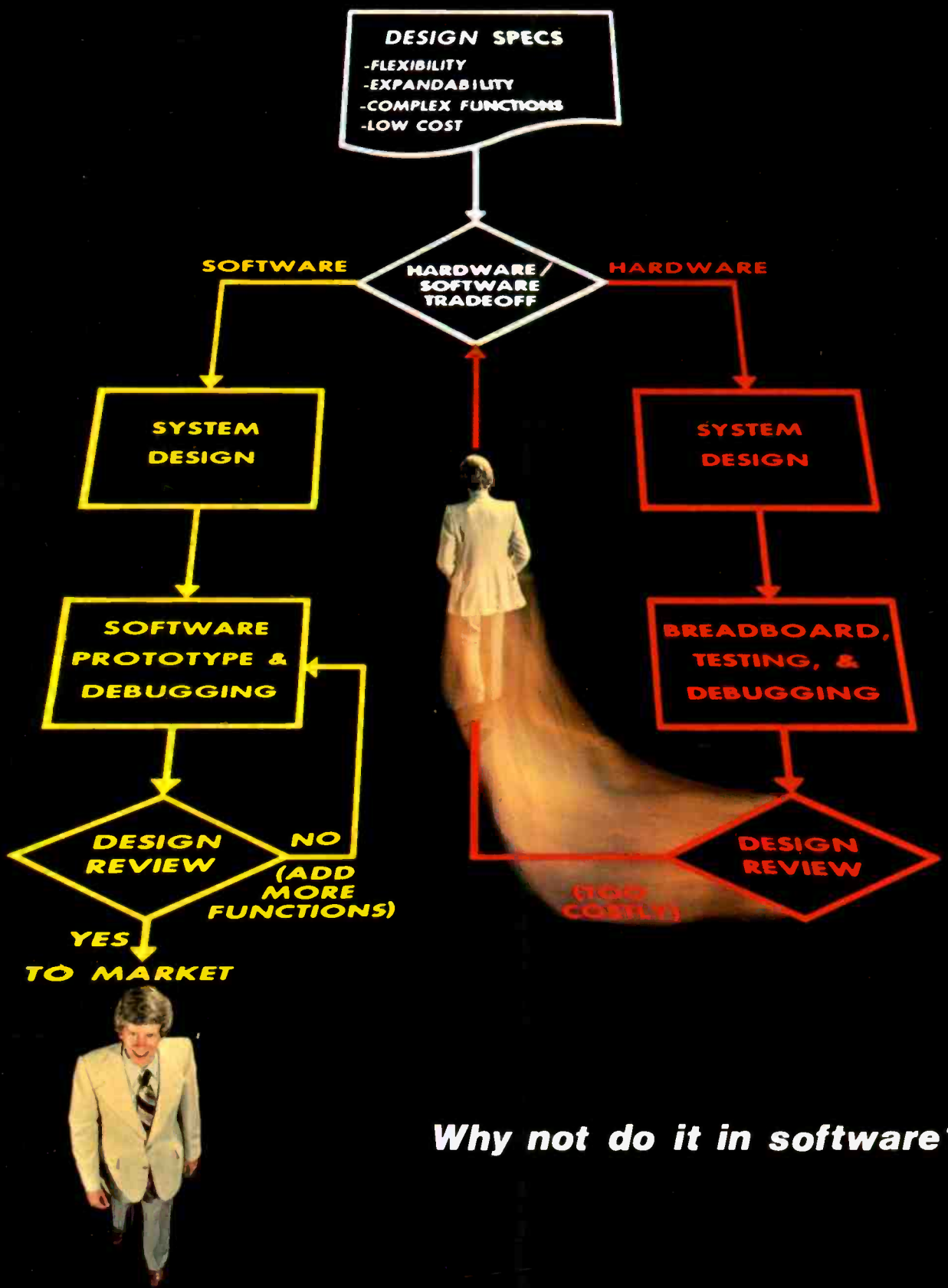
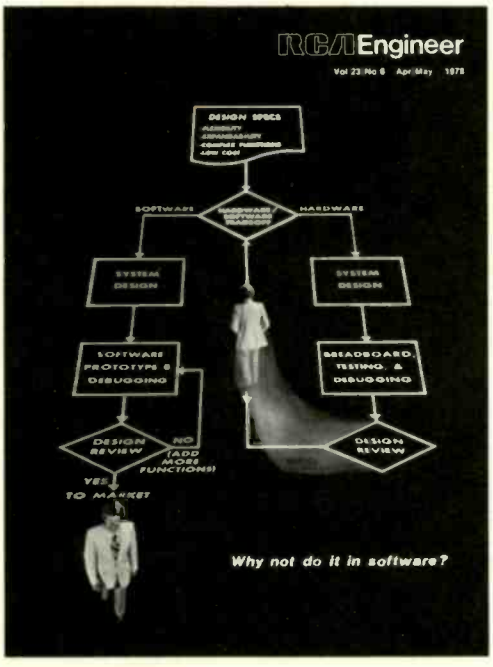
RCA Engineer
Vol 23|No 6 Apr|May 1978



Why not do it in software?

RCA Engineer

A technical journal published by
RCA Research and Engineering
Bldg. 204-2
Cherry Hill, N.J. 08101
Tel. 222-4254 (609-338-4254)
Indexed annually in the Apr/May issue.



Our cover: Why *not* do it in software? We "rigged" the design specs to make software the winner for this system. But, in the real world, more systems are being built to these design specs. As John Tower of the Advanced Technology Laboratories in Camden found out with our "cover" system, the correct hardware/software tradeoff can mean the difference between a competitive product and starting over.

Photography: Bob O'Neill and Karen Hamburg, GCS, Camden, N.J.

- To disseminate to RCA engineers technical information of professional value
- To publish in an appropriate manner important technical developments at RCA, and the role of the engineer
- To serve as a medium of interchange of technical information between various groups at RCA
- To create a community of engineering interest within the company by stressing the interrelated nature of all technical contributions
- To help publicize engineering achievements in a manner that will promote the interests and reputation of RCA in the engineering field
- To provide a convenient means by which the RCA engineer may review his professional work before associates and engineering management
- To announce outstanding and unusual achievements of RCA engineers in a manner most likely to enhance their prestige and professional status.

Software—the future is now

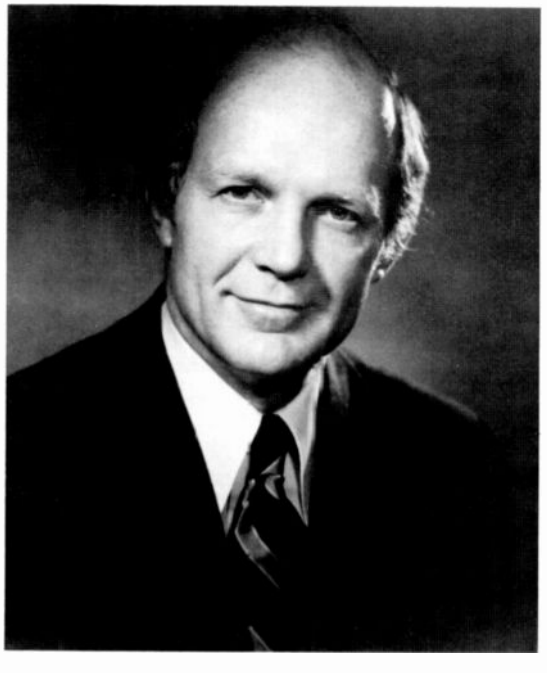
This special issue on computers and software is both timely and important to each individual engaged in engineering of electronic components, products, and systems. Its message is simply stated—the knowledge and application of computer technology is a necessary skill for every engineer, and now is the time for acquiring this skill.

The use of computers for engineering in the past was limited to highly specialized applications, in part because of the high cost of computer time. However, there has been an inexorable trend toward ever lower cost per computer operation, paced by the ubiquitous integrated circuit and more recently by software developments, such that this powerful tool is now a very cost-effective technique.

Today we have computer hardware which ranges from pocket units through micro and mini to mainframe products with an almost unlimited problem-solving capability. Software systems have evolved to the point where the computer user no longer need be a full-time specialist in programming. However, the disturbing gap between hardware and software specialists must be bridged by more than the relatively few individuals who do so today if we are to grow as a cost-effective company in our highly competitive industry.

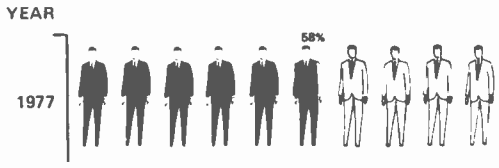
Clearly we face a need for self-renewal somewhat akin to the conversion from vacuum tube to transistor skills of the 1950s. Many of us fear the software area, primarily out of ignorance. It is clear that this need not be so, since the programming art has evolved to an organized discipline that can be learned and used as a skill much as we learned other engineering basics as students.

The message should be clearly understood. Hardware engineers no longer can afford to ignore the software discipline. It is their future as individuals and it is RCA's future as well. This issue should help show you the way to make software part of your future.



William C. Hittinger
Executive Vice President
Research and Engineering
New York, NY

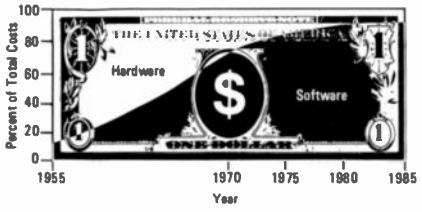
Do it in software



Why?

It's cost-effective, and that's how competitive products must be designed. There are many other technical, professional, and business reasons.

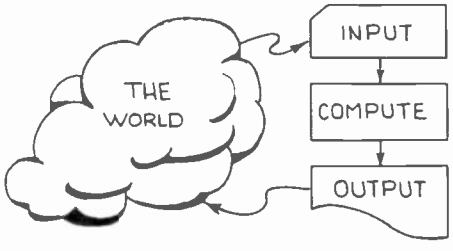
1, 4



When and where?

Right now, and in just about every product imaginable. Today software represents as much as 80% of total system development costs.

48, 57, 62, 68, 74



How?

Learn software soon. Use one of the several formal and informal paths available.

7, 12

What?

Vocabulary may pose an initial problem, but you'll still use your basic engineering problem-solving approach.

20, 28, 34, 39

coming up

Each year, our **anniversary issue** highlights the most important technological events of the year at RCA. The next one (Jun/Jul) will cover **silicon-on-sapphire semiconductors, very-large-scale integrated circuits, optical video disc memories, digital television,** and more.

Future issues will have **space technology, manufacturing,** and **energy** themes.

RCA Engineer

Vol 23|No. 6 Apr|May 1978

why should you get involved with software?

how can you get involved?

what is software?

on the job/off the job

where can you use software?

how do you manage software development?

departments

Why every engineer should be interested in software

J. C. Volpe

An engineer's basic job is to solve technical problems, and software has become a necessary part of the bag of tricks.

Software has become an integral part of practically every major system produced for the government and it is also becoming an increasingly influential element of consumer, commercial, and industrial applications. Just as traditional hardware could not exist without wiring, few major products or businesses can survive without an intimate involvement in software and computers.

The software business

Computers and software are important to our customers and to us.

To begin with, our government customers recognize the great degree of adaptability that comes from computers and software. They are realizing that their systems will not become obsolete as fast as they would have in the past. Update and modification can be done without "cutting metal." However, industry and government customers are concerned that they may be paying an undue price for using this relatively immature field to achieve the rewards of adaptability.

The attitude is somewhat different for commercial products. For years, product planners had looked at all kinds of gadgetry that may have had attractive sales appeal but also were too complex for assembly and maintenance. The difference today and in the near future is that with the microprocessor and its associated software, we have the best of all possible worlds. We can obtain a great variety of sophisticated control with low-cost standard components. Therefore, the product can be sophisticated and yet very practically implemented—with a real payoff to the designer, the manufacturer, and the customer.

From the supplier's viewpoint, the theoretical non-recurring costs in software form an attractive business baseline. Once a program is completed it can be used in any quantity of production versions of the same product without additional cost. Thus, software costs appear theoretically non-recurring compared with the replication costs of hardware—theoretically because most programs are actually updated or improved in some way from time to time.

Reprint RE-23-6-2
Final manuscript received April 4, 1978.

Developing a major software capability can affect a company's entire organizational structure.

Another important aspect of the software business is the capitalization and investment needed to support large software programs. Beyond a minicomputer-sized software job, a substantial program generation center is needed, which includes the equipment—the basic computer with an impressive array of peripherals—and a highly trained staff to operate it.

Also, with software come new forms of data storage and retrieval, libraries, vaults for master tapes, etc. So, as our industry becomes involved in computer programming, buying a computer and developing programs are only the first stages of involvement. Changes have to occur in plant layout, air conditioning, special power requirements, specially furnished computer rooms, skills of personnel; in other words, the whole organizational structure is affected.

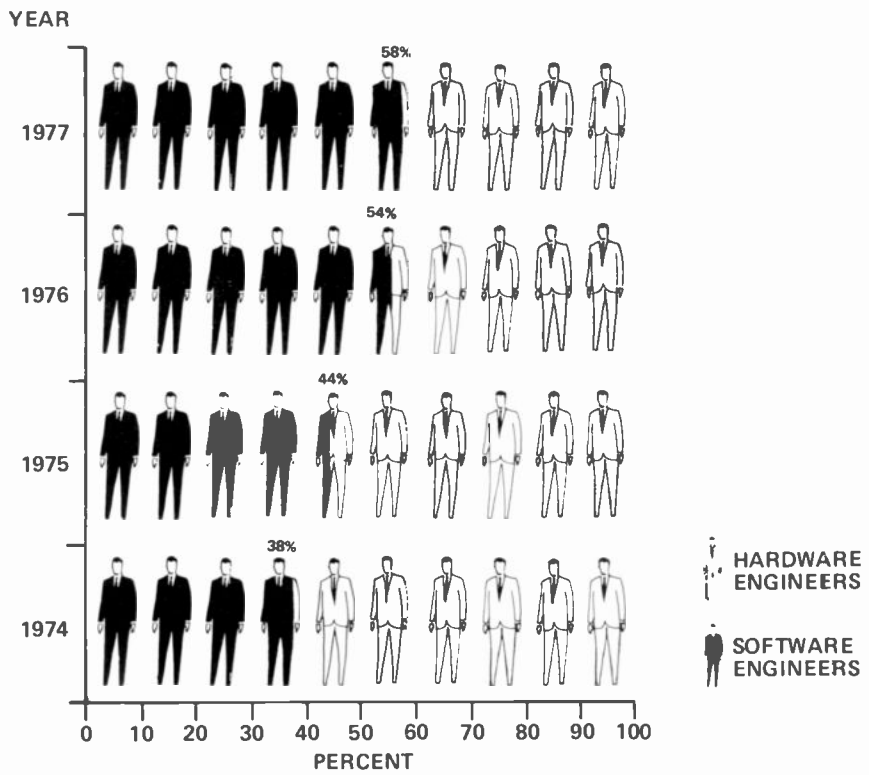
The impact on engineers

One of the most significant of these changes—and the one of most interest in this paper—is the new skills required with the transition to software.

Hardware engineers and programmers alike will have to make the transition to computer science engineers.

Change, transition, and discipline all depend on people—in this case, engineers and programmers. I think we already have several experiences here at Moorestown that illustrate how engineers can make the transition. Some engineers have made a concerted effort through training programs; others took on-the-job training assignments and gradually worked their way up in complexity to accomplishing a journeyman-level effort in the software field; others were more or less forced, by economic pressures, to learn software. Of course, some engineers are exceptionally gifted. They probably could change careers to biology or chemistry if they so choose; such engineers have no problem acquiring software skills.

In general, then, no matter how they are doing it, engineers have been making the move toward software skills. They have the mathematical and logical capabilities, can learn the language, and thus begin to cope with the software



Software engineering ranks are growing dramatically and steadily. RCA's largest engineering center, Missile and Surface Radar, has seen its percentage of software engineers grow from 38% in 1974 to 58% in 1977.

world. After that, they have been applying time and effort to fill in the background on how the computer will work for them and what specific language levels are required.

Of course, a *mass* changeover of engineers from one field to another is neither desirable nor practicable. The switch has to be one of varying pace and degree. Software is no panacea for all system problems. The skills and track record we have developed over the years in hardware designs ought to temper the transition. We have to make clean tradeoffs and determine whether a system or a change in a system can be done more effectively and economically in hardware, software, or a particular combination of the two. But this requires our technical staff to be as knowledgeable in software as they are in hardware, and that's why engineers, at all levels, must become more familiar with software.

Preparation for the transition has to begin at all levels of management. As much as time permits, every engineering manager as well as every design, system, and test engineer should go as far as possible in learning computers and software. Very few engineers or engineering managers are going to be isolated from the interface. And education is probably the best way to cope.

Of course, people will be making adjustments from both sides in this process. The success of software as a technology will be realized when software is no longer generated just by programmers, but rather by computer science engineers. The distinction is that computer science engineers should have a grasp of the total system requirements. This represents yet another type of transition—from the individual programmers of yesterday to the new types of computer science engineers of tomorrow.

In a company like ours, the trend is toward larger numbers of people working together on large projects. In a small company, one man can usually do his programming his way, but when 25 people at RCA are trying to fit all their work together through one central processor, they have to do it by the numbers, so to speak, using some common rules.

Some problems in the past in large software efforts have been caused by the reluctance of individuals to adjust to that interface problem. I don't like to think that the management solution is to break the spirits of some individuals. Getting all types of individual contributors to pull together as a team is a much more worthy objective of management. Software and hardware engineers must learn

to merge their talents, practice some reasonable conformity, and achieve success for themselves and their company.

Making the switch

The transition is certainly not a binary function even though most software is binary. At the conceptual level, software is perhaps not much different than hardware; functional requirements can be defined without knowing details of implementation. The software implementation level, of course, requires in-depth skills in programming, coding, testing, and hardware/software integration.

However, for most engineers the transition can be made rather painlessly in the conceptual part of a software program at one end, or the testing part at the other end. A good circuit designer or system designer can easily start and quickly become effective in either of those parts of software, especially if he adds a limited amount of formal or self education. From the employer's side of it, some small dip in the individual's efficiency curve will take place. But that will be followed by the greater eventual effectiveness of that individual.

Of course, not everyone should attempt the full transition to software, although most engineers should learn more about it. If all barbers decided to become plumbers we would have a problem getting our hair cut to our satisfaction, and the same goes for every other craft. But there is a need for software skills, and the percentage of engineers heading into that field is growing.

Engineers at Moorestown are getting some help in making the switch.

Once an experienced individual at Moorestown decides to become software oriented, he can expect some help. Specific individuals can qualify for special training to help them through the transition. In our limited experience here at Moorestown—three training programs—I have no complaints about the results. I believe our business operations manager has no complaint either when he looks at our dollar return for our investment in these individuals. The turnaround has been fairly rapid (a matter of a few months) for the formal training period and followup.

Those who do not qualify for special training programs still have a wealth of resources (e.g., local schools, in-house courses, books, periodicals) available to help them learn computer technology. [*Ed. note*: Such resources are described by L. Shapiro and P. Anderson in this issue.]

The future of software...and engineers

The opportunities in the software field will continue to grow for the foreseeable future. MSR, for example, has seen a doubling of the number of software engineers over the past two years, with a corresponding limited growth on the hardware design side.

Software as a discipline is already showing signs of maturity. It has moved appreciably from the programming art of the 50s and 60s to an organized discipline containing a discrete set of operations that can be learned and applied effectively. We should see the payoffs of standardization as hardware/software combinations become more miniaturized and lower in cost. We will also see greater discipline through the generation of modular software packages, just like off-the-shelf chips at a sub-component level. Through this maturing process, software will lose some of its mystique and become an even more broadly applied technology. It will not require the degree of specialization to implement that it does today. This will represent the next higher level of software design, and the non-computer engineer will be able to cope with this "off-the-shelf" software very satisfactorily if he obtains some basic education in computer science.

In the past, some hardware engineers have resisted learning the software discipline. Many have stopped with FORTRAN or APL. That situation has to change. Ultimately, mastery of computer science may become so fundamental to engineering that it becomes a question of survival.

Joe Volpe, as Chief Engineer of Missile and Surface Radar, has direct responsibility for the largest concentration of software skills in RCA.

Contact him at:
Engineering
Missile and Surface Radar
Moorestown, NJ
Ext. 3952

Computer science: how you can get involved

P. G. Anderson

Chances are that you're going to want to, or have to, work with, or at least cope with, computers and software.

Why computer science?

First, let's establish why you should get involved with computer science. The importance of this infant discipline is illustrated by pointing out the successful applications of supercomputers. But some more down-to-earth personal experiences may bring it closer to home.

The pocket calculator that made our slide-rules obsolete was first introduced to the public in 1971. The article I read said,"However, it might be quite a while before they replace the pencil and paper system, since present cost is about $395. It is hoped the cost will come down soon." (*Popular Electronics*, May 1971.) Most of us waited until the price had dropped by half, and today we are replacing those big old clunkers with the latest model at $29.95. The original $400, four-function model is now a Christmas stocking-stuffer (soon to be found in cereal boxes and handed out by insurance salesmen?).

Calculators are just one example of the enormous success of electronics science, engineering, and technology at mass production of tiny integrated circuits that take the place of yesterday's monstrous constructions. The inflation-ignoring price drops are accounted for by an economy-of-scale production process that, once the (expensive) kinks are ironed out, it's as easy (almost) to stamp out a million circuits as a dozen. If a need or market can be found for a million circuits, then the price-per-unit drops almost to handling and packaging costs.

The cost of a novel circuit, however, is so high that a special-purpose, limited-edition device is prohibitively expensive. The answer, therefore, is to invent and mass-produce a general-purpose circuit that can be easily modified by the user to fit special needs. These general-purpose circuits are known as microprocessors or microcomputers; the user modification is known as programming. Some computer circuit chips are available for under $10; "evaluation kits" for $100-$200; and full-fledged hobby computers with a video monitor, keyboard, and storage tape for $600.

You don't need to become a computer scientist to apply computer science any more than you needed to be a mathematician to apply calculus.

A retired electrical engineer I met told me that he had personally witnessed the birth and death of both the vacuum tube and the transistor. A computer scientist half his age can (with a bit more exaggeration) claim to have witnessed the birth and death of computers. Computers used to be the beasts in air-conditioned rooms with false floors and were too expensive for all but the most important computations. Their grandchildren, the computers-on-a-chip, not only stand alone but are components within other systems: sewing machines, milking machines, teaching machines, automobiles, smart thermostats, microwave ovens, memory typewriters, tv sets, tv games, pinball machines..... The list is limited only by the engineer's inventiveness and familiarity with these new devices.

Super-computers are not disappearing, or even declining in population, but affordable minicomputers and embedded microcomputers are showing up everywhere. Actually, the larger systems are a principal tool for developing smaller systems. The relationship between the few giants and the ubiquitous micros is yet to be seen. But the all-encompassing program libraries and data bases are likely to reside in the large, central computers, which will deliver the required pieces to the distributed small computers on demand. Similarly, the small computers will reciprocate as data-gatherers and remote editing stations. Hook-up durations will be minimal—infinitesimal compared to today's time-sharing costs.

Processes that traditionally have been controlled by analog mechanisms (pressure-driven valves, centrifugal governors, thermostats, etc.) are now being digitally controlled by these embedded computers. The cost of transforming analog to digital signals is more than justified by the level of sophisticated control that can now be done: combinations of events can be monitored, their histories used to evaluate present events, and their records kept. Most importantly, since computers are general-purpose machines, they can be re-programmed when the rules need updating.

Modern engineers, to maintain their competitive edge, have an entirely new component to master. The technology to implant computers into new products, systems, or processes is probably close enough to existing practices to make the transition possible (inputs go here; outputs go there; power supplies go here; mind the timing pulses;...). But these new devices process *data* or *information;* and these quantities, their appropriate structuring, and methods of their mastery comprise a new discipline, computer science.

Computer science, the discipline which provides system flexibility, deals with an invisible product,—the computer's

Super-computers are not disappearing, or even declining in popularity, but affordable minicomputers and embedded microcomputers are showing up everywhere.

The difference between hardware and software activities is that in the former, design leads to production, and in the latter, design leads to more detailed design.

programs, or "software." The popular image is that the computer builder, the hardware engineer, is responsible for them. In fact, the lion's share—often 75% of the cost—of large hardware/software (i.e., "embedded computer") systems, etc., is attributable to the software. Some figures: as far back as 1972, the Air Force spent between $1 and $1.5 billion on software and one third of that money on hardware;* software costs now amount to 1% of the U.S. gross national product.

To maintain currency, to stave off personal obsolescence, you need to know computer science. It's now as important to your discipline as calculus always has been. No, you don't need to become a "computer scientist" to apply computer science, any more than you needed to be a "mathematician" to apply calculus. But the more you get the better you are.

Some engineers do elect the "retread" career path and become full-time programmers, but that is not necessary; you can stay within your present field but increase your effectiveness by learning how to create and use computer software.

Other important reasons for getting on board are: there are always those (how well intentioned?) experts waiting to snow you, and the fun, fascination, and challenge of the field.

What is computer science?

Now that you're convinced that you should learn about computer science, let's find out exactly what it is. Computer science is the study of *information processing*, the means of representing and structuring data and the methods for processing it. This is: data structures and algorithms.

We need to fit this bare-bones abstraction into a broader context. A systems analyst (high-titled programmer) has the problem of determining the information-flow needs of a system (organization, device, ...) and designing the data structures and processing functions to meet those needs (to construct the "information system" shown in Fig. 1). These detailed requirements are converted into a working, tested, documented computer program by a coder (low-titled programmer). In the cases of large, complex information-processing systems, this development goes through several iterations. An analyst assesses the needs and designs the highest-level functions to meet them; then each of these functions is analyzed into its data-structure requirements and component subfunctions; and so forth, dividing and conquering until the last, lowliest detail is dispensed with. This reflects and formalizes what we have known all along as "the problem-solving process," which is what engineers all along have known as "design." The difference between hardware and software activities is that in the former, design leads to production, and in the latter, design leads to more detailed design.

An *algorithm* is a computational recipe, a set of rules that prescribe what calculations are to be made with what data, in which order to perform them, and what to do with the results. To be useful, an algorithm should specify a computational sequence that stops in a reasonable length of time (otherwise it is only of interest to academic or very rich computer scientists). Computer scientists study various properties of algorithms, such as the number of steps involved in an execution (average, worst case, etc.), the amount of computer memory needed to represent the algorithm and to use for work space, and numerical accuracy, and with the tradeoffs involved among these issues. More recently, as with traditional hardware engineering, software engineers (modern-titled programmers) have become concerned with quality engineering, which, in software, is the study of properties of algorithms conducive to reliability and maintainability.

A study of *data structures* involves, at the simplest extreme, efficient ways of representing and approximating numbers (whole numbers, fractional numbers, tradeoff between range and precision, etc.), characters (English text), and simple relations. The plot thickens when we group these simple objects into: complex numbers, vectors, matrices, data tables, and mixed records (e.g., employee payroll records). The most complex data structures involve aggregates that reference or "point to" one another, like the cross-references in encyclopedias and library card catalogs. These data complexes have a much richer

*Proc. of Symposium on The High Cost of Software, Naval Postgraduate School, Monterey, Cal. (Sep 17-19, 1973).

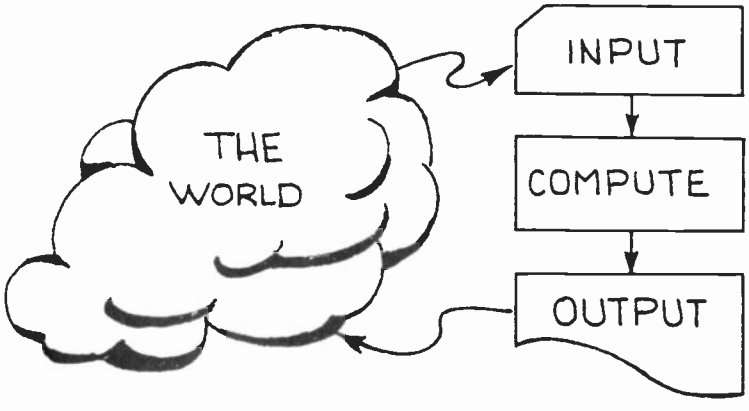


Fig. 1
Information flow. The cloud is a banking system, an airport, a chemical factory,....

potential for structure than conventional paper filing systems, which are generally organized only according to a single scheme. An unlimited number of relationships can be simultaneously used as organizational principles for a data complex. Computer data structures thus range from the simplest (a single numerical variable representing an individual's sex), to simple groupings (a list of employees' names, addresses, and earnings organized by payroll number), to unimaginably convoluted data complexes (a database of information about a community organized by *all* the relationships among the members)!

Saying that computer science is the study of algorithms and data structures is just slightly more revealing than saying mathematics is the study of numbers. To probe deeper, we need to examine computer science's application areas, its tools and techniques, and the machines that it controls.

Computer applications are, briefly, everything. This makes it quite difficult to structure a discussion. (For an interesting attempt at exhaustion, see the list of 2600 applications in *Computers and People,* August 30, 1974.) Any simple classification scheme must be unsatisfactory, although there are a few large, relatively well-defined areas. *Business and commercial data processing* is characterized by the handling of large amounts of data with relatively little arithmetic. This contrasts with *scientific computing,* which performs extensive calculations but handles little data (this is known as "number crunching"). A third major area is *systems programming,* which deals with the computer system itself to make it a more useful tool. The divisions are far from clean—for example, management science optimization problems for the business community require heavy mathematical computing; whereas statistical calculations by scientists and engineers require bulk data processing.

The pressures exerted on computer science from the application areas have resulted in the development of many techniques and theories as extensive as those of the traditional sciences. Numerical analysis has taken a new direction and importance now that its techniques can be automated with such astounding speed. Almost half of the computer programs run in a commercial data-processing operation involve searching and sorting large data files, so this has become an extensively studied and developed area. What is the best way to sort a hundred, a thousand, or a million records? The answers to such questions involve a design of interesting algorithms and interesting data structures. Process simulation is an area almost unheard-of prior to the computer age. Many experiments cannot be performed on actual systems, so the systems are modeled and computer programs written to simulate them. Examples range from experiments at stock market and betting strategies to the structure of communication systems. Simulation is one facet of "computer-aided design," which also includes interactive graphics with its own special techniques and theories.

Saying that computer science is the study of algorithms and data structures is just slightly more revealing than saying mathematics is the study of numbers.

The more language systems you know, the more you will be free of any particular restricted ways of viewing the world.

In addition to the technical techniques of computer science, we now have the product design and development techniques of software engineering that address the problems associated with large software projects, their scheduling, reliability, and maintainability.

All of these application areas and their associated technologies require tools. Since computer programs are more like engineering *designs* or mathematical *theorems* than like any other traditional products, we are not surprised that the main tools are like the main tool of human thought: symbolic languages. As soon as we have a new computer and some rudimentary processes for getting programs and data into the machine and answers out, we demand a symbolic system for designing and coding programs. The simplest of these are *assemblers,* which provide a one-to-one symbolic representation of the machine's operation codes (which are represented by a sequence of ones and zeros within the machine). Assembly languages are, unfortunately, complex and tedious to use, requiring a specialist coder, and are unique to each computer model.

