many facilities; network programming is all being done in-house. In all areas of software applications, AEC is contributing substantially. Appreciable manpower in numerical techniques is also being invested to help solve the formidable mathematical problems encountered in AEC projects.

## SUMMARY AND PROSPECTS

During the past 25 years, the total computing power within the AEC-supported facilities has risen at a rate approaching a doubling each year. Figure 1 shows the growth since 1959 for large computers (defined as systems costing more than \$1 million each). The units used are shown on the left scale as Univac equivalents and on the right scale as CDC 6600 equivalents. Each large computer was rather arbitrarily assigned a performance factor based on capability to do AEC jobs in the various laboratories. There is a turn down in power evident during the last few years. This will probably continue, not because of lack of need, but rather because of the relatively

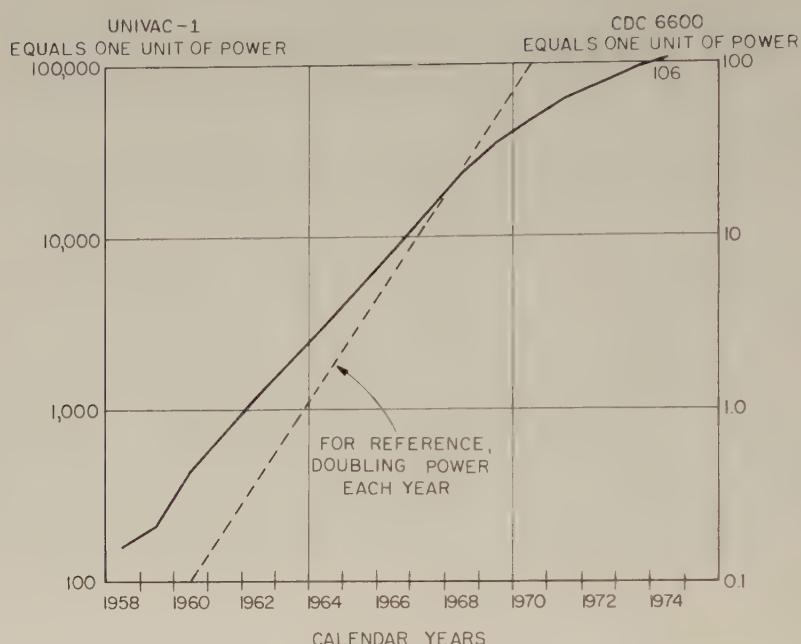


Fig. 1. Total computing power of large computers in the AEC (computers costing more than \$1 million).

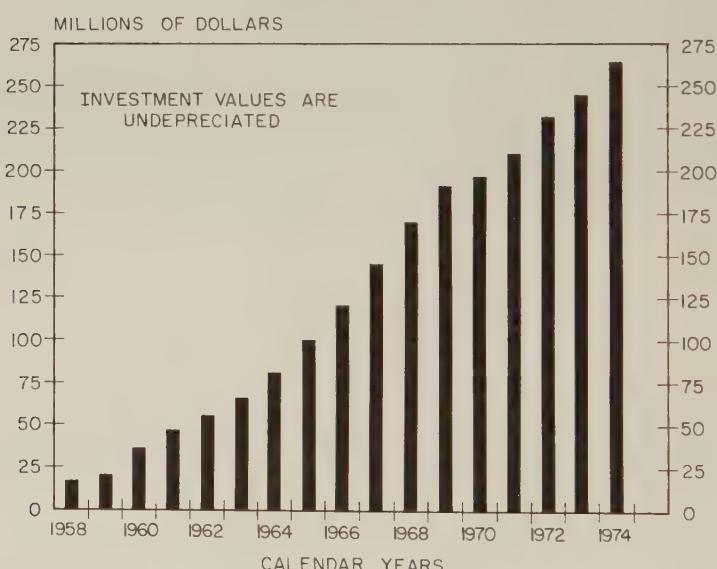


Fig. 2. Capital investment in large computers in the AEC (computers costing more than \$1 million).

small improvements from generation to generation, the simultaneous increase in cost per system, and the slowdown in budget expenditures for capital equipment. Figure 2 graphically demonstrates the turndown in AEC expenditures for large systems.

These large systems today are being used much in the same way as in the beginning. Calculations have been more sophisticated, problems being attacked are formidable, and use is spreading. The energy laboratories are solving very large systems of differential equations encountered in hydrodynamics and particle transport; the high-energy physics facilities are confronted with more and more bubble chamber and spark chamber analyses, other nuclear physicists and chemists are extending their use of systems into all aspects of scientific research, controlled thermonuclear reaction physicists are tackling multidimensional plasma physics problems; facility and production management is requiring greater and greater power; and general engineering design and evaluation are showing increasing demands on large systems. Newer projects such as laser fusion have not as yet made their move into large-scale computing but doubtlessly will in the near future.

The AEC is engaged in a gradual upgrading program for its computer facilities. From year to year, depending on both the needs of each facility and availability of improved capability, more powerful computers are replacing lesser ones, the latter being moved to satisfy other growing needs. Figure 3 shows the gradual increase in the number of large computers within the AEC family. The real revolution in computing is taking place at the small or minicomputer level. The realization of the cost effectiveness of using minicomputers in data acquisition and control systems is bringing these

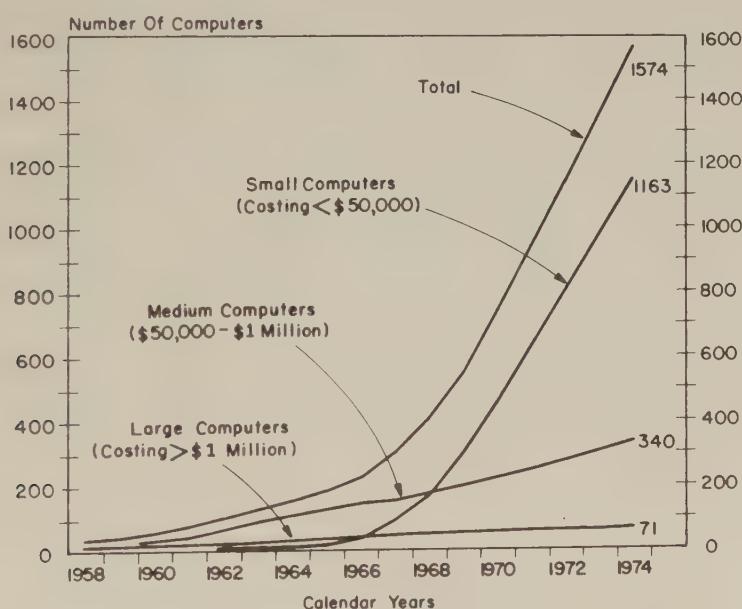


Fig. 3. Number of computers in the AEC.

into use in exceedingly large quantities. In the past 8 years the number of such systems has increased by a factor of approximately 20 and is, at this time, also 20 times greater than the number of large systems. Figure 3 also shows the number of computers costing less than \$50,000 each installed at AEC sites. The prices for this class of computer are dropping sharply, and it is expected that the rise shown will continue through the indefinite future.

The Division of Thermonuclear Research (CTR) of the AEC made several studies in 1973 to determine computer needs to meet the program schedule developed for its Magnetic Confinement Systems program. It was concluded that several IBM 360/195 or CDC 7600 class systems would be needed by 1975 and several even more sophisticated systems by 1980. To meet these needs, a National CTR Computer Center was established at the Lawrence Livermore Laboratory in 1974 with initial satellite stations at the CTR-supported research facilities at LLL, Los Alamos Scientific Laboratory (LASL), Princeton Plasma Physics Laboratory (PPL), and Oak Ridge National Laboratory (ORNL). These were to be tied into the large central facility by high speed dedicated telephone lines. As of the time of this writing, the selection of equipment has not been made.

In January 1975 AEC quietly went out of existence and was replaced by the Energy Research and Development Administration (ERDA).

## BIBLIOGRAPHY

Annual Reports of the AEC-supported laboratories.

Fernbach, S., and A. Taub, *Computers and Their Role in the Physical Sciences*, Gordon and Breach, New York, 1970.

Greenspan, H., C. N. Kelber, and D. Okrent, *Computing Methods in Reactor Physics*, Gordon and Breach, New York, 1968.

Hewlett, R. G., and F. Duncan, *Atomic Shield 1947/1952. Vol. II. A History of the USAEC*, Pennsylvania State Univ. Press, University Park, Penn., 1969.

*Inventory and Cost Data Concerning the Utilization of Automatic Data Processing (ADP) Equipment in the Federal Government for Fiscal Years 1959, 1960 and 1961*, Executive Office of the President, Bureau of the Budget, Washington, D.C.

*Inventory of Automatic Data Processing Equipment in the Federal Government*, Executive Office of the President, Bureau of the Budget, Washington, D.C., August 1962.

*Inventory of Automatic Data Processing Equipment in the Federal Government*, Executive Office of the President, Washington, D.C., July 1965.

*Inventory of Automatic Data Processing Equipment in the Federal Government*, Executive Office of the President, Washington, D.C., July 1966.

*Inventory of Automatic Data Processing in the U.S. Government*, General Services Administration and Federal Supply Service, Washington, D.C., 1969.

*Inventory of Automatic Data Processing in the U.S. Government*, General Service Administration and Federal Supply Service, Washington, D.C., 1970.

*Inventory of Automatic Data Processing in the U.S. Government*, General Service Administration and Federal Supply Service, Washington, D.C., 1971.

*Semi-Annual Reports of the AEC*, Washington, D.C., particularly Volumes 7, 13, 14, 15, 24.

Smith, A. E., *A Survey of Large-Scale Computers and Computer Projects*, Office of Naval Research, Dept. of the Navy, Washington, D.C., Revised July 1950.

Wiek, Martin H., *A Survey of Domestic Electronic Digital Computer Systems*, Ballistics Research Laboratories, Aberdeen Proving Ground, Maryland, Report No. 971, December 1955.

- Wiek, Martin H., *A Second Survey of Domestic Electronic Digital Computing Systems*, Ballistic Research Laboratories, Aberdeen Proving Ground, Maryland, Report No. 1010, June 1957.
- Wiek, Martin H., *A Third Survey of Domestic Electronic Digital Computer Systems*, Ballistics Research Laboratories, Aberdeen Proving Ground, Maryland, Report No. 1115, March 1961.
- Wiek, Martin H., *A Fourth Survey of Domestic Electronic Digital Computing Systems*, Ballistic Research Laboratories, Aberdeen Proving Ground, Maryland, Report No. 1227, January 1964.

Sidney Fernbach

## AUDITION

In 1969, Schroeder [1] reviewed the first 10 years of the use of computers in acoustics. His review covered most areas of acoustics from speech processing and the psychological and physiological aspects of audition, to the design of public auditoriums and acoustic image processing. In addition to his own wide range of interests, Schroeder's position with the Bell Telephone Laboratories gave him first-hand knowledge of most of the applications unique to acoustics, since only such a large, dedicated facility had the resources to acquire and maintain the computers available in the early 1960s.

As in other fields, two innovations have led to the widespread use of computers in acoustics. First, the introduction and rapid development of minicomputers has led to their availability to all but the most modest of facilities. Second, the advent of time-sharing of the mainframe by a large number of users at remote locations has made possible applications which would be uneconomical for a single user of a large machine.

In this paper we will concentrate upon those applications which are unique to audition. We will emphasize the role that the small laboratory computer has played in furthering research in psychoacoustics and auditory physiology, and their potential for application in the diagnostic and remedial aspects of audiology. We must not, however, neglect the importance of the large digital computer.

Most large computer applications in audition are of a conventional nature. The auditory researcher or clinician is simply a consumer of computer services for statistical analysis of experimental data or the storage of clinical records. Standardized programs for factor analysis, an analysis of variance, curve-fitting, etc., available at any computer center are, of course, applicable to the analysis of the results of auditory experiments. Additional programs specialized for the analysis of behavioral data are available in standard languages [2, 3].

An area of applications in which little specialization is required for immediate application in audition is computer-aided instruction (CAI). Of course, for teaching university or high school students the fundamentals of acoustics, anatomy, and

physiology of human communications, or introductory psychoacoustics, special programmed materials must be prepared. However, the education of deaf students should reap major benefits from CAI. Since interactions with the computer depend only upon vision, the deaf student is not at as great a disadvantage with CAI as he would be in the conventional classroom. Recent reports by Barnes and Finkelstien [4] and Suppes [5] indicate success in teaching mathematics, English grammar, chemistry, computer programming, and logic to deaf students. For as long as computers are unable to "talk," deaf education should benefit from each innovation introduced in CAI.

## SPECIALIZED APPLICATIONS

An experimental technique that would be impractical to use without the aid of a large digital computer is multidimensional scaling (MDS) of perceptual data. Multi-dimensional scaling methods are based upon mathematical models of human performance in decision-making tasks. An observer is asked to assess the similarity of combinations of stimuli which differ in several attributes. Each attribute itself is measured along a single continuum, that is, it may be unidimensional. For sounds, simple attributes such as pitch, loudness, and duration come quickly to mind. Others, such as timbre, vibrato (modulation), and volume have also been scaled. Sounds created in the laboratory for investigations of the auditory system often vary only in one attribute, pitch or loudness, perhaps, while those occurring in nature usually differ in several.

The assumptions upon which MDS methods are based are simply that: (1) psychological magnitudes may, in general, be represented as points in a space of several dimensions; and (2) the similarity of two stimuli is inversely proportional to the distance between their loci in the multi-dimensional space. Several experimental paradigms exist for assessing the similarity of sample pairs or triplets. Once similarity judgments have been made, the results are used as data for input into one of several existing MDS computer programs. These programs minimize the number of dimensions required to represent the data while also minimizing the "stress," a sort of "badness of fit" statistic.

Given a geometrical representation of the data, it is often possible to correlate the axes of the space with stimulus attributes. Some programs also permit transformation of the input data or rotations of the geometrical representation which may further clarify the relationship between spatial representation and stimulus attribute. The details of these methods are available in a recent two volume text by Shepard *et al.* [6].

These MDS techniques, which were first applied to auditory perception (see Schroeder for references), have recently been applied to a number of problems in the behavioral and social sciences [6]. They are, however, still finding new applications in audition [7, 8] and in speech perception [9-13].

Models of the ear have existed since man first began to investigate his sense of hearing. As knowledge of the fine structure of the inner ear has accumulated, the models offered to describe auditory functions have become more complex. Mechanical action

of inner ear structures in response to acoustical input has been of primary interest. Since analytical solutions for wave motion in the complex geometry of the cochlea are difficult to derive, computer simulations have been of great value in understanding this first stage of frequency analysis of the acoustical signal [14-18].

Recently, stochastic models have been developed for nerve firing patterns in the auditory nerve [19, 20] as well as the entire system [21-26]. These models are based upon physiological and behavioral findings rather than upon approximations to anatomical structures. They serve a dual purpose: to summarize current knowledge of auditory neurophysiology and to suggest psychoacoustical tests of physiological theories. In principle, the vast interest in neural network analysis could be applied to auditory system performance.

The impact of computer technology on audition has been most spectacular with the introduction of the minicomputer into physiological and behavioral laboratories. Their low cost and high degree of flexibility permit each researcher to tailor his installation to suit his own needs. A small computer may simply act as a passive data collector, or it may control the progress of the experiment. It may be used to control conventional signal generators or, through one or more D/A converters, it may be the only signal source.

The number of minicomputer installations in laboratories of auditory physiology or psychoacoustics is too great to permit a detailed description of each one. We must, therefore, be selective, describing only those laboratories which appear to be typical of a number of similar installations, or those which seem in some way to have a unique approach to their problems. As always, in being selective we face the possibility that some important system will be omitted. Unfortunately, this possibility is increased because the computer has become so common that authors no longer feel compelled to devote methods sections to detailed descriptions of their particular systems. However, one new journal (*Behavior Research Methods and Instrumentation*) devotes from one-third to one-half of its pages to computer technology, and a significant proportion of the contributions have come from auditory laboratories.

In auditory electrophysiology the functioning of the ear is investigated through the recording and analysis of electrical activity of the sensory and neural tissues of the auditory system. This electrical activity can be recorded at every level of the auditory pathway from the sensory receptor cells in the cochlea to the cortex of the brain, although the voltages involved are often at the microvolt level. Since living tissue is a reasonably good conductor, potentials can be recorded at some distance from their site of origin. This poses two problems. First, simply determining the generator site is often a difficult problem. Then, too, a recording electrode is sensitive to a number of biological signals, many of which are not related to the auditory function. Such signals are, in effect, noise in which the desired signals are embedded.

Special purpose digital computers which are designed to extract analog signals from noise are well suited to this task of recording the electrical responses of sensory cells within the cochlea, the action potential of the acoustic nerve, or that component of cortical activity due to acoustical stimulation. For the receptor potentials, electrodes are surgically implanted in the cochlea of a small animal (guinea pig or chinchilla) so that the biological potentials can be conducted to experimental apparatus. In some

cases the signals are processed on-line by a small special purpose computer; however, a number of researchers prefer to record the electrical activity using an instrumentation tape recorder, after appropriate amplification, for processing off-line at a later time. The analysis technique is essentially the same in either case. Signal averages are accumulated in synchrony with stimulus presentations. Thus the electrical activity time-locked to the acoustical input should be enhanced, while the asynchronous "noise" should cancel out. Good signal recovery can be accomplished with as few as 30 to 50 replications.

Parameters of the averaged waveform which vary systematically with changes in the acoustical signal are studied in an attempt to deduce the mechanisms involved. Within the cochlea, at least four biological potentials have been studied. Two are dc or resting potentials and are thought to be related to the metabolic activity of the sensory cells. The other two are true receptor potentials which vary in amplitude, frequency, and site of origin with changes in the intensity and frequency of the stimulus.

While most of the commercially available signal averaging computers may be adequate to perform these functions, computers have been built especially for the processing of bioelectric signals in analog form [27-30]. In addition to the averaged signal, these computers are able to display an amplitude histogram and ogive (cumulative distribution) of the sampled data. With the advent of general purpose minicomputers, the attractions of the special purpose (often single function) computer has diminished. For example, the LINC [30] and its successors [31] are able to perform these averaging functions but can be programmed to carry out many other useful functions as well.

However, special purpose computers are often used in the processing of the electroencephalic potentials which can be recorded from the scalp in response to sensory stimulation. This averaged evoked response (AER) was first reported in 1939 [32], but thorough investigations were not initiated until special purpose computers became available [33, 34]. Since then, interest in this area has flourished for several reasons. It is, first of all, one electrophysiological technique that can be used with humans since surgically implanted electrodes are not required. Physiologists who wish to work with human subjects must resort to these external recording sites. Although they pose greater difficulties in tying electrical responses to auditory system functions, investigations of the correlation between acoustical signal parameters and components of the AER continue [35-46].

The measurement of the AER lends itself to objective methods of audiology. A number of situations arise in the audiology clinic in which the person under test is unable or unwilling to cooperate with the tester. Factors such as physical or mental infirmities, or potential financial gain to the patient require testing procedures which do not rely upon the listener's conscious cooperation. Then the evoked response to audiometric tones is a means by which the audiologist may determine whether the patient's auditory system is intact. Evoked response audiology is also quite useful in determining the hearing levels of young children and infants in whom emotional, mental, and central nervous system disorders often produce symptoms similar to peripheral deafness [47].

A recent review of the evoked response literature [33] reveals the level of activity in this area. At this point in time, it seems that the application of averaging computers

to objective audiometry has been quite successful. Extensions into diagnostics have not yet been as rewarding. That is, there are not now analysis procedures which permit determination of the site or nature of the lesion detected by these audiometric techniques.

Within the cochlea, information carried in the acoustical stimulus is encoded into neural form for transmission to the central nervous system. A pair of cranial nerves, one for each ear, carries this information toward the brainstem. Each nerve is composed of nearly 30,000 individual fibers, the axons of single neural cells, or neurons, the cell bodies of which are found near the inner ear. An individual neuron transmits information by means of a unique electrochemical wave which propagates from the cell body to the distal end of the axon where specialized endings act to stimulate the next neuron in the chain.

This electrochemical wave, often called a neural spike potential, may be observed only if an extremely fine microelectrode is carefully inserted into the axon. The electrode must approach  $1 \mu$  in diameter, since the diameter of auditory axons is only a few microns.

Neural spikes have a unique waveshape for all stimulus conditions. Information must, therefore, be encoded by the rate at which the spikes are generated. In a whole nerve, additional coding may depend upon the number of fibers that are active, and perhaps upon which of the fibers are active. Firing rates vary with stimulus changes from a few spikes per second to nearly 1500/sec. Since modern digital computers operate in the microsecond region, they are well suited to the recording and analysis of spike activity within each single neuron.

Neural spikes may be treated as binary events in time since their waveshape is invariant with stimulus changes. For a given single fiber, we may be interested in the time between occurrences of neural spikes, or we may investigate the time interval from a stimulus event to the generation of a neural spike. Such data are often displayed in histogram form [48]. In the first case, an interspike-interval (ISI) histogram results. Each vertical bar in the histogram represents the number of interspike intervals with durations falling within its narrow temporal window or bin. One hundred or more such bins cover the expected range of interspike intervals. The second, and perhaps more widely used technique, yields a poststimulus time (PST) histogram. Each stimulus event resets a counter to zero and starts its clock. The neuronal spike stops the clock, and the time bin corresponding to the indication of the counter is incremented by one. The resulting histograms show one or more peaks with the majority of neural firings occurring at intervals of time which bear simple relationships to times characteristic of the given neuron.

Kiang's [49] pioneering work in this area led to the confirmation of several aspects of theories of cochlear action. Examination of the range of stimulus frequencies to which a neuron will respond reveals one frequency at which it is most sensitive. Different fibers display different frequencies of maximum sensitivity, thus covering the audible range. Kiang found that in response to acoustic clicks, the peaks in a PST histogram are separated by time intervals equal to the reciprocal of the unit's "best" or most sensitive frequency.

In addition, for a limited low frequency range, Kiang found a systematic relationship

between this best frequency and the latency of response. His results support the theory that mechanical events at different frequencies are distributed longitudinally along the sensory tissues of the cochlea. Finally, Kiang established that neural firings occur on the rarefaction phase of acoustic stimulation.

More recent work by Kiang [49], his co-workers [50, 51], and others [52-56, 68-70] has used these analysis techniques to shed further light on the mechanical events which precede the transformation of mechanical activity within the cochlea into neural energy.

The data processing techniques first used in studying the temporal discharge patterns of the acoustic nerve fibers have become standard for single unit work throughout the central auditory pathway [57-67]. The entire procedure, from collection and counting of neuronal spikes to display of the resulting histograms, is easily programmed on a small laboratory computer equipped with the appropriate A/D and D/A peripherals.

An outstanding example of a computerized auditory physiology laboratory is described by Aitkin, Anderson, and Brugge [63]. In their laboratory a LINC computer controls the acoustical calibration of the stimulus presentation apparatus, controls stimulus parameters during the experiment, and collects and displays single unit data as well. The flexibility and efficiency of this system may be inferred from several recent papers from their laboratory [63, 64, 68, 69]. In addition to several studies of single neuronal units, the same facility was used in an application of the Mössbauer technique to an investigation of the vibratory motion of the cochlear partition.

Although similar physiological studies cannot be carried out on humans, it is hoped that data obtained in studies of lower animals will yield results that are applicable to an understanding of the human auditory system.

Large electrodes placed near the auditory nerve record the sum of electrical activity in all of the individual fibers. The resulting bioelectrical signal is in analog form. Its amplitude, waveshape, and delay vary with stimulus changes, and signal averaging techniques are necessary to recover this whole nerve action potential from background noise. Using such methods, the potential can be recovered when the recording electrode is as far away as the ear canal of the listener [70, 71]. Since the existence of a normal action potential usually indicates healthy sensory receptor cells, the state of the cochlea can be assessed in humans. This procedure promises to be useful, perhaps in conjunction with AER audiometry, in determining the site of lesions within the auditory system.

Since it is virtually impossible to record from electrodes implanted within his auditory system, much of the research and clinical testing of the human listener must be performed psychoacoustically. A listener is asked to respond differently to perceptible differences in the acoustical signals presented to him. He may be asked simply to indicate whether or not a sound has been presented, or he may have to distinguish small differences in auditory attributes. That is, he may be asked to indicate which stimulus of a given set is higher (or lower) in pitch, which is louder, etc. In some psychoacoustical experiments he is asked to adjust a test signal until it matches in some attribute another signal presented for comparison. Most of the psychophysical methods of experimental psychology, both classical and modern, have found their way into psychoacoustics and audiology.

Psychoacoustics has adapted to the introduction of computers more quickly than any of the sensory sciences because it has been heavily dependent upon electronic apparatus for the generation and control of auditory stimuli. Whereas visual psycho-physics, for example, can still make use of the lenses, prisms, and paper cutouts of the nineteenth century; sirens, tuning forks, and other mechanical noise makers gave way to oscillators, amplifiers, and electronic switching apparatus. A degree of electronics sophistication is almost a prerequisite for the auditory researcher and a valuable asset to the practicing clinician.

It seems, then, quite natural for digital computers to find widespread applications in psychoacoustics. Early applications involved only a few large machines [72, 73], but the introduction of the minicomputer opened the door for nearly every researcher. Dedicated computers now range from medium size to the smallest mini's [74, 75].

Laboratory computers serve three major functions in the psychoacoustics laboratory. First, they simply replace the special timing and programming devices necessary to conduct many psychoacoustical experiments. In the past these functions were first performed by relays controlling vacuum tube amplifying and switching apparatus. Next came discrete solid-state logic. A number of different experimental paradigms could be implemented given enough AND gates, OR gates, flip-flops, etc. To change from one experiment to another often meant rewiring, but change-overs were expedited by the addition of a removable "programming board" through which all connections were made. Although these systems were a vast improvement over their noisy predecessors, there were still several difficulties. Psychoacoustical studies often required the presentation of one of several signal levels, or tones of various frequencies. More sophisticated studies required signals composed of multiple-tone complexes or noise of various bandwidths, still with rapid changes from one given signal to another. With discrete logic, a separate analog source and the associated timing and gating logic is required for each complex signal desired by an experimenter. The number of oscillators, filters, and attenuators required grows quickly with increased complexity of the stimulus array.

The answer to this dilemma was clear. The analog equipment was modified to allow programming of frequencies, attenuations, and bandwidths. However, this called for increased sophistication in the control logic. Signal parameters must be stored, somehow, for introduction to the programmable analog device at the appropriate time. Such experiment-control devices evolved into special purpose digital controllers. The cost of these devices, including hardware and design- and assembly-time, often became prohibitive. Those who did produce such special-purpose apparatus all too often discovered that the controller designed for one series of experiments was useless for the next without major changes in hardware.

Fortunately, at about this period of time the minicomputer appeared on the scene. Interfaced to the same programmable analog devices, it could perform any of the special purpose functions. It could be changed from one experimental procedure to another simply by introducing another program since the procedure resided in core rather than in hardware. The only disadvantage was the necessity for an experimenter to give up logic-design for machine-language computer programming. The introduction

of powerful computers and interpreters for the minis has minimized this objection to computer-controlled experiments.

In addition to their efficiency and flexibility in controlling standard experimental procedures, minicomputers have enabled experimenters to devise procedures that would be nearly impossible without the aid of a computer. In the classical as well as the more modern [76, 77] psychophysical methods, listeners were tested in blocks of experimental trials with stimulus parameters fixed for the entire block. Information concerning listener performance as a function of the stimulus variable, the psychometric function, can be obtained only by repeating the procedure for several signal levels. Often only one point on this function is of interest. For example, an experimenter may wish to determine the signal levels for 75% of correct responses for various listening conditions. In the block-trials procedures, results must be obtained at several points above and below the point of interest, a curve fitted to these points, and the signal level for the point of interest interpolated from the curve.

Experimental procedures based upon sequential testing of statistical hypotheses have eliminated much of the unnecessary data collection found in the fixed block methods [78-82]. In these procedures the experimental variable is adjusted according to the listener's performance on a small number of previous trials. Thus the listener and the experimental apparatus interact to arrive at a predetermined level of performance in a smaller number of experimental trials than fixed-block procedures would require. The level obtained, number of trials, and statistical reliability vary with the particular adaptive procedure chosen.

While any of the adaptive procedures can be implemented with digital logic modules, they seem to be ideally suited to computer implementation. Experience with special purpose programming devices for fixed procedures may have made researchers more cautious about investments in hardware which performs only one of the several adaptive procedures available. The flexibility of a small digital computer allows the experimenter to implement any of the popular procedures, or to experiment with new procedures of his own.

Computers have played an added role in the growth of these adaptive procedures. Monte Carlo testing of the procedures allows for comparisons of the methods for efficiency, reliability, and reputability when analytical comparisons were difficult to obtain [83-86].

Minicomputers have been used to control standard psychoacoustics experiments and to implement response-contingent (adaptive) paradigms because they have proven to be more flexible and more economical than a collection of special-purpose devices. These could, in theory, perform such functions equally as well as the computer does. However, a number of laboratories have discovered that the digital computer's full power is realized when it is used as a digital signal generator.

An example is the work of Pollack who was among the first to introduce a mini-computer into his laboratory [87]. Pollack has thoroughly investigated the ability of listeners to distinguish small differences in the temporal pattern of complex pulse trains generated by the computer [88-101]. The fine differences in sequential encoding and the pseudorandom perturbations in pulse rate used by Pollack would be nearly impossible to achieve using a reasonable number of discrete flip-flops, logic gates, etc.

Computer-generated output is not limited to pulse trains, however. Any signal that can be described analytically or statistically as a set of sampled points can be produced by the computer. For most auditory work the samples must be played at 8 to 10 kHz. With low-pass filtering at one-half the sampling rate, aliasing of the acoustical output is avoided. All of the current digital signal processing techniques are applicable to the generation and modification of auditory test signals. Although a limited number of such experiments have been conducted in the past [72, 73], it was the introduction of the mini that again was responsible for widespread use of digital signal processing in audition [102-112].

Using a small computer, an experimenter can generate signals with a number of precisely controlled harmonics [102]. He can obtain repeatable bursts of random noise [72] or pulse trains with very complex temporal structures [88]. The availability of such signals is limited by the ingenuity of the programmer, rather than by his hardware. Often, however, signals simply cannot be produced by any other means. For example, Patterson and Green [113, 114] investigated the temporal resolving power of the ear using what they chose to call "Huffman sequence" signals. These signals are produced by "ringing" digital all-pass filters. Although the amplitude characteristics of such filters are identical, they can be made to differ in their phase spectra. The resulting signals then have equal long-term energy spectra, but for each filter the energy in a designated spectral region can be delayed for a few milliseconds. It is somewhat surprising to find that listeners were able to detect the presence of this energy for delays as short as 2 to 3 msec. These results, together with those from several other experiments, indicate that the auditory system possesses temporal resolving powers an order of magnitude better than the other exteroceptive sensory systems [115-117].

In addition to its fine temporal resolving power, the auditory system is also capable of very fine frequency resolution. This is a paradoxical situation since these powers are usually reciprocally related in most natural and man-made systems. Indeed, the earliest studies of man's auditory sense dealt with his ability to perceive pitch.

In addition to the many conventional experiments dealing with pitch perception and frequency analysis in hearing, recent interest has focused upon the existence of contrast enhancement phenomena [118-122]. Work in progress in this area by Feth, Green, and Yost [122], among others, illustrates several complimentary applications of the computer.

In vision, these contrast enhancement effects, first described by Mach, have been observed for a number of years [121, 123, 124]. Briefly, Mach bands occur whenever a light-dark boundary is encountered. Apparently lateral connections within the retina act to make the perceived contrast greater than that in the visual display. That is, the light side appears lighter because of the proximity of the dark side, and vice versa.

An analogous situation has been suggested for the auditory system since it was discovered that the response to the frequencies contained in an acoustical signal is distributed along the cochlear partition. Since the selectivity of this mechanical analysis seemed insufficient, contrast enhancement through lateral inhibition was postulated as an explanation for the eventual fine spectral resolution [118, 119]. When studies of single fibers of the auditory nerve indicated good frequency resolution at this early level, the contrast enhancement hypothesis gained support.

Attempts to show "edge effects" in hearing using sharply filtered noise bands to mask probe tones did not produce unequivocal results. For although one experimenter claimed to produce such effects [119], others have disputed his results [120].

A better understanding of the Mach band effects was accomplished in vision when the visual system was modeled as a linear spatial filter [123, 124]. Feth, Green, and Yost have adapted this model to the auditory system. In order to determine the spatial filter transfer function, the stimuli must vary sinusoidally in the spatial dimension. Wide band random noise, delayed and attenuated before being recombined with the original, produces the desired pattern. Once the spatial transfer function has been determined, it is Fourier-transformed into the spread-of-excitation function. This excitation function should display direct evidence of inhibitory activity.

A digital computer was programmed to produce the wide band noise signal using a uniformly distributed pseudorandom number generator to produce addresses in a table of normally distributed samples. Noise bursts of any desired duration can be produced since each sample is calculated in real-time just before being converted to an analog voltage by the D/A converter. To produce the desired spectrally rippled noise bursts, samples were stored for a number of sample periods before being added to a later sample.

The stimulus generation program was a subroutine of an adaptive psychophysical procedure designed to determine the depth of ripple at which the listener could just distinguish a rippled noise from one with uniform spectrum on approximately 71% of the trials. The adaptive procedure, written in FORTRAN, ran on the laboratory computer calling the appropriate stimulus generating subroutine when necessary. The entire program was written so that the experimenter need not be present after an initial introductory session in which the listeners were given simple instructions concerning start-up of the program and calibration of signal levels.

After determining a listener's ripple sensitivity as a function of frequency, the desired transfer function, another program performed the Fourier transform on the same computer. One machine was used to generate the signals, to control the experiment, and to process the data. Had an X-Y plotter been available at the time, the original data in the form of transfer functions, and the processed result, the spread-of-excitation functions, could have been plotted in a form suitable for the final publication of results.

## THE FUTURE

Digital computers, especially the smaller laboratory-size ones, have become firmly entrenched in most aspects of auditory research, and there is no reason to believe that their numbers will not continue to increase. The level of sophistication of many applications should likewise increase. In auditory physiology with human subjects, an increased sophistication in digital signal processing techniques may lead to the recovery of much more useful information from electrical responses recorded at the scalp. Progress toward this goal may begin with animal studies in which single-unit activity is sampled simultaneously at several sites within the auditory system and also

at the surface. Selective alteration, or destruction, of a given anatomical structure might then produce variations in the scalp response which could be identified after processing of the waveform. Signal processing techniques such as autocorrelation and Fourier transformation of the evoked response waveform will ultimately replace the current practice of inspecting the amplitude and latency of voltage peaks in the averaged waveform. Results from such animal studies might indicate the signal processing necessary to identify the site of lesion in human listeners who have suffered some hearing impairment.

In more basic physiological research, the digital computer should become the signal generator as is the case in psychoacoustics. Physiological investigations using complex noise bursts, multiple tone arrays, and time-varying signals should soon replace those using single frequency tones or brief clicks as the stimuli. In addition, simultaneous recordings from multiple electrodes at one level of the auditory system, or from electrodes at several anatomical sites, will greatly increase the volume of data to be processed. A computerized facility should be capable of generating the stimuli and also recording, in synchrony, from the multiple electrodes. Multidimensional displays of the results, and perhaps multivariate analyses, may help to elucidate the complex mechanisms underlying auditory processing. Recently, a successful study using multiple electrodes in the cochlea and deconvolution of the recorded waveforms indicated that these procedures are not doomed to failure by their inherent complexity [125].

In psychoacoustics the laboratory computer will continue to replace single-purpose devices and collections of discrete logic modules for the control of experimental procedures. The availability of computerized facilities should stimulate an increased use of adaptive-testing procedures. As inter-active testing procedures evolve, we will soon be able to test several listeners simultaneously. Simple time-sharing of the control computer should enable each listener to perform in an individualized sequential test procedure.

The increase in volume of data collected can be handled by the same computer programmed to analyze and display the experimental results directly from its files of listener responses.

The impact of the minicomputer on psychoacoustics will be felt most strongly, however, wherever it is realized that in addition to its utility as a controller, the laboratory computer is also a powerful digital signal processor. Experimenters have always been aware of fundamental differences between their single-frequency tones, clicks, and bursts of noise in the laboratory, and speech, music, and environmental sounds which the auditory system must process. The ultimate goal has always been to understand how the ear processes the information in these "real" sounds.

Real sounds, however, are usually conveyed by nonstationary random waveforms which do not easily lend themselves to a high degree of stimulus control. While it is true that some aspects of recorded speech (or music) can be manipulated, the primary source has always been a human being. Production of a set of test signals is hardly under the direct control of an experimenter. Even when the experimenter himself assumes the role of speaker, he is capable of only gross control of the temporal and spectral composition of his utterances. Thus, while it is known that laboratory signals

are unnatural, they have been the best available. One always could hope that a well-behaved auditory system would reveal enough of itself in processing these artificial signals that the mechanisms for speech and music perception, or for auditory localization, would become apparent. The choice of test signals was dictated by the limitations of experimental apparatus, rather than by their suitability for auditory processing.

Using his computer as a digital signal generator, the psychoacoustician is no longer limited to simple tones, clicks, and noise bursts. Any waveform that can be described analytically or statistically can be produced by a digital computer for storage, D/A conversion, and eventually for presentation to the auditory system.

Recently several experimenters have reported listener responses to computer-generated music [126-130]. One of the more exciting recent developments is computer synthesized speech [131-134]. While direct verbal communications with computers is not yet a reality, synthetic speech has attained high levels of intelligibility and quality. Computer-driven speech synthesizers may someday replace tone and noise generators to the extent that the latter replaced tuning forks and sirens nearly 50 years ago. Although a few reports of perception of synthetic speech have appeared [135-140], the level of activity should increase manyfold as the necessary hardware becomes available.

There exists at present a wide gap between the methods used in the laboratory and those found in most audiology clinics. While most researchers have adopted the experimental paradigms of modern psychophysics, established clinical procedures are, for the most part, based upon methods introduced in the late nineteenth century. One cannot simply say that clinicians have failed to keep pace with the experimental world, for the newer methods present more than one difficulty in attempting the transfer from lab to clinic. These newer experimental methods were designed primarily for laboratory use. A listener is often given hours of practice before data collection is begun; then the experiment may continue for weeks. In the laboratory, most listeners are young, college undergraduates in the prime of their physical and mental health. This choice of methods and listeners is excellent when the goal is to discover the ultimate in human auditory system performance; however, the clinic often presents just the opposite situation. In the clinic, testing time is limited. The listeners are often the very young or the middle-aged or elderly. Most do have some hearing deficiency, and many of those who do not may be simulating a loss, perhaps to obtain some personal gain. Since established methods seem to work very well, it is not surprising to find the classical paradigms firmly entrenched in clinical practice.

If the computer is to find application in the audiology clinic, it must be used to improve upon the present situation. There is little doubt that a computer could be programmed to conduct many of the conventional auditory tests just as an audiologist conducts them. But, what good would it serve? Under this type of computer control, we would simply have a more expensive audiometer. One would still need a qualified audiologist to consider the test results and the circumstances under which they were obtained, and to arrive at an interpretation of the outcome. Inference is still a weak suite for most digital machines.

This is not to say, however, that digital computers will not find applications in the audiology clinic. On the contrary, within the next few years they should begin to

revolutionize clinical practice. This will be done both by relieving the clinician of dull, nonessential tasks, and through the introduction of sophisticated new clinical tests.

Let us consider the former prospect first. Presently many students and some full-trained audiologists are employed in the task of screening hundreds or thousands of people, hoping to detect early signs of hearing disorders. The test procedure is simple, straightforward, and unexciting. In short, the perfect spot for automatic audiometers. But, conducting the tests is only half of the problem. Results must be recorded, filed away, and, perhaps years later, retrieved from storage. These tasks require some sort of automated record keeping. At some time in the future, perhaps we will combine the tasks and eliminate the middle men and women. Envision a number of automatic audiometers, at remote locations, connected by telephone lines to one time-shared central processor. The computer could control the testing and record and store the results. Aberrant results would produce the same result as is the case today—an appointment with the nearest audiology clinic for further testing.

Within the clinic itself, that same computer, or its counterpart, could relieve the audiologist of much of the routine paper work now associated with the job. A computer interfaced to several video display-keyboard combinations could handle patient scheduling, case-history taking, and the assembly, recording, and reporting of test results. Each of these tasks could be made nearly foolproof, always with the option for human intervention. Take, for example, the history taking. Upon arriving for his appointment, the patient is asked by a receptionist to sit at the computer terminal and answer the questions which appear on the screen. The flow of information can be programmed so that negative answers eliminate unnecessary in-depth questioning, and, likewise, positive responses call forth questions to elicit more details. At worst, some patients will be unable or unwilling to respond in such a fashion, in which case the audiologist will have to act as interpreter.

In the actual tasks of audiological testing, many of the methods of the psychoacoustical laboratory will have to be modified. We will need, first of all, adaptive testing procedures that are intended for use with the expected clinical population. Perhaps a technique that first teaches the listener to make the appropriate responses, before concentrating upon efficient data collection. One computer should be able to test, simultaneously, several patients in different rooms on a time-shared basis.

Stimuli for the various auditory tests may one day be produced by direct digital generation. Complex, time-variable signals designed to measure auditory acuity or to determine the site of lesion may replace the "pure tone" and recorded speech test signals in use today.

## REFERENCES

1. M. R. Schroeder, Computers in acoustics : Symbiosis of an old science and a new tool, *J. Acoust. Soc. Amer.* **45**, 1077-1088 (1969).
2. J. C. Ogilve and C. D. Creelman, Maximum likelihood estimation of receiver operating characteristic curve parameters, *J. Math. Psychol.* **5**, 377-391 (1968).
3. V. Gourevitch and E. A. Galanter, A significance test for one parameter isosensitivity functions, *Psychometrika* **32**, 25-32 (1967).

