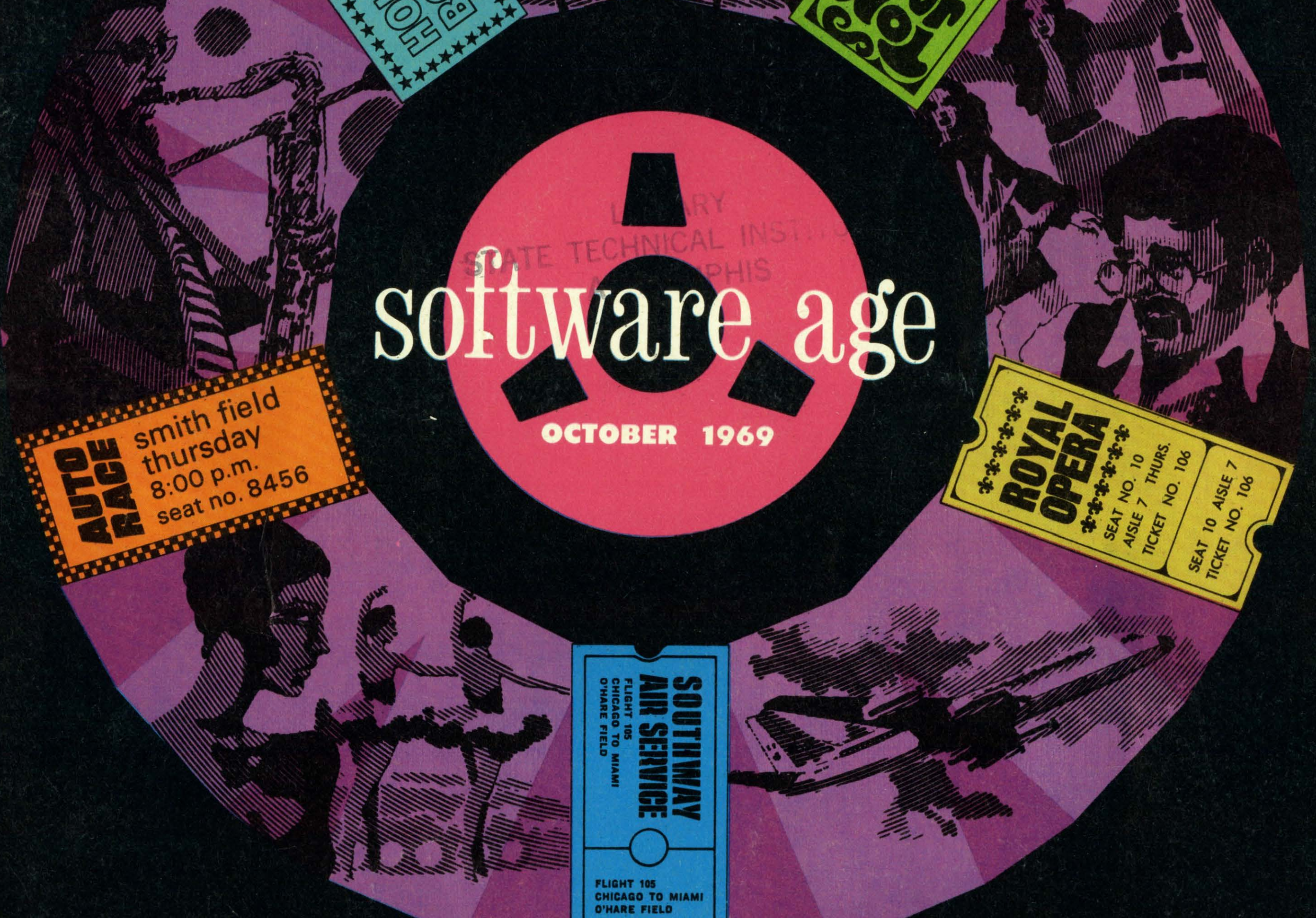

HOLLYWOOD
BOWL
TOM JONES
October 18
SECTION B
SEAT 26
ROW 7
software age
OCTOBER 1969
ROYAL
OPERA
SEAT NO. 10
AISLE 7 THURS.
TICKET NO. 106
SEAT 10 AISLE 7
TICKET NO. 106
AUTO
RACE
smith field
thursday
8:00 p.m.
seat no. 8456
SOUTHWAY
AIR SERVICE
FLIGHT 105
CHICAGO TO MIAMI
O'HARE FIELD
FLIGHT 105
CHICAGO TO MIAMI
O'HARE FIELD
TICKET NO. 10601

A RETURN TO SIMPLICITY

MORE THAN ENOUGH, we have heard from one quarter or another about how our society has been, is becoming and will be "de-humanized" by automation, mechanization, computerization, et al.

Life today is indeed complex. At times it feels as if we are existing in a morass of people, places and events rather than in a well-ordered scheme of things.

In this kind of social milieu, the critic-observer seems to have a point, for if we're not careful, we may end up reducing man to a mere automation. We will be a society of machine-made men rather than man-made machines.

However, to the extent that the observers are over-fearful, then computerization becomes a means to an end and a means of remedying our problems rather than causing them. With great irony, it brings us back to the simple life, for it can replace some of the idiotic, time-consuming and "dehumanizing" activity we've had to do for too much of our lives.

A case in point is reflected in the October cover, born of reports of the continual implementation and growth of a ticket buying and selling system developed by Computicket. Although taking two years to develop and program, the system is relatively simple to the customer—and after all, that's where it counts. And because it is simple, it can be explained briefly:

At either a Network terminal, situated at a "high traffic" location, or at a box office terminal at the event site, the customer places his order with the terminal operator. The request can vary from the general, such as the best seat on the house at whatever price, to selecting a particular seat in a particular row in a particular section. He sees his request on a display scope and verifies it with the operator who sends it off to the central computer. The computer answers his request by positing the tickets in a matter of seconds —even complete with parking information. The system has local, national, and international application for sporting, theater and other entertainment events.

Taken along with other widely applied time-saving computerization, Computicket and similar systems are good examples of how we can solve—not cause—problems due to having too many people in too small a space. They will liberate—not de-humanize—the public so that it can spend its time more wisely and less inhumanly than merely waiting in line—as long as the music from "Oliver" isn't reduced to Fortran and we aren't given robots in the outfield.

software designers and programmers: develop Foxboro systems that can talk to cement

(and chemicals and textiles and electricity and paper and...)

Today Foxboro's computer controlled processing systems are performing monitoring and direct digital control of kiln and wet blending operations for cement production. They're also controlling processing functions around the world in power plants, refineries, for the chemical industry, pulp and paper, textiles and food.

That's today. Tomorrow's programs are even more challenging, and they're underway now.

The complexity, size and world-wide diversity of existing projects and development programs now underway at Foxboro are greater than at any other time in our 60-year history. Our modern, multimillion dollar facility, located between Boston and Cape Cod, is staffed by one of the world's most experienced teams of computer professionals.

Many of them are developing systems to meet this year's requirements. Others are working on systems for the next decade, and beyond.

APPLICATIONS PROGRAMMERS. Opportunities exist at varying levels of experience, requiring creative individuals who can take complete responsibility for bringing a system's software from its introductory stage of design through to full real time simulation testing and installation. Experience is required in one or more of the following: Real Time Executives and/or Real Time Input-Output, Time Sharing, Batch Control, DDC, Setpoint Control and Multilevel Programming.

SOFTWARE SYSTEMS DESIGNERS AND PROGRAMMERS. You will work on developing a new generation of digital technology. Present programs encompass advanced Real Time Systems, Computer-based CRT Display Systems, and Man-Machine Communications. At least 3 years' programming experience is required, related to leading development efforts in operating systems, languages and communication, and control software development. You will be responsible for all phases of program designing, coding, checkout and documentation.

Write to Mr. Tom Bryant, Professional Placement Office, The Foxboro Company, Dept. SA10, Neponset Avenue, Foxboro, Mass. 02035. Foxboro is an equal opportunity employer.

FOXBORO

software age®

OCTOBER, 1969 **Vol. 3–No. 10**

Publisher .... David W. French
Editor .... John F. Sikora
Managing Editor .... Elspeth Taylor
Editorial Assistant .... Laura Brandt
Traffic Manager .... Betsy Pavkovich
Circulation .... Ann Vickers
Business .... Stephenie Jensen
Resume Control .... Sue Bullington

CIRCULATION OF THIS ISSUE
MORE THAN 130,000
BPA Audit Applied For

SOFTWARE AGE
2211 FORDEM AVENUE, MADISON, WISCONSIN 53701
PHONE: (608) 249–0128 OR (800) 356–8141

John Antil

Los Angeles, Calif.: Richard Faust, 9800 S. Sepulveda Blvd. (213) 776–0100

San Francisco, Calif.: Ernesto Montano, 595 Buckingham Way, Suite 226 (415) 731–8240

Philadelphia, Pa.: Richard D. Clemmer, 27 Acoma Lane, Collegeville, Pa. (215) 489–9141

Houston, Texas: W. Royce House, 4040 Southwest Freeway, Houston, Texas (713) 622–2868

SOFTWARE AGE is published monthly by

press tech inc.

2211 Fordem Avenue, Madison, Wisconsin 53701
Subscription free to qualified readers. Others, 10/yr. Individual copies, $1. Foreign subscriptions, $15/yr.
For change of address please return your address label from the magazine.
Controlled circulation postage paid at Madison, Wisconsin.

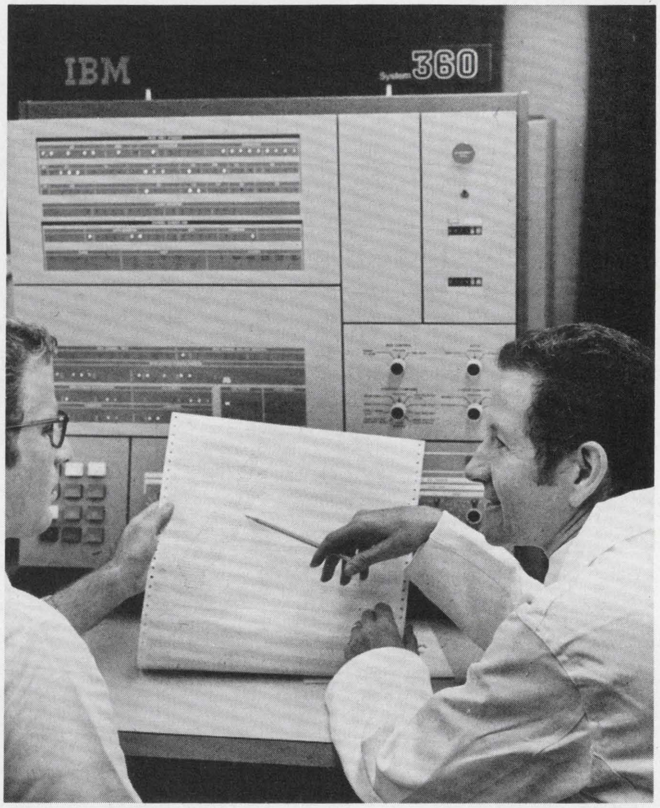
new applications

In the heart of Appalachia a total of 450 elementary school teachers are being instructed in ''new math''. The Appalachia Educational Laboratory, based in Charleston, West Virginia, is taking their program into the Appalachia area to provide individualized teaching instruction. The schedule calls for eight week stops in or near schools in three communities, during which teachers from the area are given an average of 30 hours of instruction. This includes 25 hours of review in new math and five hours of how to teach it—the latter being interspersed throughout the review segment.

The basis of the AEL experiment is the IBM 1500 Instructional system. It consists of a television-like display unit with a typewriter keyboard; a light pen or electronic probe which the student uses to indicate his answers on the screen; and an image projector on which is mounted another screen for the showing of color or black and white images.

Sixteen instructional stations are being used at which the teachers receive individualized instruction. Specially equipped IBM computers, when linked to the control unit and audiovisual devices, form the 1500 instructional system.

The teachers' records and course revisions (provided by the computer during instructional interactions with the teacher) are stored on magnetic tape and transmitted over telephone lines to Pennsylvania State University where the material is evaluated by an IBM System/360 Model 67.



Studies are being performed at the New England Medical Center to determine the long-term effects of radiation on man's heredity. Under the auspices of the Atomic Energy Commission (AEC) chromosomes are being examined with the aid of a computer for abnormalities. The percent of chromosome cells that are abnormal indicates the severity of radiation damage and is being determined with photomicrographs and an optical scanning device linked to a computer. The scanner measures the contour of the chromosome, including its length, mass and the ratio of its short arms to total length. Using a 35 mm photograph of a cell magnified 400 times, the scanner can measure the film density of 614,000 different points on each frame in a few seconds. The scanner transfers the most interesting of these points—which are like the dots in a newspaper photo—into the computer by measuring the dot's lightness or darkness. A report on the analyzed findings is then printed by the computer for review by the researcher.

Chromosome analysis with the aid of the computer reduces the time necessary to analyze the chromosomes of each patient and also provides less deviation in measurement, allowing the geneticists to be more precise in their analysis.

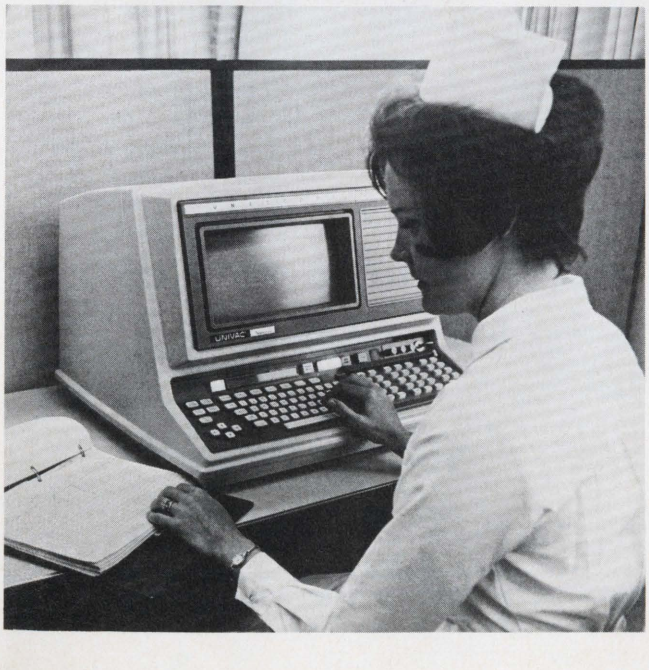
UNIVAC computers are now being used to assist nurses in caring for hospital patients. Computers can tell when a patient needs a pill or a bath. They can help detect hidden illnesses and they can keep track of everyone's complete medical history.

These are only a few of the jobs that are being performed by the new ''total medical information systems'' that are now entering use in today's modern hospitals.

The computer's memory is constantly updated with new information on the condition of the patients, changes in the schedule of treatment and other critical data as well as every event affecting the operation of the hospital. Hospital personnel can then tap the computer memory by means of UNISCOPE terminals located at various strategic points. Using the keyboard a nurse can ask the computer for the schedule of treatment for each patient in her care. Her question and the computer's reply, appear on the unit's screen.

There are many other possible uses for the computer. A few UNIVAC applications that are particularly interesting include:

- The storage of complete medical records of each person in a community. At the Danderyd Hospital, Stockholm, Sweden, a UNIVAC computer remembers the records of over 1,500,000 people. The computer can retrieve any record, or portions of a record, and flash information on a screen, in less than one second.
- The communication of computers with each other in nationwide and worldwide information networks. The Clinique Universitaire at the Universitat de Sherbrooke near Montreal, Canada, recently used a UNIVAC UNISCOPE 300 visual communications terminal to request a medical record from the Danderyd Hospital's computer in Sweden. The information appeared on the clinic's screen a few seconds later.