Thus we have *higher-order programming languages,* which are standardized and are closer to the problem area and so worth learning by engineers, accountants, and other users. Even if one is not a proficient programmer, one can still read a program in such a language. The most widely used languages are: COBOL for business data processing, FORTRAN for engineering and science, PL/1, PASCAL, and ALGOL for general application, and SNOBOL for text processing. If one is fortunate enough to have a conversational or interactive computer system, then the introductory language BASIC is usually supplied, and occasionally the powerful but cryptic APL is available for general-purpose application.

These higher-order languages must be translated to machine codes. This is done by a *compiler,* a large computer program built by software specialists known as *systems programmers.* Compiler construction involves some of the deepest results and finest theories in computer science.

Another tool, also the product of systems programmers, is the computer's *operating system.* This was hinted at above with the rudimentary programs to do the basic input and output. A computer operating system has the responsibility for scheduling the computer's resources, bookkeeping,

Programmable calculators for under $30 can give you a realistic feel for algorithms.

servicing the input-output devices and their interface with the user's programs, and keeping the computer operator informed of the system's operation status. An operating system extends and enriches a computer.

Computer science is not a branch of electrical or mechanical engineering, but these disciplines provide the wherewithall for running our programs, so computer scientists are expected to understand the physical components (at least functionally) so they can configure a computer system to meet an organization's needs. This means a knowledge of input-output devices (card equipment, printers, paper-tape devices, magnetic tape and disc drives, drums,...), memory devices (some fast and expensive; others slow and cheap), communications equipment, graphics equipment, and, of course, computers themselves.

Internally, all computers are *not* the same. Even though they are all "general-purpose machines" and can simulate one another's behavior, they have individual strengths and weaknesses that make them more or less suitable for specific applications. For example, some computers support floating-point (i.e., scientific notation) calculation directly in hardware, and others support it in software; this choice can affect the run-time of certain programs by a factor of ten. Data-path width is another issue that affects certain classes of application; a computer that handles large words as a single unit is preferable for communications processing to one which handles smaller units.

What YOU must do

Now that you know what it is you want to learn, how do you go about learning it? As with most other fields, you can only get so far by reading about computer science. You must pay your dues by getting into programming. Fortunately, this is now easily done. Programmable calculators for under $30 can give you a realistic feel for algorithms; home hobby computers for under $1000 are fun toys that can provide you an extensive training ground and be a useful servant and tool as well; minicomputers for under $50,000 are easy to find spare time on (at that price they don't need to be used all three shifts), and those are *real* computers. Employers are more and more easily convinced to allow their employees use of their computer equipment for do-it-yourself education.

When you have located a computer system for your training, you will need some guides. Computer vendors supply user manuals for their systems, but these are more suited for reference use once you know what questions to ask. In the old days (ten years ago), these manuals were all that was available, but now there are hundreds of texts available for beginners. A list of recommended ones to get started using a programming language (your primary tool) is given below. (Use your vendor-supplied manuals with the text; standardization is still one of our fond wishes.)

The SNOBOL 4 Programming Language
R.E. Griswold, J.F. Bage, and I.P. Polonsky
Prentice Hall, 1971

An Introduction to Programming—A Structural Approach Using PL/1 and PL/C
Richard Conway, David Gries
Winthrop, 1973

Structured Programming in APL
Dennis P. Geller and Daniel P. Freedman
Winthrop, 1976

Fortran Programming—A Spiral Approach
Charles B. Kreitzberg and Ben Schneiderman
Harcourt, Brace, Jovanovich, 1975

BASIC
Samuel Marateck
Academic Press, 1974

Structured COBOL
A.S. Phillippakis and L.J. Kazmier
McGraw-Hill, 1977

You will have met the first solid milestone in your computer-science education when you can use one of these languages to solve problems along the lines of numerical analysis (computational calculus), bookkeeping, and sorting. Next—and please don't stop until you have met this milestone—is the ability to specify, construct, and use a software *library;* that is, a collection of reusable programs and subroutines tailored for your applications. This milestone marks the transition from amateur to pro. Your software library converts the computer from a handy gizmo to a business partner. (Libraries available from vendors and user groups can widen the differences among computer systems.)

Once you have this running start, please be modest and don't believe you know it all. The alternative is to become a "computer hack" (for a lucid discussion of them, see *"Computer Power and Human Reason"* by Joseph Weisenbaum, W.H. Freeman and Co.). To solder is not to be an electrical engineer. To get a solid footing in computer science, a good-bet short reading program is two books by Niklaus Wirth, *Systematic Programming—An Introduction* and *Algorithms + Data Structure = Programs* (Prentice-Hall). With these under your belt, you will be able to map out your own additional reading program. Donald E. Knuth has written the first three volumes in his *Art of Computer Programming* (Addison-Wesley); these are heavy on the

One trap to avoid: thinking that your only programming language is the programming language.

mathematics but worth your effort in terms of understanding how to design and analyze programs.

If you succeed this far—master Wirth, get into Knuth—then you need little more overt guidance; you *are* into computer science. Courses and books and journal articles are available, and you are the selector. As your advisor, all I can say is: "these things helped me" or "look at that."

One trap to avoid: thinking that your *only* programming language is *the* programming language. Yours may be the best, but it cannot have an exclusive right to all good ideas, and it will be a force fit to make it apply to many of your applications. On the other hand, other languages may also be inadequate. But the more language systems you know, the more you will be free of any particular restricted ways of viewing the world. Eventually you will be able to program in your problem's ideal language (which exists only in your mind), and then translate the program-design into the program-code for whatever language you may be stuck with—even assembly language.

There are two domestic professional societies directed at computer science. They both support worthy periodicals, regional chapters, and special-interest groups.

The IEEE, known to most engineers, has a Computer Society which publishes magazines: *Computer, Transactions on Computers,* and *Transactions on Software Engineering.* The first is built to keep us educated, with survey articles and invited papers. The second and third are vehicles for research papers with areas of stress: computer-system architectures (consider the roots of IEEE) and software architecture and construction.

The ACM (Association for Computing Machinery) publishes *Communications of the ACM* with articles of general interest (specifically, for us, they have covered recommended CS curricula, GRE exams, and self-assessment procedures); *Journal of the ACM* for the learned theoretical papers in computer science; *Computing Surveys,* filled with gem-quality tutorial articles which keep us all up to date and serve the students and their professors with ready-made lectures; *Computing Reviews,* which publishes critical reviews of the important books and articles; and *Transactions on Mathematical Software* for algorithm dissemination.

ACM has approximately thirty special-interest groups and committees (SIGs and SICs) devoted more narrowly to certain application areas, techniques, problem areas, etc. The SIGs and SICs publish periodicals and sponsor conferences.

There are many other periodicals: *ComputerWorld* is a weekly in newspaper format; *Datamation* and *Computer Decisions* are "free-if-you-qualify" magazines; *Byte, Creative Computing, Dr. Dobbs Journal of Computer Calisthentics and Orthodontia,* and lots of others serve the exploding computer hobby world. This is a tiny sample of what is available. They are all worth looking at and—for your technical library—subscribing to (like most things of this kind, these periodicals are uneven in their quality).

Computer science demands an exactitude that would astound a brain surgeon.

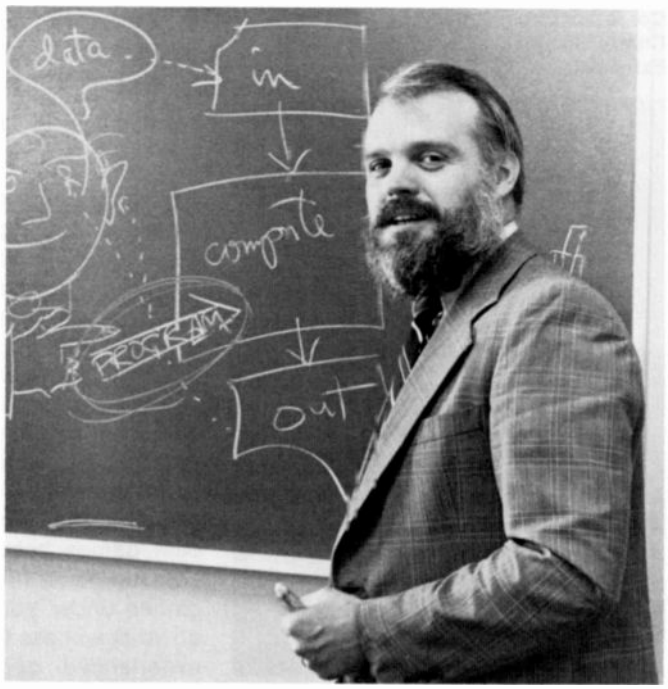
Many periodicals not strictly devoted to computer science are also occasionally good sources for information—especially of the broad survey type. Keep your eyes on *Scientific American, American Scientist, Science,* and *IEEE Spectrum.*

Computer science is growing at a fast rate, both in its intellectual challenge and in its practical application. It demands an exactitude that would astound a brain surgeon. And its rewards are commensurate. Good luck!

Reprint RE-23-6-11
Final manuscript received March 15, 1978.

Pete Anderson, himself a "retread" to computer science, started out in mathematics before learning computer science in RCA's Computer Division from 1965 to 1971. At RCA he participated in the software development efforts for the Spectra/70 family of computers. Since 1971 he has been an Associate Professor of Computer and Information Science at the New Jersey Institute of Technology. He maintains a consulting relationship with RCA and is now, in addition, president of his own software and consulting company, Peter G. Anderson & Associates, Inc.

Contact him at:
Peter G. Anderson & Associates, Inc.
30 Middleton Rd.
Moorestown, N.J.
609-235-5803



Resources available for learning computer science

L. Shapiro

All the other papers in this issue have, in some way, underscored the importance of applying computer science in engineering work. This paper outlines some of the ways to learn how.

The computer is revolutionizing our society. With maturing LSI technology, we can visualize computing capabilities approaching those of the human brain packaged in containers of comparable size. Even at this very early stage, the computing power of such small devices is phenomenal. In essence, just as the automobile and airplane extended man's ability to run and fly, the computer is extending man's ability to calculate, remember, and reason.[1]

[1] This avoids the question "can computers think?" since the term "thinking" can be defined to include the creation of great works of art as well as development of great philosophies and religions.

Louis Shapiro has been associated with the RCA Engineering Education activity both as staff member and consultant since 1966 and has videotaped many of their courses. Currently, he is working on CEE's software engineering curriculum.

Contact him at:
Engineering Education
Corporate Engineering
Cherry Hill, NJ
Ext. 5020

In more practical terms, the infiltration of inexpensive readily available computing capabilities into various products is proceeding with increasing rapidity. We now see computers in satellites, sewing machines, automobiles, and microwave ovens. In effect, every engineer needs an expertise in computers as an essential tool in his work. This article explores the resources available to RCA's technical staff for study in this important field.

For many engineers, hardware and software meet at the microprocessor level.

A very large area of present activity is at the so-called microprocessor level. For many engineers, it is here that hardware and software (programming) meet and join hands. The hardware engineer needs to know programming requirements to design computer architecture; the programmer must know the architecture. The engineer working with a microprocessor must, therefore, be conversant with both hardware and software.

Programming is, in a sense, like playing chess.

Computer expertise necessarily includes a high degree of pure skill and experience in the structuring and writing of programs. Here you face many hours of program writing on many reams of programming forms before a significant degree of maturity can be achieved. A useful analogy would be the development of an expertise in playing chess at approximately the tournament level. Even with the most concentrated study of past games between chess masters, you had better have many hundreds of games under your belt before sitting down at a chess table to face a skillful, experienced opponent—and in the field of computers, the cold stare of computer hardware backed by a poorly written set of specifications can be a formidable opponent indeed. It is with the above thoughts in mind that the following group of resources for the study of computers is presented.[2]

Accredited schools

Most schools have a computer center available for student use. Use such a center as much as possible to obtain the fullest benefit of "learning by doing." In any case, whether study is being pursued outside of RCA or by means of an internal RCA program, bear in mind that the "hands-on" aspect of the learning experience is essential if you want to achieve a practical expertise.

If possible, an accredited school should be your first choice.

For many individuals, study within the structure of an accredited school is highly effective. The regular class meetings, homework assignments, and examinations represent a combination that is hard to beat. In addition, there is the added incentive of possible course-work accreditation toward a degree, with the accompanying increase in professional stature. The appendix[3] lists schools that offer evening or Saturday courses in computer science for the New Jersey-Philadelphia-New York City area where there may be a considerable choice in school selection.

[2] A grateful acknowledgement is made to Mmse. Whitehead and Mattice of the GCS Camden Library for their sustained cooperation and endless patience with the author throughout the preparation of this paper.

[3] The data in the appendix were obtained by means of an independent survey made by the Corporate Engineering Education group, December 1977 - January 1978. Actual course offerings will, of course, depend upon student demand and registration statistics.

However, as will be evident from the appendix, this choice shrinks rapidly as the student progresses toward the more advanced aspects of computers.

Course offerings change very rapidly in the academic world. The information in the appendix should therefore be used as a guide only for contacting convenient schools in your area for coursework of interest. Courses given by county or community colleges tend to be introductory. However, they may be quite valuable as a first introduction into a given area of computer science.

The Association for Computing Machinery (ACM), 1133 Avenue of the Americas, New York, N.Y., 10036, publishes the *ACM Administrative Directory* which lists all colleges and universities in the United States having computer science departments or computer centers, including the names of key individuals to contact. Also included is a complete list of computer-related professional societies and other organizations. As of this writing, the price of this directory is $10.00 if a check is enclosed with the order.

Individual study programs can often be tailored to your background and interests.

In some instances, schools will offer individually designed coursework, either as independent study programs or else in the form of correspondence courses. One school having a well-established correspondence-type curriculum is the Pennsylvania State University, Department of Independent Study by Correspondence, 3 Shields Building, University Park, Pa. 16802. A complete guide to available correspondence courses from some seventy colleges, universities and comparable educational agencies is available through the National University Extension Association, Suite 360, One DuPont Circle, Washington D.C. 20036 for $2.00 per single copy.[4]

Individual study programs, on the other hand, are normally negotiated directly with the school department concerned. Probably an effective way of investigating this avenue of study would be to select a school in your area with a well-developed computer science facility and request an interview with the appropriate department head. You may be able to develop an independent study program which may suit your needs and also carry credit toward a degree.

[4] This publication is actually distributed by Peterson's Guides, 228 Alexander Street, Princeton, N.J. 08540 (609-924-5338).

Table I

Locally developed RCA courses. A number of RCA locations operate on-going in-house training programs. As of this writing, plans for coursework to be presented in 1978 have generally not been finalized. However, the following information has been offered to the writer with the understanding that it is only tentative. In some instances only plans for the spring (of 1978) were available.

Burlington

Microcomputer fundamentals
Microprocessor-based systems design
Software (RCA video tape)
Software (Vendor video tape)

Camden

Software engineering
Microcomputer fundamentals
Minicomputer peripherals
Microprocessors for logic design

Hightstown

Microcomputer fundamentals
Space technology update (emphasizes microcomputers)

Lancaster

APL language
Finite element analysis (computer-aided thermal and structural analysis)

Moorestown

Microcomputing with bit-slicing microprocessors
Advanced software design techniques
The HP 9830 desk-top computer
Microprocessor fundamentals for technicians

New York (Globcom)

Fortran programming
Microcomputer fundamentals
Microprocessor-based systems design I

Princeton

Minicomputer peripherals
Modern logic design I

Somerville

Programming, interactive graphics for design engineers
Modern logic design II
Microprocessors for logic design

Table II

CEE courses in computers. Complete CEE course offerings are listed in the *RCA Engineering Education 1977-1978* catalog which was mailed with the June-July (1977) issue of the *RCA Engineer*. Interested RCA employees should contact their training representative (or equivalent person responsible for the administration of in-house training courses) to make their wishes known. A minimum number of applications is usually required before a course is considered for presentation.

C2—FORTRAN programming. A good introduction to FORTRAN.

C7—Interactive graphics for design engineers. Introduces the use of the Applicon Graphic System/701 as a design automation tool for design engineers.

C11—COBOL programming. Develops basic skills in the use of COBOL.

C30, C31—Modern logic design I and II. Develops the ability to apply modern logic design methods to typical design problems

C35—COS/MOS integrated circuits. Enables the digital designer to incorporate COS/MOS integrated circuits into his designs.

C51—Microcomputer fundamentals. Develops an understanding of the basic concepts of small computer systems. Consideration is presently being given to expand this eight session course to twelve sessions to increase the problem-solving time available to the student. A laboratory course, CL51, designed to accompany C51, is now avaliable.

C52—Minicomputer peripherals. Explores the peripherals commonly used in minicomputer systems.

C55—Microprocessors for logic design. Explores the microprocessor in depth. This course is somewhat dated and a revised version, C56, Microprocessor-Based Systems Design I, is presently under development.

RCA in-house courses

A second major source of convenient classroom-type instruction is the in-house training program offered at many RCA locations.

RCA-sponsored courses generally fall into two categories: those prepared by the location itself, usually to meet specific local needs; and those prepared by the Corporate Engineering Education (CEE) group at Cherry Hill, usually more general in nature. These courses may be given, wholly or in part, outside of working hours. The locally prepared courses (see Table I) are developed and presented by instructors recruited either from within the company or from nearby colleges or universities. CEE courses (Table II) are prepared under the supervision of RCA-oriented engineer-educators.

In either case, these courses have the important advantage of being

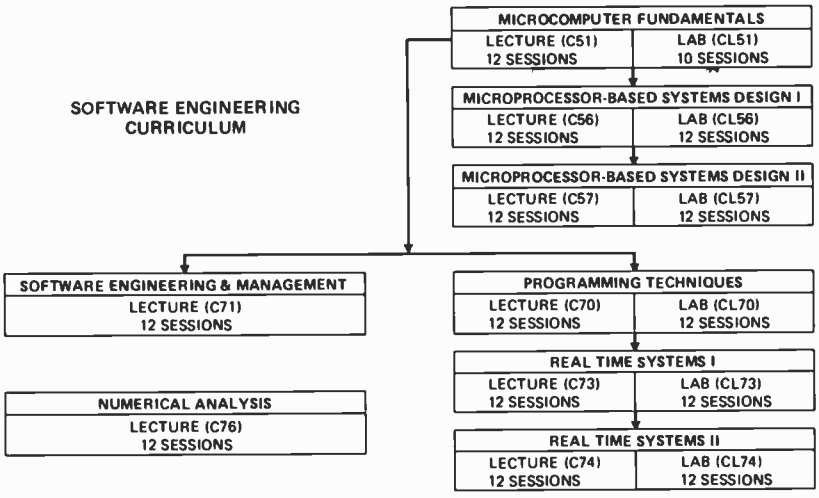


Fig. 1
What's coming in software engineering? This figure shows the structure of the software engineering curriculum presently under development by the Corporate Engineering Education (CEE) activity.

fashioned to meet company needs directly and are free of the usual college or university constraints and traditions. Thus, as locally necessary, the level and demands of the subject material may be altered, the course itself lengthened or shortened, and additional material may be inserted to meet immediate or unique classroom objectives. Normally, no college credit is available for such coursework, although successful negotiations between class members and colleges for subsequent credit may take place.[5]

CEE is now working to develop a complete software curriculum.

The continued rapid entrance of computers into the RCA product and service lines has particularly brought into sharp focus the need for development of sound software skills on the part of the RCA technical staff. In response to this need, the CEE activity devised the curriculum outlined in Fig. 1. Note that most of the key courses have both lecture and laboratory sections so that ample practice will be available.

Individual study

The so-called "short course" has become increasingly popular.

The short course is based on one to five consecutive days of full-time instruction, sometimes with evening workshops. Such courses are given by professional societies, colleges and universities, private educational companies and, in some instances, by producers of computer equipments. They do not carry college credit and normally offer a certificate of completion or the equivalent. The author of this article would place the effectiveness of such short courses below that of conventional coursework where substantial homework assignments between lessons play a major role in the learning experience.

Short courses are expensive, usually from $75.00 to $100.00 or more per day, exclusive of motel and dinner expense where the location is distant from the student's home. A comprehensive list of short courses and associated events[6] is maintained by the Engineer's Joint Council in their publication *Learning Resources, A Directory for Engineers, Scientists and Managers.* This directory is published three times yearly and may be available at the larger public or university libraries.[7]

A fairly extended list of short courses is also available in *Computer* (published by the IEEE Computer Society). Most of the popular journals also contain partial listings of short courses and associated events.

A number of accredited correspondence courses are available.

The private correspondence schools that flourished in the 30s and 40s are now facing stiff competition from the community colleges with their aggressive student recruiting policies, and the short courses with their increased cost effectiveness (for their producers). However, a number of home study courses in computer science have been accredited by the Accrediting Commission of the National Home Study Council[8] and probably provide competent well-designed instruction. Such courses do not, however, usually offer credit toward a college degree. The schools to contact are listed in Table III.

Get some help in evaluating the level of instruction to insure that you will not be swamped with subject matter beyond your comprehension, or find that you have enrolled in a course more suited to a lesser degree of professional involvement. We do not know whether any of these courses include training kits. If they do, however, this would be a positive feature. A certain amount of packaged coursework is available in which the entire course is purchased over the counter, so to speak, and no responsibility is thereafter assumed by the originating agency or company. Such courses sometimes come complete with audio cassettes. Interested parties should thumb through magazines such as *Byte, Kilobaud* or *Popular Electronics* for advertisements. The *CES Newsletter,*

[5] The December 21, 1977 issue of the National Report for Training and Development of the American Society for Training and Development (ASTD) reported that the New York State Education Department found that 68% of 170 responding degree-granting institutions in the state granted college credit for noncollegiate courses upon the recommendations of the Program on Noncollegiate Sponsored Instruction (operated by the New York State Education Department). This amounted to 89% of the nearly 1100 students requesting such credit. More information about the related survey and the Program may be obtained from Dr. John McGarraghy, Office on Noncollegiate Sponsored Instruction, New York State Education Dept., 99 Washington Ave., Rm. 1845, Albany, N.Y., 12230.

[6] In the category of associated events is included seminars, conferences, workshops and similar meetings of professional individuals for the purpose of enhancing their professional abilities.

[7] Available at the RCA Camden Library.

[8] The National Home Study Council, 1601 Eighteenth St., N.W. Washington, DC 20009, was organized in 1926 as an association of schools to establish educational standards and ethical practices. Since 1952, it has been working with the U.S. Office of Education and other accrediting organizations. In 1959, it was listed by the US Office of Education as a "nationally recognized accrediting agency."

published by the American Society for Engineering Education, also carries a column of advertisements for such courses. Contact Assoc. Editor Thomas F. Talbot, Univ. of Alabama, Birmingham, AL. 35294.

Kits offer another alternative.

One way to become initiated into the world of computers would be to purchase a microprocessor-based kit from one of the major manufacturers—wend your way through the instructions to get the equipment in operation—and then try your hand at various programs. RCA, for example, produces a microtutor and a VIP kit, both of which can be used as vehicles for developing a programming expertise. The laboratory, CL51, associated with the CEE course C51, Microcomputer Fundamentals is based on the RCA VIP.[9]

Sybex, 2020 Milvia St., Berkeley, Cal. 94704 (415-848-8233) offers a line of microprocessor self-study courses complete with audio cassettes and text. The Heath Company, Benton Harbor, Michigan 49022 (616-982-3411), offers a microcomputer training course complete with kit trainer based on the Motorola 6800 microprocessor chip.[10]

A number of equipment manufacturers also market kits of various types complete with instruction books. However, many seem to be little more than a means for increasing equipment sales. The investment involved can easily exceed $1000. The beginner would be well advised to avoid such a purchase.

A fair amount of tutorial material has been published.

Tutorial material may roughly be divided into three categories: major publications, popular material, and textbooks. These are listed in Table IV.

A number of periodicals treat almost any area relating to computers.

The periodicals listed in Table V have

Table III
These correspondence schools offer courses in computer science.

Computer programming

Automation & Training Universal, Inc.
425 Lincoln Street
Denver, Colorado 80203
Founded 1966.Courses in computer programming, data processing, and drafting.

Capitol Radio Engineering Institute (CREI)
3939 Wisconsin Avenue
N.W. Washington, D.C. 20018
A division of McGraw-Hill Continuing Education Center. Founded 1927. Courses in computer programming, engineering, and college level electronics subjects.

Herzing Institutes, Inc.
174 West Wisconsin Avenue
Milwaukee, Wisconsin 53203 Founded 1965. Courses in data processing, medical transcription and medical office assisting.

ICS-International Correspondence Schools
Scranton, Pa. 18515 Founded 1890. Courses in high school and college level subjects, technology, engineering, vocational trades, business and industrial subjects.

La Salle Extension University
417 South Dearborn St.
Chicago, Illinois 60605
Founded 1908. Courses in high school subjects, business, vocational, and college level subjects; stenotype and interior decorating.

Lincoln Extension Institute, Inc.
1401 West 75th St.
Cleveland, Ohio 44102
Founded 1922. Courses in industrial and supervisory management development topics.

Computer repair technician

McGraw-Hill Continuing Education Center
3939 Wisconsin Ave.
N.W. Washington, D.C. 20016
Founded 1971. Courses in engineering, electronics, and automative technology; air conditioning and appliance servicing.

National Radio Institute (NRI)
3939 Wisconsin Avenue
N.W. Washington, D.C. 20016
A division of McGraw-Hill Continuing Education Center. Founded 1914. Courses in air conditioning, appliance servicing, computers, radio-TV repair, and automotive topics.

National Technical Schools
4000 South Figueroa St.
Los Angeles, Cal 90037
Founded 1905. Courses in electronics, appliance and radio-TV servicing, air conditioning, and automotive subjects.

Table IV
Published tutorial material available for individual study

Major tutorial publications:

IEEE[11]
345 E. 47th St.
New York, N.Y. 10017

The IEEE offers a number of tutorial-type publications which consist mostly of selected reprints. It would be advisable to review them (in the library) before investing, since a minimum background in computer technology is usually assumed.

IEEE Computer Society
5855 Naples Plaza, Suite 301
Long Beach, Cal. 90803

This society offers twelve tutorials as well as a fairly large number of books on computers, computer applications, and a digest of papers. The same precaution is advised here, e.g., look before you buy.

Association for Computing
Machinery (ACM)
1133 Avenue of the Americas
New York, N.Y. 10036

The ACM publishes much technical and tutorial material including a survey and tutorial quarterly, *Computing Surveys*. The above precautions are advised.

Popular tutorial material:

A good deal of tutorial material has appeared in popular magazines such as *EDN* (Engineering Design News) *Electronic Design*, and *Electronics*. *EDN* has published some of this material separately.[12] Despite attractive format and illustrations, however, the writer has found most of this material lacking in substance for design engineers. Apparently, space considerations limit the range of in-depth treatment needed. After one has already developed a background in computers, however, such material may be interesting to review for sidelights that may apply to problems at hand.

Textbooks:

For serious individual study, especially for the beginner, there is hardly anything currently available that is really more suitable than one of the better texts. A possible combination for the engineer would be one of the first two texts listed below in combination with a kit or microtutor. The first (Peatman) might hold a slight edge. Either, however, would provide a good foundation in the subject of microcomputers and, incidentally, in the field of computers as a whole since microcomputers by their nature represent much of mini- and main-frame computers in microcosm. The recommended text for technicians (Marcus/Lenk) provides a good overview of the field of computers and their peripherals. The writer, in fact, would also recommend it for engineers who are not familiar with computer hardware and who might wish to get some notion as to how things fit together.

At the Engineering Level:

Microcomputer-Based Design; John B. Peatman (McGraw-Hill 1977)

Microprocessor Systems Design; Edwin E. Klingman (Prentice-Hall 1977)

Microcomputers/Microprocessors; John L. Hilburn, Paul N. Julich (Prentice-Hall 1976)

Digital Computer Circuits and Concepts, Bill R. Deem, Kenneth Muchow, and Anthony Zeppa (2nd Edition, Reston 1977)

Techniques of Program Structure and Design; Edward Yourdon (Prentice-Hall 1975)

At the Technician Level:

Computers for Technicians; Abraham Marcus, John D. Lenk (Prentice-Hall 1973)

Undoubtedly, many other fine texts are available in each of the above categories. However, the above have been available to the author for examination and we feel that they can serve a useful purpose in any self-study program.

[9] The possibilities inherent in the RCA COSMAC VIP as a learning device are explored in *COSMAC VIP The RCA Fun Machine*, Joseph A. Weisbecker; first published in BYTE, August 1977, and subsequently reprinted in *RCA Microprocessor Technology*, available in the RCA Libraries. See also Harry Kleinberg's article in this issue.

[10] Further information may be obtained from the Philadelphia office of the Heath Co., 215-CU8-0180.

[11] Orders for particular publications should be sent directly to the IEEE Service Center, Dept. PB, 445 Hoes Lane, Piscataway, N.J. 08854.

[12] See, for example, "Microprocessors, New Directions for Designers," edited by Edward A. Torrero, Hayden Book Company, Inc., Rochelle Park, N.J. 07662.

been separated into broad categories which may of course, overlap.

Indexes and general references may point the way to other literature.

The two indexes shown below list all generally available literature dealing with computers within a few months after it is published. Most people consider the second of these simpler to use.

The Engineering Index
Engineering Index, Inc.
United Engineering Center
345 East 47th St.
New York, N.Y. 10017

Computer and Control Abstracts
INSPEC (Institution of Electrical Engineers [British] in association with the IEEE)
Savoy Place
London WC2R OBL

The following book[13] lists reference sources for scientific and technical fields including computer science. Topics covered include bibliographies, encyclopedias, dictionaries, handbooks, almanacs, yearbooks, directories, etc.

Scientific and Technical Information Sources: Ching-Chih Chen (The MIT Press 1977)

Additional RCA internal services

The RCA Engineer reaches almost every engineer within the company and often includes articles dealing with accomplishments in the computing arena.

Occasional issues of this bimonthly publication are devoted almost entirely to specialized areas such as computers. For example, the February-March issue of 1977 dealt comprehensively with microprocessor system design and applications; this issue treats software. The Technical Communication Programs group, which publishes the *RCA Engineer*, also publishes collections of articles dealing with specific subjects. Thus, a recent (1977) publication included reprints of twenty-four articles reviewing microprocessor technology—hardware, software, and applications.[14] Engineers working in this field may, therefore, find it valuable to maintain a file of these publications as a reference source.

[13] Available in the RCA Camden library.

[14] This publication, *RCA Microprocessor Technology*, is available in the RCA libraries, or you may obtain your own copy by sending a check or money order for $2.00 payable to RCA Technical Communication Programs to this activity at Bldg. 204-2, Cherry Hill, N.J. 08101.

Table V
These periodicals treat computers at various levels of expertise. Most are available through RCA's major libraries.

Electronics generally but tending to emphasize computers and often containing tutorial computer articles:

Electronic Design News (EDN)
270 St. Paul Street
Denver, Col. 80206

Electronic Design
Hayden Publishing Co., Inc.
50 Essex St.
Rochelle Park, N.J. 07662

Electronics
McGraw-Hill, Inc.
1221 Ave. of the Americas
New York, N.Y. 10020

Devoted to computers. Popular style—highly readable.