4. O. D. Barnes and A. Finkelstien, The role of computer assisted instruction at the National Technical Institute for the Deaf, *Amer. Ann. Deaf* **116**, 466–468 (1971).
5. P. Suppes, Computer-assisted instruction for the deaf, *Amer. Ann. Deaf* **116**, 500–508 (1971).
6. R. N. Shepard, A. K. Romney, and S. B. Nerlove (eds.), *Multidimensional Scaling: Theory and Applications in the Behavioral Sciences*, Seminary Press, New York, 1972.
7. T. Carvelas and B. Schneider, Direct estimation of multidimensional tonal dissimilarity, *J. Acoust. Soc. Amer.* **51**, 1839–1848 (1972).
8. J. L. Hall and M. R. Schroeder, Monaural phase effects for two-tone signals, *J. Acoust. Soc. Amer.* **51**, 1882–1884 (1972).
9. B. J. McDermott, Multidimensional analyses of circuit quality judgments, *J. Acoust. Soc. Amer.* **45**, 774–781 (1969).
10. L. W. Graham and A. S. House, Phonological apposition in children: A perceptual study, *J. Acoust. Soc. Amer.* **49**, 559–566 (1971).
11. S. Singh, D. R. Woods, and A. Tishman, An alternative MD-SCAL analysis of the Graham and House data, *J. Acoust. Soc. Amer.* **51**, 666–668 (1972).
12. A. S. House and L. W. Graham, Comment on "An alternative MD-SCAL analysis of the Graham and House data," *J. Acoust. Soc. Amer.* **51**, 668–669 (1972).
13. I. K. Jeter and S. Singh, A comparison of phonemic and graphemic features of eight English consonants in auditory and visual modes, *J. Speech Hearing Res.* **15**, 201–210 (1972).
14. J. L. Flanagan, Computational models for basilar membrane displacement, part 2, *J. Acoust. Soc. Amer.* **34**, 1370–1376 (1962).
15. C. D. Geisler, A model of the peripheral auditory system responding to low-frequency tones, *Bioophys. J.* **8**, 1–15 (1968).
16. S. M. Khanna, R. E. Sears, and J. Tonndorf, Some properties of longitudinal shear waves: A study by computer simulation, *J. Acoust. Soc. Amer.* **43**, 1077–1084 (1968).
17. W. D. Keidel, Biophysics, mechanics, and electrophysiology of the human cochlea, in *Frequency Analysis and Periodicity Detection in Hearing* (R. Plomp and G. F. Smoorenburg, eds.), Sijthoff, Leiden, 1971.
18. A. E. Hubbard and C. D. Geisler, A hybrid computer model of the cochlear partition, *J. Acoust. Soc. Amer.* **51**, 1895–1903 (1972).
19. J. L. Hall, Comment on "Proposed explanation of synchrony of auditory-nerve impulses to combination tones," *J. Acoust. Soc. Amer.* **50**, 1555 (L) (1971).
20. E. deBoer, P. Kuyper, and G. Smoorenburg, Proposed explanation of synchrony of auditory-nerve impulses to combination tones, *J. Acoust. Soc. Amer.* **46**, 1579–1581 (L) (1969).
21. D. M. Green and R. D. Luce, Detection of auditory signals presented at random times, *Perception Psychophys.* **2**, 441–450 (1967).
22. R. D. Luce and D. M. Green, Detection of auditory signals presented at random times II, *Perception Psychophys.* **7**, 1–14 (1970).
23. D. M. Green and R. D. Luce, Detection of auditory signals presented at random times III, *Perception Psychophys.* **9**, 257–268 (1971).
24. D. M. Green, Fourier analysis of reaction time data, *Behav. Res. Methods Instr.* **3**, 121–125 (1971).
25. R. D. Luce and D. M. Green, A neural timing theory for response times and the psychophysics of intensity, *Psychol. Rev.* **79**, 14–57 (1972).
26. B. Leshowitz, Receiver operating characteristics and psychometric functions determined under simple- and pedestal-detection conditions, *J. Acoust. Soc. Amer.* **45**, 1474–1484 (1969).
27. W. A. Clark Jr. (1958), *Average Response Computer (ARC-1)*, Quarterly Report on Electronics, Massachusetts Institute of Technology, pp. 114–117.
28. W. A. Clark, M. A. Goldstein, R. M. Brown, C. E. Molnar, D. F. O'Brien, and H. E. Zieman, The average response computer (ARC): A digital device for computing averages and amplitudes and time histograms of electrophysiological responses, *Trans. IRE* **8**, 46–51 (1961).
29. A. M. Engebretson, A Digital Computer for Analyzing Certain Bioelectric Signals, M.S. Thesis Department of Electrical Engineering, Washington University, St. Louis, Mo., 1963.

30. W. A. Clark and C. E. Molnar, A description of the LINC, in *Computers in Biomedical Research*, Vol. 2 (R. W. Stacy and B. Waxman, eds.), Academic, New York, 1965.
31. R. J. Clayton, Comparison of the LINC, LINC-8, and PDP-12 computers, *Behav. Res. Methods and Instr.* **2**, 76 (1970).
32. P. A. Davis, Effects of acoustic stimulation during sleep, *J. Neurophysiol.* **2**, 494-499 (1939).
33. G. Hnatiow and J. P. Reneau, *Evoked Response Audiology: A Bibliographic Review*, Vol. 1, Monograph Supplement Central Wisconsin Colony and Training School Research Proceedings, Madison, Wis., 1968.
34. E. L. Lowell, C. T. Williams, R. M. Ballinger, and D. P. Alvig, Measurement of auditory threshold with a special purpose analog computer, *J. Speech Hearing Res.* **4**, 105-112 (1961).
35. H. Davis and S. Zerlin, Acoustic relations of the human vertex potential, *J. Acoust. Soc. Amer.* **39**, 109-116 (1966).
36. G. A. McCandless and D. E. Rose, Evoked cortical responses to stimulus change, *J. Speech Hearing Res.* **13**, 624-634 (1970).
37. B. A. Weber, Habituation and dishabituation of the averaged auditory evoked response, *J. Speech Hearing Res.* **13**, 387-394 (1970).
38. A. J. Derbyshire, S. B. Osenar, L. R. Hamilton, and M. E. Joseph, Problems in identifying an acoustically evoked potential to a single stimulus, *J. Speech Hearing Res.* **14**, 160-171 (1971).
39. E. M. Glaser, Cortical responses of awake cat to narrow-band FM noise stimuli, *J. Acoust. Soc. Amer.* **50**, 490-501 (1971).
40. L. W. Keating and H. B. Ruhm, Within average variability of the acoustically evoked response, *J. Speech Hearing Res.* **14**, 179-188 (1971).
41. R. H. Lane, L. L. Kupperman, and R. Goldstein, Early components of the averaged electroencephalic response in relation to rise-decay time and duration of pure tones, *J. Speech Hearing Res.* **14**, 408-415 (1971).
42. M. I. Mendel and R. Goldstein, Early components of the averaged electroencephalic response to constant level clicks during all-night sleep, *J. Speech Hearing Res.* **14**, 829-840 (1971).
43. E. J. Moore and J. P. Reneau, Induced biophysical artifacts in averaged electroencephalic response (AER) audiometry, *J. Speech Hearing Res.* **14**, 82-91 (1971).
44. P. H. Skinner and F. Antinoro, The effects of signal rise time and duration on the early components of the auditory evoked cortical response, *J. Speech Hearing Res.* **14**, 552-558 (1971).
45. J. W. Bauer, K. C. Squires, and P. H. Lindsay, Computer signal detection by monitoring auditory evoked potentials, *Perception Psychophys.* **11**, 301-308 (1972).
46. J. R. Madell and R. Goldstein, Relation between loudness and the amplitude of the early components of the averaged electroencephalic response, *J. Speech Hearing Res.* **15**, 134-151 (1972).
47. H. R. Myklebust, *Auditory Disorders in Children*, Grune and Stratton, New York, 1954.
48. G. L. Gerstein and N. Y-S. Kiang, An approach to the quantitative analysis of electrophysiological data from single neurons, *Biophys. J.* **1**, 15-28 (1960).
49. N. Y-S. Kiang, T. Watanabe, E. Thomas, and L. Clark, *Discharge Patterns of Single Fibers in the Cat's Auditory Nerve*, M.I.T. Press, Cambridge, Mass., 1964.
50. M. B. Sachs and N. Y-S. Kiang, Two-tone inhibition in auditory nerve fibers, *J. Acoust. Soc. Amer.* **43**, 1120-1128 (1968).
51. M. L. Wiederhold and N. Y-S. Kiang, Effects of electrical stimulation of the crossed olivocochlear bundle on single auditory-nerve fibers in the cat, *J. Acoust. Soc. Amer.* **48**, 950-965 (1970).
52. M. L. Wiederhold, Variations in the effects of electric stimulation of the crossed olivocochlear bundle on cat single auditory-nerve-fiber responses to tone bursts, *J. Acoust. Soc. Amer.* **48**, 966-977 (1970).
53. T. J. Goblick Jr. and R. R. Pfeiffer, Time-domain measurements of cochlear nonlinearities using combination click stimuli, *J. Acoust. Soc. Amer.* **46**, 924-938 (1969).
54. R. R. Pfeiffer and C. E. Molnar, Cochlear nerve fiber discharge patterns: Relationship to cochlear microphonics, *Science* **167**, 1614-1616 (1970).

55. J. E. Rose, J. F. Brugge, D. J. Anderson, and J. E. Hind, Phase-locked response to low frequency tones in single auditory nerve fibers of the squirrel monkey, *J. Neurophysiol.* **30**, 769-793 (1967).
56. J. E. Rose, J. E. Hind, D. J. Anderson, and J. F. Brugge, Some effects of stimulus intensity on response of auditory nerve fibers in the squirrel monkey, *J. Neurophysiol.* **34**, 685-699 (1971).
57. D. D. Greenwood and J. M. Goldberg, Response of neurons in the cochlear nuclei to variations in noise bandwidth and to tone-noise combinations, *J. Acoust. Soc. Amer.* **47**, 1022-1040 (1970).
58. A. Starr and R. Britt, Intracellular recordings from cat cochlear nucleus during tonal stimulation, *J. Neurophysiol.* **33**, 137-147 (1970).
59. R. A. Lavine, Phase locking in response of single neurons in cochlear nuclear complex of the cat to low frequency tonal stimuli, *J. Neurophysiol.* **34**, 467-483 (1971).
60. G. Moushegian and A. L. Rupert, Response diversity of neurons in ventral cochlear nucleus of kangaroo rat to low frequency tones, *J. Neurophysiol.* **33**, 351-364 (1971).
61. R. L. Smith and J. J. Zwislocki, Responses of some neurons of the cochlear nucleus to tone-intensity increments, *J. Acoust. Soc. Amer.* **50**, 1520-1525 (1971).
62. J. M. Goldberg and P. E. Brown, Response of binaural neurons of dog superior olfactory complex to dichotic tonal stimuli: some physiological mechanisms of sound localization, *J. Neurophysiol.* **32**, 613-636 (1969).
63. L. M. Aitkin, D. J. Anderson, and J. F. Brugge, Tonotopic organization and discharge characteristics of single neurons in nuclei of the lateral lemniscus of the cat, *J. Neurophysiol.* **33**, 421-440 (1970).
64. J. F. Brugge, D. J. Anderson, and L. M. Aitkin, Responses of neurons in the dorsal nucleus of the lateral lemniscus of cat to binaural tonal stimuli, *J. Neurophysiol.* **33**, 441-458 (1970).
65. G. R. Bock, W. R. Webster, and L. M. Aitkin, Discharge patterns of single units in inferior colliculus of the alert cat, *J. Neurophysiol.* **35**, 265-277 (1972).
66. L. M. Aitkin and W. R. Webster, Medial geniculate body of the cat: Organization and responses to tonal stimuli of neurons in ventral division, *J. Neurophysiol.* **35**, 365-380 (1972).
67. A. Starr and M. Don, Responses of squirrel monkey (*Sciurus sciureus*) medial geniculate units to binaural click stimuli, *J. Neurophysiol.* **35**, 510-517 (1972).
68. D. J. Anderson, J. E. Rose, J. E. Hind, and J. F. Brugge, Temporal position of discharge in single auditory nerve fibers within the cycle of a sine wave stimulus: Frequency and intensity effects, *J. Acoust. Soc. Amer.* **49**, 1131-1139 (1971).
69. W. S. Rhode, Observations of the vibration of the basilar membrane in squirrel monkeys using the Mössbauer technique, *J. Acoust. Soc. Amer.* **49**, 1218-1231 (1971).
70. H. Portman and J. M. Aran, Electrocochleography in the infant and small child. Techniques of objective audiometry, *Acta Otolaryng.* **71**, 235-261 (1971).
71. W. Saloman and W. W. Eberling, Cochlear nerve potentials recorded from the ear canal in man, *Acta Otolaryng.* **71**, 319-325 (1971).
72. S. M. Pfafflin and M. V. Mathews, Detection of auditory signals in reproducible noise, *J. Acoust. Soc. Amer.* **39**, 340-345 (1966).
73. B. S. Atal, M. R. Schroeder, and K. H. Kuttruff, Perception of coloration in filtered Gaussian noise-short-time spectral analyses by the ear, in *Proceedings of the 4th International Congress on Acoustics*, Copenhagen, 1962, Paper H31.
74. P. B. Denes and M. V. Mathews, Laboratory computers: Their capabilities and how to make them work for you, *Proc. IEEE* **58**, 520-531 (1970).
75. G. D. Creelman, Rapid response and flexible experimental control with a small on-line computer: PSYCHLE, *Behav. Res. Methods Instr.* **3**, 265-267 (1971).
76. H. Davis and S. R. Silverman, *Hearing and Deafness*, 3rd ed., Holt, Rinehart, and Winston, New York, 1970.
77. D. M. Green and J. A. Swets, *Signal Detection Theory and Psychophysics*, Wiley, New York, 1966.
78. W. J. Dixon and A. M. Mood, A method for obtaining and analyzing sensitivity data, *J. Amer. Stat. Assoc.* **43**, 109-126 (1948).
79. T. N. Cornsweet, The staircase method in psychophysics, *Amer. J. Psychol.* **75**, 485-491 (1962).

80. G. B. Wetherill and H. Levitt, Sequential estimation of points on a psychometric function, *Brit. J. Math. Stat. Psychol.* **18**, 1–10 (1965).
81. M. M. Taylor and C. D. Creelman, PEST: Efficient estimates on probability functions, *J. Acoust. Soc. Amer.* **41**, 782–787 (1967).
82. H. Levitt, Transformed up-down methods in psychoacoustics, *J. Acoust. Soc. Amer.* **49**, 467–477 (1971).
83. I. Pollack, Methodological determination of the PEST (Parametric estimation by sequential testing) procedure, *Perception Psychophys.* **3**, 285–289 (1968).
84. R. M. Rose, D. Y. Teller, and P. Rendleman, Statistical properties of staircase estimators, *Perception Psychophys.* **8**, 199–204 (1970).
85. I. Pollack, Methodological examination of the PEST (Parametric estimation by sequential testing) procedure II, *Perception Psychophys.* **9**, 229–230 (1971).
86. P. Rendleman, R. Rose, and D. Y. Teller, Statistical properties of Pollack's PEST procedure, *Perception Psychophys.* **9**, 208–212 (1971).
87. I. Pollack, P. Headly, and E. Maas, Modest computer-controlled psychoacoustical facility, *J. Acoust. Soc. Amer.* **39**, 1248 (A) (1966).
88. I. Pollack, Asynchrony: The perception of temporal gaps within auditory pulse patterns, *J. Acoust. Soc. Amer.* **42**, 1335–1340 (1967).
89. I. Pollack, Asynchrony II: Perception of temporal gaps within periodic and jittered pulse patterns, *J. Acoust. Soc. Amer.* **43**, 74–76 (1968).
90. I. Pollack, Discrimination of mean temporal interval within jittered auditory pulse trains, *J. Acoust. Soc. Amer.* **43**, 1107–1112 (1968).
91. I. Pollack, Periodicity discrimination for auditory pulse trains, *J. Acoust. Soc. Amer.* **43**, 1113–1119 (1968).
92. I. Pollack, Differential asynchrony thresholds for auditory pulse trains, *J. Acoust. Soc. Amer.* **45**, 446–449 (1969).
93. I. Pollack, Effect of masking noise and pulse level upon jitter detection, *J. Acoust. Soc. Amer.* **45**, 1022–1024 (1969).
94. I. Pollack, Discrimination of restrictions in sequentially-encoded auditory displays: Block designs, *Perception Psychophys.* **9**, 57–60 (1971).
95. I. Pollack, Discrimination of restrictions upon auditory sequentially encoded information: Block parity, *Perception Psychophys.* **9**, 253–256 (1971).
96. I. Pollack, Discrimination of restrictions upon sequentially encoded information: Variable length periodicities, *Perception Psychophys.* **9**, 321–326 (1971).
97. I. Pollack, Discrimination of restrictions in sequentially blocked auditory displays: Shifting block designs, *Perception Psychophys.* **9**, 335–338 (1971).
98. I. Pollack, Interaural correlation detection for auditory pulse trains, *J. Acoust. Soc. Amer.* **49**, 1213–1217 (1971).
99. I. Pollack, Depth of sequential auditory information processing: III, *J. Acoust. Soc. Amer.* **50**, 549–554 (1971).
100. I. Pollack, Spectral basis of auditory "jitter" detection, *J. Acoust. Soc. Amer.* **50**, 555–558 (1971).
101. I. Pollack, Amplitude and time jitter thresholds for rectangular-wave trains, *J. Acoust. Soc. Amer.* **50**, 1133–1142 (1971).
102. R. D. Patterson, *Residue Pitch as a Function of the Number and Relative Phase of the Components*, Technical Report No. 15, Center for Human Information Processing, University of California, San Diego, 1971.
103. A. Ahumada, Jr. and J. Lovell, Stimulus features in signal detection, *J. Acoust. Soc. Amer.* **49**, 1751–1756 (1971).
104. E. C. Carterette, A. Barnebey, J. D. Lovell, D. C. Nagel, and M. P. Friedman, On-line computing with the Hewlett-Packard 2116B moving-head disk operating system, *Behav. Res. Methods Instr.* **4**, 89–94 (1972).
105. J. L. Hall, Monaural phase effect: cancellation and reinforcement of distortion products  $f_2 - f_1$  and  $2f_1 - f_2$ , *J. Acoust. Soc. Amer.* **51**, 1872–1881 (1972).

106. J. L. Hall, Auditory distortion products  $f_2 - f_1$  and  $2f_1 - f_2$ , *J. Acoust. Soc. Amer.* **51**, 1863–1871 (1972).
107. B. Leshowitz and E. Cudahy, Masking with continuous and gated sinusoids, *J. Acoust. Soc. Amer.* **51**, 1921–1929 (1972).
108. J. D. Lovell and E. C. Carterette, Digital generation of acoustic stimuli, *Behav. Res. Methods Instr.* **4**, 151–155 (1972).
109. D. A. Ronken, Intensity discrimination of Rayleigh noise, *J. Acoust. Soc. Amer.* **45**, 54–57 (1969).
110. D. A. Ronken, Monaural detection of a phase difference between clicks, *J. Acoust. Soc. Amer.* **47**, 1091–1099 (1970).
111. D. A. Ronken, Some effects of bandwidth-duration constraints on frequency discrimination, *J. Acoust. Soc. Amer.* **49**, 1232–1242 (1971).
112. D. A. Ronken, Changes in frequency discrimination caused by leading and trailing tones, *J. Acoust. Soc. Amer.* **51**, 1947–1950 (1972).
113. J. H. Patterson and D. M. Green, Discrimination of transient signals having identical energy spectra, *J. Acoust. Soc. Amer.* **48**, 894–905 (1970).
114. J. H. Patterson, Masking of tones by transient signals having identical energy spectra, *J. Acoust. Soc. Amer.* **50**, 1126–1132 (1971).
115. D. M. Green, Temporal auditory acuity, *Psychol. Rev.* **78**, 540–551 (1971).
116. B. Leshowitz, The measurement of the two click threshold, *J. Acoust. Soc. Amer.* **49**, 462–466 (1970).
117. C. E. Robinson and I. Pollack, Forward and backward masking: Testing a discrete perceptual-moment hypothesis in audition, *J. Acoust. Soc. Amer.* **50**, 1512–1519 (1971).
118. G. von Bekesy, *Sensory Inhibition*, Princeton University Press, Princeton, N.J., 1966.
119. E. C. Carterett, M. P. Friedman, and J. D. Lovell, Mach bands in hearing, *J. Acoust. Soc. Amer.* **45**, 986–998 (1969).
120. H. Rainbolt and A. M. Small, Mach bands in auditory masking: An attempted replication, *J. Acoust. Soc. Amer.* **51**, 567–574 (1972).
121. F. Ratliff, *Mach Bands: Quantitative Studies on Neural Networks in the Retina*, Holden-Day, San Francisco, 1965.
122. L. L. Feth, D. M. Green, and W. A. Yost, Rippled noise investigations of lateral inhibition in the ear, *J. Acoust. Soc. Amer.* **52**, 141 (1972) (Abstract).
123. F. W. Campbell, The human eye as an optical filter, *Proc. IEEE* **56**, 1009–1014 (1968).
124. T. N. Cornsweet, *Visual Perception*, Academic, New York, 1970.
125. L. U. E. Kohlöffel, Studies of the distribution of cochlear potentials along the Basilar membrane, *Acta-Oto Laryngol. Suppl.* **288** (1971).
126. F. H. Slaymaker, Chords from tones having stretched partials, *J. Acoust. Soc. Amer.* **47**, 1569–1571 (1970).
127. W. J. Dowling and W. Fujitani, Contour, interval, and pitch recognition in memory for melodies, *J. Acoust. Soc. Amer.* **49**, 524–531 (1971).
128. W. J. Dowling, Recognition of inversions of melodies and melodic contours, *Perception Psychophys.* **9**, 348–349 (1971).
129. A. J. M. Houtsma and J. L. Goldstein, The central origin of the pitch of complex tones: Evidence from musical interval recognition, *J. Acoust. Soc. Amer.* **51**, 520–529 (1972).
130. D. Deutsch, Octave generalization and tune recognition, *Perception Psychophys.* **11**, 411–412 (1972).
131. P. B. Denes, Some experiments with computer synthesized speech, *Behav. Res. Methods Instr.* **2**, 1–5 (1970).
132. J. L. Flanagan, C. H. Coker, L. R. Rabiner, R. W. Schafer, and N. Umeda, Synthetic voices for computers, *IEEE Spectrum* **7**(10), 22–45 (1970).
133. J. L. Flanagan, The synthesis of speech, *Sci. Amer.* **226**(2) 48–55 (1972).
134. A. Oppenheim, Speech analysis-synthesis system based on homomorphic filtering, *J. Acoust. Soc. Amer.* **45**, 458–465 (1969).
135. A. E. Rosenberg, A computer-controlled system for the subjective evaluation of speech samples, *IEEE Trans. Audio Electroacoust.* **AU-17**, 216–221 (1969).

136. W. A. Ainsworth and J. B. Millar, A simple time-sharing system for speech perception experiments, *Behav. Res. Methods Instr.* **3**, 21-24 (1971).
137. W. A. Ainsworth, Perception of synthesized isolated vowels and h-d words as a function of fundamental frequency, *J. Acoust. Soc. Amer.* **49**, 1321-1324 (1971).
138. W. A. Ainsworth, Duration as a cue in the recognition of synthetic vowels, *J. Acoust. Soc. Amer.* **51**, 648-651 (1972).
139. W. Strange and T. Halwes, Confidence ratings in speech perception research: Evaluation of an efficient technique for discrimination testing, *Perception Psychophys.* **9**, 182-186 (1971).
140. J. R. Barclay, Noncategorical perception of a voiced-stop: A replication, *Perception Psychophys.* **11**, 269-273 (1972).
141. C. Y. Suen and M. P. Beddoes, Discrimination of vowel sounds of very short duration, *Perception Psychophys.* **11**, 417-419 (1972).

Lawrence L. Feth

## AUERBACH CORPORATIONS

The Auerbach Corporations are a group of companies providing a full range of services required to utilize information as a management and executive tool.

Auerbach's diverse activities place the corporations in direct and continuing communication with all computer equipment manufacturers, as well as with a cross-section of EDP users in business, the service fields, and government. The professional staff thus represents a correspondingly diverse expertise: businessmen, scientists, engineers, and educators; also, specialists in finance and accounting, marketing, manufacturing, distribution, personnel, engineering, and computer systems; and persons experienced in the advanced analytical techniques of systems analysis, socioeconomic planning and evaluation, and the application of mathematical and computer techniques in problem solving.

Founded in 1957 by Isaac L. Auerbach, the firm has grown to include two subsidiaries, employing over 400 people on two continents. Auerbach Associates, Inc. provides professional and consulting services, and Auerbach Publishers, Inc. is the world's leading publisher of reference services and books on the subject of data processing.

The corporate personality of Auerbach has, from the beginning, been a reflection of its founder. Isaac L. Auerbach's pioneering role in the information sciences dates back to his involvement with the original design team of BINAC and UNIVAC I, the first commercial electronic computers. He also directed the development of such military systems programs as the SAGE system radar target detection equipment for real-time data processing, and the ATLAS Missile Guidance Computer. As a division manager for 8 years with the Burroughs Corp., he was responsible for organizing and

directing that company's entire Defense and Space Research and Development effort. It was the desire to bring those sophisticated skills acquired at Burroughs to a more expansive marketplace that prompted him to establish his own corporation.

In its early years, Auerbach was heavily involved in the design of real-time systems and large command and control systems, such emphasis reflecting the experience brought to the new corporation by Auerbach and the original staff members. Today, this strong design capability remains; however, corporate emphasis has shifted to provide a greater concentration on the applications aspect of computer science. This emphasis on the cost effective use of computing power is evident in both the consulting and publishing companies of Auerbach.

## CONSULTING AND PROFESSIONAL SERVICES

Auerbach Associates employs the resources of a highly trained and experienced group of senior scientists, engineers, educators, and management specialists, involved in the problems, projects, and advanced developments of information technology and computer science.

This team has become active within a wide spectrum of major consulting areas, including: Acquisition and Merger Studies; Command and Control; Computer Programming; Computer Selection; Design Automation; Facilities Management; Library Systems; Math Modeling and Simulation; Operations Research; Personnel System Design and Evaluation; Product and Marketing Planning; Program Management and Monitoring; Social Systems Surveys; Socio-Economic System Design and Evaluation; State-of-the-Art Studies; Systems Analysis; Information System Design; Storage and Retrieval; Software; Real Time Communications; System Effectiveness; Training and Course Design; Transportation Systems; and Urban Planning.

Among the analytical techniques employed by the company are mathematical simulation models, input-output analyses, field surveys, cost effectiveness analyses, and market research.

Auerbach's existence in this area is directly related to the growing complexities of management problems. As the management process becomes more complex, due to increased size and scope, new technology, and rapidly changing social and economic developments, the need for outside professional consultants possessing a wide range of skills has been increasingly recognized.

The applications capability of the consulting group has been defined by the creation of specialized consulting divisions which evolved to serve the needs of additional applications areas as they became apparent. These divisions are currently: the Federal Systems Division; the Commercial-Industrial Division; the Research and Development Division; the Socio-Economic Division; the Health and Justice Division; and the Automated Systems Division.

## PUBLISHING ACTIVITY

Auerbach first entered the publishing field in 1961 with the *Auerbach Standard EDP Reports* which since that time have become the standard monthly updated encyclopedic reference service for qualitative and quantitative analyses of commercially available computer systems worldwide. The Auerbach product line has since been expanded to cover minicomputers, input/output equipment, software, data communications, and time sharing as part of the Auerbach Computer Technology Reports series.

In 1973 Auerbach released the first service in the Auerbach Information Management Service series. This product, *Auerbach Data Processing Management*, deals with the management problems associated with a data processing installation. It is the first regularly updated management information service designed for the data processing manager. In 1974 Auerbach released *Auerbach Computer Programming Management*, an updated reference service for the programming manager that deals with the technical as well as the personal side of his job.

Future expansion of the Auerbach Information Management Series will include non-EDP disciplines as well as extensions of the current coverage. Auerbach expects this series to become the standard for information managers just as the Auerbach Computer Technology Reports have become the standard for hardware and software analysts.

## COMPANY DEVELOPMENT TO MEET CUSTOMER NEEDS

Auerbach's early involvement with projects such as the Ballistic Missile Early Warning System (BMEWS); AUTODIN, the worldwide Defense Communications Agency's data communications network; and OPCON, an advanced strategic operations control center for the U.S. Navy; soon earned the company a reputation for excellence in real-time computer and communications systems.

As the company grew, it began to expand into the commercial/industrial market-place, and soon was doing work for all of the major computer manufacturers and many of the nation's major companies. Its consulting assignments have also been heavily oriented toward helping companies realize increased efficiency from their computer and data communications installations. Equipment selection, development of management information systems, programming assistance, and the development of proper procedures and controls have all been significant parts of Auerbach's consulting assistance to business and industry, both in the United States and Europe.

Auerbach specializes in product and market planning for manufacturers of computers and related equipment. Auerbach has been heavily involved in the development of equipment advances which characterized the computer industry throughout the 1960s. Auerbach also became involved in assisting United States manufacturers in

their efforts to penetrate international markets. The company had maintained European offices in Amsterdam, The Netherlands, and most recently in London, England.

As national priorities began to change and more emphasis began to be placed on social issues, Auerbach was among the first computer-oriented consulting firms to bring information technologies and systems management techniques to bear on social problems.

Among the more significant of these has been Auerbach's involvement with a variety of programs aimed at alleviating unemployment. Working with the Department of Labor and the Department of Health, Education and Welfare, Auerbach has conducted a 3-year study of the impact of the Work Incentive Program, evaluating the effectiveness of the program in more than 60 locations across the nation. A project rating system has also been developed so that the government can determine relative project strengths and weaknesses. Auerbach also went into the field to evaluate employment offices in more than 30 metropolitan areas and developed guidelines for use in the overall appraisal of the Human Resources Development Program.

In the direct application of computer technology to meet human needs, Auerbach assisted the New York State Employment Service in the design, development, and implementation of an Area Manpower Data System to perform a job matching procedure based on the analysis of job applicant and job opening. This Job Bank and Screening System compares a file of job orders with an applicant's qualifications and presents the best jobs available to an applicant for his selection. This system has been particularly effective among lower-skilled job applicants because it reduces the amount of time required to screen a large number of jobs, and thus eliminates some of the discouraging aspects of job hunting.

In addition to these employment-oriented programs, Auerbach has also assisted federal, state, and local governments with computer-use studies, development of information and data communications systems, and the application of system management techniques to government programs. In these assignments, Auerbach has assisted boards of education, police and fire departments, various welfare agencies, courts, and public libraries, and in Pennsylvania has served on a governor's commission to improve efficiency in state government.

Auerbach's socially-oriented consulting work has also included several programs dedicated to providing improved health care. The most significant of these has been the company's involvement with the National Cancer Institute in developing the program for the National Cancer Plan. In the conduct of this work, Auerbach consultants have worked with NCI officials and the nation's leading physicians and surgeons to organize, draft, and document a blueprint for future cancer research and cancer treatment programs.

Despite its diversification into other business and social areas, Auerbach continues its involvement with defense-oriented programs working on a number of projects involving computer-use within the military services and large-scale communications systems. In addition, the company continues to appraise advanced computer techniques and equipment for a department of the federal government to forecast

what emerging technologies will be available for government use in the post-1975 period.

## GROWTH OF PUBLISHING ACTIVITY

Company growth has also been experienced in the publishing field and Auerbach today is recognized as the leading provider of printed information about computing equipment and computer use.

From its beginning in 1961, Auerbach publishing activity has grown to include a list of reference services which includes virtually every area of computers and data communications. The company now publishes regularly updated reports and notebooks on general purpose computers, small business computers, minicomputers, data communications, software, time sharing, and input/output peripherals. In addition, Auerbach provides specially prepared notebooks for computer manufacturers and users in Europe and a separate document for other overseas markets.

From its basic reports and notebooks services, Auerbach has also expanded to provide special subject guides, a series of tutorial books on virtually every data processing interest area, and digests of equipment available in various product areas.

Auerbach reference services are generally regarded as unmatched in their field, not only because of their depth and breadth of coverage but also because of their unique presentation of information. Specially designed charts are used to provide quick reference information regarding specifications, performance, and price for nearly all equipment available in a specific product class. In addition, a report numbering system is used which makes it possible to have consistent indexing throughout the entire product line.

In 1973 the *Auerbach Data Processing Management*, the first product in the Auerbach Information Management Series, was introduced. This was followed by the introduction of *Auerbach Computer Programming Management* in 1974. Auerbach expects this new series to continue to expand and become a significant contributor to the state-of-the-art of management science.

## LOOKING TO THE FUTURE

Throughout their 15 years of existence the Auerbach Corporations have been in the forefront of the development of computer technology and the increasingly efficient use of that technology in government, business, and industry.

Isaac L. Auerbach, the company's founder and president, has been one of the first proponents of the theory of distributed intelligence systems. He believes that the growth of information technology must be in direct relationship to the real needs of informa-

tion and computer users. Looking to the future, Auerbach says, "It will be the objectives of our companies to stay at the forefront of technological progress to meet real human needs."

*Milton Strassberg*

## AUTOCORRELATION

An important step in applying computer simulation is to estimate parameters from observations on random variables. Although a considerable body of literature exists on the design of experiments, few methods have been developed specifically for simulation experiments, and the classical approaches assume the existence of independent data on which to base statistical tests. Quite the contrary, many statistics important in a simulation do not satisfy the independence condition.

For example, suppose that one wishes to determine the mean waiting time in an investigation involving a single-server queueing system with Poisson-arrival, exponential-service distributions and a first-in, first-out queueing discipline. This is usually done by collecting the waiting times of a number of successive customers and dividing by that number. Waiting times measured this way, however, are not independent, for whenever a queue forms, the waiting time of each customer in the queue obviously depends upon the waiting times of the preceding customers. Any series of data in which this property of having certain values affect other values is said to be *autocorrelated* [1]. Furthermore, the autocorrelation increases rapidly with increasing utilization of the service facility. The extent to which data are autocorrelated can be quantified in ways to be discussed later.

Calculation of autocorrelation coefficients is also found in the study of cyclical components of a time series such as sets of economic data [2-5], in the output of electrical communications devices, and in the yearly index of sunspots [6]. In these references discrete and continuous, as well as stationary and nonstationary, series are examined; also included are estimation reliability and problems inherent in computation and sampling. The following discussion will be limited to consideration of data taken at equal intervals of time and forming a stationary time series (one that fluctuates or oscillates about a constant mean).

## COEFFICIENT OF AUTOCORRELATION

The autocorrelation coefficient can be most easily appreciated as an extension of the *simple* correlation coefficient between two variates  $x$  and  $y$  which is a measure of the fit of a linear equation,  $y = a + bx$ , where the constants  $a$  and  $b$  are found by least-squares techniques. The simple correlation coefficient is defined as the covariance of the variates  $x$  and  $y$  divided by the square root of the product of the individual variances, or

$$\begin{aligned} r_{yx} &= \frac{s_{xy}}{\sqrt{(s_x^2)(s_y^2)}} = \frac{\frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{n-1}}{\left\{ \left[ \frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n-1} \right] \left[ \frac{\sum_{i=1}^n (y_i - \bar{y})^2}{n-1} \right] \right\}^{1/2}} \\ &= \frac{\sum_{i=1}^n x_i y_i - \left( \sum_{i=1}^n x_i \right) \left( \sum_{i=1}^n y_i \right) / n}{\left\{ \left[ \sum_{i=1}^n x_i^2 - \left( \sum_{i=1}^n x_i \right)^2 / n \right] \left[ \sum_{i=1}^n y_i^2 - \left( \sum_{i=1}^n y_i \right)^2 / n \right] \right\}^{1/2}} \end{aligned} \quad (1)$$

which ranges from negative unity to positive unity. A value of  $r_{yx} = 0$  corresponds to the absence of correlation,  $r_{yx} = +1.0$  corresponds to perfect positive linear correlation, and  $r_{yx} = -1.0$  corresponds to perfect negative linear correlation. Values of the dependent variate  $y$  are assumed to be normally distributed around the least-squares line, and the observations, as well as their deviations from the least-squares line, are assumed to be independent of one another. The fraction of variation in  $y$  associated with a given variation in  $x$  is equal to the square of the correlation coefficient and is referred to as the coefficient of determination.

If, however, one is concerned with time-series data such as that obtained from the queueing-simulation problem discussed earlier, a similar method of development may be used for the calculation of an autocorrelation coefficient between the values of the variate  $x_i$  and the same variate at a constant lag or interval of time  $x_{i+j}$ . The *auto*-correlation coefficient for a particular lag  $j$  is then given by

$$r_j = \frac{\sum_{i=1}^{n-j} x_i x_{i+j} - \left( \sum_{i=1}^{n-j} x_i \right) \left( \sum_{i=1}^{n-j} x_{i+j} \right) / (n-j)}{\left\{ \left[ \sum_{i=1}^{n-j} x_i^2 - \left( \sum_{i=1}^{n-j} x_i \right)^2 / (n-j) \right] \left[ \sum_{i=1}^{n-j} x_{i+j}^2 - \left( \sum_{i=1}^{n-j} x_{i+j} \right)^2 / (n-j) \right] \right\}^{1/2}} \quad (2)$$

which is the same equation as that for the simple correlation coefficient between two variates  $x$  and  $y$ , with the value  $x_{i+j}$  being used in place of  $y_i$ . Examination of  $r$  calculated as a function of  $j$  will indicate those periods or lags over which the time series data appear to be correlated. Several other autocorrelation or serial correlation statistics have been collected by Walsh [7].

As an example illustrating the phenomenon of autocorrelation, two sets of stationary time series data have been considered: (1) number of fatal accidents and (2)

productivity in bituminous coal mines in the United States over a span of twenty years, 1946–1966 (obtained from the U.S. Bureau of Mines Health and Safety Analysis Center, Denver, Colorado). The time interval between the successive observations was one year. The autocorrelation between the successive number of fatal accidents was 0.8698 and successive productivity was 0.99865. With such high degree of autocorrelation between values in successive years, one can fairly accurately predict values of these variables for future years. Predictions employing linear regression analyses for both number of fatal accidents and productivity are shown in Tables 1 and 2.

**TABLE 1**

Relationship between Number of Fatal Accidents for Each Successive Year

Linear Equation,  $Y = (57.2937) + X (0.944346)$

Coefficient of Autocorrelation<sup>a</sup> = 0.869862

Coefficient of Determination = 0.75666

$Y^b$	$Y\text{-actual}^c$	$Y\text{-calc}$	Percent difference
795	925	808.049	-12.7
985	795	987.475	24.2
862	985	871.32	-11.6
494	862	523.801	-39.3
550	494	576.684	16.7
684	550	703.227	27.9
449	684	481.305	-29.7
397	449	432.199	-3.8
334	397	372.705	-6.2
360	334	397.258	18.9
392	360	427.477	18.7
427	392	460.53	17.5
326	427	365.151	-14.6
246	326	289.603	-11.3
290	246	331.154	34.6
275	290	316.989	9.3
263	275	305.657	11.1
252	263	295.269	12.3
218	252	263.161	4.4
251	218	294.325	35
227	251	271.66	8.2

<sup>a</sup> The coefficient of correlation  $r_j$  was calculated according to Eq. (2).

<sup>b</sup>  $X$  = fatalities in year  $i$ .

<sup>c</sup>  $Y$  = fatalities in year  $(i - 1)$ .

**TABLE 2**

Relationship between Productivity of Each Successive Year

Linear Equation,  $Y = (4.83116E-3) + X(0.940654)$ Coefficient of Autocorrelation<sup>a</sup> = 0.99865

Coefficient of Determination = 0.997302

$X^b$	$Y\text{-actual}^c$	$Y\text{-calc}$	Percent difference
0.73	0.71	0.691508	-2.7
0.78	0.73	0.738541	1.2
0.79	0.78	0.747948	-4.2
0.82	0.79	0.776167	-1.9
0.87	0.82	0.8232	0.4
0.9	0.87	0.85142	-2.2
0.94	0.9	0.889046	-1.3
1.03	0.94	0.973705	3.6
1.16	1.03	1.09599	6.4
1.24	1.16	1.17124	1
1.29	1.24	1.21827	-1.9
1.34	1.29	1.26531	-2
1.43	1.34	1.34997	0.7
1.55	1.43	1.46284	2.3
1.62	1.55	1.52869	-1.5
1.73	1.62	1.63216	0.8
1.85	1.73	1.74504	0.9
1.98	1.85	1.86733	0.9
2.1	1.98	1.9802	0
2.21	2.1	2.08368	-0.9
2.31	2.21	2.17774	-1.6

<sup>a</sup> The coefficient of correlation  $r_f$  was calculated according to Eq. (2).<sup>b</sup>  $X$  = productivity in year  $i$ .<sup>c</sup>  $Y$  = productivity in year  $(i - 1)$ .**REFERENCES**

1. G. Gordon, *System Simulation*, Prentice-Hall, Englewood Cliffs, N.J., 1959.
2. R. Brown, *Statistical Forecasting for Inventory Control*, McGraw-Hill, New York, 1959.
3. G. Fishman, *Spectral Methods in Econometrics*, Harvard University Press, Cambridge, 1969.
4. J. Lesourne, *Economic Analysis and Industrial Management*, Prentice-Hall, Englewood Cliffs, N.J.
5. P. Rao and R. Miller, *Applied Econometrics*, Wadsworth, Belmont, Cal., 1971.
6. A. Ralston and H. Wilf, *Mathematical Methods for Digital Computers*, Wiley, New York, 1960.
7. J. Walsh, *Handbook of Nonparametric Statistics*, Van Nostrand, Princeton, N.J., 1962.

*J. R. Zelonka*

# AUTOMATA THEORY

Much of mathematics studies *functions*, rules  $f : X \rightarrow Y$  for assigning to each element  $x$  of the set  $X$  an element  $f(x)$  of the set  $Y$ . In computer science, we seek to realize functions by *computations*, which are sequences of data manipulations under the control of a *program*. Programs are run by *machines*. In specifying a machine we have to specify the *syntax*, or legitimate forms, for the data structures that may occur as *input* to the machine, as descriptions of the *internal state* of the machine, and as *output* from the machine. A program must be so presented as input structure that it will be incorporated into the internal state structure in a fashion that will cause the machine to process data in such a way as to realize some given function. The *semantics* of the machine direct how it will change state, read input and provide output at each stage, and thus determines how the program will be interpreted.

*Automata theory* is the mathematical study of questions of realization, decomposition, simulation, complexity, and computation abstracted from the above considerations. In this article we shall sample some portions of this theory, starting with the notion of a *Turing machine* [1, 2], a formalization of the digital computer which preceded the computer itself by almost a decade: The machine consists of a control box in which may be placed a finite program; an indefinitely extendable tape, divided lengthwise into squares (i.e., depending on our choice of mathematical fiction, we may consider the tape as comprising an infinite string of squares of which all but finitely many are blank; or as a finite tape, to the ends of which arbitrarily many new squares may be added as required); and a device for scanning, or printing on, one square of the tape at a time, and for moving the tape, all under the command of the control box. We start the machine with a finite string (i.e., sequence) of symbols from some alphabet  $X$  (we denote by  $X^*$  the set of all such strings) on the tape, and with a program in the control box. The symbol scanned, and the instruction now being executed, determine what new symbol shall be printed on the square, how the tape shall be moved, and what instruction is to be executed next. If and when the machine stops, the result of our computation, a new string from  $X^*$ , may be read off the tape.

## MANY NOTIONS OF EFFECTIVENESS

Let  $Z$  be a Turing machine, specified by its program, i.e., list of instructions. We may associate with  $Z$  a numerical function  $f_Z$  simply by placing a number  $n$  suitably encoded as a string  $(n)$  on the tape and start  $Z$  scanning the leftmost square of  $(n)$ . If and when  $Z$  stops, we decode the result to obtain the number  $f_Z(n)$ . If  $Z$  never stops, we leave  $f_Z(n)$  undefined. (Of course, we can only associate  $f_Z$  with  $Z$  after we have chosen our encoding and decoding.) *Turing's hypothesis* (also called *Church's thesis*) is that a function is *effectively computable* if and only if it is an  $f_Z$  for some  $Z$ .

Since each Turing machine is described by a finite list of instructions, it is easy to show that we may *effectively* enumerate the Turing machines

$$Z_1, Z_2, Z_3, \dots$$

so that, given  $n$ , we may effectively find  $Z_n$ , and given the list of instructions for  $Z$ , we may effectively find the  $n$  for which  $Z = Z_n$ .

This implies that we can effectively enumerate all partial recursive functions as

$$f_1, f_2, f_3, \dots$$

simply by setting  $f_n = f_{Z_n}$ .

Say that  $f_n$  is *total* if  $f_n(w)$  is defined for all  $w$ . We might ask: Is there an effective procedure for generating those  $f_n$  which are total; i.e., does there exist a total recursive function  $h$  such that  $f_n$  is total if and only if  $n = h(m)$  for some  $m$ ? The answer is “No.” For if such an  $h$  existed, define  $f$  by

$$f(n) = f_{h(n)}(n) + 1$$

Then  $f$  is total recursive, and so  $f = f_{h(m)}$  for some  $m$ . Then  $f_{h(m)}(m) = f_{h(m)}(m) + 1$ , a contradiction!

A more subtle argument shows that there is no effective procedure for telling whether or not  $f_n$  is total. This is just one example of the many things we can prove to be undecidable by any effective procedure. To say that we cannot effectively tell that  $f_n$  is *total* is just the same as saying that we cannot tell effectively when  $Z_n$  will stop computing no matter what tape it is started on. We may thus say that “the halting problem for Turing machines is unsolvable.”