Engineers in Texas are using the computer to project the state's current and future highway needs. Assisted by a pair of photo-equipped airplanes and two IBM computers, the engineers compile vast amounts of information, ranging from projected population growth to the details of acquiring right-of-way and calculating the amount of earth fill needed for grading—as well as the total construction cost.

In preliminary highway planning, engineers enter statistics of growth projections, (compiled by the computer), traffic counts, and, occasionally, motorists' points of origin and destinations, into the computer and get the preferred routes.

Once a general strip of area for a new highway is selected, crews in leased airplanes equipped with special cameras fly over the area and take photographs. Each general strip is photographed from two different angles. Then technicians produce a large aerial map, in which the two images of the area overlap. With special equipment, engineers view it in 3-D. The known reference points in the three-dimensional aerial photograph are used to determine the distances and calculate the elevations of the land and objects in the picture. Information on the elevations is fed into the computers, and they produce cross-sectional maps of the land, which show the slope or grade and the land elevations along each possible highway route in the strip photographed. These maps and the data compiled enable the engineers to determine easily the amount of cutting and filling that would be needed along each route under consideration.

The system also shows property ownership lines along various highway routes being considered. The computers inform the engineers just how much of each parcel of property must be purchased to obtain the necessary right-of-way for a particular route.

The IBM computers are housed in Austin, but they perform services for the Houston Urban Office and each of the 25 Highway Department district offices across the state. Typewriter-like devices are now being installed in some district offices to enable engineers to communicate directly with the computers over telephone lines.

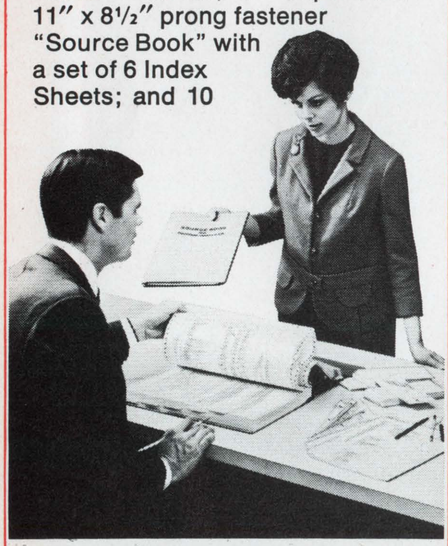
THREE CHEERS

FOR EDP PROGRAMMERS

Here they are: 3 new Wilson Jones program documentation and verification tools. Designed to make the programmers' job easier and more efficient in computer systems of every size.

1 Program Verification Organizer

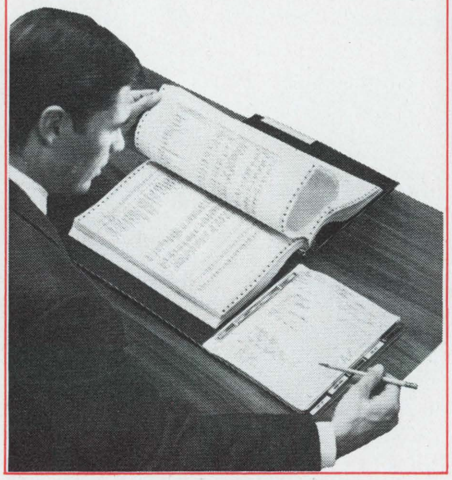
For source documents, printouts and control cards of irregular or unmatched volume. Three separate elements: a "Super-Flex" Nylon Post Binder for 14 7/8″ x 11″ unburst sheets; a "Super-Flex" 11″ x 8 1/2″ prong fastener "Source Book" with a set of 6 Index Sheets; and 10



insertable data processing Titles; and a clear vinyl 14 7/8″ x 11″ Envelope for 150 parameter cards. Loaded Organizers can also be prepared for filing in Wilson Jones "Data-Rack" with the addition of Hanger Sets.
No. 02-1411M. $4.25 complete.

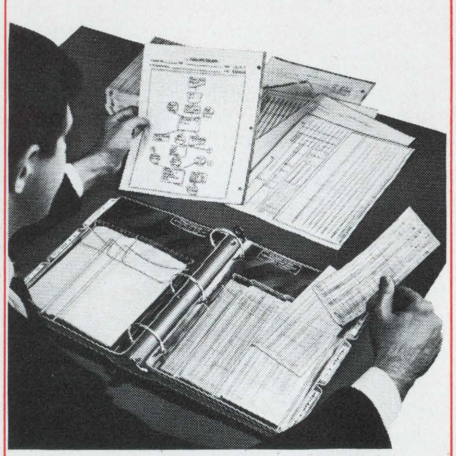
2 Program Documentation Binder

One-piece "Super-Flex" construction for systems in which source documents, parameter cards, and the program printout on 14 7/8″ x 11″ sheets are all of a volume that can be housed in a single unit. Occupies a minimum of space...provides maximum convenience. The printout side can be loaded from top or bottom as in a Wilson Jones nylon post binder. The document side includes a vinyl 11″ x 8 1/2″ Envelope for control cards and an Index with printed insertable Titles. 6 colors for coding source material.
No. 03-1411. $3.40 complete.



3 Control Card Pocket Indexes

Three 11″ x 8 1/2″ plastic pocket Index Sheets. Each holds up to 40 punch cards. Pockets are staggered to avoid bulking-up the cards. Sheets have a tab for in-



sertable Titles, both to identify the cards held and to serve as divider guides to the contents of loose leaf "Source Books," Documentation Binders and Procedure Manuals. Index Sheets keep control cards *right with* all related documents in the same sections of ring or prong binders.
No. 2-11. Set of 3 sheets $1.95.

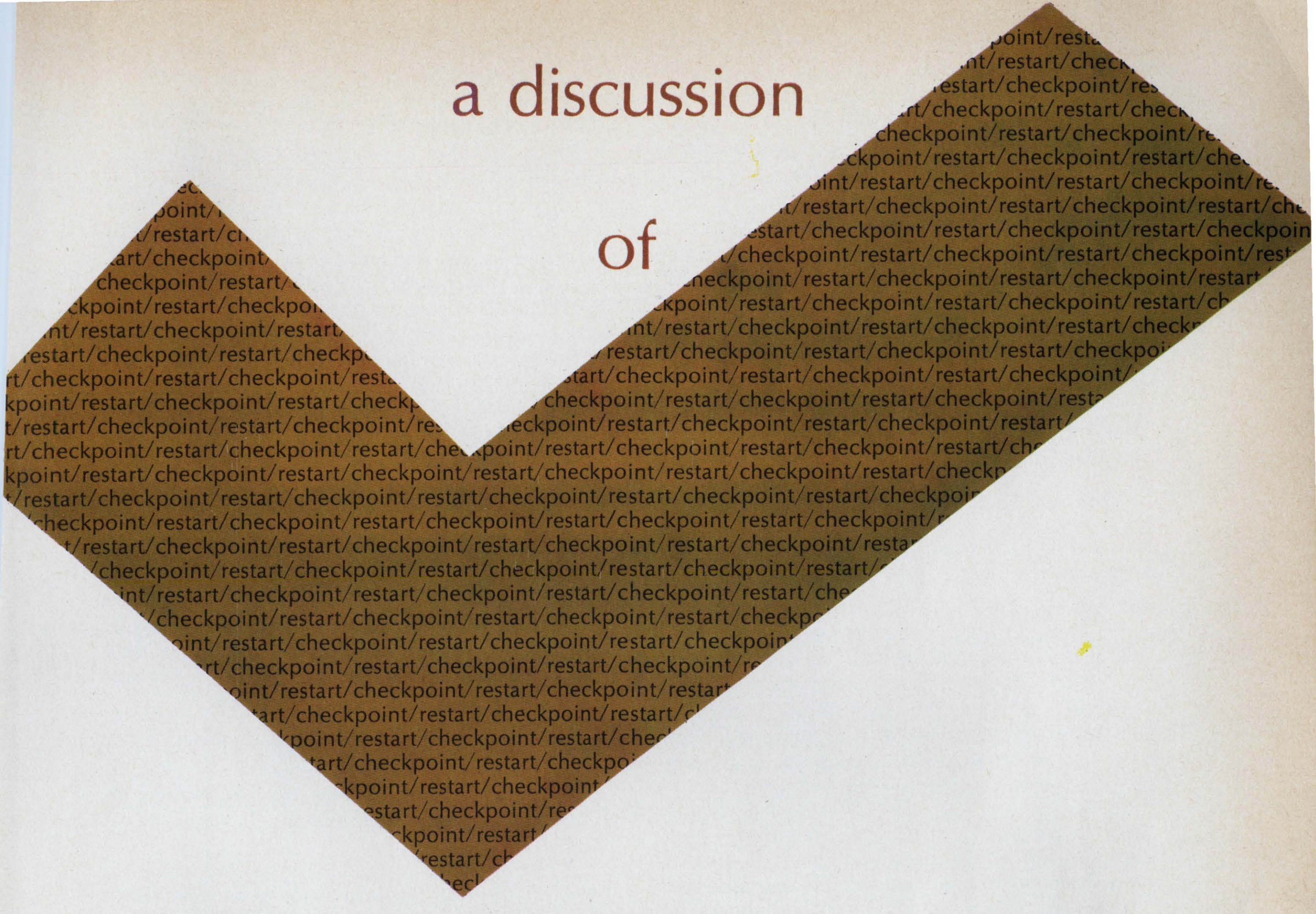
ASK YOUR OFFICE OR "TAB" SUPPLY DEALER FOR FULL INFORMATION OR WRITE:

WJ RECORD KEEPING ESSENTIALS

WILSON JONES
INVENTOR OF THE NYLON POST BINDER
A Division of *Swingline*® Inc. 6150 TOUHY AVENUE · CHICAGO 60648

For more information, circle No. 2 on the Reader Service Card

a discussion of checkpoint / restart



by David P. Jasper

INTRODUCTION

Any serious systems designer or user of an operating system/hardware hybrid must resolve the question: What kind of recovery ability will be provided? The question is answered with increasing difficulty as operating systems become more comprehensive; as asynchronous and unrelated tasks are performed with apparent simultaneity; and as the use of large scale "blackboard" storage becomes more popular. (The "blackboard" storage is a reference to the analogous use of drum, disk, and extended core space as temporary reusable storage.)

The problem to be solved is usually thought of in terms of providing, with or through the operating system, a generalized checkpoint capability either for the user's application oriented program or for the system itself and its drone or "background" functions.

The problem has become increasingly difficult to solve, and recently the problem has been avoided by designing systems without any intermediate recovery capability. As will be pointed out in the discussion, no intermediate checkpoints may be a valid solution; but as yet I know of no designer who has tackled the question or chosen a solution, good or bad, that is supported by reasons that reach deeper than simple expedience. Of special interest is a data center environment which must both provide the tools —machine time and software—for some clients and use these same tools to provide answers for other clients. The question of recovery ability must then be faced by a data center from both aspects; as a system designer, and as a user.

Since the Checkpoint/Restart problem has been under examination for some time, a review of the basic principles and design problems thus far is useful in two senses:

1) The first and most important step to the solution of any problem is a good definition of that problem; it is, in fact, the prerequisite for understanding the problem.
2) A record of thought thus far prevents recursion and repeated exploration down the same blind alleys.

PROBLEM DEFINITION

Checkpoints—By definition, a checkpoint is that collection of data that truly represents the status at a given time of a program and its peripheral activities. A checkpoint can be proven to *truly* represent the status if it is used to restart the program successfully. A checkpoint can take many forms. Matrix optimization programs with virtually no input/output can, as a byproduct of the calculations, produce parameters at each iteration (N). If these parameters are stored on an external device and subsequently recalled and used as multipliers on the original input matrix elements, the matrix will be restored to iteration level N. A tape sort could restart the current purge phase with little more than the original control deck parameters. In both of the preceding examples, the Checkpoint/Restart technique is tailored to the application.

Traditional Checkpoints—Traditionally a generalized checkpoint has been available which records all possible status information so that a checkpoint procedure is available for any program. For the traditional checkpoint, generalization was based on the thesis that the input data to the program can be restored to a given previous status. If the input integrity cannot be guaranteed, a restart is meaningless. In this sense "input data" is meant to include the coding in the program, for the program is indeed data and a given switch setting is input.

Shortcomings of Traditional Methods—The checkpointing of "program coding" is as simple or sophisticated as one may wish. The ultimate in design simplicity is, of course, an image of all program storage—a snapshot. When the program is a complex intertwining of an operating system and application-oriented coding, it may be more desirable to separate the components and reinterface at restart time. This case is most meaningful when there is more than one unrelated task (individual application-oriented program) sharing the same facilities. The desirability of performing separable checkpoints hinges on the decision to checkpoint the entire system or the individual programs. When a "snapshot" of the entire system is not preferred, the procedure for restarting expands to include not only a refreshing of program storage, but restart must also reinterface the application-oriented coding with the generalization system code. This procedure must necessarily include a regeneration of the system control blocks, queues, etc. The regenerated interface (control blocks) must adequately reflect the current system configuration and allocations, and yet remain *logically* identical to the environment in which the application-oriented code originated.

Redefining the Problem—An attempt to simplify reinterfacing at restart time gives rise to the first candidate for a restartable-checkpoint restriction: that checkpoints be taken when no I/O requests are queued (quiesced environment). The first corollary is that restart is performed in a quiesced environment. A logical extension from the restart interface problem is that the system interface can best be regenerated by the system itself; hence the job is resubmitted through the normal job input stream for restart.

Thus far the restrictions suggested would simplify checkpoint/restart coding, increase the burden on the applications programmer, but not wholly eliminate or exclude any type of program from the list of checkpointable. The checkpointing of teleprocessing programs has always been a special case due to the nature of the input. Even when a checkpoint is physically possible, it would not be meaningful to restart most applications which would require resubmitting the input. For this discussion we will limit the subject to nonteleprocessing environments, although it is hoped this paper will provide the starting point for deeper investigation of the subject.

Let us then consider input data which is stored on external devices. Reviewing the traditional forms of a generalized checkpoint reveals that both input and output data were processed serially; and, in fact, the data storage was organized in a physical sequential manner than can be thought of as lists. A traditional restart is now composed of

1) Refreshing program storage from some stored image, and
2) A logical repositioning of the external input lists and output lists.

For physical sequential organized data, a physical position (block count) corresponds to the logical position when coupled with a memory snapshot.

Thus, a snapshot of program storage and some block counts are adequate information to restart a program. By this we have satisfied our definition of a restartable checkpoint and met our criteria of restoring all the input data to a given status in time.

Projecting Traditional Methods—Applying similar logic today concerning externally stored data, we discover that new devices and data set organization complicate our task of logically repositioning the input and output data. Consider the program that writes the output in the same physical position that contained the input (update-in-place).

The Random Access Paradox—This update-in-place programming technique is very logical to use on a single-drum, disk, or any bulk random access storage space; e.g., extended core storage where there is no opportunity to copy. Even when there is a second device available, this technique is very economical with execution time if there are a small number of transactions to the file relative to time required to perform a complete copy operation. Such a program does not fit our criteria for restartable checkpoints, since the input is being continually modified or destroyed; hence, that program cannot even be restarted from the "top." Obviously any permanent I/O error, hardware failure, or program bug that arises is disaster unless another copy of the input is available for recovery. Copies are expensive; in fact, copying defeats all of the reasons for using an "update-in-place" procedure. One can only conclude that using an update-in-place procedure without a convenient means of recovering the input is bad programming or at best extremely dangerous. The question remains: Is it possible for a generalized checkpoint to correct or alleviate the update-in-place problem?