Computer
IEEE Computer Society
5855 Naples Plaza, Suite 301
Long Beach, Cal. 90803

Microprocessors
IPC Business Press Limited
205 East 42nd St.
New York, N.Y. 10017

Computerworld
797 Washington St.
Newton, Mass. 02160

Computer Home Hobbyist. Two excellent highly readable journals:

Byte
Byte Publications, Inc.
70 Main St.
Peterborough, N.H. 03458

Kilobaud
1001001, Inc.
Peterborough, N.H. 03458

And one journal of **out-of-the-way articles** for people deep into home computers:

Computer Calisthenics and Orthodontia
People's Computer Company
Box E, 1263 El Camino Real
Menlo Park, Cal. 94025

Highly Technical—for the knowledgeable computer engineer:

Journal of the Association for Computing Machinery
Association for Computing Machinery
1133 Ave. of the Americas
New York, N.Y. 10036

Communications of the ACM
Association for Computing Machinery
1133 Ave. of the Americas
New York, N.Y. 10036

Computing Reviews
Association for Computing Machinery
1133 Ave. of the Americas
New York, N.Y. 10036

IEEE Transactions on Software Engineering
and

IEEE Transactions on Computers
IEEE Computer Society
(contact IEEE Service Center)
445 Hoes Lane
Piscataway, N.J. 08854

The Computer Journal
The British Computer Society
29 Portland Place
London W1N 4HU

Software—Practice and Experience
John Wiley & Sons, Ltd.
Baffins Lane, Chichester
Sussex, England

Computing
Springer Verlag
175 Fifth Ave.
New York, N.Y. 10010

Specialized publications:

CAD—Computer-Aided Design
IPC Business Press Limited
205 East 42nd St.
New York, N.Y. 10017

Computer Decisions
Hayden Publishing Co., Inc.
50 Essex St.
Rochelle Park, N.J. 07662

Computer Graphics and Image Processing
Academic Press, Inc.
111 Fifth Ave.
New York, N.Y. 10003

Computer Review
GML Corporation
594 Marrett Rd.
Lexington, Mass. 02173

ACM Transactions on Mathematical Software
Association for Computing Machinery
1133 Ave. of the Americas
New York, N.Y. 10036

Datamation
Technical Publishing Company
1301 South Grove Ave.
Barrington, Ill. 60010

Simulation
The Society for Computer Simulation
1010 Pearl St.
La Jolla, Cal. 92037

Interface Age
McPheters, Wolfe and Jones
13913 Artesia Blvd.
Cerritos, Cal. 90701

Creative Computing
P.O. Box 789-M
Morristown, N.J. 07960

RCA library services are available to most engineers.

Technical libraries are maintained at 18 RCA locations within the continental United States. Nine of these are staffed by at least one librarian and publish bulletins periodically listing new acquisitions such as texts, reports, proceedings, etc. In addition to a limited circulation, these bulletins are available for review at the libraries themselves. Thus each RCA employee has access, within local library constraints, to the 80,000 books and 600 periodicals stocked in these libraries. In addition, RCA librarians have access to catalogs listing the location of desired material in other public and private libraries. In some instances, interlibrary loans are possible, or copies of desired articles may be obtained.

Each of the nine RCA libraries staffed by at least one professional librarian maintains either a microfilm or microfiche facility for review or study of data such as technical papers, military specifications or military or commercial standards.

In addition to maintaining previously-mentioned engineering and computer indexes, the larger RCA libraries have access to an on-line search service through Lockheed's DIALOG and System Development Corporation's ORBIT systems. Upon providing the RCA librarian with either subject or author data, the computer system will within a few minutes provide a list of titles dealing with the desired information; it can also quickly supply selected abstracts.

RCA Technical Abstracts can lead the way to other RCA documents.

RCA Technical Abstracts is a company-private monthly bulletin listing abstracts of papers, reports, books, theses and similar material authored by RCA employees. The bulletin, in addition, includes MIT reports available through RCA's participation in the MIT Industrial Liaison Program.

Copies of *RCA Technical Abstracts* are distributed to all RCA libraries and to RCA management throughout the company. Copies of documents listed in the bulletin may be obtained through your local RCA librarian.

Conclusion

A number of paths are open to help you learn more about computers. None of the paths are easy; all require extra effort, hours, and initiative. But the knowledge you will gain, in many cases, will be as important to you and to RCA as your basic engineering education.

Reprint RE-23-6-10
Final manuscript received January 15, 1978.

Appendix A

SCHOOLS that offer evening or weekend courses in computer science. (All courses are evening courses unless otherwise specified.)

Philadelphia-Camden area: The following ten schools show the indicated offerings in their evening programs. Any one of the ten would represent a means for obtaining a good introduction into the field of computers in accordance with the particular area of interest of the prospective student. However, for individuals seeking to obtain comprehensive, advanced engineering-oriented expertise in computers via evening course work, the only suitable school is the University of Pennsylvania (see comment accompanying this school below).

School	Comments
Burlington County College Pemberton-Browns Mills Rd. Pemberton, N.J. 08068 609-894-9311	Offers data processing, computer concepts, computer programming, FORTRAN, COBOL, RPG2, and business systems analysis and design. Some of these courses are also given at the Willingboro and Cinnaminson Campuses. However, all information may be obtained from the main campus at Pemberton
Camden County College Little Gloucester Road Blackwood, N.J. 08012 609-227-7200	Offers computer science, data processing, computer programming (including advanced techniques), principles of system analysis, and computer mathematics (number systems, Boolean algebra, etc.)
Drexel University 32nd & Chestnut Streets Philadelphia, Pa. 19104 215-895-2400	Computer-oriented courses are offered in three departments as follows: *Business:* management information systems *Electrical Engineering:* computer logic and computer design. *Special Studies:* computing machines, data processing for business applications, advanced computer programming The Evening College offers a six-year program leading to the BS in Computer Science. Computer-oriented courses are offered in two departments as follows.
La Salle College Olney Ave. & 20th St. Philadelphia, Pa. 19141 215-848-8300	*Computer and Information Science:* computing, algorithm and data structures, file and data management systems, information systems design and programming languages. *Electronic Physics.* introduction to microprocessors, pulse, and digital electronics. The course on microprocessors is also given on Saturday mornings.
Rutgers University (Camden) 311 North 5th Street Camden, N.J. 08102 609-757-6057	The following courses are offered at an elementary level: introduction to computing, introduction to programming languages, and commercial data processing. A Saturday course is offered in computer programming for business and social science.
St. Joseph's College City Avenue at 54th Street Philadelphia, Pa. 19131 215-879-7400	The *Business Department* offers a basic course in computer science (emphasizing programming).
Temple University Broad St., Montgomery Ave. Philadelphia, Pa. 19122 215-787-7201	The *Computer Science Department* offers a very substantial evening program leading to the masters degree in either Arts, Science, or Business Administration. The thesis requirement has been replaced by a project which may also be completed by evening work. A PhD with a Computer Science major is available in Business Administration. The Ambler campus in Ambler, Pa., offers a limited group of undergraduate and graduate computer courses.
University of Pennsylvania Philadelphia, Pa. 19104 215-243-7502	The *College of Engineering and Applied Science* offers only graduate courses in its Computer and Information Science Program. These courses, however, begin at the introductory level so that individuals with an engineering background can safely enroll. Available courses

cover a very wide range of computer-oriented subjects and it is possible to complete course requirements for the MSE and PhD degrees via evening work alone. An additional thesis is required for the master's degree and a dissertation for the doctor's degree. A time limit of 7 years is allowed for completion of the MSE requirements and an additional 5 years is then allowed for completion of doctoral requirements. The latter includes a thesis defense. It should be noted that matriculation for either of these degrees requires a bachelor's degree in an appropriate discipline.

The *Wharton* evening school offers courses in introduction to computer programming, and management information systems.

Villanova University
Villanova, Pa. 19085
215-527-2100

The *Computer Science Department* offers courses in algorithms and data structures, computer programming and advanced computer programming. The *Electrical Engineering Department* offers introduction to computers and programming. The *Engineering Department* offers introduction to computers in engineering. The *Business Administration Department* offers introduction to computers.

Widener College
14th and Chestnut Sts.
Chester, Pa. 19013
215-876-5551

The *Engineering Department* offers introduction to computer use in science and engineering. The *Management Department* offers data processing and information systems, and advanced computer science.

Princeton-Hightstown area: As noted by the comments below, advanced work via evening courses is available only at Rutgers University—and then only in the Mathematics Department. Individuals desiring advanced engineering-oriented study should consider travel either to Pennsylvania University (Philadelphia) or to schools such as The N.J. Institute of Technology (Newark), or the Stevens Institute of Technology (Hoboken).

Mercer County College
1200 Old Trenton Road
West Windsor, N.J.
08561
609-586-4800

Mercer offers the following evening courses: basic computer software, computer operations, RPG programming, computer operations management, cooperative management.

Rider College
P.O. Box 6400
Lawrenceville, N.J. 08648
609-896-0800

The *School of Business Administration* offers: introduction to computer programming and information science, computer systems—FORTRAN, computer problems—COBOL.

Rutgers University
New Brunswick, N.J.
08903
201-932-7386

The *Mathematics Department* offers introduction to computers, introduction to computer languages, computer programming, elementary communications data processing, information processing methods, nonnumerical problems and computer programming. The *Electrical Engineering Department* offers computer system design.

The North Jersey area: A number of fine schools are available. Engineering-oriented computer study at the advanced level is available at schools such as The New Jersey Institute of Technology, The Stevens Institute of Technology, and the Teaneck Campus of Fairleigh Dickenson.

Trenton State College
Pennington Road
Trenton, N. J. 08625
609-771-2111

The *Mathematical Sciences Department* offers: *Undergraduate*—elements of computing, introduction to computer science: *Graduate:* advanced computer programming.

Fairleigh Dickenson
Madison Campus
Madison, N.J. 07940
201-377-4700

Undergraduate: The *Mathematics Department* offers three semesters of computer programming. The *Management and Computer Systems Department* offers a course in fundamentals of computer usage.

Graduate: The *Management and Computer Systems Department* offers a course in management information systems.

Fairleigh Dickenson
Rutherford Campus
Rutherford, N.J. 07070
201-933-5000

The *Management and Computer Systems Department* offers an undergraduate course, fundamentals of computer usage; and a graduate course, management of information systems.

Fairleigh Dickenson
Teaneck Campus
Teaneck, N J. 07666
201-836-6300

Offers a substantial evening program as follows: *Undergraduate: Electrical Engineering Department:* digital computer calculations, introduction to minicomputers, computer switching circuits, logic design with microprocessors; *Mathematics and Computer Science Department:* Introduction to computer programming (lect/lab), data structures, theory of computation, non-numeric computation, current topics; *Engineering Technology Department;* computer analysis lab. *Accounting and Quantitative Analysis Department:* fundamentals of computer usage.

Graduate: Electrical Engineering Department: logical design with integrated circuits, microprocessors and microcomputers. *Mathematics and Computer Science Department:* introduction to computer programming, COBOL, computer architecture, assembly language, software design, commercial systems and applications, modelling and simulation of discrete systems, real time computer systems, operating systems.

Kean College of New Jersey
Morris Avenue
Union, N.J. 07083
201-527-2000

The *Computer Science Department* offers computer arithmetic, electronic data processing, business-oriented programming, computer programming, computer operating systems, large scale information programming, and advanced assembly language.

Middlesex County College
Woodbridge Ave.
Edison, N.J. 08817
201-548-6000

The *Computer Science Department*, which is part of *engineering and engineering technologies* offers a very substantial program in computer science and computer programming at the associate degree level. Evening and Saturday courses include: data processing, introduction to computers, introduction to FORTRAN, introduction to computer science, data structures, computer programming for engineers, computers in society, introduction to COBAL, assembly language (2 semesters), operating systems, systems analysis, advanced ANS COBOL, advanced programming techniques, microprocessor applications and programming, advanced microprocessor applications, RPG programming, and business information systems. The *Business Department* offers introduction to data processing both evenings and Saturday.

Monmouth College
Cedar Ave.
West Long Branch, N.J
07764
201-222-6600

Offers a sizable program as follows:
Undergraduate: Computer programming and lab, fundamentals of data processing, ANSI-COBOL and lab, assembly language programming and lab, compiler analysis and lab, database, also, independent study.

Graduate: discrete mathematics, minicomputers.

New Jersey Institute of Technology
323 High Street
Newark, N.J. 07102
201-645-5140

The *Computer and Information Science Department* offers a comprehensive program at both undergraduate and graduate levels as follows: *Undergraduate:* computer programming and problem solving, computer programming and business problems, introduction to computer science, machine and assembly language programming, principles of operating systems, numerical calculus. *Graduate:* computer programming (Saturday morning), data management system design, computer system design, formal languages, microcomputers and applications, model analysis and simulation, design of interactive systems. Course requirements for the MS in Computer Science may be completed by means of evening or Saturdy coursework. Doctor of Engineering Science degrees are available in Electrical and Mechanical Engineering. These degrees, however, require a minimum of one academic year in full-time residence. A proposal is in process for a joint program with Rutgers

University leading to a doctoral degree in computer and information science. The following two courses are given at **Drew University,** Madison, N.J.: data system design, graph theory. The *Graduate School* offers an MS in Computer Science (Thesis required) and a PhD in Electrical Engineering with a major in Computer Science (dissertation required). A year in residence is required for the PhD. Otherwise, all coursework can be completed in the evening.

Rutgers University (Newark)
101 Warren Street
Newark, N.J. 07102

The *Computer Science Department* offers a substantial number of evening undergraduate courses; the *Graduate School* offers design and development of information systems (two semesters), introduction to computer technology, and seminar in information and decision.

Somerset County College
Rte 28 & Lamington Rd
North Branch, N.J. 08876
201-526-1200

The *Data Processing Department* offers a sizable evening and Saturday curriculum as follows: introduction to data processing, computer augmented accounting (2 semesters), business data processing, introduction to data processing systems, computer operations (3 semesters), COBOL (2 semesters) BAL (2 semesters), RPG, BASIC, PL/1, scientific programming.

Seton Hall University
South Orange Ave.
South Orange, N.J. 07079
201-762-9000

The *Center for Computer and Information Sciences* offers introduction to the use of the digital computer, numerical applications, and computer programming.

Union County College
Springfield Ave.
Cranford, N.J. 07016
201-276-2600

The *Business Department* offers a course in computer programming.

Union County Technical Institute
1776 Raritan Road
Scotch Plains, N.J. 07076
201-889-2000

Offers coursework in data processing, programming, assembly language and business applications leading to the associate degree in applied science. This degree can be earned in four years of evening coursework.

Stevens Institute of Technology
Castle Point Station
Hoboken, N.J. 07030

Offers a substantial number of evening courses in computer subjects in the *Electrical Engineering Dept.* and in the *Pure and Applied Mathematics Dept.* The *Management Science Dept.* offers a course in management uses of the computer. A masters degree in either Computer Science or Engineering (Electrical—Computer Science) is possible with evening work. Six years is allowed for completion, thesis is optional. The PhD requires one year in residence.

New York City area: The New York City area contains a number of very fine schools which offer computer science courses at both undergraduate and graduate levels. Among the more prestigious are the Polytechnic Institute of New York (formerly the Polytechnic Institute of Brooklyn) and Columbia University. However, considerations of available course offerings and transportation convenience should also be taken into consideration.

Adelphi University
Garden City, L.I.,
New York 11530
516-294-8700

A business-oriented school. Offers an evening undergraduate course in computer programming.

City University of New York
The City College
Convent Ave. at 138th St.
New York, N.Y 10031

The Computer Science Department in the Engineering School offers a few undergraduate courses and a full curriculum of graduate courses in computer science in the evening. Thesis is optional for the MS in Computer Science; the PhD in Computer Science requires a dissertation and one year in residence.

City University of New York
Queens College
65-30 Kissena Boulevard
Flushing, N.Y. 11367
212-520-7471

A liberal arts four year college offering a BA with major in Computer Science. Present evening offerings are the following: introduction to computers, computers and programming, introduction to discrete structures, numerical calculus, data structures, programming computers, computer organization, and systems programming. Additional independent study is available in computer languages.

Columbia University School of Engineering and Applied Science
116th St. & Broadway
New York, N.Y. 10027
212-280-2931

Offers a substantial evening graduate program in computer science. The MS degree may be earned by evening work alone. Course work required for the doctorate may also be completed by evening work. The doctorate dissertation, however, may require a period of full-time study.

Fordham University
Bronx, N.Y. 10458
212-933-2233

Offers evening courses in computer/information systems, introduction to COBOL programming, and advanced COBOL programming.

Hofstra University
100 Fulton Avenue
Hempstead, N.Y. 11550
516-560-3491

The *Computer Science Department* offers 9 undergraduate and 5 graduate courses in the evening. The undergraduate courses may be credited toward a BA or BS in Computer Science. There is no graduate degree in computer science.

New York Institute of Technology
268 Wheatley Road
Old Westbury, N.Y. 11568
516-686-7520

The combined *Electrical Engineering—Computer Science Department* offers a comprehensive evening program leading to the degrees of BS in Computer Science and MS in Computer Science. All necessary course work can be completed in the evening.

New York University
Washington Square
New York, N.Y. 10003
212-598-3591

The *Data Processing and Systems Analysis Institute* of the *School of Continuing Education* offers evening courses leading variously to certificates or diplomas. The *Computer Science Dept.* offers a few evening undergraduate courses credited toward the BS in computer science and also a very substantial computer Science.

Pace University
Pace Plaza
New York, N.Y. 10038
212-285-3323

A non-engineering school. Offers evening, Saturday and Sunday courses in Computer Science including introduction to computing, COBOL programming, systems programming, information processing, information concepts, information processing—systems designs, and programming languages.

Polytechnic Institute of New York
(Formerly Polytechnic Institute of Brooklyn)
333 Jay Street
Brooklyn, N.Y. 11201
212-643-5000

The *Computer Science Div.* of the *Electrical Engineering Dept.* offers a very substantial group of evening courses which allows all, or almost all, of the course work required for the MS or PhD in Computer Science to be completed in the evening. A smaller but still substantial number of evening courses are offered at the Farmingdale (Long Island) and the Westchester campuses.

Pratt Institute
215 Ryerson St.
Brooklyn, N.Y. 11205
212-636-3600

The *Computer Science Dept.* offers a comprehensive evening program with the following degrees available: BS in Computer Science, BS in Data Systems Management, and MS in Computer Science. Requirements for all programs can be completed in the evening. There is a Saturday course in computer auditing. The *Electrical Engineering* (undergraduate) *Department* offers no evening courses.

St. John's University
Grand Central & Utopia Parkways
Jamaica, N.Y. 11439
212-969-8000

The *Mathematics Department* offers graduate courses in the evening which may be credited toward an MS in Mathematics with a specialization in computer science.

What is software?

S. A. Steele

Changing from hardware engineer to hardware/software engineer? You'll use your basic engineering discipline, but you'll have to learn a new methodology, pick up a new vocabulary, and get used to dealing with an invisible product. Start here.

Software has become big business in a relatively brief span of 25 years. From the straightforward clerical programs driving early computing machines, software has expanded into sophisticated, interconnected program packages and computer subsystems solving complex engineering and scientific problems. Progress in computer hardware—micro-, mini-, and compact large-scale computers (many of them "embedded" in systems or subsystems as part of end-product equipment)—has accelerated the growth of software development, both in scope and complexity. Today software is a major, integral part of such diverse engineering systems as communications, process control, and command and control systems.

With this growth pattern an established fact, software development—both the software technology itself and its relationship to equipment technology—has become an *essential* part of the system development process. As an example, computer control of a modern phased-array radar system requires a software system design an order of magnitude more complex than the ones used in the computer centers found at many companies. Software has become not only the driving cost element in many system developments, but also an extreme risk area. The trend in activity is illustrated in Fig. 1,[1] which shows the projected use of software over the next 10 years as being so extensive that it can *not* be supported adequately by current technology.

"Let's see, software engineers are the people who use computer science as part of their software effort to produce computer programs that are part of the software that is part of the computer system that..."

Reprint RE-23-6-4
Final manuscript received April 28, 1978.

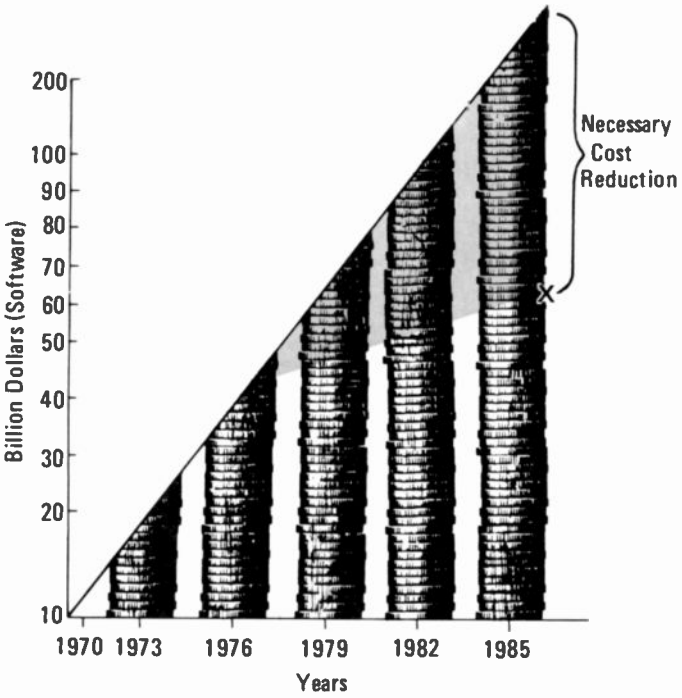


Fig. 1
If software grows as projected, costs will be far too high unless new support technology reduces software development costs to a reasonable level.

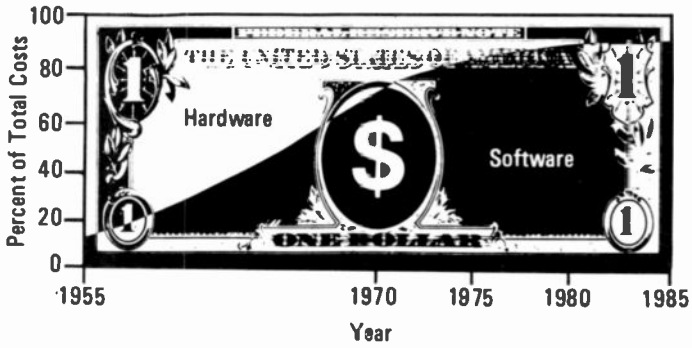


Fig. 2
Hardware/software mix has shifted strongly to software in the past twenty years in the example of command and control systems shown here. Decreasing hardware costs have been more than offset by increasing software costs.

Fig. 2 shows the trend in the hardware-software mix for implementing command and control systems. Further, on a per-unit cost basis, decreasing hardware costs have been more than offset by growing software costs. Barna[2] states, "Every study in existence indicates that at least 90% of the cost of data processing in 1985 will be people costs instead of hardware costs, making data processing the most labor-intensive industry."

Demystifying software

Despite the impressive accomplishment record of the computer sciences, software remains a mystery to many engineers, one complicated by an entirely new vocabulary. Moreover, because one cannot see, touch, or hear software, working with it is especially frustrating to engineers accustomed to working with hardware.

But it needn't be so, and it *must* not be. All engineers and managers need a basic understanding of software; software development is everyone's task. The key to full utilization of computer power lies in developing the technology to minimize today's requirement for large numbers of highly trained personnel to develop and maintain software. For the present, however, software remains the critical path in dealing with data processing in the next decade. The abstract nature of software and its level of quality (number of errors) have created the need for special emphasis on improved software management and development. This article attempts to eliminate some of the mystery that hardware engineers see in software. It should provide that basic understanding, already alluded to, that all engineers and managers will need as software becomes more and more a part of their lives. To do this, the article starts with some basic definitions, explains software architecture and the software development process, and then presents the problems that software engineers face today, along with some potential solutions.

Basic software elements and definitions

Be prepared to learn a whole new vocabulary.

Software or *computer software*, as used here, includes the computer program and computer data that direct the computer hardware in its computational or control functions; also included is the associated documentation related to the product. *Software effort* is the effort required

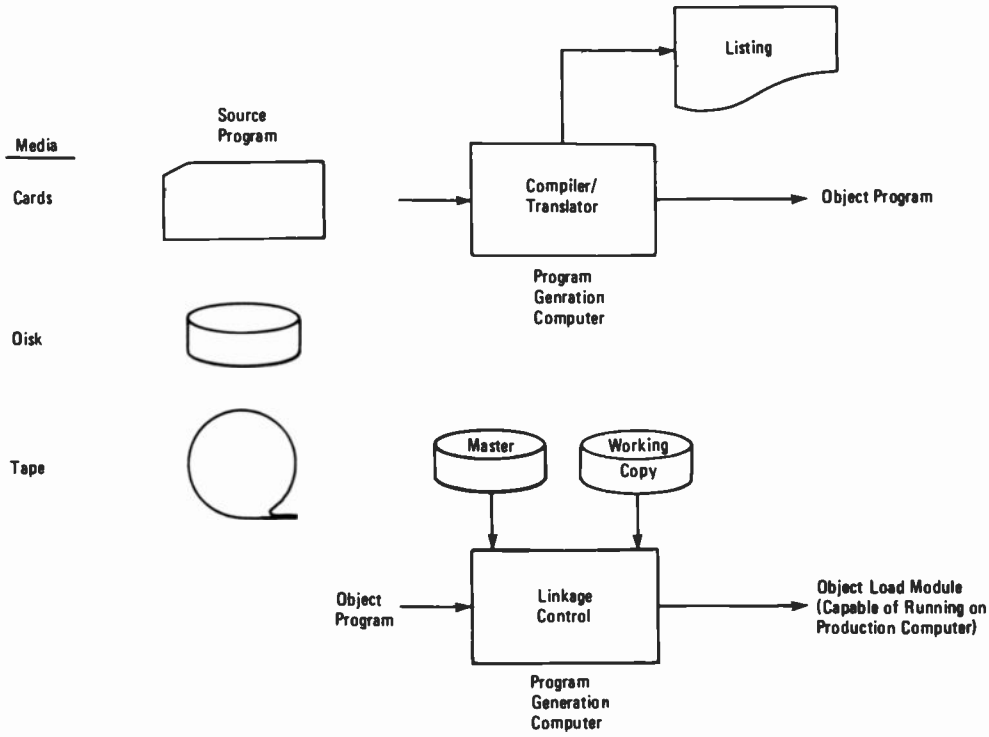


Fig. 3
What happens after the engineer's program is written? The familiar programs that most hardware engineers have written for problem-solving applications are *source* programs, usually written in a high-level language such as FORTRAN. The hardware engineer stops his effort here and waits for the computer to run the program and produce an answer, but the software engineer must look deeper into the process. The *compiler* translates the source program into an *object* program, one which is capable of being executed by the particular computer being used. Next, *linkage control software* makes software changes and picks the proper versions of all the software available, and the *object load module* runs on the computer. Systems may have these "preliminary" actions take place on a separate *program-generation computer,* with the actual program running on the *production computer.*

to generate the computer program, data, and associated documentation. *Computer science* is the theory and development of information processing, and *computer programming* is the conversion of the problem or tasks into the basic steps that the computer can carry out. *Software engineering* (an emerging discipline) is the total process of definition, design, building, testing, and documentation required to develop a computer-program product. The term *software engineer*[3] has come into wide use and indicates one who can specify, design, and test software from a total-system point of view.

The term *computer*, as used here, indicates electronic machinery which, by means of stored instructions and data, performs rapid, often complex calculations, or compiles, correlates, and selects data. Computer systems may require more than one computer—in some cases external computers are needed to prepare programs for use on the *production computer,* which produces the output. A *computer program* is a series of instructions or statements in a form acceptable to the computer equipment that is designed to execute the operations. These programs may be either machine-dependent or machine-independent, and may be general-purpose or specialized in nature. *Computer data* is the representation of facts or instructions in a form suitable for acceptable interpretation and/or processing by the computer. A *computer system* is the aggregate of computer equipment, computer programs, and computer data.

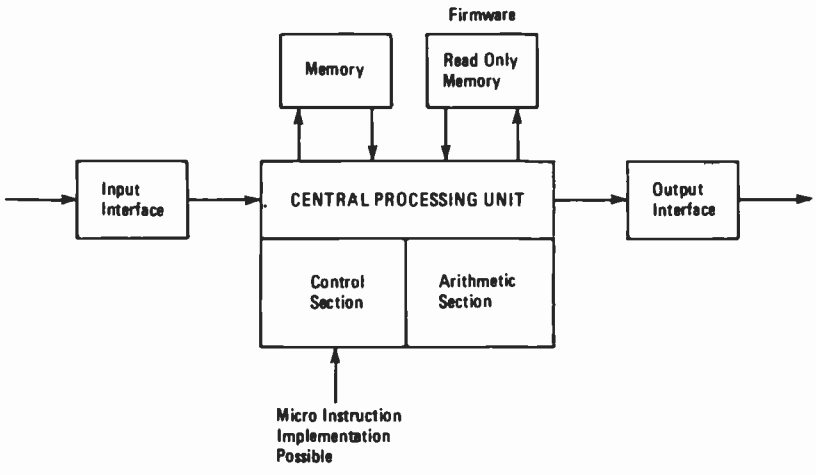
A *real-time* system may be defined as one that controls an environment by receiving data, processing the data, and taking action or returning results quickly enough to affect the functioning of the environment at that time. In a typical real-time system, external devices are connected into the computer so that data can be processed from the devices, computed and analyzed, and then used to generate commands to be sent back to the devices. Response time, or the act of carrying this out in a very fast and responsive way, is a key feature of real-time systems.

Software (or program) architecture is a structural description of a total software system, designed to define the system's programs, their content, and their associated components, both in terms of data flow and program hierarchy. In effect, the software architecture is analogous to the better-known hardware block diagram, providing visibility into both content and function of the system's constituent parts.

Within the computer are three types of software: operating system software, application software, and support software.

The *operating system* (or executive, or monitor) software is responsible for scheduling the resources of the computer complex and controlling such functions as the flow of data between the sensors and the computer complex. *Application software* is responsible for solving unique computational and data-manipulation tasks associated with the specific problem.

Fig. 4
Basic computer architecture consists of four parts—input and output interfaces, memory, and central processing unit. The memory section stores the computer-program statements and data, and the central processing unit (CPU) does the actual computations and manipulations. The data in the memory section changes as it is acted upon by the CPU, but read-only memory is essentially permanent. Because it can be changed by changing chips, it is classified as "firmware"—more difficult to alter than software, but less difficult than altering hardware. The CPU is divided into two sections—control and arithmetic. The control section takes instructions and data from memory and sends them to the arithmetic section, which executes the programs in hardware. The control section was also formerly all hardware, but "micro-programming" now makes it possible to replace this hardware with software.



When a computer complex is used only as a tool to solve a problem in analysis, the application software is of major importance. In modern engineering systems, however, the designer is confronted with an overall system design problem stressing hardware and software issues and in some areas developing or modifying an operating system.

The third category, *support software,* covers items such as assemblers, compilers, debuggers, editors, configuration and control aids, and other software tools used both during program preparation and real-time execution.

It is important to understand the basic mechanics of how a computer program is generated so that the final product can operate.

Fig. 3 represents the program-generation process by which a software engineer may enter a program and have it processed so that it will be loaded and executed in the computer. The *source program* (initial definition or input program), written in assembler or various types of high-level languages (FORTRAN, PL/1, ALGOL, etc.),[4] is placed on tape, disk, or cards. The process that converts this input program to an *object program* (one capable of being executed) is called *compiling,** or sometimes *translating.* This compilation step may be performed either within the production (system) computer or in an external computer for execution later on the production computer. Support software is also required for linking and controlling various changes and versions of the software. This total process of generating software for execution on a production computer can be called *process construction* and is often complex enough that it can *not* be carried out on the production system. The planning and acquiring of modern program-generation facilities to support software engineering development is a major task with today's systems.