A most interesting result of Turing’s paper is that there is a *universal* Turing machine, i.e., one which when given a coded description of  $Z_n$  on its tape, as well as the data  $w$ , will then proceed to compute  $f_n(w)$ , if it is defined. This is obvious if we accept Turing’s hypothesis, for given  $n$  and  $w$  we find  $Z_n$  effectively, and then use it to compute  $f_n(w)$ , and so there should exist a Turing machine to implement the effective procedure of going from the pair  $(n, w)$  to the value  $f_n(w)$ . A proper proof takes somewhat longer!

Many attempts besides Turing’s have been made to formalize the notion of effectiveness. Each formalization has yielded a class of procedures equivalent to those implementable by Post-Turing machines (or a subclass thereof). We briefly give examples of two such approaches.

The first is that of *recursion*. Starting with the simple functions

$$N(x) = 0$$

the constant function with value zero,

$$S(x) = x + 1$$

the successor function, and

$$U_j(x_1, \dots, x_n) = x_j$$

which selects the  $j$ th of  $n$  arguments, we build up new functions by the three operations of

*Composition:* given functions  $h, g_1, \dots, g_m$  form the new function

$$f(x_1, \dots, x_n) = h(g_1(x_1, \dots, x_n), \dots, g_m(x_1, \dots, x_n))$$

*Recursion:* given functions  $g, h$  form the new function

$$f(0, x_2, \dots, x_n) = g(x_2, \dots, x_n)$$

$$f(x+1, x_2, \dots, x_n) = h(x, f(x, x_2, \dots, x_n), x_2, \dots, x_n)$$

*Minimization:* given the function  $g$ , form the new function

$$f(x_1, \dots, x_n) = \mu y[g(x_1, \dots, x_n, y) = 0]$$

the least  $y$  such that  $g(x_1, \dots, x_n, y)$  is zero whereas  $g(x_1, \dots, x_n, z)$  is defined but nonzero for  $z < y$ .

The class of functions so obtained is called *partial recursive*: recursive because of the importance of the recursion scheme, partial because the use of minimization leads to functions not defined for all values of their arguments. It can be shown that a function is partial recursive if and only if it is an  $f_z$ , i.e., it can be computed by a Turing machine  $Z$ .

We next turn to the notion of a recursive computer to provide a general framework for proving that various computer models compute only partial recursive functions: If at some time a computer has  $n$  registers, the  $j$ th of which contains the word  $w_j$ , we may represent the *complete state* of the computer by the string  $w_1bw_2b\dots bw_n$  where we assume the spacing symbol  $b$  is not used in the strings  $w_j$ . More generally: A *recursive computer*  $\mathcal{C}$  on the alphabet  $Y$  is a partial recursive (i.e., effectively computable) function  $f_{\mathcal{C}} : Y^* \rightarrow Y^*$ . We call  $\hat{Q} = Y^*$  the set of *complete states* for  $\mathcal{C}$ ; and call  $\xi \in \hat{Q}$  a *halting state* if  $f_{\mathcal{C}}(\xi) = \xi$ . A *computation* for  $\mathcal{C}$  is then any finite sequence

$$\xi_0, \xi_1, \xi_2, \dots, \xi_n$$

of elements of  $\hat{Q}$  such that  $\xi_{j+1} = f_{\mathcal{C}}(\xi_j)$  for  $0 \leq j < n$ , and, further,  $\xi_n = f_{\mathcal{C}}(\xi_n)$  is a halting state. Given a state  $\xi$  of  $Q$ , we define  $f_{\mathcal{C}}^*(\xi)$  to be the halting state of the computation of  $\mathcal{C}$  which starts with  $\xi$  (and is thus unique, if it is defined).

A program  $\Pi$  for  $\mathcal{C}$  is then a pair of maps  $\alpha : (X^*)^m \rightarrow Q$  and  $\beta : Q \rightarrow (X^*)^n$  which are partial recursive on  $(X \cup Y)$ . The (partial) function computed by  $\mathcal{C}$  with program  $\Pi$  is then the function  $\mathcal{C}_{\pi} : (X^*)^m \rightarrow (X^*)^n$  defined by commutativity of the diagram

$$\begin{array}{ccc}
 & \mathcal{C}_{\pi} & \\
 (X^*)^m & \xrightarrow{\quad} & (X^*)_n \\
 \downarrow \alpha & & \downarrow \beta \\
 Q & \xrightarrow{f_{\mathcal{C}}^*} & Q
 \end{array}$$

In other words,  $\mathcal{C}_\pi$  is the computation obtained (1) by using  $\alpha$  to read the data and appropriate instructions into memory, then (2) computing with  $\mathcal{C}$  until the computation halts, whereupon (3)  $\beta$  is used to read the result from the registers. In any case, it is then straightforward to verify the following theorem [3a, Theorem 6.2.4.] which provides the framework for proving partial recursiveness of various computed functions:

#### THEOREM

If  $\mathcal{C}$  is a recursive computer and  $\Pi$  is a program for  $\mathcal{C}$ , then the function  $\mathcal{C}_\pi$  is partial recursive.

The general recursive computer may be specialized to yield a whole family of mathematical models of computation. For instance, we may structure  $Q$  to comprise a set of registers of which any number are preceded by arrows, and then restrict the next-state function to be a total recursive function which in some sense executes the instructions stored in the arrowed registers to get a model of a parallel computer—and still be assured that it only computes partial recursive functions.

In specifying a computer, we shall assume [3b] that there is an underlying graph  $G = (V, E \subseteq V \times V)$ , such that a register is located at each node  $V$ . We next assume that the machine has associated with it a set  $B$  of possible values which may occur in its registers. Certain of these values may be decoded uniquely as operational or jumping commands. Let  $B_1 \subseteq B$  be the set of “command values.” We then define  $v(v, b)$  to be the instruction encoded by value  $b$  if it occurs in register  $v$ . (The dependence of  $b$  on  $v$  takes account of one type of relative addressing. It is straightforward to include index registers, too, but we shall not do it here.)

We postulate that the computer can execute any of a finite repertoire  $g_1, \dots, g_k$  of operations

$$g_j : B^{m_j} \rightarrow B^{n_j}$$

and  $\tilde{g}_1, \dots, \tilde{g}_l$  of jump instructions

$$\tilde{g}_j : B^{r_j} \rightarrow \text{finite subsets of } V$$

We then require that  $v(v, b)$  for  $b \in B_1$  be of the form

- (i)  $(v'_1, \dots, v'_{n_j}) := g_j(v_1, \dots, v_{m_j})$  with each  $v'_i$  and  $v_t$  in  $V$ ; or
- (ii)  $J_{\tilde{g}_j(v_1, \dots, v_{r_j})}$  with each  $v_t$  in  $V$ ; or
- (iii) a pair of instructions, one of form (i) and one of form (ii)

and denote the set of such forms by  $\mathcal{J}(g_1, \dots, g_k; \tilde{g}_1, \dots, \tilde{g}_l)$ .

Finally, we must adopt a convention as to what happens when several values are being simultaneously loaded into one register. We adopt, completely arbitrarily, the convention that it retains its old value. Putting all the pieces together, we obtain the following general definition:

A parallel stored program machine (PSPM) is determined by a quintuple

$$M = [G = (V, E), B, \{g_1, \dots, g_k\}, \{\tilde{g}_1, \dots, \tilde{g}_l\}, B_1, v]$$

where  $G = (V, E)$  is a directed graph, the address structure of  $M$ ,  $B$  is the set of possible

*register contents*,  $\{g_1, \dots, g_k\}$  is a set of *operation labels*,  $\{\tilde{g}_1, \dots, \tilde{g}_l\}$  is a set of *jump labels*,  $B_1 \subset B$  is the set of *instruction encodings*, and

$$\nu : V \times B_1 \rightarrow \mathcal{J}(g_1, \dots, g_k; \tilde{g}_1, \dots, \tilde{g}_l)$$

is the *instruction decoding function*.

A state of  $\mathcal{C}_M$  is then given by specifying the contents of each register, and which registers are to be decoded for instruction execution. If  $V$  is infinite, we may allow only a finite set of registers to be “nonempty,” distinguishing a  $b_0 \in B$  to represent “empty contents.” Then  $\mathcal{C}_M$  has state set

$$\hat{Q}_2 = \{(f, S) | \{v | f(v) \neq b_0\} \text{ is finite, and } S \text{ is finite}\} \subset B^V \times \mathcal{P}(V).$$

We then define the next-state function  $\delta_M$  by taking

$$\delta_M(q, S) = (q', S')$$

where  $(q' S')$  is defined as follows:

- (a) We decode  $\nu(v, q(v))$  for each  $v \in S$ .
- (b) If decoding yields an operation of the form

$$(v'_1, \dots, v'_{n_j}) := g_j(v_1, \dots, v_{m_j})$$

we then take  $q'(v'_r)$ ,  $1 \leq r \leq n_j$ , to be the  $r$ th component of  $g_j(q(v_1), \dots, q(v_{m_j}))$ —save that, should another operation specify a different value for  $q'(v'_r)$ , we simply set  $q'(v'_r) = q(v_r)$ . In general, we set  $q'(v) = q(v)$  unless a new value is specified in the above manner by some  $\nu(v, q(v))$  for  $v \in S$ .

(c) If decoding yields a jump of the form  $J_{\tilde{g}_j(v_1, \dots, v_{r_j})}$ , we include each element of the finite set  $\tilde{g}_j(q(v_1), \dots, q(v_{r_j}))$  in  $S'$ . If  $v$  is in  $S$  and  $\nu(v, q(b))$  is a pure operation (i.e., of form i), then we also include  $v$  in  $S'$ .  $S'$  contains all and only the elements of  $V$  specified in the preceding two sentences.

There is a genuine sense in which  $\delta_M$  is a partial recursive function—so that  $\mathcal{C}_M$  is a recursive computer—so long as  $\nu$ , the  $g_j$ 's, and the  $\tilde{g}_k$ 's are partial recursive.

This formulation is very general. For example, it includes the *tessellation automata* [3a, Chap. 10] on an Abelian group  $V$  with neighborhood template  $\{g_1, \dots, g_n\}$  and transition function  $f : B^n \rightarrow B$  on taking

$$\nu(v, b) = [v_1 := f(v + g_1, \dots, v + g_n), J_{\tilde{g}_j(v)}]$$

for all  $v \in V$ , all  $b \in B$ , where  $J_{\tilde{g}_j(v)} = \{v, v + g_1, \dots, v + g_n\}$ . We return to this below.

Still other approaches to effective computations are due to Church, Post, and Markov.

## DIFFICULTY OF COMPUTATION

A Turing machine may blithely assure us that a problem is effectively solvable and that it will eventually solve it—but the Universe may disappear before it finishes the computation. One reaction to this contretemps is to search for more natural models

intermediate between finite-automata and full-fledged Turing machines, and to study the problem areas they characterize. Such is the work relating automata and context-free languages and the work on linear-bounded automata, trades-off between determinism and nondeterminism, number of tapes, etc. Here we simply present the highly influential axiomatic approach to complexity of computation due to Blum [4].

Let now  $Z_1, Z_2, Z_3, \dots$  be an enumeration of Turing machines with a fixed alphabet, and fixed choice of encoding and decoding functions; and let  $f_1, f_2, f_3, \dots$  be the corresponding enumeration of the partial recursive functions. Let  $F_n(x)$  be the number of steps  $Z_n$  takes to compute  $f_n(x)$ . (Thus, if we find  $m$  and  $n$  such that  $f_m = f_n$ , we will not expect, in general, that  $F_m(x) = F_n(x)$  for any given  $x$ .) The functions  $F_n(x)$  are partial recursive, and satisfy: (1) For all  $n$  and  $x$ ,  $f_n(x)$  is defined if and only if  $F_n(x)$  is defined, and (2) There exists a total recursive function  $\alpha$  such that for all  $n$  and  $x$

$$\alpha(n, x, y) = \begin{cases} 1 & \text{if } F_n(x) = y \\ 0 & \text{else} \end{cases}$$

To see (2), we use Turing's hypothesis, noting that we may find  $\alpha(n, x, y)$  effectively by supplying  $Z_n$  with input  $x$ , and letting it run for  $y$  steps.

Rabin [5] has shown there are functions which are arbitrarily hard to compute. We shall state a related result. First notice that, given  $h(x)$ , it is easy to find a function  $f$  such that no matter *what* machine  $Z_n$  computes it, we will have  $F_n(x) \geq h_n(x)$ , by having it take at least  $h(x)$  steps just to write  $f(x)$  on the tape. To make the question interesting, we must ask if there is a function which takes a long time to *compute*. The answer in the affirmative may be stated:

Let  $h(x)$  be any partial recursive function. There exists a function  $f(x)$ , defined just for those  $x$  for which  $h(x)$  is defined, *taking only the values 0 and 1*, and such that for any machine  $Z_n$  which computes  $f$ , we have  $F_n(x) \geq h(x)$  for all but finitely many  $x$ .

Now, suppose  $Z_i$  and  $Z_j$  both compute  $f$ . We shall say  $Z_j$  is no faster than  $Z_i$  if  $F_i(x) \leq F_j(x)$  for all but finitely many  $x$ . The latter phrase merely ensures that our comparison is not vitiated by any purely transient advantage that  $Z_j$  may have, e.g., in ordinary computer terminology, due to a *table look-up* which allows the machine to obtain  $f(x)$  very quickly for the finitely many  $x$  in the table.

One question that immediately arises is: Does every partial recursive function  $f$  have a fastest program, i.e., is there a  $Z_n$  computing  $f$  such that any  $Z_m$  computing  $f$  is no faster than  $Z_n$ ? The answer to this question is negative—there are very many functions with no fastest program. In fact, *Blum's speed-up theorem* says much more.

Let  $r(x, y)$  be any total recursive function. Let  $Z_m$  and  $Z_n$  both compute  $f$ . We shall say  $Z_m$  is an *r-speed-up* of  $Z_n$  if  $r(x, F_m(x)) < F_n(x)$  for all but finitely many  $x$ . Thus  $Z_m$  is faster than  $Z_n$  if it has an *e-speed-up*  $Z_n$  for  $e(x, y) \equiv y$ .

Since we may choose  $r(x, y)$  to grow quickly indeed, e.g.,

$$r(x, y) = 2^{3^{57x+y}}$$

we see that an *r-speed-up* of  $Z_n$  may be very much faster indeed.

The speed-up theorem tells us that no matter how large we choose  $r$  to be, we may find a 0-1 valued function  $f$  such that *any*  $Z_n$  for  $f$  has an *r-speed-up*  $Z_m$  which computes  $f$ , and so *ad infinitum*. This is a most surprising result, and the proof is a long one.

We close with a warning in which is implicit a whole program of research on difficulty of computation. We must discourage the reader who wishes to treat the speed-up theorem as a truth about real computers, rather than partial recursive functions—the  $r$ -speed-up fails for finitely many  $x$ , and this finitude may well contain all those integers (e.g.,  $0 \leq x < 2^{32}$ ) that we consider in a real computer. And yet the theorem is not without interest!

## CATEGORICAL MACHINE THEORY

Categorical machine theory embraces, among others, two convergent themes: the first unifying automata theory and system theory; the second applying category theory and universal algebras to the tree-processing problems which have proved central to the study of programming languages. A central problem in both automata theory and system theory has been the *realization problem*—namely, given a description of how a system processes inputs to yield outputs, to construct a *realization* of it, i.e., a state-variable description of a system which processes inputs in the specified way. Automata theorists concentrated on the cases in which the realization was a *finite-state* machine, while control theorists emphasized *finite-dimensional linear* realizations. Arbib and Zeiger [6] provided a presentation of the automata-theory of realization which yielded the results of the linear theory in a straightforward way which motivated and improved the results of the control theorists (e.g., Ho and Kalman [7]).

At this stage, it may help to summarize the finite-state case: a *finite automaton* is a quintuple  $(X, Y, Q, \delta, \beta)$  where  $X$  is a finite set (the set of inputs),  $Y$  is a finite set (the set of outputs),  $Q$  is a finite set (the set of states),  $\delta : Q \times X \rightarrow Q$  is the next state function and  $\beta : Q \rightarrow Y$  is the output function. Thus if the system is in state  $q$  and receives input  $x$  at time  $t$ , then its output pattern will be  $\beta(q)$  at time  $t$ , and it will change to state  $\delta(q, x)$  at time  $t + 1$ .

We introduced  $X^*$  as the set of all finite sequences of input symbols, and include in it  $\Lambda$ , the “empty string” of 0 symbols.  $X^*$  is a semigroup under concatenation [i.e., concatenation is associative  $(w_1 w_2) w_3 = w_1 (w_2 w_3)$ ] and has identity  $\Lambda$ . We then extend the applicability of  $\delta$  so that  $\delta$  maps  $Q \times X^*$  into  $Q$  by repeated application of the equality (Fig. 1):

$$\delta(q, w'w'') = \delta(\delta(q, w'), w'')$$

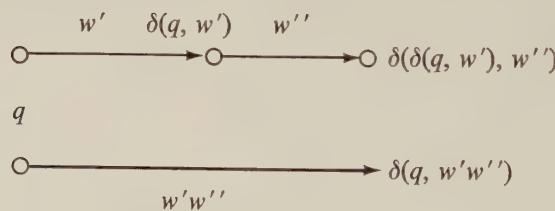


Fig. 1

We may associate with each state  $q$  of a machine  $M$  the way it produces an output for each input string. This is expressed by the function  $M_q : X^* \rightarrow Y$ , where  $M_q(w) = \beta[\delta(q, w)]$ . Clearly  $M$  behaves the same if started in two states  $q, q'$ , with the same input-output function, i.e., if  $M_q(w) = M_{q'}(w)$  for all input strings  $w$ . So if our interest in  $M$  is in its external behavior, we may replace it by its *reduced form* which has one state for each *distinct* function  $M_q$ . Let  $M$  in state  $q$  have input-output function  $f$ , and let  $q'$  be a state *reachable* from  $q$ , i.e., we can find  $w \in X^*$  such that  $\delta(q, w) = q'$ . Then  $q'$  has function  $g = M_{q'}$ , where

$$\begin{aligned} M_{q'}(w') &= M_{\delta(q, w)}(w') = \beta[\delta(\delta(q, w), w')] = \beta[\delta(q, ww')] \\ &= M_q(ww') = f(ww') = fL_w(w') \end{aligned}$$

where  $L_w : X^* \rightarrow X^*$  is the “left multiplication by  $w$ ” function:  $L_w(w') = ww'$ .

Then in our reduced form for  $M$ , we replace each state reachable from  $q$  by its input-output function  $fL_w$ —and the reduced form (restricted to states reachable from  $q$ ) has finitely many states just in case there are only finitely many distinct functions  $fL_w$  (so that infinitely many strings  $w \in X^*$  must yield the same  $fL_w$ ). Thus our interest centers on machines of the form

$$M_f = (X, Y, Q_f, \delta_f, \beta_f)$$

where  $f$  is a given function mapping  $X^*$  to  $Y$ , and where

$$\begin{aligned} Q_f &= \{g : X^* \rightarrow Y \mid g = fL_w \text{ for some } w \in X^*\} \\ \delta_f(g, x) &= gL_x \\ \beta_f(g) &= g(\Lambda) \end{aligned}$$

especially in the case when  $f$  is such that  $Q_f$  is a finite set.

With such a machine  $M_f$ , we associate a semigroup  $S_f$ , namely the collection of transformations of the state set  $Q_f$  induced by the input strings to  $M_f$ , i.e.

$$S_f = \{s : Q_f \rightarrow Q_f \mid \exists w \in X^* \text{ such that } s(q) = \delta(q, w) \text{ for all } q \in Q_f\}$$

This semigroup is finite if and only if  $Q_f$  is finite. If  $M$  is not in reduced form, we associate with it (and some specified starting state  $q$ ) the semigroup of the reduced machine  $M_f$  where  $f$  is just  $M_q$ .

We may introduce a function  $i_f : S_f \rightarrow Y$  by the definition  $i_f(s) = f(w)$  if  $s(q) \equiv \delta(q, w)$ . This does not depend on the choice of  $w$ . We may then define a new “semigroup machine” with “state-output”  $i_f$  as

$$M(S_f, i_f) = (S_f, S_f, Y, \delta, \beta)$$

where  $\beta(s, s') = s \cdot s'$  (where the  $\cdot$  denotes semigroup multiplication) and  $\beta(s) = i_f(s)$ .

We may also introduce the semigroup machine of  $S_f$  itself as simply  $M(S_f) = (S_f, S_f, S_f, \dots)$ .

Along a different line, computer scientists had been building on the theoretical linguistics of Chomsky to provide a formalism for programming languages which made use of derivation trees. However, for a long time the theory concentrated on the manipulation of strings, rather than the manipulation of the underlying trees. However, this situation changed with the paper of Thatcher and Wright [8] who showed, *inter*

*alia*, that ordinary automata theory could be generalized to yield a theory of tree automata. Subsequent papers, e.g., Rounds [9] and Thatcher [10], convinced many workers that programming language theory should be couched in terms of tree manipulation. Meanwhile, Eilenberg and Wright [11] showed the appropriateness of categorical notions due to Lawvere [12] to tree theory.

An approach [13a] to automata theory which unifies these two themes can be briefly presented here. (The reader unfamiliar with category theory may find an exposition in Arbib and Manes [13b].)

Let  $\mathcal{K}$  be a category, and  $X : \mathcal{K} \rightarrow \mathcal{K}$  be a functor. We define  $\text{Dyn}(X)$  to be the category whose objects are “dynamics with input  $X$ ” given by morphisms

$$QX \xrightarrow{\delta} Q \text{ in } \mathcal{K},$$

and whose morphisms are “dynamorphisms,” i.e., a dynamorphism

$$[QX \xrightarrow{\delta} Q] \xrightarrow{f} [Q'X \xrightarrow{\delta'} Q']$$

is a  $\mathcal{K}$ -morphism

$$Q \xrightarrow{f} Q'$$

for which we have the commutativity

$$\begin{array}{ccc} QX & \xrightarrow{\delta} & Q \\ \downarrow fX & \delta' & \downarrow f \\ Q'X & \xrightarrow{\delta'} & Q' \end{array}$$

In ordinary automata theory, we take  $\mathcal{K} = \mathcal{S}$ , and  $X = - \times X_0$ . A dynamics is then a map

$$Q \times X_0 \xrightarrow{\delta} Q$$

and we have the usual dynamorphism condition

$$\begin{array}{ccc} Q \times X_0 & \xrightarrow{\delta} & Q \\ f \times id_{X_0} & \downarrow & \downarrow f \\ Q' \times X_0 & \xrightarrow{\delta'} & Q' \end{array}$$

Note that in this case we can always define a *free* machine

$$(Q \times X_0^*) \times X_0 \xrightarrow{Q\mu_0} Q \times X_0^* : ((q, w), x) \mapsto (q, wx)$$

together with a map

$$Q \xrightarrow{Q\eta} Q \times X_0^* : q \mapsto (q, \Lambda)$$

so that we have the “universal property” that for each  $f$  and  $\delta'$  we can find exactly one dynamorphism  $\psi$  such that we have:

$$\begin{array}{ccc} Q & \xrightarrow{Q\eta} & Q \times X_0^* \\ \downarrow f & \downarrow \psi & \downarrow \psi \times id_{X_0} \\ Q' & & Q' \times X_0 \\ & & \xrightarrow{\delta'} Q' \end{array}$$

We use this property of the classical case as the defining property of our general theory: We say that a functor  $X : \mathcal{K} \rightarrow \mathcal{K}$  is an *input process* just in case the forgetful functor  $Dyn(X) \rightarrow \mathcal{K} : (Q, \delta) \rightarrow Q$  has a left adjoint  $Q \rightarrow (QX^\circledast, Q\mu_0)$ .

We then define a *machine in a category  $\mathcal{K}$*  to be a septuple

$$M = (X, Q, \delta, I, \tau, Y, \beta)$$

where  $X$  is an input process in  $\mathcal{K}$ ,  $(Q, \delta)$  is an  $X$ -dynamics,  $\tau : I \rightarrow Q$  is a  $\mathcal{K}$ -morphism ( $I$  is called the *initial state object*, and  $\tau$  the *initial state*, of  $M$ ), and  $\beta : Q \rightarrow Y$  is a  $\mathcal{K}$ -morphism ( $Y$  is called the *output object*, and  $\beta$  the *output map*, of  $M$ ).

This definition not only includes ordinary automata (“state-behavior” machines with  $\mathcal{K} = \mathcal{S}$ , the category of sets and functions), but also linear systems (“decomposable machines” with  $\mathcal{K} = \text{modules over a fixed ring } R$ ), tree automata (arbitrary machines with  $\mathcal{K} = \mathcal{S}$ , this being a nontrivial theorem), and stochastic automata (special  $\mathcal{K}$  made to order). It is expected that a continued study will both elucidate old ideas and create new ones. For example, “state behavior” machines clarify which properties of ordinary automata are *not* expected to carry over to, say, tree or stochastic automata. “Decomposable” machines lay bare the principles of linear systems theory to the extent that it is obviously possible to do the same thing in just about any category (countable coproducts being a more-than-sufficient formal requirement). This extends the approach to group machines of Arbib [14]. This is in contradistinction to much recent literature (e.g., Brockett and Willsky [15]) in which linear systems are imitated too literally. The confusion is due to two coincidences, specifically (1) a linear system  $F : Q \rightarrow Q$ ,  $G : X \rightarrow Q$  is the same thing as a linear map  $Q \times X \rightarrow Q$ , and (2) the free monoid  $X^*$  shares the same underlying set as that  $R$ -module which is the countably infinite weak direct sum of copies of  $X$ .

## DECOMPOSITION THEORY

We often try to realize the behavior of a system in terms of an interconnection of simpler subsystems. Hedetniemi [16] has asked “What is a decomposition theory?” and identified four major components: A collection of machines to be decomposed, a concept  $\mathcal{S}$  of simulation, a class  $\mathcal{P}$  of primitives, and a class  $\mathcal{C}$  of compositions whereby machines may be constructed from the primitives, and from aggregates so formed from those primitives. The goal of such a theory is usually a theorem of the

form “for any  $M$  in  $\mathcal{M}$ , there exists a composition  $C(M)$  using primitives of  $\mathcal{P}$  and composition rules of  $\mathcal{C}$  such that  $C(M)$  simulates  $M$ .” We shall often want accompanying theorems telling us whether there exists some algorithm for constructing  $C(M)$ , telling us the extent to which the machine so constructed is unique, and identifying whether the pieces from which  $C(M)$  has been constructed are in some useful sense simpler than the original pieces. As we shall discuss further below, this makes us realize that an important element of our theory will be the identification of our criteria for simplicity.

We may immediately realize that *realization theory* is a special case of the general decomposition theory, with  $\mathcal{M}$  a set of *behaviors*,  $\mathcal{P}$  a set of structures, no compositions beyond the identity composition in  $\mathcal{C}$ , and the concept of simulation  $\mathcal{S}$  being simply that which takes us from a structure to its behavior. The general problem here reduces to finding a structure  $\mathcal{P}$  that exhibits the behavior described by some given  $\mathcal{M}$ . This special case reminds us that any adequate theory will combine both *structural* and *behavioral* descriptions. In a realization theory,  $\mathcal{C}$  will tell us how to form a structure to get a desired behavioral result.

## TESSELLATION AUTOMATA

We have seen that it is possible to program a universal Turing machine which, given for any Turing machine  $Z$ , a suitable encoded description  $e(Z)$ , would simulate any computation which  $Z$  could execute. This suggested to von Neumann the existence of an automaton  $A$  which when furnished with the description of any other automaton  $M$  (composed from some suitable collection of elementary parts) would construct a copy of  $M$ . He outlined how one could use such a “universal constructor”  $A$  to solve the problem of self-reproduction: To build an aggregate out of our elementary parts in such a manner that “if it is put into a reservoir in which there float all these elements in large numbers, it will then begin to construct other aggregates, each of which will at the end turn out to be another automaton exactly like the original one” [17].

However, there is a great gap between the known universal computer and the posited universal constructor. For our universal Turing machine only simulates symbolic operations—any physical realization of such a machine would use components of a nature and complexity quite different from that of the tape symbols. This point requires some emphasis. Von Neumann’s 1951 paper has left many people with the impression that there is no new logical problem here—that the existence of a self-reproducing machine is reducible to an instance of the recursion theorem of recursive function theory. This is not so. There are two *fixed point theorems* to prove.

One might suspect that, given a list of elementary parts, a universal constructor for machines comprising only those parts might itself have to be constituted of a more complex variety of components. The proof that this suspicion is wrong gives us the first fixed point theorem: There does exist a set of components from which may be built a universal constructor for automata built from that very set of parts.

The universal constructor  $A$ , supplied with a description of an automaton, will build

a copy of that automaton. However,  $A$  is *not* self-reproducing:  $A$ , supplied with a copy of its own description, will build a copy of  $A$  *without* its own description. The passage from a universal constructor to a “self-constructor” is, in essence, a fixed point theorem reducible to the recursion theorem.

Von Neumann gave the first construction of a self-reproducing automaton in a manuscript on “The Theory of Automata: Construction, Reproduction, Homogeneity” which was incomplete at the time of his death, and which A. W. Burks has since edited for publication by the University of Illinois Press. He replaced the reservoir in which the organism floated by a “tessellation” of identical cells, which were finite automata with only 29 states [18].

The trouble is that it is very complicated to program something with such simple components. For instance, you can’t just have two wires crossing. One way of solving the cross-over problem is to code messages so that, when a message comes to a crossing, it goes both ways, but only a decoder in the desired direction will be able to “understand” the message. Another way is to introduce a large array of cells such that if you put a pulse in any corner, it will come out of the opposite one. Thatcher [19] has since produced a polished version of von Neumann’s scheme, using the same set of components.

Arbib [20] has shown how to reduce the complexity of the programming immensely, but only at the cost of more complicated *modules*. The living cell, with its synthetic machinery involving hundreds of metabolic pathways, can rival any operation of our module, as well as being under the control of DNA molecules, we believe, with far more bits of information than our cell can store. So, perhaps we lose biological significance by unduly limiting the information content of the module.

## LOOP-FREE COMPOSITION

We say that the finite automaton  $M$  *simulates* the machine  $M'$  if, provided we encode and decode the input and output appropriately,  $M$  can process strings just as  $M'$  does. We require the encoder and decoder to be memoryless (i.e., operate symbol by symbol) in order to make  $M$  do all the computational work involving memory.

If  $M$  simulates  $M'$ , we write  $M'|M$  and say that  $M'$  *divides*  $M$ . If both  $M|M'$  and  $M'|M$ , we say that  $M$  and  $M'$  are *weakly equivalent*. The utility of semigroups in finite automata theory is emphasized by the result that  $M(S_f, i_f)$  is weakly equivalent to  $M_f$ . It turns out that semigroups, as well as machines, have a natural concept of divisibility. We say a semigroup  $S$  *divides* a semigroup  $S'$  if there is a subsemigroup  $S''$  of  $S$  of which  $S'$  is the image under a homomorphism  $z$ :

$$S' \xrightarrow{z} S'' \rightarrowtail S$$

If  $S$  and  $S'$  have associated maps  $i : S \rightarrow Y$ ,  $i' : S' \rightarrow Y'$ , then we say that  $(S, i)$  *divides*  $(S', i')$  if the situation above holds, and there is a further mapping  $H : Y \rightarrow Y'$

such that

$$i(z(s)) = H(i'(s)) \text{ for all } s \in S$$

The important tie-up between the machine and semigroup concepts of divisibility is that the machine  $M_g$  divides the machine  $M_f$  with each started in their initial state, if and only if the pair  $(S_g, i_g)$  divides the pair  $(S_f, i_f)$ .

It is well known that any finite automaton may be simulated by a network of *modules* (i.e., *one-state* finite automata) provided we allow loops in the network—such constructions form the central theme of any text on switching theory. In fact, we may use copies of just one module (the *Sheffer stroke module*) to build such a network. In other words, a very simple set of components suffices for the construction of arbitrary finite automata *if loops are allowed*. In this section we describe some theorems on the restrictions following from the outlawing of loops between machines (of course, there may be loops in the internal structure of the machines we are combining).

Given machines  $M'$  and  $M$ , and a map  $z : \hat{X} \times Y \rightarrow X'$ , we define  $M' \times {}_z M$ , the semidirect product of  $M'$  and  $M$  with connecting map  $z$  by letting  $M$  receive input directly while  $M'$  receives system input in  $\hat{X}$  combined with the output in  $Y$  of  $M$ , via the map  $z$ . To get *series* composition (albeit preserving the output of  $M$ ) we make  $z$  independent of  $X'$ ; to get *parallel* composition we take  $z(x', y)$  to be just  $x'$ .

Our precise notion of loop-free composition is that of repeated formation of semi-direct products, as well as memoryless codings to obtain machines simulable by such combinations. We let  $SD(M)$  denote the set of machines obtainable in this fashion, from a set  $M$  of machines.

We say that a machine  $N$  is *irreducible* if for any collection  $M$  of machines for which  $N \in SD(M)$ , there actually exists a machine in  $M$ , with semigroup  $S$ , such that  $M(S)$  simulates  $N$ , i.e.,  $N$  cannot be replaced by a combination of machines with “smaller” semigroups.

We call  $M$  an *identity-reset machine* if an input either acts as a reset, returning the machine to a state determined by that input ( $\delta(q, x) = q_x$ , all  $q \in Q$ ) or else acts as an identity, leaving the state unchanged ( $\delta(q, x) = q$ , all  $q \in Q$ ).

Of special importance is the two-state identity reset machine, which we call the “flip-flop”  $F$ : It has states  $\{q_0, q_1\}$  and inputs  $\{e, x_0, x_1\}$  with  $\delta(q, e) = q$  ( $e$  is the identity), and  $\delta(q, x_i) = q_i$  ( $x_i$  resets to state  $q_i$ ); and with state and output equal. We note that any identity-reset machine may be obtained by loop-free synthesis from copies of  $F$ .  $F$  has semigroup  $U_3 = \{1, r_0, r_1\}$  whose elements are the state-maps corresponding to  $e$ ,  $x_0$ , and  $x_1$ , respectively. We let  $UNITS$  denote the set of semigroups which divide  $U_3$ . In fact,

$$\text{UNITS} = \{U_0, U_1, U_2, U_3\}$$

where  $U_0 = \{1\}$ ,  $U_1 = \{1, r_0\}$ , and  $U_2 = \{r_0, r_1\}$ . Then we owe to Krohn and Rhodes [21] the following elegant results:

1. A machine  $M_f$  is irreducible if and only if  $S_f$  is an element of  $UNITS$  or is a simple group (i.e., has no nontrivial normal subgroup).
2. Given a machine  $M_f$ , we can realize it by loop-free synthesis from flip-flops and from the machines of (not necessarily all) the simple groups which divide semigroup  $S_f$ .

## COMPLEXITY OF FINITE FUNCTIONS

Consider networks made of elements that have a unit delay in their operation and that compute any Boolean function; in other words, given any configuration of 0's and 1's on their inputs, one moment of time later they produce a 0 or 1. (They may be threshold elements, for which each input line has some associated weight  $W_i$ , and the overall element has some associated threshold 0. At any moment of time we take the weighted sum  $\sum W_i X_i$  of the input values to determine whether the output should be 1 or 0 one moment of time later determined by whether or not that sum exceeds 0.) Later we shall assign other tasks to such networks, but for now let us consider pattern recognition, and consider the input to be a spatially arrayed collection of squares rather than to be a collection of lines. A two-dimensional pattern upon this array will be quantized so that each square either records a 1, if it is relatively bright, or a 0, if it is relatively dark. Then there is a layer of components that can compute arbitrary Boolean functions, each one connecting to some subset of the squares on the "retina" above. If this single layer of associator elements is read out by a single threshold element, the resultant structure is a "one-layer" Perceptron which classifies patterns on the retina into those that yield an output 1 and those that yield an output 0.

Rosenblatt and others asked, "Given a network, can we 'train' it to recognize a given set of patterns by using feedback, on whether or not the network classifies a pattern correctly, to adjust the 'weights' on various interconnections?" The answers (summarized in Nilsson [22]) have mostly been of the type, "Well, if a setting exists that will give you your desired classification, I guarantee that my scheme will eventually yield a satisfactory setting of the weights." Minsky and Papert [23] went on to ask, "Given a pattern recognition problem, how much of the retina must each associator unit 'see' if the network is to do its job?" They analyze this question both for "order-limited Perceptrons," in which the "how much" is the number of retinal units to which an associator unit may be connected, and for "diameter-limited Perceptrons," in which "how much" is specified in terms of the largest diameter of any retinal region to which an associator unit may be connected.

Let us now place this work in the context of network complexity by recalling a very simple observation due to Winograd that has surprisingly powerful consequences. Suppose that we are using networks made of arbitrary Boolean elements and we look at a box that has two sets of input lines coming in and a set of output lines coming out. We imagine that on the first set of input lines we code the members of some set  $X_1$ , on the second set of input lines we code the members of set  $X_2$ , and on the output lines we code the members of some set  $Y$ . The idea then is that we represent some element  $x_1$  in  $X_1$  on the first set of lines,  $x_2$  in  $X_2$  on the second set, and we hold that configuration of inputs over a period of time. We want the network to be such that after this period the encoding of some function  $\phi(x_1, x_2)$  will appear on the output. Then the question we might ask is, "How long must it take for this to happen?": If we limit the elements to only have two inputs, an output line whose signal depends on the values of eight input lines (i.e., one of those input lines could not be destroyed without leading to an occasional miscomputation) comes from a neuron that can only

be affected by at most two inputs and, similarly, each of those two input neurons can only be affected by at most two inputs; so, working back in this way, we see that the computation time required to compute the output cannot be less than three simply because of the problems of fan-in. In fact, if the function is even more complicated, we might well expect more time is required for various things to be moved back and forth.

Winograd was able to show for Abelian groups [24] and Spira was able to show for arbitrary groups [25] that not only did such considerations yield a lower bound on the time required for group multiplications, but, in fact, they could build networks (with elements of the specified complexity, e.g., two input lines per component) that would multiply within one unit of time of that lower bound. To get this fast computation requires a very redundant encoding of the inputs. A simple example of how computation time depends upon encoding is that of multiplying two numbers together. If we multiply numbers in the ordinary decimal notation, all the inputs have to interact. However, if we multiply together numbers that are expressed simply as a string of numbers, with the  $j$ th prime in the prime decomposition of the encoded number, then multiplication becomes much easier. We simply add the exponents. The operation is very fast, but the encoding is very long. We thus conclude that there is a tradeoff between the number of components in each layer and the number of layers or time of computation.

Winograd and Spira stressed that, if we bound the number of inputs per component, we can then proceed to discover how many layers of components we require. Minsky and Papert have tackled the complementary problem of asking, "If we fix the number of layers in the network, how complicated must the elements become in order to get a successful computation?" Minsky and Papert were able to show, surprisingly enough, that a one-layer Perceptron which computes so simple a function as parity—i.e., "give output 1 if the number of squares illuminated is odd, and give output 0 if the number of squares illuminated is even"—requires at least one neuron that is connected to all of the inputs.

## REFERENCES

1. A. M. Turing, On computable numbers, *Proc. London Math. Soc.*, [2]42, 230–265 (1936).
2. E. L. Post, Finite combinatory processes—Formulation I, *J. Symbolic Logic* 1, 103–105 (1936).
- 3a. M. A. Arbib, *Theories of Abstract Automata*, Prentice-Hall, Englewood Cliffs, N.J., 1969.
- 3b. K. Čulík and M. A. Arbib, Sequential and jumping machines and their relation to computers, *Acta Inf.* 2, 162–171 (1973).
4. M. Blum, A machine-independent theory of recursive functions, *J. Assoc. Comput. Mach.* 14, 322–336 (1967).
5. M. O. Rabin, *Degree of Difficulty of Computing a Function and a Partial Ordering of Recursive Sets*, Hebrew University, Jerusalem, Israel, 1960. [For an abstract, see his Speed of computation of functions and classification of recursive sets, *Bull. Res. Council Israel*, 8F, 69–70 (1959).]
6. M. A. Arbib and H. P. Zeiger, On the relevance of abstract algebra to control theory, *Automatica* 5, 589–606 (1969).
7. B. L. Ho and R. E. Kalman, Effective construction of linear state-variable models from input/output functions, *Regelungstechnik* 14, 545–548 (1968).
8. J. W. Thatcher and J. B. Wright, Generalized finite automata theory, *Math. Syst. Theory* 2, 57–81 (1968).

9. W. C. Rounds, Mappings and grammars on trees, *Math. Syst. Theory* **4**, 257–287 (1970).
10. J. W. Thatcher, Generalized<sup>2</sup> sequential machine maps, *J. Comput. Syst. Sci.* **4**, 339–367 (1970).
11. S. Eilenberg and J. B. Wright, Automata in general algebras, *Inform. Control* **11**, 452–470 (1967).
12. F. W. Lawvere, Functorial Semantics of Algebraic Theories, Ph.D. Thesis, Columbia University, 1963.
- 13a. M. A. Arbib and E. G. Manes, Machines in a category, *SIAM Rev.* **16**, 163–192 (1974).
- 13b. M. A. Arbib and E. G. Manes, *Arrows, Structures, and Functors: The Categorical Imperative*, Academic, New York, 1975.
14. M. A. Arbib, Coproducts and decomposable machines, *J. Comput. Syst. Sci.* **7**, 278–287 (1973).
15. R. Brockett and A. H. Willsky, Finite-state group homomorphic sequential machines, *IEEE Trans. Automatic Control* **AC-17**, 483–490 (1967).
16. S. Hedetniemi, *What is a Decomposition Theory of Automata?* Technical Report No. 34, Thesis Project on the Theory and Applications of Automaton Theory, The University of Iowa, 1970.
17. J. von Neumann, The general and logical theory of automata, in *Cerebral Mechanisms in Behavior, Proceedings of the Hixon Symposium* (L. A. Jeffress, ed.), Wiley, New York, 1951.
18. J. von Neumann, *The Theory of Self-Reproducing Automata* (A. W. Burks, ed.), University of Illinois Press, Champaign, Ill., 1966.
19. J. W. Thatcher, Universality in the von Neumann cellular model, in *Cellular Automata* (A. W. Burks, ed.).
20. M. A. Arbib, A simple self-reproducing universal automaton, *Inform. Control* **9**, 177–189 (1966).
21. K. B. Krohn and J. L. Rhodes, Algebraic theory of machines, I, *Trans. Amer. Math. Soc.* **116**, 450–464 (1965).
22. N. Nilsson, *Learning Machines*, McGraw-Hill, New York, 1966.
23. M. Minsky and S. Papert, *Perceptrons*, M.I.T. Press, Cambridge, Mass., 1969.
24. S. Winograd, On the time required to perform multiplication, *J. Assoc. Comput. Mach.* **14**, 793–802 (1967).
25. P. M. Spira, The time required for group multiplication, *J. Assoc. Comput. Mach.* **16**, 235–243 (1969).

*Michael A. Arbib*

## AUTOMATIC INDEXING: PROGRESS AND PROSPECTS

An index is an array of symbols, systematically arranged, together with a reference from each symbol to the physical location of the item symbolized.

Taube [1]

Automatic indexing is the use of machines to extract or assign index terms without human intervention once programs or procedural rules have been established.

Stevens [2]

The problem of automatic analysis reverts to the problem of automatic *translation*, where the target language is no longer a natural language but an artificial one, constructed for the expression of the document content.

Gardin [3]

## THE NATURE AND DEFINITION OF INDEXING

### Introduction

To R. L. Collison [4], "The trouble with indexing is that even today we are still at the elementary stage of learning how to do it. We do not know enough about its technique . . ." and we certainly do not know enough about its theory. Indexing and its associated paraphernalia constitute a strange process. Consequently, the researcher and reviewer are confronted by an interesting situation: on the one hand, examples of the product of indexing—the index—are plentiful and ubiquitous; on the other hand, attempts to formalize either the process of indexing or the relationship between its exemplars are virtually nonexistent. Unfortunately, this lack of order in (or, should we say, lack of a science of) the fields of indexing and indexing theory is reflected in the diversity of studies in automatic indexing.