A Solution to the Paradox—If the operating system is going to protect the user of the update, it must preserve the input. One method might be to design a system access method so that a segment of the input and output is maintained in a second or "active" buffer located on another device or in main storage. For purposes of discussion, let us assume that the data records in this active buffer are all maintained in TTR order (the location or address of the logical record on the random access device). Thus, the access method does not write on the input data set and simulates the writing by updating the "active buffer." All read requests would first scan the activity buffer for the "current" record and then go to the external device. All write requests would always write to the active buffer only. When the buffer is filled to capacity, the access method would checkpoint the problem program and the relatively small active buffer and then proceed to dump the buffer on the external device. Because the input is being clobbered in steps, only the most recent checkpoint would be valuable for restart. Although this technique would work logically, it could prove impractical or at best, somewhat inefficient since reasonable size "active buffers" would tend to fill quite frequently. It would, however, avoid a total copy of the BRASS file (Bulk Random Access Storage Space).

The technique suggested above for "holding back" on writing the output involves an overhead for maintaining the "active buffer" which must be considered in evaluating the per cent of CPU invested in providing check points. If the active buffer is maintained in main storage, the overhead is small; and in some applications where repeated accesses are made to the small logical record, this high activity buffer would actually be more efficient than access to an external device. However, the capacity of a buffer in main storage would be quite limited; and, in fact, checkpoints would be every few seconds or less in most applications. If the high activity buffer is maintained on some external device, even an unused portion of the one being updated, or external core storage, the capacity would increase but the overhead of copying this information at each checkpoint could also become prohibitive. Of course, the location of the activity buffer could be optional and suited to the application; but many applications would suffer with either option.

THE COST OF RECOVERY

The overhead required in taking a checkpoint is not trivial. Traditionally checkpoints have been taken at the end of an output tape reel; i.e., about 5–10 minute intervals. For moderate to large scale machines, a checkpoint written on tape would probably not require much more than one to five seconds (i.e., 0.8 to 4.0 × 10^5 characters). If my math is not too rusty, that means that about 0.18% to 1.7% of the CPU time was invested in taking checkpoints.

The Cost of Intermediate vs. Total Rerun—Not entirely unreasonable in view of modern reliabilities would be the elimination of any checkpoint procedure. When a total rerun is possible, a valid question might be how often will a given program execute without a failure. If T is the program running time, S is the number of successful runs, then we can speculate at what point the overhead of taking a checkpoint plus the average time to restart, equals the average time to rerun the total program.

$$S \cdot T(\Delta T) + \bar{R} = \bar{T}$$

where $\bar{T} = \frac{1}{2}T$

$\bar{R} = \frac{1}{2}R$

$$S = \frac{T - R}{2T(\Delta T)}$$

where T = program run time in minutes

S = number of successful runs

R = time span between checkpoints in minutes

ΔT = fraction of run time used in taking checkpoints $= \dfrac{\text{chkpt time}}{R}$

In Figure 1 various values for ΔT and R (the point at which the curve crosses the Y-axis) have been selected. In each case, the curve asymptotically approaches a value for the number of successful runs as the run time increases. The dashed line indicates a constant value of 50 hours. If 50 hours was the mean time to failure, then all values of run time (T) which fall on curves below the broken line can be run more economically by not taking a checkpoint and by rerunning from the beginning of the program. The purpose of this graph is not to indicate specific values but to demonstrate that there are several important factors to be weighed before determining that any checkpoint is worth taking. We have seen in this not too unrealistic example a strong case for not taking any checkpoints. When the time required to take a checkpoint is quite long (≥2.5 seconds) and the entire system can be relied upon for not more than one failure in each 100-hour period, the time of a given run can exceed an hour before checkpoints should be considered. If checkpoints are taken in an environment where there is a mixture of jobs of all durations, then the breakeven point for taking checkpoints would presumably be decided on a typical job size or a weighted average. Perhaps a nominal fee could be based on the reliability of the hardware and result in less average cost per computer user than using checkpoints. But the purpose of this discussion is not to "make book."

Certainly the value of a checkpoint cannot always be considered in terms of rental dollars or CPU time alone. Checkpoints have another use as "getoffs", a means by which a program can be suspended so that the entire machine con-

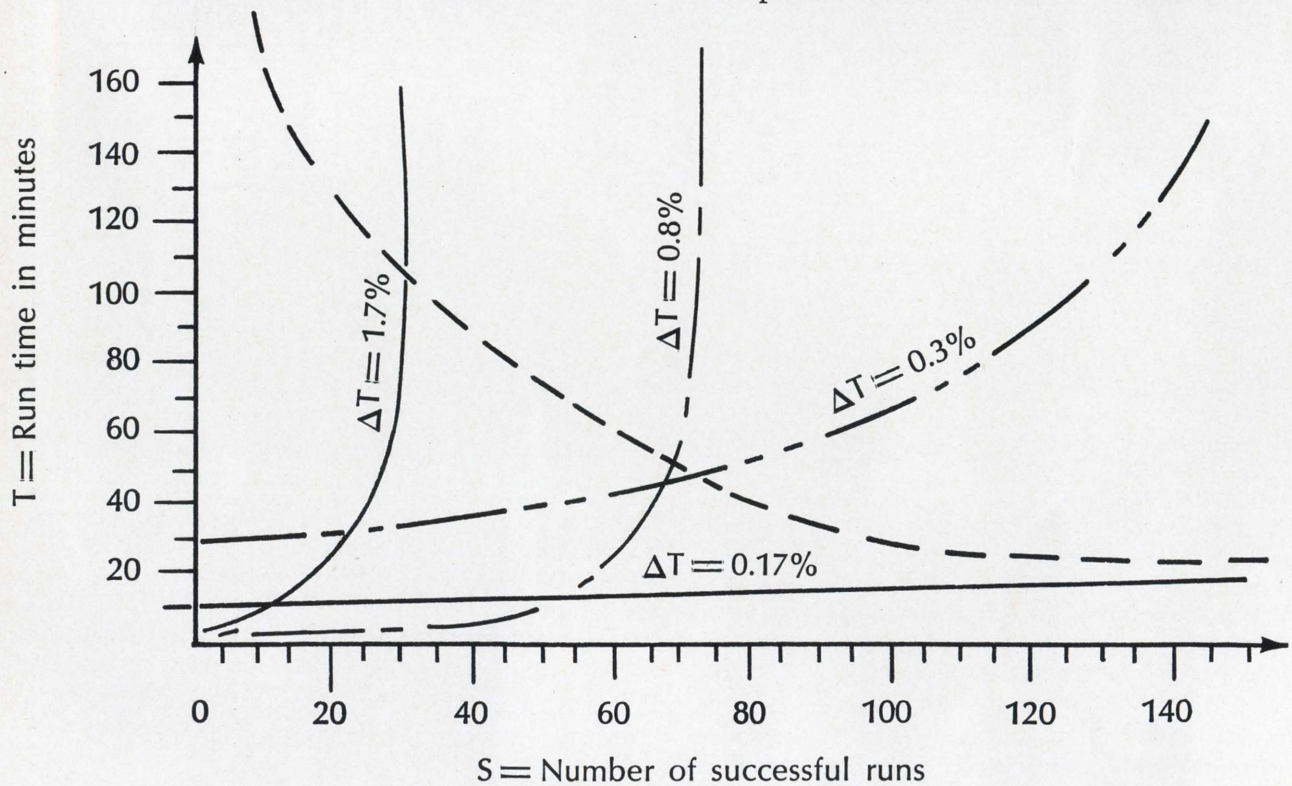


FIGURE 1

Graphs of equal CPU time invested in restarting a program with and without a checkpoint. The broken line represents S × T = 100 hours.

figuration can be freed for other use and the interrupted program resumed at some later time. In nearly all conditions which require a checkpoint, we note that the most "current" checkpoint is the only required one and that the most current checkpoint could have occurred an instant prior to the hardware failure. The only exception to using the most current checkpoint would be those failures that proceed undetected and cause the program to degenerate before the failure is detected, and the checkpoint is taken after the initial failure occurs.

CHECKPOINTING FREQUENCY

We have previously stated that a restartable checkpoint was based on the ability to reconstruct the input environment. All data is modified in stages. In the process of modification, these stages vary from the very minute changes which occur in a register during the process of a cascading binary addition, to the storing of a word in memory, to the completion of modification to an entire logical record, to the development of a new block of logical records in memory, and finally, to the writing of this block on an external device. Actually, the process can be thought of as continuing in increasingly broader stages to the modification of an entire file and the process of modification performed by the complete execution of an entire program or task, and can be carried on to include the modifications due to a week, or month, or year of running the program. In order to take a restartable checkpoint, we must consider the most opportune time; or, in other words, during which of these stages of modification should we take a checkpoint?

Practical Frequency—We noted previously the finality of modification of an input and indicated that during the modification of data in program storage there exists a copy on the external device; and it is not until the modified version is actually written, clobbering the input, that we cannot return to a previous input status. The suggestion was to use a high activity buffer to increase the time between the points of no return, and thus minimize the number of checkpoints that must be taken. But we have not yet defined what input data is significant. It is obvious a given program can read a logical record, clobber one word, and write out the logical record updating the input and this program can be validly restarted any number of times from any checkpoint. Thus we must define significant input, i.e., significant input is that input which will affect the outcome of the program. Thus, significant input must either be tested and affect control, or be executed. From this assumption, we can say little about the unknown problem program other than that significant input data must be in main storage to be tested or affect control, or be executed. Accepting this observation, we discover that data which is not yet read (or not yet written) is not significant to the program throughout that span of time until a read or write transpires. Therefore, any snapshot or program storage for this quiesced (no I/O in process) span of time can be considered a valid checkpoint only until another form of external action is begun. Since input is always available on an external device for refreshing memory, we conclude that the modification of input is the only point of no return and thus conclude that only write operations limit this peculiar span of time.

Recovery Time in Excess of Checkpoint Period—In further attempt to define the frequency and stage at which checkpoints should be taken, let us consider an I/O device failure. A request for I/O is queued for issuance by an I/O supervisor. The supervisor attempts the I/O and discovers the failure. If at this point the request is left queued and main storage is dumped, the program would be restartable. Permanent errors encountered which are due to physically unreadable input could not be overcome by a restart at this point with a new I/O device allocated.

The problem of unreadable input can only be cured by backing up to the point at which the unreadable input was written. This point is seldom spanned by the time period covered by the traditional forms of checkpoint. Normally this will require rerunning the entire program that originally wrote the bad output data since, even if the checkpoints are available, it is at best difficult to locate the checkpoint immediately prior to the particular bad write.

Main frame or CPU failures are subject to the same logical processes. If main storage and the registers could be captured at the occurrence of bad write (into main storage), then we would have recovery capability at the microscopic stage of modification.

But we have already observed that checkpointing before every write was too frequent and allowed us to recover only from the latest checkpoint in the general case. We also noted that for some causes of failure the latest checkpoint actually occurred after the actual failure, rendering the checkpoint useless. It is also logical to assume that, as checkpoints are taken more frequently, the likelihood increases that the last checkpoint will occur after the actual failure and prior to the point where the failure is detected.

The tendency is to want to take checkpoints very infrequently; perhaps every one-quarter hour, one-half hour, or hour, or even longer. Since most jobs are shorter than this, the checkpoint would have to be taken on the overall system or assume each job of shorter than one-half hour duration could be restarted from its beginning.

VALIDITY OF CHECKPOINTING

The question cannot be absolutely resolved. The answer is dependent on a great number of variables: lengths of typical runs, CPU overhead in taking a checkpoint (particularly significant if massive copies of blackboard storage are required), frequency of taking checkpoints, and protection for the "update-in-place" user. However, I hope to make one point clear—that the ability to recover must be available. The only question is how much time must be spent in the recovery operation. As has been indicated, generalized intermediate checkpoints have many deficiencies; they can definitely be uneconomical, and they do not guarantee recovery. The implication is that, if each job can be rerun from its beginning, this may oftentimes be the best solution. This ability to rerun must be guaranteed by both the operating system and the applications program. Indeed, the same objections for update-in-place or blackboard storage apply to the operating system.

Operating systems which maintain catalogs frequently do so by deliberately updating that which we have called "significant" input; i.e., it is used downstream. For very short duration tasks the system designer may desire to make copies of the "master file" only after several levels of modification amounting to hours of computer time. To recover, he must save all the input transaction files and reapply these in the same order to his copy of the "master file." In this case, many hours of time wasted in producing copies of a large "master file" can be saved. The same can apply to operating systems; however, the data bank or catalog may be operated on by many users and it may be undesirable to require rerunning of many jobs from various users to reinstate the master file.

CONCLUSION

Generalized intermediate checkpoints are not practical. The operating system that provides comprehensive user services and uses blackboard storage must protect itself and the user with some form of recovery ability probably widely separated to minimize long winded copying operations.

The individual applications programs are best off by adopting a specific recovery capability suited to the length of run and ratio of transaction file size to master file size. In short, the nature of a good economical checkpoint is application dependent.

If the preceding discussion was sufficiently rigorous, then perhaps clearer decisions can be made regarding recoverability, and perhaps prevent wandering attempts to find solutions to ill-defined problems. On the other hand, if there are weaknesses in the premises or conclusions reached, then we know those weak points to attack for a better solution.

personal lines

Dr. Roy C. Amara, Stanford Research Institute Vice President, has joined Dyna/Comp, Inc., as Systems Vice President . . . At Computer Response Corporation, Robert K. Rathbun has been named Systems Programming Project Manager; Robert R. Andrews, Computer Centers Manager; Morton L. Danuff, Manager of Technical Services; and William B. Joy, Technical Manager of the Pittsburgh branch office.

Joseph G. Brodnicki has been designated System Project Director and Donald P. Moffet Assistant for Business Planning at the Honeywell Information Services Division in Minneapolis . . . H. Fred Koehler and Robert N. Tullos have been named to newly created Chief Engineer posts with the Computer Products Division of Ampex Corporation, Koehler for core memory products, Tullos for tape memory products.

Delta Data Systems has named Charles Bradham as Installing Analyst in the Proprietary Systems Division . . . Joseph F. Bednar has been promoted to Director of Management Information Systems for the Equipment Group at TRW, assuming broader responsibilities for MI services for TRW plants in Cleveland and Minerva, Ohio, and Harrisburg and Danville, Pennsylvania, and technical responsibility for the divisional system design and computer operations groups.

J. D. Kee has been appointed General Manager for Customer Engineering Operations, International Liaison and Special Programs and George W. Rich has been named staff General Manager of Corporate Development for Control Data Corporation . . . Comprehensive Computers Systems, Inc. has designated Leonard A. Yardeni as Vice President of Management Services and Director of EDP Research and Development, which will include having charge of development of advanced computer technology and technique applications.

Richard A. De Lyser has been named Vice President for Facilities Management by Intech, Inc. . . . Ned J. Cooney has been elected Vice President for Data Services of Greyhound Computer Corporation . . . Computer Sciences Corporation has appointed Jerome J. Popkin as Director of Planning.

Richard N. Germano has been promoted to the newly created position of Vice President of Manufacturing Services Division of Applied Computer Sciences, Inc. The new division was formed to further concentrate the efforts in the design and installation of computer systems for manufacturing companies, especially in the areas of production and inventory control.

Francis Sinclair Webster III, has been elected Vice President in charge of the Technical Division of Computer Complex, Inc. . . . At Advance Data Corporation, a new computerized information service company, Peter J. Saia has been appointed Vice President of Computer Operations, to direct the time-sharing function of the company.

Kenneth W. Garr has been named Group Manager of Management Information Systems for Sperry Rand Corporation's Univac Federal Systems Division . . . Manfred Wildmann is the Manager of the newly created Terabit Memory Systems Department of Ampex Corporation . . . Louis C. Ray has been named Manager of Applications Development by Information International, where he will be responsible for the design of new applications for the company's line of programmable visual analysis equipment.