The fundamental ingredients of a minimal computer-program package are the source program, the object program, and the associated source program listing needed to read and understand the program. Since customer maintenance of the product will be at the source level, complete correlation is required between source and object, and the program-generation process must support this correlation. It should be noted that this minimal computer-program package is inadequate documentation for changes; design and as-built documentation is also required, including flowcharts.

Computer architecture can be divided into input/output, processing, and memory.

Fig. 4 is a basic computer-architecture block diagram showing the central processing unit (CPU), the input and output interface units, and the memory associated with the central processing unit. Fundamentally, the CPU is made up of a control section and an arithmetic section capable of the basic mathematical manipulations. Generally, the memory contains the computer-program statements and associated data. Under CPU control, each instruction of the computer program is taken from its storage in memory, decoded in the control unit, and then executed by the hardware.

Writing programs making use of the computer hardware instruction set is a normal programming activity. Modern computers, however, often have a *micro-programming* feature.[5] Micro-programming is a technique that substitutes micro-instructions (software) for the logic gates (hardware) that are normally used as the decoding mechanism in the control unit of the CPU. If this level exists on a particular

*If written in assembly language, this is called *assembling*, producing an object program where the ratio of source to object is frequently one to one. This allows programming many of the machine's basic activities.

computer, then the conventional programming level for the computing complex can be called *macro-programming*. This issue is important to an understanding of current trends. Essentially, by means of micro-programming, it is possible to make a particular computer with its hardware instruction set look like another computer. This is accomplished not by rewiring the computer's control mechanism, but by changing its control mechanism's micro-instructions via micro-programming. This process, which allows software that will run on one machine to run on another, is called *emulation*.

Another term creeping into the vocabulary these days is *firmware*. Firmware is essentially a program resident in a computer (often read-only memory) that is effectively hardwired. This program can be fetched from memory and executed as any stored program, but it cannot be changed easily and for many practical purposes is essentially a hardware box. The method of developing this computer program is identical to that for any computer program. However, since it cannot be changed easily after it has been "burned" into memory, its use and associated configuration control are different. The use of microprocessors (processing on a chip) and advanced hardware architectures forces the software engineer to fully understand hardware constraints.

This summary list is by no means complete. Nonetheless, the terms included here cover most of the basic software items and provide a means of understanding the development process and some of its inherent problems and peculiarities.

The software development process

In one respect, at least, software is like hardware—successful design and development require an orderly, logical approach.

A general software development approach, applicable to both large and small efforts,[6] includes the following:

A top-down approach to problem definition and resolution.

Organization of the system into logical development phases.

Definition of required inputs and outputs from each phase.

Established organizational responsibilities and interfaces.

Incorporation of scheduled reviews and approvals.

Implementation of management controls.

Development of documentation as an integral part of the system.

The detailed phasing of the software-development process used to address these broad requirements is discussed in a future *RCA Engineer* by Howery and in an earlier *RCA Engineer* article on the subject.[7] The major phases are system requirements definition, computer-program system performance requirements, computer-program system design, computer-program implementation, computer-program integration and validation, equipment and computer program integration, system testing and acceptance, and system operation and maintenance.

While analogies exist between hardware and software development, the software-development process must be understood on its *own* merits to insure successful development with its own set of standards.[8] Some of the similarities and differences in hardware/software development are discussed below.

System definition—*What takes the place of the hardware block diagram?*

In hardware design, the block diagram is a standard tool for giving definition, design layout, and *visibility* of the equipment content and development process. The software analog to the block diagram must provide similar definition, description, and visibility, but in terms of control between program packages or modules and the data to be processed, rather than hardware subassemblies. The term "software architecture" then becomes roughly equivalent to the hardware block diagram in concept, but takes on different forms to show the complete control and data issues involved. Two examples of architecture information are:

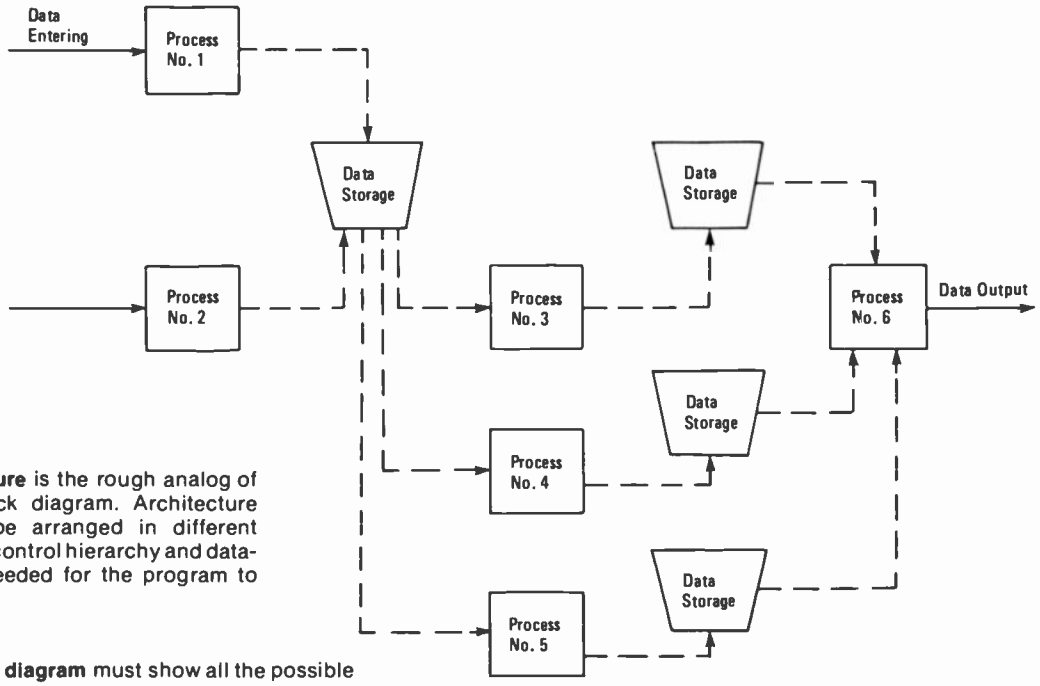
data flow, which indicates the path of data throughout the various program packages and storage areas; and the *program hierarchy*, which structures the programs, subprograms, modules, and routines in a hierarchical tree showing their capabilities for interaction. Examples of these are shown in Fig. 5.

Configuration management and control—*Hardware units are generally built and delivered to the customers individually, but software packages, on the other hand, are copies of one master program.*

With many copies in existence, all subject to user alteration and modification to meet the peculiar requirements of individual systems, the number of configurations of the various "masters" can easily become confusing. Moreover, software packages are often developed as versions indicating different levels of capability. This is another dimension in the control issue and proliferates more "masters." Both source and object programs must be controlled along with the specific computer loads, including data unique to any one system. Coupling this problem with the "invisible" nature of these programs makes the configuration management and control process particularly complex.

Production—*When does the software go into production, if ever?*

Or, in a slightly different analogy, when is the software turned over to the "factory" for building? Today, the concept of building software using a factory philosophy is in its infancy, and qualified software engineers are involved throughout the actual building process. Moreover, in a literal sense, software never goes into production, because copies are always available. However, the word "production" is often used in software after a final product has been generated and standard and preselected modifications can



Software architecture is the rough analog of the hardware block diagram. Architecture information can be arranged in different forms to show the control hierarchy and data-flow operations needed for the program to work.

Fig. 5a
Software data-flow diagram must show all the possible paths that data can take through the program.

be put in for different customers. The term is also used at times for the process of building the coding.

Development tools—*In supporting the development process, software tools are perhaps even more important than the basic test tools used by the equipment engineer.*

Software tools range from the ones used in program debugging during the process of code writing to those needed during test and integration. Developing tools for real-time debugging is especially complex, since software "test probes" will simply corrupt software operation and distort results. Effective monitoring of software requires a combination of hardware monitoring (e.g., on the backplane of the system) and software data-collection "hooks" placed in the system.

The time-sequential nature of software adds a major complexity to the debugging process since it is impossible to "see" all of the software at one time. Without extensive tools, software debugging is more difficult than equipment debugging. Because software tools can easily take up 10-20% of the software development budget, this area demands early and continuous evaluation.

Maintenance and reliability—*How can you apply these terms to "invisible" products?*

In software terms, maintenance includes detecting latent defects after the software has been delivered, and adding enhancements to the basic program. This definition is not consistent with the normal concept of equipment maintenance, but is used in a generic sense as maintenance of the software product after it has been delivered. The level of maintenance and how the customer operates the system must be defined early in the development.

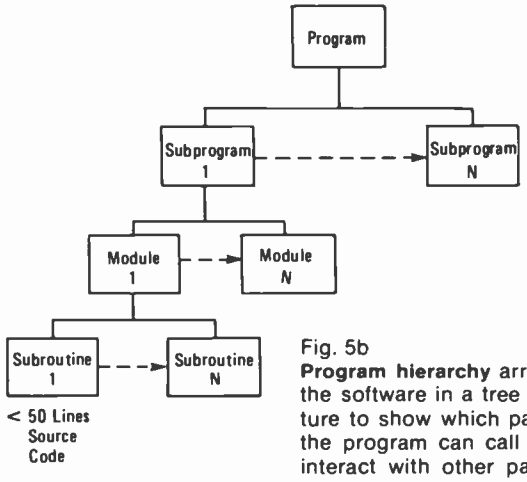


Fig. 5b
Program hierarchy arranges the software in a tree structure to show which parts of the program can call on or interact with other parts of the program.

The term software reliability is normally used in a generic sense to describe the process of assessing operational software performance. Software is considered reliable if its use does not produce more than a tolerable frequency of system failures or unexpected performance. The software reliability process addresses such areas as data collection and analysis on software errors, mathematical approaches to software reliability, software management practices, definition and enforcement of good programming practice, and techniques to increase the thoroughness and cost-effectiveness of testing.

Product patentability—*How do you protect your software developments?*

Although exceptions exist, at the present time it is not possible to patent software, even if it replaces an equivalent (patentable) hardware technique. Copyright mechanics, however, do exist. Also, software products are usually

delivered in object-program form so that users cannot make modifications easily. Licensing of software products to limit their general use is also common.

To be successful in software development, three major skills must be identified. First, the application of the proposed software must be fully understood. Second, the computer hardware characteristics and the overall software architecture must be well understood and laid out. Third, the building process of the software code and associated testing must be clearly defined and made visible to management.

Problems and approaches

The problems. . .

Since the early 1970s, the complexity and cost of software development have attracted increasing attention, and both software developers and users are attacking the associated problems from diverse directions and with a wide variety of techniques. Developers—particularly those involved in major real-time sensor control systems—have been working closely with Department of Defense user agencies to achieve a broad-based mutual understanding of the principal problems and formulate both general and specific approaches to their resolution. These cooperative efforts have isolated seven major areas of concern to both developer and user:[9]

1) Basic misunderstanding of the software system engineering function. The roles of system engineering and software engineering must be defined and recognized as principal elements in a highly complex development discipline.

2) Incomplete and/or inadequate specifications. The volume and scope of software specifications tend to lead to omissions, distortions, and misunderstandings.

3) Incorrect partitioning of hardware and software functions, occuring when an interface is designed either first from the hardware side or the software side and not as an integrated interface.

4) Ineffective software test planning, leading to excessive testing in some areas and inadequate checks in others.

5) Lack of visibility and understanding of the software development discipline, resulting in inadequate control of creative software development.

6) Inherent growth in system core and time requirements—a tendency that occurs even in highly successful projects.

7) Inadequate involvement of the system user during development, frequently resulting in an end product that does not satisfy his needs.

These are by no means the only problems in software development, but they are representative of the most pressing immediate and long-term needs.

. . . attacking the problems

To assist industry and government in resolving these broad problems, the Director of Defense Research and Engineering has issued a generalized software-technology objective list[10] covering four broad areas and setting specific goals within them:

1) *Project management*—planning, estimating, and control of development; configuration control; requirements validation; risk analysis; software quality; technology transfer.

2) *System architecture*—new computer system architectures; impact on software cost, timeliness, and quality; hardware-software-firmware tradeoffs; information-system security; flexibility in data-retrieval systems.

3) *Programming environment*—languages to provide for effective control of software development; automation of clerical aspects; tools for test and validation; maintainability-enhancement techniques.

4) *Reusable software and tool availability*—specification of standard software; adaptable standard software for

Stu Steele is Manager, Software Design and Development—MSR, responsible for the largest software skill center in RCA. He has been involved for 20 years in real-time control and applications of engineering systems, including metrological data processing, spacecraft control and radar control. He has taught extensively in the computer control, software engineering and digital systems area.

Contact him at:
Software Design and Development
Missile and Surface Radar
Moorestown, N.J.
Ext. 2021

other applications and hardware; nationwide language-control facilities; repositories and distribution systems for reusable software; high-order language consolidation.

These and other objectives are compatible with general industry goals that recognize the inadequacy of present state-of-the-art software development techniques to support future requirements. As an example, if the cost of software development using present technology is projected into the 1980s, it will be far too high (Fig. 1)—it must be reduced by *80%* if we are to effectively use the projected hardware growth and professional personnel availability.

Structured programming is a necessity, and some automation seems possible.

Presently, the industry is using formalized techniques in development, such as structured design and structured programming. *Structured design* is associated with the allocation of requirements to partitioned and modular program packages or modules with the interfaces clearly defined. *Structured programming* is associated with using basic and limited language constructs that result in very readable computer-program listings. With special tools to aid the software designer in their use, these techniques can be very effective in attacking the overall problem. More complete definitions of modern programming practices can be found in Myers.[11]

Approaches to these problems are myriad, but a few areas of advance hold special promise. One is the automation of system- and module-specification languages. This is accomplished by means of a general-purpose module generator to accept functional specifications as input for the production of a high-level-language program. These systems can be problem-oriented and would require a variety of generators. The concept is for a specification to be fed directly to the automatic system to produce the programs, thereby eliminating the need for large program-development teams.

Implementation languages will also be getting better, with improved control structures, language facilities to help validate program correctness, and extensions to existing compilers (via pre-processors) for standardizing new features. Design languages will emerge to aid in structured design, and computer-assisted tools will be available for data-base design. Design and implementation languages will be integrated.

Another major area being addressed is requirements definition and validation. This is the key technology area in large-scale systems, where successful software system development depends so completely on firm, accurate requirements. Efforts are currently underway[3] to develop requirements languages and techniques for comparing requirements with specifications.

Software development often "reinvents the wheel."

A perennially expensive area where standardization is desirable is utility/support software—editors, debuggers, input/output handles, standard subroutines (e.g., sine and cosine subroutines), etc.—that it is clearly wasteful to redesign on every new job. Some of these packages are now being considered for implementation in chip form as firmware. From an engineering-design standpoint, the software for these chips must be correct, since it is not economically feasible to recall thousands of chips in order to repair the program bug. This is an example of the type of tradeoff involved for a firmware or straight software approach.

In new engineering systems, distributed computing is the major trend. This normally occurs using micro- and mini-computers, distributed and partitioned to solve specific functions. From a software point of view, this aids in dividing and conquering the "large-scale" software system problem. However, more emphasis must be placed on the control software among machines.[12]

Conclusion

Software is an integral part of modern engineering systems, involving the total spectrum of development, from definition through test. Development of a system employing software requires a *total* system discipline that uses existing experience in both equipment design and computer science/software. At the conceptual level, product development is similar in hardware and software, but there are many differences at the detailed level. For successful system development, the design engineer *must* become familiar with these differences.

One key feature in modern system development is the hardware/software tradeoffs that will guarantee a cost-effective system design. And within the software design itself, extensive tradeoffs are required in areas of software architecture and implementation. Fortunately, texts on the techniques of software design are now becoming more common, so that today it is practical for a designer to function more comfortably and effectively in the software world.

As an industry, we are wrestling with an environment that has requirements outpacing technology; a design engineer contemplating a way to use the technology must recognize this. The challenge is there, the future is exciting, and with added complexity comes an opportunity for successful system development using all the technology available.

References

1. Prywes, N.; "Preparing for future needs," *Computer* (Jun 1975).
2. Barna, B.; "The DP industry: the next five years," *Computer Decisions*. (Jan 1978).
3. Christiansen, D.; "Computers; consolidating gains," *IEEE Spectrum*, (Jan 1978).
4. Schein, F.; *Introduction to Computer Science*, Schaum's outline series, McGraw Hill.
5. Torrero, E.A.; *Microprocessors: New Directions for Designers*, Hayden Book Company, (1977).
6. Transworthe, R.C.; *Standard Development of Computer Software*, Prentice Hall, Inc., Englewood Cliffs, N.J. (1977).
7. Steele, S.A.; Dupell, R.A.; Fleishman, A.; and Hatcher, E. T.; "Managing computer program development," *RCA Engineer*, Vol. 19, No. 5 (Feb/Mar 1974).
8. Woolbridge S ; *Systems and Programming Standards*, Petrocelli/Charter, New York, (1977).
9. Steele, S.A.; 'Characteristics of Managing Real-Time Software Development for Military System," AIAA Computers in Aerospace Conference, Los Angeles (Nov 1977).
10. "FY 79-83 Research and Development Technology Plan," Office of the Director of Defense Research and Engineering (Sep 1977).
11. Myers, W.; "The need for software engineering," *IEEE Computer* (Feb 1978).
12. Stone, H.A.; "Computer systems: what the future holds," *Computer Science and Scientific Computing*, Academic Press, New York (1976) Edited by J.M. Ortega.

Programming in CHIP-8

H. Kleinberg

Learning programming on RCA's VIP is relatively easy and painless, and CHIP-8 is one of the major reasons.

This paper introduces one of the most powerful parts of the COSMAC VIP—the CHIP-8 interpreter. The paper is written primarily for those who have not been exposed to a programming language in any depth, but who are familiar with numbering systems and with such basic computer concepts as memory and instructions.

If you don't have that background, you should still be able to get a good idea of what kind of functions are available to the CHIP-8 user. This paper by itself, however, will probably not prepare you to write a program without a bit more coaching. [Try the "Good Guide Special" described in this paper's final section, "Taking the next step."—Eds.]

Descriptive information about the VIP and its accessories is readily available and most of the information appearing in the instruction manual will not be repeated here. But it is worthwhile to review briefly how the system's data structure is organized. The COSMAC VIP handles data in "bytes," each consisting of 8 bits. A byte can, therefore, take on any one of $2^8 = 256$ configurations. For example, 01101101 is equivalent to decimal 109. For ease of handling, this awkward (to people) entity is treated as though it were broken up into 2 subgroups of 4 bits each, although no such fracture occurs in reality. Each subgroup can now assume $2^4 = 16$ configurations, hence the name "hexadecimal," shortened to "hex." These 16 binary configurations, when expressed in hex, are labeled from 0 through 9 (borrowed from our decimal system) and A through F (equivalent to decimal 10 through 15). Our earlier example of decimal 109 now becomes 0110 1101, or 6D in hex, and the byte has now been expressed as two hex digits.

Using the hex system is an acquired skill, and cumbersome in the early learning stages, but it is far easier to cope with than the pure binary which it masks. In all that follows, hex digits are used, so your counting and arithmetic must be based on that system, a fact of which you will be reminded periodically.

You have now been relieved of the burden of coping with long binary strings, but you have not been taken out of the level of machine instructions, where you must concern yourself with the many specific details of the computer's structure. To alleviate that chore, a program called CHIP-8 has been written as part of the VIP system package. CHIP-8 is a special type of program called an interpreter.

Why use an interpreter?

An interpreter provides you with a simplified language that is tailored to a specific application.

In the spectrum of software, interpreters lie between assembly-level programs on the one hand, and the more elaborate and generalized compilers such as Fortran on the other.

The designer of an interpreter starts by picking a set of functions that would be useful in his specific application. For each function he writes a machine-language program, which is later activated when the user calls out the appropriate code in his "higher-level" program. Some of the functions can be quite complex when worked out in machine code.

As with any other product design, an interpreter is a result of certain compromises. There are conflicts between the range of application (flexibility) of the language, the speed of execution, the amount of memory required to store it, the complexity as seen by the user, and so on. In the case of CHIP-8, all these questions were settled in favor of simplicity and economy.

Harry Kleinberg, formerly with RCA's Computer Division, is now Manager of Corporate Standards Engineering. He can't seem to get the 1s and 0s out of his blood, though; besides this article, he recently wrote a book titled *How You Can Learn to Live with Computers.*

Contact him at:
Corporate Standards Engineering
Cherry Hill, N.J.
Ext. 6616

Author Kleinberg with his computer system: the VIP (right), tv display, and cassette recorder (left).

It is important to keep in mind the distinction between the interpreter and the computer on which it is run. By its nature, an interpreter language is designed for a specific application, sacrificing features that would be useful in other, different applications. For example, CHIP-8 is specifically designed for controlling the video display and hex keyboard of the VIP, but is a very poor language for numerical computation. The VIP's 1802 microprocessor, however, does not have that restriction. Using the same 1802, we could write another interpreter that would be excellent for vector calculations and very poor for video games. The interpreter reflects conscious restrictive choices that in no way alter the generality of the computer on which the interpreter is run.

The basics of CHIP-8

CHIP-8 instructions have a simple, consistent format.

Each instruction is 2 bytes (4 hex digits) long, and each performs a distinct, well defined function. The general format is ABBB, where A is always part of the instruction code, and B is either part of the code or data that you must supply. Fig. 1 lists the instructions in numerical order. (Remember that A,B....F are numbers, not letters.)

In order to understand the instructions, you must first become familiar with a vocabulary of 1 word and 7 symbols.

VX and **VY**, or **X** and **Y**, stand for the program *variables*. A key feature of CHIP-8 is the use of 16 variables, which are actually memory locations containing numbers over which you have complete control. They can signify whatever you want them to signify. You can set them to any value you choose, you can compare them, you can increment them. Each variable is identified by one of the hex digits, 0 through F, but in Fig. 1 and in the descriptive material where the general form of the instructions is used, they are referred to as VX, VY, or simply X or Y, where X and Y, of course, range from 0 to F. Thus, you will be performing such operations as "check if variable 7 (V7) is equal to zero" or "add 1C to the present value of variable B (VB)." Each variable represents one byte of data and its value is, like all other data, expressed in a program as a pair of hex digits.

Only a few other symbols are used in describing the CHIP-8 instructions:

I, called a pointer in the manual, is a memory address. In general, it identifies the beginning of a string of data to be read from or written into the memory. The pointer is your major tool for storing and retrieving data, and it is important to note which instructions use it or modify it.

MI refers to the data stored at the location(s) addressed by I. In a sense, I is the post-office box, MI is the letter stuffed in that box.

MMM indicates memory addresses, other than I, that are to be supplied by the programmer. Since CHIP-8 was designed to run with a small memory, the most significant hex digit in the address will always be 0, i.e. 0MMM.

KK represents a 1-byte value or number that you will supply in your program.

Instruction	*Operation*
***00E0**	Erase display (all 0's)
***00EE**	Return from subroutine
0MMM	Do machine-language subroutine at 0MMM (subroutine must end with D4 byte)
***1MMM**	Go to 0MMM
***2MMM**	Do subroutine at 0MMM (must end with 00EE)
***3XKK**	Skip next instruction if VX = KK
***4XKK**	Skip next instruction if VX ≠ KK
***5XY0**	Skip next instruction if VX = VY
***6XKK**	Let VX = KK
***7XKK**	Let VX = VX + KK
***8XY0**	Let VX = VY
8XY1	Let VX = VX OR VY (VF changed)
8XY2	Let VX = VX AND VY (VF changed)
8XY4	Let VX = VX + VY (VF = 00 if VX + VY ≤ FF, VF = 01 if VX + VY > FF)
8XY5	Let VX = VX - VY (VF = 00 if VX < VY, VF = 01 if VX ≥ VY)
***9XY0**	Skip next instruction if VX ≠ VY
***AMMM**	Let I = 0MMM
BMMM	Go to 0MMM + V0
***CXKK**	Let VX = random byte (KK = mask)
***DXYN**	Show n-byte MI pattern at VX, VY coordinates. I unchanged. MI pattern is combined with existing display via EXCLUSIVE-OR function. VF=01 if a 1 in MI pattern matches 1 in existing display.
EX9E	Skip next instruction if VX = hex key (LSD)
EXA1	Skip next instruction if VX ≠ hex key (LSD)
***FX07**	Let VX = current timer value
***FX0A**	Let VX = hex key digit (waits for any key pressed)
***FX15**	Set timer = VX (01 = 1/60 second)
***FX18**	Set tone duration = VX (01 = 1/60 second)
FX1E	Let I = I + VX
***FX29**	Let I = 5-byte display pattern for LSD of VX
***FX33**	Let MI = 3-decimal digit equivalent of VX (I unchanged)
FX55	Let MI = V0:VX (I = I + X + 1)
FX65	Let V0: VX = MI (I = I + X + 1)

Fig. 1
CHIP-8 vocabulary includes 31 instructions. Ones with asterisks are used more commonly than others and so are explained in text. Parts of instructions in **bold type** identify the instruction; the rest of the instruction message tells what variables, memory locations, etc., are to be acted upon. Programs like CHIP-8 are dynamic systems, with new features being continually introduced. If it is important that your information is current, make sure you have the latest copy of the Manual.

N refers to the number of bytes to be used in setting up a pattern to be displayed.

How to read a CHIP-8 program

A sample working program shows how CHIP-8 works.

Fig. 2 lists a sample program, called "Jumping X and O." Here is what the program is written to do. First, a solid 6x6-spot block appears in the upper right quadrant of the tv display. A 5x5 "X" pattern appears in the center and jumps randomly to a new location every 1/5 second. When the X overlaps the 6x6 block, the X disappears, an "O" pattern appears in the center of the screen, and repeats the process, being replaced by the X when an overlap with the block

Address	Instruction code	Comments
0200	**A24C**	Set I to block pattern
0202	**6530**	V5 = 30 } Coordinates of block
0204	**6604**	V6 = 04 }
0206	**D566**	Show block at V5, V6
0208	**A240**	Set I to X pattern
020A	**2212**	Do subroutine at 0212
020C	**A246**	Set I to O pattern
020E	**2212**	Do subroutine at 0212
0210	**1208**	Go to 0208 (return to X pattern)
0212	**611E**	} Set V1, V2 to center coordinates
0214	**620D**	}
0216	**D125**	Show the pattern
0218	**630C**	Set V3 to 0C (= 1/5 second)
021A	**F315**	Set timer from V3
021C	**F407**	Timer → V4
021E	**3400**	Skip if V4 (timer) = 0
0220	**121C**	Return to 021C if V4 ≠ 0
0222	**4F01**	Skip if VF = 01 (checking for overlap)
0224	**122E**	Go to 022E if VF ≠ 01 (overlap, switch patterns)
0226	**D125**	If no overlap, show old pattern to erase
0228	**C13F**	Random number (6-bit) to V1 } new coordinates
022A	**C21F**	Random number (5-bit) to V2 }
022C	**1216**	Go back to 0216 to show pattern in new location
022E	**D125**	Show old pattern to erase
0230	**00EE**	Return from subroutine (will switch patterns)
0232	**0000**	Space for future changes
0234		
0236		
0238		
023A		
023C		
023E	**0000**	Space for future changes
0240	**8850**	
0242	**2050**	} X pattern
0244	**8800**	}
0246	**F888**	
0248	**8888**	} O pattern
024A	**F800**	}
024C	**FCFC**	
024E	**FCFC**	} Block pattern
0250	**FCFC**	}
0252		
0254		

Fig. 2
"Jumping X and O" program was designed to explain a number of CHIP-8 instructions and entertain *RCA Engineer* editors. Instruction explanations in text use specific lines of the program listing as examples. See Fig. 3 for the program in action.

occurs. The program continues until the machine is stopped; Fig. 3 shows the program in action.

This sample program will be used to illustrate the VIP instruction set, but before proceeding to those details, it would be worthwhile to look at the format of the sample to note some of the conventions that are used.

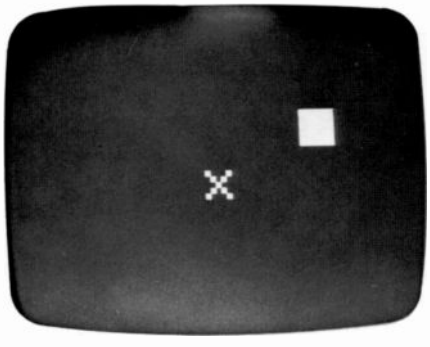
The first column in the program contains the memory addresses in which the instructions are stored. All instructions occupy two adjacent byte locations, so it is standard practice to write them as complete 4-digit words, using only even-numbered memory locations to identify them. It is understood that the adjacent odd-numbered locations are used for the second byte of each word.

The memory column does not, of course, get entered into the computer—it tells "where," not "what." But still you must keep track of these addresses and list them for each instruction. As you will see, all references made in the program to other parts of the program use these addresses to find the right place. And in using these addresses, *remember hex.*

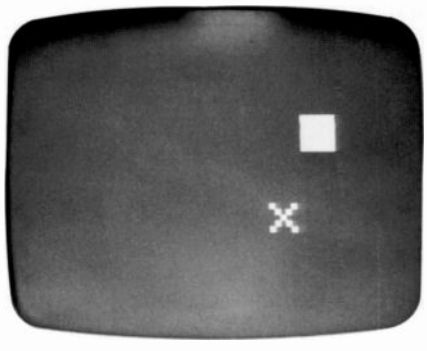
Note that the first instruction of the sample program is stored in location 0200. This is a mandatory requirement, since CHIP-8 was designed with the assumption that the program starts there. Don't try to surprise the interpreter on this point—it will always have the last laugh.

The next column in Fig. 2 contains the actual 2-byte instructions (described in the next sections). These two bytes are the only part of the program that actually goes into the machine.

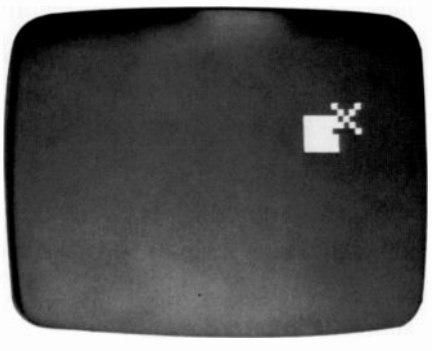
The last column contains your comments on each instruction—a memo to yourself and to whoever else studies your program. Each comment should be a brief description, either in mathematical or English form, of what the instruction does. The comment is of no value to the computer and plays no part in executing the program, so you can easily leave it out if you are willing to accept that at some later date you will not be able to understand what your own program is all about. The value of good comments cannot be overstated. Every nontrivial program, without exception, will eventually be revised by the writer or by someone else and, while using good comments will not guarantee easy revision, their lack will guarantee that a simple change becomes a titanic struggle. You will eventually develop your own natural shorthand for comments, but in the beginning, be generous with them.



Program starts with an X in the center of the display and a block at the upper right corner.



The X jumps to a new (random) location every 1/5 second.



When the X and the block overlap,

Fig. 3

What the instructions mean

CHIP-8 has 31 instructions, but you can write most programs using only 20.

We will concern ourselves with the 20 instructions in Fig. 1 that are marked with asterisks. The remaining 11 are no less legitimate, but tend to be more valuable to the advanced VIP user. Of the 20 sample programs in the VIP manual, all but one or two use only the group of instructions we will examine, so there is no doubt that many interesting programs can be written using them.