It is not the goal of this article to present an exhaustive coverage of the field of automatic indexing. Rather, attention is directed to an analysis and critique of present-day approaches and a discussion of some possible future directions in automatic indexing. Should further in-depth information be required, the reader is referred to the following comprehensive state-of-the-art reports: Edmundson and Wyllis [5], Artandi [6], Stevens [2, 7], and the System Development Corporation [8]. Additionally, the *Annual Review of Information Science and Technology* [9] contains considerable discussion related to advances in automatic indexing.

### *Plan of the Article*

After a brief consideration of the various levels of mechanized or automatic indexing, we shall consider the role of the *index* and of the *indexing system* in the process of information storage and retrieval. Following a formal definition of indexing, we shall discuss and present examples of current techniques of automatic indexing with respect to the operations of document input, document analysis, index entry creation, and index compilation. Finally, we shall consider why the achievement of truly automatic indexing is dependent on advances from current research in document representation.

### **Indexing: From Manual to Automatic**

The history of indexing is a history of manual indexing. Developments in manual indexing have followed an unbroken line from the literary expansion that took place during the Hellenistic period of ancient Greece to the present-day development of specialized information centers. Despite its lengthy history, indexing remains much of an art. It is only within the last decade that this artful practice has been placed under the scrutiny of the scientific method. We believe that advancements in the formalization of indexing will come from the extension of indexing theory [10–13] and further in-depth studies of the composition and growth patterns of actual large indexing systems.

Unfortunately or fortunately, depending on whether one is an historian or a reviewer, automatic indexing does not have a lengthy history. Actually, development of techniques of automatic indexing have, by and large, followed developments in computer processing technology. In the way of a milestone, the shift from number-oriented to character-oriented processing (in both hardware and software developments) gave impetus to research in the automatic indexing of textual data. However, as the following opinions by Artandi [14] and Carroll [15] suggest, the general level of success of automatic indexing techniques is open to criticism:

Computer-aided indexing methods are mainly concerned with the computer manipulation of the product of manual indexing. Automatic indexing and extracting methods are still largely limited to the definition of content through text characteristics [14].

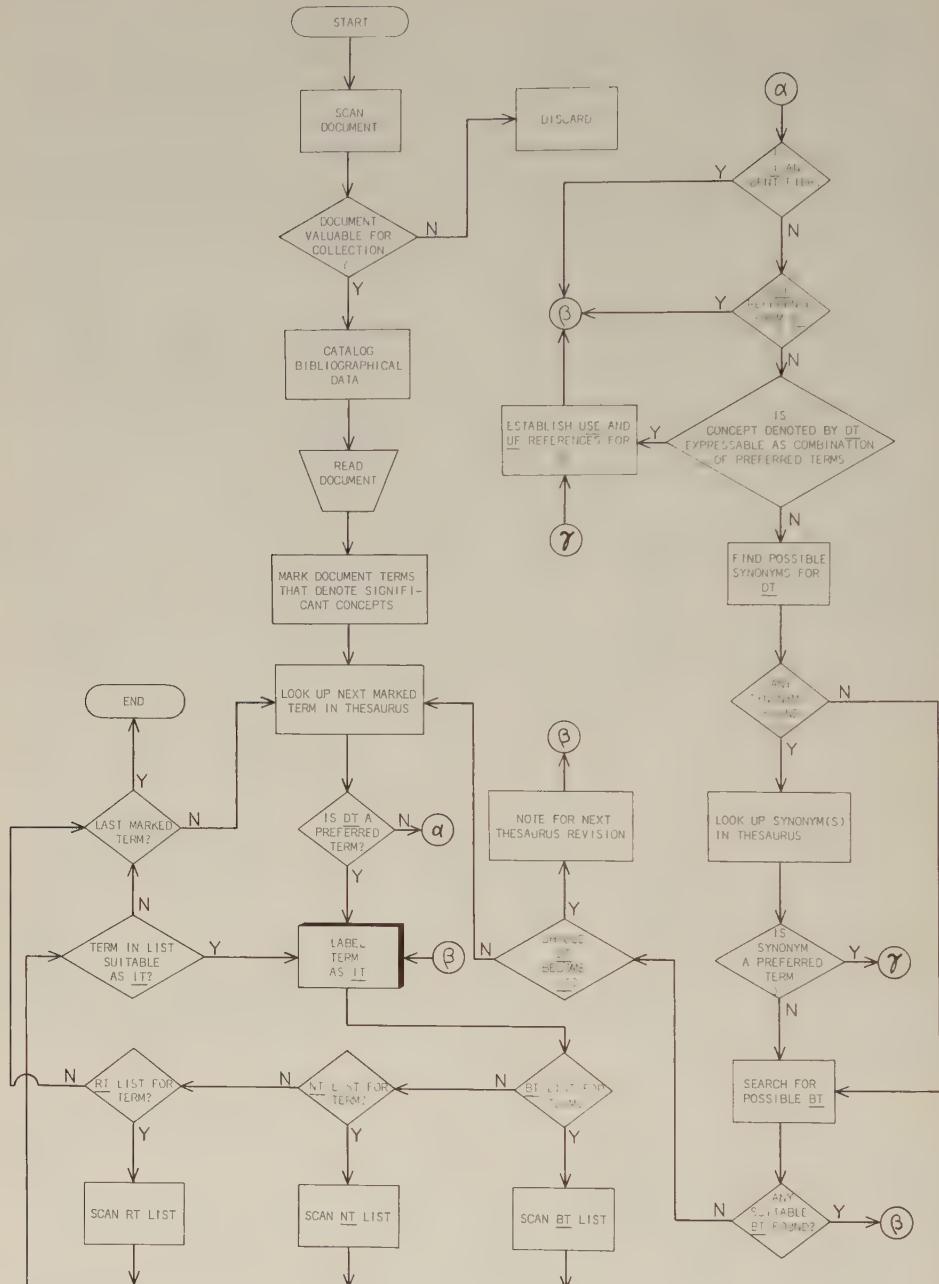
In most classification or indexing procedures, some intelligent being still must read every document acquired and make some rational judgment regarding its content. The ultimate goal of automatic indexing is to make these steps unnecessary [15].

Even to the not-so-casual observer, there seems to be some confusion as to the goals of automatic indexing: should attempts be made to automate the operations of the human indexer? Or, should attempts be directed toward the automation of some representational scheme? Or, rather, should automatic indexing research be directed toward a combination of these two approaches? We shall return to this problem in the section entitled *Toward Automatic Indexing*.

The term automatic indexing is really a misnomer. The concept "automatic" is usually associated with the concept algorithmic.<sup>1</sup> An algorithm, as a logical, well-defined sequence of instructions or operations, could be implemented by either a human or by a machine. For example, the algorithm depicted in Fig. 1 could be applied to documents by either a human indexer or a computer if the document were in suitable machine-readable form. Consequently, for the purpose of the discussion in this article, we shall restrict our attention to *computer-based* indexing.

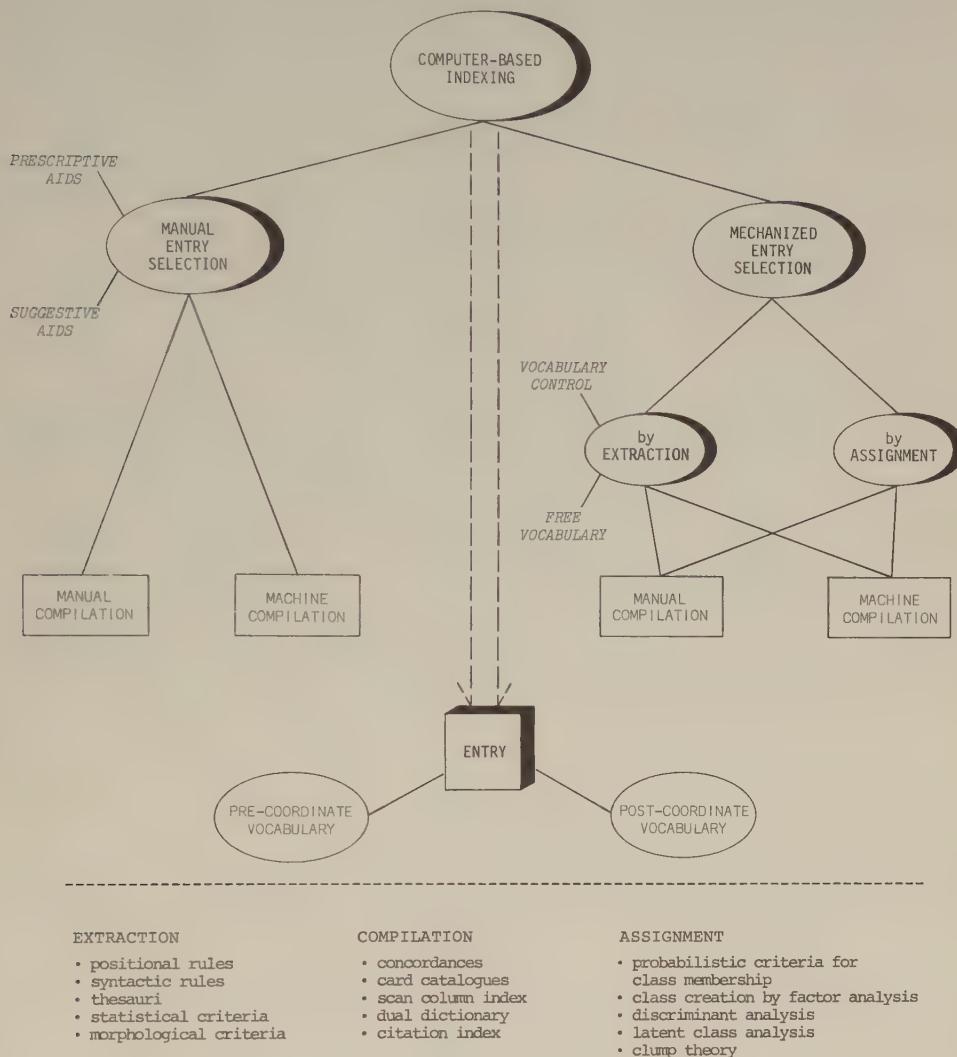
At this point one may wonder what are the possible implementations of computer-based indexing? Figure 2 presents a hierarchy of the possible uses of computer-based indexing involving either manual or mechanized (via computer-applied algorithms) index-entry selection. In terms of the sophistication of the computer algorithms, the hierarchy spans the range from simple machine compilation of *manually* selected entries to the machine compilation of *mechanically* assigned entries (see Stevens [7]). In mechanical entry selection, index terms are either *extracted* from the text of the document or *assigned* (from a master list) based upon the identification of keywords in the text of the document. In the case of extraction, the index terms either are unaltered text words (*free vocabulary*) or are created as the result of *prescribed* rules of usage (examples include synonyms, use of generic or related terms, removal of plural endings). When entry selection is effected by means of assignment rules, the index terms, by definition, are the result of the application of a *controlled vocabulary*. In computer-based indexing systems, an index entry may be formed using either multiple terms or a single term. In the case of a multiple-term entry (used as a con-

<sup>1</sup> Loosely, an algorithm is a procedure that terminates in some finite time [16].



**Fig. 1.** A flow chart for indexing with a thesaurus.

junction of several concepts), we describe the vocabulary of the index as being *pre-coordinate*; in the case of the uniterm entry, we describe the vocabulary of the index as being *postcoordinate* (i.e., the creation of a multiterm concept is done through the intersection of index entries at the time of the search of the index).



**Fig. 2.** Hierarchy of the uses of computer-based indexing with examples of the operations of extraction, compilation, and assignment.

With this brief overview of the various possible implementations of computer-based indexing techniques, we direct our attention to some pertinent history. Table 1 presents some of the important developmental work that has contributed to the

**TABLE 1**Landmarks in Computer-Based Indexing<sup>a</sup>

Procedure	Text			Nontext		
	KWOC	Contextual		Contextual		Relational
		KWIC	Relational	KWOC	KWIC	
Manual	Classification schemes	Latin squares	Roget's Thesaurus			
			Dictionaries			
	Concordances		Concordances			
Computer-based	Luhn 1957	Luhn 1959	Bernier (thesaurus) 1957	Fugmann GREMAS 1963	ISI Rotaform 1963	Mooers 1951
	Baxendale 1958	Many variations	Doyle (semantic maps) 1962	CAS screen generation 1965	ISI Wiswesser 1968	Meyer 1962
	Maron 1961	"	Gardin (SYNTOL) 1964		Petrarca <i>et al.</i> 1971	Gluck, Rasmussen 1963
	Edmundson, Wyllys 1961	"	Hays 1964			Morgan 1965
	Salton 1969	"	Quillian 1968			
	Borko 1970	Petrarca, Lay 1969	Shank, Tessler 1969			
			Fugmann 1970			
			Rush 1972			

<sup>a</sup> This table is not exhaustive; no judgmental value should be associated with exclusion.

current state-of-the-art of computer-based indexing. Many of these research efforts are discussed in the section entitled Examples of Computer-Based Indexing. We have chosen, for ease of presentation, to make the distinction between computer-based indexing of textual data and of nontextual data. By this partition we draw a distinction

between textual document processing and, for instance, chemical formula/structure processing. In each case the object is to create index entries that in some manner serve to represent the original document.<sup>2</sup> Furthermore, within each category of document processing we have found it convenient to make a distinction between index entries that are independent of the context of the original document and those entries that preserve some degree of the context of the original document. Hence

*Noncontextual index entries:*

**KWOC:** Key Word Out of Context—uniterm or precoordinated terms.

*Contextual index entries:*

**KWIC:** Key Word In Context—a keyword within a display of some portion of its contextual environment in the source document.

**RELATIONAL:** A keyword with a display of some portion of its contextual/relational environment in the source text.

It is interesting that the conceptual operations associated with textual relational indexing are the reverse of those associated with nontextual relational indexing. In textual relational indexing one starts with a linear string (sentence) of data elements and the object is to create a structure that represents or identifies the relations between the elements in the linear string; in nontextual relational indexing the goal is to derive index entries from a linearization of complex (relational) structures. The problem of the linearization of chemical structures has been essentially solved (see Morgan [18] and Davis and Rush [19]); however, the identification of textual relations is, in contrast, in its infancy. We will return to the problem of document representation in the section entitled Toward Automatic Indexing.

Before considering specific examples of computer-based indexing techniques, we shall attempt to be a little more precise about the nature and role of indexing in the process of information transfer.

### Communication, IS&R, and Indexing

We shall present, without extensive introduction or motivation, some basic definitions which are germane to the analysis of computer-based indexing techniques:

**Data element:** A data element is the smallest entity which can be recognized as a discrete element of that class of entities named by a specific attribute for a given unit of measure with a given precision of measurement.

**Document:** A document is a well-ordered set of data elements.

**Indexing process:** The indexing process is characterized by the operations of identification (recognition) and representation of data elements and relations.

**Indexing system:** An indexing system is a system for the application of the indexing process to the document space. The output from the indexing system is the index.

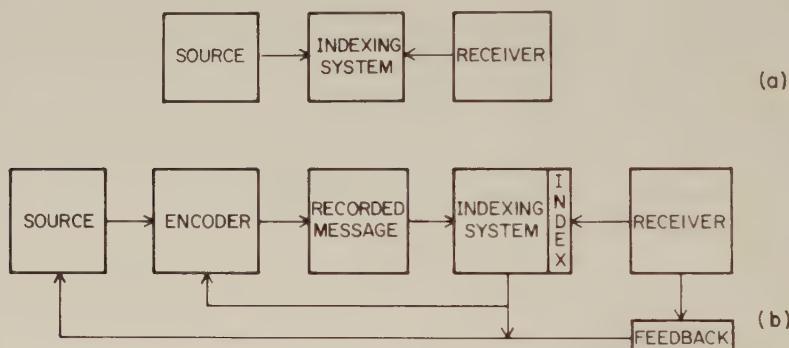
<sup>2</sup> We consider both aggregates of words and chemical formula/structures to be examples of documents [12].

*Document space:* A document space is an ordered set of documents.

*Index:* An index is the image of composite order-preserving mappings performed on the document space.

*Index entry:* An index entry is an expression such that the following data element ( $d$ ) relation (REL) holds:  $d_j \text{ REL } d_k$ , where for each  $j \in \{k\}$  s.t.  $d_j \text{ REL } d_k \forall j, k$ .  $\{k\}$  are the partitions of the document,  $d_j$  is the  $j$ th data element of the document.

As depicted in Fig. 3(a), an indexing system serves as the intermediary between a set of data sources and a set of data users (receivers). The problem becomes one of representing messages that come from a number of unrelated sources. We assume that each message (document) is an ordered collection of data elements and relations between data elements. It should be obvious that message representation must be effected so as to guarantee the maximum degree of overlap between the experience set that is the representation and the experience sets of the classes of potential receivers. In this way the meaning of the document is preserved.

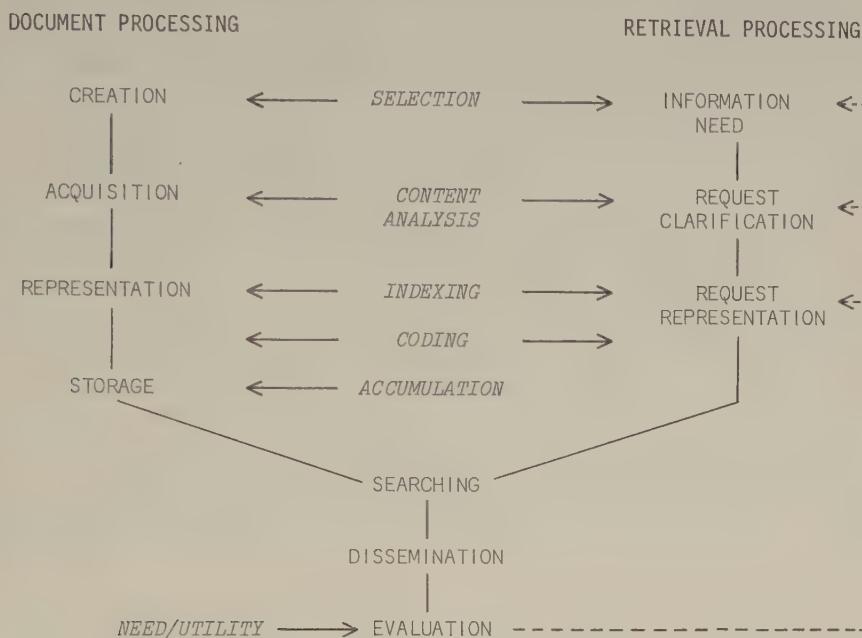


**Fig. 3.** (a) The position of the indexing system in data transfer. (b) The indexing system and feedback operations.

The indexing system provides the transformations and the interface experience set required for effective communication between the source(s) and the receiver. As a result of the operation of the indexing system, a receiver is able to *discover* the existence of a source; this permits the existence of the feedback loop depicted in Fig. 3(b). Feedback between a receiver and a source is often a prerequisite for effective communication. The crucial point is that not only is the index the product, but it is all that remains of the original document space. Accordingly, we assume that the original documents are not directly available to the receiver; hence the index is the receiver's only point of access to the document collection. Under such constraints it should be clear that accurate retrieval depends on the exactness of the operation of the indexing system.

Let us now consider the function of the indexing system from a more abstract level. Information storage and retrieval (IS&R) systems can be described in the terminology

of Marschak [20] as the combination of two *purposive processing chains*. These two chains are depicted in Fig. 4. We have chosen to call them *document processing* and *retrieval processing*. Both processes, which are greatly simplified in this figure, actually involve multilevel and multistep operations designed to expedite the transfer of data.



**Fig. 4.** The index as the interface between IS&R processes of document processing and retrieval processing.

The document processing chain shows the flow from document creation to document acquisition (by the IS&R system), representation, and storage. The first three stages are paralleled by the retrieval processing chain in the conception (realization) of the information need, the clarification of the request, and the representation (coding) of the request. Both chains share, through the representation stage, the operations of selection, content analysis, indexing, and coding. Document storage involves, in addition, the process of accumulation. The two processing chains merge at the searching operation where retrieved data (potentially, information) are disseminated for evaluation. The dotted lines in the figure indicate the possibility of repeated cycling through the retrieval process. The indexing system operates on the document space (and the document representation) produced by the first stages of the document processing chain. Thus the index is the *document* which the retrieval processing chain uses for the search operation. To reiterate, *accurate retrieval depends on the exactness of the operation of the indexing system*.

It should be clear, at this point, that there exists a problem in delineating the precise scope of a general computer-based indexing system. The actual itemization of the

components/operations that fall within the box labeled "indexing system" depends on the specific implementation, i.e., the number of manual inputs and operations, the scope of the operating system being used, the set of utility programs that are available, etc. Accordingly, we have selected the operations of *input*,<sup>3</sup> *identification*, *representation*, *entry creation*, and *output* as being germane to a description of a particular computer-based indexing system.

*Input* relates to those operations required to obtain data in computer-manipulatable form. Thus input includes keyboarding, optical scanning, magnetic-character reading, and the whole range of analog signals (e.g., voice) which may be operated on by a computer program.

Once data have been obtained in machine-readable form, they must be partitioned according to a set of well-defined *identification* procedures. Among these are identification of document boundaries, titles, bibliographic data, and various other elements which are of interest within a particular indexing system.

The process of identification is followed by that of *representation*, a process involving essentially symbol/symbol transformations. For instance, a word may replace a word, a phrase a paragraph, a molecular formula a structure, and a name a picture.

*Entry creation* is the process of constructing an index entry from data element representations. This process involves, in particular, formatting operations such as inversion, articulation, and keyword extraction.

Finally, the set of entries created at the previous step is ordered in some way, redundancies are eliminated, final formatting operations are performed, and the resultant document is made available to a class of users in some way. This process we call *output*.

While these five classes of procedures are not cleanly separated in most indexing systems, we consider them to be the basic processes in indexing and will use them as the basis for subsequent discussion.

## EXAMPLES OF COMPUTER-BASED INDEXING

### An Overview

Figure 5 presents a matrix of the possible levels of automation associated with the five classes of computer-based indexing procedures mentioned above. It should be recognized that although an indexing system may be described as computer based, in actuality only a minority of its procedures may be performed automatically, while the remaining procedures are accomplished either manually or by means of some semi-automatic process (i.e., with limited manual intervention). As an example of this range of degrees of automation, the procedure of document input (depending on whether the

<sup>3</sup> We believe that a distinction should be made between input as a *product* (e.g., a magnetic tape resulting from a previous document processing operation) and input as a *source* (e.g., the text of this article).

document is viewed as a source or as a product) could involve either manual key-punching, optical character recognition (with some manual editing), or direct processing from a previously prepared magnetic tape.

PROCEDURES	OPERATION		
	MANUAL	SEMI-AUTOMATIC	AUTOMATIC
INPUT			
IDENTIFICATION			
REPRESENTATION			
ENTRY CREATION			
OUTPUT			

**Fig. 5.** Matrix of the level of automation associated with the computer-based indexing procedures.

As has been implied in the section entitled The Nature and Definition of Indexing, the processes of data element identification and document representation are the crucial steps for successful computer-based indexing and subsequent document retrieval. Consider the following title (a document resulting from *identification*):

Effect of a selective beta-adrenergic blocker in preventing falls in arterial oxygen tension following isoprenaline in asthmatic subjects.

If, for the sake of exposition, we postulate the existence of a computer-based indexing system that incorporates role indicators (R), generic-specific relations, formula lists, and a word guide or controlled vocabulary as *analysis documents*, then the following system *representation* might be obtained:

$R_{19}$  of a selective beta-adrenergic receptor (beta receptor) blocking drug (drug) in  $R_6 R_3$  in arterial (cardiovascular system) oxygenation tension (airway resistance)  $R_{30}$  isoprenaline ( $C_{14}H_{22}N_2O_3$ ) in asthmatic subjects.

It is conceivable that this indexing system would automatically generate (among others) the following entries from the system representation of the document (*entry creation*):

beta-blocking drug  
beta receptor  
airway resistance  
 $C_{14}H_{22}N_2O_3$   
beta-adrenergic blocker,  
effect of, in preventing falls following isoprenaline in asthmatic subjects

Virtually all computer-based indexing systems automate the index-output process. We wish to stress that although output procedures are sometimes very complex, there

is a considerable difference in computational sophistication between algorithms that are designed to format index entries and algorithms that are designed to create the system's representation of the document space. However, one should not minimize the importance of the format of the "index display" as the vehicle for a receiver's rapid assimilation and comprehension of index data.

### Specific Examples of Computer-Based Indexing Systems

With the foregoing as background, we turn to a discussion of specific examples of computer-based indexing systems. Table 2 presents a summary of some general, specific, and experimental computer-based indexing systems. The column labeled "type" is derived from the partition depicted in Table 1 and the subsequent columns are a compression of the matrix of Fig. 5. These representative indexes are described and characterized in the sections entitled General Indexes, Specialized Indexes, and Experimental Indexes.

**TABLE 2**

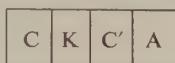
Examples of Computer-Based Indexes<sup>a</sup>

Name	Ref.	Type	Input	Identifi- cation	Repre- sentation	Entry creation	Output
<b>General indexes</b>							
B.A.S.I.C.	21	KWIC	M	A	A	A	A
Biosystematic index	22	KWOC/REL	M	M	A	A	A
Chemical Industry Notes	23	KWOC	M	S	A	A	A
Project MEDICO	24	KWOC/REL	M	A	S	A	A
Double KWIC	25	KWIC/REL	M	S	A	A	A
Articulated	26	KWOC/REL	M	S	A	A	A
Citation	27	KWOC	M	M	S	A	A
Multiterm	28	KWOC/REL	M	A	A	A	A
<b>Specialized indexes</b>							
HAIC	29	KWIC	A	A	A	A	A
Chemical Substructure index	30	KWOC/REL	M	A	A	A	A
GREMAS (chemical reactions)	31	REL	M	M	M	A	A
KLIC	32	KWIC	A	A	A	A	A
DuPont CS <sup>4</sup>	33	REL	M	A	A	A	A
CAS Patent Concordance	34	KWOC	A	A	A	A	A
<b>Experimental indexes</b>							
SYNTOL	35	KWOC/REL	M	A	A	A	A
	36						
On-line indexing (Doyle)	37	REL	M	S	—	—	—
On-line indexing (Thompson)	38	REL	M	S	A	S	A
Book indexing (Borko)	39	KWOC	M	S	A	S	A

<sup>a</sup>A = automatic, M = manual, REL = relational, and S = semiautomatic.

### *General Indexes*

**KWIC Indexes.** The *Biological Abstracts* subject index called B.A.S.I.C. (Biological Abstracts Subjects in Context) (Fig. 6) exemplifies the KWIC index, the only type of index that is widely produced by automated means. The purpose of the KWIC index is to permit the user to locate documents of potential interest to him by first locating a keyword within an alphabetical listing and then, by checking the contextual settings in which the keyword occurs, to enable him to narrow his choices to those that he judges correspond to his interests. Many variants of the KWIC index have been produced, but all are based upon the notion of the cyclic Latin square, known for centuries [40]. A typical KWIC index entry may be characterized as follows:



where C and C' represent segments of the context in which the keyword, K, occurred in the document (either C or C' might be empty, but not both) and A is a pointer (address) to the location of the keyword (and its context) within the document.

EEN BACTERIA EXPOSED TO	MONOCHROMATIC LIGHT OF DIFFERENT WAV	22030
EELY MOVING EYE GRATING	MONOCHROMATOR /SPECTRAL SENSITIVITY	46414
S-SP ISOCHRYYSIS-GALBANA	MONOCHRYYSIS-LUTHERI DIURON NEBURON M	22258
A CLADOSPORIUM FUSARIUM	MONOCILLIUM CONIOTHYRIUM-OLIVACEUM S	16322
F CONDUCTING BUNDLES OF	MONOCOT AND DICOT PLANT LEAVES SORGH	10630
IA AMSINCKIA-HISPIDIA-D	MONOCOT DICOT LIST/ ADDITIONS TO THE	58993
ND IN 24 PARGANAS INDIA	MONOCOTS DICOTS/ A BOTANICAL EXPLORA	23827
RAMS BINOMIALS PRIORITY	MONOCOTS DICOTS/ A RECONSIDERATION O	17950
CT OF MURSHIDABAD INDIA	MONOCOTS DICOTS/ A SKETCH FLORA OF C	23825
SIDES OF LEAVES IN SOME	MONOCOTYLEDONOUS AND DICOTYLEDONOUS	16829
ICHOTOMOUS BRANCHING OF	MONOCOTYLEDONOUS TREES ASCLEPIAS-D H	44067
MALAD-MADH AREA PART 1	MONOCOTYLEDONS HABITAT FLOWERING FRU	58986
TAXONOMIC NOTES ON SOME	MONOCOTYLEDONS OF ALASKA AND NORTHER	35699
AL NOTES ON NEW OR RARE	MONOCOTYLEDONS OF THE FRENCH ANTILLE	35701
FRENCH ANTILLES MARINE	MONOCOTYLEDONS 39TH CONTRIBUTION HAL	35701

Fig. 6. Illustration of the B.A.S.I.C. index (author's reproduction).

In the example of Fig. 6, the keyword occurs near the middle of the entry and the address is an abstract number. Shading is introduced to facilitate visual location of the keyword.

Input of the data to be processed by the KWIC indexing procedure is typically manual (automatic if a prior processing step yields the appropriate input as a by-product), but the remaining processes of identification, representation, entry creation, and output are usually automatic.

**KWOC Indexes.** There are a multitude of examples of the KWOC index of which that to *Chemical Industry Notes*, illustrated in Fig. 7, is typical. A simple noun phrase,

consisting of a noun modified (optionally) by one or more adjectives, serves as the keyword. A KWOC index entry may be characterized as



where K is a keyword and A is a set of addresses.

ANCHOR CHEMICAL CO. ltd.	K216, F235,	E462
ANGUS	D308	
ANTHRAQUINONE	A259	
ANTHRAQUINONE-BASED DYES	A259	
ANTI-FOAMS, SILICONE	E530	
ANTIOXIDANTS	D270, F284	
ANTIOZONANT AFD	E531	
ANTIOZONANTS	D270	
APPLIED SYNTHETICS CORP.	E436	
AQUITAINE CHEMICALS INC.	D292, K203	
AQUITAINE-ORGANICO	D291, D292, K203	

Fig. 7. Illustration of the Chemical Industry Notes index (author's reproduction).

In the example of Fig. 7, input is manual, identification is partly manual (flags introduced into the input), and the remaining processes are automatic. The addresses are abstract numbers.

Textbooks are often indexed by the KWOC technique, although in the vast majority of cases the entire index is produced manually. Hybrid indexes combining some KWOC entries and some KWOC/relational entries are also commonly used with textbooks. Again, the indexes are manually produced.

CORNELIUS RJ-----*	58★P AUSTR I MIN METALL	65	185
BJORLING G	J CHEM UAR	66	9 187
CORNELIUS WO-----*	33★Z FISCH-----	31	535
PANDIN TJ	MARINE BIOL	67	1 60
CORNELIUSSEN R-----*	*IN PRESS-----		
PETERLIN A	J POLY SC A2	67	5 957
-----	-PRIVATE COMMUNICATIO=		
MEINEL G	J POL SCI B L	67	5 613
CORNELL CM-----*	50★SURGERY-----	28	735
SALVINI E	MIN RAD FIS	67	12 70
CORNELL D-----*	60★CHEM ENGNG PROG-----	56	68
REISS LP	IND ENG PDD	67	6 486
SEMMELBA.R	CHEM ENG SC	67	22 1237

Fig. 8. Illustration of the Science Citation index (author's reproduction).

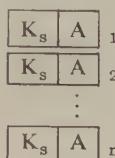
A rather unique example of the KWOC index is afforded by the Science Citation Index (SCI) [27, 41], illustrated in Fig. 8. An entry in this index is characterized as



where  $K_p$  is a primary keyword of the form



where N is an author name and S is a specifier amounting to a bibliographic citation, and  $I_s$  is a set of secondary index entries of the form



where  $K_s$  is a secondary keyword (an author name), and A is a bibliographic citation amplified by a code indicative of document type.

The significance of an SCI entry is that the document represented by  $K_p$  (and authored by element N of  $K_p$ ) has been cited by one or more of  $I_{s_i}$  ( $1 \leq i \leq n$ ,  $n =$  total number of citings of  $K_p$ ). The interpretation is that since one knows what the document represented by  $K_p$  is *about*, one can reasonably assume that the document(s) represented by the  $I_{s_i}$  are also at least in part *about* the same thing. The SCI is the only index which permits a user to trace data both forward and backward in time.

**KWOC/Relational Indexes.** In this category we include those indexes whose entries are KWOC in nature but which are modified by a relational element of some kind. Typical relational elements are those called *role indicators* [42]. For instance, a keyword that has a *set* of referents may be modified by some means so that the combination has a single referent. The keyword ASH might be modified as follows.

ASH (the plant)  
 ASH (wood)  
 ASH (residue of combustion)  
 ASH (process)

To be sure, the increase in specificity is not perfect (e.g., many kinds of trees called "ash" exist). Nevertheless, the purpose of the KWOC/REL index is to provide a degree of specificity that the keyword alone does not provide.

In the example of Fig. 9 [43], input is manual, but the processes of identification, representation, entry creation, and output are automatic. Examination of Fig. 9 reveals a series of partial index entries (no address has been appended to the keyword/relation string), the first element of which represents a keyword and successive elements of which represent relations. For instance, 57432 is a keyword (a number standing for a specific chemical compound). This keyword is found (by computer) in a dictionary in which certain other elements are contained that (1) bear specific relationships to the keyword or (2) specify certain relationships into which the keyword may enter (determined by the context of the keyword in the document). A normalized frequency is also computed for each keyword, and its value is appended to the keyword as a

measure of the keyword's importance in the document. Thus 57432 has "importance" 2, is called AMOBARBITAL (synonym), 5-ETHYL-5-ISOAMYLBARBITURIC ACID (synonym), is a member of the class BARBITURATES, and has the trade name AMYTAL. In addition, AMYTAL is seen to be modified by the term THERAPY and the combination AMYTAL/THERAPY has weight 1. The entire set of index entries, together with the bibliographic data, constitute something of a telegraphic abstract [44].

BUCHANAN DS  
AN APPROACH TO MANAGEMENT OF STATUS EPILEPTICUS.  
SOUTHWEST MED 47,187-9, JUL 66

NO. OF WORDS = 1760  
(2) BARBITURATES

57432 (2) AMOBARBITAL, 5-ETHYL-5-ISOAMYLBARBITURIC ACID, BARBITURATES,  
AMYTAL

50066 (3) 5-ETHYL-5-PHENYLBARBITURIC ACID, PHENOBARBITAL, BARBITURATES  
(2) ANTICONVULSANTS

57410 (3) DIPHENYLHYDANTOIN, 5,5-DIPHENYL-2,4-IMIDAZOLIDINEDIONE,  
HYDANTOINS

AMYTAL/ THERAPY (1)  
AMOBARBITAL/ THERAPY (1)  
PHENOBARBITAL/ ADMINISTRATION (1), EFFECT (1), THERAPY (2)  
ANTICONVULSANTS/ EPILEPSY (1), THERAPY (1)  
DIPHENYLHYDANTOIN/ DOSAGE (2)  
BARBITURATES/ ACTIVITY (1)

**Fig. 9.** Illustration of a KWOC/REL index (after Ref. 43).

A second example of the KWOC/REL type of index is afforded by the articulated index. Classic examples of manually produced articulated indexes include the *Subject Index to Chemical Abstracts* and the *Subject Index to Nuclear Science Abstracts*. Lynch has studied articulated indexes with the aim of automating their production [45]. Full automation has not been achieved, but an articulated index has been produced by Lynch *et al.* by semiautomatic means (see Fig. 10) [46]. The principal problems to be resolved before articulated indexes may be produced entirely automatically are (1) the construction of normal-form strings and (2) the selection of keywords.

A normal-form string consists of noun phrases separated by articulation points (usually function words, such as prepositions and conjunctions), viz.,



where —— represents a noun phrase and  $\square$  represents an articulation point. Each noun phrase is a candidate keyword (heading, in Lynch's terminology). Articulated index entries may be characterized by an algorithm of the following kind [47] (assume  $n$  noun phrases in the normal-form string):

Assume a normal-form string

$$K_1 \square K_2 \square K_3 \dots \square K_n$$

Generate from this string  $n$  canonical notations, each of the form

$$K_i —— K_1 \square K_2 \dots \square K_{i-1} \square, \square K_{i+1} \dots \square K_n \quad (1 \leq i \leq n)$$

where  $K_i$  is the  $i$ th noun phrase as subject heading (keyword) and the string following the dash is a *canonical modifier*. Call the portion of the canonical modifier to the left of the comma the *initial part* and that portion to the right of the comma the *final part*. Apply the following procedure to each canonical notation, in turn, to obtain all articulated index entries for a normal-form string of  $n$  components ( $n - 1$  articulation points).

1. If there is no initial part of the canonical modifier, the index entry is complete.
2. For each incomplete index entry, form complete entries as follows.
  - a. Beginning with the last component of the initial part, generate  $i$  subheadings (secondary keywords) and canonical submodifiers by extracting the last, the last two, . . . , the last  $i - 1$ , and last  $i$  components.
  - b. If the initial part exists, extract the first, and only the first, component of the final part as a subheading.
3. Continue application of 1 and 2 until all entries are complete.

Various alternatives may be introduced, but the above algorithm suffices for illustration.

#### ABSTRACTS

OF CURRENT PUBLICATIONS, SMRE(2)  
EXCHANGE OF PUBLICATIONS + BIBLIOGRAPHIES +,  
ON MINE SAFETY RESEARCH, GRICE C.S.W.(19)

#### ACCIDENTS

ANALYSIS OF DATA ON, MAGUIRE B.A.(2)  
IN HAULAGE AND TRANSPORT FOR 1960, MAGUIRE B.A.(7)  
CALCULUS TO STUDY OF, IN MINES, APPLICATION  
OF PROBABILITY, WYNN A.H.A.(3)  
CONTROL OF, CHARTS + SUMMARIES FOR, DAWES J.G.(17)  
CONTROL CHARTS, WIDGINTON D.W.(1)  
DUE TO HAULAGE, CAUSES OF, NORTH OF ENGLAND INSTITUTE  
OF MINING AND MECHANICAL ENGINEERS(7)

**Fig. 10.** Illustration of Safety in Mines index (author's reproduction).

The segments of an articulated index entry may be thought of as answers to questions such as WHERE, WHY, WHAT, HOW, WHEN, and WHOSE, with respect to one of the noun phrases as keyword. For instance, the articulated index entry

Sulfur isotopes,  
in pyrite from Black Forest, 00000

contains two segments both of which answer WHERE with respect to "sulfur isotopes." But in the entry

Pyrite,  
sulfur isotopes in, from Black Forest, 00000

the segments answer the questions WHAT and WHERE with respect to "pyrite." When the KWOC/REL index is viewed in this way, it can be seen that the *Predicasts Abstracts* of Fig. 11 exhibit the properties of a KWOC/REL index just as does the articulated index, but in a more highly organized form.

A final example of the KWOC/REL index is illustrated in Fig. 12. This index, the multiterm index developed by Skolnik [28], consists of index entries of the form



where M is a multiterm of the form B/C/D/... and A is an address. The sequence of elements B, C, D, ... in the multiterm is established as *generic to specific* or, where such a succession of relations is not possible, by *chronology* or by some prescribed order, as for chemical reactions, where the order is

product/reactant(s)/process/conditions/catalyst, solvent, etc./equipment/use of product/  
property of product

Each element of the multiterm may also have one or more role indicators appended to it.

**KWIC/Relational Indexes.** The Double-KWIC (D-KWIC) index provides a level of relational capability not easily realized through use of the standard KWIC index [25]. The relational aspects of the D-KWIC index are purely syntactic. From the illustration of Fig. 13, it can be seen that an index entry such as

File management  
evaluation of \* software packages . . . the 215

provides a useful syntactic relationship between "file management" and "evaluation" from which one may infer the existence of some conceptual relationship.

The D-KWIC index is produced by procedures that, except for input which is usually manual, are fully automatic. The input strings (titles, etc.) are first KWICed in the usual fashion, then the keyword is isolated and is replaced by an asterisk (to mark the location of the keyword in the original string; see Fig. 13). The remainder of

SIC No.	PRODUCT A	EVENT	PRODUCT B	YEARS			QUANTITIES	UNIT OF MEASURE	JOURNAL	SOURCE DATE	PAGE
				B	S	L					
28 153 205	Phenol	used in	plastic materials	65	80	.6±	1.2±	bil lbs	Chem Week	1/1/23/68	10
28 153 206	Phenol, synthetic	capacity by	producer	69	71	1550. d	2375. d	mill lbs	#OPD Rep	3/ 3/69	29
28 153 206	Phenol, synthetic	consumption	cumene as % of phenol	71	71	1020.	1020.	mil lbs	OPD Rep	4/21/69	30
28 153 206	Phenol, cumene-based	consump of	cumene as % of phenol	66	69	75	48.	%of total	O&G Jour	3/ 3/69	92

Fig. 11. Illustration of the Predicasts abstracts (author's reproduction).

the string is again KWICed, but only selected strings are subordinated to the keyword previously isolated. In this way, a better organization of "secondary keywords" is obtained than is possible using the standard KWIC procedure.

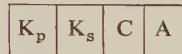
```
REACTANT -R / PROCESS / CATALYST / USE / PROPERTY / PRODUCT -PQ /
PROCESS / CATALYST / USE / PROPERTY / PRODUCT -PQ / REACTANT -R /
CATALYST / USE / PROPERTY / PRODUCT -PQ / REACTANT -R / PROCESS /
USE / PROPERTY / PRODUCT -PQ / REACTANT -R / PROCESS / CATALYST /
PROPERTY / PRODUCT -PQ / REACTANT -R / PROCESS / CATALYST / USE /
```

**Fig. 12.** Illustration of the Multiterm index (author's reproduction).

FILE ACTIVITY			
INFLUENCE OF *, FILE SIZE, AND PROBABILITY OF SUCCESSFUL			
RETRIEVAL ON EFFICIENCY OF FILE STRUCTURES .....	175		
FILE MANAGEMENT			
EVALUATION OF * SOFTWARE PACKAGES .....	THE 215		
PACKAGES .....	THE EVALUATION OF * SOFTWARE 215		
SESSION SUMMARY .....	* SYSTEMS: 525		
SOFTWARE PACKAGES .....	THE EVALUATION OF * 215		
SUMMARY .....	* SYSTEMS: SESSION 525		
SYSTEMS: SESSION SUMMARY .....	* 525		
FILE ORGANIZATION			
CONTROLLED * .....	USER 245		
CONTROLLED * AND SEARCH STRATEGIES .....	USER 183		
SEARCH STRATEGIES .....	USER CONTROLLED * AND 183		
STRATEGIES .....	USER CONTROLLED * AND SEARCH 183		
USER CONTROLLED *	..... 245		
USER CONTROLLED * AND SEARCH STRATEGIES .....	183		
FILE PROCESSOR			
THE HUMAN-READABLE/MACHINE-READABLE (HRMR) MASS MEMORY AS THE *			
FOR A NETWORK OF DOCUMENTATION/ANALYSIS CENTERS .....	339		

**Fig. 13.** Illustration of the Double-KWIC index (author's reproduction).

A D-KWIC index entry may be characterized as



where  $K_p$  and  $K_s$  are primary and secondary keywords, respectively, C is context, and A is an address. Control over the specific values of  $K_p$  and  $K_s$  may be obtained by use of word control lists (stop lists or thesauri), frequency criteria, and procedures for the standardization of inflectional suffixes [25, 47, 48].