Delta Data Systems has named Walter McCown as Manager of Systems Implementation for its newly acquired subsidiary, Association Processing Corporation. From his Pennsylvania base, McCown will be responsible for implementing new-customer clubs on the membership accounting system, orienting club officials to the use of the system and converting the client's existing data.

COMMENTARY

COMMENTARY

The Fall Joint Computer Conference sponsored by the American Federation of Information Processing Societies (AFIPS), will be held in Las Vegas, Nevada from November 18–20 in the Las Vegas Convention Center. The original plans for the exhibit at the Convention Center called for 780 booths. However, because the space was sold out so quickly, an additional 200 10′ x 10′ units were blocked out at the Sahara Hotel. Currently 988 exhibit booths have been sold to 337 firms. When compared to the 427 booths manned by 174 Companies at the last AFIPS sponsored convention, we can note that there is an upsurge in exhibit activity among manufacturers of computers and computer peripherals. Samuel F. Needham has attributed part of the increase in booth sales to the 65 first-time exhibitors. (Concurrent with the general increase in booth sales, there is an increase in activity among producers of remote terminals for time-sharing.)

The FJCC also includes a 29 session technical program which will be described in more detail in the November issue of SOFTWARE AGE. Other highlights of the convention include: The Computer Science & Art Theatre, The Computer Art Exhibit, The Music Exhibit, and an all Conference Luncheon at which R. Buckminster Fuller will speak.

conference countdown

OCTOBER

17–18 Northeastern Regional Conference of the Association for Computing Machinery, Albany, N. Y. Contact: Dr. E. D. Reilly, Computer Science Dept., State University of New York at Albany, Albany, N. Y. 12203.

19–22 1969 National Conference of the American Records Management Association, St. Louis, Mo. Contact: ARMA National Headquarters, 24 N. Wabash Ave., Chicago, Ill. 60602.

24 Fourth Annual ACM Symposium on the Application of Computers to Problems of Urban Society, New York, N. Y. Contact: Jessica Hellwig, Computer Center, Columbia University, New York, N. Y.

26–30 Joint Conference on Mathematical and Computer Aids to Design, Anaheim, Calif. Contact: E. G. Kimme, Collins Radio Company, 19700 Jamboree Road, Newport Beach, Calif. 92663.

27–29 Data Processing Supplies Association's Fall Meeting, New York, N. Y. Contact: Data Processing Supplies Association, 1116 Summer Street, Stamford, Conn. 06905.

27–31 BEMA Annual Exposition & Conference, New York, N.Y. Contact: Paul Notari, BEMA, 235 E. 42nd St., New York, N.Y. 10017.

28–30 24th Annual Instrument Society of America (ISA) Conference & Exhibit. Contact: Ray Cooley & Associates, Inc., 4848 Guiton St., Houston, Texas 77027.

30–31 ADAPSO's 27th Management Conference, Atlanta, Georgia. Contact: Everett T. Suters, 551 Fifth Avenue, New York, N. Y. 10017.

NOVEMBER

10–11 Digitronics User Association's 4th Annual Conference, New York, N. Y. Contact: Mrs. Lyme Sawyer, Digitroics Users Association, Box 113, Albertson, N. Y. 11507.

17–18 Digital Equipment Computer Users Society Meeting, Las Vegas, Nevada. Contact: Angela Cossette, Digital Equipment Computer Users Society, Maynard, Massachusetts 01754.

17–19 1969 IEEE Symposium on Adaptive Processes (8th), Penn. State University, State College, Penn. Contact: Prof. G. J. McMurtry, Electrical Engineering Dept., Penn State University, University Park, Penn. 16802.

18–20 Fall Joint Computer Conference, Las Vegas, Nevada. Contact: AFIPS, 245 E. 47th St., New York, New York 10017.

19–20 ACUTE (Accountants Computer Users Technical Exchange) Conference, Atlanta, Georgia. Contact: ACUTE, 947 Old York Rd., Abington, Pa., 19001.

20–21 IEEE Computer Group Data Acquisition & Control Workshop Meeting, Las Vegas, Nevada. Contact: Dr. Albert Hopkins, MIT Instrument Lab., Station #35, 75 Cambridge Pkwy., Cambridge, Mass. 02142.

20–21 1969 Data Processing Conference, Empire Division (13), New York, N. Y. Contact: Registrar, Conference '69, P.O. Box 1926, Grand Central Station, New York, N. Y.

DECEMBER

1– 3 Second International Forum on "Data Entry", Hollywood, Florida. Contact: Robin B. Connors, International Business Forms Industries, 5223 River Road, Washington, D. C. 20016.

CHECKMATE

COMPUTERIZED CHESS

When did it start?

Shortly after World War II, several mathematicians from leading universities and industry, who were directly involved in the development of computers, started thinking about the possibility of computers playing chess. In 1949, Dr. Claude E. Shannon of Bell Telephone Laboratories presented a paper at the Institute of Radio Engineers entitled "Programming a Computer for Playing Chess". In 1950, Dr. Shannon built a machine which he designed merely to play a number of end games.

At first, the computer and its chess playing ability were widely misunderstood. It was assumed that the great speed with which a computer performs its calculations would enable it to analyze every move possible on the board. Much of the misunderstanding was due to exaggerations reported by lay reporters. When the layman is told that some computers execute over a million operations per second, it is easy to assume that such computers would analyze all possible combinations of a chess game within a few hours, or perhaps a few days.

How many different possibilities would the computer have to analyze? Well, why don't you try the following exercise:

> Take any middle game and count all legal moves White has for his next move. Then, count all legal moves for Black. On the average, your count will be 30 to 40 moves for White and the same number for Black. Now, combine each legal move of White with each legal move of Black, and what do you get? Over 1000 possible combinations on just one move! Two consecutive moves would lead to a million different positions, three moves to a billion, four moves to a trillion and so on, multiplying the number of possible variations by 1000 each time a move is made.

A good chess game between evenly matched players will average 40 to 50 moves. Since each new move generates 1000 possible combinations and each one must be combined with the 1000 variations of the next move, it is easy to see that 25 moves would generate a total number of 10^{75} (1 and 75 zeros) variations. If we assume that a game has 25 moves to go and a computer could analyze a million variations per second, it would take the computer 10^{69} seconds to decide which move to make next.

How long is 10^{60} seconds? Well, our planetary system is estimated to be 4½ billion years old, which in round numbers is 10^{18} seconds.

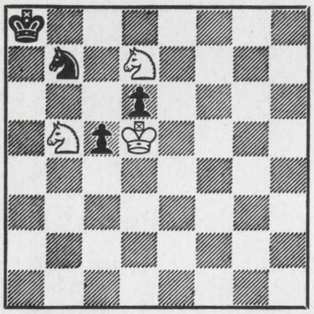
Question: If 10^{20} seconds is 100 times the life of our planetary system, how long is 10^{60} seconds?

Answer: Longer than any chess player would be willing to wait for a dumb computer to make up its mind.

P.S. More about Computerized Chess at a later date. Would you and your friends be interested in forming a Computerized Chess Club? The only purpose of such a club would be to establish lines of communication and exchange technical information. Please write to this author for additional details.

Problem 24:

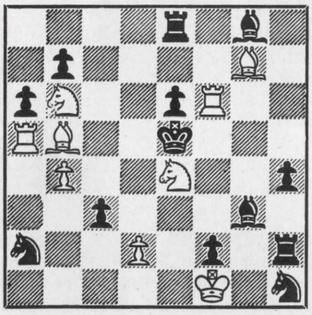
W. A. SHINKMAN



Mate in four moves.

Problem 25:

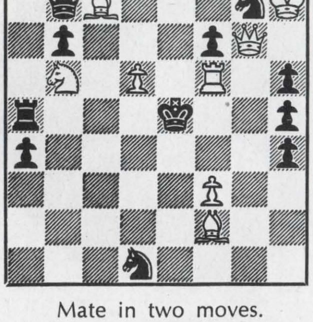
S. LOYD



Mate in three moves.

Problem 26:

W. KENNARD



Mate in two moves.

Solution to Problem 21

1 B-R8 KxR
2 K-N7 K-K4
3 K-N6 mate

Solution to Problem 22

1 N-B5 BxR
2 N-N7 B&N
3 BxB mate

Solution to Problem 23

1 P-Q7 (the only move to win)
1 ... K-B2
2 P-B8(Q)ch KxQ
3 0-0-0 ch K-any move
4 White takes Rook and wins.

SOFTWARE AGE

READERSHIP SURVEY

READERS:

Help us bring you editorial material that is in your specialty. Just fill out the coupon below and return it to us.

I already receive SOFTWARE AGE ☐ Yes ☐ No

Enter my subscription. ☐ Yes ☐ No

The information below gives my professional status.

Name ____________________

Company ____________________

Title ____________________

Home Address ____________________

City ____________________ State __________ Zip __________

Signature ____________________

Circle the most appropriate items below to describe your present position and experience.

MY SPECIALTY:
1—Systems Analysis
2—Math/Math Analysis
3—Operations Research
4—Scientific Programmer
5—Business Programmer
6—EDP Management
7—EDP Instructor
8—EDP Consultant
9—__________ other, incl. EDP student, operator, technician, etc.

PRIME EXPERIENCE IN:
1—Aerospace/Defense Industry
2—Non-Defense Industry
3—Commercial/Finance
4—Research Organization
5—Govt./Military
6—EDP Consulting
7—Transportation/Utilities
8—Education/Library
9—__________ other, specify

I WAS BORN:
A—1920 or before
B—1921–25
C—1926–30
D—1931–35
E—1936–40
F—1941–45
G—1946–50
H—1951–55

I HAVE:
☐ Technical Degree
☐ Non-Technical Degree
☐ No Degree

RENTAL CHARGE ON THE EQUIPMENT I WORK WITH:
☐ $1,500 or less, monthly
☐ $1,500–2,500
☐ $2,500–5,000
☐ $5,000–10,000
☐ $10,000–20,000
☐ over $20,000 monthly

Suggestions for SOFTWARE AGE

____

____

____

____

____

better books

by Dennie Van Tassel

THE ART OF COMPUTER PROGRAMMING volume II: Seminumerical Algorithms. By Donald E. Knuth. Reading, Massachusetts: Addison-Wesley Publishing Company, Inc. 1969. 624 pages. $18.50.

The author dedicates the first volume of this 7 volume series to the Type 650 computer "with whom I have spent many pleasant evenings"; moreover Knuth, the erudite Stanford author, includes excerpts and quotations from *McCall's Cookbook* and a Shakespearian play.

And in the preface of his first volume where he explains how he chose to write this profound series the author also says the process of preparing programs for a digital computer can—among other things —be an aesthetic experience much like composing music or poetry.

Nonetheless, this is an excellent reference textbook, the second volume in the series. The first volume was published last year and covers basic concepts and information structures. Each volume is an independent entity with the exception that the first volume does have some useful introductory material. The complete seven volume series is meant to provide a unified, readable, and theoretically sound knowledge concerning computer programming techniques, together with a study of their historical development. The prerequisites for useful reading of these texts is some machine language background, a little calculus, and a basic knowledge of programming techniques.

The second volume provides a very thorough coverage of random number theory and computer arithmetic. The section on random number covers techniques, statistical tests, random sequences, and the "meaning" of randomness.

The computer arithmetic section covers number systems, floating-point arithmetic, multiple-precision arithmetic, radix conversion, rational arithmetic, and polynomial arithmetic. Since this 600 page volume only covers two subjects one can guess at the completeness with which each subject is covered.

These volumes tend toward the technical and in fact are recommended for graduate students. I have used my copy several times and have found its usefulness unexcelled for comparing different approaches to a technical computing problem. One indication of a thorough and well researched book is the quality of its inclusions which includes here, an index, glossary, exercises and answers, an index of notation, and reference sources.

HOW COMPUTERS DO IT. By David Moursund. Belmont, California: Wadsworth Publishing Company. 124 pages. 1969. $3.00.

We know how birds and bees do it, how fish and fleas do it . . . But computers?

This 124-page book not only has a good title and cover but the rest of the text is as eye catching as the front cover. **HOW COMPUTERS DO IT** is designed to teach how to give a detailed step-by-step procedure for solving a computer problem—or, in other words—the logic or flowcharting of computer problems.

Because of its flow chart orientation, the small 6x9 paperbound book deals with problem analysis and algorithm development without focusing on a particular programming language.

financial currents

RCA is claiming a "long head start" in the explosively competitive and fast growing time-sharing market with its new large-scale Spectra 70/61.

Described by President Robert W. Sarnoff at the dedication of the company's new peripheral equipment plant at Marlboro, the 70/61 is designed to serve large users by handling bigger jobs at three times the speed of the 70/46. The more powerful Spectra supports 350 terminal users, largest number yet announced.

RCA expects the two systems together to capitalize on a shift in the market—advancing remote processing from a 13 percent share in 1968 to about 50 percent in 1973. Sarnoff also predicted that by 1975, an estimated 80 cents of the average dollar spent on systems will be for peripheral equipment such as manufactured at Marlboro.

* * *

Another strong note of optimism about time-sharing was sounded by Com-Share President Robert F. Guise, Jr., to the Cleveland Society of Security Analysts, where he supported predictions of the data service industry growing to the $10 billion mark by 1975, with time-sharing coming in for 10 to 50 per cent of the business.

Com-Share, an independent in time-sharing, increased its sales revenues by 400 per cent last year, but showed a net loss of $1,526,010—not unexpected, according to Guise.

Com-Share's management, he said, "made a conscious decision to forego profits in order to expand its operations and achieve the economies of scale, long-term growth, and long-term profitability that must be met if the company is to survive successfully in the long run. His company, he added, is now positioned to realize profitable growth.

* * *

Still on the time-sharing theme:

Computer Complex Inc., of Houston, will add a fourth computer to its time-sharing system—a $1 million SDS 940 from Scientific Data Systems. The new computer will go on-line during the fourth quarter of this year to handle the needs of customers, including clients in industry, government, science and education.

AL/COM made its bid for prominence in the time-sharing field when it activated the first of 20 Dual AL-10 computer systems scheduled for installation at Mathematics Park in Princeton, New Jersey by 1971.

In addition, AL/COM will test the centralized dual computer concept, which it pioneered, and in which pairs of matched computers in banks are interlinked to form a nucleus of computer power for a national network of users.

Each of the two PDP-10 units in a Dual AL-10 system—capable of serving up to 50 users simultaneously—backs up its mate and assumes its workload in case of a malfunction, and each AL-10 system backs up the next to assure reliability of time-sharing service.

* * *

In an attempt to expand the proprietary systems market, Delta Data Systems has introduced additional versions of the Delta General Ledger System for Burroughs and Honeywell computers. The System was originally designed for the IBM System/360.

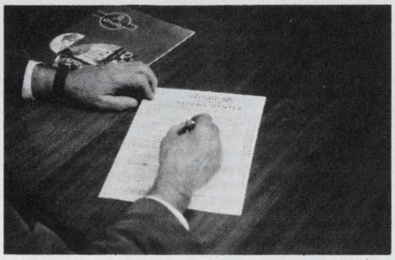
* * *

Computer Technology, Inc., is operating a $15 million "triplex" computer complex in Dallas—possibly one of the world's largest and most sophisticated systems—which integrates three large-scale central computers, five smaller remote computers and more than 225 remote terminal units.

Aimed at "unparalleled commercial and scientific data processing capabilities" for the company's clients in the Southwest, the system each week processes some 4,000 remote-entry jobs while simultaneously handling more than 500 major business applications in a multi-programming time-sharing mode.

Can an EDP man attending FJCC find that better job without letting the world know he's looking? SURE HE CAN!

SOFTWARE AGE Magazine's FREE Resume Center protects your privacy . . . speeds your search . . . costs you nothing!