Note that the instructions, except the first two, have one, two, or three numbers for you to provide—these are the X, Y, I, K, and M discussed in the "vocabulary" section. It is important to write all the remaining digits—identified by **bold face** type in the following explanation—exactly as they appear in the general form, since the interpreter uses them to identify the operation.

For the sake of cohesiveness, the instructions that we will look at have been arbitrarily divided into six groups:

Group 1 — manipulating the variables

6XKK makes KK become the new value of variable X. As it is the first instruction considered, a brief look at its structure is in order. The "6" is the code that the interpreter will use to translate this operation into the proper set of basic machine instructions. "X" is the identifying number of the variable that you have selected, and "KK" is the hex value to which you want to set variable X. Thus, 6530 (line 0202 in Fig. 2) changes the value of variable 5 to the value 30.

7XKK has the same structure, but the "7" translates to an addition. The instruction adds KK (remember hex) to the present value of VX. For example, 7A1B increases the current value of variable A by 1B.*

8XY**0** Note that in this case the first and last digits are both part of the instruction code. By using this instruction, the value of variable Y also becomes the value of variable X. For example, 8320 copies the value of variable 2 into variable 3.

*This same instruction can be used for decrementing VX by taking advantage of the fact that the variable is a 2-digit number with no carry into a third digit. In such a system (to use a decimal example) you can subtract 1 by adding 99, since n + 99 = n + 100 - 1 and when you throw away the 100 (the 3rd digit) you have n - 1. It is analagous to a 2-digit odometer in an automobile; if you advance it 99 places it brings you back to 1 short of your starting point. In the hex system you add FF (the biggest number) in place of 99 to subtract 1, but the principle remains the same and can be extended to subtract any number you wish.

Group 2—transfer of control

The computer normally steps through a program in sequence. It sets up an instruction from memory, executes it, and then proceeds to the instruction in the next memory location. But the ability to depart from this sequence is one of the essential features of any computer, and CHIP-8 provides a flexible system for doing so.

Breaking out of sequence may be done in two ways—unconditionally (the jump is made whenever the instruction is encountered) or conditionally (the sequence is or is not broken, depending upon whether some condition has been satisfied). This group of instructions deals with the unconditional transfers, often called "go to," or "jump."

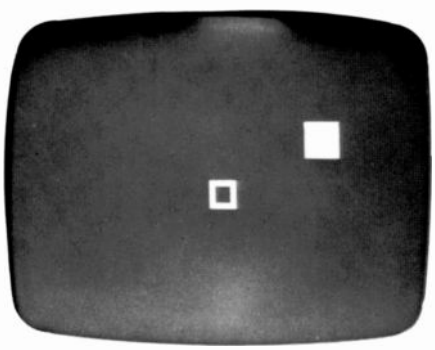
1MMM has the computer perform, not the next instruction in memory, but the one stored at location 0MMM. As an example, see Fig. 2, line 0210. Normally, the next instruction to be done would be 0212 but instead, the 1208 instruction makes the computer go back to do the one at memory location 0208. Jumps may be either forward or backward in the program, and there is no restriction on how many places may be skipped.

An unusual use of this instruction is for stopping the machine. When a program comes to a point demanding some kind of reset or fresh start, a 1MMM instruction referring to its own memory location will put the VIP into an endless cycle until the operator takes the appropriate action.

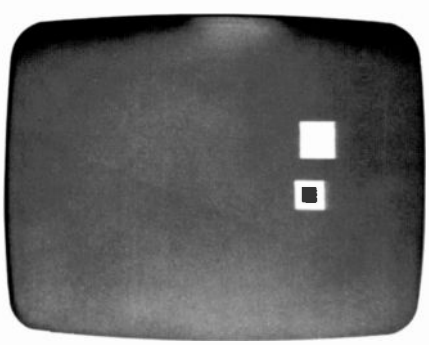
2MMM has the computer do the subroutine at 0MMM, and **00EE** has it return from the subroutine.

These two instructions must be considered together. While the 2MMM instruction also transfers unconditionally, it is not the same as the previous case. The difference lies in the nature of a subroutine, which is a package of instructions performing some function that will be used at more than one point during the course of a program. (For example, computing sin x or the roots of a quadratic equation might be subroutines in a scientific program.) When a subroutine is finished, you want the program to return to the point from which the jump was made and pick up its normal sequence. Since the subroutine may be entered (called) from any place in the program, the obvious question is "How do I know which place is the right returning point?"

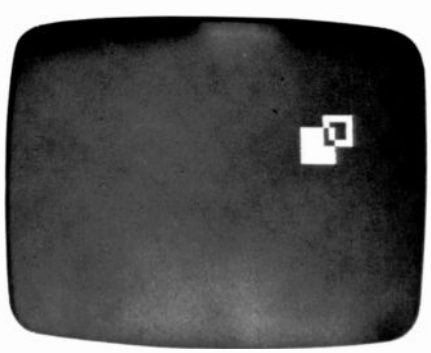
In Fig. 2, lines 020A and 020E present a typical case, in that they both have the program jump to the same point, 0212, the beginning of a subroutine that makes the X or O jump



the X disappears and an O appears in the center of the display.



The O then starts jumping to random locations



until it overlaps the block, and the whole process begins over again.

every 1/5 second. Clearly, before this program transfer is made; the computer must store the location of the "jumping-off" point so that at the end of the subroutine (line 0230) it can correctly return to 020C or 0210 as appropriate. The 00EE—return from subroutine—instruction makes the proper transfer back by retrieving the stored "jumping-off" address.

CHIP-8 provides enough storage to "nest" up to 12 subroutines. This means that, in the course of executing one subroutine, another may be called and executed, and so on. The situation is analogous to the use of nested parentheses in an algebraic statement, and the same care must be taken so that each unit is begun and terminated properly with the 2MMM and 00EE instructions.

Group 3—conditional skips

This group of instructions alters the sequence of events in a program depending on whether or not some stated condition has been met. They give the computer its most powerful function, the ability to follow one course of action or another, depending on the value of data in the machine.

In these CHIP-8 instructions, if the stated condition *is* met, then the computer skips the next instruction in line and executes the following one. If the stated condition is *not* met, then the computer continues in its normal sequence, i.e., it does not skip. In a program, it looks like this:

Line n states the condition to be met.
Line n+1 is executed if the condition is *not* met.
Line n+2 is executed if the condition *is* met.

In Fig. 2, lines 021E through 0226 show examples. Note that when the condition is not met, there is only space for one instruction (line n+1). In most cases that instruction will have to be an unconditional jump to the place where the relevant code has been stored.

The instructions in this group are:

3XKK Skip next instruction if VX=KK
4XKK Skip next instruction if VK ≠ KK
5XY0 Skip next instruction if VX = VY
9XY0 Skip next instruction if VX ≠ VY

Group 4—memory control

AMMM sets the pointer, I, to the memory address 0MMM. All following instructions that use memory will go to this address until you do something that changes I. See Fig. 2, lines 0200 and 0208, for an example.

FX29 sets up the pointer for displaying hex digits. Part of the VIP is an "operating system," which takes care of loading memory from the keyboard, reading to and from the tape cassette, etc. The bit patterns for displaying any of the hex digits are already stored in memory as part of this operating system. The FX29 instruction provides access to those patterns, so that you do not have to work them out for yourself. It sets I to the correct address of the 5-byte pattern for the hex digit in the least-significant digit of VX.

FX33 translates the value of VX into its 3-digit decimal equivalent and stores the result in the memory starting at I. This instruction is very useful for presenting video-game scores in decimal form.

Group 5—miscellaneous control

The VIP has a timer and an audible tone generator. The timer, which automatically counts down to zero when it is set by the programmer, is useful for such things as making a display flash, timing a permissible period for some keyboard action to take place, etc. The audible tone, whose duration you can set, can be used to celebrate collisions, end of program, etc.

FX15 sets the timer to the value of VX. The smallest value, (01 in VX) is equivalent to 1/60 second, so the maximum time (FF in VX) is 255x1/60 = 4.3 seconds.

FX07 transfers the current timer value into VX. This instruction lets a program monitor the progress of a countdown. Fig. 2, lines 021A through 0220, demonstrates a typical timing loop. V3, which stores the hex number equivalent to 1/5 second, is transferred to the timer (line 021A), and from there to V4 (line 021C), where it is tested (lines 021E, 0220) until it is found to equal zero.

FX18 turns on the audible tone for the length of time specified by the value of VX. Again, the smallest increment is 1/60 second.

Caution—In the above three instructions, since 1/60 is a decimal number, you will be going back and forth between the hex and decimal systems. Watch your arithmetic.

CXKK sets a random byte into VX. KK acts as "mask"—where KK has a 0 bit, no entry is made into VX. The mask is a way of controlling the range of the random number; for example, C703 would set into V7 a 2-bit random number (since 03 has two bits) with equal probability of any 2^2=4 values. In Fig. 2, lines 0228 and 022A provide further examples.

Group 6—input and output

FX0A sets the least-significant digit of VX to the value of the next key pressed. The program will pause here until some key is pressed.

00E0 erases the entire display.

DXYN displays a pattern on the screen. Most of your attention in programming the VIP will center around the display. For full information, you should refer to the *VIP Instruction Manual,* since we can concern ourselves here only with those characteristics that reflect into this CHIP-8 instruction.

In this instruction, N defines how many bytes make up the patterns you want to display, X and Y describe where on the screen you want it to appear and, while I does not appear in the code, its current value determines where the computer will find the first byte to be shown. Therefore, before this instruction can be used you must have taken care of the

following functions: 1) the bit pattern to be displayed must have been set up in the memory; 2) I must have been set (be pointing) to the address of the first byte in the pattern; and 3) the two variables defining the pattern's screen location must have been set to the proper values, X for horizontal (increasing left to right), Y for vertical (increasing top to bottom).

For example, in Fig. 2, line 0206, D566 means "show a 6-byte pattern taken from the memory starting with I." The upper left-hand corner of this pattern will be at coordinates specified by V5 and V6. Note that the three preceding instructions have set the values of the coordinates (lines 0202 and 0204) and have set I to 024C (line 0200). At location 024C, the beginning of the pattern has been stored, so everything is all set to go.

In the VIP, the pattern that is being displayed is superimposed over any existing pattern on the display, with the feature that where a new spot overlays an existing spot, the screen is erased. When this happens, the computer puts 01 into variable F, making possible the programmed detection of overlaps or collisions between the old and new patterns. Line 0222, in Fig. 2 for example, senses VF to see whether the jumping pattern has overlapped the fixed block. If it hasn't (VF=0), line 0226 starts the procedure to move the X or O randomly again. If there has been an overlap (VF=1), line 0224 causes the program to switch from X to O, or vice versa, and start over. You can also use the "blankout on overlap" feature in selectively erasing all or part of a pattern by rewriting it in the same place.

Writing a program

Familiarity with the CHIP-8 instructions does not, by itself, make you a programmer, any more than learning the alphabet made you literate.

Learning the instructions is a necessary first step which, with the exercise of some practice and patience, can lead to whatever level of proficiency you may want to achieve in writing programs. In fact, one of the most important uses of the VIP is in providing a fun way of acquiring exactly that skill.

While the actual design of a program is something you must do for yourself, a few paperwork tools and helpful hints can make the job easier and more organized. Here are some suggestions to help you get started.

Many people find it useful to make a flowchart of their programs before they start coding. It's a good way of organizing your thoughts, and it lets you test the program on paper before you have invested a lot of time in coding. It is also very helpful in identifying those parts of the program that could be done as subroutines, and so reducing the amount of coding you have to do. But with or without a flowchart, think about what you want to do before you plunge ahead; a half hour of planning will more than pay for itself in easier coding.

Keep a list of the 16 variables and what meanings you have assigned to them, i.e., what they represent in your program. Note that VO and VF are occasionally set aside for special use. Make sure they're clear and available when the instruction will be using them.

Be sure to keep track of the memory addresses in which your program is stored. For the first rough drafts you might want to leave spaces every so often in the program to allow for changes. These spaces can later be bridged by unconditional jumps. If you decide to close up the gaps later, or if you find you have to squeeze in another line of code, make sure that you have properly modified all the "jump to" addresses that have been affected. Another trick is to label "jump" points with arbitrary designations (e.g., Greek letters) until you are satisfied that you have progressed to the point of specifying accurate addresses.

Make a chart of the display to help you plan the program, work out the bit patterns, and establish the coordinates. It should also carry the memory addresses in which the data to be displayed is stored.

If you have occasion to put data, such as display patterns, in the memory, keep a list of where you have put them. You may never find them again otherwise.

Don't forget that CHIP-8 expects to find your program starting in location 0200. If for some reason you want to violate that rule, then 0200 must have an unconditional jump to whatever starting point you have selected.

Be generous with your comments. It's surprising how difficult it becomes, even after a short coffee break, to remember or reconstruct the brilliant trick you thought up while you were deeply engrossed in your creative approach to the problem.

Don't forget to return properly from a subroutine, have fun, and *remember hex*.

Taking the next step

If you are interested in knowing more about CHIP-8, send for the additional information that is available. The package includes a step-by-step description of how the sample program was developed, and would be a good guide if you are starting out on your first adventures in programming. It also includes copies of the following charts and forms which you can reproduce for your own use: a display layout chart, a variables assignment sheet, and a program coding sheet. Write or call the Editor, *RCA Engineer*, and ask for the "Good Guide Special." The address is Building 204-2, Cherry Hill, N.J.; telephone ext. 4254.

Acknowledgments

Ann Merriam, J.W. Wentworth.

References

1. COSMAC VIP instruction Manual, RCA Solid State Division, Somerville, N.J.
2. Microcomputer Fundamentals (CEE Course C-51) RCA Corporate Engineering Education, Cherry Hill, N.J. See Study Guide, Chapter 8.

Reprint RE-23-6-12
Final manuscript received December 7, 1977.

Software: microcomputer vs. minicomputer

K. Schroeder

Changing from mini to micro?
Programming and debugging are significantly different.

Designing and implementing software for the microcomputer and the minicomputer are significantly different activities. Underlying the obvious similarities in the primary function of the software are important differences. Specifically, these two kinds of software are required to run on appreciably different hardware (Table I). They are required to perform applications of considerably different character. Also, they are written, edited, and debugged using different methodologies. The importance of these differences is not necessarily obvious to the uninitiated and the effects can become more or less significant, depending upon the specific hardware systems and applications under consideration. However, it is useful to generalize about these differences to understand what one may face when attempting to program a microcomputer for the first time after having had experience in programming minicomputer (or midi-computer) systems.

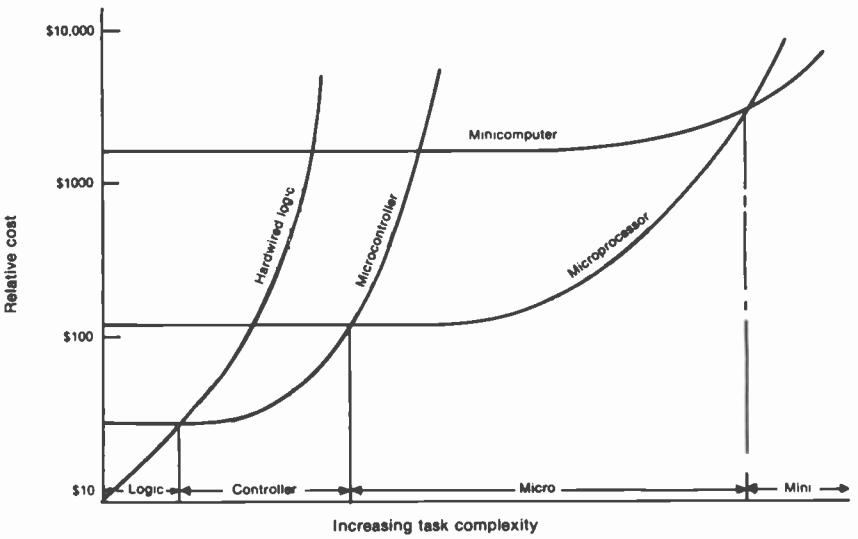


Fig. 1
As tasks become more complex, the most cost-effective means of performing them changes from hardwired logic to microcontrollers to microprocessors to minicomputers.

How hardware differences affect software

Microcomputers have traded computing power for economic and size advantages.

Microcomputers are generally less expensive and smaller hardware systems than minis, and therein lies their utility. Micros can bring intelligent control to applications that either do not need, or cannot justify, a larger, more expensive minicomputer (Fig. 1). However, this economic and size advantage is gained at the expense of computing power and hardware facility. This sacrifice is reflected in the microcomputer's software. A microcomputer programmer must often compensate for hardware limitations in software. Often the most significant hardware limitation is execution speed. However, even in applications requiring only modest computational speed, many other missing hardware resources must be compensated for in software, thereby lengthening and complicating the programming task.

Micros predominantly have shorter data words than minis do.

A computer's data-word size is its fundamental data representation. It specifies

Reprint RE-23-6-22
Final manuscript received February 28, 1978.

Table I
Differences in hardware between mini and micro are a major reason behind the differences in software.

Feature	Micro 0101100111010100	Mini 0101100111010100
Data-word size (bits)	4, 8, 16	12, 16, 18, 32
Execution speed (cycle time)	Slow (500 ns—10 μs)	Medium-fast (200 ns—1 μs)
Addressable memory	Small-medium (512-64k bytes)	Medium-large (4k-128k words)
Instruction repertoire size	Smaller (30-150 typical)	Larger (70-300 typical)
Assembly-language programming	Tedious, slower	More efficient, faster
Interrupt capability	Single-level static priority	Multi-level dynamic priority

the number of bits that can be stored into or retrieved from its main memory during a single memory cycle. (Even "bit" manipulations are usually word-addressed.) Generally, the larger the data word, the greater the efficiency and power of a processor's internal operations.

The majority of microcomputers have either 4-, 8-, or 16-bit data-word lengths. The 8-bit version presently dominates both the marketplace in dollar sales volume and current microprocessor-based design. This is partially because a byte (8 bits) is a convenient data representation for many micro applications. More significantly, however, the 4-bit versions have very limited data-handling capabilities and the 16-bit versions are considerably more expensive. Sixteen-bit micros are primarily selected to provide software compatibility with minicomputers for which software has already been written or for applications in which software compatibility with a mini is of paramount importance. However, as microcomputer prices continue to decline, the 16-bit machines will become more competitive and will become more widely used.

The minicomputer is commonly available in 12-, 16-, 18-, and 32-bit versions. The 16-bit-version mini dominates the market because it conveniently allows a larger data representation than the 8-bit micro, yet allows efficient byte handling when required by packing two bytes per data word. Many minis also facilitate byte addressing of memory to enhance their byte-handling capabilities.

Micros characteristically have smaller instruction sets.

Normally, a machine's instruction size is a small multiple (1, 2, or 3) of its data-word size. The instruction size is a direct indication of the computational power and size of a machine's instruction set. (This is the set of instructions directly executed by the hardware.)

The microcomputer, usually having a smaller data word, thus also has a smaller instruction size, limiting the power of its instruction set. The micro usually has fewer instructions, less powerful instructions, fewer memory addressing modes, and fewer data types that can be handled directly by the hardware. Thus, the microcomputer program requires more assembly-language instructions than the equivalent program implemented on a mini. This makes assembly-language programming a more tedious, less efficient, and more error-prone task for the micro than for the mini.

Memory addressing can be inefficient with micros.

One memory-addressing limitation problem encountered with micros and not with minis is "out-of-page" reference errors. A microcomputer often has "paged" memory, i.e., memory is divided up, for addressing, into blocks or "pages" and some of the micro instruction formats can only reference memory locations within the same page as the instruction. This technique of addressing is used to limit the number of bits required to specify an operand's address. However, when an attempt is made to reference, within one of these short instructions, a location outside the current page of memory, an "out-of-page" reference error occurs. This restriction can be avoided by using indirect addressing or using full-address instructions. These techniques, however, create inefficiences in execution speed or memory space and may not be desirable to use throughout a program. Anticipating and compensating for this addressing restriction complicates the writing of micro software.

Because micros have fewer general-purpose registers, intermediate results must often be swapped back and forth to memory.

Another feature of microprocessors that limits their computational power in comparison to the mini is their limited internal register sets. Normally, the micro has fewer hardware registers for use as accumulators or index registers. This can necessitate frequent saving and re-storing register contents into main memory to save intermediate data results or address pointers. This required swapping of information not only slows down execution speed, but forces the programmer to keep track of where such information is being stored and determine what allocation of those registers will minimize that program overhead.

Since micros have less computational hardware, more operations must be done in software.

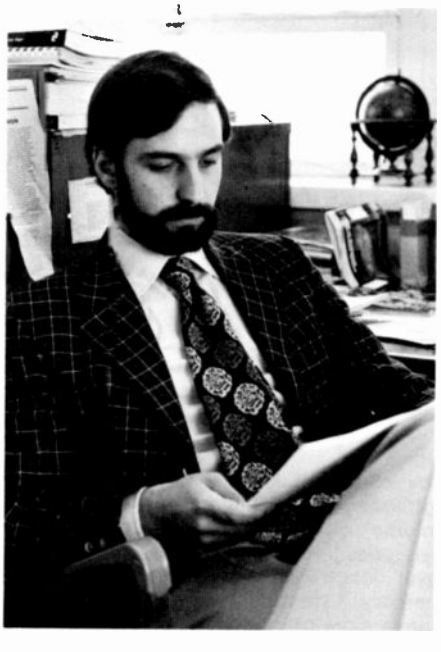
The micro usually lacks other computational hardware features that many minis use to speed execution of complex numerical calculations. Such hardware includes hardware multiply and divide (single and double precision) and multiple position shift facilities. Also missing are floating-point hardware facilities. Such operations must be done in software and become the responsibility of the programmer, thereby complicating his task. This added code can also considerably lengthen the program.

The stack facility available on many micros is limited, in contrast to the ones on standard minicomputers.

The micro's stack often requires the explicit handling of both the stacking data and the stack pointer register. A few micros implement a stack in a separate small memory space within an organization that effectively has an open bottom. Once the stack is filled, any attempt to push additional data onto the stack will destroy the first entry on the stack without any warning or hardware protection. In this organization, the size of the stack memory absolutely limits the depth of the stack. This stack limitation may restrict subroutine call nesting or the permitted level of context switching that the computer can

Ken Schroeder has eight years of software experience in both minicomputer and microcomputer systems. Now working on microcomputer-based consumer products, he has also worked with software for medical instrumentation, laboratory automation, and navigation satellites.

Contact him at:
TV Microsystems Research
RCA Laboratories
Princeton, N.J.
Ext. 3325



handle, since these actions normally require entries on the stack.

In contrast, minis normally implement their stacks in main memory, which gives virtually unlimited stack depth. On many minis, when an attempt is made to overflow the permitted stack area, a hardware indication is generated. This permits software to detect such an occurence and take appropriate action. Many minis have implicit stack-handling and will adjust stack pointers automatically. Some minis have facilities that automatically stack the program state upon interrupt or other context switching. Thus, using a stack facility on a micro generally requires more code and is more complex to program than on a mini.

Microcomputers generally have relatively primitive interrupt-handling structures.

Micros commonly only have a single level of hardware priority and often lack a vector-generation capability. In contrast, minis commonly have multi-level dynamic priority-arbitration schemes and also frequently have vector-driven response systems. When implementing an application requiring significant interrupt-response capability using a micro, the programmer must make up for this lack of hardware facilities in software, thereby complicating his programming task.

Microcomputer systems usually need external equipment for debugging.

Certain computer hardware features are often helpful when debugging software and diagnosing software failures. One of these is a hardware "trap"—vectoring the program to a specific address in memory upon the occurence of a predefined machine state. The attempt to execute an illegal instruction or address nonexistent memory are examples of "trap"-generating occurences. These "trap" features are standard on minis but are lacking on micros. The microcomputer programmer cannot, however, really compensate for them in software, so this function in the debug phase of software usually must be replaced by the use of a logic "analyzer" or other external debug hardware. The microcomputer programmer should become familiar with the use of such devices.

How application differences affect software

Attempts to save memory costs often lead to complicated unstructured programs.

Microcomputers are customarily applied in very cost-sensitive applications. Typically, these are applications with moderate- to high-volume system-replication requirements, where small individual economies reap large total savings. Minicomputers, in contrast, are more typically used in low- to medium-replication-volume applications, which are not typically as cost-sensitive.

Approximately 60% of the cost of the average microcomputer system, in final application configuration, is memory cost. Since assembly-language programming can generate code that is more memory-efficient than compiler-generated code, it tends to dominate micro programming (Fig. 2). A determined effort is usually made to squeeze the required software into the minimum amount of memory. This activity is commonly called "bit-bumming." However, recent efforts to bring modern engineering techniques to the "art" of writing software has led to the foundation of a new branch of study called "software engineering." This new discipline has shown that "bit-bumming" and other software techniques that sacrifice code clarity and structure to minimize program space have serious side effects in programs of any significant size. Specifically, such efforts lead to the production of unstructured programs, which are difficult to debug, difficult to document and, most important, difficult to understand. Such programs can create very expensive support problems and can only be cost-effective in applications with very large replication volumes and applications that will remain very stable and will not be modified or extended after initial comple-

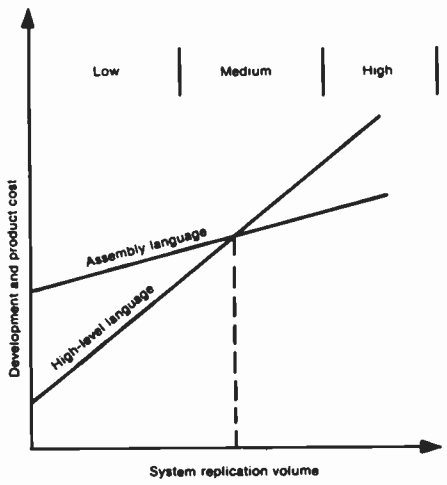


Fig. 2
The choice of language level used in a software system depends on replication volume of system. Assembly language generates more efficient code than high-level languages, and so requires less memory. Assembly language, however, requires more programming effort.

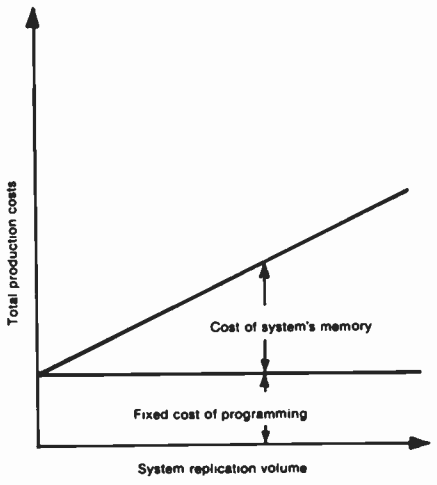


Fig. 3
Cost to deliver a system depends on both programming and memory costs. (Slope of memory costs may vary with price breaks for volume purchases.) Since saving a few bytes can potentially reduce the number of chips required, "bit-bumming" becomes a necessary evil for microcomputer programmers.

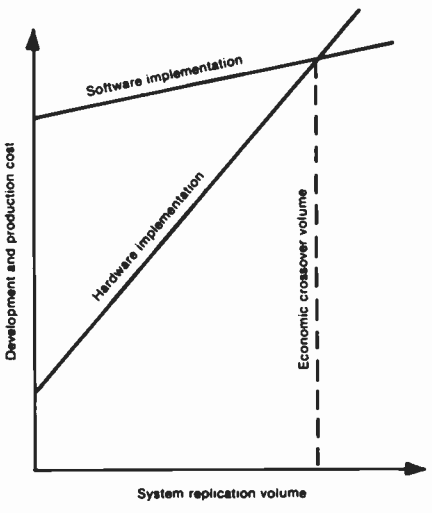


Fig. 4
Hardware/software interface for microcomputers often has the possibility of tradeoffs. In this example, a serial input-output port to a terminal device can be done in software (high initial cost) or a USART chip (increasing cost with volume).

tion. Since this is the environment in which many microcomputers are used, "bit-bumming" is a skill often required by microcomputer programmers. This is especially true since saving a few bytes can potentially reduce the number of memory chips required (Fig. 3).

Memory efficiency is not as crucial in typical mini applications. Additionally, memory for minis normally is only available in 4-k word quantities, so unless this increment boundary is avoided, no cost savings are realized by reducing memory requirements.

Because microcomputers usually work in a dedicated task environment, the programmer must write software normally handled by the mini's supervisory software.

Microcomputers are primarily used in dedicated single-task programming applications. Normally, software in such an environment controls the base-machine hardware and is not integrated into an existing operating system or standardized software monitor. The microcomputer programmer directly programs all software functions normally handled in a mini by such supervisory software. For example, the maintenance of the system clock and the control of all peripheral devices are the programmer's responsibility. Normally, a microcomputer system has fewer computer peripherals to handle than a typical mini system; however, the nature of these peripherals is appreciably different. Typical minicomputer interfaces make peripherals appear logical and time-independent, i.e., all the software operations required are clearly logically related to obvious functions of the device. Typical micro devices have simple controllers, which require more detailed software control and can impose serious timing constraints upon the program in controlling the hardware—constraints which, if violated, can cause serious and difficult-to-isolate intermittent problems.

Suitable "off-the-shelf" application program packages are not often used with micros.

Applications involving minis often use various mathematical and application software packages available from the hardware manufacturer to facilitate system implementation. Also, many mini programmers write general-purpose software packages for a particular application area and use them repeatedly in subsequent applications to improve software-development efficiency. In the micro world, such generalized packages are rarely used in final product configurations. More characteristically, concise and efficient code is written for each application and is customized for maximum efficiency for the individual case, thus making programming less efficient and driving up software development costs.

Programmers have to make sure that the relatively slow micro systems are not too slow for the task at hand.

Micros are generally put to work in applications for the monitoring, analysis, and control of time-dependent (real-time) processes. Since micros have slow execution rates, it is often necessary to write very efficient programs to meet performance requirements. Programming in high-level languages has been shown to be much more efficient than programming in assembly language and so is rapidly dominating mini software. Assembly language, however, still dominates microcomputer programming in order to meet execution-speed requirements, since compilers do not yet generate very efficient code. The attempt to save execution time has an equivalent activity to "bit-bumming"—using similar unstructured programming techniques that minimize the execution time of programs but at the expense of clarity. This approach has the same inherent program-debugging and product life-cycle support problems as "bit-bumming" and thus should only be a last-ditch attempt to save an effort about to fail to meet required speed specifications. Intelligent system design dictates that a projected 50% of capacity throughput surplus be included to both facilitate unexpected system growth and anticipate throughput-requirement and load-fluctuation estimation errors. This philosophy should preclude the need for such unstructured code optimization. Compromises in these design guidelines may be necessary in applications with large replication volumes and correspondingly high cost sensitivities.

How differences in system implementation affect software

Microcomputer programmers have to be more careful with addressing memory, which is allocated in disjointed segments.

To gain reliability and cost efficiency, a microcomputer's (final-product) program is held in primary memory and is not kept in mass-storage peripherals. Primary memory is normally segmented into a nonvolatile read-only memory (ROM) program-storage area and volatile (read/write) random-access memory (RAM) scratchpad area. This partitioning of memory space imposes another constraint on the programmer—the program must be partitioned into disjointed ROM and RAM sections. Programming must not attempt to write into ROM space nor execute code in RAM space inadvertently. Additionally, the stack area must be maintained in RAM. Observing the boundaries of these memory areas is the programmer's responsibility. This is in contrast to minicomputer systems, where typically the program is loaded from some mass-storage peripheral device into memory composed uniformly of nonvolatile core memory, in which no such partitions exist.

The hardware-software interface is much closer for the microcomputer.