### *Specialized Indexes*

The distinction we make between general indexes (discussed in the previous section entitled General Indexes) and specialized indexes is based upon the nature and scope of the subject matter indexed rather than on index type. Thus we consider a subject index for chemistry or physics to be a general index, but a molecular formula index

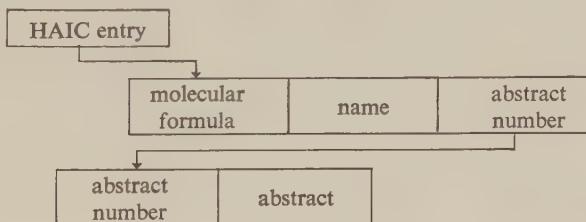
or a star catalog to be specialized indexes. There are many specialized indexes, a few of which are discussed below. The examples have been chosen to illustrate a range of specialties. The selections have been arbitrary and they are by no means exhaustive.

**KWIC Indexes.** The *Hetero Atom in Context (HAIC)* index, illustrated in Fig. 14, is a good example of a specialized application of the KWIC indexing technique to a set of data whose elements have a well-defined structure, namely molecular formulas.

Mn <sub>2</sub>	Ni	O <sub>4</sub>
(C <sub>2</sub> H <sub>8</sub> N <sub>4</sub>	Ni	O <sub>4</sub> )x
(C <sub>3</sub> H <sub>10</sub> N <sub>4</sub>	Ni	O <sub>4</sub> )x
(C <sub>4</sub> H <sub>12</sub> N <sub>4</sub>	Ni	O <sub>4</sub> )x
(C <sub>6</sub> H <sub>2</sub> N <sub>2</sub>	Ni	O <sub>4</sub> )x
(C <sub>8</sub> H <sub>12</sub> N <sub>4</sub>	Ni	O <sub>4</sub> )x
(C <sub>10</sub> H <sub>16</sub> N <sub>4</sub>	Ni	O <sub>4</sub> )x
(C <sub>12</sub> H <sub>8</sub> N <sub>2</sub>	Ni	O <sub>4</sub> )x
(C <sub>14</sub> H <sub>24</sub> N <sub>4</sub>	Ni	O <sub>4</sub> )x
(C <sub>18</sub> H <sub>16</sub> N <sub>4</sub>	Ni	O <sub>4</sub> )x
(C <sub>18</sub> H <sub>20</sub> As <sub>2</sub>	Ni	O <sub>4</sub> )x
(C <sub>22</sub> H <sub>18</sub> N <sub>4</sub>	Ni	O <sub>4</sub> )x
(C <sub>22</sub> H <sub>24</sub> N <sub>4</sub>	Ni	O <sub>4</sub> )x
((C <sub>24</sub> H <sub>22</sub> N <sub>4</sub>	Ni	O <sub>4</sub> )x
(C <sub>38</sub> H <sub>28</sub> As <sub>2</sub>	Ni	O <sub>4</sub> )x
(C <sub>42</sub> H <sub>36</sub> As <sub>2</sub>	Ni	O <sub>4</sub> )x
C <sub>12</sub> H <sub>8</sub>	Ni	O <sub>4</sub> •C <sub>2</sub> H <sub>8</sub> N <sub>2</sub> •2H
C <sub>26</sub> H <sub>20</sub> N <sub>10</sub>	Ni	O <sub>4</sub> •2C <sub>5</sub> H <sub>5</sub> N
C <sub>12</sub> H <sub>28</sub> N <sub>8</sub>	Ni	O <sub>4</sub> •2Cl <sub>1</sub>
C <sub>27</sub> H <sub>27</sub> N <sub>6</sub>	Ni	O <sub>4</sub> •C <sub>10</sub> <sub>4</sub>
C <sub>29</sub> H <sub>31</sub> N <sub>6</sub>	Ni	O <sub>4</sub> •C <sub>10</sub> <sub>4</sub>

Fig. 14. Illustration of the HAIC index (author's reproduction).

In this index the truncation or "wrap-around" of the input string common in general KWIC indexes is avoided because the index entry is made to be self-addressing. That is, it serves as a pointer into the *Formula Index to Chemical Abstracts*. The relationship between a HAIC index entry and a CA abstract is illustrated below.



The HAIC index also differs from general KWIC indexes in that several levels of sorting are applied to each set of entries having a particular keyword. For instance, it can be seen from Fig. 14 that within the group of formulas with **Ni** as the keyword, the formulas are secondarily ordered by the portion of the context to the left of the keyword. This makes it possible to locate not only formulas having a particular element (e.g., **Ni**) but those having, in addition, say, the grouping **O<sub>4</sub>P<sub>2</sub>S<sub>4</sub>**.

Input for the HAIC is obtained in machine-readable form as the by-product of other operations, hence all processes necessary to its production are automated. (The HAIC index was discontinued in 1971.)

ST	ACHYBOTRYS	*
T	ACHYCARDIAS	*
TR	ACHYLOBANE	*
TR	ACHYPHYLAXIS	*
	ACHYRANTHES	*
TR	ACHYSAURUS	*
BR	ACHYURAN	*
AST	ACI	*
DIPS	ACI	*
AC	ACIA	*
ARB	ACIA	*
PIST	ACIA	*
ENCEPHALOMAL	ACIA	*
GL	ACIAL	*
SP	ACIAL	*
INTERF	ACIAL	**
EM	ACIATION	*
GL	ACIATION	*
SEB	ACIC	*
THOR	ACIC	*
	ACID	*****
ANT	ACID	*
	ACIDAZOL	*
	ACIDEMIA	*
LACT	ACIDEMIA	*
	ACIDEMIAS	*
	ACIDES	*
	ACIDIC	***
	ACIDIFIED	*
	ACIDIMETRIC	*
HAD	ACIDIN	*
	ACIDITIES	**
	ACIDITY	**
	ACIDO	**

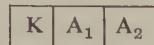
Fig. 15. Illustration of the KLIC index (author's reproduction).

A second special application of the KWIC indexing technique is exemplified by the Key Letter In Context (KLIC) index [32] illustrated in Fig. 15. Here words are processed to provide the user a means of determining those words that contain a particular sequence of letters in common. In the illustration of Fig. 15, it is easy to see that 14 words contain the letter sequence ACID. The asterisks to the right in each entry provide frequency-of-occurrence data for the word indexed (\* = low frequency,

\*\*\*\*\* = high frequency). The KLIC index serves simultaneously as the index and as the set of data indexed. Its use in automated retrieval systems has been discussed elsewhere [32, 49].

The production of the KLIC index, like the HAIC, is entirely automatic, although in both instances input could be manual.

**KWOC Indexes.** The *Patent Concordance* [34], prepared by the Chemical Abstracts Service, is illustrative of a specialized KWOC index (see Fig. 16). The index entries take the form



where K is a keyword consisting of the *name* of the country in which the patent was issued and the *number* assigned by that country to the patent (e.g., Israeli 25131). In other words, K is a noun phrase consisting precisely of ADJECTIVE-NOUN. A<sub>1</sub> and A<sub>2</sub> are two addresses, the first of which points to the corresponding patent issued in another country and the second points to a CA abstract (the second may be empty). If A<sub>2</sub> is empty, the entry acts like a HAIC entry in that A<sub>1</sub> addresses another location

PATENT NUMBER	CORRESPONDING PATENT	CA REF. NUMBER
156474	BRIT 1177240	72, 12416Q
	GER 1643961	
156630	GER 1902078	
156810	BRIT 1212613	72, 79085Z
	FR 1603226	
	US 3542779	
156951	GER 1814293	
157057	GER 1920222	
157087	GER 1918042	
157288	BRIT 1243646	73, 18869G
	US 3607104	
157467	GER 1900711	
157475	GER 1931090	
157593	GER 1810836	

### Israeli

21179	NETH	6504911
24062	BRIT	1108848
24642	BRIT	1073039
24981	DAN	112814
25131	BRIT	1120152
25260	NETH	6608865
26022	S AFR	67 03320
26256	FR	AD92917

Fig. 16. Illustration of the Patent Concordance (author's reproduction).

in the *Patent Concordance*. This index enables the user to locate all patents issued that correspond to a given patent.

**KWOC/Relational Indexes.** The *Biosystematic Index to Biological Abstracts* [21] is both a KWOC index and a relational index because the keyword is accompanied by a series of elements related to it either taxonomically or subjectively. As illustrated in Fig. 17, a *Biosystematic Index* entry takes the form

K	TC <sub>1</sub>	TC <sub>2</sub>	...	TC <sub>n</sub>	SH	A
---	-----------------	-----------------	-----	-----------------	----	---

where K is a keyword, a major taxonomic category name such as Spermatophyta; TC<sub>i</sub> ( $i \geq 0$ ) are the names of the taxonomic categories hierarchically subordinate to the keyword (e.g., angiospermae → monocotyledonae → agavaceae); SH is a set of subject headings (as abbreviations) associated with the abstract containing mention of the organism, and A is the address (abstract number). Referring to Fig. 17, one finds, for example, the entry

SPERMATOPHYTA  
ANGIOSPERMAE  
MONOCOT      ARACEAE      PL MORPH      10582

which indicates that Abstract 10582 has been associated with the subject heading PLANT MORPHOLOGY and an organism of the arum family,<sup>4</sup> the implication being that if one is interested in the morphology of *Araceae*, one should examine Abstract 10582.

The input and the identification steps in the production of the *Biosystematic Index* are manual. The remaining procedures are effected automatically.

A second example of a specialized KWOC/REL index is afforded by the Chemical Substructure Index (CSI) produced by the Institute for Scientific Information [30] and illustrated in Fig. 18. The data element processed to produce a CSI entry is a Wiswesser linear notation [50] for a chemical entity. Given a canonical form of such a notation, certain structural features, expressed explicitly within the notation, are selected to serve as keywords, the remainder of the notation being appended to the keyword to specify the relation of the substructure represented by the keyword to the chemical structure represented by the entire notation. An additional KWOC-like feature of the CSI entry is the portion referred to as QUIKSCAN, an alphabetical sequence of the Wiswesser symbols that cause CSI entries to be produced. The form of the CSI entry is

K	R	Q	A
---	---	---	---

where K is the keyword, R is the relational framework in which the keyword is found, Q is the QUIKSCAN feature, and A is the address (abstract number plus compound number within the abstract). The CSI makes possible a degree of manual generic

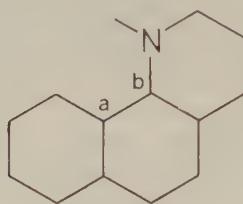
<sup>4</sup> A familiar example of which is the jack-in-the-pulpit (Indian turnip).

**Fig. 17.** Illustration of the Biosystematic index (author's reproduction).

searching that an alphabetically ordered list of Wiswesser linear notations cannot possibly afford. Of course, since the notation in its original form occurs as a CSI entry, specific searches may also be carried out.

WLN

QUIKSCAN ABSTR CPD



## TB666 CN

....MV&T&J	MNTV	175602-	1
....NV BUTT&J D	NTUV	"	25
....NV BUTT&J DIR	NRTUV	"	27
....NV&T&J D	NTV	"	2
....NV&T&J DIR	NRTV	"	5
....NV&T&J DIVR	NRTV	"	7
....NV&T&J D2	NTV	"	3
....NV&T&J D2- AT5NTJ	NTV	"	11
....NV&T&J D2- AT6N DNTJ D	NTV	"	15
....NV&T&J D2- AT6N DOTJ	NOTV	"	9
....NV&T&J D2- AT6NTJ	NTV	"	13
....NV&T&J D2G	GNTV	"	4
....NV&T&J D2NI&I	NTV	"	14
....NV&T&J D2R	NRTV	"	6
....NV&T&J D2UU1	NTUV	"	8
....NV&T&J D2Y	NTVY	"	12
....NV&T&J D3NI&I	NTV	"	10

## TB666\_-

....COVJ FQ	OQTV	175494-	33
....COVJ FQ E1- AT6NTJ	NOQTV	"	28
....COVJ FQ E1M- BT5N CSJ ER	MNOQRSTV	"	16
....COVJ FQ E1M- BT5N CSJ ER D	MNOQRSTV	"	17

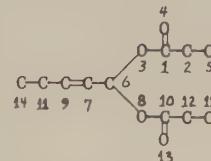
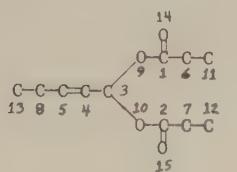
**Fig. 18.** Illustration of the Chemical Substructure index (author's reproduction).

**Relational Indexes.** In this category we include those indexes whose organization is based upon the relational aspects of the data indexed. Most of the examples in this category are unsuited to direct use by people. Rather they serve as the basis for automatic retrieval based upon a person's interests. Two examples from the many available are discussed.

The DuPont Company's Chemical Structural Storage and Search System ( $CS^4$ ) [33] consists in part of a relational index whose entries have the form



where R is a relational network and A an address.<sup>5</sup> The relational network is a chemical structural formula represented as a canonical-form string (Fig. 19), the production of which has been fully described elsewhere [18, 44]. This relational index, which serves for the location of those structures which possess some element(s) in a specified relational framework, is clearly unsuited to direct manual use. Hoffman has described its use within the  $CS^4$  [33].



CANONICAL FORM

Atom	Code	Bond 1		Bond 2		Bond 3	
		Type	No.	Type	No.	Type	No.
1	C	2	14	1	6	1	9
2	C	2	15	1	7	1	10
3	C	1	4	1	9	1	10
4	C	2	5	1	3		
5	C	2	4	1	8		
6	C	1	1	1	11		
7	C	1	2	1	12		
8	C	1	5	1	13		
9	O	1	1	1	3		
10	O	1	2	1	3		
11	C	1	6				
12	C	1	7				
13	C	1	8				
14	O	2	1				
15	O	2	2				

COMPACT LIST

Atom	Code	Attached Atom	Bond Type*
1	C	2	-
2	C	5	1
3	O	6	1
4	O	-	2
5	C	-	1
6	C	7	1
7	C	9	1
8	O	10	1
9	C	11	2
10	C	12	1
11	C	14	1
12	C	15	1
13	O	-	2
14	C	-	1
15	C	-	1

\* Of the bond joining the atom with this row number to the lowest numbered atom attached thereto.

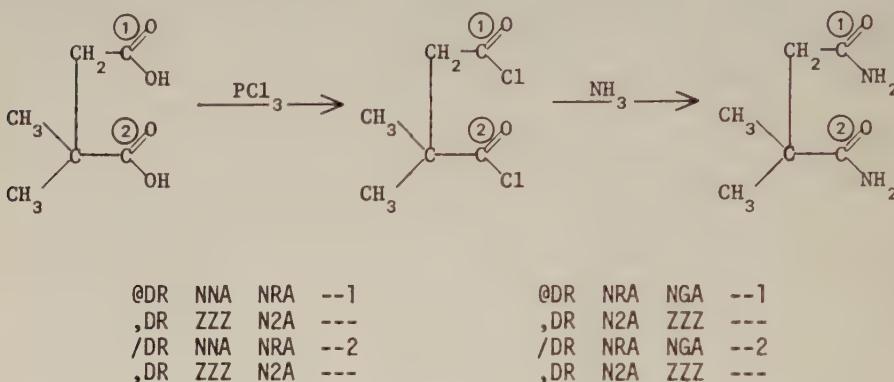
Fig. 19. Illustration of the  $CS^4$  index (author's reproduction).

Input to the  $CS^4$ 's relational index is effected manually. All other processes are automatic.

The second relational index that we shall describe is one concerned with chemical

<sup>5</sup> In the literature this component of the entry has been called a "registry number" [51] or an "identification number" [52]. We recognize that the problems inherent in attempts to automate addressing are nontrivial, but we cannot treat them here. See Ref. 52 for more detail.

reactions. The index, which we shall call the Chemical Reactions Index (CRI), was developed as part of a general information storage and retrieval system by Fugmann and his colleagues at Farbwerte Hoechst [31].<sup>6</sup> In this index a sequence of chemical reactions, such as that depicted in Fig. 20, is transformed into an index entry by representing those parts of the chemical species involved in the reaction that undergo change, together with the chronology of change [53]. Structural features are represented in the GREMAS notation. Finally, an index entry is completed by affixing an address to the reaction representation.



This representation may be simplified to:

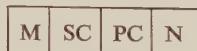
$\text{@DR NNA NRA --1}$ $,\text{DR ZZZ N2A ---}$	$\text{@DR NRA NGA --1}$ $,\text{DR N2A ZZZ ---}$
--	--

**Fig. 20.** Illustration of the GREMAS "Chemical Reactions Index" (author's reproduction).

The form of the CRI entry is



where R is a set of relational strings  $R_1 \dots R_n$  and A is an address. A relational string,  $R_i$ , has the form



where M is a sequence marker (e.g., @DR marks the start of an R), SC is the code for the starting reactive center, PC is the code for this center in the product, and N is the number of the affected center in the entire structure (N may also indicate a class of reactions). If SC or PC would otherwise be empty, the code ZZZ is used. When a set

<sup>6</sup> This index does not exist entirely as an isolate. Rather, it, with other indexes, forms the central data base of the IDC [54].

of relational strings, R, would contain essentially redundant elements, these may be eliminated (see Fig. 20), otherwise, as many strings R, as are necessary to describe all changes that have occurred in R.

This relational index is generally intended for computer-based searching, although its manual use is feasible (especially if transformed into a KWOC/REL or KWIC/REL index).

Input, identification, and representation are manual, although studies leading to the automation of these steps are underway [54]. The remaining processes are automated.

Although each CRI entry pertains to a one-step reaction, multistep reactions can be traced by observing whether two or more CRI entries form a linked list, all elements of which have the same address. For the example of Fig. 20, this idea can be depicted as follows:

@ DR	NNA	NRA	-1	,DR	ZZZ	N2A	-	173
@ DR	NRA	NGA	-1	,DR	N2A	ZZZ	-	173

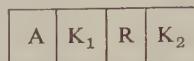
### *Experimental Indexes*

In this section we shall briefly consider some examples of indexes and indexing methods that we characterize as "experimental." It should be obvious that a considerable research effort has also preceded the development of each of the indexes described in the sections entitled General Indexes and Specialized Indexes; however, we are unable, within the scope of this article, to describe all of these important research efforts. We have chosen the following examples of indexing research because they either provide a basis for some present-day computer-based indexes or else they serve as an introduction to some of the representational research described in the section entitled Toward Automatic Indexing.

**Indexing Based on the SYNTOL Model.** The SYNTOL model of language [35] was designed to provide a nonambiguous representation for all natural language propositions. In addition, the representations were designed to be amenable to subsequent computer manipulation. Following DeSaussure, language was viewed as the composition of two reference axes: the syntagmatic and the paradigmatic. Syntagmatic relations described the linear order of the discourse (descriptors and the relations that tie them together) and paradigmatic relations consisted of an *a priori* designation of the underlying thought represented by the discourse (i.e., semantic classes).

The computer-based implementation of the SYNTOL model (see Bely *et al.* [36]), which produces a KWOC/RELATIONAL index, utilizes a lexicon and a dictionary that, together, comprise some 25,000 entries (the paradigms) and a set of routines that perform a syntactic analysis on the input text. Three types of relations are identified: type 1, *coordination* (formal relations, e.g., equality); type 2, *associative* (association

of subject to action); and type 3, *consecutive* (presence of one element that affects another). Figure 21 illustrates an input text, an intermediary analysis, and the final document representation. The output from the intermediary analysis lists the text words, their grammatical classes, their morphological codes, and the extracted keywords and relations. The final representation lists the syntagms that have been identified, the unrelated uniterms, and those syntagms that have been rejected by the processing. The format of the entry is



where A is the document number, K<sub>1</sub> and K<sub>2</sub> are keywords (K<sub>2</sub> may be absent), and R is one of the previously defined relations.

RESUME 58-12-01417			
(			
RECHERCH	NX	FP	EXPERIMENTAL
EXPERIMENTA	AF	FP	
SUR	PU	O	
LA	RC	FS DL FS	
PREVENTION	NX	FS	N32
DES	PO	P	
CRISE	NX	FF	CRISE
PAR	PR	O	
LA	RC	FS DL FS	
PROCAINE	NX	FS	PROCAINE
INTRAVEINEU	AF	FS	INTRAVEINEUX
)			
58-12-01417			
LISTE DES SYNTAGMES			
PROCAINE	/3/	CRISE	
PROCAINE	/3/	EPILEP	
PROCAINE	/2/	PROTECTION	
PROTECTION	/2/	EPILEP	
DIRECT	/2/	CORTEX	
ELECTROCHOC	/2/	CORTEX	
ELECTROCHOC	/2/	STIMULATION	
STIMULATION	/2/	ETRE	
XYLOCAINE	/3/		
LISTE DES ISOLATS			
EXPERIMENTAL			
INTRAVEINEUX			
SUJET			
SYNTAGMES REJETES PAR R.N.			
PROCAINE		INTRAVEINEUX	
ELECTROCHOC		ETRE	
SUJET		PROCAINE	

Fig. 21. Illustration of SYNTOL input text, intermediary analysis, and final document representation (author's reproduction).

**Keyword Structures.** Doyle [37] has demonstrated that a text can be reduced to a "semantic roadmap," a two-dimensional network of terms and relations which it is argued would, ultimately, make information retrieval more effective. Doyle opines:

The association map is a gigantic, automatically derived thesaurus. Confronted by such a map, the searcher has a much better "Association network" than the one existing in his mind, because it corresponds to words actually found in the library and, therefore, words which are best suited to retrieve information from that library [55].

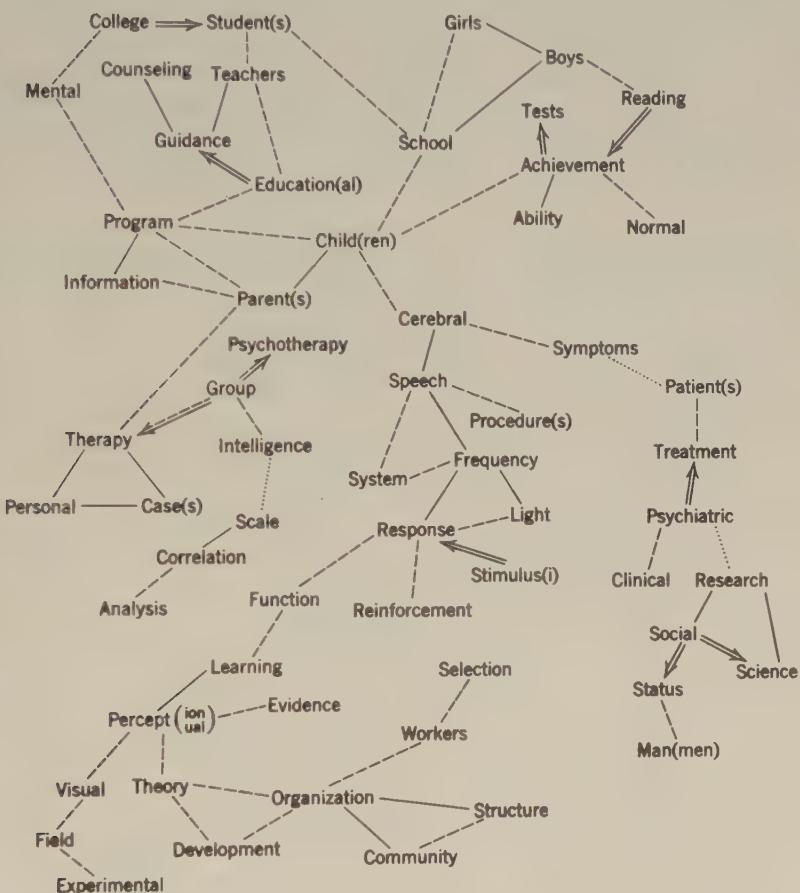
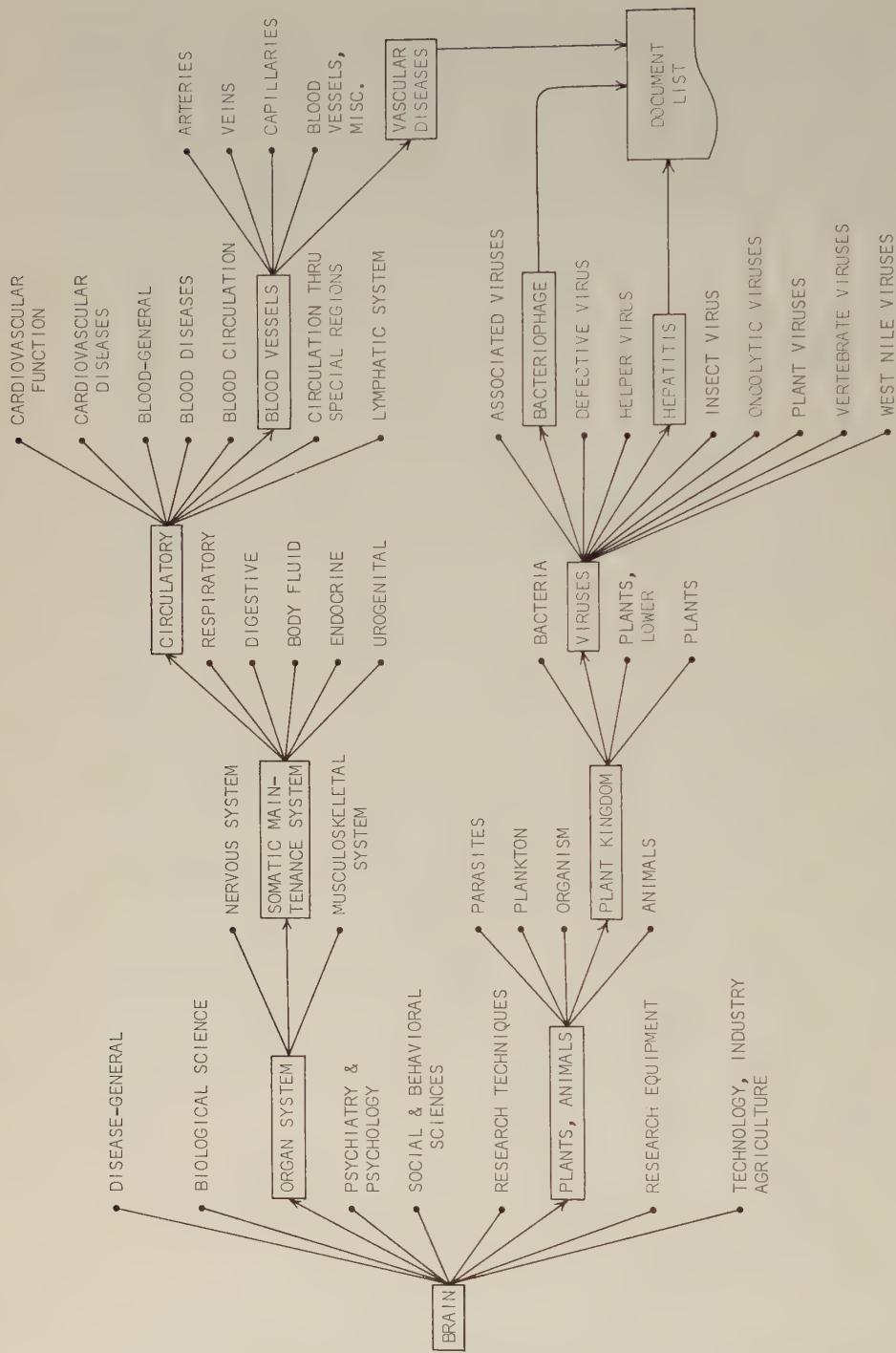


Fig. 22. Illustration of a "semantic roadmap" (reproduced from H. Borko, *Automated Language Processing*, Wiley, New York, 1968, by permission of the publisher).

As depicted in Fig. 22, the "relations" among terms are actually statistical weights which represent *associations* between terms. The associations are represented as

## AUTOMATIC INDEXING



**Fig. 23.** Illustration of Thompson's classification index. Reprinted with permission from the *Journal of the American Society for Information Science*.

normalized frequencies of cooccurrence of pairs of words and are generated utilizing the Pearson correlation coefficient. Operationally, the association map could be implemented as a relational index to the various associations between words in a document. Implicit in Doyle's approach is the assumption that frequency of co-occurrence is a valid relation for an indication of the content of a document.<sup>7</sup>

Thompson [38], at the Stanford Biotechnology Laboratory, has recently developed an on-line retrieval system that utilizes an index term display that is conceptually related to the "roadmap" developed by Doyle. Thompson describes his system as a "classification index" that displays the system's hierarchical classification structure to the searcher. Thus, utilizing an interactive CRT, the user is able to "merely point at alternatives which seem most appropriate to him" while remaining within the constraints of the indexing system. Figure 23 depicts a CRT display where the searcher has narrowed his search term to "hepatitis" and "bacteriophage" and "vascular diseases." The result of this search is a list of matching documents.

Other "on-line indexing" systems include the *Negotiated Search Facility* developed by Bennett [56] and the *Browser* system developed by Williams [57].

**Computer-Based Book Indexing Techniques.** What is possibly the oldest form of index, the back-of-the-book index, still is derived essentially through manual processes. As we have shown, certain specific forms of documents (with some restriction as to content or to length) lend themselves to application of computer-based indexing; however, the automation of the steps of representation and index creation for books remains essentially unsolved. We are forced to admit that, despite the apparent sophistication of computer-based indexing techniques, the field remains in its infancy.

A notable advance toward automatic book indexing has been reported by Borko [39] who has developed the SAINT (Semi-Automatic Index for Natural Language) system for the computer-assisted indexing of books. This system uses the power of the computer to process rapidly the input text, to count, sort, eliminate duplication, and finally suggest lists of two-word terms which are then manually edited by an experienced indexer. The operations performed by this computer-based indexing systems include:

- Reading of text (including page and paragraph numbers)
- Elimination of function words
- Counting occurrences of words
- Sorting words alphabetically and by frequency
- Printing of word lists (manual editing)
- Combining of synonyms and words having the same root form
- Listing of word pairs (manual editing)
- Compilation of cross references
- Printing of the index (manual editing)

<sup>7</sup> For completeness, we wish to point out that there is a considerable body of literature dedicated to statistical and probabilistic methods of index term selection. The early Luhn study [58] that postulated a relationship between word frequency and its "importance" in a document has fostered studies on various forms of statistical association, clump theory, factor analysis, and probabilistic indexing (to mention but a few). The interested reader should consult Stevens [2] for a review of these approaches. Probably the most sensible application of these techniques can be found in Salton's SMART system [59].

Figure 24 depicts a typical final output from the SAINT system. The general form of the entries is as follows:

$K_1 \text{ page}_{i_1},\text{para}_{j_1} \dots \text{ page}_{i_n},\text{para}_{j_n}$

or

$K_1$	See	$K_2$
	See also	

where  $K_1$  and  $K_2$  are words and the synonym and related term references (*See* and *See also*) for a given  $K_1$  point to the document partitions (page<sub>i</sub>, paragraph<sub>j</sub>) occupied by  $K_2$ .

AUTOMATA										
14 5	14	6	15	3	31	1	34	5	37	3
61 2	62	2	62	3	67	1	241	2	287	1
366 1	367	1								
SEE ALSO AUTOMATIC ANALYSIS										
AUTOMATED ANALYSIS										
1 2	6	4	7	4	99	3	120	2	120	4
211 1	248	1	250	1	288	1	316	4		
AUTOMATED BILINGUAL DICTIONARY	SEE ALSO CLASSIFICATION AUTOMATIC									
294 3	294	4	294	5	316	4				
AUTOMATED CLASSIFICATION	SEE ALSO DOCUMENT CLASSIFICATION									
AUTOMATED CLASSIFICATION	SEE CLASSIFICATION AUTOMATED									
AUTOMATED CONTEXT ANALYSIS										
6 4	120	2	211	1	288	1				

Fig. 24. Illustration of the SAINT system index (author's reproduction).

### Summary

In the foregoing discussion we have attempted to provide an overview of the principal types of indexes currently in use. The illustrative examples, although varied, represent no more than a small sampling of those available in every subject area. We have certainly slighted the fields of music, art, literature, law, medicine, and many others. But we believe we have dealt fairly with the general concepts and expect that the reader may be able to extend his knowledge of indexes and indexing beyond that gained from the discussion. A few bibliographical clues are provided following the list of references at the end of this article.

## TOWARD AUTOMATIC INDEXING

Every word that a native speaker of a language uses in his natural language has a definite and useful function. Contrary to some erroneous popular notions, natural languages are highly economical and efficient systems. In most cases no word in a

properly formulated sentence is redundant or dispensable. Thus . . . artificial English like systems which permit the deletion of words in sentences often lose information or create misinterpretations.

Moyne [60]

By definition, the function of a metalanguage in semantic analysis is to singularize or to differentiate—the two notions are interchangeable—the documents of a corpus by the interplay of complex correspondances between natural language formulations and equivalent expressions in a metalanguage.

Gardin [61]

. . . an index node points to each relational statement. Other pointers (arguments of particular predicates) point to the index nodes of the relational statements they represent.

Montgomery [62]

### General Directions

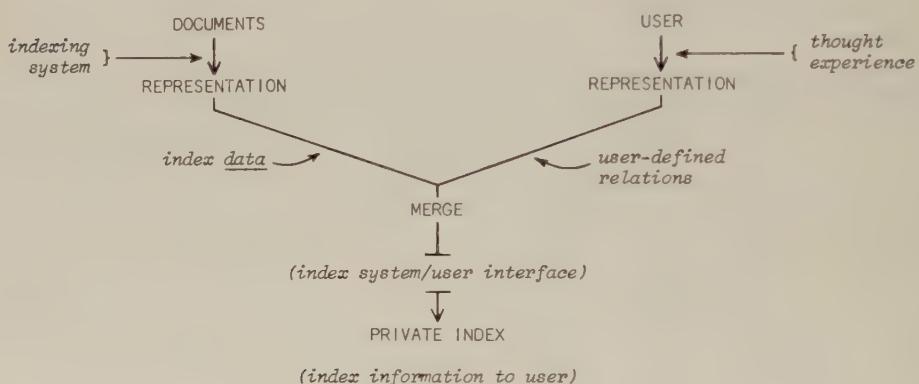
We have repeatedly stressed that the accuracy and the utility of an index are directly related to the completeness and the accuracy of the indexing system's representation of the document space. It is through this representation that a user can be alerted to, and can be placed in contact with, a source's message. Although many of the example indexes presented in the section entitled Examples of Computer-Based Indexing show considerable sophistication and a reasonable level of automation, it can be argued that they fail to provide both a complete and a *flexible* representation of the original document. We have further argued that the direction of current research in computer-based text processing (and indexing) is toward a "structural" representation of the source document. However, a distinction should be made between structures which are *physically derived* from a linear string of words and those structures which are *intellectually imposed* on a linear string. Both interpretations of structure are pertinent to the development of enhanced document representations.

Consider the following example from Lewis Carroll's *Through the Looking Glass*:

*'Twas brillig, and the slithy toves  
Did gyre and gimble in the wabe:  
All mimsy were the borogoves,  
And the mome raths outgrabe.*

Thanks to an abundance of function words, this "nonsense" passage exhibits considerable structural information—i.e., we can derive syntactical, positional, and relational information concerning the unidentified words in this passage without recourse to knowledge of words like "brillig." Thus, with a theory of the logical structure of function words (see Fries [63]), we can physically derive structural information about utterances of our language. However, a single structural analysis may admit a multitude of interpretations. If we say "ship sails today" do we mean "the ship sails today"? or "ship the sails today"? or even "shipped sails today"? Our thesis can be clearly stated: each symbol has a set of referents in someone's world; the logical structure of an utterance provides the means and the locants for the selection of a specific (person-specific) set of referents.

Research in document representation should not be interpreted as being purely syntactical in nature; rather, the identification of syntactical features is a prerequisite for the identification of higher order relations between data elements. As depicted in Fig. 25, research in document representation must be complemented by further basic research concerning how humans communicate (see Montgomery [62], Pepinsky [64]) in order to model satisfactorily how an index user presents his own "unique reality." We believe the goal of automatic document representation is to provide a scheme that permits the facile integration and interpretation of a user's *experience space*. In this way, the indexing system, rather than being rigid, provides a flexible interface that is adaptive to a user's intellectual structure (i.e., the relations that the user perceives among the classes of data contained in the document store).



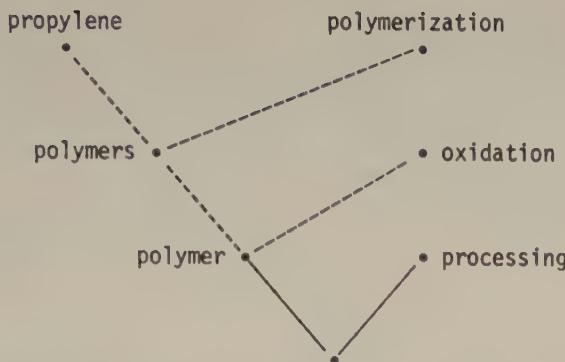
**Fig. 25.** Dynamic system representation amenable to user intellectual imposition of "meaning."

In the remainder of this section, several research efforts are described and certain experimental indexes are presented which should suggest to the reader the directions that the study of automatic indexing is tending.

### Examples of Research Efforts in Document Representation

#### TOSAR

Fugmann [65] has recently described the TOSAR—topological method for the representation of synthetic and analytical relations of concepts—system for the manual generation of graphical displays of concepts and relations. This system was developed to characterize chemical reactions, but the approach also appears to be valid for the representation of data contained in natural language utterances. For example, a string like "the processing of oxidized polypropylene" is represented by the graph



(where dashed lines indicate inferences drawn from the text). In this scheme, labeled nodes represent substances or processes. Each level of the graph (unlabeled nodes) represents a particular stage of a complex chemical process (e.g., combination, separation, and action of a catalyst). While this representation process is manually implemented at present, a detailed discussion of an algorithm for the computer manipulation of these structures may be found in Nickelsen and Nickelsen [66].

It should be obvious that once such graphs are prepared, they can be used for "substructure" searches or for "full-structure" searches, i.e., for "polypropylene processing" or "the processing of oxidized polypropylene." Figures 26a and 26b depict a more elaborate TOSAR graph. For additional details on the philosophy and implementation of this novel approach, the reader is directed to Refs. 65-67.

In the oligomerization of olefins, such as propylene, to obtain unsaturated oligomers, e.g. hexenes and nonenes, the activity of the catalyst steadily decreases as a result of the catalyst being loaded with the oligomers formed. A reusable catalyst is regenerated from the oligomerization mixture as follows.

Propylene and the oligomerization products are distilled off from the catalyst in a fractionation column A under pressure. In a second column B in series with the first, propylene is separated from the oligomer mixture, which consists mainly of hexene and nonene, by fractional distillation. The propylene is partly returned to the oligomerization reactor, while the remainder is introduced into the lower part of the column A to free the catalyst collecting there from the last traces of oligomer. This operation is carried out at a temperature below the decomposition range of the catalyst and below the boiling point of the contaminating oligomers.

The purified trialkylaluminum catalyst is continuously removed from the base of the column A and returned to the oligomerization reactor. The hexene fraction formed can be used for the production of isoprene by cracking.

Fig. 26a. Patent abstract.

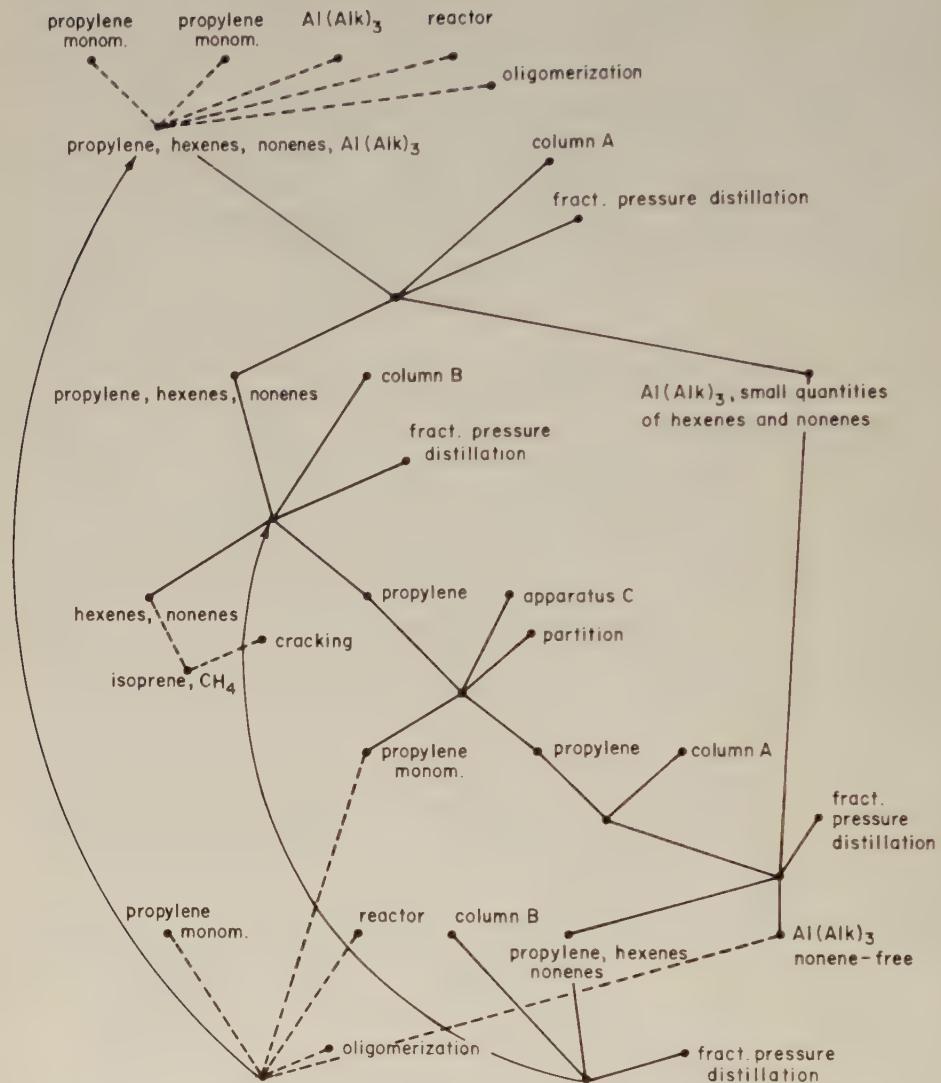


Fig. 26b. TOSAR representation (author's reproduction).

### GRAIT

Drawing on the Dependency Grammar introduced by Hays [68], the Conceptual Dependency Parser of Shank and Tesler [69], and the Quillian [70] concept/relation network model of human memory, Rush *et al.* [71–75] have developed a series of research algorithms generically labeled GRAFT (Graphic Representations for Automatic Indexing of Text), designed to transform English text into a metalanguage

SENTENCE: The routine microscopic blood inspection that is a universal feature of present medical practice is generally depended on as one of the most useful indicators of the state of a person's health.

STRUCTURAL  
REPRESENTATION:

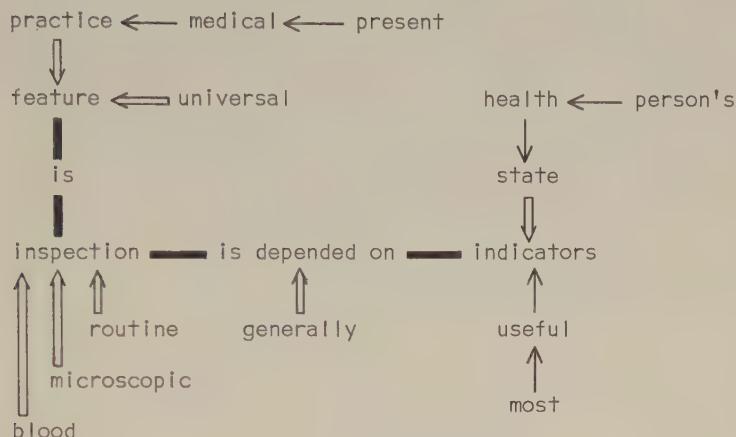


Fig. 27. Graphical representation developed by Rush et al. [75].

He finished the work after his friends left.