1. Fill out the special resume form from magazine or at Center.

2. Our Remington operator will photocopy the resume free and return *all* copies to you.

3. You place your resume in the company lock boxes of your choice.

Your job is simple. You follow the same procedure at the center that you do in S/A—but on a highly accelerated basis. Fill out the special resume form at the S/A RESUME CENTER. Our Remington operators will provide you with as many *free* photo copies of your resume as you require.

You personally deposit them in the lock boxes of those companies which interest you. The resume form includes blanks where you can enter your hotel, telephone number and room number. Only the participating company representative has access to the lockbox. At his convenience, he can contact you at your hotel to arrange an interview.

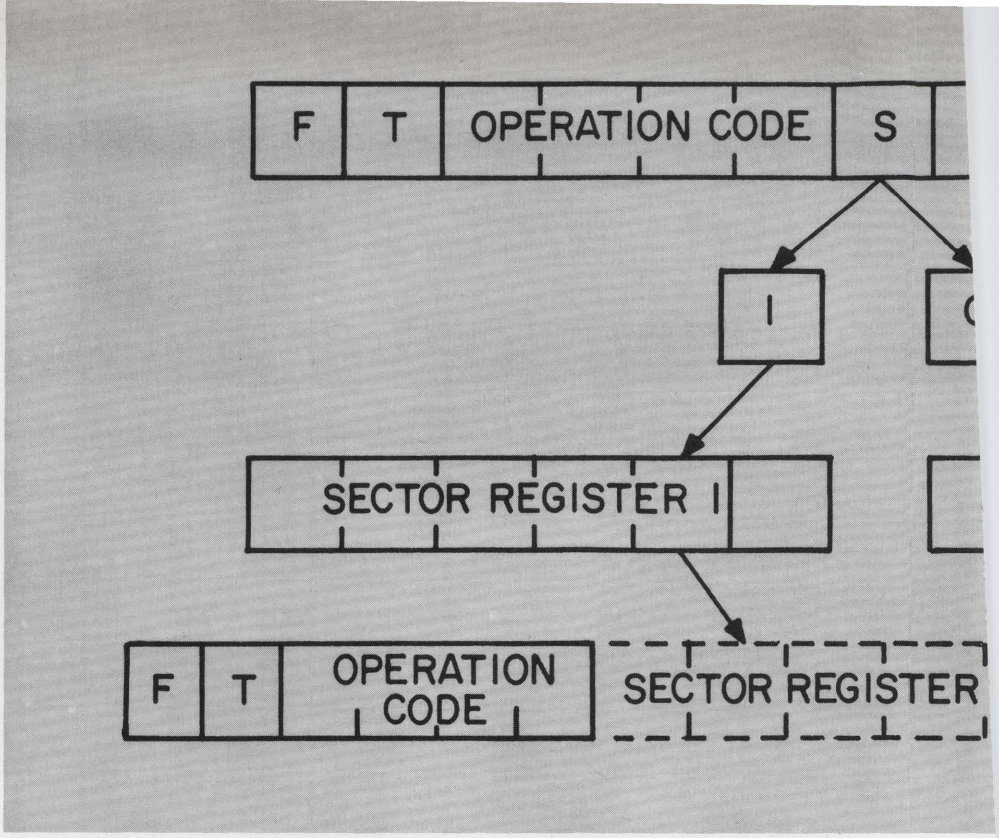
The entire operation of the S/A RESUME CENTER is designed to provide speed, convenience, simplicity and maximum confidence with no middlemen involved. You determine exactly the companies you want—no others will see your resume or know of your interests.

The S/A RESUME CENTER is FREE. There is no charge to you or the company for this valuable service. The resume center is sponsored solely by the advertisers in SOFTWARE AGE magazine.

software age
resume center

El Dorado Room of the Frontier Hotel
Las Vegas, Nevada
9 A.M. to 7 P.M., Wednesday, November 19th.
9 A.M. to 7 P.M., Thursday, November 20th.

SOFTWARE AGE Magazine, 2211 Fordem Ave., Madison, Wis. 53701 David W. French, Publisher



ADDRESSING CO
SECTOR ORIENT

by Jame
Honeywell, Co
Divi

Introduction

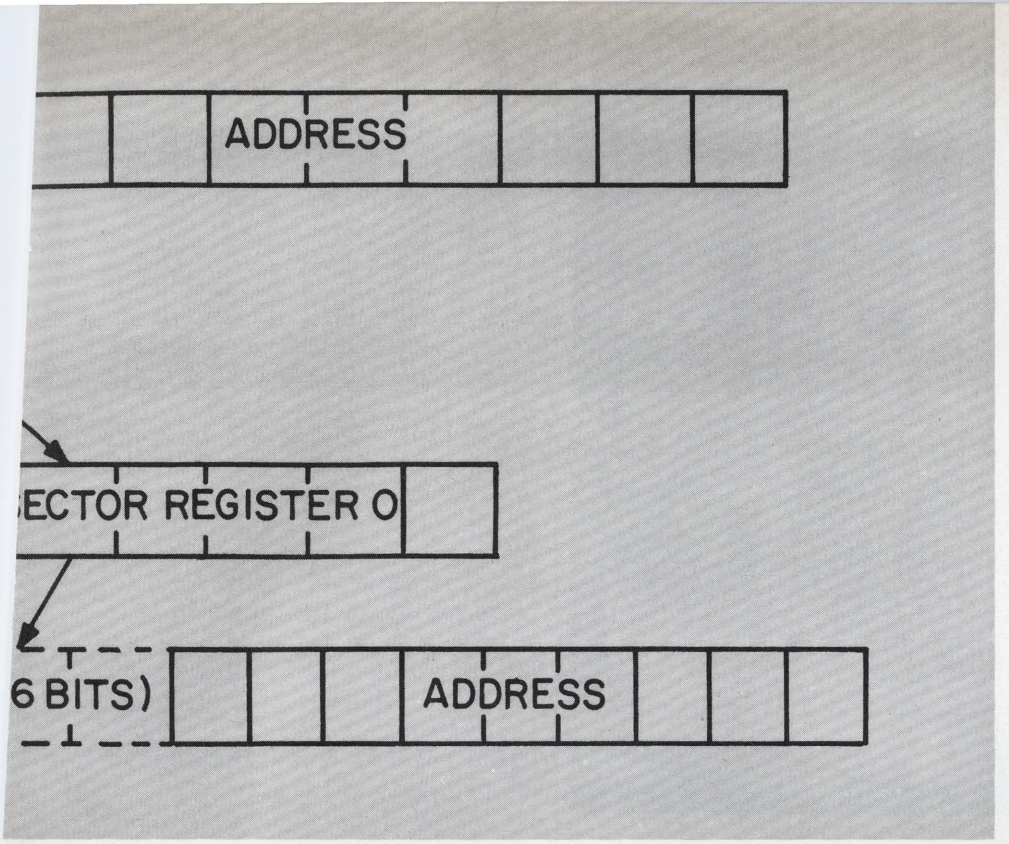
To achieve more effective addressing in sector-oriented computers, we need to consider several basic questions. Can the use of more complex software reduce the housekeeping burden to a tolerable level? Can more efficient use be made of the available address bits in an instruction word?

The *typical* data processed by a computer are operation codes and addresses. In the usual single-address, word-oriented computer, a typical *word* consists of an operation code and an address (Figure 1). There may be some bits reserved for use as modifiers of either the operation code or the address. The number of bits required to specify the desired information may exceed the number available in the machine word. How can the available number of bits be increased?

Sector orientation is an attempt to get more mileage out of those bits reserved for address information. Some of the address bits are supplied from one or more hardware registers. The problem is thus reduced to supplying values to these registers. The bits available for use as an address in the instruction word are now used to specify (1) a location relative to a sector, and (2) a register in which the sector address may be found. Very few bits are generally used to specify the latter. Typically, one is sufficient, giving a choice between two sectors.

Indirect Addressing

Words which do not contain operation codes may contain more address information (Figure 2). Thus, an indirect reference (where the address specified in the instruction word is insufficient to identify



NSIDERATIONS IN TED COMPUTERS

s Butler
mputer Control
sion

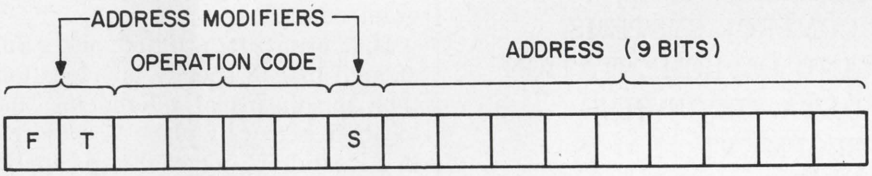


Figure 1. Typical 16-Bit Machine Instruction Word

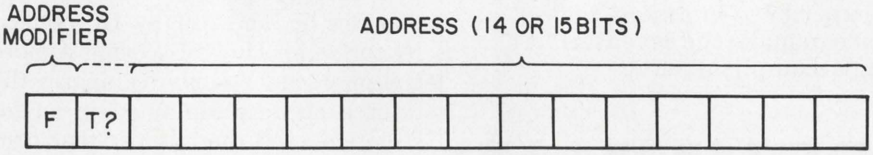


Figure 2. Indirect Address Word

the desired location) is usually sufficient to specify where the address of the desired location may be found.

Most (if not all) sector-oriented computers have a sufficient number of bits in their words to specify any address available in the machine, if none of those bits is used up in operation code or modifiers. Sometimes the numbers is *barely* sufficient. The sector specification in non-indirect addresses is from one or more hardware registers which may be fixed or variable. In the latter case, it may simply track the program register or it may be set under program control. The choice of register will be made by one or two bits in the instruction word. Typically, a single bit is used to distinguish whether the sector reference is from one of two registers (Figure 3). Two bits would enable a choice from one of four registers, but using an additional bit in the instruction word would reduce the number of addressable words within the sector by a factor of two.

Fixed Registers

A typical sector-oriented computer may have a fixed register, probably with value zero, and a register which is tracking the program counter. The result is a choice of reference to "sector zero" or "base sector" and the sector in which the instruction is located. A common variant is setting the value of register containing the "base sector address" under program control.

If a computer word contains enough bits to specify any location, then the means of referencing that location is to (1) place the address in a location in one of the specifiable sectors, and (2) let the instruction indirectly reference the address through the specified sector. By this means, we can operate upon any memory location.

This scheme works, but with a considerable housekeeping burden placed on the programmer. The burden can be, and usually is, reduced by software. How? By using a more complex loader which compares the desired address contained in an instruction with the address that contains the instruction. If they match, then an algorithm that sets the sector reference to "own sector" is used. If they do not, then an algorithm

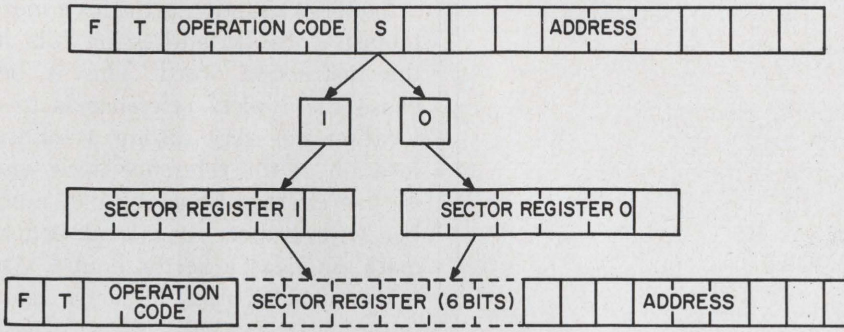


Figure 3. Address Formation by Sector Orientation

that sets the reference to the desired sector is used, if that is possible. If this is not possible, then an indirect reference is generated by placing the whole address in a location which can be indirectly referenced. These three schemes are for addresses in "own" sector, "base" sector, and "cross sector reference", respectively.

Housekeeping

Additional housekeeping may now be needed in the case of "cross sector reference." If address modification is required, indexing or indirect, for example, then the modifying information must be correctly placed. This may present a problem if indirect addresses have modifier capabilities other than direct addresses. At this point, with the machine design, the effect of indexing before or after indirect becomes important, as does the effect of limiting indirect addresses to any specific number of levels. These considerations are important to the extent they affect the number of bits left in a word to specify address, and how many bits are used in specifying modifiers of addresses. The choices are varied, and depend upon the design goals selected for the hardware and the software. The additional housekeeping required of the loader means the size of the loader is larger than in a nonsector-oriented machine with a large word size. Thus, the larger number of words for a given amount of core is partially nullified.

Second, the greater number of memory accesses, caused by the increased number of indirect references, will slow down the effective speed of the computer. This latter effect is sneaky. It shows up only in the big programs which use a lot of core, and not on the small jobs which fit entirely within a sector.

There is truly a balance in word size and a number of words in computers. The more memory there is, the larger the word size should be —if the increased memory size is for the purpose of running larger problems with more far-ranging memory accesses. If the increased memory is for running concurrent small programs, then sector-orientation and small word size do not give these disadvantages.

Hence, there appears a need for varying designs, since the ultimate use of the machine has a direct influence on the design of such a basic choice as word size. And this considers instructions and addresses as the only interesting data contained in a word! Add the effects governed by mathematical precision requirements and ease of character manipulation, and it is easy to see why there are so many different kinds of computers. There isn't a best computer, there is only a "best computer for my application."

Why Sector Orientation?

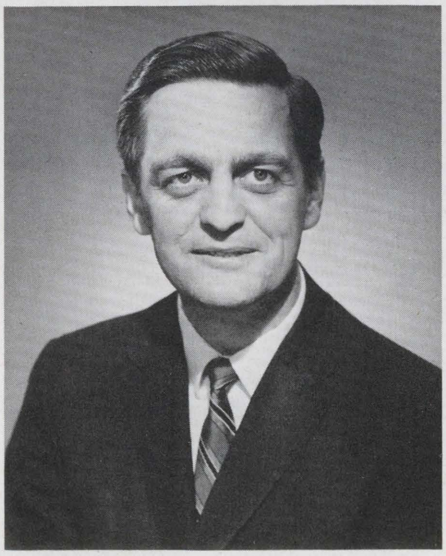
Sector orientation provides a more intensive use of addressing bits in the instruction word. This is because the *typical* reference is to a location not very distant from the location of the reference itself, and is thus expressible as a small number with respect to a local benchmark, such as a sector bound. Another choice, used in some computers, carries this a step further, and allows addresses to be "relative forward" or "relative backward." The disadvantage, of course, is that additional bits now must be used to determine the choice of the addressing algorithm. There are many possible address selection algorithms. The more addresses the machine may have, the more algorithms are possible. The smaller the word size, the more algorithms are necessary. Address bits are used to select the address algorithm. The simplest of these work well for the machines of largest word size. This is simply to give the whole address in the instruction word. Address modification by indexing was the first complexity added to address algorithms. Next came indirect addressing. This became the basis which made sector-oriented machines practical. Sectors are a form of addressing relative to a set of fixed bounds.

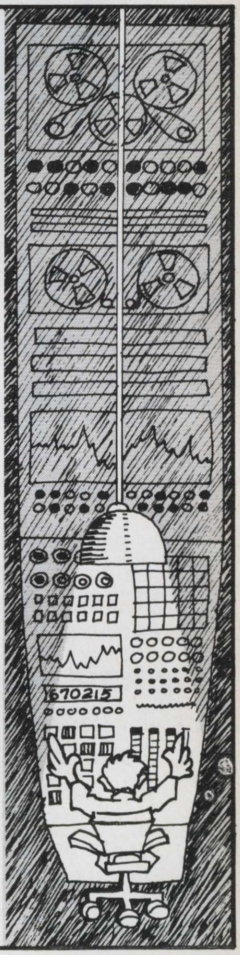
One of the problems of sector-oriented machines, which require the loader to do the housekeeping, is that the locations of the sector bounds relative to the program may have a considerable effect on the program. First, it is impossible to move an instruction containing a memory reference from one location to another, since the memory reference is a function of the location of the instruction. Second, the speed is a function of the number of memory accesses, which may vary with the order of loading of subprograms, due to indirect cross-sector references produced by the loader.