Because microcomputer hardware systems are custom-made for specific applications, in contrast to the general nature of minicomputer system hardware, the two systems have major differences in the integration of hardware and software. In microcomputer systems, the hardware/software interface is closely coupled, i.e., one is often directly traded off for the other. In contrast, the fundamental hardware is much more standardized with the minicomputer, so software is written to run on that hardware without substantial change.

In the micro's case, for example, a serial input-output port to a terminal device customarily may either be implemented in software or done in hardware external to the CPU (Central Processing Unit) by a Universal Synchronous-Asynchronous Receiver Transmitter (USART) chip. In a mini system, such an interface is almost always performed by a standardized serial interface board. The engineer who implements a microcomputer system must understand such hardware/software trade-offs (Fig. 4). Thus, a microcomputer programmer must be more familiar with hardware than his minicomputer counterpart.

Software development is harder in micro systems because of the lack of peripherals so useful in debugging and simulation.

Custom microcomputer systems are very well suited for efficiently performing well-defined relatively-fixed tasks. Unlike minicomputers, they are not well suited for general-purpose computation or software

Table II
Software development tools are considerably more primitive for the micro. Of the applicable tools listed here that are used extensively with minis, many are nonexistent or less powerful with micros.

Text editor	For composing and modifying program source.
Assembler	Translates assembly-language programs into machine code.
Macro assembler	An assembler that permits the representation of commonly appearing sequences of instructions with shorthand "macro" names.
Cross-assembler	An assembler that executes on one (host) computer, but generates machine code for another (target) computer.
Compiler	Translates a high-level-language program into a language suitable for a particular computer.
Cross-compiler	A compiler that executes on one (host) computer, but generates code for another (target) computer.
Loader	For loading an executable module from some peripheral device into memory.
Linking-loader	A loader that combines many relocatable object modules into an executable module. It makes appropriate modifications to each module for resolving changes in references between the modules.
Cross-reference listing	An assembler output that lists all references made to each label or other symbol in the program.
Debugger	Permits the testing and verification of a program's operation by observing intermediate results at various stages of execution.
Debugger-simulator	A debugger that uses simulation to run on one machine and facilitate the debugging of a program written to run on another machine.

development. They generally lack three important system development tools: 1) the large secondary storage (disks, tape drives, etc.) required to hold utility programs; 2) language translators (assemblers and compilers); and 3) high-speed hard-copy devices (line printers, etc.), which are desirable during program development, but are rarely required in a microprocessor's final configuration. These facilities are required in any significant software development effort. Thus, microcomputer software is often developed, simulated, and initially debugged on alternate computer systems—timeshared systems, minicomputer systems, and specially-configured (typically more expensive) microcomputer-based development systems.

In these development systems not based on the micro, the language-translation programs used to convert programs into micro machine code are called "cross-assemblers" and "cross-compilers." These programs run on one machine, the larger "host" development computer, and produce code for the microcomputer or "target" machine. Additionally, these "host" machines often also have "simulator-debuggers," programs that simulate the running of the "target" processor and help debug the machine code by using the significant resources of the "host" system. In contrast, most mini software is developed on the mini itself.

The software tools (Table II) available to help develop software for micros are considerably more primitive than those available for minicomputer software development. The text-editing systems available for micros are considerably less powerful. Many micros lack the availability of macro-assemblers and linking loader facilities. Also, few high-level-language processors generate code for micros. In fact, many micros have no compiled high-level languages at all (but this is rapidly changing). Many manufacturers only provide cross-compilers which must be run on "host" development computer systems and have no resident versions that run on the micro itself. Resident software is, however, becoming more common as language processors and text editors can be stored on a single chip. The most popular high-level language in the micro world at present is BASIC, an interpretive language. This is a reflection of the efficient use of memory characteristic of interpretive language implementations. Interpreters are, however, often not acceptable in real-time applications because they execute programs slowly, so assembly language still dominates the programming of micros.

Conclusions

A large number of contributing factors makes programming microcomputers different from programming minicomputers. The relatively limited hardware facilities of the micro requires software to perform functions normally available in hardware on the mini. The lack of efficient high-level languages for the micro makes assembly-language programming dominate micro applications, whereas high-level languages dominate mini applications. The limited instruction set of the micro relative to the mini makes assembly-language programming more tedious and complex on the micro. The typical area of application of micros gives these systems higher cost sensitivity than typical mini applications. This leads to extensive custom hardware in micros and also to an increased degree of interaction between hardware and software design not found in typical mini systems. The software tools available for developing software for the micro are appreciably different and less powerful, requiring the programmer to develop different implementation methodologies. Thus, aside from the obvious similarities of the primary function of the software, developing software for the micro and the mini can be appreciably different activities.

References

1. Schroeder, Kenneth; "Microcomputers vs. minicomputers: selection criteria," *IECI '77 Proceedings*, pp. 190-194 (Mar 1977).
2. Ogdon, Carol A.; "Fundamentals of microcomputer systems—chapter 4," *Mini-Micro Systems*, pp. 72-79 (Nov-Dec 1977).
3. Bass, Charles; and Brown, Dean; "A perspective on microcomputer software," *Proc. IEEE*, pp. 905-909, Vol. 64 No. 6 (Jun 1976).
4. Gibbons, Jim; "When to use high-level languages in microcomputer-based systems," *Electronics* (Aug 1975) pp. 107-111.

FLECS: a structured programming language for minicomputers

T. M. Stiller

The FLECS language—FORTRAN plus some added features—makes programs easier to read, write, and modify.

This article is not intended as a tutorial on the subject of structured programming, as several adequate texts on that subject already exist (for example, see Ref. 1). Our discussion will be limited to a few general concepts of structured design, a specific language facility to support them, and some examples of the "structured approach" to software development.

We have chosen FLECS (*F*ORTRAN *L*anguage with *E*xtended *C*ontrol *S*tructures) as the illustrative vehicle because it is in the public domain, has already been adapted to several computers in use within RCA, and can be readily adapted to almost any computer for which a FORTRAN compiler is available. While FORTRAN is not the most desirable language in which to develop software, it is relatively powerful, reasonably efficient, and almost universally available for mini- and larger computing systems.

What is FLECS?

FLECS is a programming language that was developed by Terry Beyer at the University of Oregon; it consists of all the statement types supported by any particular FORTRAN compiler plus a dozen additional control statements. The added statement types produce an organizational facility that complements the program development process, rather than increases the number of details requiring attention; they do not, however, restrict the programmer from using any special features of the base language. Used properly, these control statements produce a program with a logical structure that can be readily distinguished from the details of its implementation. Programs written in this composite language are passed through a FLECS preprocessor, which produces: 1) an intermediate file in which the control structures have been converted to acceptable FORTRAN equivalents; and 2) an indented source listing for documentation purposes. The intermediate file is then compiled in the normal manner to produce an executable program. Additional details of this process may be found in the *FLECS User's Manual*.[2]

What is a structured statement?

A structured statement is a compound statement that consists of two parts: a *control phrase* and its *scope*. The scope of a structured statement consists of one or more statements which change the state of the program's working storage, communicate with peripheral devices, perform function or subroutine calls, etc., and in that context, may be considered the "active element" of the statement. The control phrase of a structured statement defines the conditions under which its scope is to be executed. The control phrase is also composed of two elements: a *keyword,* which specifies the nature of the control, and a *specification,* which identifies the control parameters. These terms are illustrated in Fig. 1, where we analyze the two structured statements native to FORTRAN: the logical IF and the DO.

FORTRAN's structured statements have a number of inconsistencies.

Although the examples of Figs. 1a and 1b are both structured statements in the sense we have defined them, there are several annoying inconsistencies between them. First, the scope of the logical IF is limited to exactly one simple statement, but the scope of the DO may contain more than one statement. Second, the specification of the logical IF is enclosed in parentheses, but the DO specification is not. Finally, the specification of the DO statement requires an ancillary clerical detail, that of inventing a statement number to signal the end of its scope. This last point is more bothersome than the others, since it introduces a chore that has nothing whatever to do with the programming task at hand.

In contrast to this, FLECS provides a uniform format for all structured statements.

First, the scope of a FLECS structured statement may consist of a single simple statement or any number of statements, simple or structured, terminated by a FIN statement. Second, the specification of a control phrase is always enclosed in parentheses. Third, and most important, the need for statement numbers as

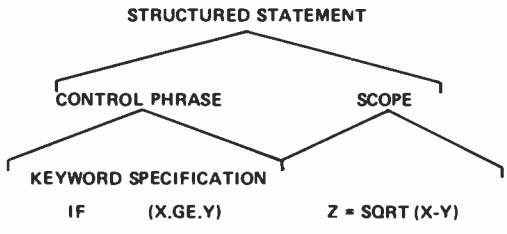


Fig. 1a
Logical IF is always written as one line; specification is within parentheses. Compare with DO statement in Fig. 1b.

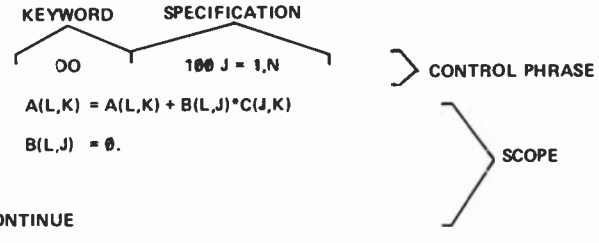


Fig. 1b
DO statement takes many lines; specification is not written within parentheses; programmer must invent statement number to signal end of statement's scope.

Structured statements in FORTRAN have a number of inconsistencies.

destinations for GO TO statements or terminations for DO loops can be completely replaced by the use of the appropriate structured statements. Finally, as a matter of convenience to the programmer, a structured statement whose scope consists of a single simple statement may be combined with its control phrase on a single line. All of the examples shown in Fig. 2 are valid FLECS structured statements.

Why are structured statements important?

Structured statements are important in software development because they complement a technique commonly used when transforming the statement of a physical problem into a computer program. Even the most complex piece of computer software consists of little more than sequences of statements to perform some specific actions and the decision-making logic to determine the conditions under which those actions should be performed. The structured statement, with its concept of control phrase and scope, provides a natural methodology for decomposing a physical problem into a series of programming steps. On the other hand, the rather limited structured statements available in FORTRAN do very little to support this methodology.

What types of structured statements are available?

The types of structured statements supported by FLECS divide rather naturally into two categories: decision structures and looping structures. Decision structures may be further subdivided into skip-action, alternative-action, and select-action types. The looping structures are subdivided on the basis of the number of times the loop is executed: a fixed number of times, a variable number of times (including none), or a variable number of times (but at least

```
IF  (X.GT.Y)
    Z = SQRT(X-Y)
    Y = Y**2
    FIN
```

Fig. 2a
IF statement with multi-line scope. (If control phrase is a single statement, entire structured statement can be written on one line.)

```
DO  (I = 1,N)  A(J,K) = B(J,I)*C(I,K)
```

Fig. 2b
DO statement with a single-line scope. Note that statement number for termination of loop is not needed.

```
DO  (I = 1,N)
    A(J,K) = B(J,I)*C(I,K)
    IF (J.EQ.I)
        D(I) = B(J,I)
        FIN
    FIN
```

Fig. 2c
Nested multi-line structured statement. Note indented structure. FIN statements, rather than "invented" statement numbers, terminate loops.

FLECS has a uniform format for all structured statements: they may consist of either a single statement (Fig. 2b) or a number of statements (Figs. 2a and 2c) ending with a FIN statement; specifications are always within parentheses; and statement numbers are not needed for termination of DO loops.

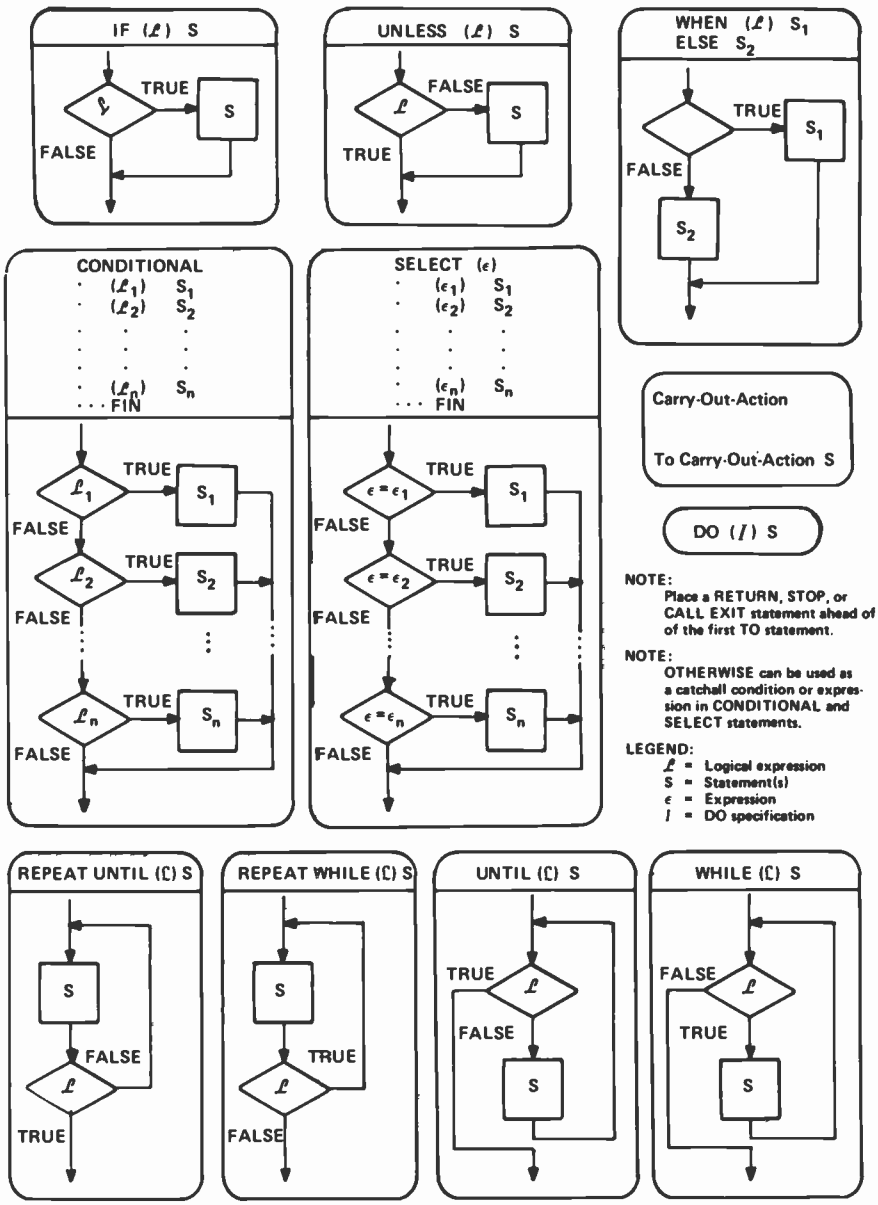


Fig. 3
All the structured statements used in FLECS, in programming and flowchart form. IF, UNLESS, WHEN/ELSE, CONDITIONAL, SELECT, REPEAT UNTIL, REPEAT WHILE, UNTIL, and WHILE can cover all programming possibilities.

once). Fig. 3, which has been reproduced from the FLECS User's Manual, summarizes the control structure forms and illustrates both the programming format and the flowchart equivalent for each form.

While the structural statement forms of Fig. 3 seem to be rather complete, we frequently encounter a deficiency in the looping structures—the inability to escape from a loop at a point other than its beginning or end. In general, this situation can be resolved by employing a combination of looping and skip-action statements; however, the requirement for such a structure arose frequently enough that a more direct approach was indicated and we added the LOOP/EXIT structure of Fig. 4 to the FLECS language. Note that this structure is not inconsistent with structured programming philosphy, since the next statement to be executed upon exit from the loop is the statement that follows the loop structure.

What do FLECS users say?

Said one engineer, "Not only can I write and debug programs faster in FLECS, but six or eight months later I can still understand how they work."

A programmer said that he "had written special-purpose-language compilers in FLECS that he would never have attempted in FORTRAN." Since there was no other high-level language on the minicomputer he was using, he pointed out that what once had been a hybrid system requiring support from a large mainframe was now completely self-contained, with attendant reductions in turn-around time and training time for users of the system.

Another programmer said, ". . . I think FLECS is hard to beat. . . . The benefits include . . . programming ease, both initial programming and modifications, top-down structure ability, much more readable programs, English-language use for routine names and looping, and elimination of contorted branching." She has used FLECS in data analysis, scheduling, and moving data to and from different computer systems and databases.

What other features does FLECS provide?

Besides clarifying and enhancing the concepts of structured statements, FLECS also provides the programmer with a powerful procedural capability.

A *procedure* is nothing more than a group of statements, simple or structured, which the programmer wishes to have executed as a unit. A procedure is similar to a subroutine or function in that it may be invoked at more than one point in the program. However, it is unlike a subroutine or function subprogram in that a procedure is defined as part of the calling program. As a result, a procedure neither requires nor allows communication via passed parameter lists, but rather must test and modify program variables directly.

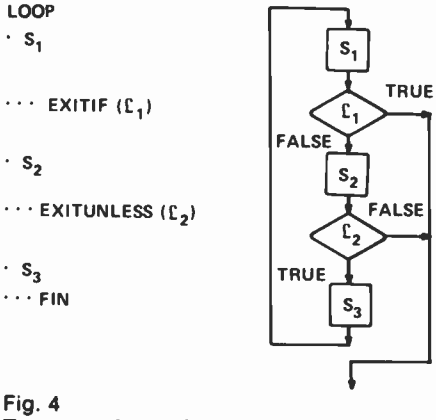


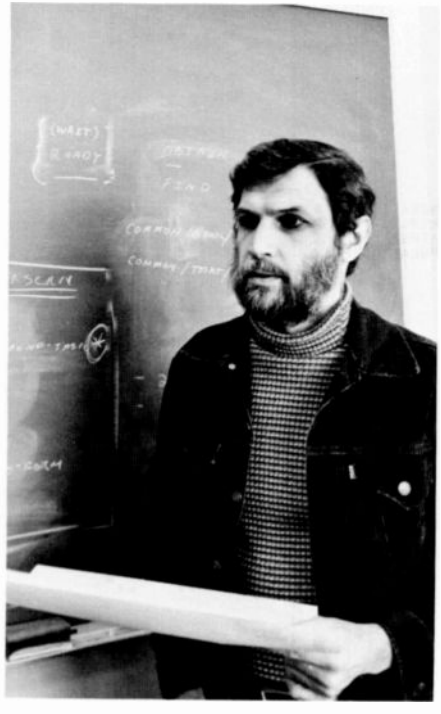
Fig. 4
To escape from a loop at a point other than its beginning or end, use this LOOP EXIT structured statement. This statement is not absolutely necessary, but avoids indirect solutions when an escape is needed.

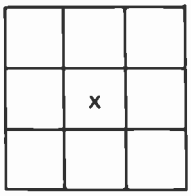
FLECS works well in the "top-down" programming approach.

With respect to structured programming, the FLECS procedural capability has a real advantage because a procedure may be invoked at a point in the program prior to its definition and the procedure's name may be as long as sixty-three characters. To attach proper significance to these points, observe first that the liberal naming convention permits the programmer to choose a name that is somewhat descriptive of the task which the procedure is to perform, and second that the decision to perform a task can be made without immediate regard for the details of how the task is to be carried out. This facility leads one rather naturally to the so-called *top-down* approach to software development, in which one attempts to decompose a problem into a small number of relatively independent steps. The process is then repeated for each of the initial steps until the complete details of the solution have been specified. While it is rare that a single attempt at this process yields a workable solution, it is generally far more successful than the so-called *bottom-up* approach, in which one attempts to specify the handling of the bottom-level details first and then collect those processes into successively larger units until the program is complete.

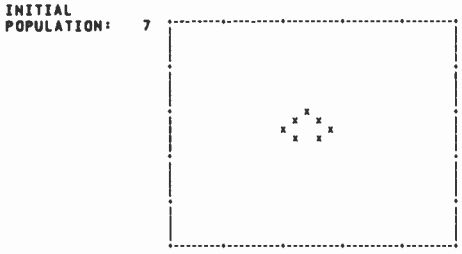
Tom Stiller, a Fellow of RCA Laboratories, has worked with the non-numeric aspects of computing for the last 14 years—work ranging from the development of function-evaluation subroutines and general-service programs up to multiprogramming operating systems and simulation of new computers prior to their manufacture.

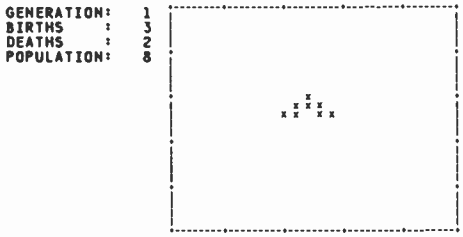
Contact him at:
Systems Research Laboratory
RCA Laboratories
Princeton, N.J.
Ext. 3181

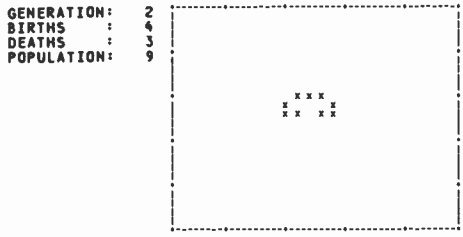


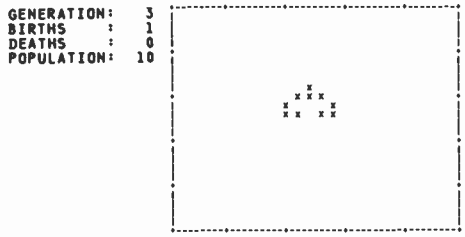


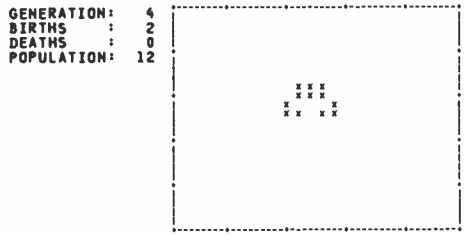
← Fig. 5
In the game of LIFE, each cell's "life" and "death" depend on its surrounding neighbors. A cell with the company of at least two but no more than three cells will survive; otherwise it will die of loneliness or overcrowding. An unoccupied cell can become populated only if it has exactly three neighbors. The progression of LIFE shown here was produced by the initial conditions given in the FLECS program listed in Fig. 6.

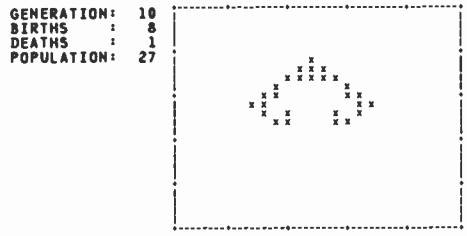












How about some examples of this structured technique?

To give some substance to the concepts we have been discussing, let us examine a program to play John Horton Conway's game of LIFE.[3] The "playing surface" consists of a rectangular array of cells. Each cell, with the exception of those on the border, has eight neighboring cells, as shown in Fig. 5. The game begins by specifying a configuration of "populated" cells. Each generation is formed from the previous one according to the following rules:

1) If a cell is occupied, its occupant will survive if it has the company of at least two but no more than three neighbors; otherwise it dies of loneliness or overcrowding.
2) If a cell is unoccupied, it will become populated if it has exactly three neighbors; otherwise it remains unoccupied.

If we augment these rules with some conventions regarding board size, and a method of specifying initial population and duration of game, we have enough information to specify a program to play the game. To be explicit, let us make the following assumptions:

1) The game is played on a 24 x 24-cell board.
2) The duration of the game is specified by typing the number of generations to be traced.
3) The program terminates when the end of the input dataset is sensed.
4) The initial population is specified by typing the row numbers and column numbers of the occupied cells.
5) The initial population list is terminated by entering row and column coordinates of zero.
6) A row or column number outside the range 1-24 (other than 0,0) results in an error message.

Fig. 6 is a reproduction of the FLECS indented source listing of an implementation of the above specifications. This particular implementation is a result of one pass through the top-down structured approach outlined above. While the program is neither particularly efficient in its use of working storage nor elegant in execution detail, it was designed and written, essentially as shown, from the top down; it did execute as expected the first time; the structure of the program is clearly visible; and the impact of a proposed change, say to the COUNT-NEIGHBORS procedure, can be readily predicted.

What does all this mean to me?

In the two years that FLECS has been in use within RCA Laboratories, we have seen both programmers and engineers produce more reliable software in less time than had been required with standard FORTRAN programming techniques. (See the box on the previous page.) In addition, as those programmers and engineers became more familiar with structured programming techniques, their programs began to exhibit a clear-cut distinction between logic design and implementation detail. This is certainly an advantage for the engineer who must modify an existing program, the engineer who must get a new application running correctly as quickly as possible, and for RCA, as the productivity of its engineers is increased.

References

1. McGowan, Clement L., and Kelley, John R.; *Top-down Structured Programming Techniques*, Petrocelli/Charter, New York (1975).
2. Beyer, Terry; *FLECS User's Manual*, Available from Customer Services, RCA TACS, Cherry Hill, N.J.
3. Gardener, Martin; "Mathematical Games," *Scientific American* (Oct 1970).

Reprint RE-23-6-17
Final manuscript received March 9, 1978.

Fig. 6 →
Indented source listing of FLECS "LIFE" program shows some of the advantages of using FLECS. The lack of programmer-supplied statement numbers and "continue" statements is immediately obvious. Note the English-language names for loops, the easily discernible overall structure (at top) that comes from top-down programming, indented structure showing nested loops, and cross-reference table.

```
      ----------------------------------------

00001       INTEGER ROW,COLUMN,MOVE,MOVES,DISPLY(24),X,BLANK
00002      1,   BORN,DIED,POP
00003       LOGICAL*1 BOARD(24,24),BIRTH(24,24),DEATH(24,24),EOF,ERROR
00004       DATA X /'X'/, BLANK /' '/
00005 C
00006 C     ARITHMETIC STATEMENT FUNCTION TO SIMPLIFY ROW/COLUMN TEST
00007 C
00008       LOGICAL VALID
00009       VALID(I)=I.GT.0.AND.I.LT.25
00010 C
00011 C     MAIN PROGRAM
00012 C
00013       LOOP
00014       .  SET-UP-BOARD
00015       .  GET-GAME-REQUEST
00016       ...EXITIF(EOF)
00017       .  PLAY-GAME
00018       ...FIN
00019       CALL EXIT
00020 C
00021 C     FORMAT STATEMENTS
00022 C
00023 1000  FORMAT(T10,'LOCATION NOT ON BOARD',2I5)
00024 1010  FORMAT(T30,' INCOMPLETE GAME DEFINITION.')
00025 1020  FORMAT(T30,' +---------+',3('---------+'),'---------+')
00026 1030  FORMAT(T30,' +',23A2,A1,'+')
00027 1040  FORMAT(T30,' |',23A2,A1,'|')
00028 1050  FORMAT('1',///T40,'INITIAL POPULATION:',I4,////////)
00029 1060  FORMAT('1',///T45,'GENERATION:',I4,/
00030      1,             T45,'BIRTHS     :',I4,/
00031      2,             T45,'DEATHS     :',I4,/
00032      3,             T45,'POPULATION:',I4,////)
00033 1070  FORMAT('1                  ALL DIED OUT IN GENERATION:'I3)

      ----------------------------------------

00034       TO SET-UP-BOARD
00035       .  DO(COLUMN=1,24)
00036       .  .  DO(ROW=1,24)
00037       .  .  .  BOARD(ROW,COLUMN)=.FALSE.
00038       .  .  .  BIRTH(ROW,COLUMN)=.FALSE.
00039       .  .  .  DEATH(ROW,COLUMN)=.FALSE.
00040       .  .  ...FIN
00041       .  ...FIN
00042       ...FIN

      ----------------------------------------

00043       TO GET-GAME-REQUEST
00044       .  ERROR=.FALSE.
00045       .  EOF=.TRUE.
00046 C     .
00047 C     .  AN END-OF-FILE CONDITION WILL
00048 C     .  LEAVE VARIABLE "EOF" SET .TRUE.
00049 C     .
00050       .  READ(5,*,END=10) MOVES
00051       .  POP=0
00052       .  LOOP
00053       .  .  ERROR=.TRUE.
00054 C     .  .
00055 C     .  .  AN END-OF-FILE CONDITION WILL
00056 C     .  .  LEAVE VARIABLE "ERROR" SET .TRUE.
00057 C     .  .
00058       .  .  READ(5,*,END=10) ROW,COLUMN
00059       .  .  ERROR=.FALSE.
00060       .  ...EXITIF(ROW.EQ.0.AND.COLUMN.EQ.0)
00061       .  .  WHEN(VALID(ROW).AND.VALID(COLUMN))
00062       .  .  .  BOARD(ROW,COLUMN)=.TRUE.
00063       .  .  .  POP=POP+1
00064       .  .  ...FIN
00065       .  .  ELSE WRITE(6,1000) ROW,COLUMN
00066       .  ...FIN
00067       .  EOF=.FALSE.
00068 10    .  CONTINUE
00069       .  IF(ERROR) WRITE(6,1010)
00070       ...FIN

      ----------------------------------------

00071       TO PLAY-GAME
00072       .  MOVE=0
00073       .  DISPLAY-BOARD
00074       .  WHILE(MOVE.LT.MOVES)
00075       .  .  MOVE=MOVE+1
00076       .  .  COMPUTE-NEXT-GENERATION
00077       .  .  POPULATE-NEXT-GENERATION
00078       .  .  WHEN(POP.GT.0) DISPLAY-BOARD
00079       .  .  ELSE
00080       .  .  .  WRITE(6,1070) MOVE
00081       .  .  .  MOVE=MOVES
00082       .  .  ...FIN
00083       .  ...FIN
00084       ...FIN

      ----------------------------------------

00085       TO COMPUTE-NEXT-GENERATION
00086       .  BORN=0
00087       .  DIED=0
00088       .  DO(COLUMN=1,24)
00089       .  .  DO(ROW=1,24)
00090       .  .  .  COUNT-NEIGHBORS
00091       .  .  .  WHEN(BOARD(ROW,COLUMN))
00092       .  .  .  .  DEATH(ROW,COLUMN)=NBORS.LT.2.OR.NBORS.GT.3
00093       .  .  .  ...FIN
00094       .  .  .  ELSE BIRTH(ROW,COLUMN)=NBORS.EQ.3
00095       .  .  ...FIN
00096       .  ...FIN
00097       ...FIN

      ----------------------------------------

00098       TO POPULATE-NEXT-GENERATION
00099       .  DO(COLUMN=1,24)
00100       .  .  DO(ROW=1,24)
00101       .  .  .  WHEN(BOARD(ROW,COLUMN))
00102       .  .  .  .  IF(DEATH(ROW,COLUMN))
00103       .  .  .  .  .  DEATH(ROW,COLUMN)=.FALSE.
00104       .  .  .  .  .  DIED=DIED+1
00105       .  .  .  .  .  POP=POP-1
00106       .  .  .  .  .  BOARD(ROW,COLUMN)=.FALSE.
00107       .  .  .  .  ...FIN
00108       .  .  .  ...FIN
00109       .  .  .  ELSE
00110       .  .  .  .  IF(BIRTH(ROW,COLUMN))
00111       .  .  .  .  .  BIRTH(ROW,COLUMN)=.FALSE.
00112       .  .  .  .  .  BORN=BORN+1
00113       .  .  .  .  .  POP=POP+1
00114       .  .  .  .  .  BOARD(ROW,COLUMN)=.TRUE.
00115       .  .  .  .  ...FIN
00116       .  .  .  ...FIN
00117       .  .  ...FIN
00118       .  ...FIN
00119       ...FIN

      ----------------------------------------

00120       TO COUNT-NEIGHBORS
00121 C     .
00122 C     .  TEST FOR TOP OR BOTTOM ROW
00123 C     .
00124       .  SELECT (ROW)
00125       .  .  (1)
00126       .  .  .  IS=1
00127       .  .  .  IE=2
00128       .  .  ...FIN
00129       .  .  (24)
00130       .  .  .  IS=23
00131       .  .  .  IE=24
00132       .  .  ...FIN
00133       .  .  (OTHERWISE)
00134       .  .  .  IS=ROW-1
00135       .  .  .  IE=IS+2
00136       .  .  ...FIN
00137       .  ...FIN
00138 C     .
00139 C     .  TEST FOR COLUMN ON LEFT OR RIGHT EDGE
00140 C     .
00141       .  SELECT (COLUMN)
00142       .  .  (1)
00143       .  .  .  JS=1
00144       .  .  .  JE=2
00145       .  .  ...FIN
00146       .  .  (24)
00147       .  .  .  JS=23
00148       .  .  .  JE=24
00149       .  .  ...FIN
00150       .  .  (OTHERWISE)
00151       .  .  .  JS=COLUMN-1
00152       .  .  .  JE=JS+2
00153       .  .  ...FIN
00154       .  ...FIN
00155 C     .
00156 C     .  DO NOT COUNT OCCUPANT AS NEIGHBOR
00157 C     .
00158       .  WHEN(BOARD(ROW,COLUMN)) NBORS=-1
00159       .  ELSE NBORS=0
00160       .  DO(J=JS,JE)
00161       .  .  DO(I=IS,IE)
00162       .  .  .  IF(BOARD(I,J)) NBORS=NBORS+1
00163       .  .  ...FIN
00164       .  ...FIN
00165       ...FIN

      ----------------------------------------

00166       TO DISPLAY-BOARD
00167       .  WHEN(MOVE.EQ.0) WRITE(6,1050) POP
00168       .  ELSE WRITE(6,1060) MOVE,BORN,DIED,POP
00169       .  WRITE(6,1020)
00170       .  DO(ROW=1,24)
00171       .  .  DO(COLUMN=1,24)
00172       .  .  .  WHEN(BOARD(ROW,COLUMN)) DISPLY(COLUMN)=X
00173       .  .  .  ELSE DISPLY(COLUMN)=BLANK
00174       .  .  ...FIN
00175       .  .  WHEN(MOD(ROW,5).EQ.0) WRITE(6,1030) DISPLY
00176       .  .  ELSE WRITE(6,1040) DISPLY
00177       .  ...FIN
00178       .  WRITE(6,1020)
00179       ...FIN
00180       END

      ----------------------------------------

           PROCEDURE CROSS-REFERENCE TABLE

      00085 COMPUTE-NEXT-GENERATION
            00076

      00120 COUNT-NEIGHBORS
            00090

      00166 DISPLAY-BOARD
            00073  00078

      00043 GET-GAME-REQUEST
            00015

      00071 PLAY-GAME
            00017

      00098 POPULATE-NEXT-GENERATION
            00077

      00034 SET-UP-BOARD
            00014


      (FLECS VERSION 22.44)

      ----------------------------------------
```

program structure at a glance

very few statement numbers needed

English-language names for loops

indents show nested loops

cross-reference table

Type-by-mouth system aids handicapped student

P. B. Pierson

How do you simultaneously read and write when you've lost the use of your limbs? This type-by-mouth device using modern technology was developed to aid a handicapped person in just such a predicament.