A      □      —— B      —— T      ○

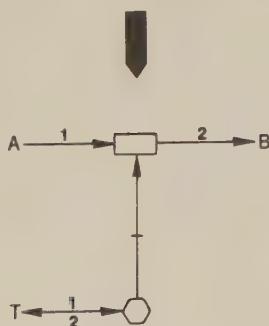


Fig. 28. A case grammar structural analysis and representation (after Young [73]).

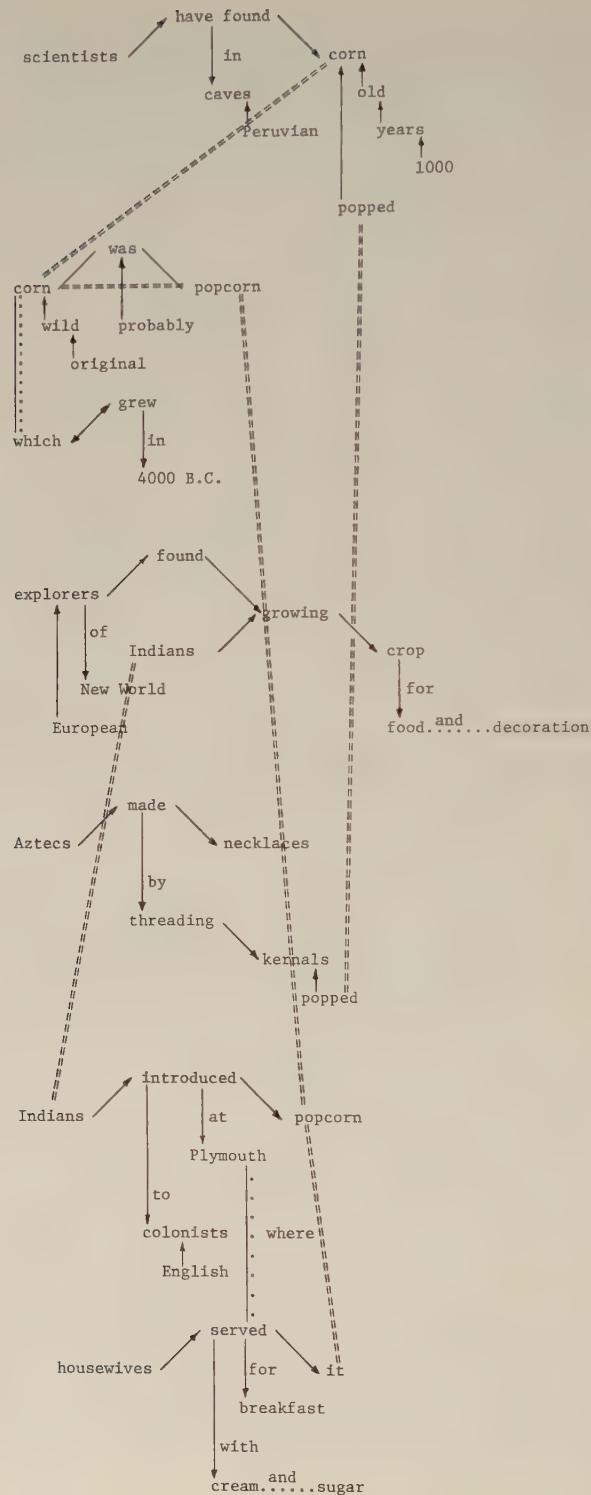


Fig. 29. Sentence representation developed by Strong [74].

suitable for a variety of automatic indexing processes. Among these algorithms are procedures for automatic syntax analysis, clause identification, and role analysis by means of case grammar.<sup>8</sup>

Basic to the GRAIT processing is the definition of *sentence* as an ordered triple,  $N_iRN_j$ , where  $N_i, N_j$  are names and  $R$  is a relation. Briefly, a *name* is a language string assigned to an object or construct and a *relation* is (1) a language string assigned to a behavior imputed to an object or construct, or (2) a language string which has only a linguistic function. Thus names are associated with nouns and noun phrases, and relations are associated with predicates and selected function words (see Young [73]).

In one experimental implementation of this categorization, a graph is formed for each input sentence in the following way. Construct a chain of names and primary relations. The names will consist of simple names or of the right-most word of a composite name. The relations in complex names are used to determine how the components of a complex name are to be joined to the main chain. Secondary relations are represented in the graph by directed edges (arcs) whose type and direction depend upon the specific secondary relation. Furthermore, pronouns and relative pronouns are replaced by their referents whenever these exist. The results of this procedure are illustrated in Fig. 27. An interesting variation of this basic procedure is depicted in Fig. 28. This representation, developed by Young [73], shows the names in the basic name-relation-name triple.

It should be pointed out that these structures, exhibited as graphical structures, are the result of computer manipulation of input data; hence only a linearization of the structures is stored in the memory of the computer system.

Finally, we consider the representation scheme (an extension of GRAIT) in the early stages of development by Strong [74]. The network displayed in Fig. 29 is based on a syntactic analysis and is constructed as follows. The verb (the primary element of each clause) is connected to its subjects and objects with diagonal edges. Vertical edges denote modification and dotted edges denote conjunction. Dashed edges depict implied and intersentence relationships. Major case grammar roles are identified by the type of verb-noun edge, and minor case roles may be used to label the modifier edges. It is believed that the resultant network constitutes an improved representation of text for natural language processing.

## CONCLUSION

The purpose of an index is to provide the user direct access to items of data in a collection which would otherwise be accessible only by sequential means. The problem of indexing is that of providing, *a priori*, direct access to data items that meet the unknown demands of unidentified users. The degree to which an index satisfies the demands of a user (or a collection of users) is in direct relation to its fidelity as a representation of the original data collection in which the user is interested. Thus, as

<sup>8</sup> For an extensive review of case grammar, see Refs. 76 and 77.

we have said, accurate retrieval depends on the exactness of the operation of the indexing system. Any process which renders an item of data inaccessible is an undesirable element of an indexing system.

The question is: Can an index be devised that will make all items of data in a collection directly accessible to all users all the time? The answer to this question is not known. But we can identify certain characteristics which such an index should exhibit:

1. The index, as the retrieval interface, must be dynamic rather than static.
2. The index must represent both *data elements* and the *relations* (explicit and implicit) among them.
3. The index must represent the original collection of data with high fidelity.

If such an index is to be realized, as we believe it can, then research effort needs to be directed toward those steps in the indexing process that we have called *identification* and *representation*. The work of Fugmann and his colleagues in Germany and of Rush and his associates in the United States appears to be converging to a solution of the representation problem, but no important results have yet been obtained toward the solution of the identification problem. Yet the development of data description languages [78], the work of Strong [74], and various developments in computational linguistics [62], pattern recognition [79], and artificial intelligence [80] suggest that the identification problem is solvable.

Computer-based indexing is in its infancy. A great deal of interesting and challenging work remains to be done to bring computer-based indexing to maturity.

## REFERENCES

1. M. Taube, *Studies in Coordinate Indexing*, Documentation Incorporated, Bethesda, Md., 1956.
2. M. E. Stevens, *Automatic Indexing: A State-of-the-Art Report*, National Bureau of Standards Monograph No. 91, March 30, 1965, p. 3.
3. J. C. Gardin, Semantic analysis procedures in the sciences of man, *Soc. Sci. Inf.* **8**(1), 24 (1969).
4. R. L. Collison, *Indexes and Indexing*, deGraff, New York, 1959, p. 20.
5. H. P. Edmundson, and R. E. Wyllys, Automatic abstracting and indexing—Survey and recommendations, *Commun. ACM* **4**(5), 226–234 (1961).
6. S. Artandi, A selective bibliography survey of automatic indexing methods, *Spec. Lib.* **54**(10), (1963).
7. M. E. Stevens, *Progress and Prospects in Mechanized Indexing*, working paper presented for the Symposium on Mechanized Abstracting and Indexing, Moscow, September 28–October 10, 1966.
8. System Development Corporation, *A System Study of Abstracting and Indexing in the U.S.*, 1966, PB-174-249.
9. C. A. Cuadra (ed.), *Annual Review of Information Science and Technology*, Wiley, New York (Vols. 1–2); Encyclopaedia Britannica, Chicago (Vols. 3–7), 1966–1972; American Society for Information Sciences (Vols. 8–10), 1973–1975.
10. L. B. Heilprin, *Mathematical Model of Indexing*, Documentation Inc., Bethesda, Md., 1957, AD-136-477.
11. B. C. Landry and J. E. Rush, Toward a theory of indexing, *Proc. Amer. Soc. Inf. Sci.* **5**, 59–64 (1968).

12. B. C. Landry and J. E. Rush, Toward a theory of indexing—II, *J. Amer. Soc. Inf. Sci.* **21**(5), 358–367 (1970).
13. B. C. Landry, *A Theory of Indexing: Indexing Theory as a Model for Information Storage and Retrieval*, Computer and Information Science Research Center Technical Report No. TR-71-13, The Ohio State University, 1971.
14. S. Artandi, Document description and representation, in *Annual Review of Information Science and Technology*, Vol. 5 (C. A. Cuadra, ed.), Encyclopaedia Britannica, Chicago, 1970, p. 161.
15. J. M. Carroll and R. Roeloffs, Computer selection of keywords using word-frequency analysis, *Amer. Doc.* **20**(3), 227 (1969).
16. B. C. Landry, B. A. Mathis, N. M. Meara, J. E. Rush, and C. E. Young, Definition of some basic terms in computer and information science, *J. Amer. Soc. Inf. Sci.* **24**(5), 328–342 (1973).
17. H. Wellisch, A flow chart for indexing with a thesaurus, *J. Amer. Soc. Inf. Sci.* **23**(3), 185–194 (1972).
18. H. L. Morgan, The generation of a unique machine description for chemical structures—A technique developed at Chemical Abstracts Service, *J. Chem. Doc.* **5**, 107–113 (1965).
19. C. H. Davis and J. E. Rush, *Information Retrieval and Documentation in Chemistry*, Greenwood, Westport, Conn., 1974, Chaps. 8 and 9.
20. J. Marschak, Economics of information systems, *J. Amer. Stat. Assoc.* **66**(333), 195 (1971).
21. *The Subject Index to Biological Abstracts*, BioSciences Information Service of Biological Abstracts, Philadelphia.
22. *The Biosystematic Index to Biological Abstracts*, BioSciences Information Service of Biological Abstracts, Philadelphia.
23. *CIN Index*, Chemical Abstracts Service, American Chemical Society, Columbus, Ohio.
24. S. Artandi, Computer indexing of medical articles—Project MEDICO, *J. Doc.* **25**(3), 214–223 (1969).
25. A. E. Petraca and W. M. Lay, The double-KWIC coordinate index, *J. Chem. Doc.*, **9**(4), 256–261 (1969).
26. J. Armitage and M. Lynch, Computer generation of articulated subject indexes, *Proc. Amer. Soc. Inf. Sci.* **6**, 253–257 (1969).
27. E. Garfield, Science Citation Index—A new dimension in indexing, *Science* **144**, 649–654 (1964).
28. H. Skolnik, The multi-term index: A new concept in IS&R, *J. Chem. Doc.* **10**(2), 81–84 (1970).
29. *HAIC Index*, Chemical Abstracts Service, American Chemical Society, Columbus, Ohio.
30. C. E. Granito and M. D. Rosenberg, Chemical Substructure Index (CSI)—A new research tool, *J. Chem. Doc.* **11**(4), 251–256 (1971).
31. R. Fugmann, W. Braun, and W. Vaupel, GREMAS—ein Weg zur Klassifikation und Dokumentation in der organischen Chemie, *Nachr. Dok.* **14**(4), 177–190 (1963).
32. A. K. Kent, The Chemical Society Research Unit in information dissemination and retrieval, *Sven. Kem. Tidskr.* **80**(2), (1968).
33. W. S. Hoffmann, An integrated chemical structure storage and search system operating at DuPont, *J. Chem. Doc.* **8**(1), 3–13 (1968).
34. *Patent Concordance*, Chemical Abstracts Service, American Chemical Society, Columbus, Ohio.
35. R. C. Cross, J. C. Gardin, and F. Levy, *L'automatisation des Recherches Documentaires, Un Model Generale: le SYNTOL*, Gauthier-Villars, Paris, 1964.
36. N. Bely, A. Borillo, N. Siot-Decauville, and J. Virbel, *Procédures d'analyse Semantique Appliquées et la documentation Scientifique*, Gauthier-Villars, Paris, 1970.
37. L. B. Doyle, Indexing and abstracting by association, *Amer. Doc.* **13**, 378–390 (1962).
38. D. A. Thompson, Interface design for an interactive information retrieval system: A literature survey and a research system description, *J. Amer. Soc. Inf. Sci.* **22**(6), 361–373 (1971).
39. H. Borko, Experiments in book indexing by computer, *Inf. Storage Retr.* **6**(1), 5–16 (1970).
40. J. R. Sharp, *Some Fundamentals of Information Retrieval*, London-House and Maxwell, New York, 1965, p. 94.
41. E. Garfield, Citation indexing for studying science, *Nature* **227**, 669–671 (1970).
42. B. A. Montague, Patent indexing by concept coordination using links and roles, *Amer. Doc.* **13**(1), 104–111 (1962).

43. S. Artandi, *Automatic Indexing of Drug Information, Project MEDICO—Final Report*, Graduate School of Library Science, Rutgers, The State University, New Brunswick, N.J., 1970, PB-190-807.
44. C. H. Davis and J. E. Rush, *Information Retrieval and Documentation in Chemistry*, Greenwood, Westport, Conn., 1974, Chap. 4.
45. J. Armitage and M. Lynch, *Articulation in the Generation of Subject Indexes by Computer*, presented at the 153rd National Meeting of the ACS Chemical Literature Division, Miami Beach, Florida, April 1967.
46. J. Armitage *et al.*, Experimental use of a programme for computer aided subject index production, *Inf. Storage Retr.* **6**(1), 79–87 (1970).
47. W. M. Lay, The Double-KWIC Coordinate Indexing Technique: Theory, Design, and Implementation, Ph.D. Thesis, Department of Computer and Information Science, The Ohio State University, 1973.
48. A. E. Petrarca and W. M. Lay, The double-KWIC coordinate index II: Use of an automatically generated authority list to eliminate scattering caused by some singular and plural main index terms, *Proc. Amer. Soc. Inf. Sci.* **6**, 227–282 (1969).
49. D. C. Colombo and J. E. Rush, Use of word fragments in computer based retrieval systems, *J. Chem. Doc.* **9**(1), 47–50 (1969).
50. E. G. Smith, *The Wiswesser Line-Formula Chemical Notation*, McGraw-Hill, New York, 1968.
51. F. A. Tate, Handling chemical compounds in information systems, in *Annual Review of Information Science and Technology*, Vol. 2 (C. A. Cuadra, ed.), Wiley, New York, 1967, pp. 285–309.
52. M. J. Ebersole, A General Method for Automatic Generation of Data Record Descriptors, M.S. Thesis, The Ohio State University, 1971.
53. R. Fugmann and W. Bitterlich, Reaktionen Dokumentation mit dem GREMAS System, *Chem. Ztg.* **96**(6), 323–329 (1972).
54. E. Meyer, The IDC System for chemical documentation, *J. Chem. Doc.* **9**(2), 109–113 (1969).
55. Ref. 37, p. 383.
56. J. L. Bennett, On-line access to information: NSF as an aid to the indexer/cataloger, *Amer. Doc.* **20**(3), 213–220 (1969).
57. J. H. Williams, Browser: An automatic indexing on-line text retrieval system, AD-693-143, 1969.
58. H. P. Luhn, The automatic creation of literature abstracts, *IBM J. Res. Develop.* **2**(4), 159–165 (1958).
59. G. Salton, *Automatic Information Organization and Retrieval*, McGraw-Hill, New York, 1968.
60. J. A. Moyne, Information retrieval and natural language, *Proc. Amer. Soc. Inf. Sci.* **6**, 259–263 (1969).
61. Ref. 3, p. 34.
62. C. A. Montgomery, Linguistics and information science, *J. Amer. Soc. Inf. Sci.* **23**(3), 195–219 (1972).
63. C. Fries, *The Structure of English*, Harcourt, Brace & World, New York, 1952.
64. H. B. Pepinsky (ed.), *People and Information*, Pergamon, Elmsford, N.Y., 1970.
65. R. Fugmann *et al.*, TOSAR—A topological method for the representation of synthetic and analytical relations of concepts, *Angew. Chem. Int. Ed. Engl.* **9**(8), 589–595 (1970).
66. I. Nickelsen and H. Nickelsen, Mathematische Analyse des TOSAR-Verfahrens, *Inf. Storage Retr.* **9**, 95–119 (1973).
67. R. Fugmann, The theoretical foundation of the IDC system: Six postulates for information retrieval, *ASLIB Proc.* (February 1972).
68. D. G. Hays, Dependency theory: A formalism and some observations, *Language* **40**(4), 511–525 (1964).
69. R. C. Shank and L. Tesler, *A Conceptual Dependency Parser for Natural Language*, Preprint #2 from the International Conference on Computational Linguistics, Sweden, 1969.
70. M. R. Quillian, Semantic memory, in *Semantic Information Processing* (M. Minsky, ed.), MIT Press, Cambridge, Mass., 1968.
71. J. E. Rush, *A New Approach to Indexing*, presented before the Student Chapter of the A.S.I.S., Case-Western Reserve University, Cleveland, Ohio, January 1969.

72. *The Ohio State University, Computer and Information Science Research Center*, Report to the National Science Foundation, August 1972, p. 2.1.
73. C. E. Young, Development of Language Analysis Procedures with Application to Automatic Indexing, Ph.D. Thesis, The Ohio State University, 1973.
74. S. M. Strong, An Algorithm for Generating Structural Surrogates of English Text, M.S. Thesis, The Ohio State University, 1973.
75. J. E. Rush, H. B. Pepinsky, B. C. Landry, N. M. Meara, S. M. Strong, J. A. Valley, and C. E. Young, *A Computer Assisted Language Analysis System*, The Ohio State University, Computer and Information Science Research Center, 1973.
76. W. A. Cook, S.J., *Introduction to Tagmemic Analysis*, Holt, Rinehart & Winston, New York, 1969.
77. C. J. Fillmore, The case for case, in *Universals in Linguistic Theory* (E. Bach and R. Harms, eds.), Holt, Rinehart & Winston, New York, 1968, pp. 1-88.
78. Anon., *June 1973 Report of the CODASYL Data Base Task Group*, available from the Association for Computing Machinery, New York, 1973.
79. M. S. Watanabe (ed.), *Frontiers of Pattern Recognition*, Academic, New York, 1972.
80. M. A. Arbib, *The Metaphorical Brain*, Wiley-Interscience, New York, 1972.

#### ADDITIONAL REFERENCES

- Barlow, H., and S. Morgenstern, *Dictionary of Musical Themes*, Crown, New York, 1948.
- Bowles, E. A. (ed.), *Computers in Humanistic Research*, Prentice-Hall, Englewood Cliffs, N.J., 1967.  
See particularly Section IV, "Computers in Language and Literature" and Section V, "Computers in Musicological Research."
- Kehl, W. B., et al., An information retrieval language for legal studies, *Commun. ACM* **4**, 380-389 (1961).
- Kosch, A., and D. Aichele, *Was bluht denn da?*, Kosmos-Gesellschaft der Naturfreunde, Franckh'sche Verlagshandlung, Stuttgart, 1966.
- Overmeyer, L., The evolution of a computer-produced index for diabetes literature, *Proc. Amer. Soc. Inf. Sci.* **3** (1966).
- Tuckerman, B., Planetary, lunar and solar positions, 601 B.C. to A.D. 1, at five-day and ten-day intervals, *Mem. Amer. Phil. Soc.* **56** (1962); and Planetary, lunar and solar positions, A.D. 2 to A.D. 1649, at five-day and ten-day intervals, *Mem. Amer. Phil. Soc.* **59** (1964).

*Bertrand C. Landry and James E. Rush*

# AUTOMATICALLY PROGRAMMED TOOLS—NUMERICALLY CONTROLLED MACHINE TOOLS

Probably the most universally accepted definition of numerical control (N/C) is that offered by the Electronics Industries Association (EIA) which defines N/C as "a system in which actions are controlled by the direct insertion of numerical data at some point. The system must automatically interpret at least some portion of this data." Numerical control is not simply an electronic replacement for a machine operator, but is a manufacturing concept. N/C affects virtually every aspect of the manufacturing process. When applied to small production lots or parts requiring complex machining, N/C can offer distinct advantages over traditional manufacturing methods. Some of the potential savings include reduced setup time, reduced tooling costs, improved machine utilization, reduced scrap and inspection costs, and reduced inventory levels. In addition, such intangibles as greater ease of incorporating engineering revisions, greater freedom of design, reduced lead time, and greater accuracy contribute to the advantages of N/C.

Basically, an N/C machine consists of a reader, a control unit, and the machine tool. The input medium contains preprogrammed machine instructions, which are transmitted to the control unit by the reader. The control unit interprets an instruction and causes the machine tool to execute it. This process is repeated until the job is finished.

Preparation of the input medium is the function of the part programmer who studies the workpiece drawing and determines the best machining sequence. He then converts this sequence into the coded instructions which are used to control part production. The part programmer looms very important in the N/C operation as it is he who controls the efficiency of N/C machining.

To date, N/C has been applied mostly to conventional metal removal equipment; however, applications such as assembly, packaging, wire wrapping, tube bending, inspection, and material storage have been done with very favorable results. In addition, many new applications are anticipated because of the potential advantages offered by N/C.

## HISTORY OF NUMERICAL CONTROL

Numerical control is usually thought of as a modern concept; however, systems quite similar to present-day numerical control were used 250 years ago. M. Falcon invented a perforated-card-controlled knitting machine in the early 1700s. The needles of the knitting machine were controlled via mechanical linkages established by cards

which were chain-synchronized to the action of the machine. Although this machine was not nearly as refined as modern N/C machines, it did operate on a similar principle using replaceable cards to make different patterns in much the same way that tapes are used today to change machining patterns.

A later device, the automatic player piano, patented by M. Fourneaux in 1863, was so similar to modern numerical control that it read perforated-paper tape to actuate the keys. It is interesting to note that the tape was read by air, a principle that is still used on some numerical-control tape readers.

Numerical control as it is presently known did not originate until 1947. At that time, John T. Parsons needed a quick, economical, and accurate way of producing templates for inspecting helicopter blades. Previously, these templates were made by jig-boring overlapping holes to define the contour, and then by blending with a file.

Parsons conceived a new way of doing the job by coupling data processing equipment with a jig borer. Thus Parsons' accomplishment marked the birth of numerical control.

On the strength of this development, the USAF Air Material Command awarded Parsons a study contract which was concerned with the ability of the aerospace industry to get quickly into emergency production of highly complex and frequently modified products. Parsons turned to the Servomechanisms Laboratory of M.I.T. for assistance, and later M.I.T. was awarded the prime contract for the development of an experimental milling machine under digital control. In 1952, M.I.T. successfully demonstrated a prototype N/C machine.

The acceptance of numerically controlled machines by industry is best illustrated by the sales of numerical control equipment. From the time that M.I.T. completed its original experimentation in 1954 until 1959, there were only 179 N/C machines delivered to industry. These machines were mainly complex contouring machines often costing several hundred thousand dollars. In the next 5 years, approximately 3400 N/C machines were sold. This dramatic increase was largely caused by a greater emphasis on point-to-point systems and mass production of N/C machines. In 1962, one manufacturer used mass production techniques to reduce the price of N/C drilling machines to less than \$10,000. Demand for N/C equipment has continued to grow as a result of further improvements in control systems, reduced costs, and improved programming techniques.

## AXIS NOMENCLATURE

The Cartesian or rectangular coordinate system is well suited for describing motion of the two or three mutually perpendicular axes normally found in machine tools. Standards governing the names and sign conventions applied to the axes in machine tools have been established by EIA in their document RS267 and have been adopted by the International Standards Organization (ISO) in their document ISO/R841. The guiding principle in these standards is the use of the conventional right-hand coordinate

system within which movement of the cutting tool is described relative to the workpiece.

For machines capable of doing drilling operations, the  $z$  axis is selected to be parallel to drill movement such that retracting the tool from the workpiece is a movement in the  $+z$  direction. The  $x$  axis is perpendicular to the  $z$  axis, parallel to the work-holding surface, and horizontal wherever possible. Where more than one axis fits this description, the axis permitting the longer movement is the  $x$  axis. The  $+x$  direction is normally to the right when facing a machine that has the principal rotating tool mounted vertically. For machines on which the principal rotating tool is horizontal, the  $+x$  direction is to the right when looking from the tool toward the workpiece. For machines such as lathes, for which the workpiece rotates, movement of the cutting tool radially outward from the spindle defines the  $+x$  direction.

For machines such as shapers that are not capable of drilling-type operations, the  $z$  axis is perpendicular to the work-holding surface and positive in the direction that withdraws the tool from the work. The  $x$  axis is parallel to the principal direction of cut and positive in the direction of cutting.

In all cases the  $y$  axis is defined in terms of the  $x$  and  $z$  axes such that the conventional right-hand coordinate system is completed.

For machines having more than one movement in the  $x$ ,  $y$ , or  $z$  directions, the secondary motions are named  $u$ ,  $v$ , and  $w$ ; and the tertiary motions are  $p$ ,  $q$ , and  $r$  respectively. The letter  $r$  can also be used as a rapid travel command in the  $z$  axis.

Where rotation is possible about one or more of the primary axes, the letters  $a$ ,  $b$ , and  $c$  are used to refer to rotation about the  $x$ ,  $y$ , and  $z$  axes, respectively. The direction of positive rotation is determined by pointing the thumb of the right hand in the positive  $x$ ,  $y$ , or  $z$  direction; the fingers will then curl in the positive rotational direction. When absolute angles of rotation are required, location is relative to lines parallel to and in the same direction as  $+y$ ,  $+z$ , and  $+x$  for  $a$ ,  $b$ , and  $c$ , respectively.

## CONTROL SYSTEMS

Numerical control systems are of two basic types—point-to-point and continuous path contouring. Point-to-point systems direct the tool to a position without regard for the path taken. Since the controller need not coordinate the axes to achieve a particular path, control is comparatively simple and inexpensive. This type of system is well suited for machines such as drill presses, punch presses, and spot welders where an operation is performed after positioning. The addition of feed-rate and overshoot control can extend the capability of the point-to-point controller to accommodate operations such as milling parallel to the  $x$  or  $y$  axes.

Continuous-path contouring systems not only control the final position of the machine tool, but also the path taken to that position. By coordinating the feed-rate control of the axes, control of the tool path is achieved. Three types of paths are available—linear, circular, and parabolic. The simplest and most widely used type of contouring is linear interpolation for which the feed-rates of the axes are coordinated

to yield a straight line movement. Circular paths are available through circular interpolation, which uses linear contouring moves to approximate a circular arc. Parabolic interpolation allows the path to be a parabola through the starting point, an intermediate point, and the end point. This type of interpolation finds its greatest application in making free-form shapes. Coordinated axes motion requires more controller sophistication and hence more expense than point-to-point systems. Contouring control is most often found on milling machines and lathes.

## INFORMATION TO THE MACHINE

During the life-span of N/C, a variety of input media have been used including motion-picture-type film, computer cards, 5-in.-wide punched plastic tape, and perforated paper tape. Manufacturers and users of N/C machines soon realized the need for standardization. Consequently, in 1961 EIA designated the standard input medium as 1-in.-wide, 8-channel, perforated tape (document RS244).

The same document also specifies punching codes for the symbols, letters, and numerals used in N/C. This codeset uses channel 8 to represent an end-of-block or carriage return and channel 5 to provide an odd parity. The remaining six channels provide codes for up to 64 symbols. The N/C industry found this codeset adequate, but the computer and communications industries wanted maximum capacity from the eight channels. As a result, they established ASCII (American Standard Code for Information Interchange): a codeset based on an even parity with seven channels for symbol representation. Although incompatible with EIA codes, ASCII provides twice as many possible codes. In 1968, ISO adopted the ASCII codeset as its standard (document ISO/R840); however, the United States N/C industry was deeply committed to the EIA code and has continued to use it.

The EIA and ISO codesets are read with two types of tape readers: block and single character. A block reader reads an entire block or command consisting of several characters at a time. Single-character readers read individual characters assembling the command in a memory or storage. Although the memory adds to the price of single-character readers, it also allows deletion of information that is unchanged from one block to the next with resulting shorter tapes and lower tape preparation costs. As developments in electronics have reduced the cost of memories, the single-character readers have increased in popularity to the point that few new machines have block readers.

Regardless of the type of reader, its purpose is to convey commands to the control system. The commands must tell the machine where it is to go, how it is to get there, what it is to do on the way, and what is to be done upon reaching the desired location. This information is supplied via a series of words that make up the command. The desired location is specified via a dimension word for each axis. A two-digit preparatory function word is used to specify the type of operation, i.e., point-to-point move, linear interpolation, drilling cycle, cutter compensation. The feed, speed, and tool used are

indicated in their respective words. In addition, a two-digit miscellaneous function word controls such things as the direction of spindle rotation, coolant, clamping, and program stops. A three-digit sequence number is also contained within the command to inform the operator of the current program step.

Commands for a particular machine may require all or part of these words. The form of the command is dependent upon the format used. The four most common formats are the fixed sequential, word address, tab sequential, and the universal or tentative compatible format.

The fixed sequential format was developed for use with block readers; consequently, each word must appear in order in every command whether or not it has changed from the previous command. This redundancy increases the length of the tape and increases tape preparation costs; however, it allows use of the less expensive type of reader.

The remaining three formats are intended for use with single-character readers. The memory allows the redundancy to be eliminated. The word address format is based on inclusion of a letter at the beginning of each word to direct the information to the appropriate storage area. The tab sequential format is quite similar to the word address format, except a tab is included in each word instead of the letter. The words comprising the command are supplied in a specific order as with the fixed sequential format. However, with the tab sequential format the words can be shortened to just the tab if the information has not changed. The tabs also allow the words to be arranged in easily read columns on the printout. The universal or tentative compatible format is a combination of the word address and tab sequential formats. In the universal format both the letter designation and the tab are included in each word. Although this format results in longer tapes than the tab sequential format, it makes the printout easier to read since the address letter is included in each column. All three of these formats usually will result in shorter tapes and hence lower tape preparation costs than the fixed sequential format.

## PART PROGRAMMING

Before any numerical control unit can be used it must be programmed for the desired part. Programming can either be done manually or with the aid of a computer. Although not necessary, the computer is very useful in eliminating laborious calculations in complicated problems. In addition, the computer can be coupled with a tape punch to directly produce N/C tapes. The resulting reduction in manual computation and data handling may result in greater speed, accuracy, and ease of programming via the computer.

Whether or not a computer is used, programming starts with the blueprint. Conventional drawings may need to be modified to more closely parallel the N/C coordinate system for greater ease of programming. It may even be desirable to alter the part design to take advantage of the increased capability offered by N/C. Nevertheless, the

drawings and part specifications are the basis for the N/C program. The part programmer is responsible for this transition. He must determine how to make the part—the setup, the sequence of operations, the tooling, the cutter paths, feeds, and speeds. These decisions must be translated into machine instructions.

### **Manual Programming**

In manual programming the part programmer computes the values of the words and formulates the machine instructions on a program sheet or manuscript. The tape is manually punched from the manuscript via a tape preparation typewriter. Once punched, the tape must be checked for errors. Tape checking techniques include comparing the hard copy with the manuscript and retyping the manuscript in the verification mode as well as trials on the machine. The final test is not passed, however, until a good part has been produced via the tape.

### **Computer-Assisted Programming**

In computer-assisted programming the part programmer symbolically codes the machining instructions for input to the computer. Three types of computer programs are available to convert the symbolic code to machine instructions: general purpose, special purpose, and user-written programs. Automatic programmed tool (APT) is the primary general purpose N/C programming language. Since its inception in 1961, APT has been very important to the growth and development of N/C. Its capability ranges from simple positioning to complex 5-axis milling. Continual improvements are being provided by the Illinois Institute of Technology Research Institute under the APT Long Range Program which is supported by the major aerospace, automotive, machine tool, and control unit manufacturers. The APT system is by far the most comprehensive, versatile, and powerful N/C programming system in existence.

APT processing first provides a generalized solution to each part program. This solution is then used by a postprocessor to produce a tape for a particular N/C machine. The communication from the generalized or machine-independent solution to the particular solution is the cutter location (CL) file, which contains the tool end coordinates. Postprocessing allows the generalized solution to be translated into the tape formats for different machine/controller combinations.

Part programming with APT requires a machining background in addition to computer programming knowledge. Since APT is designed to describe the actual machining operation, several conventions are used to cover the variety of machine tools. The tool is considered to move relative to a fixed workpiece. For contouring, control of the tool path is established by considering the tool to slide along two surfaces known as the drive and part surfaces. Motion continues along these two surfaces until a third surface, the check surface, is encountered. For point-to-point work, the tool path is not important and, hence, the final tool position is specified explicitly.

The surfaces, tools, and tool motions are specified via geometry definition, tool

definition, and tool motion commands, respectively. Geometry definitions are used to describe the part, drive, and check surfaces in terms of planes, cylinders, spheres, cones and general quadratic surfaces. In the special case of 2-axis  $xy$  contouring, lines and circles can be defined; but in reality, they are infinite planes and cylinders perpendicular to the  $xy$  plane. The tool is defined as a three-dimensional surface of revolution. The tool motion commands are used to move the tool about the workpiece in the desired machining sequence, and the direction of each tool movement is considered relative to the last movement. The proper direction is determined by imagining oneself riding on the tool and going either right (GORGT) or left (GOLFT), forward (GOFWD) or backward (GOBACK), up (GOUP) or down (GODOWN) from the last movement. These three types of APT statements then are used to provide the tool coordinate information for the CL file.

Other types of APT statements are used for the N/C machine auxiliary functions and for editing purposes such as printing, punching, copying and cutter path matrix transformations. Since the APT compiler is primarily written in FORTRAN, loops, subroutines, and FORTRAN-like expressions can be used to great advantage in the part program.

To illustrate the use of APT for two-dimensional contouring and point-to-point applications, consider the part drawing of Fig. 1 and the APT program listing of Fig. 2. An explanation of the APT statements used in the part program follows to aid in understanding the APT language.

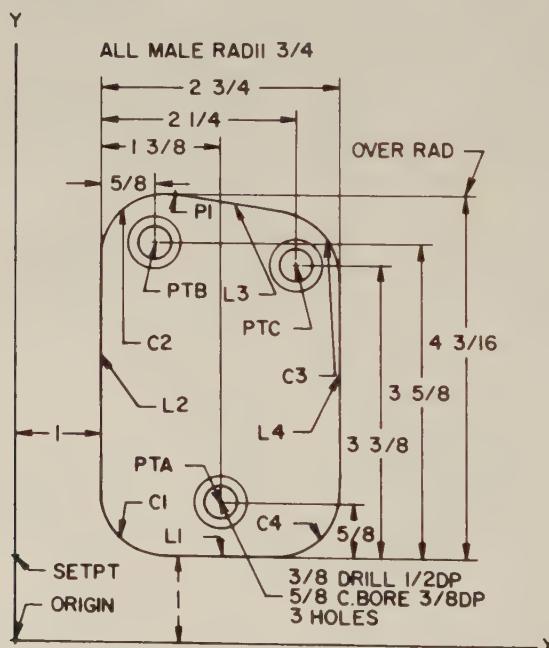


Fig. 1. APT part drawing.

```

1.      NO POST
2.      CLPRNT
3.      SETPT=POINT/0, 1
4.      L2=LINE/1, 0, 0, 1, 1, 0
5.      L1=LINE/SETPT, PERPTO, L2
6.      L4=LINE/PARREL, L2, XLARGE, 2.75
7.      C1=CIRCLE/YLARGE, L1, XLARGE, L2, RADIUS, .75
8.      C4=CIRCLE/YLARGE, L1, XSMALL, L4, RADIUS, .75
9.      C2=CIRCLE/1.75, (5.1875 - .75), .75
10.     PI=POINT/C2, ATANGL, 81.25
11.     L3=LINE/PI, ATANGL, -8.75, XAXIS
12.     C3=CIRCLE/YSMALL, L3, XSMALL, L4, RADIUS, .75
13.     PTA=POINT/2.375, 1.625
14.     PTB=POINT/1.625, 4.625
15.     PTC=POINT/3.25, 4.375
16. PPRINT START JOB WITH 1 INCH DIAMETER END MILL
17. PPRINT SET HEIGHT OF CUTTER .500 INCHES BELOW PART SURFACE
18.     CUTTER/1.0
19.     FEDRAT/10
20.     FROM/SETPT
21.     GO/TO, L2
22.     TLLFT, GOLFT/L2, TANTO, C2
23.     GOFWD/C2, TANTO, L3
24.     GOFWD/L3, TANTO, C3
25.     GOFWD/C3, TANTO, L4
26.     GOFWD/L4, TANTO, C4
27.     GOFWD/C4, TANTO, L1
28.     GOFWD/L1, TANTO, C1
29.     GOFWD/C1, TANTO, L2
30.     GOTO/SETPT
31.     STOP
32. PPRINT CHANGE TO SUBLAND DRILL 5/8D COUNTERBORE X 3/8D
     DRILL
33. PPRINT SET HEIGHT OF DRILL .500 INCHES ABOVE PART SURFACE
34.     CUTTER/.625
35.     RAPID
36.     GOTO/PTA
37.     GODLTA/0, 0, -1.125, 5
38.     GODLTA/0, 0, 1.125
39.     RAPID
40.     GOTO/PTB
41.     GODLTA/0, 0, -1.125
42.     GODLTA/0, 0, 1.125
43.     RAPID
44.     GOTO/PTC
45.     GODLTA/0, 0, -1.125
46.     GODLTA/0, 0, 1.125
47.     GOTO/SETPT
48.     FINI

```

**Fig. 2.** APT part program listing.

**NOPOST**

results in no postprocessing and, hence, no paper tape. To generate tape for a particular N/C machine a command of MACHIN/xxxxx would select the proper post-processor to create the tape.

**CLPRNT**

causes the tool end coordinates stored in the CL file to be printed upon completion of the generalized solution.

[Statements 3–15 are geometry definition.]

**SETPT=POINT/0, 1**

defines the point SETPT by its *x* and *y* coordinates.

**L2=LINE/1, 0, 0, 1, 1, 0**

defines the line L2 by the coordinates of two points on the line. The LINE in APT actually defines a plane perpendicular to the *xy* plane.

**L1=LINE/SETPT, PERPTO, L2**

defines the line L1 as perpendicular to L2 and passing through SETPT.

**L4=LINE/PARREL, L2, XLARGE, 2.75**

defines the line L4 as parallel to L2 and offset 2.75 in. in the +*x* direction

**C1=CIRCLE/YLARGE, L1, XLARGE, L2, RADIUS, .75****C4=CIRCLE/YLARGE, L1, XSMALL, L4, RADIUS, .75**

define circles C1 and C4 as being tangent to two lines with radii of 0.75 in. The modifiers before each line specify which of the four possible circles is desired.

**C2=CIRCLE/1.75, (5.1875-.75), .75**

defines the circle C2 by the *x* and *y* coordinates of its center and the value of the radius, respectively. Notice the use of the arithmetic expression for the *y* coordinate.

**P1=POINT/C2, ATANGL, 81.25**

defines P1 as the intersection of the periphery of C2 and a radial line at an angle of 81.25°. This definition was used to define the point of tangency of C2 and L3.

**L3=LINE/P1, ATANGL, -8.75, XAXIS**

defines L3 as passing through P1 and at an angle of -8.75° with the *x* axis.

**C3=CIRCLE/YSMALL, L3, XSMALL, L4, RADIUS, .75**

is similar to statements 7 and 8.

**PTA=POINT/2.375, 1.625****PTB=POINT/1.625, 4.625****PTC=POINT/3.25, 4.375**

defines the *x* and *y* coordinates of the centers of the three holes

**PPRINT START JOB WITH 1 INCH DIAMETER END MILL**

**PPRINT SET HEIGHT OF CUTTER .500 INCHES BELOW PART SURFACE**

provide program documentation and also setup information to the machine operator.

**CUTTER/1.0**

is a tool definition statement which describes a 1-inch diameter end mill.

**FEDRAT/10**

is an auxiliary statement which specifies a 10-inches-per-minute feedrate.

**FROM/SETPT**

defines SETPT as the starting point for the following motion commands.

**GO/TO, L2**

causes the tool to GO via the shortest path TO a point of tangency on the near side of the check surface, L2. TO, ON, and PAST are modifiers which specify the final position of the tool relative to the check surface as: tangent to the near side, centered on, or tangent to the far side, respectively. In addition, the final position of the tool can be defined in terms of the point of tangency of the drive and check surfaces via the TANTO modifier.

**TLLFT, GOLFT/L2, TANTO, C2**

causes the tool to make a left turn (GOLFT) and to continue along the drive

surface, L2, to the point where the drive surface is tangent to (TANTO) the check surface, C2. The TLLFT word specifies the tool to be tangent on the left side of the drive surface looking in the direction of motion. This relationship applies until a new relationship is specified. The other possible tool-drive surface relationship are tool tangent on the right side (TLRGT) and tool centered on the drive surface (TLON).

GOFWD/C2, TANTO, L3

GOFWD/L3, TANTO, C3

GOFWD/C3, TANTO, L4

GOFWD/L4, TANTO, C4

GOFWD/C4, TANTO, L1

GOFWD/L1, TANTO, C1

GOFWD/C1, TANTO, L2

complete the machining of the outside of the workpiece. Notice that the current check surface becomes the drive surface of the next command.

GOTO/SETPT

STOP

The GOTO/SETPT ends the contouring and returns the tool to SETPT. Notice the difference between GOTO/ which is a point to point command and GO/TO which is a startup command for contouring. The STOP command stops the machine for a tool change.

PPRINT CHANGE TO SUBLAND DRILL 5/8D COUNTERBORE X 3/8D  
DRILL

PPRINT SET HEIGHT OF DRILL .500 INCHES ABOVE PART SURFACE  
provide instructions to the machine operator.

CUTTER/.625

defines the cutter as a 5/8 inch drill.

RAPID

GOTO/PTA

These statements move the tool under rapid travel to PTA. The RAPID command only applies to the motion command following it.

GODLTA/0, 0, -1.125, 5

specifies an incremental point to point motion relative to the previous location. Since the only motion is a z movement, this command causes the drill to feed down  $1\frac{1}{8}$  in. at a feedrate of 5 in./min. The fourth argument in this statement is an optional feedrate which in this case replaces the previous 10-in./min. feedrate.

GODLTA/0, 0, 1.125

retracts the drill.

RAPID

GOTO/PTB

GODLTA/0, 0, -1.125

GODLTA/0, 0, 1.125

RAPID

GOTO/PTC

GODLTA/0, 0, -1.125

GODLTA/0, 0, 1.125

result in similar point-to-point motions for drilling the holes at PTB and PTC.

GOTO/SETPT

returns the tool to SETPT.

FINI

signifies the end of the part program and stops the machine.

This program would produce the generalized solution and print the CL file which contains the input data for the postprocessor. The function of the postprocessor is to

convert a general APT solution to the program for a particular N/C machine. The postprocessor considers the machine tool and controller capabilities and structures the commands into the proper tape format and coordinate system. Selection of speeds, feeds, and auxiliary functions; proper acceleration/deceleration of machine slides; and allowable overshoot can be provided in the postprocessor. When possible, the postprocessor takes advantage of any advanced interpolation capability (circular and/or parabolic) of the controller. In addition, checks may be made to insure that maximum axes travel has not been exceeded, that the tool and machine do not interfere, and that an "input bound" condition does not exist at the reader. In essence, the postprocessor customizes the general solution to fit the particular N/C machine.

A family of APT-based programming languages including ADAPT, AUTOSPOT, SUBADAPT, and UNIAPT are available for use on smaller computer systems. These languages offer a subset of the capabilities of APT while retaining many of its characteristics. In addition, most of these languages can use the APT postprocessors.