For the above reasons, it becomes clear more care must be exercised in programming for a sector-oriented computer; more complex software is required to keep the housekeeping burden to a tolerable level; and efficient use should be made of the available address bits in an instruction word. ■

James W. Butler, a senior principal programmer, heads a quality assurance group for software at Honeywell, Computer Control Division, in Framingham, Mass. With Honeywell for the past seven years, he was engaged in process control programming before coming to the Computer Control Division two and a half years ago. Prior to joining this company, Mr. Butler was with Goodyear Aerospace for 11 years in positions from development engineer through computer facility manager. His experience covers both the analog and digital computer fields.

A lieutenant commander in the Naval Air Reserve, Mr. Butler holds a B.A. in mathematics from Ohio Wesleyan University (1949) and a M.A. in mathematics from Kent State University (1951). He has served as an instructor in programming for company-sponsored classes at both Honeywell and Goodyear.





TROUBLE-TRAN PRESENTS XTRAN'S ADVENTURES IN FORTRAN

By GEORGE N. VASSILAKIS

Send your ANSWER to the problems posed here in each issue to:

TROUBLE-TRAN EDITOR

software age

P. O. Box 2076
2211 Fordem Ave., Madison, Wis. 53701

You can also profit by submitting PROBLEMS for this feature. If your problem in FORTRAN programming is selected for use in this feature, you will receive $50.00

TROUBLE-TRAN'S Objectives:

1. To have fun.
2. To promote USA Standard FORTRAN by pointing out differences and inconsistencies of existing FORTRAN Compilers.
3. To alert programmers to the physical limitations of hardware.

PROBLEM OF THE MONTH

Problem 18: By D. A. Kapusta, Canonsburg, Pa.

Here is an interesting problem which may surprise some of you.

```
      I =100.
      F=-I+100
      Z=SGN(F)
      WRITE(6,10) Z
   10 FØRMAT(F8.2)
      STØP
      END
```

```
      FUNCTIØN SGN(I)
      IF(I)10,20,30
   10 SGN=-1.0
      GØ TØ 40
   20 SGN=0.0
      GØ TØ 40
   30 SGN=+1.0
   40 RETURN
      END
```

What is the value of Z?

Answer to Problem 15

We have a winner! Correction, two winners!

Congratulations to Michael D. Kelley and Gary Goodman of Stanford University.

Lack of space and time prevent the publication of the winning solution. However, I will be glad to send a listing to those who send me a self addressed stamped envelope.

The winning program had 32,050 operations, and it was one of the few entries that had the correct count. Obviously, those of you who submitted smaller estimates had overlooked a few operations, or had the wrong count. Entries which came close to winning were submitted by K. Williams, Kalamazoo, Mich.; C. Moler, University of Michigan; K. G. Slane, Rocky River, Ohio; and R. J. Baumel, Princeton, N. J.

One reader who solved this problem in less than ten statements had used the following rules:

1. All odd numbers less than or equal to 7 are prime numbers.
2. Above 7, any odd number which is not evently divisible by 3,5, or 7 is a prime number.

The problem was not that simple. If your list of prime numbers includes 121, 169, and other numbers whose square root is a prime number, you have probably used the same wrong rules.

All entries with estimates under 50,000 were keypunched and run. Many entries had incorrect counts in nested DO loops. Even though it had not been stated in the rules. a DO statement was counted as two operations (one add and one compare). Operations needed for formating and printing output were not counted.

In general, the "sieve" method was the best.

Answer to Problem 16

The problem was to find out what values of I would be printed by the following program:

```
      DØ 10 I=1, 10
      I=I+1
      WRITE(6,5) I
    5 FØRMAT(I5)
   10 CØNTINUE
      STØP
```

Here are the results:

1. Over 90% of the present FORTRAN compilers *DO NOT* give a diagnostic and the program is executed with I=2,4,6,8,10. The following systems are included in this category:
   - IBM 360 FORTRAN compilers
   - CDC 6400, 6500, 6600
   - GE 200, 400, MARK II
   - B5500
   - SDS 940, SIGMA 5
   - SPECTRA 70/46
   - HONEYWELL 1648
   - Purdue University (PUFFT) on the IBM 7094/7040
2. Same results were obtained from many time-sharing systems using the BASIC language.
3. UNIVAC 1108 FORTRAN V gave a diagnostic and then executed printing values of I = 2,3,4,5,6,7,8,9,10.
4. IBM 7094, 7044 and other second generation systems give the same results as the UNIVAC 1108, but NO DIAGNOSTICS.
5. A time-sharing system on a CDC 3300 gave the following diagnostic: "ERROR 1046 AT STATEMENT 2
   THE RUNNING INDEX IN A DO *MAY* BE CHANGED WITHIN THE LOOP".
6. PDP/10 FORTRAN V executed with I=1,3,5,7,9.

SIMPLIST/70

An Elementary List Processing Language

John David Canter
Point Park College
Pittsburgh, Pennsylvania

SIMPLIST/70 is an elementary list processing language designed to facilitate the manipulating of information in both simple list and complex list structure formats. It reduces the time normally required to master a list processing language while maintaining the flexibilities inherent in list techniques, including dynamic allocation of storage, hierarchical data structures, recursion and garbage collection. SIMPLIST/70 is general purpose in that it includes the capabilities of both symbolic and arithmetic processing. The language consists of twenty-five macros for the creating, processing, analyzing and destroying of form-free lists, and for logical decision making and program control. The basic element of SIMPLIST/70 is a quadri-cell which contains a data element and FROM, TO and HIERARCHY address pointers. A SIMPLIST/70 compiler has been written in Assembler Language for the IBM System/360, and requires a minimum of 32K bytes for operation.

The design of the language is explored, and the function and application of each of the twenty-five macros is examined in detail.

(continued from last month)

The macros encompassed by the SIMPLIST/70 system are organized into four general groups. These groups are: 1. input/output operations; 2. arithmetic functions; 3. general manipulative routines; 4. logical operations.

There are two general macros for accomplishing the inputting and outputting of information in list form in the SIMPLIST/70 language. The input instruction is called OBTAIN, and is written as:

OBTAIN LISTA

where LISTA is the name to be assigned to the list be inputted. This macro takes information from an input source and stores it on a character-by-character or byte-by-byte basis in the currently available memory cells. Additionally, it creates a constant for the address of the list, stores the address for future reference, and zeroes the FROM portion of the HEAD cell and the TO portion of the TAIL cell of the list. Thus, a three-item list called ARRAY, with elements A, B and C, processed as

OBTAIN ARRAY

might lead to the following:

A	0	976	@

10

B	10	1442	@

976

C	976	0	@

1442

The output macro is called DISPATCH and is written as:

DISPATCH LISTA

where LISTA is the name of the list to be written. This macro prints, on a line by line basis, the total contents of the list cited by the DISPATCH instruction, including associated sub-lists if a list structure is being processed. Either 120 or 132 characters maximum per line may be outputted (user selectable) with as many lines printed as is necessary to exhaust the entire contents of the list being written.

There are four macros in SIMPLIST/70 which provide for arithmetic manipulation of lists. The macros are designed to permit the adding of one list to another, the subtracting of one list from another, the multiplying of one list by another, and the dividing of one list into another. The four arithmetic macros are recursive in nature, and process all appended list structure elements.

The addition macro is known as ADD and is written as:

ADD LISTA,LISTB,ZERO,LESSTHAN, GREATER

in which the contents of LISTA are added algebraically to the contents of LISTB. The result of this arithmetic operation resides at LISTB, with LISTA remaining unchanged. If LISTA and/or LISTB refer to list structures in place of simple lists, all sub-list values of the structured lists will be recursively processed, and the final result will reflect the values of all elements of the structure. The size of the result should be no larger than the number of positions available in the receiving list. When the result of the addition is zero, as in the adding of like values with unlike signs, the program automatically branches to the address designated symbolically by ZERO. If the ADD results in a negative value, the program will automatically branch to the address indicated symbolically by LESSTHAN. If the result yields a positive value, then the program will automatically branch to the instruction represented symbolically by GREATER. The user may substitute in ADD macro any conditional branch to names he desires, so long as he defines those names as symbolic instruction addresses somewhere in the program. Thus, if LISTA, with a value of 123, is to be added to LISTB, with a value of —123, at the conclusion of this operation, LISTA will retain its 123 value, LITB will contain 000, and the program will branch to the ZERO routine. The branch to ZERO (or to a user-supplied symbolic address in the first address position) will occur because of its positional equivalence to and dependence upon a ZERO result.

The macro for subtraction is very much similar to the macro for addition, with the exception that the word SUBTRACT is substituted for the word ADD. Thus, the macro is written as:

```
SUBTRACT   LISTA,LISTB,ZERO,
           LESSTHAN,GREATER
```

In this macro, LISTA is subtracted from LISTB, the result is stored in LISTB, and LISTA remains unchanged. As is true with ADD, recursion will be performed if list structures are being processed, and the size of the result should be no larger than the number of positions in the receiving list. Conditional branching is as described under the ADD macro. If LISTA, contents 150, is subtracted from LISTB, value 100, the result of —50 will be reflected in LISTB, and the program will branch to the LESSTHAN routine. This is because the result is a negative value, and LESSTHAN (or any other user-supplied symbolic address in the second address location) will point to the to-be-executed-next address on a negative answer.

The macro for multiplication is called MULTIPLY and is written as:

```
MULTIPLY   LISTA,LISTB,ZERO,
           LESSTHAN,GREATER
```

in which LISTB, the multiplicand, is multiplied by LISTA, the multiplier, and the result or product resides in LISTB. LISTA remains unchanged, and conditional branching and recursion are as indicated under the ADD macro description. If LISTB, contents 0097, is multiplied by LISTA, contents 13, the resulting value of 1261 will be placed in LISTB, and LISTA will continue to reflect its original value of 13. Branching will be to the GREATER or third position address because of the positive (greater than zero) result. The two high-order zeroes preceding the 97 in the LISTB value are required so that the receiving field will be large enough to accept the result.

Division is accomplished through the utilization of the DIVIDE macro. This macro is written as:

```
DIVIDE   LISTA,LISTB,ZERO,
         LESSTHAN,GREATER
```

LISTA, the divisor, is divided into LISTB, the dividend, and the quotient replaces LISTB. No reminder is maintained. LISTA remains unchanged, and recursion, size and conditional branching are as before. If, for example, LISTA (407) were divided into LISTB (13946), the quotient of 34 would be stored in LISTB, and the remainder of 108 would be lost.

The general manipulation macros of the SIMPLIST/70 system permit the user to copy lists, add items to lists, delete items from lists, move items from one list to another, destroy entire lists no longer required, and create complex list structures from single lists.

The macro for copying lists is known as CARBON and is written as:

```
CARBON   LISTA,LISTB
```

LISTA refers to the name of the list to be copied. LISTB refers to the name to be assigned to the duplicate of the list being copied. At the completion of this instruction, LISTA not only resides in its original location, and with its original name, but also exists in duplicate form at LISTB with the name as symbolically designated by LISTB.

The SCRUB macro is one which permits the user to destroy an entire list of information when he no longer has need for that list of information. As indicated earlier, the return of cells to the AVAILABLE category is done automatically as a by-product of certain of the macro instruction. The SCRUB macro, which is written as

```
SCRUB   LISTA
```

provides the programmer with additional delete facilities. This macro operates recursively, erasing all members of the list or list structure under consideration, and returning each of the cells to the AVAILABLE category.

There is one macro which is designed for removing the first item from a list, and one for removing the last item from a list. The macro which destroys the first or HEAD cell is known as CAPITATE and is written as

```
CAPITATE   LISTA
```

This macro removes the HEAD cell from the desired list, returns that cell to AVAILABLE, finds the next cell in the list, and makes that cell the new HEAD cell. In making the next cell the HEAD, the system stores the address of that cell as the name, and sets the FROM area in the quadri-cell to zero. Thus, if a list originally containing ABCDE were subjected to the CAPITATE macro, it would then appear as BCDE, and the location of the "B" would become the list's new HEAD cell.

The instruction which removes the last cell from a given list is known as TRUNCATE and is written as

```
TRUNCATE   LISTA
```

where LISTA refers symbolically to the list whose last member is no longer desired. This instruction removes the TAIL cell and returns it to AVAILABLE. Additionally, it makes the cell previous to the one just removed the end point on the list by inserting a value of zero in the TO portion of the quadri-cell. If a list whose original contents were "12347Q$" were truncated, the result would be "12347Q" and the TO address of the "Q" cell would be set to zero.

As there are macros for removing the first and/or last of a list to the SIMPLIST/70 system provides macros for inserting and/or deleting individual characters from any position on a specified list.

The macro for inserting a character into a list is entitled GRAFT and is written as

GRAFT LISTA,POSITION,CHARACTER

LISTA is the symbolic name of the list into which the character is to be inserted. POSITION designates, in absolute decimal form, the location into which the character is to be placed. CHARACTER refers to any legitimate computer character, again in absolute form, which is to be inserted into the desired position in the selected list. Thus, for example, the code for inserting a "$" into the fourth position of a list entitled INPUT would be

GRAFT INPUT,4,$

The macro which permits the copying of a single data element from one list to another is known as MOVEONE. This macro, which is written as

MOVEONE LISTA,LISTB

takes the currently accessible character in LISTA, searches LISTB for an available cell, and moves the single character from LISTA to LISTB. In order to find available cells in an already constructed list, the system searches for a data element of "!". Thus, if the MOVEONE macro is to be utilized, the list into which elements from another list are to be inserted must be initialized with "!" data elements where desired. If, in LISTA, with data elements $QA147E, the current position contains the data element of "A", and if LISTB contains elements 439X!L7!9, at the completion of

MOVEONE LISTA,LISTB

LISTB contains 439XAL7!9.

The macro which is designed to remove a given character, on a positional basis, from a desired list is entitled SCALPEL and is written as

SCALPEL LISTA,POSITION

LISTA refers symbolically to the list from which the character at the location indicated by the symbolic POSITION, in absolute decimal form, is to be removed. If it were desired to remove the fourteenth member of a list named ALPHA, whose contents prior to the SCALPEL operation were ABCDEFGHIJKLMNOPQR, the instruction to accomplish this would be

SCALPEL ALPHA,14

At the conclusion of this instruction, ALPHA would now contain ABCDEFGHIJKLMOPQR.

The STAMPOUT macro is provided to permit deletion of the currently accessible location from a given list. It is written as

STAMPOUT LISTA

where LISTA refers to the list, the current location of which is to be deleted and returned to AVAILABLE. Thus if, in a list of elements AB*14769, the currently available cell contains the "7", and STAMPOUT is issued, the resultant list will contain AB*1469.

APPEND is used to concatenate and physically join the members of one list to the members of another. It is written as

APPEND LISTA,LISTB

in which the elements of the list symbolically referred to as LISTB are attached to the end of the list symbolically referred to as LISTA. At the completion of this macro, if one were to refer to LISTA, he would obtain not only LISTA's contents but the contents of LISTB as well. Additionally, LISTB retains its individual addressability. If LISTA originally contained 123, and LISTB contained ABC, at the conclusion of an

APPEND LISTA,LISTB

operation, LISTA would now contain 123ABC, in that sequence, and LISTB would retain its original value of ABC.