Ed. Note: The hobby article feature in the *RCA Engineer* reports on ways that members of RCA's technical community apply their professional talents and training to off-the-job pursuits. The articles published thus far have explored projects which hobbyists have undertaken mainly for the pleasure involved in developing an idea or gadget.

In this issue, we look at a project developed for a different reason—a need to help a courageous quadriplegic tackle the rigors of a college curriculum. Preparing the required assignments at a university is enough of a task for most people, but consider the added burden of not being able to translate your thoughts through the physical act of writing. The author, Paul Pierson, through personal dedication and the skills of his profession, plus the generosity of others, answered that need and in the process gained new insight into the relationship of technology and human communications.

—F.J.S.

Andrew (Drew) Batavia, my wife's cousin, was left paralyzed from the neck down in 1973 as a result of an auto accident while a counselor at summer camp. He completed high school through the use of a special motorized wheelchair, which he controlled by puffing and sucking on a tube. He typed his homework assignments with a mouth-stick clenched between his teeth. This required that the typewriter be positioned directly in front of him, leaving no room for a book stand. He could not read and write at the same time, presenting a serious impediment to academic pursuits.

An idea for a solution

I hoped to help him with this problem by building a type-by-mouth device—a computer printer interface which could interpret a puff-suck code and convert that information into ASCII code for printing. The air tube could be located near his mouth, allowing him to both type and turn pages on a bookstand with a mouth stick. This capability to read and write simultaneously would be necessary in college, which he had the courage to face within just two years of the accident.

Help in funding the project

Devices which performed such an interface were very expensive at the time, as were electrically driven printers. I planned to provide my design, fabrication, test, and integration time but lacked the personal resources to buy a suitable printer or the materials for the interface.

After requests to many companies for a donation of a printing terminal, I got a positive response. General Electric at Waynesboro, Va., agreed to donate a Term Net 300 printer, plus the use of its time-share computer system for Drew to store and retrieve his college notes. The printer came with a cassette tape unit and was worth $5,500. I was impressed with the generosity and helpfulness of all those at General Electric. That helpfulness continued long after presentation of the gift. Material costs for my interface unit were paid by my wife's parents. All I had to provide was time, knowhow, and dedication.

The first design

The functional specification for my interface unit (which I called PUFF-A-TYPE) went through two revisions to resolve Drew's operational difficulties and incorporate his recommendations.

I set up an alphabet matrix as shown in Fig. 1 and identified each character with a light emitting diode (LED). Only one LED would be lit at any time and a sequence of puffs and sucks would select the character to be typed and finally activate the printer. The Puff-Suck Sequence worked as follows:

First Puff: Light moves right, by internal clock, for duration of Puff.

First Suck: Light moves up, by internal clock, for duration of Suck.

Second Puff: If spot at shift position, shift (and hold) and

Reprint RE-23-6-24
Final manuscript received April 21, 1978.

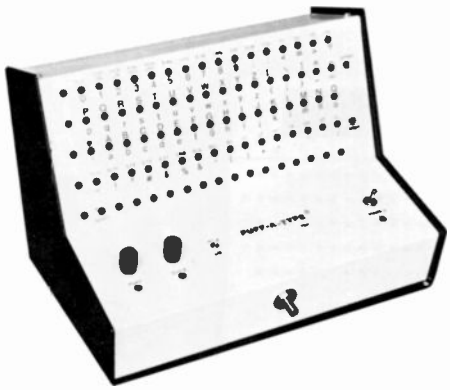


Fig. 1
Layout of the first model of the PUFF-A-TYPE interface unit. Air tube connector is at the bottom center of the panel. Puff/suck threshold adjustment controls are located to the left of the HI/LO switch to permit a change in spot speed. Spot movement depended on duration of puff or suck action, requiring strenuous respiratory effort.

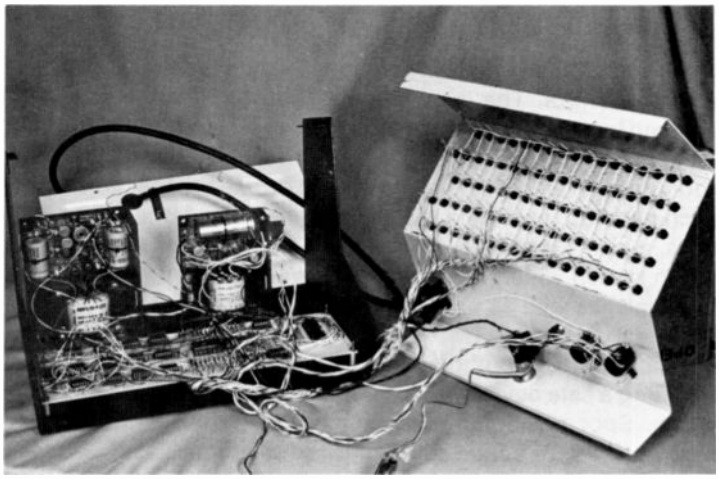
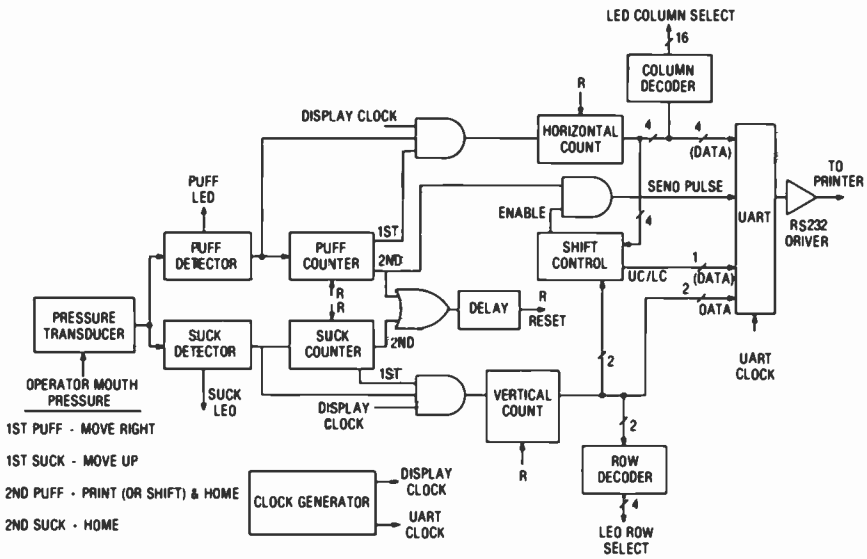


Fig. 2 ↗
Internal view of the PUFF-A-TYPE unit. At left is the logic circuitry for decoding puff/suck commands. At right is the front panel with its string of LEDs.

Fig. 3 →
Logic-circuit elements of the basic PUFF-A-TYPE design. Horizontal and vertical counters determine spot position and present ASCII code to the UART for transmission to the printer. These counters are controlled by decoding puff or suck sequences detected by the pressure transducer. The second design replaced the puff/suck detectors and counters with up, down, right, and left switches and added an 80-character memory with 32-character display between the horizontal/vertical counters and the UART. Detected puff and suck commands loaded the selected characters of the desired case into memory. Special codes were provided for editing and finally for transfer from memory to printer. In the third design, code-converting programmable read-only memories were added between the position counters and the display memory, providing a more efficient panel layout.



then Home Spot (to lower left). Otherwise print character of spot position, then Home Spot. Reset Puff/Suck count.

Second Suck: Home Spot, Reset Puff/Suck count.

Fig. 2 shows the innards of the first design and Fig. 3 is block diagram of the designed hardware.

The first test

After working Saturdays and several evenings per week for approximately six months, the unit was ready for Drew. The printer had been presented by General Electric two months before this big event, and the whole family gathered with hopeful anticipation. He tried the unit, complimented me, and finally told me his real feelings: "I can't use it!" Several significant problems had to be resolved for the system to be of any value.

First, the line of text being typed by the printer is not visible to a seated operator. Drew would have to rely entirely on memory for words and letters already typed as he composed his text: an extremely difficult task! Second, it was difficult to successfully "land" on the desired character with the moving spot, and it was frustrating and time consuming to have to reset all the way to the home position after an error. Third, the air pressure required to activate the puff/suck detector was exhausting to his weak respiratory system.

The second design

To solve the first problem, we considered the use of mirrors, placing the unit directly below Drew, or the electronic display of characters printed. The latter offered the potential of soft-copy editing, prior to final hard-copy printing. The problem of cost then recurred. Alphanumeric displays which were multi-character and readable at a distance were very expensive.

After my success with General Electric, I decided again to request equipment donations.

Burroughs Corporation, Plainfield, N.J., agreed to donate a 32-character SELF-SCAN® display panel with upper and lower case alphabetics, full ASCII compatible character set, and memory. Similarly Endicott Coil Co., Inc., Binghamton, N.Y., provided the high-voltage power supply required to drive the SELF-SCAN panel. My project was destined for success through the generosity of a concerned electronics industry!

My second problem, simplifying the character selection process, was solved by redesigning the spot-positioning circuits to provide up, down, right, left, and diagonal spot-position control. The cumbersome "Puff-suck" code was replaced by a mouth-operated joystick, provided by my car-pooling co-worker, James Tyson. Jim's hobby is metal-work (he is an electrical engineer), and he has a respectable machine shop, complete with milling machine, lathe, drill-press, etc., in his garage. He designed and built an effective mouth switch which detected up, down, right, left, and all diagonals, as well as puff and suck commands.

A standard pipe mouthpiece was fitted to the mechanism, providing a safe device which could be gripped easily in the mouth. Spot positioning was achieved by deflecting the joystick in the direction of the desired character.

Once located, a puff entered the upper-case character, and a suck entered the lower-case character. The spot would then be moved to the next desired character. Characters entered were displayed on the SELF-SCAN panel before transmission to the printer. I added an 80-character memory which could be stepped forward or backward to permit editing of a full line before printing. Characters entered in error were, therefore, not catastrophic to an already handicapped typing effort.

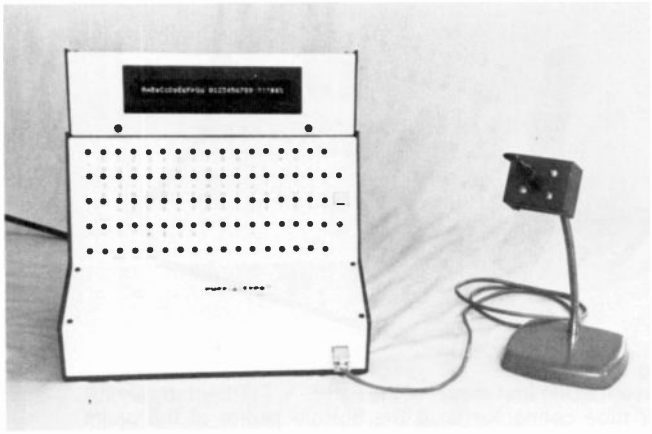


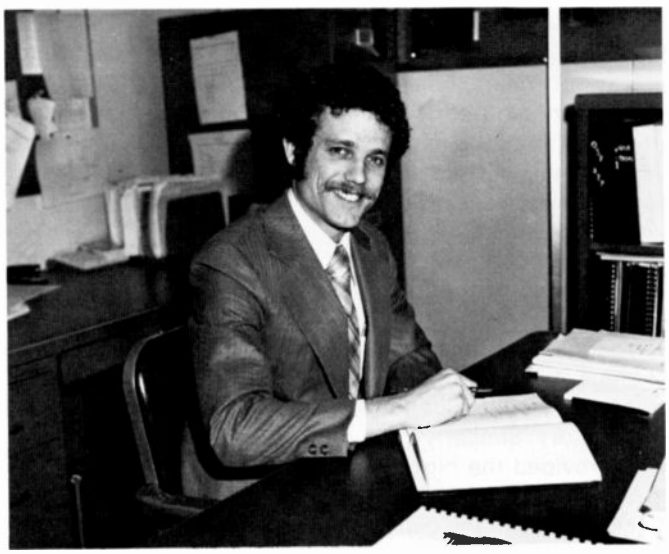
Fig. 4
The second design of the PUFF-A-TYPE unit incorporated a 32-character display panel to provide an editing capability. Mouth-operated joystick gave a greater degree of flexibility in operation.

The third problem, respiratory exhaustion, was also solved by the switch design. Pressure sensing was performed through a sensitive diaphragm which interrupted LED light beams.

Fig. 4 shows the second generation system and Fig. 5 is a closeup view of the mouth-operated joystick. These improvements seemed to work well and, at first, pleased Drew. But after a few months, the equipment fell into disuse. He hesitated several months before telling me of his new frustrations, for fear of offending me. Even though the device permitted simultaneous reading and writing, he found typing was actually slower than by pecking with the mouth-stick. So for nearly eight months, the system sat unused.

Paul Pierson is responsible for the development of video tape recorders and laser beam recorders which generate high-quality images on film using laser beams. The images are derived from outputs of sensors on aircraft and satellites, such as the RF-4 reconnaissance aircraft and the Landsat satellite. His experience in data processing, electro-mechanical systems, and man/machine interfacing equipped him well for undertaking the unusual project described in this article.

Contact him at:
Recording Systems
Government Communications Systems
Camden, N.J.
Ext. 2830



Jim Tyson, a co-worker of the author, designed and built the mouth-operated switch for the PUFF-A-TYPE in his home machine shop.

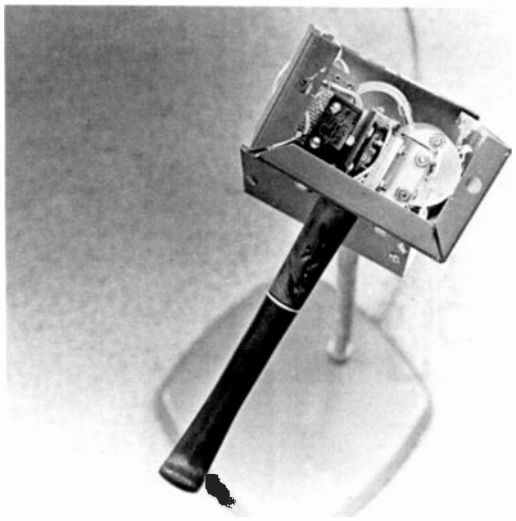


Fig. 5
Mouth-operated joystick detects motion in all directions as well as puff/suck commands. Three basic components make up the unit: pipe mouthpiece, puff/detector (circular component at right inside the unit), and an up, down, left, right, diagonal switch matrix (at left of detector).

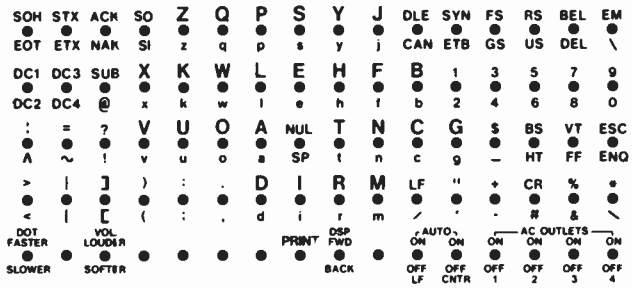


Fig. 6
Character layout arrived at for the third design. The letters having a higher probability of occurrence are located around the Home position (NUL/SP). Lower row of LEDs indicates the area of operational control where the operator controls spot speed, print command editing, etc., by puffs and sucks. The unit has the capability of computer interface through complete ASCII character subset (note extra characters at upper left and right).

The third design

To speed up the typing process, Drew and I discussed how the display character matrix could be rearranged. For greater efficiency, the Home position should be in the center of the matrix. Character positions should be determined by their probability of occurrence in English text (this information is available in some dictionaries). Characters and operations most likely to occur (e.g., T, E, and space) should be clustered near the center of the panel, surrounded by characters of decreasingly lower probability of occurence. After entry of each character, the spot should return to the home position at the center of the panel.

The character layout selected is shown in Fig. 6. Home position is space (SP) or erase (NUL). The only changes required were the display panel artwork and the addition of code-converting programmable read only memories. This layout offers several advantages. First, spot motion time is minimized. Second, the same physical motions are repeated identically every time a given character is repeated. This aids in memorization, and reduces thought time required to begin moving toward the next character. Drew enjoys the improvement and presently types with relative ease, at about 13 words per minute.

Reflections

One frustrating aspect of a home electrical engineering development project is that technology moves faster than a project can be completed. During the course of my design, microcomputer technology grew from interesting concept to wide-spread application. My entire system of three circuit boards could now be replaced by a small microcomputer card, and the capabilities of such a card would far outrun my design. I was often tempted to introduce new circuit technologies, but gave in to the least-time-to-implement approach of using standard MSI circuits for modifications.

Another interesting problem I found was the hesitancy of my end user (Drew) to discuss problems he found with the unit. He was not familiar with the iterative process most development projects require. This problem was increased somewhat, I guess, by the fact that I was not receiving any monetary compensation for my efforts. Criticism was considered by the family to be equivalent to complaints about a purchased gift. They did not, at first, understand that I wanted to hear Drew's complaints and his suggestions for improvements. After removal of this block to communications, progress became much easier.

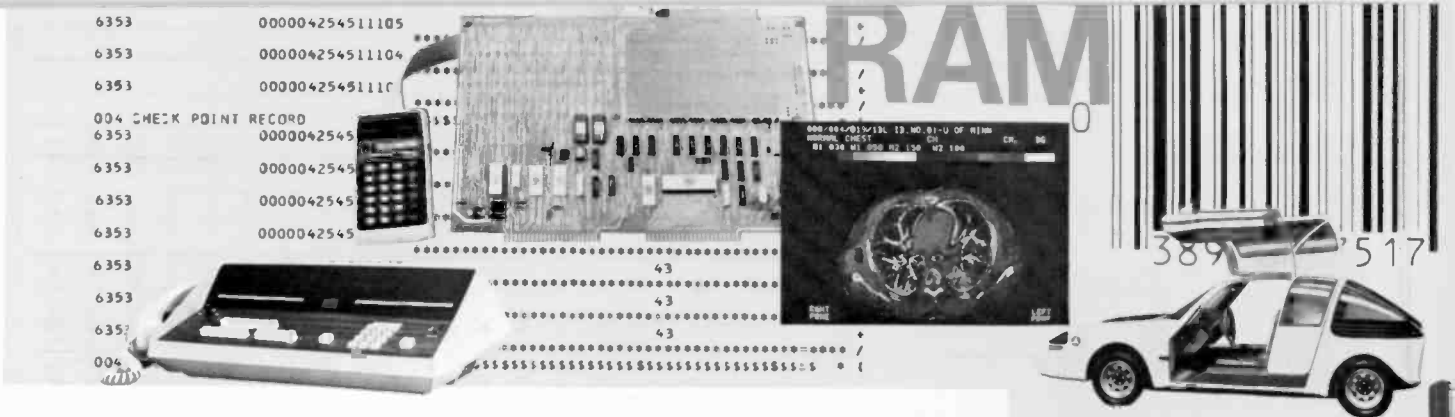
Future plans

Before designing an improved system using present technology capabilities, I am waiting for several development thresholds in word processing and voice recognition technologies. I would like to build a hands-off, voice-activated typing system with full document editing capability and multiple document storage. Various candidate components exist today, but I would like to see more sophisticated software packages, and higher-capacity, lower-cost storage media before I begin this advanced system project.

Despite communication problems, long completion schedules, and losing ground to advancing technology, I found this project one of the most rewarding endeavors of my personal and professional career.

Acknowledgments

Earlier I mentioned the generosity of several corporations in contributing hardware and other support to the project. In particular, I would like to thank Marjorie G. Grimes, Manager, Employee Relations, General Electric Co. and Richard L. Singer, Manager, Advertising and Sales Promotion, Burroughs Corporation, for their help in obtaining the needed equipment.



THE PERVASIVE COMPUTER

The computer today is as basic to our way of life as automobiles, televisions, banking, or medicine. In fact, these basic products and services are better today because of the computer.

P. M. Russo|C. C. Wang

Since ENIAC, the first electronic digital computer in 1946, computer technology has evolved at a phenomenal rate. ENIAC was a roomfull of vacuum tubes and cost hundreds of thousands of dollars. It could only operate for a few hours a day, at millisecond speed, with hardwired programming. Today, only three decades later, an LSI (large scale integration) microcomputer on a square of semiconductor material, a fraction of an inch on each side, costs only a few dollars and can operate continuously from a battery at microsecond speed. Additionally, this same semiconductor chip may contain several kilobytes of stored-program memory.

Since von Neumann introduced the concept of a modern stored-program computer in 1945,[1] computer technology has evolved through three generations; see Fig. 1. First generation machines were based on vacuum-tube logic while the emerging memory technologies—mercury delay lines, Williams tubes, magnetic drums, and magnetic cores—were under development. For second generation systems, magnetic core finally began to dominate the storage-medium market whereas the transistor displaced the vacuum tube in the central processing unit. It was during this period that magnetic tapes and disks were introduced as competing technologies for secondary storage. Eventually, rapidly advancing disk technology provided the technical base to spur the development of third-generation systems. In these systems, integrated circuits were used as cost-effective substitutes for the discrete transistor in the implementation of control logic.

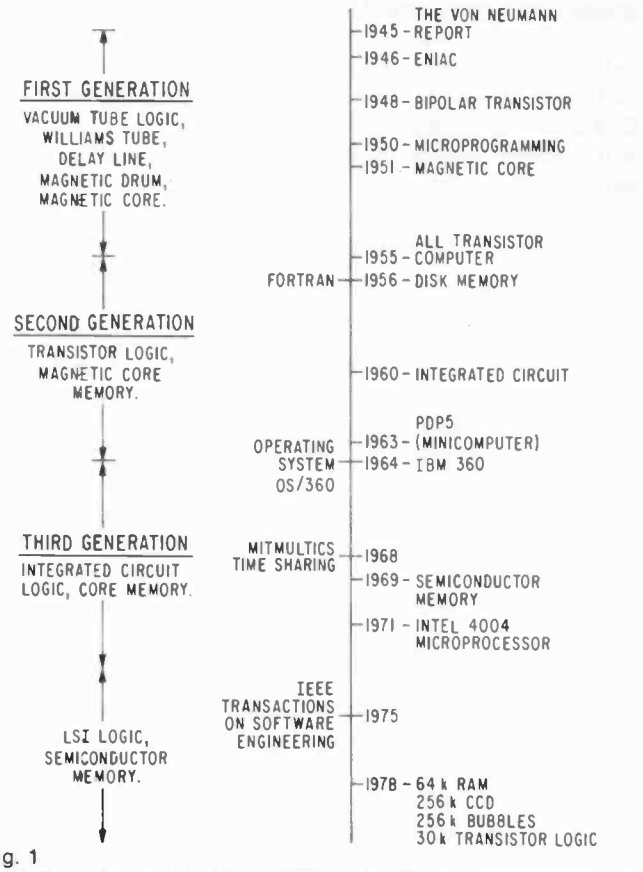


Fig. 1
Evolution of computer technology. Three generations of phenomenal growth starting with vonNeumann's concept of stored program control and presently pushing the state of the art in large scale integration (LSI) and semiconductor memories.

While third generation mainframes were reaching the marketplace, the cost-reduced minicomputer emerged as an effective alternative for some purposes. The subsequent introduction of semiconductor memory (1969), which has rapidly displaced core memory, and the explosive growth in LSI technology (Fig. 2) have provided the impetus for this next generation of minicomputer and microcomputer systems. The ongoing microprocessor revolution represents just one aspect of this new era in computer technology.

The concept of a stored program, and hence software, is fundamental to the evolution of computer technology. Software technology, itself, has undergone several evolutionary stages of development. In the first stage, assembler and high level language compilers, for languages such as FORTRAN and COBOL, were developed to facilitate the preparation of software programs. As computers became both more complex and more costly, operating system software was developed to manage and improve the use of system resources. Time sharing systems grew out of these developments to provide computing power to any location having telephone service, further extending the commercial and industrial utilization of computers.

The phenomenal growth in computer applications has stimulated the evolution of software technology. Complex data base management systems are becoming more and more important in centralizing and organizing the voluminous data generated by the workings of a modern society. As software programs become more complex, the development and management of these software systems can no longer be viewed as routine. For these complex systems, the development costs associated with software have already overshadowed the cost of the hardware (Fig. 3). Structured programming and software engineering are

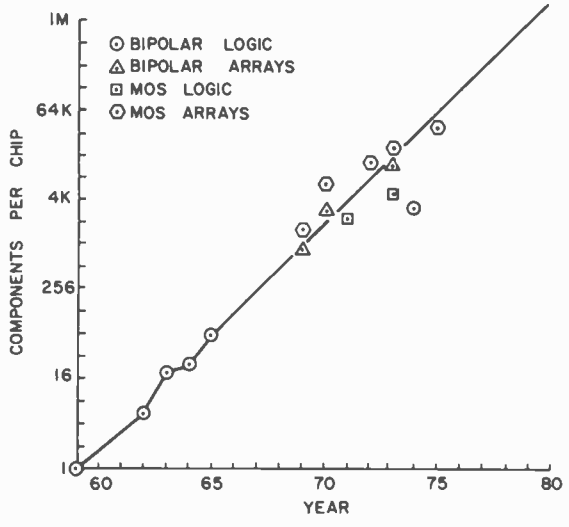


Fig. 2
Evolution of semiconductor technology has paced that of computers. In particular, LSI growth of the 70s has pushed development of modern minicomputers and microcomputers.

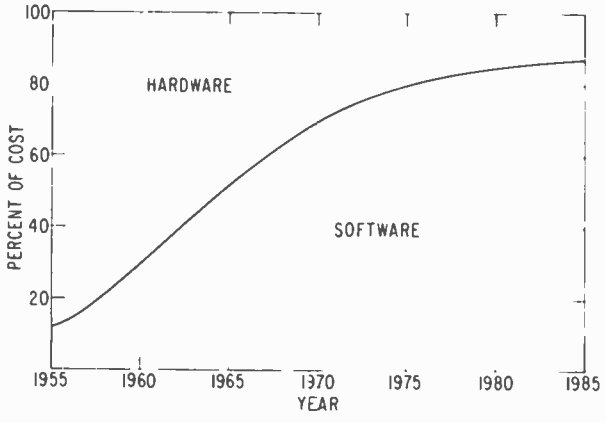


Fig. 3
Software costs presently represent close to 80% of the total system development costs. This trend is expected to continue, with software development costs approaching 90%.

new emerging disciplines that are providing the necessary tools to address these problems.

With the continuing evolution of both hardware and software technologies, the development of new computer applications will undoubtedly become evermore widespread. The total spectrum of computer applications can never be covered exhaustively. The following sections present overviews of mainframe computer, minicomputer, and microprocessor applications. These three natural applications groupings will, hopefully, bring into focus both the pervasiveness of computers in modern society, and the fundamental importance of software to these applications.

Mainframe computers

Large mainframe computers characterized by the IBM 370 and the CDC 7000 series remain ubiquitous. Over 50,000 IBM 370s have already been shipped. Over the two decades, from 1958 to 1978, the performance of large mainframe computers has improved by a factor of 250, Fig. 4; whereas the improvement in the performance/price ratio has improved by a factor of 200. The change in the total system price has been minimal. With system costs staying at the millions of dollars level, the applications of costly mainframe computers are still restricted to the traditional high performance or high throughput categories. Examples of these categories are overviewed below.

Scientific computations

Mainframe computers have been used extensively in the scientific community for computationally complex applications. In the physical sciences, computer simulations have proved to be powerful tools and have provided invaluable insight into many hitherto unexplained phenomena. In mathematics, computers are now established as effective alternatives to the classical approach to theorem proving; the proof that any map can be colored with a minimum of four colors is a good example. In other areas, meteorologists have been using the most powerful computers available to model, simulate, and predict the changes in our atmospheric environment. Economists have modeled the various elements in our economic system and employed digital computers to predict their interactions.