In summary, with APT the user can satisfy virtually all of his programming requirements from simple 2-axis positioning to complex 5-axis contouring. In addition, part programmers need only be trained for one language. The extensive geometric and arithmetic capability of APT permits programming of extremely complex parts. Through the use of postprocessors, APT general solutions can be translated into tape for many different machine/controller combinations. However, the many advantages of APT depend upon extremely large computers. Since few organizations own computers as large as 256K, in-house availability of APT is severely limited. In addition, the flexibility of APT generally entails greater processing cost for applications where programming languages of a more specialized nature are available.

Although quite similar to general purpose programs, special-purpose programs are designed for a particular control system, machine tool, computer, or special type of work. They need contain nothing extraneous to the application and, therefore, may require less computer time and/or a smaller computer. Their specialized nature may also make programming easier and eliminate the need for postprocessing. Unfortunately, special-purpose programs have an inherent limitation: they lack the flexibility of a general purpose program. Changing demands may result in the need for additional programs with the resulting inconvenience of different part programming languages and perhaps different computers. This lack of flexibility must be carefully balanced against the advantages of a special-purpose language when considering implementation of computer-assisted programming.

User written programming languages are a special category of special purpose programs. These languages are tailor-made for the user's particular needs. Therefore, they offer the possibility of still greater efficiency, smaller computers, and greater ease of programming. The major drawback with this type of program is developmental cost in time, talent, and money. This drawback and the increasing availability of a wide variety of special purpose programs are reducing the popularity of user written languages.

## ADVANCES IN N/C—DNC AND ADAPTIVE CONTROL

### DNC

Direct numerical control (DNC) enhances the capability of conventional N/C by utilizing an on-line computer to supply input directly to the machine. As a result, machine input can be considerably faster, more flexible, and more responsive to programming changes. In addition, the computer may be used to record and feedback machine and operator performance as the basis of a real-time management information system.

At present there are two basic types of DNC systems: plug-in and integrated. The plug-in system is so named because the computer is "plugged in" behind the tape reader. With this method the on-line computer supplies the controller with the same information that would be supplied by a conventional tape reader. In an integrated system the computer supplies interpolated data directly to the controller. With this system the tape reader and hard wired interpolator of conventional N/C controllers would be eliminated. The main difference between the two systems is the location of the interpolation—either in the controller or in the computer.

These systems offer several advantages. The tape reader can be eliminated, which means not only a several thousand dollar reduction in initial cost, but also an elimination of one of the most troublesome components of the N/C system. Since perforated tape is no longer required, there is also a savings in tape preparation, handling, and storage costs. Part program debugging may be greatly simplified through interactive communication between the programmer and the computer. In addition, the basis of a real-time management information system can be provided by monitoring operator and machine performance. DNC may also be used to compensate for predictable machine error.

Although both types of DNC have the above advantages, each system provides unique advantages. If the DNC computer fails, the plug-in system may allow operation of the machine if a tape reader is available whereas the integrated system would be inoperable. The plug-in system allows remote location of the computer and, thus, control of several machines simultaneously. The integrated DNC computer must be relatively close to the machine tool because of the high data transmission rate required. Fewer machines can be controlled simultaneously by this system because of the allowable length of the transmission lines. An integrated system eliminates the substantial cost of the hard wired interpolators. The major disadvantage of either system, at present, is the initial cost. Although computer hardware costs are diminishing, the DNC system generally must control more than one machine to be competitive with conventional N/C.

### Adaptive Control

Despite the advantages that N/C offers, it has not solved the problem of what feeds and speeds should be used during the cutting operation. In fact, the problem has been

increased since the part programmer may make his selections without the benefit of observing the results. Considerable research has been directed at the problem of finding optimal feeds and speeds for a variety of cutting conditions. Unfortunately, the ever-increasing combinations of work and tool materials as well as the ever-changing conditions of cut make this a continuing problem. A possible solution is offered by the application of adaptive controls to metal cutting machines. R. M. Centner defines an adaptive control system as "any system that measures some aspect of an actual machining operation and uses this measure to adjust a controlled input. . . ."

Ideally an adaptive control system should monitor the cutting operation and adjust the feed and speed so as to maximize profit. Research attempts to develop such a system have encountered difficulty in measuring tool wear on-line, and consequently have failed to produce such a system. Instead, research efforts have resulted in systems which attempt to either maximize the metal removal rate while operating within a series of constraint limits or adjust feed and speed according to empirical relationships. Despite their limitations, these systems have substantially increased the productivity of metal cutting machines. Unfortunately, the high cost of these research units made them impractical for commercial applications; therefore, less sophisticated units have been developed which yield the majority of these improvements. Further cost reductions can be realized when the adaptive control unit can be integrated with a DNC system such that the DNC computer performs the adaptive control computations. In the future, utilization of the DNC computer by the adaptive control unit may make more sophisticated adaptive control practical. Integration of the two systems could lead to the control of additional variables and eventually to a truly self-optimizing self-correcting machine tool.

## BIBLIOGRAPHY

- Childs, J. J., *Principles of Numerical Control*, Industrial Press, New York, 1969.  
Dyke, R. M., *Numerical Control*, Prentice-Hall, Englewood Cliffs, N.J., 1967.  
*IBM Application Program H20-0309-3, System/360 APT Numerical Control Processor (360A-CN-10X)*  
Version 4, Part Programming Manual, 1969.  
IIT Research Institute, *APT Part Programming*, McGraw-Hill, New York, 1967.  
Leslie, W. H. P., *Numerical Control User's Handbook*, McGraw-Hill, Maidenhead, Berkshire, England, 1970.  
Modern Machine Shop, *NC Guidebook & Directory*, Gardner, Cincinnati, 1972.  
Olestren, N. O., *Numerical Control*, Wiley-Interscience, New York, 1970.  
Roberts, A. D. and R. C. Prentice, *Programming for Numerical Control Machines*, McGraw-Hill, New York, 1968.  
Society of Manufacturing Engineers, *Introduction to Numerical Control in Manufacturing*, Prentice-Hall, Englewood Cliffs, N.J., 1969.  
Thornhill, R. B., *Engineering Graphics and Numerical Control*, McGraw-Hill, New York, 1967.

J. L. Goodrich and Robert D. Thrush

# AUTOMATION

In the business world, the words "automation" and "computers" are often used synonymously, while in manufacturing the use of computers as the central element in automation systems is only now becoming dominant. Automation is defined by Webster as the quality of being self-acting or self-regulating, particularly as applied to machinery or mechanical devices, and derives from the Greek *automatose*, self-acting. The key element in understanding automation systems is the concept of control, and automation technology may be defined as any device or system that replaces or augments man in the sense of control. The meaning of automation, and the importance of the use of computers in automation, can best be seen by examining the history of the industrial revolution.

## HISTORICAL PERSPECTIVE

Table 1 shows the key steps in the industrial revolution. The first two steps, division of labor and application of mechanical power to production processes, resulted in the formation of factories during the latter half of the eighteenth century as the basic unit of production. The development of standardized parts, made possible by precision machine tools during the first part of the nineteenth century, made possible the assembly of parts produced in many factories and the field repair of complex machines using standardized spare parts. The development of transfer systems, the best known example of which is the assembly line in automobile manufacturing, first occurred during the last part of the nineteenth century and resulted in another significant increase in manufacturing productivity. By World War I, open loop mechanical

Table 1.

Steps in Industrial Revolution

Division of labor
Mechanical power
Standardized parts
Transfer devices
Open loop control
Closed loop control
Computer control

control devices, such as cams, gears, and levers, were well established in continuously operating machinery. Open loop control is still often used in such applications as numerically controlled machine tools. Closed loop devices appeared approximately

during World War II, and the simple control loop, based on a sensor, differential amplifier, and actuator to control the physical variable being sensed, is the basis of most industrial control techniques, particularly the continuous batch processing industries such as chemical processing and oil refining. The final stage in the industrial revolution, the introduction of computers as control devices, will produce fundamental changes in the concepts of automation in industrial and service area applications that will have significant impacts on our economy during the next few decades.

## AUTOMATION AND SOCIAL IMPLICATIONS

Automation is a term most often applied to the last two stages of the industrial revolution. The term first appeared in 1935 in the use of D. S. Harder in connection with automobile manufacturing. The term has often been equated with cybernetics, which derives from the Greek word for steersman and has the connotations of steering or control and is used synonymously with automation and computer technology by the Russians, but has had a rather esoteric, theoretical connotation in this country. Cybernetics was popularized by Norbert Wiener in 1948 in a book by that name. Dr. Wiener predicted mass unemployment to follow the widespread application of automatic machines, and this and other similar forecasts led to what has been called the "Automation Hysteria" of the 1950s. The mass unemployment has not materialized, and in fact the President's Commission on Technology, Automation and the Economy in 1966 absolved automation from the responsibility of unemployment and instead pointed out that increased productivity was directly responsible for our high standard of living and that increased productivity was not incompatible with full employment. The emotional connotations of the word still remain although a conference on Pattern Information Processing Systems held by the Engineering Foundation in 1972 pointed out that "automation technology replacing people was yesterday's concern; quality of job and product is today's issue." This is essentially the same conclusion reached by the President's Commission in 1966.

This technology does have the potential for widespread impact on both our economy and our social institutions, and continued study and assessment of the implications of this technology should be carried out by the federal government.

## APPLICATIONS OF AUTOMATION

The concepts and applications, both current and potential, of automation are very wide ranging in a technological economy as advanced as ours. Some specific examples are indicated here, and detailed discussions of computer automation systems appear elsewhere in this Encyclopedia. Examples:

Automation applied for improved safety in dangerous or hazardous occupations such as underground tunneling and mining, fire fighting, remote operations—space, undersea, underground—and law enforcement.

Automation applied for improved public protection and safety in such instances as banks, homes and apartments, and computer data banks.

Automation applied for quality control of essential public services such as clinical testing and measurement, environmental measurements, mass education and educational testing, medical board examinations, individual self-paced education, and testing.

Automation applied to tasks deemed boring and tedious and so produce *better quality products with better use of manpower* such as mass production tasks, clerical/arithmetical tasks, inventory control, and inspection operations.

Automation applied to ease the conversion to the metric system by converting numerically controlled tool industries, handbooks, engineering drawings, and product description efforts.

Automation applied to produce better quality services at less cost through automated merchandise identification, automated customer identification, automated inventory management, material handling, inspection operations, patient monitoring—EKG, coronary care units, etc.—patient diagnosis, and computer-aided instruction.

Automation applied to complex information processing tasks such as problem solving, simulation and modeling, pattern recognition—photo interpretation, molecular structure—replacement for humans in hazardous or remote environments, and processing of routine paperwork involved in information services such as banking, insurance, government, and law.

## A COHERENT VIEW OF AUTOMATION TECHNOLOGY

One commonality of automation systems is in the basic concept: augmenting or replacing people in a control sense. Analyzing the sensory, control, and effector capabilities and limitations of human beings gives a useful indication of the capabilities needed by general purpose automation systems.

A human being operates on a multilevel control system. To locate and pick up an object requires vision, touch, and kinesthetic feedback in controlling some 27 degrees of freedom in the arm and hand. The kinesthetic control of muscle contraction has relatively low accuracy and may be implemented reflexively when temperature or pain sensors are actuated or when super reflexes, such as walking or grasping, are used. A higher level of control occurs when touch sensors or vision sensors are brought into play. These complex sensors are capable of analyzing multiple inputs or images to give finer control of the basic servo loops in the muscles. Finally, the mind is capable of monitoring the central nervous system, correlating various sensory inputs and control signals, and changing the control program as needed.

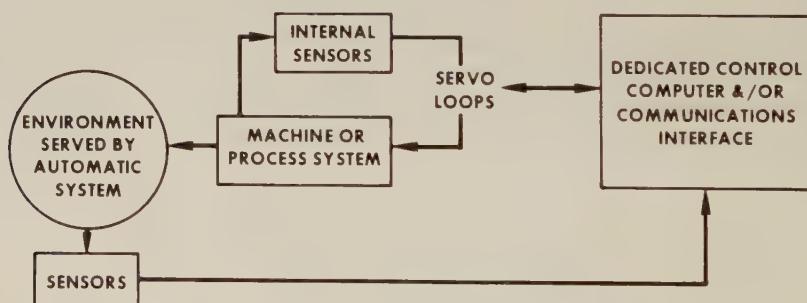
Real-time adaptive control and decision-making capability such as those described above are now possible with control systems using a computer or its derivatives instead of a person. Most importantly, the control program can be changed very quickly, allowing the concept of general purpose automation systems. Since the control programs are in software, and can be quickly changed, this concept is called soft

automation. Hard automation, in which the control program is fixed with electronic, electrical, or mechanical control devices, underlies the special purpose automation systems that dominate in American industry today.

It is essential to distinguish between hard and soft automation systems to avoid confusion based on preconceived ideas of automation. Soft automation systems are just appearing and offer possibilities in price and performance that are impossible with hard automation. For example, using direct numerical control (DNC) of machine tools, a mechanical system of low accuracy and high precision can be calibrated and control programs corrected to give a machining accuracy equivalent to the precision or repeatability of the machine. Since much of the cost of a several hundred thousand dollar machine tool is in attaining the mechanical accuracy of the system, this is a significant possibility.

General purpose automation systems can have the low costs of volume production, ease of debugging and programming of an established product, and can be depreciated over the life of the machine tool, not the product being produced. Further, increases of up to an order of magnitude are reported for N/C machine tools and variable part production systems, and even more significant gains may be possible when total production systems and labor intensive services are automated.

A framework for discussing general purpose automation systems is diagrammed in Fig. 1. The technologies embodied here are all application independent. In manufacturing, this schematic could represent continuous process control (e.g., petroleum refining), DNC machine tools, automatic inspection machines for quality control, stacker cranes, material transfer machines or industrial robots, and automatic assembly machines. Only the first four of these are developed in industry to any degree. Industrial robots operate with no complex sensors, essentially blind, and there are no general purpose automatic assembly machines. Even in the first three categories there is need for much more work: N/C machine tools operate with open loop mechanical systems, and the Numerical Control Society has formulated an experiment to integrate automatic quality control with DNC manufacturing.



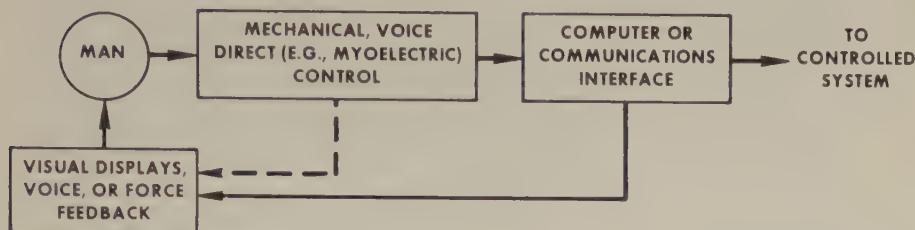
**Fig. 1. General automation system.**

In the service areas, remote diagnosis for maintenance and repair or for medical applications embodies the same sensor and control concepts, and automatic remote

branch banks, automated garbage collection, street cleaning, grounds keeping, and janitorial machines would all be the same concept.

New possibilities appear in servo position sensors and control, in complex sensors, and in operating systems software.

The man/machine interface for exercising supervisory control or man-in-the-loop control of such a generalized system is shown in Fig. 2.



**Fig. 2.** Man/machine interface.

The dominant technique for communicating with a computer is through a standard typewriter keyboard in a Teletype or CRT terminal although more sophisticated interactive modes are used: for example, in interactive graphics for computer-aided design. Control interfaces and feedback systems for man-and-the-loop control of remote machine systems is still in a very rudimentary stage, despite the work done by the AEC and NASA in remote manipulators. Problems in touch sensors, force feedback, visual bandwidth compression, and in different basic control modalities and the translations between them, are the subject of experimental work in different laboratories throughout the country. The state-of-the-art is just now reaching the point of capturing interest of several agencies in such applications as remote planetary explorations, undersea exploration, remote fire fighting, remote mine rescue, and handling of dangerous objects such as explosives and radioactive materials.

An integration of these two subsystems into a total supervisory control or man-and-the-loop control system is shown in Fig. 3. For full-time man-and-the-loop control, such as in fire fighting and mine rescue, the computer system would probably be replaced by computer interfaces and communications processing systems unless special augmented capabilities or control mode translations were required that would use a computer.

A total automation system, such as might occur in a factory, a bank, a hospital, or a large retail distribution center, might look like this. It is the integration of various subsystems into such a total system that eventually offers the largest payoff in terms of increased productivity and cost reductions. By the same token, the largest problems exist at this level of integration. One large company has a plant with an automatic manufacturing system, automatic inspection and quality control systems for both incoming material and finished products, and an automated management and accounting system; all these subsystems are implemented on computers, none of which can communicate with each other. The outputs of one computer program must be retrieved

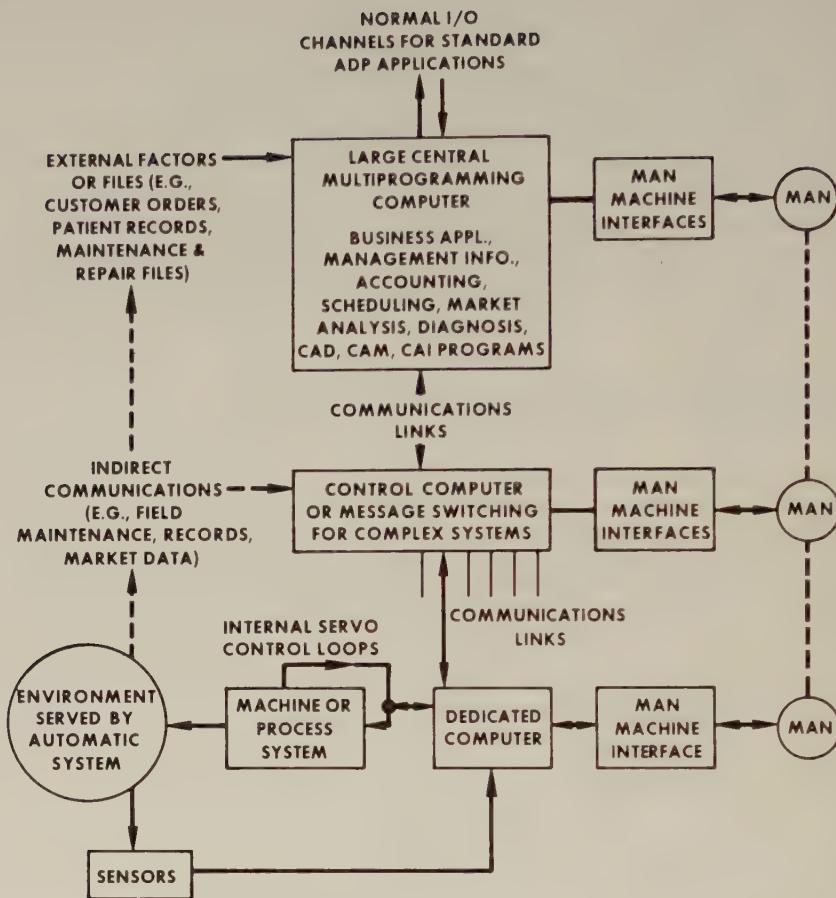


Fig. 3 Generalized total automation systems.

by a man and reformatted and reentered into a computer again. To avoid such problems as this on a greatly magnified scale when companies try to integrate subsystems manufactured by tens or even hundreds of other companies, such as might occur in highly automated factories and services in the 1980s, a framework of guidelines and standards must be set up now by the federal government, working with the private sector, to provide an institutional structure within which subsystems can be produced by different companies for maximum efficiency of the total system. The use of performance standards and standardization only of communications interfaces will avoid limiting the development of new technology.

## IMPLICATIONS OF THE NEW CONCEPTS OF AUTOMATION

Considering automation from the point of view of control, the development of control systems has moved from mechanical cams, gears, levers, etc. to hydraulic, pneumatic, electrical, electronic, and finally the computer and its derivatives. This final step allows:

Complex sensory processing.

Correlation of data inputs and control of interrelated actuators.

Adaptive control programs for on-line optimization.

The ability to process symbolic data, such as the English language.

Quick changes of control programs: the concept of general purpose automation.

The last concept has several implications. Once control programs have been written and debugged, they may be rapidly interchanged, allowing mechanical systems that have a multitude of uses within the general class of tasks defined by the mechanical system itself. This leads to the idea of less expensive machine systems that can be depreciated over the life of the machine rather than from inventories of spare parts and, most importantly, the concept of custom design at mass production cost, a trend which has appeared more and more recently in a consumer-oriented society and which can be fully met by the new automation systems. This will change not only manufacturing but will fundamentally alter wholesale and retail distribution systems in the service areas of the economy.

An example of such flexibility is the laser suit cutter made by Hughes for Genesco. A computer controls a laser beam to cut out patterns for men's suits from a single layer of cloth that moves past the cutting beam. A man will cut 50 layers of cloth simultaneously with one pattern and a band saw, which results in 50 suits of one pattern. The computer-controlled laser works with only one layer of cloth but at 50 times the speed. Despite the fact that there is not net increase in output, with the computer-controlled system one suit can be a 48L and the next a 32S, reducing inventory of odd sizes and, in fact, once an individual's body measurements are on file in the computer, the suits can indeed be custom designed. The retail distributors have become quite excited about the possibilities of such production systems.

The ability to change control programs will modify the nature of our basic manufacturing industries; the ability of computers to process symbolic data may have an even more revolutionary impact on the entire economy, particularly the service areas. Many services are involved with processing information and might be characterized as software services: law, insurance, finance, real estate, education, and government all depend on processing information as their basic objective. Medicine, wholesale and retail trade, and maintenance and repair services also depend on information (e.g., for diagnosis) although not so thoroughly as the services in the previous list.

Computers can process words as well as data. Current abilities generally treat words as abstract symbols, as in bibliographic or information system search and retrieval systems or in word processing systems. The last concept, as promoted by

IBM, conceives of automated or semiautomated systems devoted to processing words as opposed to data: the MT/ST and MT/SC typewriters, Automated Typing Systems (ATS) using time-shared computers, and IBM has developed a prototype-newspaper publishing system for the Japanese in which the first hard copy is the final output from the on-line phototype setter. Current artificial intelligence research includes attempts at writing programs that can interpret the meaning of entire sentences, as opposed to treating words as independent symbols, and this will result in automation systems able to cope with the growing problems of the information-oriented services.

## AUTOMATION IN FOREIGN COUNTRIES

Other industrialized countries besides the United States are developing and using computer-based automation systems. For example, the Russians have installed the world's first "self-learning" adaptive control system in a steel pipe production plant. This control system reduced wall thickness variations by 20 to 25%, allowing more stringent designs and thus saving more than 255,000 rubles (\$308K) annually. The system will pay for itself in less than a year.

The Japanese are automating everything they can under the pressure of a declining work force: newspaper publishing, banking, ship building, manufacturing. There are fewer man hours in a ton of Japanese steel than there are in a ton of American steel; Sony is building a plant on the West Coast of the United States to compete with United States industry with United States labor.

## CONCLUSIONS

The implications of this technology may be summarized as follows:

1. Custom design at mass production costs, implying reduced inventories, ease of introduction of new technology, the automation of short production runs in small businesses, and fundamental changes in distribution services.
2. Increased productivity and reduced cost: increases in productivity of up to an order of magnitude in integrated automation systems due to the efficient scheduling and control and to the inherent speed of computers.
3. Alteration of repair strategies: the production of spare parts as needed, and the availability of built-in testing or remote testing (for example, over phone lines) of consumer appliances such as TV sets. This technology has been developed by NASA and the military for use in systems where direct test and repair is not feasible, such as in satellites or undersea systems, and the first civilian application has now appeared in the Volkswagen computerized testing system.

4. Processing of words, both as symbolic data and by meaning, to allow automation of information-oriented service industries.

To achieve these benefits, particularly in the service areas in such applications as medicine, education, and government services, will require overcoming major obstacles in technology, in software development, and in standardization of both software and hardware interfaces to achieve the integration of components into highly productive total systems such as factories, hospitals, retail stores, and schools. Basic institutional changes may also be required, such as the division of labor in service applications, before the widespread application of automation technology becomes economically feasible.

The computer as the basic control element in automation systems does represent the next fundamental step, possibly the last major step, in the industrial revolution, and the widespread application of this technology will result in fundamental changes in our industrial base in both character and productive capability and will also allow automation of service area tasks for the first time. The computer, then, offers us both an unparallelled opportunity and a challenge to implement this technology for the benefit of all while solving the potential social problems associated with that implementation.

*John M. Evans, Jr.*

## BABBAGE, CHARLES

Charles Babbage, the inventor of the first automatic computer, was born on December 26, 1791<sup>1</sup> in Totnes, Devonshire. His father, Benjamin Babbage, a banker, and his mother néé Betty Plumleigh Teape, also came from Totnes. Later the family moved to London.

As a child Charles Babbage was very delicate and a greater part of his education was conducted by private tutors at home. In adolescence his health became stronger and he was sent to the Academy of the Rev. Stephan Freeman at Forty Hill, Enfield, Middlesex for 3 years. In October 1810 Babbage was admitted to Trinity College, Cambridge, but later on moved to Peterhouse. He graduated in mathematics in 1814. In Cambridge Babbage had a lot of friends, such as John Herschel (later Sir John) and George Peacock (later Dean of Ely) with whom he founded the so-called Analytical Society. This society played an important part and had an impact on the development

<sup>1</sup> Many sources, even some of the old issues of *British Encyclopaedia*, indicate 1792 by mistake as the year when Babbage was born.



Fig. 1. Charles Babbage (1791–1871).

of mathematics in England. The purpose of the society was to popularize modern analysis, and the first step in this respect was the translation of the S. F. Lacroix's *Differential and Integral Calculus* [I] from French. The founders of the society rented a special office where they held meetings regularly, read papers, and discussed. The society even succeeded in publishing a volume of *Transactions*, where, due to various reasons, only works of Babbage and Herschel were published. Lacroix's book was also translated and published by Babbage with the help of Herschel and Peacock.

On graduating from Cambridge, Charles Babbage settled in London and spent the rest of his life there.

On March 14, 1816 Babbage was elected member of the Royal Society at the suggestion of the eminent astronomer William Herschel, his son John Herschel, Peter Roget, and others.

During the summer of 1828, while touring Italy, Babbage read a short announcement from Cambridge made in the newspaper *Galignani*: "Yesterday the bells of St. Mary's rang on the election of Mr. Babbage as Lucasian Professor of Mathematics." Later Babbage considered the appointment to this chair, once held by Isaac Newton, as the only honor which he had ever received in his own country. During the same time Babbage had applied for Secretary of the Royal Society, a post once held by his friend John Herschel, but his demands were declined. In a very kind letter, Sir Robert South

wrote to Babbage that if he was not to have a post occupied once by Halley, he may shortly enjoy the greater one of filling a chair once held by Newton [2].

The great popularity of Babbage is due undoubtedly to his pioneer work on the invention of automatic computers, but one must immediately point out that he had dealt with a number of other things and was a man of many interests.

Babbage himself described in a joke in his autobiography how the thought to create a machine which would automatically calculate and print tables originated: this happened in 1812 or 1813 while he was still a student of mathematics in Cambridge. His whole life from that moment on was connected with the construction, perfection, and attempts to build this machine. Before any further detailed consideration of the ideas of Babbage in this field, one should quote Prof. M. V. Wilkes, one of the veterans of the contemporary computer technique [3]:

In writing on Babbage as a computer pioneer, one must at once admit that his work, however brilliant and original, was without influence on the modern development of computers. The principles that Babbage elucidated, but regrettably failed to communicate, had to be rediscovered by the men, who, 100 years later, built the first automatic computers. The ultimate loss was not perhaps in itself a great one, since, as soon as progress of a practical sort was achieved, the subject quickly developed far beyond the point to which Babbage had taken it.

There are many reasons for this failure, but perhaps the most substantial one has been pointed out very successfully by H. H. Goldstine [4].

It is curious how delicate are the timing and balance of what we now refer to as research and development. There seems to be an optimal time for discovery, as well as an optimal period for perfection of the idea. Thus, if one is late in starting or dilatory in carrying out the task, one may, as Sir John Couch Adams almost did in connection with the discovery of Neptune, find oneself "scooped." If on the other hand, one undertakes the task of finding or inventing something too soon before this optimal point, one needs to expend too much energy and therefore frequently fails. An outstanding example of this is probably Babbage.

One has to point out that Babbage had invented two computers which differed considerably one from the other. The first one, called a "differences engine," was designed to draw up tables of functions according to the method of the differences. The idea about the second machine originated later. It is indeed the forerunner of an universal automatic computer, and Babbage referred to it as the "analytical engine."

The story about the building of the differences engine (Fig. 2) was a very dramatic one, and the whole of the eleventh chapter of Weld's *History of the Royal Society* [5] is dedicated to it:

On the 1st April, 1823, a letter was received from the Treasury, requesting the Council to take into consideration a plan which had been submitted to Government by Mr. Babbage for "applying machinery to the purposes of calculating and printing mathematical tables;" and the Lords of the Treasury further desired "to be favoured with the opinion of the Royal Society on the merits and utility of this invention."

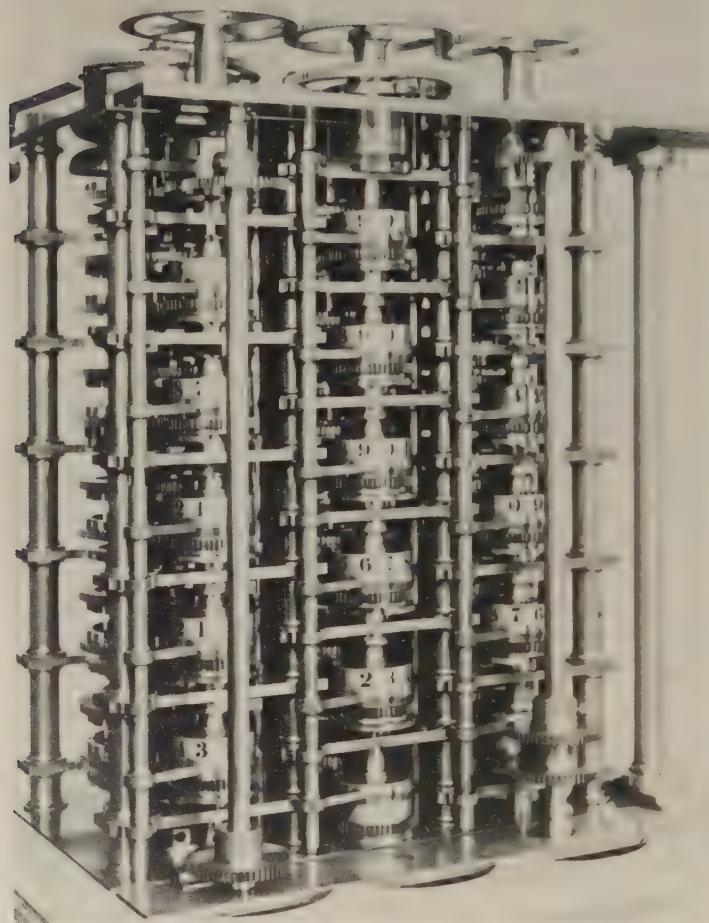


Fig. 2. Portion of the difference engine.

This is the earliest allusion to the celebrated calculating engine of Mr. Babbage in the records of the society. But the invention had been brought before them in the previous year by a letter from Mr. Babbage to Sir H. Davy, dated July 3, 1822, in which he gives some account of a small model of his engine for calculating differences, which "produces figures at the rate of 44 a minute, and performed with rapidity and precision all those calculations for which it was designed" [5].

A committee headed by Sir H. Davy reached a favorable decision in support of the project. After a meeting with the secretary of the exchequer, Babbage was given £1500 to construct the machine provided that it would be a state property. An intensive work began to make detailed drawings, to produce the necessary tools, and to instruct craftsmen to execute the construction. "The mechanical department was placed

under the management of Mr. Clement, a draughtsman of great ability and the practical mechanic of the highest order” [5].

On demand of the government in 1828, Babbage delivered a report about the state of affairs which read [5, p. 375]:

The machine has required a longer time and greater expense than was anticipated, and Mr. Babbage has already expended about £6000 on this object. The work is now in a state of considerable forwardness, numerous and large drawings of it have been made, and much of the mechanism has been executed, and many workmen are occupied in its completion.

Another committee was appointed to inspect the state of affairs of the machine and after long discussions, numerous letters, and reports, Babbage was given the sum of £7192/4/8 by the government to pay the expenses that had already been made and another £600 in advance to continue the work on the machine. But the committee recommended that a special fireproof building should be built to safeguard the drawings and the machine. Babbage suggested that the building should be built in the yard of his house.

In 1831 the new buildings were built and £17,000 had already been spent, but an unforeseen trouble stopped the work. Mr. Clement, whom Babbage appointed to superintend the mechanical department, set different demands as a compensation for the mounting of the machine in a new building. These demands were not amended for in compliance with the decision of the state bodies and Mr. Clement retired, taking the drawings and the machinery which had been created to produce the special parts of the machine with him. Clement was offered a sum of money to give back what he had taken, but he refused and thus the work for the building of the machine ceased.

During that time Babbage was deprived of the opportunity to use his own drawings of the differences engine. While considering the general principles of operation of the machine, he came to a completely new idea of an automatic computer, that of the analytical engine (Fig. 3). In May 1835 he wrote in a letter:

I am myself astonished at the power I have been enabled to give to this machine, a year ago I should not have believed this result possible. This machine is intended to contain a hundred variables, or numbers susceptible of changing, and each of these numbers may consist of twenty-five figures. The greatest difficulties of the invention have already been surmounted and the plans will be finished in a few months.

During one of his frequent visits to Italy in 1841, Babbage described the plan of his analytical engine to the most eminent Italian mathematicians and engineers in great detail. M. Menabrea, a military engineer who was present at the meeting, received permission from Babbage to publish a description of the machine [6]. During the following year this description was translated into English by Lady Lovelace, Lord Byron’s daughter and an admirer of Babbage. Ada Augusta Lovelace’s translation was accompanied by a vast commentary, whose volume and importance rather exceeded the very translation itself. Judging by the correspondence between Ada and Charles, one can easily notice that the latter had taken an active part in the edition of these

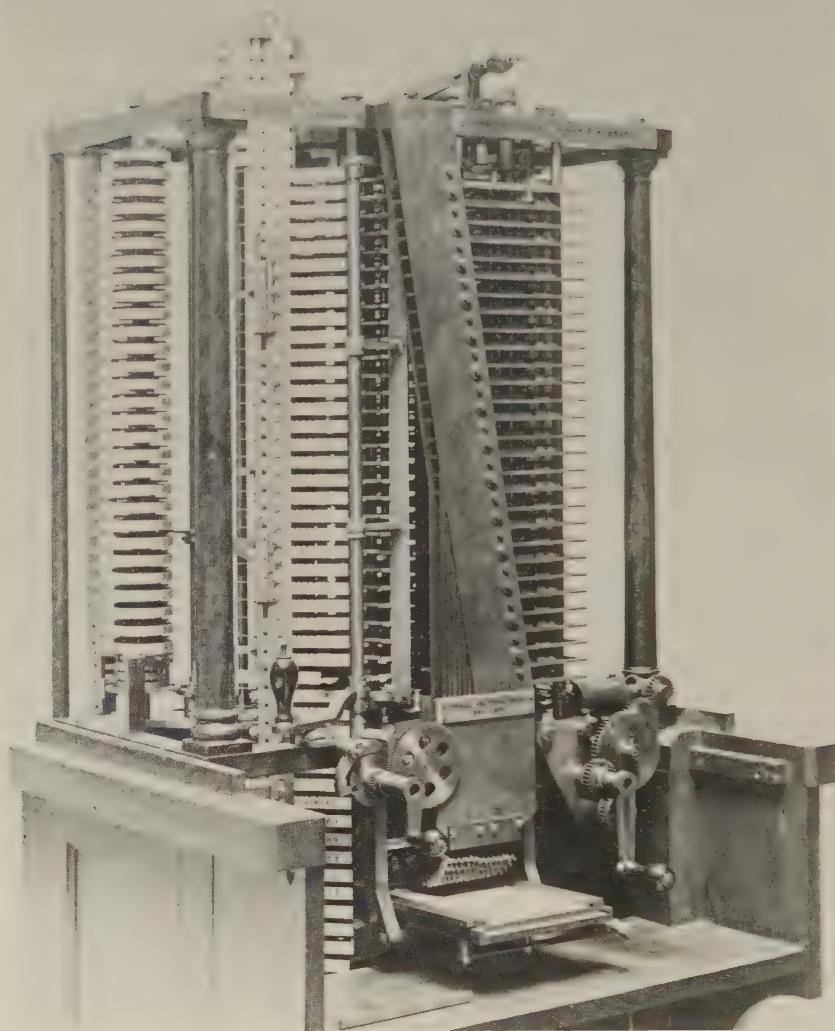


Fig. 3. Portion of the analytical engine.

commentaries. A very famous quotation of Ada Augusta's commentaries says: "We may say most aptly, that the Analytical Engine weaves algebraical pattern just as Jacquard loom weaves flowers and leaves."

The great enthusiasm originating from the project of the second machine did not help its construction and execution. It was in this connection that Babbage wrote in his autobiography the following prophetic words:

The great principle on which the Analytical Engine rests have been examined, admitted, recorded and demonstrated. The mechanism itself has now been reduced to

unexpected simplicity. Half a century may probably elapse before any one without those aid which I leave behind me, will attempt so unpromising task. If, unwarmed by my example, any man shall undertake and shall succeed in really constructing an engine embodying in itself the whole of the executive department of mathematical analysis upon different principles or by simpler mechanical means, I have no fear of leaving my reputation in his charge, for he alone will be fully able to appreciate the nature of my efforts and the value of their results.

Despite the great variety of interests that Charles Babbage had, he was a mathematician most of all. His interest in this subject developed while he was still at school reading *Ward's Young Mathematicians Guide*. He was extremely keen on mathematics in Cambridge and a fervent fighter for the modernization of mathematics in Great Britain. Babbage's original works in mathematics were rather various. A part of them were devoted to the calculation of functional equations. Perhaps one of his most interesting works in mathematics [7] of these series was the study of the functional equation

$$\psi^n x = x$$

which at present is expressed as

$$\psi(\psi(\dots\psi(x)\dots)) = x$$

In the special case of  $n = 2$ , Babbage found all possible solutions of the equation

$$\psi(\psi(x)) = x$$

It seems that Babbage himself assessed this result highly since the marks of the sealing wax of his private letters consisted of the symbols  $\psi^2 x = x$ . These works of Babbage's perhaps have been forgotten, and in 1969 the problem for the solution of the functional equation  $\psi(\psi(x)) = x$  was published as an open one (*Amer. Math. Monthly*, Problem 5530).

Charles Babbage dealt with many other functional equations originating from geometrical problems.

A theorem of the theory of numbers which is analogous to the famous theorem of Wilson for divisibility of  $((n - 1)! + 1)$  by  $n$  belongs to Babbage. It is as follows: The number

$$\frac{(n + 1)(n + 2)\dots(2n - 1)}{(n - 1)!} - 1$$

can be divided by  $n^2$  only when  $n$  is prime [8].

Babbage was interested in the problems of summing of infinite divergent series and explicit solving of recurrent equations. The latter is closely connected with the calculation of tables and the use of computers. He created a method for summing of series which he called "Method of expanding horizontally and summing vertically" [9]. It

consisted of the representation of a series as a double series and a change of the order of summing. Naturally the use of this method for summing of divergent series of the type

$$\sum_{k=1}^{\infty} (-1)^{k+1} k^{2s}$$

which Babbage treated took him to most unexpected results. He tried in vain to explain them.

Babbage examined a number of problems connected with games of chance [10]. A most typical example of one of them is the following:

A gamester began a series of bets on an event whose chance of occurring is one-half, by staking the sum  $U$ . Whenever he wins he makes the next succeeding stake less than his last by the quantity  $v$ , but if he loses, he then increases his stake by the same quantity. Supposing he should win  $p$  times, and lose  $q$  times, what will he have gained or lost on the whole number  $p + q$ ?

Deciphering was closely associated with the rest of Babbage's interests. "Deciphering is, in my opinion, one of the most fascinating of arts, and I fear I have wasted upon it more time than it deserves" [11]. Babbage devoted a lot of time on compiling dictionaries where the words were spelt with equal numbers of letters and had features suggesting the equal number of letters in them. The purpose was to use them in deciphering. Babbage aimed at systematization of the methods of deciphering which is called today cryptanalysis. "I am myself inclined to think that deciphering is an affair of time, ingenuity and patience; and that very few ciphers are worth the trouble of unravelling them" [11].

Babbage made an enormous contribution to the edition of mathematical tables [12]. It is curious to know that he himself checked all the logarithmic tables edited by him, and thus he found a lot of similar mistakes in the tables edited before [13].

Babbage had an original approach and possessed the ingenuity of an explorer. An experiment connected with the convenience of the use of the tables is described in Ref. 14. According to Babbage himself, "the object of the experiment for which those volumes were printed, was to ascertain the colour of the inks and the tints of papers least fatiguing to the eye. For that purpose one hundred and forty differently coloured papers were chosen and ten different colours of ink were employed." This experiment, which is of an ergonomical character, was connected with Babbage's project for a machine that would calculate and print mathematical tables all by itself.

An invention of Babbage's concerning lighthouses, which was met with an enormous interest but not in England, contains the idea of self-correcting code [15]. It is as follows:

Make each lighthouse repeat its own number continually during the whole time it is lighted. . . .

Lighthouses must not be numbered in the order of their position. But every lighthouse must have such a number assigned to it, that no digit occurring in the number denoting

the several lighthouses nearest to it on either side shall have the same digit in the same place of figures.

Further on there is a description with numerous examples of how to detect and correct mistakes in such a numbering.

Another invention of Babbage's, reinvented by Helmholtz later, was the ophthalmoscope. In the report of the Professor of Ophthalmalic Surgery in University College, London, T. W. Jones [16], one can read

Dr. Helmholtz of Konigsberg, has the merit of specially inventing the ophthalmoscope. It is but justice that I should here state, however, that seven years ago Mr. Babbage showed me the model of an instrument which he had contrived for the purpose of looking into the interior of the eye. . . . This ophthalmoscope of Mr. Babbage, we shall see, is in principle essentially the same as those of Epkens and Donders, of Coccins and of Meyerstein, which themselves are modifications of Helmholtz's.

It is difficult to numerate here all the topics and problems which excited Babbage and the contribution he made to the various fields. His book, *On the Economy of Machinery and Manufacturers* [17], is an early example of operations research. There are fundamental works of Babbage in the field of statistics and insurance [18]. Besides everything else, he was one of the founders of the Statistical Society, and was regarded and treated as the father of that society.

Babbage was infatuated by experimental work on magnetic induction and published a number of papers in this field [19].

Some of his publications were dedicated to geological problems [20, 21]. It was Babbage who long before Charles Darwin expressed an opinion that the crust of the earth moved and could go up and down at some spots.

Babbage was keen on astronomy, linguistics, and many other fields of human knowledge. He was one of the ardent zealots of scientific cooperation between different nations, and he popularized the organization of scientific congresses and meetings of scientists of different countries.

The most detailed source about the life of Babbage is his excellent autobiography [11]. It is worth quoting at least the first sentence of this book: "What is there in a name? It is merely an empty basket until you put something into it." There is no doubt whatsoever that Babbage put a lot in this basket and his name acquired an important meaning.

One can learn a lot about Babbage's life by Maboth Moseley's biographical novel, which came as a result of considerable study of original documents [22].