While the APPEND macro has the ability to join a multitude of diverse lists into one integrated data structure, the resulting list is nonetheless mono-dimensional or linear in nature. In order to create the complex list structures referred to in other sections of this paper, SIMPLIST/70 provides the ATTACH macro. The function of this command is to join one list to another, in somewhat the same vein as the APPEND macro. Unlike APPEND, however, which effects horizontal relationships, ATTACH creates junctures resulting in vertical or prime-list-to-sub-list hierarchies. The difference lies in the fact that the APPEND macro joins lists through the TO and FROM address pointers, while the ATTACH macro joins lists through the TO and HIERARCHY pointers. The macro is written as:

ATTACH LISTA,LISTB,POSITION

in which LISTA is the name of the list to which LISTB is to be hierarchically joined, and POSITION refers in absolute decimal form to the relative location of the LISTA element where the attaching is to occur.

As an example, consider the following hypothesis:

LISTA contains "CANINEBOVINEFELINE"
LISTB contains "DOG"
LISTC contains "POODLE"
LISTD contains "TOY"
LISTE contains "COW"
LISTF contains "GUERNSEY"
LISTG contains "CAT"
LISTH contains "SIAMESE"

It is desired to create an hierarchical structure in which the sub-classifications of the main list are to be arranged in a generic-to-specific categorization. The SIMPLIST/70 code to accomplish this is:

```
ATTACH   LISTA,LISTB,006
ATTACH   LISTB,LISTC,003
ATTACH   LISTC,LISTD,006
ATTACH   LISTA,LISTE,012
ATTACH   LISTE,LISTF,003
ATTACH   LISTA,LISTG,018
ATTACH   LISTG,LISTH,003
```

As a result of executing the above instructions, a list structure of

```
CANINE      BOVINE      FELINE
  DOG          COW          CAT
    POODLE     GUERNSEY     SIAMESE
      TOY
```

would be created. This resulting structure is now recursively processible.

There are four macros in SIMPLIST/70 which permit the programmer to manually adjust the communications address cell to point to the HEAD of a list or the TAIL of a list, or for pushing down or popping up lists when using LIFO (last-in-first-out) or FIFO (first-in-first-out) list processing. In LIFO processing, items are both added to and removed from the HEAD of a list, and an item placed in a list immediately before the current item may not be accessed until the current item is removed. In FIFO processing, items are added to the TAIL of a list and removed from the HEAD. Thus, in FIFO manipulations, list items are accessible in the same sequence in which they were placed in the list.

The first of the SIMPLIST/70 address manipulation macros is known as BEFORE, and is written as

BEFORE LISTA

This macro obtains the address of the character in the list immediately preceding the character currently being accessed, and places this address in the communications address cell of that list. Thus, if in the following list the character "9" at address location 4096 were currently accessible, and a BEFORE macro were issued, the LISTA communications pointer would be set at 1108, the address of the prior cell in the list.

4	0	4096	@

1108

9	1108	3600	@

4096

6	4096	0	@

3600

The AFTER macro is written as

AFTER LISTA

and is the antithesis of the BEFORE macro. AFTER obtains the address of the character immediately following the one currently accessible, and places that address in the list's communications address cell. In the preceding example, if an AFTER had been issued in place of the BEFORE macro, the LISTA communications pointer would have been set to 3600, the address of the next cell in the list.

The macro which points the communications address cell to the HEAD cell of a list is called HEAD, and is written as

HEAD LISTA

That which points to the TAIL of a list is called TAIL, and is written

TAIL LISTA

Forward processing from the HEAD of a list may be controlled through the use of the previously described AFTER macro, while reverse tracing from the TAIL of a list may be effected through the use of the BEFORE instruction.

The last of the major units of macros which the SIMPLIST/70 programming language provides are those of logical operations and decision making. The system makes available five macros for the scanning of lists in search of given characters, the comparing of two lists for similarities and/or dissimilarities, and for unconditional branching. One of these five macros provides two variant forms.

The macro which permits the searching of a given list or list structure for a particular character is known as SCAN. It is written

SCAN LISTA,CHARACTER,MATCH

in which the list (LISTA) is scanned for the data element in its absolute form (CHARACTER). If the character being sought is found anywhere in the list being analyzed, including attached hierarchical sub-lists, then the program branches automatically to the MATCH symbolic address. If not, the program executes the next sequential instruction. In addition, if a match is found, the system stores a "1" in its SCAN result cell, and the cell address where the match is effected is stored in the SCAN communications address cell. If an unequal or nonmatch condition results, then a "0" is stored in the SCAN result cell, and zeroes are stored in the resultant address field.

The purpose of the STOPLOOK macro is also to analyze a given list for a particular character. Unlike SCAN, however, which searches an entire list structure for the desired character, STOPLOOK analyzes only the currently addressable cell in the desired list. The instruction is written as follows:

STOPLOOK LISTA,CHARACTER,
UNEQUAL,EQUAL

LISTA refers to the list the currently available cell of which is to be analyzed. CHARACTER refers to a single character, in absolute form, which is to be matched against the position currently available in the selected list. If the character in the STOPLOOK instruction matches the position under evaluation, then SIMPLIST/70 will automatically branch to the instruction symbolically represented by EQUAL. If, however, the position under evaluation does not match the character as designated in the STOPLOOK macro, then the program will branch to the UNEQUAL routine. As an example, suppose the currently available cell of a list called LISTZ contains a "?". If the following instruction were executed

STOPLOOK LISTZ,?,ONE,TWO

the program would branch to TWO, the symbolic routine for an equal or match condition. Conversely, if the character did not match, i.e., the currently active position of LISTZ did not contain a "?", then the program would branch to the routine entitled ONE, the unequal or nonmatch address.

A variant form of the STOPLOOK macro provides the ability to compare the currently active position of one list with the currently active position of a second list. The macro is written as

STOPLOOK LISTA,LISTB,UNEQUAL,
EQUAL

in which the current data element of LISTA is compared to the current data element of LISTB. If the lists at their currently addressable single positions are equal, the program automatically branches to the address symbolically represented by EQUAL. Else, the program branches to the UNEQUAL routine.

In addition to the single character match which the SCAN macro provides, and to the previous positional match for which the STOP-LOOK macro is designed, SIMPLIST/70 also includes an instruction for nonpositional comparison of all the entries in one list against all the entries in another. This macro, which is known as FLOAT, is written

FLOAT LISTA,LISTB

With this instruction, all entries in LISTA are compared nonpositionally with all entries in LISTB. LISTA and LISTB may both refer to list structures, in which case the program will conduct a recursive scanning. If a character in LISTA matches a character anywhere in LISTB, then the instruction will store a "1" in a result list in the position corresponding to the location of the LISTA element currently active. If, however, a character in LISTA matches no character in LISTB, then this instruction will enter a "0" into the corresponding position of the result list. At the conclusion of this macro, the result list will consist of 0's and 1's corresponding to the characters in LISTA which do and do not match characters in LISTB respectively. The length of LISTA and LISTB need not be the same size, and may be variable. For example, LISTA consists of elements AD14738 and LISTB consists of elements 963C28A93B. As a result of issuing

FLOAT LISTA,LISTB

a resultant list would automatically be created by the program. This list will, in the example being considered, consist of "100001" because the first, sixth and seventh positions of LISTA match elements (not necessarily positionally) in LISTB. Thus, both lists contain data elements A, 3, and 8, and since these data elements reside in positions one, six and seven of LISTA, the result list contains 1's in these positions and 0's in all others.

Having conducted a pattern matching, information retrieval or other similar operation with the FLOAT macro, it is now possible to determine whether the results were as desired. In order to do this, SIMPLIST/70 provides the MASK instruction, which is written as

MASK LISTA,EQUAL

LISTA is the name of the pattern field which the programmer creates in advance. This pattern field is comprised of 0's and 1's corresponding to the desired pattern resulting from the application of the FLOAT macro. The user-generated pattern field may be either a simple list, or may consist of an hierarchical list structure. The MASK instruction automatically compares, on a positional and one-for-one basis, its LISTA pattern with the program-generated pattern list occurring as output from the last executed FLOAT macro. If the pattern field as designated by LISTA in the MASK instruction matches completely, on a positional basis, the result list which was automatically produced by the FLOAT macro, then the program will branch to the address as designated by the symbolic EQUAL. Otherwise, the next sequential instruction will be executed. As indicated, the user may build an hierarchical list structure of permissible masks, in which case SIMPLIST/70 will process the acceptable patterns recursively against the resultant list.

In referring to the preceding example, the FLOAT macro, as we have seen, generated a resultant pattern list of "1000011." If the programmer were to then issue an instruction of

MASK LISTA,HIT

in which LISTA contains the desired pattern or mask, the program will branch to the symbolic HIT address if pattern and result are identical. Thus, if LISTA refers to a pattern of 1000011, HIT will be executed. If LISTA, for the current example, refers to anything other than 1000011, and there is no hierarchical structure to be recursively evaluated, then a non-hit will result, and the next sequential instruction will be executed.

The final instruction which SIMPLIST/70 makes available to its users is an unconditional branch. It is known as POINT, and is written as

POINT ADDRESS

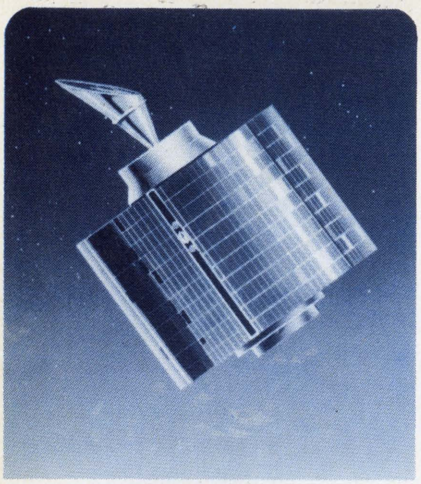
This macro generates an unconditional branch to the instruction whose name is symbolically represented in the address field.

While SIMPLIST/70 has a limited repertoire of twenty-five macros, it nonetheless provides the basic and essential elements of a full list processing language, including dynamic allocation of memory, hierarchical data structures, recursion and garbage collection. Written in Assembler Language for the IBM System/360, it is applicable to a wide range of today's third generation computer systems.

BIBLIOGRAPHY

1. Dupchak, Robert. *TIPL: Teach Information Processing Language.* The RAND Corporation, RM-3879-PR (October 1963).
2. Farber, D. J. et al. "The SNOBOL3 Programming Language." *Bell System Tech. J.* (July–August 1966).
3. Feigenbaum, Edward A. ed. *Computers and Thought.* McGraw–Hill Book Company, N. Y. (1963).
4. Foster, J. M. *List Processing.* American Elsevier Publishing Company, Inc., New York, N. Y. (1968).
5. Gelernter, H., Hansen, J. R., and Gerberich, C. L. "A FORTRAN-Compiled List-Processing Language." *Journal of the ACM.* (April 1960), 87–101.
6. Harris, Edward H. *STRIGOL—A String Oriented Language.* New Mexico State University, University Park, New Mexico (September 1967).
7. Hays, David G. *Introduction to Computational Linguistics.* American Elsevier Publishing Company, Inc., New York, N. Y. (1967) 27–31.
8. Higman, Bryan. *A Comparative Study of Programming Languages.* American Elsevier Publishing Company, Inc., New York, N. Y. (1968) 18–22, 95–101.
9. McCarty, J. et al. *LISP 1.5 Programmer's Manual.* MIT Press, Cambridge, Mass. (1962).
10. Meadow, Charles T. *The Analysis of Information Systems.* John Wiley and Sons, Inc., New York, N. Y. (1967) 198–202.
11. Newell, A. ed. *Information Processing Language-V Manual.* Prentice Hall, Englewood Cliffs, N. J. 2nd edition (1963).
12. Raphael, B. *Survey of Computer Languages for Symbolic and Algebraic Manipulations.* Stanford Research Institute, Menlo Park, California (March 1967).
13. Salton, Gerard. *Automatic Information Organization and Retrieval.* McGraw–Hill Book Company, New York, N. Y. (1968) 97–105.
14. Teitelman, Warren. *Design and Implementation of FLIP, A LISP Format Directed List Processor.* Bolt, Beranek, and Newman, Inc., Cambridge, Mass. (July, 1967).
15. Vallee, Jacques F. *Search Strategies and Retrieval Language.* Northwestern University, Evanston, Illinois (1967).
16. Yngve, V. H. *COMIT Programmers' Reference Manual.* The M.I.T. Press (June 1962).

John Canter graduated in 1960 from University of Pittsburgh, Phi Beta Kappa. He is currently Chairman of the Department of Information Science and Director of the Computer Center at Point Park College, which will offer the first B.S. degree in Information Science this year. He previously worked with the IBM Corporation as an instructor in unit record and computer systems, served as director of the computer center of a multi-state retail drug chain, and served as Assistant Director of the University of Pittsburgh's Knowledge Availability Systems Center.



new products

Sperry Rand Corporation's Univac Division has announced the UNIVAC 8414 Disk Subsystem, **the latest model in its 8400 series of direct access storage subsystems.** The UNIVAC 8414 has an access time averaging 60 milliseconds. The basic 8414 Subsystem begins with two drives, which provide over 58 million bytes of on-line storage, when used with 9000 series computer systems. This may be expanded, in increments of 29,176,000, to eight drives providing a total capacity in excess of 233 million bytes. Standard features of the 8414 include record overflow—permitting record sizes larger than one track in length, and file scan—providing the ability to search both the data and key areas of the record.

The disk pack used with the 8414 is composed of a stack of 11 disks providing 20 recording surfaces. There are 200 useable tracks per surface with 7204 byte positions per track. The 8414 access mechanism consists of a group of 20 arms with one read/write and erase head for each of the disk surfaces on which data is stored. In operation, the mechanism moves heads horizontally from one position to another without returning to a home position. Although the UNIVAC 8414 was initially announced for use with the UNIVAC 9000 series computers, it will be available later for the 1100 series computers.

For more information, circle No. 13 on the Reader Service Card

The Institute of Advanced Technology, a subsidiary of Central Computer Corporation of Anaheim, California, has developed **a new multi-media approach to computer training.** The technique was designed to instruct students in several computer programming languages, computer operation, and keypunching. It incorporates a SYSTEM multi-media learning carrel, plus fully-integrated audio/visual materials, manuals, and programmed textbooks.

Typically, a particular concept is first presented to the student via a stereo headset. Then he receives instructions to view several 35mm slides that are projected onto a screen before him. Finally, he is directed to read programmed materials that reinforce his understanding. Throughout the presentation, the student responds to questions in a coordinated workbook. Some of the assets of the technique are that the students may proceed at their own rate of learning, at their own convenience and for the amount of time they have available.

Management oriented courses are also offered to provide an introduction to data processing principles, data communication and systems analysis. The Technique was developed to help fulfill the growing need for personnel qualified in the various aspects of the Data Processing field.

For more information, circle No. 14 on the Reader Service Card

A new electronic Data Sorter, designed to free digital computers from the time consuming task of sorting, has been announced by Astrodata, Inc. of Anaheim, California. The new Data Sorter, the Model 1561, is an on-line peripheral device which stores data internally, sorts it, and then returns it to the computer. The only computer time required by the sorter is the time to place the data in the Sorter and to remove it after the sort is complete.

The Data Sorter can be used to do sorting jobs which use magnetic tapes, disk files, high speed printers and other computer peripherals for data source or destination. The capacity of the unit is 65,536 words of approximately 40 bytes each and data records of up to 5,437 bytes in length may be stored.

Although the time required to sort a file will generally depend on the number of records to be sorted and on any preordering of the particular file being used, the Sorter is said to do the sorting portion of any computer program faster than most computers can without using a comparable amount of computer time.

For more information, circle No. 15 on the Reader Service Card

CompuScan Model 370, is a new **optical character recognition system** that is said to be the world's first system with unlimited character recognition capabilities. It was primarily designed to convert vast amounts of printed or typed data into machine usable form. The model 370 learns fonts directly from the material being scanned—even materials with intermixed type faces and symbols—and converts microfilm images of the text directly to magnetic tape.

The CompuScan Model 370 consists of 5 basic elements:

(1) A Precision Optical Scanner, a flexible high speed scanner, which has been described as the electronic *eye* of the system. This high resolution flying spot scanner makes accurate differential density measurements on microfilm images of the input data.

(2) A Precision Film Tranport which presents the microfilm (16 mm or 35 mm) input data to the scanner. Up to 15,000 pages of text can be stored on one roll of film.

(3) A General Purpose Computer which sequences and controls all phases of the OCR operation, including periodic calibrations of the scanner system on a completely automatic basis. It accepts information from the scanner film transport and display unit in the foreground while providing flexible page formatting, including logical decisions as a result of the text read, in the background. The computer can output to magnetic tape, punch cards or to the display.

(4) A Recognition Unit which recognizes all characters and symbols. Special purpose circuitry within the unit provides high speed comparison of unknown characters with the reference alphabet.

(5) A Display Unit which is a large screen TV type device which displays single characters, lines of text, video images or digitized images or graphical data. It permits the operator to enter unrecognized characters, to intervene during the acquisition phase for new fonts via light pen, or to intervene during digitizing of graphical data.

The CompuScan Model 370 will initially be offered through a service bureau operation. Cost and time factors, on a per character of per page basis, are expected to be a fraction of current OCR or key punch processing of comparable data.

For more information, circle No. 16 on the Reader Service Card

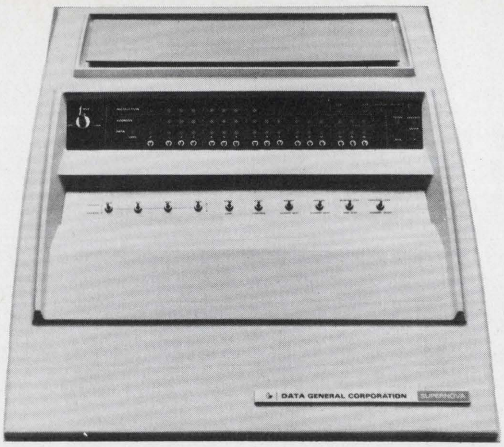
InveCom Systems is a new company engaged in the manufacture of state-of-the-art computer peripherals for inventory and related problems. One of their new products is an **Automatic Inventory Acquisition System,** the System 1000, which is said to be able to cut inventory costs by 60%. The System consists of a portable optical reader and processor that weighs 3¾ pounds and is battery operated. It gives the operator the option of optically reading digital in-coded property tags as well as manual input of information through the use of either digital switches or a 10-key keyboard. The information, after being checked and formated by the processor, is then recorded serially on magnetic tape. Each tape cassette may contain up to 6,000 unit entries.

Some possible applications include property control of capital items such as office furniture, capital inventory of warehouse stock, utility meter reading, hospital drug control, vending machine counter reading, security control, etc.

For more information, circle No. 17 on the Reader Service Card

TRW has entered the commercial software market with the introduction of a new product, called **GIM (Generalized Information Management)**. GIM is a computer based information management system offering rapid, simple and precise data selection. The machine independent, computer-based, general purpose system can be used for storage, maintenance and retrieval of information. GIM meets needs in a wide variety of business research, Government, and education applications. One of GIM's features is that the user can converse with the computer in English. Robert Muchmore, vice-president and general manager of the Software and Information Systems Division, said GIM is the first of a series of software products to be made commercially available by TRW during the coming year.

For more information, circle No. 18 on the Reader Service Card



Data General Corporation of Southboro, Mass., has announced their second computer, **the Supernova, a small-scale, general purpose computer.** The Supernova is said to be the fastest of all the *mini* computers. Its add time, using conventional core memory, is 800 nanoseconds. Using read-only memory (ROM), which is interchangeable with core, add time is 300 nanoseconds—one machine cycle. Because Supernova is able to overlap the *fetch* and *execute* portions of its arithmetic and logical instructions while operating from high-speed ROM, these instructions can be fully executed in one 300 nanosecond cycle. This is supposed to make the effective operating speed of Supernova considerable faster than that of conventional small computers.

A basic Supernova configuration has 4096 words of 16-bit core memory. High-speed read-only memory may be interchanged with core, and the two types of memory can be mixed. Programs developed with the 800 nanosecond core memory can also be used with the 300 nanosecond read-only memory. Supernova is completely compatible with Nova, Data General's first computer, and any programs developed for Nova will run on Supernova.

The desk top model shown *above* contains facilities for expansion of core memory, plus interfaces for input/output devices, including Teletype, magnetic tape units, disks, card readers, line printers, or cathode-ray-tube displays. Space devoted to I/O interfaces and memory may be swapped within the basic configuration. Expanded versions of the Supernova can be configured to contain up to 32,768 16-bit words of core and up to 64 input/output devices.

For more information, circle No. 19 on the Reader Service Card

For more information, circle No. 9 on the Reader Service Card

S/A market place

where you can reach 130,000 programmers, mathematicians, analysts and EDP managers

RATES

Classified—75¢ per word, minimum order $7.50. Blind ads are an additional $10.00. Payment must accompany copy.

Display Classified—One inch by one column, $75.00. Column width is 2¼". Help wanted and employment services ads will be accepted in the classified section. Ads of ⅛ page or more will be listed in the Advertisers' Index, and will qualify for the free resume service.

Closing Date—the 10th of the month preceding (i.e. May issue closes April 10th). Send order and remittance to:

software age
Classified Department
P. O. Box 2076, Madison, Wisconsin 53701

EMPLOYMENT AND SEARCH COMPANIES

PROGRAMMERS (3)
Large financial firm needs (1) Senior, salary to $28,000; (2) Intermediates, salaries $16,000 to $23,000 to help open new company locations. Entire fee, traveling, if necessary—moving expenses are all paid by your new employer.
ERIN EXECUTIVE AGENCY
132 W. 34 St., New York, N. Y. 10001
Phone: (212) 594–8296

DATA PROCESSING POSITIONS IN THE SOUTHWEST
Contact:
DATA PROCESSING CAREERS
Richard Kemmerly, Suite 1109, Stemmons Towers West, Dallas, Texas 75207. 214/637–6360.

PRODUCTS

ADD / SUBTRACT 6-DIGIT HEXADECIMALS in seconds with 100% accuracy. The pocket HEXADDER, $15. Free brochure from HEXCO, Dept. SA, P. O. Box 55588, Houston, Texas 77055.

WANTED: Used 20 Drawer Tab Card Files. MacDonald, Box 68–501, Miami, Fla. 33168.

PROFESSIONAL SERVICES

EDP RESUMES
A comprehensive manual on how to write your own resume, written especially for EDP personnel. Send $4.25 to
K & H ASSOCIATES
Dept. S, P. O. Box 265
Horseheads, New York 14845

SAMIS / 3G—The Structural Analysis and Matrix Interpretive System installed on third generation computers. $5,000.00 includes revised documentation, source and object decks, warranted program performance. IBM 360/50–65–75, UNIVAC 1108, CDC 6400–6500–6600. **INTERNATIONAL ANALYTICS INC., 14937 Ventura Blvd., Suite 305, Sherman Oaks, Calif. 91403. Phone (213) 783–1441.**

HELP WANTED

PROGRAMMER/ANALYST needed by Syracuse University Library. Familiarity with IBM 360 computer operating under OS/360. Languages: Assembly and/or PL/1. Background and experience in commercial programming desirable. Will work on problems involved in interfacing a variety of input/output devices and large file organization. Apply: **Mrs. Elizabeth S. Newlove, Syracuse University Library, Syracuse, New York 13210.**

CLASSIFIED ADVERTISING ORDER FORM

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25
26	27	28	29	30
31	32	33	34	35

______Words = $______

☐ Blind Ad $10.00 Insert______time(s) Total Enclosed $______

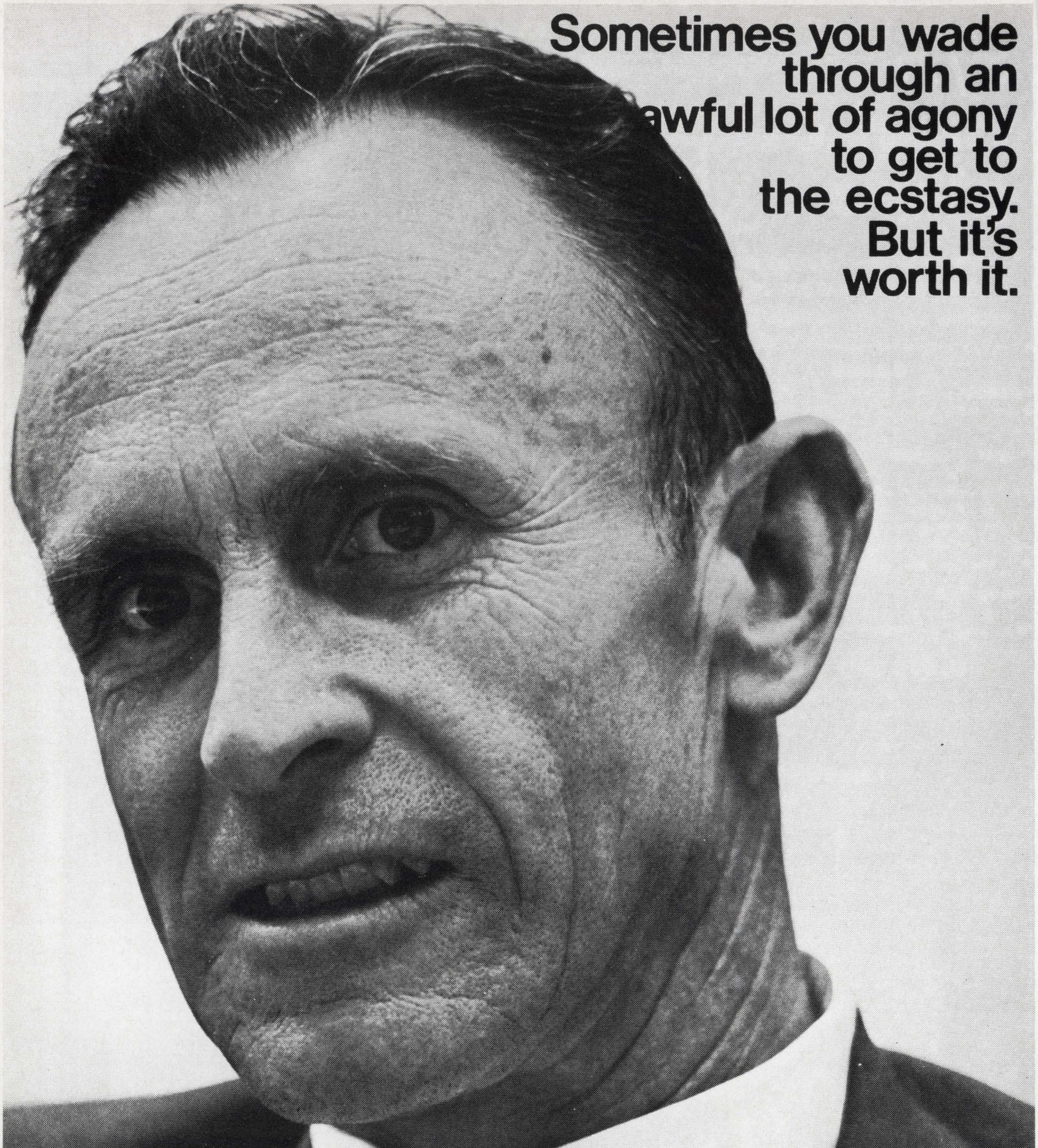
NAME______

ADDRESS______

CITY______ STATE______ ZIP______

Signature______

WORD COUNT: Includes name and address (unless blind ad). The name of the city (Des Moines), state (New York) and zip code are counted as a total of two (2) words. Each abbreviation, initial, single figure or group of figures or letters count as a word. Symbols such as 35mm, COD, AC, etc. count as a single word. Hyphenated words count as two (2) words.



The gut-feeling of value received: you've got the best of nine possible worlds. Plus the chance to examine the other eight. So if you're reaching for the ecstasy, we can help.

Right now openings exist in: Spacecraft Structural Design · Temperature Control Materials & Processing · Application of Microelectronics & Transistors · Guidance & Control Systems · Electro Optics · Propulsion Systems Analysis · Space Vehicles Design · Trajectory Design & System Analysis · Systems Design & Integration · Deep Space Support Systems · Electromechanical Spacecraft Instrumentation · Scientific Programming · TV Image Processing and Electronic Packaging.

Send your resume to:
JET PROPULSION LABORATORY
California Institute of Technology
4806 Oak Grove Drive,
Pasadena, California 91103
Attn: Professional Staffing Dept. 10

"An equal opportunity employer." Jet Propulsion Laboratory is operated by the California Institute of Technology for the National Aeronautics and Space Administration.

software age

CONFIDENTIAL INQUIRY

Your original copy of this form will be retained at the offices of SOFTWARE AGE and will be used for no other purpose than to notify the specific firms which you have checked (on the reverse side) of your interest.

THIS INQUIRY IS IN DIRECT RESPONSE TO YOUR ADVERTISEMENT IN SOFTWARE AGE MAGAZINE

TYPE OR PRINT CLEARLY FOR PHOTO REPRODUCTION

JOB DESIRED: ____________________

List computer hardware knowledge (names of systems, tape, disk, terminals, etc.): ____________________

Programming specialties and years of experience (commercial, scientific, theoretical, experimental, analog, etc.): ____________________

Systems programming on which you have had development experience (compilers, assemblers, executives, monitors, O.S., etc. Indicate for what computer): ____________________

Programming languages used and extent of experience (COBOL, FORTRAN, etc.): ____________________

Applications programmed (aerospace, banking, insurance, math subroutines, compilers, etc.): ____________________

Systems analysis experience (card design, flow charting, operation analysis, etc.): ____________________

EDP management experience (include years and number of people reporting to you): ____________________

SALARY: __________ (current) __________ (desired) DATE OF AVAILABILITY: __________

EDUCATION: Indicate major as well as degree unless self-explanatory.

Degrees			
Years			
Schools			

EMPLOYMENT: Indicate present employment and previous jobs below.

Employer			
City			
Years	to	to	to
Title or Function			

Name __________ Signature __________

Home Address __________ (city) __________ Home Phone __________

__________ (state) __________ (ZIP code) Age __________ U. S. Citizen? __________

Security Clearance __________ Location Preference __________

Marital Status __________

Military Status __________

BE SURE YOU HAVE CHECKED ON REVERSE SIDE THE COMPANIES YOU WANT TO SEE THIS INQUIRY. PUT FORM IN STAMPED ENVELOPE AND MAIL TO:

software age
MAGAZINE
P. O. BOX 2076
2211 FORDEM AVE., MADISON, WIS. 53701

check your interests here

Fill in the confidential inquiry form on the other side of this sheet. This form provides all the information advertisers require to screen applicants. If further information is desired, you will hear from the advertiser direct. Then, check below the boxes of those companies to which you want copies of your form sent. Mail to SOFTWARE AGE, P.O. Box 2076, 2211 Fordem Avenue, Madison, Wisconsin 53701. (**Please do not send us your own resume. We will only process this form. A new form must be filled out for each issue in which you are answering ads.**)

EMPLOYMENT AND SEARCH AGENCIES

PRODUCTS AND SERVICES

(Use Reader Service Card)

☐ I do not now receive S/A. Please send me information on how I may receive a FREE subscription.

....
Name

....
Street Address

....
City State Zip Code

software age

P. O. Box 2076
2211 Fordem Avenue
Madison, Wisconsin 53701