The long life of Charles Babbage, full of so much hope and bitter disappointment, came to an end on October 18, 1871. He was buried in Kensal Green Cemetery, London (Sect. 84, Grave No 23003). The memoirs of his friend L. A. Tollemache [23] read:

I heard him say more than once that he would gladly give up this remainder of his life if he could be allowed to live three days 500 years hence and might be provided with scientific cicerone who should explain to him the discoveries that had been made since

his death. He judged that the progress to be recorded would be immense, for, as he said, science tends to go on, not merely with a great, but with a constantly increasing rapidity.

## REFERENCES

1. S. F. Lacroix, *An Elementary Treatise on the Differential and Integral Calculus*. Translated from French. Chapter 1 by C. Babbage, Chap. 2 by G. Peacock and J. F. W. Herschel. With an appendix (by Herschel) and notes (by Peacock and Herschel), Cambridge, 1816, 720 pp.
2. L. H. D. Buxton, Charles Babbage and his difference engines, *Trans. Newcomen. Soc.* (London) **14**, 43–65 (1935).
3. M. V. Wilkes, *Babbage as a Computer Pioneer*, The Babbage Memorial Meeting, October 18, 1971, Savoy Place, London WCZ, British Computer Society, Royal Statistical Society, 1971.
4. H. H. Goldstine, *The Computer from Pascal to von Neumann*, Princeton University Press, Princeton, N.J., 1972.
5. C. R. Weld, *A History of the Royal Society, with Memoirs of the Presidents, etc.*, 2 vols. London, 1848.
6. L. F. Menabrea, Sketch of the analytical engine invented by Charles Babbage Esq. (From the *Bibliothèque Universelle de Génève*, No. 82, October 1842), *Sci. Memo.* (London) **3**, 11, 666–731 (1843). With notes upon the memoir by the translator, Ada Augusta Countess of Lovelace.
7. C. Babbage, An essay towards the calculus of functions, Part II, *Phil. Trans. Roy. Soc. London* **106** 2, 176–256 (1816). See also *Abstracts of the papers printed in the Philosophical Transactions of the Royal Society of London*, Vol 2, 1832, p. 41.
8. C. Babbage, Demonstration of a theorem relating to prime numbers, *Edinburgh Phil. J.* **1**, 46–49 (1819).
9. C. Babbage, On some new methods of investigating the sums of several classes of infinite series, *Phil. Trans. Roy. Soc. London* **109**, 2, 249–282 (1819).
10. C. Babbage, An examination of some questions connected with games of chance, *Trans. Roy. Soc. Edinburgh* **9**, 1, 153–177 (1823).
11. C. Babbage, *Passages from the Life of a Philosopher*, London, Longman, London, 1864, xii + 492 pp.
12. C. Babbage, *Table of the Logarithms of the Natural Numbers from 1 to 108000*, J. Mawman, London, 1827, xx + 202 pp.; John Murray, London, 1844; E. & F. N. Spon, London, 1915.
13. C. Babbage, Notice respecting some errors common to many tables of logarithms, *Mem. Astron. Soc.* **3**, 65–67 (1827).
14. C. Babbage, Specimen of logarithmic tables printed with different coloured inks on variously coloured papers. In twenty one volumes, one copy only having been printed, London, 1831, *Edinburgh J. Sci. [NS]* **6**, 144–150 (1832).
15. C. Babbage, *Notes Respecting Lighthouses (Occulting Lights)*, Privately Printed, London, 1852, 24 pp.
16. T. W. Jones, Report on the ophthalmoscope, *Brit. and Foreign Med. Rev.* **14**, 549–557 (1854).
17. C. Babbage, *On the Economy of Machinery and Manufactures*, Charles Knight, London, 1832, xvi + 320 pp.
18. C. Babbage, *An Analysis of the Statistics of the Clearing-House During the Year 1839, with an Appendix on the London and New York Clearing Houses, and on the London Railway Clearing House*, John Murray, London, 1856, 33 pp. See also *J. Stat. Soc. (London)* **19**, 28–48 (1856).
19. C. Babbage and J. F. W. Herschel, Account of the repetition of M. Arago's experiments on the magnetism manifested by various substances during rotation—London. *Phil. Trans. Roy. Soc. London* **115**, 2, 467–496 (1825).
20. C. Babbage, Observations on the parallel roads of Glen Roy, *Quart. J. Geol. Soc.* **24**, 273–277 (1868).
21. C. Babbage, Observations on the Temple of Seracis at Pazzuoli, near Naples, with remarks on

- certain causes which may produce geological cycles of great extent, *Quart. J. Geol. Soc.* 3, 186-217 (1847).
22. M. Moseley, *Irascible Genius. A Life of Charles Babbage, Inventor*, Hutchinson, London, 1964, 288 pp.
  23. L. A. Tollemache and B. L. Tollemache, *Safe Studies*, Hodgeson & Son, London, 1884, viii + 429 pp.

*B. Sendov*

## BACKUS NORMAL FORM

See Specification Languages; Syntax Directed Methods

## BALAS' 0-1 PROGRAMMING ALGORITHM

See Bivalent Programming by Implicit Enumeration

## BIVALENT PROGRAMMING BY IMPLICIT ENUMERATION\*

### THE PROBLEM

A great many decision problems can be formulated as integer programs in bivalent (i.e., 0-1) variables. Such bivalent, or 0-1 programs serve as mathematical models for capital budgeting, project selection, pipeline or communications network design, structural design, switching circuit design, information retrieval, fault detection, design of experiments, facility location, industrial development planning, crew scheduling, truck dispatching, tanker routing, and a host of other scheduling problems. Mathematically, many of these problems can be associated with graphs or can be regarded as combinatorial optimization (combinatorial programming) problems.

The (pure, linear) 0-1 program can be stated as

$$\min\{cx \mid Ax \geq b, x_j = 0 \text{ or } 1, j \in N\} \quad (\text{P})$$

\* Note: The following article is being published out of alphabetical sequence due to a change of title.

where  $N = \{1, \dots, n\}$ ,  $A$  is an  $m$  by  $n$  matrix of integers, while  $b$  and  $c$  are  $m$ - and  $n$ -vectors of integers, respectively.

There are several methods for solving an integer program, of which (P) is a special case, in a finite number of steps. However, for many problems this finite number tends to become unacceptably large as soon as the number of variables exceeds 50 to 80. Thus, with the current state of the art, only part of the real world problems listed above can actually be solved; for some of them, one has to accept approximations; and for some, even that is too expensive to be practical. However, the field is only 15 years old. Improvements in algorithms come about every year, and these, together with improvements in computers and implementation techniques, lead to a steady increase in the size of problems that can practically be solved.

## IMPLICIT ENUMERATION

Implicit enumeration is an integer programming method specially designed for the 0-1 case. Its basic idea is to systematically enumerate a (hopefully small) subset of the set of  $2^n$  possible binary  $n$ -vectors while using the logical implications of the binary property to ensure that the whole set was implicitly examined; hence the name of the method. Though the best known early prototype of this approach is Balas's *additive algorithm* [1-3] (see also Ref. 4 for what is probably the best exposition to date), essential contributions have also been made by Glover [5], Geoffrion [6], and others [7-13]. The scope of this article goes beyond the original additive algorithm to give as complete and up-to-date an account of the state of the art as the limitations of space permit.

Implicit enumeration belongs to the same general class of enumerative, or search, or branch and bound methods, as the earlier, general purpose mixed-integer programming algorithm of Land and Doig [14]. Initially, the two approaches were sharply different. The Land and Doig algorithm is based on solving a linear program at every node of the search tree, whereas Balas' procedure uses logical tests requiring only additions and comparisons (hence the name "additive"). The search strategy of the Land and Doig algorithm is "breadth first," with the ensuing large storage requirements, whereas Balas's procedure follows a "depth first" strategy, with a modicum of storage requirements. However, successively improved versions of the two approaches have borrowed substantially from each other, and recent algorithms incorporate features of both: they use linear programming and logical tests, "depth first" in a more flexible version, as well as inequalities derived from the logical implications of binarity (see below).

## TERMINOLOGY AND NOTATION

We start by defining a few terms. Any binary  $n$ -vector is called a *solution*; one that satisfies  $Ax \geq b$  is called *feasible*; and a feasible solution minimizing  $cx$  is called

*optimal.* A *partial solution* is an assignment of binary values to a subset of the variables. The variables in the subset are called *fixed* (at 1 or 0, respectively), while the remaining variables are termed *free*. The *subproblem* associated with a partial solution is the one in the free variables. *Augmenting* a partial solution means fixing one or several free variables (at 1 or 0). A partial solution is a *descendant* of another partial solution if it can be obtained by augmenting the latter. A *completion* of a partial solution is a solution obtained from the latter by fixing all the free variables. The *zero completion* is the one in which all free variables are set to 0. *Fathoming* a partial solution (or the associated subproblem) means showing that it has no feasible completion better than the currently known best feasible solution (which may have been found in the fathoming process itself).

Implicit enumeration generates a sequence of partial solutions, starting with all variables free, by successively fixing variables. We will assume that  $c_j \geq 0, \forall j \in N$ , which implies no loss of generality, since any variable  $x_j$  such that  $c_j < 0$  can be replaced by  $x'_j = 1 - x_j$ , without changing the problem. The  $k$ th partial solution in the sequence is then denoted by the index set defining its fixed variables,  $J_k = J_k^1 \cup J_k^0$ , where  $J_k^1$  and  $J_k^0$  stand for the variables fixed at 1 and 0, respectively; whereas  $N_k = N - J_k$  denotes the set of free variables.

It is convenient to define  $J_k$  as an *ordered* set, so that it shows the history of its own generation; further, to have a positive element of  $J_k$  represent the index of a variable fixed at 1, and a negative one the index of a variable fixed at 0. An underline means that prior to its current binary value, the variable had been fixed at the complement of the latter. Thus

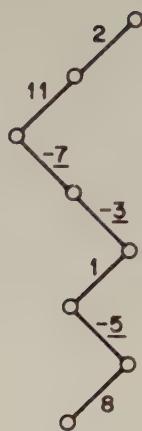
$$J_k = \{2, 11, -\underline{7}, -\underline{3}, 1, -\underline{5}, 8\}$$

denotes the partial solution

$$x_j = \begin{cases} 1 & j = 2, 11, 1, 8 \\ 0 & j = 7, 3, 5 \end{cases}$$

where the variables were fixed in the order in which they are listed, and where all the variables currently fixed at 0 had previously been fixed at 1.

A tree-representation of the enumeration procedure is helpful, where a node  $k$  corresponds to a partial solution  $J_k$ , and an arc  $(k, l)$  connects a pair of nodes corresponding to partial solutions  $J_k, J_l$  such that  $J_k^i \subseteq J_l^i, i = 1, 0$ , and  $|J_l| - |J_k| = 1$ . The arc  $(k, l)$  can then be associated with the variable  $x_j$ , where  $\{j\} = |J_l| - |J_k|$ , and assigned the symbol  $j$  if  $x_j$  is currently fixed at 1, or  $-j$  if  $x_j$  is currently fixed at 0. An underline means that the complement of the current value assignment need not be explored. Thus, at any stage of the search, the current search tree can also be viewed as representing the last partial solution. If the latter has  $i$  elements, we say that it is on the  $i$ th *level* of the search tree. A tree representation of the partial solution  $J_k$  of the above example is shown in Fig. 1.



**Fig. 1.** Tree representation of  $J_k$ .

## OUTLINE OF THE PROCEDURE

A schematic representation of implicit enumeration is shown in Fig. 2. We start with all variables free, i.e., with the partial solution  $J_0 = \emptyset$ . At the  $k$ th iteration, given a partial solution  $J_k$  and a currently known best feasible solution  $\hat{x}$  (if any), the logical implications of the value assignments defining  $J_k$  and of the binary nature of the free variables are explored through a number of tests which can have one of the following outcomes:

- (a) Nothing can be established.
- (b) It is shown that

$$x_j = \begin{cases} 1 & j \in Q^+ \\ 0 & j \in Q^- \end{cases}$$

for any feasible completion  $x$  of  $J_k$  such that  $cx < c\hat{x}$ , where  $Q^+$  and  $Q^-$  are arbitrary disjoint subsets of  $N_k$ .

- (c) A feasible completion  $\tilde{x}$  of  $J_k$  is found, such that  $c\tilde{x} < c\hat{x}$ .
- (d) It is shown that there exists no feasible completion of  $J_k$  better than  $\hat{x}$ .

In Case (a), we make a forward step by going to the next level of the search tree; i.e., we select a free variable  $x_j$  according to some choice rule, and fix  $x_j$  at 1 or 0. The choice criterion of Ref. 3 is to select (and fix at 1) a free variable  $x_{j_*}$  which, when set to 1, minimizes total infeasibility; i.e.,  $x_{j_*}$  such that

$$v_{j_*} = \min_{j \in N_k} v_j$$

where

$$v_j = \sum_{i \in M} \max \left\{ 0, b_i - \sum_{h \in J_k \cup \{j\}} a_{ih} - a_{ij} \right\}$$

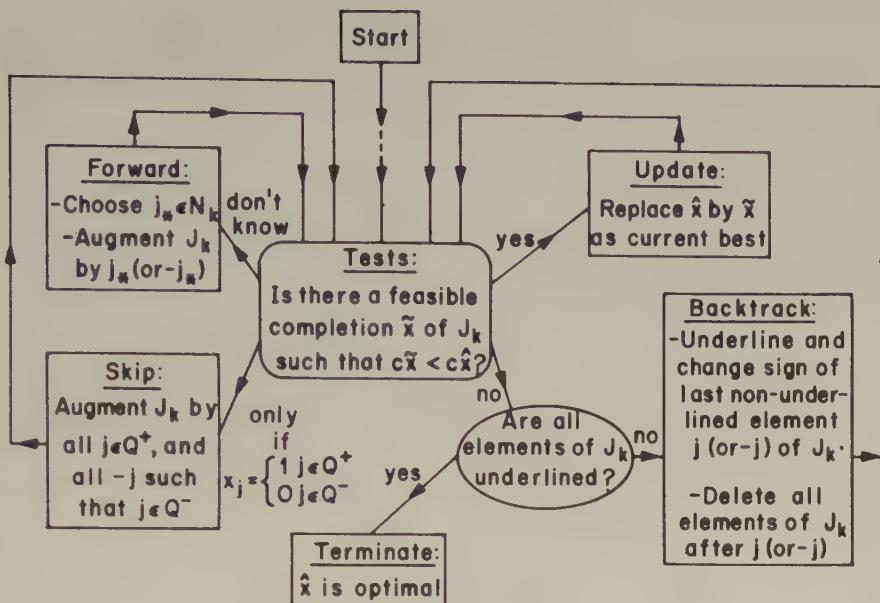


Fig. 2. Flow diagram of implicit enumeration.

In Case (b), we fix all  $j \in Q^+$  at 1 and all  $j \in Q^-$  at 0, which implies (unless  $|Q^+ \cup Q^-| = 1$ ) "skipping" some levels of the search tree; also, we underline all the elements of  $Q^+ \cup Q^-$  since they do not have to be complemented.

In Case (c),  $\tilde{x}$  replaces  $\hat{x}$  as the current best solution, and the partial solution is left unchanged.

Finally, in Case (d), if all elements of  $J_k$  are underlined, then all solutions have been explicitly or implicitly examined, and we are done: the currently known best feasible solution  $\hat{x}$  is optimal (or, if none has been found, none exists).

Otherwise, we backtrack to the nearest unfathomed node of the search tree; namely, we complement the variable  $x_j$  corresponding to the last nonunderlined element  $j$  (or  $-j$ ) of  $J_k$  (i.e., we fix  $x_j$  at 0 if it was at 1, or at 1 if it was at 0), we underline  $j$  (or  $-j$ ), and we free all variables fixed after  $x_j$ , i.e., delete all elements of  $J_k$  to the right of  $j$  (or  $-j$ ).

## LOGICAL TESTS

The following are some of the basic tests used to fathom a given partial solution  $J_k$ . Let  $M = \{1, \dots, m\}$  be the index set for the constraints. If

$$\sum_{j \in J_k^1} a_{ij} \geq b_i, \quad \forall i \in M$$

then the zero completion of  $J_k$  is feasible. Further, since the rules of the procedure guarantee that only partial solutions whose zero completion is better than  $\hat{x}$  can be generated (this is the role of the set  $Q_1^-$  to be introduced below), we are in Case (c).

Otherwise, define

$$Q_1^- = \left\{ j \in N_k \mid c_j \geq c\hat{x} - \sum_{i \in J_k^1} c_i \right\}$$

Clearly,  $x_j$  has to be 0 for all  $j \in Q_1^-$  in any completion of  $J_k$  better than  $\hat{x}$ . Further, for each  $i \in M$ , define

$$s_i = \sum_{j \in N_k - Q_1^-} \max\{0, a_{ij}\} + \sum_{j \in J_k^1} a_{ij} - b_i$$

Obviously,  $s_i$  is an upper bound on the value of the slack variable of the  $i$ th constraint in any feasible completion of  $J_k$  better than  $\hat{x}$ . Thus, if  $s_i < 0$  for at least one  $i \in M$ , then  $J_k$  has no feasible completion better than  $\hat{x}$ , i.e., we are in Case (d).

Otherwise, let

$$Q^{i+} = \{h \in N_k - Q_1^- \mid a_{ih} > s_i\}$$

$$Q^{i-} = \{h \in N_k - Q_1^- \mid -a_{ih} > s_i\}$$

and

$$Q^+ = \bigcup_{i \in M} Q^{i+}, \quad Q^- = \left( \bigcup_{i \in M} Q^{i-} \right) \cup Q_1^-$$

The sets  $Q^{i+}$  and  $Q^{i-}$  designate variables that have to be 1 and 0 respectively if the  $i$ th constraint is to be satisfied. Therefore  $Q^+$  and  $Q^-$  stand for sets of variables that have to be 1 and 0, respectively, in any feasible completion of  $J_k$  better than  $\hat{x}$ .

Thus, if  $Q^+ \cap Q^- \neq \emptyset$ , we are in Case (d), since a variable cannot be 1 and 0 at the same time. Otherwise, if  $Q^+ \cup Q^- \neq \emptyset$ , we are in Case (b); and if  $Q^+ \cup Q^- = \emptyset$ , in Case (a).

## FINITENESS AND NONREDUNDANCE

The procedure discussed above finds an optimal solution if one exists, or shows the absence of a feasible solution otherwise, in a finite number of steps. Furthermore, the procedure is nonredundant in the sense that once a partial solution  $J_k$  is discarded (backtracked from), no descendent of  $J_k$ , i.e., no partial solution  $J_t$  such that  $J_t \supset J_k$ , is ever generated. The first property follows from the fact that the rules for fixing and freeing variables, together with the validity of the tests, guarantee the exhaustiveness of the search. The second one follows from the fact that at least one nonunderlined element of every partial solution  $J_k$  is underlined in all partial solutions generated after backtracking from  $J_k$ ; whereas all nonunderlined elements of  $J_k$  remain nonunderlined in all descendants of  $J_k$ .

## COMPUTATIONAL ASPECTS OF ADDITIVE IMPLICIT ENUMERATION

As already mentioned, the above version of implicit enumeration involves only additions and comparisons. To distinguish it from versions to be discussed below, which periodically solve a linear program, we call it additive. A great advantage of additive implicit enumeration is the ease of its implementation on any computer, even a very small one. Memory requirements are minimal and roundoff errors do not occur. The degree of difficulty of a given problem for this approach depends primarily on two factors, problem structure and the number of variables. The limiting factor in problem-solving capability is, of course, computing time. For a given number of variables, an increase in the number of constraints usually tends to decrease rather than increase the computational effort required, provided that problem structure is not heavily affected. An important feature to be mentioned is that finding a "reasonably good" (often optimal) feasible solution usually takes a fraction of the total time, i.e., the time required for finding an optimal solution and proving that it is optimal.

While problems with more than 50 variables cannot be guaranteed to be solvable within reasonable time limits by this procedure *unless they have some special structure*, in view of the fact that many real-world problems do have structure, additive implicit enumeration is sometimes quite successful on much larger problems. While some success was reported in solving moderate-size capital budgeting [11] as well as other, not very strongly structured problems [10] by additive implicit enumeration, the more conspicuously successful runs all concern special structures, as, for instance, Brauer's truck delivery problems [15] involving close to 800 variables, or the switching circuit design problems of Ibaraki, Liu, Baugh, and Muroga [16], involving 140 to 400 variables. A typical example of a successful application over a long period of time is described by Thornton [17], who is using a version of Balas' additive algorithm for scheduling the production of the oxygen-free copper casting plant of the USMR Company in Carteret, New Jersey. Thornton reports having successfully solved well over 600 production scheduling problems with 50 to 120 zero-one variables for the above-mentioned plant. CPU time on a PDP 10 for computing and listing a 1-month production scheduling plan ranges between 10 and 30 sec (only part of this time is used for solving a 0-1 program).

## SPECIALIZED VERSIONS OF ADDITIVE IMPLICIT ENUMERATION

The above results were obtained with general-purpose additive implicit enumeration codes applied to 0-1 programs having some structure. Much better results can, of course, be obtained when the problem is sufficiently structured to permit the development of a specialized algorithm. The most successful among the specialized additive

implicit enumeration algorithms at present seem to be the ones developed for set partitioning. This is the problem of finding

$$\min\{cx \mid Ax = e, \quad x_j = 0 \text{ or } 1, \quad \forall j\}$$

where  $e$  is a vector of ones and  $A$  is a matrix of zeroes and ones. It is an appropriate mathematical model for airline crew scheduling and a number of other scheduling problems. A specialized implicit enumeration procedure was proposed for this problem by Pierce [18], and a closely related one by Garfinkel and Nemhauser [19]. The main fact used by these procedures is that if a variable  $x_j$  is set to 1, then all other variables whose associated columns are not orthogonal to (i.e., do not form a zero inner product with) column  $j$  can be set to 0. A certain ordering of the columns of  $A$  makes it possible to use this feature in a very efficient manner. Also, advantage is taken of the fact that all entries of  $A$  and  $e$  are binary and can therefore be stored as bits, and that certain steps of the algorithm can be implemented by using logical AND and logical OR statements. All this has made it possible to solve set partitioning problems with many hundreds of variables in a matter of seconds, which is considerably better than what one can do at this stage with other methods. For a given number of variables and constraints, a higher density of nonzero entries makes the problem easier for this approach (since the number of columns that are not orthogonal to a given column tends to be higher). Some large problems, with 1400 to 1800 variables, 40 to 100 constraints, and a density of 0.15 to 0.25, have been solved on an IBM 7094 in 1 to 15 min with the algorithm described in Ref. 19, but others could not be solved within 15 min. An improved version of Ref. 18, due to Pierce and Lasky [20], which uses the reduced costs of the optimal linear programming solution in place of the original costs (but makes no other use of the linear program), has solved 7 out of 10 problems with 15 to 60 constraints and 1200 to 3500 variables in 15 to 300 sec of CPU time on an IBM 360/67; but could not solve the remaining 3 problems within 9, 23, and 23 min of CPU time, respectively.

Implicit enumeration procedures are not affected by the addition of some linear inequalities with arbitrary positive coefficients, which are often required in order to make the model adequate.

## SURROGATE CONSTRAINTS

In the approach discussed above, logical tests are applied to one problem constraint at a time. However, any positive linear combination of some (or all) problem inequalities, called a *surrogate* constraint, is obviously satisfied by all feasible solutions, hence can be added to the problem as a new constraint. Further, as observed by Glover [5], a logical test which is inconclusive when applied to any of the individual constraints, may become conclusive when applied to such a linear combination of constraints.

The potential usefulness of a surrogate constraint can be characterized as follows (see Ref. 12). If  $u^1$  and  $u^2$  are nonnegative  $m$ -vectors used to generate the two surrogate constraints

$$(s_1) \quad u^1 Ax \geq u^1 b \quad \text{and} \quad (s_2) \quad u^2 Ax \geq u^2 b$$

we say that  $(s_1)$  is *stronger* than  $(s_2)$  if

$$\min \left\{ cx \middle| \begin{array}{l} u^1 Ax \geq u^1 b \\ 0 \leq x_j \leq 1, \quad j \in N \end{array} \right\} > \min \left\{ cx \middle| \begin{array}{l} u^2 Ax \geq u^2 b \\ 0 \leq x_j \leq 1, \quad j \in N \end{array} \right\}$$

The intuitive justification for this notion of strength lies in the fact that the minimum objective function value of  $(\bar{P})$ , the linear program obtained from  $(P)$  by replacing the conditions  $x_j = 0$  or  $1$  by  $0 < x_j < 1$ , is an upper bound on the minimum objective function value of any surrogate linear program, i.e., any linear program obtained by replacing the inequalities of  $(\bar{P})$  with a single surrogate constraint; and the closer the minimum value of a surrogate linear program comes to this upper bound, the more restrictive is the surrogate constraint; i.e., the more it “replaces” or “represents” the whole initial constraint set.

One question which is of interest in this respect is whether for every 0-1 program there exists a surrogate constraint which actually forces the minimum value of the objective function to its upper bound. As shown in Ref. 12, such a strongest surrogate constraint always exists, and it can be obtained by solving  $(\bar{D})$ , the dual of  $(\bar{P})$ . To be more specific, let  $\bar{x}$  and  $(\bar{u}, \bar{v})$  be optimal solutions to the pair of dual linear programs

$$\begin{array}{ll} (\bar{P}) & \min cx \\ & Ax \geq b \\ & 0 \leq x \leq e \end{array} \quad \begin{array}{ll} (\bar{D}) & \max ub - ve \\ & uA - v \leq c \\ & (u, v) \geq 0 \end{array}$$

where  $e = (1, \dots, 1)$  is an  $n$ -vector.

Then

$$\min \left\{ cx \middle| \begin{array}{l} \bar{u} Ax \geq \bar{u} b \\ 0 \leq x_j \leq 1, \quad j \in N \end{array} \right\} = c\bar{x}$$

i.e.,  $\bar{u} Ax \geq \bar{u} b$  is a strongest surrogate constraint, which forces the minimand to the same value as the full constraint set of  $(\bar{P})$ . (Glover [5] and Geoffrion [6] define surrogate constraint strength in slightly different terms than the above, which is based on Ref. 12. See Ref. 22 for a discussion.)

## IMPLICIT ENUMERATION WITH LINEAR PROGRAMMING

The "filter method" proposed in Ref. 12 solves the linear program ( $\bar{P}$ ) at the start, and uses the optimal values of the dual variables to generate the above strongest surrogate constraint. The latter is then used both to direct the choice of variables to be fixed and to "filter" the partial solutions to be considered, via a special logical test applied to it, before the usual tests are applied to the original problem constraints.

While a surrogate constraint of the above type is certainly helpful, a *single* constraint is of only limited usefulness. This was recognized by Geoffrion [6], who carried the idea to its logical consequences in an implicit enumeration algorithm that generates a new surrogate constraint at each node of the search tree by solving the linear program in the free variables. Actually, Geoffrion adds to the above surrogate the inequality  $-cx \geq -c\hat{x}$ , where  $\hat{x}$  is the currently known best solution, so that his surrogate constraints are of the form

$$(\bar{u}A - c)x \geq \bar{u}b - c\hat{x}$$

where, of course, some of the components of  $x$  are fixed.

Apart from generating new surrogate constraints which enhance the efficiency of the logical tests, solving the linear program for each partial solution is by itself a powerful fathoming device: if the minimum of the linear program in the free variables is not less than  $c\hat{x}$ , then obviously the current partial solution has no feasible completion better than  $\hat{x}$ . Further, the minimum of the linear program in the free variables can be replaced in this test by any lower bound on the minimum of the integer program in the free variables.

Computational experience reported by Geoffrion [6] and others shows that solving the linear program either for each partial solution or periodically, in most cases substantially enhances the efficiency of implicit enumeration. Of course, preserving numerical stability throughout the hundreds, or thousands, of pivots performed during the repeated solution and/or reoptimization of linear programs, requires much more sophisticated programming techniques than additive implicit enumeration. However, this price is usually worth paying, with the exception of some special structures, as for instance set partitioning problems with many constraints, where additive implicit enumeration by itself can handle large problems, while solving the corresponding linear programs may become very expensive because of massive degeneracy.

LP-based implicit enumeration can be viewed as a synthesis of Balas' additive implicit enumeration approach with Land and Doig's LP-based branch and bound procedure [14], as specialized to the 0-1 case. The question arises in connection with this of whether the logical tests still have any justification when linear programming is used as a fathoming device. The answer is yes, since the logical tests use the binary property of the variables, which the linear program overlooks. This is corroborated by both indirect and direct evidence. In solving eight test problems with 44 to 100 binary variables by an LP-based implicit enumeration code with and without surrogate

constraints, Geoffrion and Marsten [23] have found that *not* using the surrogate constraints has increased the running time by an average of 100% (the median being 42%). Since the surrogates are positive linear combinations of the original inequalities, their presence cannot enhance the fathoming power of the linear programs, but may, on the contrary, increase the time spent on linear programming. The only possible source of the savings are therefore the logical tests, whose efficiency is enhanced by the presence of surrogate constraints. More recently, Piper [24] has found on 12 test problems from the literature, with 30 to 100 binary variables, that removing the bulk of the logical tests from an LP-based implicit enumeration code has resulted in an increase of the computing time by an average of 98% (the median being 39%).

## FLEXIBLE BACKTRACKING

The “depth-first” search strategy embodied in the LIFO (last-in-first-out) backtracking rule discussed above has the obvious advantage of simplicity, low storage requirement, and ease of bookkeeping. That is why the first computationally successful branch and bound algorithms for general mixed integer programming (see, for instance, Beale and Small [21]), while using as their starting point the pioneering work of Land and Doig [14], have nevertheless replaced the “breadth-first” approach of the latter (also preserved in the somewhat more general version of Bertier and Roy [25]), with “depth-first” in the form of LIFO. On the other hand, the rigidity of the LIFO rule often becomes cumbersome, since it implies that a bad decision made early in the search cannot be corrected until late in the search. A strategy which preserves most of the advantages of the “depth-first” approach while getting rid of the rigidity of LIFO, is to augment the partial solution last generated whenever it cannot be fathomed; but when it can be fathomed, then to continue the search with the “most promising” unfathomed partial solution rather than with the immediate predecessor of the current one.

This strategy can be embodied in the following “flexible backtracking” rule:

If  $J_k$  is fathomed, choose any nonunderlined element  $j$  (or  $-j$ ) of  $J_k$ , and replace  $J_k$  by  $J_{k+1}$  and  $J_{k+2}$ , where  $J_{k+1}$  is obtained by underlining  $j$  (or  $-j$ ), while  $J_{k+2}$  by changing the sign of, and underlining  $j$  (or  $-j$ ), underlining all elements before, and deleting all elements after  $j$  (or  $-j$ ). Store  $J_{k+1}$  and continue with  $J_{k+2}$ .

If, for example,  $J_k = \{2, \underline{7}, -3, 5, \underline{1}, -4\}$  is fathomed, and some criterion indicates that among all the value assignments that have produced  $J_k$ , the most likely one to have been wrong was setting  $x_3$  to 0, then we replace  $J_k$  by

$$J_{k+1} = \{2, \underline{7}, -3, 5, \underline{1}, -4\}$$

and

$$J_{k+2} = \{2, \underline{7}, \underline{3}\},$$

we store  $J_{k+1}$ , and continue with  $J_{k+2}$ . This partitions the set of all not-yet-explored partial solutions into descendants of  $J_{k+1}$  and of  $J_{k+2}$ . The search tree associated with  $J_k$  is now replaced by two search trees associated with  $J_{k+1}$  and  $J_{k+2}$ , respectively (see Piper [24]).

Finding a “good” criterion for selecting the node (partial solution) for a “flexible backtrack,” i.e., for choosing the element  $j$  (or  $-j$ ) of the above rule, is a matter of experimentation. For a general-purpose mixed-integer branch and bound code, Forrest, Hirst, and Tomlin [26] report good results with the criterion prescribing the choice of the node (partial solution)  $J_s$  with the smallest “projected cost”

$$C_s = cx^s + \frac{c\hat{x} - cx^o}{p_o} p_s$$

where  $\hat{x}$  is as before,  $x^o$  and  $x^s$  are optimal solutions to  $(\bar{P})$  and to the current LP (in the free variables) respectively, while  $p_o$  and  $p_s$  stand for some measure of the “integer infeasibility” of  $x^o$  and  $x^s$ , respectively. The measure used in Ref. 26 was

$$p_s = \sum_i \min\{x_i^s, 1 - x_i^s\}$$

and accordingly for  $p_o$ .

Recent experimentation [24] with an LP-based implicit enumeration algorithm for 0-1 programming, featuring flexible backtracking, has shown that the “projected cost” criterion can be considerably improved by using a better measure of infeasibility. Among several criteria tested on a sample of problems with 30 to 100 variables, the one that performed best was the “sum of infeasibilities” proposed in Ref. 3, which replaces  $p_o$  and  $p_s$  by  $q_o$  and  $q_s$ , respectively, where

$$q_s = \sum_{i \in M} \max\{0, b_i - \sum_{j \in J_s^{-1}} a_{ij}\}$$

and  $q_o$  is defined in the same manner.

## FURTHER LOGICAL IMPLICATIONS OF BINARINESS

The logical tests discussed earlier can be generalized into a procedure which captures the logical implications of binariness by generating a set of inequalities implied by the original problem constraints (and the 0-1 property of the variables). Indeed, Balas and Jeroslow [27], and independently, Granot and Hammer [28], have shown that any linear inequality in 0-1 variables implies, and is actually equivalent to (i.e., satisfied by the same set of binary vectors as), a set of linear inequalities in the same variables, but with all coefficients equal to 1, 0, or  $-1$ . This implied set of inequalities (called *canonical*) is stronger than the original inequality in the sense that it is violated by some nonbinary vectors satisfying the latter. Actually, as shown in Refs. 29, 38, and 39, most

of these implied inequalities are either facets of the convex hull of feasible 0-1 points, or can be turned into facets by increasing some of their coefficients. Thus, by adding to the constraint set of a 0-1 program some of the implied inequalities of Refs. 27 or 29, one obtains a more tightly constrained linear program which can yield considerably stronger bounds than the original one, while the corresponding 0-1 programs are still equivalent. On the other hand, by concentrating on canonical inequalities involving as few variables as possible and examining their joint implications, one can arrive at fixing variables in a way similar to the last logical test discussed earlier (see Hammer and Nguyen [30] and Spielberg [31]).

## THE MIXED INTEGER CASE

Using the logical implications of binariness does not seem to make sense in the case of mixed-integer 0-1 programs, where part of the variables are binary and part continuous, unless some of the problem constraints involve only the binary variables (which is often the case). Nevertheless, implicit enumeration can be extended to arbitrary mixed-integer 0-1 programs by using a basic result due to Benders [32], according to which every mixed integer program is equivalent to a pure integer program in the same integer-constrained variables. Though the constraint set of this equivalent integer program is potentially very large, and obtaining each member of the set requires the solution of a linear program in the continuous variables of the initial problem, this approach is nevertheless attractive since (1) if the linear program in the continuous variables is strongly structured (say, a transportation or a transshipment problem, as in the case of plant or warehouse location), then its solution is easy; (2) only part of the constraints of the equivalent integer program are needed for finding an optimal solution, and they may be generated as needed. Algorithms based on this approach are discussed in Refs. 32, 13, and 33.

## OTHER EXTENSIONS

Implicit enumeration has been extended to quadratic pure and mixed integer 0-1 programming (see Refs. 34 and 35).

## COMPUTATIONAL EXPERIENCE WITH LP-BASED IMPLICIT ENUMERATION

To give the reader some idea of the computational efficiency of an LP-based implicit enumeration algorithm, Table 1 lists the solution times and other basic data for 21

zero-one programming problems taken from the literature and solved with Piper's FORTRAN code discussed in Ref. 24. The first three problems are randomly generated, without any special structure; problems 4-7 are Lorie-Savage type capital budgeting models used at Motorola Corp.; problems 8-11 are models generated at Texaco and IBM; finally, the last 10 are randomly generated capital budgeting problems. Details about these problems, as well as solution times with other algorithms, are available from the papers where they were first published.

TABLE 1

Computational Experience with C. Piper's Code [24]

Problem <sup>a</sup>	Size (constraints × variables)	Solution time (sec) on UNIVAC 1108	No. of nodes generated	No. of pivots
1. B-M 23	20 × 27	13	57	709
2. B-M 24	20 × 28	24	110	1549
3. B-M 25	20 × 30	42	173	2212
4. P 4	10 × 20	0	4	27
5. P 5	10 × 28	1	13	50
6. P 6	5 × 39	1	10	92
7. P 7	5 × 50	3	19	176
8. L-S B	28 × 35	1	3	33
9. L-S C	12 × 44	3	16	63
10. L-S D <sub>2</sub>	37 × 74	118	71	1202
11. L-S E <sub>1</sub>	28 × 89	1568	2566	18127
12. J-S 11	10 × 100	174	214	4042
13. J-S 12	10 × 100	88	170	2937
14. J-S 13	10 × 100	162	179	3321
15. J-S 14	10 × 100	22	57	672
16. J-S 15	10 × 100	115	248	4102
17. J-S 16	10 × 100	48	75	1680
18. J-S 17	10 × 100	7	4	250
19. J-S 18	10 × 100	26	55	870
20. J-S 19	10 × 100	75	112	2663
21. J-S 20	10 × 100	47	147	1453

<sup>a</sup> Sources: B-M, Bouvier-Messoumian [36]. P, Petersen [11]. L-S, Lemke-Spielberg [13]. J-S, Jeroslow-Smith [37].

While precise comparisons with other procedures would require a much more detailed study, it can safely be stated that with the current state of the art, implicit enumeration can solve pure 0-1 programs usually somewhat faster, and often a lot faster than other, general-purpose integer programming methods.

## REFERENCES

1. E. Balas, Programare lineară cu variabile bivalente (Linear programming with binary variables), in *Proceedings of the Third Scientific Conference on Statistics*, Bucharest, December 5–7, 1963, pp. 13–20.
2. E. Balas, Un algorithme additif pour la résolution des programmes linéaires en variables bivalentes, *C. R. Acad. Sci., Paris* **258**, 3817–3820 (1964).
3. E. Balas, An additive algorithm for solving linear programs with zero-one variables, *Operations Res.* **13**, 517–546 (1965).
4. A. M. Geoffrion, Integer programming by implicit enumeration and Balas' method, *SIAM Rev.* **7**, 178–190 (1967).
5. F. Glover, A multiphase-dual algorithm for the zero-one integer programming problem, *Operations Res.* **13**, 879–919 (1965).
6. A. M. Geoffrion, An improved implicit enumeration approach for integer programming, *Operations Res.* **17**, 437–454 (1969).
7. E. L. Lawler and M. D. Bell, A method for solving discrete optimization problems, *Operations Res.* **14**, 1098–1112 (1966).
8. J. D. C. Little, K. G. Murty, D. W. Sweeney, and C. Karel, An algorithm for the traveling salesman problem, *Operations Res.* **11**, 972–989 (1963).
9. F. Glover and S. Zionts, A note on the additive algorithm of Balas, *Operations Res.* **13**, 546–549 (1965).
10. B. Fleischmann, Computational experience with the algorithm of Balas, *Operations Res.* **15**, 153–155 (1967).
11. C. C. Petersen, Computational experience with variants of the Balas algorithm applied to the selection of R and D projects, *Management Sci.* **13**, 736–750 (1967).
12. E. Balas, Discrete programming by the filter method, *Operations Res.* **15**, 915–957 (1967).
13. C. E. Lemke and K. Spielberg, Direct search zero-one and mixed integer programming, *Operations Res.* **15**, 892–914 (1967).
14. A. H. Land and A. G. Doig, An automatic method for solving discrete programming problems, *Econometrica*, **28**, 497–520 (1960).
15. K. Brauer, Binäre Optimierung, Doctoral Dissertation, University of Saarbrücken, 1968.
16. T. Ibaraki, T. K. Liu, C. R. Baugh, and S. Muroga, *An Implicit Enumeration Program for Zero-One Integer Programming*, Report No. 305, Department of Computer Science, University of Illinois (Urbana), January 1969.
17. J. C. Thornton, *Integer Programming Methods Used in Production Scheduling of the USMR Oxygen Free Continuous Casting Facility*, paper presented at the Joint National Meeting of ORSA-TIMS-IEEE, Atlantic City, November 7–10, 1972.
18. J. F. Pierce, Application of combinatorial programming to a class of all-zero-one integer programming problems, *Management Sci.* **15**, 191–209 (1968).
19. R. S. Garfinkel and G. L. Nemhauser, The set partitioning problem: Set covering with equality constraints, *Operations Res.* **17**, 848–856 (1969).
20. J. F. Pierce and J. S. Lasky, Improved combinatorial programming algorithms for a class of all-zero-one integer programming problems, *Management Sci.* **19**, 528–544 (1973).
21. E. M. L. Beale and R. E. Small, Mixed integer programming by a branch and bound technique, in *Proceedings of the International Federation of Information Processing Congress*, Vol. 2 (W. A. Kalenich, ed.), Spartan, 1965.
22. F. Glover, Surrogate constraints, *Operations Res.* **16**, 741–749 (1968).
23. A. M. Geoffrion and R. E. Marsten, Integer programming: A framework and state-of-the-art survey, *Management Sci.* **18**, 465–491 (1972).
24. C. J. Piper, Computational Studies in Optimizing and Post-Optimizing Linear Programs in Zero-One Variables, Ph.D. Thesis GSIA, Carnegie-Mellon University, May 1974.
25. P. Bertier and B. Roy, Procédure de résolution pour une classe de problèmes pouvant avoir un caractère combinatoire, *ICC Bull. (Rome)* **4**, 19–28 (1965).

26. J. J. H. Forrest, J. P. H. Hirst, and J. A. Tomlin, Practical solution of large and complex integer programming problems with UMPIRE, *Management Sci.* **20**, (1974).
27. E. Balas and R. Jeroslow, *On the Structure of the Unit Hypercube*, MSRR No. 198, Carnegie-Mellon University, August-December 1969; published as Canonical cuts on the unit hypercube, *SIAM J. Appl. Math.* **23**, 61-69 (1972).
28. F. Granot and P. L. Hammer, *On the Use of Boolean Functions in 0-1 Programming*, Operations Research Mimeograph No. 70, Technion, Haifa, August 1970.
29. E. Balas, *Facets of the Knapsack Polytope*, Management Science Research Report No. 323, Carnegie-Mellon University, September 1973.
30. P. L. Hammer and S. Nguyen, *APOSS—A Partial Order in the Solution Space of Bivalent Programs*, paper presented at the 41st National Meeting of ORSA, New Orleans, April 1972.
31. K. Spielberg, *Minimal Preferred Variable Reduction for Zero-One Programming*, Technical Report No. 320-3013, IBM Philadelphia Scientific Center, July 1972.
32. J. F. Benders, Partitioning procedures for solving mixed-variables programming problems, *Numerische Math.* **4**, 238-252 (1962).
33. E. Balas, Minimax and duality for linear and nonlinear mixed integer programming, in *Integer and Nonlinear Programming* (J. Abadie, ed.), Elsevier, 1970.
34. E. Balas, Duality in discrete programming. II: The quadratic case, *Management Sci.* **16**, 14-32 (1969).
35. P. Hansen, Quadratic 0-1 Programming by Implicit Enumeration, University of Brussels, 1971.
36. B. Bouvier and G. Messoumian, Programmes linéaires en variables bivalentes (Algorithme de Balas), Master's Thesis, University of Grenoble, 1965.
37. R. G. Jeroslow and T. H. C. Smith, *Experimental Results on Hillier's Linear Search Imbedded in a Branch and Bound Algorithm*, MSRR No. 326, Carnegie-Mellon University, November 1973.
38. P. L. Hammer, E. L. Johnson, and U. N. Peled, *Facets of Regular 0-1 Polytopes*, CORR 73-19, University of Waterloo, October 1973.
39. L. A. Wolsey, *Faces of Linear Inequalities in 0-1 Variables*, Discussion Paper No. 7338, CORE, Louvain, November 1973.

Egon Balas

















3 9001 02863 3892

**REFERENCE**