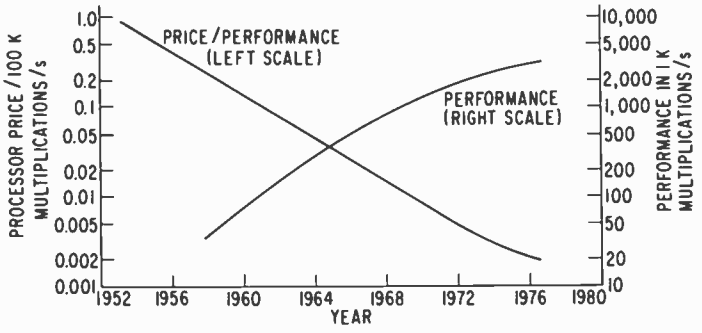


Fig. 4
More "bang/buck." Mainframe computer performance has improved by a factor of 250, while performance/price has improved by 200.

Computer-aided design

A natural application of mainframe computers is as an aid to engineering design. The most vivid example of this use is in the design of integrated circuits. On a tiny silicon chip, hundreds of thousands of elements have to be interconnected perfectly before the design can be produced. The conventional breadboard method is neither economic nor adequate to prototype the design. Computer simulation has been established as the most dependable tool for integrated circuit designers.[2,3,8]

Computers are indispensable in generating both complex artwork[4-7] and voluminous test patterns[8] in the manufacturing and testing of these devices. Besides integrated circuits, computer-aided design has also been used extensively in color television receiver design,[9] picture tube design, hybrid circuit design[10] and printed circuit board design.[11-12] This list is by no means complete.

Electrical engineers are not the only beneficiaries of computer aided design. Other disciplines of engineering have also used computers extensively. The digital-computer-based finite-element method represents a powerful general-purpose tool for structural design. Numerous computer programs are available for analyzing heat transfer and fluid mechanics systems. Buildings, automobiles, ships, aircraft and highways, just to mention a few, are all being designed with the help of computers.

Data processing

The invention of modern digital computers was motivated by the basic need for computation. However, during their evolution, computers have become instrumental in the processing of voluminous data generated by the business activities of modern society. Most businesses now have payroll, accounting, record-keeping, warehousing, billing and warranty of products routinely handled by computers.[13,14] Banks, insurance companies, and various government agencies have to depend on computers to keep track of the bank accounts, insurance policies, income tax, and social security accounts. Airlines have to depend on computers for flight information and reservations. Even the

U.S. Congress is in the process of computerizing the voluminous documents associated with legislative activities.

Besides routine business operations, computers have also proved to be valuable in marketing studies and business planning where they generate the management information needed for business decisions.[15–18]

Time-sharing systems

In recent years, time sharing has matured as an effective technology for distributing centralized computing power to a host of remote locations. A terminal connected through the telephone line to the central computer provides the user with the capability to interact with the system directly and instantly. People can access computing power, the centralized data base, and commonly used program libraries from their offices, or their homes, or anywhere a terminal and telephone are available. Improvements in the ready accessibility of computing power undoubtedly have made possible many new applications which were unrealizable before.

. . . no end in sight

Several years ago, the demise of large centralized computers was predicted, based on the ever-improving price-to-performance ratio of minicomputer systems. The large and increasing base of installed mainframes contradicts these predictions. In fact, the success of major new market entries, such as Ahmdahl and Itel, is predicated on continued growth in the use of large central computers.

Minicomputers

The introduction of minicomputers by Digital Equipment Corporation in 1963* signified a change in the evolution of computer development. Instead of stressing higher performance, minicomputers featured an order of magnitude reduction in system cost. This drastic cost reduction made computing power readily available for many new applications and spurred the distributed-processing revolution. Several broad categories of minicomputer applications are overviewed below.

Real-time control of manufacturing and testing

With minicomputers, real-time control of manufacturing and testing becomes economically feasible. Minicomputers are penetrating the industrial market at phenomenal rates. Automatic testing is commonplace.[20] Factory automation has become a reality.[21] Cigarettes, automobiles, color films, color television receivers, color picture tubes,[19] electronic assembly,[20] jet engine accessories,[20] aircraft parts,[20] semiconductor devices,[22,23] and endless varieties of other products have all been manufactured and tested under the control of digital computers.

Telecommunications

Telecommunications is becoming an increasingly large market for minicomputer systems. Expensive hardwired controllers are being replaced by less costly but more flexible mini-based systems to perform switching and multiplexing functions.[24–26] One such example is the RCA Telex system.[24] Minicomputers are becoming the essential elements in the implementation of complex data communication networks.

*PDP-5 with 1k of 12-bit words at $27,000. The popular PDP-8 with 4k of 12-bit words was introduced in 1965 at $18,000.

Data acquisition and analysis

Minicomputers are fundamental to laboratory automation applications where they collect and analyze laboratory experimental data (e.g., microwave measurements[27]). They are also used extensively in medical instrumentation. The amazing CAT (computer-assisted tomography) body scanner is one of the many imaginative applications of minicomputers in this area.[28]

Scientific computations and data processing

Cost advantages of minicomputer systems are attracting some traditional mainframe users, whose tasks can be partitioned, and who do not require large centralized data bases. The less demanding scientific and data processing applications can be satisfied easily with a minicomputer. The growth in these application areas has prompted the introduction of time sharing, high-level languages, and data-base management systems to the minicomputer world.

Besides their direct use in the manufacturing process, minicomputers are now used in the manufacturing plant to gather data for production scheduling, material inventory, and quality control.[19] Minis are also used in supermarkets and retail stores to collect transaction data to provide up-to-date business information.

Multiprocessor systems

A recent trend in the development of minicomputer systems has been to structure many dedicated minis into powerful distributed systems. In these structures, a mini is no longer viewed as a single stand-alone system, but rather as a building block in a complex system. This approach will certainly be used in many applications which have

previously been limited by the performance of minicomputers. The NBC switching central[29] and the building-management systems implemented both at the Disney World Complex and at the Wells Fargo Bank[30,31] are just a few such examples.

More units, lower cost

With annual worldwide shipments of minicomputers approaching 100,000 units and with ever more cost effective systems being introduced in an increasingly competitive market, the uses and applications of minicomputers are assured of continued and rapid growth.

0101100111010100 Microcomputers

The applications of microprocessors break down into three broad categories—industrial, commercial, and consumer.

By *industrial applications*, we mean those that employ microprocessors in the manufacturing process of the industry under consideration. These include process control, testing, data acquisition, numerical control, instrumentation, and robotics.

By *commercial applications*, we mean those used in providing new or improved services and/or ways of doing business. These include communications (data and telephony), point-of-sale systems, intelligent cash registers, gasoline pumps and weighing scales, and business uses (word processing, accounting, etc.).

By *consumer applications* we mean those that either add features or improve the cost/performance of existing consumer products (ignition control in automobiles, timing control in microwave ovens, etc.), or those that give rise to entirely new classes of consumer products (personal computers, programmable video games, etc.).

In the following three sections, we will briefly overview industrial, commercial, and consumer applications of microprocessors, and provide references to permit the interested reader to pursue the subjects in depth. One measure of the extent of the ongoing microprocessor application revolution is the February 1978 issue of the *IEEE Proceedings*, which was dedicated to that very subject.

Industrial applications

Microprocessors are having a major impact on industrial applications, including the areas of testing, control, instrumentation, data acquisition, numerical machine control, and even robotics. In all these areas, equipments offering expanded functions, better human interfaces, improved reliability, and lower cost are emerging.

Automatic test systems which can be used to automatically identify failures of components or subsystems are becoming increasingly important from both reliability and ease of maintenance points of view. Fig. 5 shows a typical test system where isolation has been provided between the subsystem under test and the signal sources and measurement devices. Note that this system is rather complete; however not every test system will contain all the indicated subsystems.

Automatic test systems have historically been used in large military applications and in the automatic testing of complex parts, where expensive test fixtures could be economically justified. The advent of microprocessors has opened many avenues for low cost dedicated testers which can now economically justify the automated and more complete testing of low cost/complexity subsystems, down to the component level.

Current economic factors are tending to accelerate the trend towards more fully automated testers.[32-35] Ever-increasing labor costs, corresponding decreases in the market price of hardware, demands for constantly improved product reliability and safety all dictate increased automation. Additionally, it is becoming increasingly desirable to catch failures early in the assembly process, since the value added (cost to troubleshoot and repair) goes up ex-

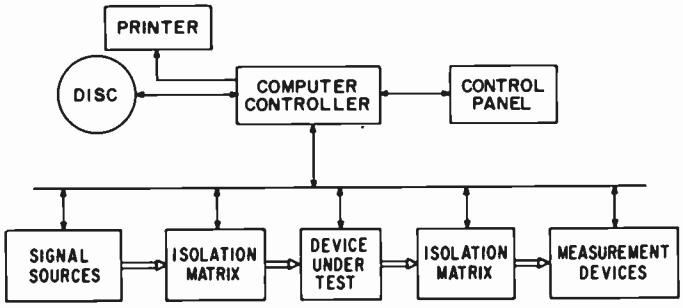


Fig. 5
Typical microprocessor-based test systems include most of the functions shown here.

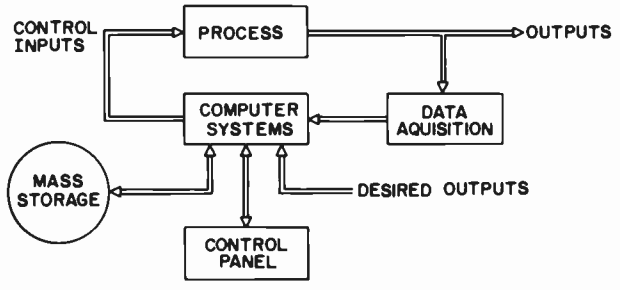


Fig. 6
Real-time computer-based control systems, in general, sample critical system or process outputs as a basis for developing system control inputs.

ponentially with the number of components in the subsystem under test.

The process control and data acquisition areas differ from testing in that they must operate in real time.[36-43] This implies that the microcomputer must have time responses rapid enough to accommodate the process control or data acquisition system under consideration. Whether we are talking about controlling an automotive ignition system or a complex chemical plant, the system must sample critical system outputs and generate suitable system inputs often enough to achieve the desired level of control. Fig. 6 presents a block diagram of general computer-based process control system.

A typical data acquisition system is shown in Fig. 7. Such a system can be stand-alone, can gather data for later analysis, or it may operate in real-time, in which case it is simply a subsystem of the general system described in Fig. 6. In any case, a microprocessor can be dedicated to data acquisition alone. A dedicated microprocessor can perform statistical analysis on the fly, format data for more efficient off-line processing, and perform periodic self-checking and auto-calibration. However, its main advantage may, in fact, be the flexibility resulting from distributed software control. Simple software changes can alter the sampling rate, specify new data formats, or alter the self-checking algorithms. This results in better human engineered outputs, modularity in hardware (improved maintenance) and slower system obsolescence.

The impact microprocessors are having on instrumentation is just beginning to be felt.[44,45,46] From simply adding new features to instruments (digital read outs, averaging, etc.) to auto-calibration, instruments will never be the same again.

A major factor in the increasing sophistication and flexibility of instruments has been the development of the IEEE Standard 488 Instrumentation Bus pioneered by Hewlett-Packard. This standard interface permits the simple interconnection of complex instruments under the supervision of single or multiple masters.

Computerized instrumentation, interconnection of equipment manufactured by different vendors, standardized control/data exchange protocols, all are now made simpler via this now accepted structure. Reference 47 is an excellent article detailing the IEEE Standard 488 Instrumentation Bus and its uses.

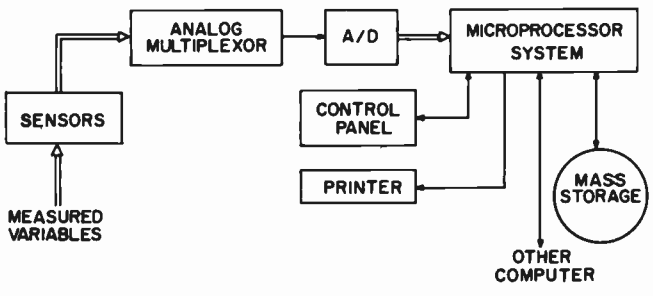


Fig. 7
Microprocessor-based data acquisition systems offer several advantages, including real-time analysis, data formatting, auto calibration, and off-line processing. Probably the major advantage is distributed software control.

Other more specialized industrial applications of microprocessors abound. From energy subsystem control, to motor speed control, to numerical machine control, to robotics, all are increasingly applying microprocessors to obtain new levels of performance at economically viable cost levels. References 48 through 50, along with the *Proceedings* of the IECI '77 and '78, contain a wealth of information on these subjects.

Commercial applications

Commercial applications of microprocessors include their uses in communications (telephony and data), in medical applications, and in business applications (from word processing to intelligent supermarket weighing devices).

The most significant emerging trend in communications is the switch from predominantly analog (voice) traffic to a more balanced mix between voice and data. By 1985, communications traffic will be evenly divided between voice and data. This will most certainly spur the development of all-digital terrestrial channels. This, in turn, will greatly accelerate the use of distributed intelligence within the network. Intelligent multiplexers, concentrators, PBAXs, modems, etc., incorporating the ubiquitous microprocessor, will emerge. More and more, computers will take over the traditional role of operators. As all-digital networks emerge, there will be an increasing need for efficient techniques for transmitting voice in digital form.

A simple example can illustrate the power of the microprocessor in a data-communications subsystem, and the literature abounds with many more examples. Fig. 8 outlines a simple microprocessor-based multiplexer. Since all the low speed channels are asynchronous, it would be very difficult to implement the multiplexer without the aid of a microprocessor. By using microprocessor control, software can sample the communications channels rapidly enough to synchronize the random low speed inputs into a well-defined high speed output data stream. References 51 through 53 give additional information on data communication applications; references 54 through 56 discuss telephony, system monitoring and other communications applications of microprocessors.

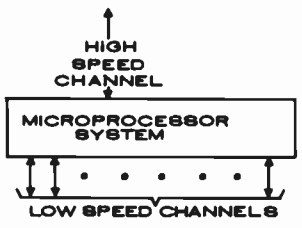


Fig. 8
The microprocessor-based multiplexer is a good example of the power of the microprocessor in data communications applications.

Microprocessors are being increasingly used in medical applications. Because of the technical and legal complexities of using new technologies in diagnosing, monitoring or treating humans, microprocessors today are being widely used on an experimental basis; their routine application is definitely only at the embryonic stage of development, however. Applications span the areas of cardiology, monitoring, clinical analysis, and prosthetics. Reference 57 describes a microprocessor-based blood gas analyzer; reference 58 covers a microprocessor-based monitor for EKG and blood pressure; reference 59 overviews the entire subject and contains a bibliography of 129 papers. Reference 60 provides additional insight into microprocessor uses in the biological laboratory.

A lot of R & D effort is being expended in the general area called "word processing" which spans applications ranging from intelligent typewriters, low-end distributed processing systems, through supermarket checkout terminals, and intelligent CRTs.

One interesting example is the new intelligent typewriter from the Qyx Division of Exxon Enterprises, Inc. It contains three microprocessors, a Z80 for word processing and master control, and a pair of F8s for drive motor control of the carriage and the rotary print head. It offers ultra-high print head positioning to permit automatic erase backspace. In addition, it incorporates intelligent features such as the ability to store and recall stock phrases, to center a line of type, and to automatically line up decimal points in columns of numbers. And all this for $1390.

Other examples abound—NCR, Financial Data Science Inc., and MSI Data Corporation have all developed teller terminals sporting microprocessors. Most of these also have telephone interfaces for interaction with centralized data bases. Yet another example is the IBM 5100 desk top computer.[64] References 61 through 63 overview other commercial applications including the intelligent CRT.

Other commercial applications of microprocessors include their use in security systems, environmental control systems, and intelligent weighing devices. In the latter, for example, the store clerk need only type in "price per pound" or "price per quarter-pound," and the price will be computed automatically and displayed. In addition, a receipt may be printed if desired. Other examples include taxi meters, automated gasoline dispensing pumps, and traffic-light controllers. In the latter, microprocessors combined with sensors, can adaptively vary light sequences to maximize traffic throughput, while minimizing waiting time for individual drivers.

Consumer applications

Microprocessors and associated LSI technology are having a major impact on consumer products.[65,66] From simply adding new features to standard products to making possible entirely new product categories, the consumer computer revolution is just beginning.

Low-cost microcomputers are already changing a host of consumer products. From exercise and coffee machines to electric ranges, from sewing machines to microwave ovens, a whole new generation of intelligent consumer products is emerging. Most home appliances typically need only relatively simple controls. For that reason 4-bit devices, with their low costs, are being shipped by the millions. Various RAM and ROM combinations and various technologies are usually available so that a specific control function can be implemented at minimal cost. In the TI TMS 1000 series, for example, p-channel MOS (pMOS), n-channel MOS (nMOS), and complementary MOS (CMOS) parts are available with 256 or 512 bits of RAM and 8k or 16k bits of ROM. Since ROM sizes always come in blocks, sufficient capacity is often available to allow for a fair degree of self-testing and diagnostic capability.

The products discussed above were predominantly examples where electronics have replaced mechanical controls in consumer applications. More interesting, however, are the host of new consumer products spawned by the ongoing revolution in LSI, many of which will be discussed below.

Nonvideo microcomputer-based games are just beginning to appear, but already most of the major toy manufacturers are getting heavily involved.[67,68] Judging from action games—such as Parker Brothers "Code Name: Sector", a one player submarine chase game, and Milton Bradley's "Battleship", a missile-firing two-player naval warfare game, and more cerebral games such as Mattel Inc.'s "Football"—toys will never be the same again. A more complex example is the Chess Challenger from Fidelity Electronics.

The video game revolution began in the early 1970s on two fronts. Atari pioneered the development of video arcade games with the 1972 introduction of "Pong." That same year, Magnavox introduced "Odyssey," a consumer ball-and-paddle game based on circuitry patented by Sanders Associates.

More recently, the consumer market has begun to evolve away from dedicated games and towards microprocessor-based programmable games.[69] Examples of such products include the Fairchild VES, the Atari VCS, and RCA Studio II.

Automotive applications of microprocessors will have a major impact on the car of the future.[70] From "under-the-hood" functions such as engine control (spark timing,[71] fuel metering, etc.) and braking[72] to "dashboard" functions (e.g., digital display of MPG), new levels of performance, economy, and information display will be achieved.[73]

Other consumer applications of microprocessors are under development. From simple systems which permit central control of up to 256 ac outlets (COBY 1, $400), to intelligent thermostats and personal computers (Radio Shack TRS-80, Commodore PET, Umtech Videobrain, etc.),[74,75] these products are emerging although it is not yet clear which categories will truly become volume products.

Conclusions

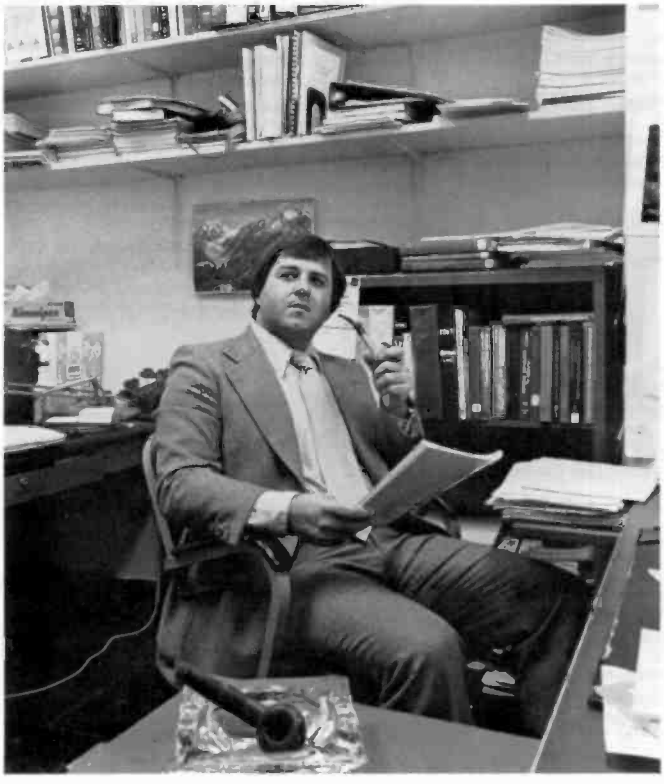
From the single-minded drive to enhance our computational capability, computers have evolved into an integral part of our society. Their penetration is so deep and wide that we can no longer divorce ourselves from it. In 1977, computer hardware represented almost 30% of the $67 billion U.S. electronics market and the percentage is continually growing (projected to be 50% by 1981). It has become one of the major driving forces of electronic technology and an indispensible contributor to the economy.

Future developments in computer technology will evolve in two different directions: both device technology and system architecture will be pushed to new frontiers by emphasis on ever-higher performance. Josephson junctions, LSI, and III-V compound integrated circuits, which promise sub-nanosecond cycle times and multiprocessor systems which increase throughput through parallel processing, are just a few examples of current efforts. These advancements will undoubtedly help computers reach many applications that are not feasible today.

Major developments in computer technology are also occurring in the other direction—lower cost. Each year, the complexity of LSI circuits doubles and cost per-function is reduced by 25%. Mass-produced computers costing a few dollars or less will be used in many applications that were not economically feasible before. The dramatic impact of microprocessors on the consumer market is just one such example. Others abound. Microcomputer shipments are already exceeding ten million per year, and this volume is growing exponentially. The computer will become ever more deeply rooted in our society, and its pervasiveness will become a symbol of modern life.

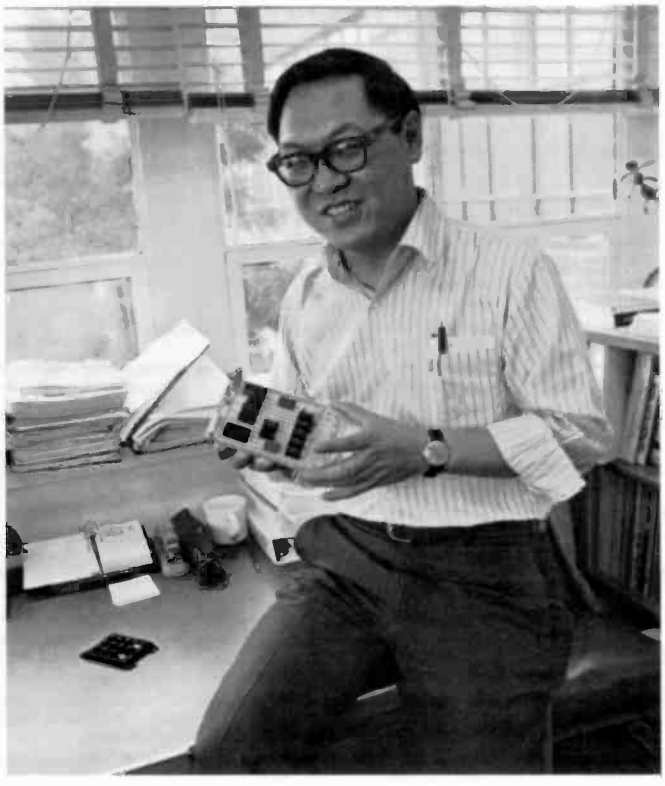
Paul Russo joined RCA Laboratories in 1970; since then he has done research in computer architecture, program behavior, system performance evaluation, microprocessors, and microprocessor applications. He is currently Head of TV Microsystems Research. Dr. Russo has taught several microprocessor courses and has received two Laboratories Outstanding Achievement Awards.

Contact him at:
Systems Research Laboratory
RCA Laboratories
Princeton, N.J.
Ext. 3234



Chih-Chung Wang was a senior scientist with B.F. Goodrich Co. before he joined RCA Laboratories in 1973. Dr. Wang has been working in computer-aided design and software development for integrated-circuit testers, and, more recently, microprocessors and microprocessor applications. He received a Laboratories Outstanding Achievement Award for this latter work.

Contact him at:
Systems Research Laboratory
RCA Laboratories
Princeton, N.J.
Ext. 3325



References

1. von Neumann, J.; "First Draft of a Report on the EDVAC," Univ. Pennsylvania, Philadelphia (Jun 1945).

Computer-aided design

2. Davis, C. B. and Payne, M. I.; "R-CAP: An Integrated Circuit Simulator," *RCA Engineer* (Jun-Jul 1975) pp. 66-71.
3. Rauch, W. A.; "FETSIM and R-CAP in Computer Aided Design," *RCA Engineer* (Jun-Jul 1975) pp. 72-76.
4. Rosenberg, L. M. and Benbassat, C. A.; "CRITIC: A Program for Checking Integrated-Circuit Design Rules," *RCA Engineer* (Dec-Jan 1975) pp. 16-19.
5. Zieper, H. S. and DeMeo, A. R.; "APAR Design Automation System," *RCA Engineer* (Dec-Jan 1975) pp. 30-35.
6. Feller, A.; Smith, A.; Ramondetta, P.; Lombardi, T.; and Noto, R.; "High-Speed CMOS-SOS LSI Using Standard Cells," *RCA Engineer*, (Dec-Jan 1975) pp. 36-40.
7. Bergman, R.; Aguilera, M.; and Skorup, G.; "MOS Array Design: Universal Array, APAR, or Custom," *RCA Engineer* (Jun-Jul 1975) pp. 40-47.
8. Hellman, H. I.; "TESTGEN: An Interactive Test Generation and Logic Simulation Program," *RCA Engineer* (Jun-Jul 1975) pp. 35-39.
9. Patel, C. B.; "CAD in Television Engineering," *RCA Engineer*, future issue.
10. Kolc, R. F.; "Design Automation for Multilayer Thick-Film Hybrids," *RCA Engineer* (Dec-Jan 1975) pp. 72-74.
11. Gargione, F. and Murphy, T. F.; "Computer Aided Design at AED," *RCA Engineer* (Dec-Jan 1975) pp. 41-45.
12. DeVecchis, J. A. and Smiley, J. W.; "Automated Design for Backplanes and Modules," *RCA Engineer* (Dec-Jan 1975) pp. 66-71.

Data processing

13. Freeman, J. W. and Mishra, D.; "Computerized Labor Control Raises Distribution Efficiency," *RCA Engineer* (Oct-Nov 1976) pp. 54-59.
14. Mishra, D. and Devarajan, A.; "Computerized Inventory Management at RCA Records," *RCA Engineer* (Oct-Nov 1977) pp. 64-66.
15. Havemeyer, C. and Lorentzen, R. R.; "SARA—A Research Planning Tool," *RCA Engineer* (Oct-Nov 1975) pp. 74-77.
16. Devarajan, A.; Martin, M.; and Mishra, D.; "How Operation Research Yields Dividends at RCA Records," *RCA Engineer* (Oct-Nov 1976) pp. 38-42.
17. Freiman, F. R.; "PRICE," *RCA Engineer* (Jun-Jul 1976) pp. 14-15.
18. Freiman, F. R.; "PRICE Applied," *RCA Engineer* (Aug-Sep 1977) pp. 24-27.

Manufacturing, process control, and testing

19. Blair, G. W.; "Computer Utilization in a Glass-Forming Operation," *RCA Engineer* (Feb-March 1976) pp. 10-13.
20. Special Issue on Automatic Test Equipment, *RCA Engineer* (Apr-May 1975).
21. Special issue on Automated Process Control, *RCA Engineer* (Oct-Nov 1975).
22. Fowler, V. and Roden, M.; "Adult II Test System," *RCA Engineer* (Aug-Sep 1975) pp. 30-31.
23. Lambert, H. R.; "Computer Controlled Automatic Test System," *RCA Engineer* (Aug-Sep 1975) pp. 20-22.

Telecommunications

24. Tyndall, E. G.; "Computer-Controlled Telelex," *RCA Engineer* (Oct-Nov 1974) pp. 68-71.
25. Correard, L. P.; "Current Concepts in Computer-Controlled Telex Switching Systems," *RCA Engineer* (Feb-March 1975) pp. 62-69.
26. Segrue, D.; "Computer-Controlled Voice/Data Switching System," *RCA Engineer* (Jun-Jul 1977) pp. 45-47.

Data acquisition and analysis

27. Perlman, B. S.; "Laboratory Automation in the Microwave Technology Center," *RCA Engineer* (Dec-Jan 1975) pp. 75-81.
28. Engstrom, R. W.; "Computerized Tomographic X-Ray Scanners," *RCA Engineer* (Apr-May 1977) pp. 18-20.

Multiprocessor systems

29. Special Issue on NBC Television Central, *RCA Engineer* (Apr-May 1976).
30. Wyant, E.; "Computer Security Management at Walt Disney World," *RCA Engineer* (Jun-Jul 1973) pp. 72-75.
31. O'Connel, J. H. and Priestley, D. M.; "Minicomputer Applications in Building Management Systems," *RCA Engineer* (Feb-Mar 1975) pp. 74-76.

Testing

32. Eleccion, M.; "Automatic Testing: Quality Raiser, Dollar Saver," *IEEE Spectrum* (Aug 1974).
33. Trifari, J.; "Micro's Boost Test-Gear Power," *Electronic Products Magazine* (Oct 1976).
34. Marcantonio, A. R.; "Microprocessor-Based Printed Circuit Board Tester," *Proceedings of IECI 78*, Philadelphia, Pennsylvania, (Mar 20-22, 1978).
35. Lyman, J.; "Automated Circuit Testers Lead the Way Out of Continuity Maze," *Electronics*, (Aug 7, 1975).

Process control and data acquisition

36. Wang, C. C.; Wu, C. T.; Russo, P. M.; Abramovich, A.; Marcantonio, A. R.; "Microprocessors in Manufacturing and Control," *RCA Engineer*, (special issue on microprocessor technology, Feb-Mar 1977).
37. Wittke, J. and Simpson, T. F.; "Microcomputers in Picture Tube Manufacturing," *RCA Engineer*, this issue.
38. Wiessberger, A. J.; "Microprocessors Simplify Industrial Control Systems," *Electronic Design* (Oct 1975).
39. Auslander, D. M.; Takahashi, Y.; and Tomizuka, M.; "Direct Digital Process Control: Practice and Algorithms for Microprocessor Applications," *IEEE Proceedings*, (special issue on microprocessor applications, Feb 1978).
40. Abramovich, A.; "An Interpolating Algorithm for Control Applications on Microprocessors," *Proc. of IECI 78*, Philadelphia, Pennsylvania (Mar 20-22, 1978).
41. Koren, Y.; "Computer-Based Machine-Tool Control," *IEEE Spectrum*, (Mar 1977).
42. Weissberger, A. J.; "Microprocessors Expand Industry Applications of Data Acquisition," *Electronics* (Sep 1974).
43. Grandbois, G.; "Log Data Under Microprocessor Control," *Electronic Design* (May 10, 1976).

Instrumentation

44. Rangle, W. C. and Keith, N.; "Microprocessors in Instrumentation," *IEEE Proceedings* (special issue on microprocessor applications, Feb 1978).
45. Wu, C. T.; "CVM—A Microprocessor-Based Intelligent Instrument," *RCA Engineer* (special issue on microprocessor technology, Feb-Mar 1977).
46. Oliver, B. M.; "The Role of Microelectronics in Instrumentation and Control," *Scientific American* (Sep 1977).
47. Loughry, D. C. and Allen, M. S.; "IEEE Standard 488 and Microprocessor Synergism," *IEEE Proceedings* (special issue on microprocessor applications, Feb 1978).

Robotics

48. Sohrabji, N.; "Microprocessors Extend Scope of Automated Manufacturing," *EDN* (Mar 5, 1978).
49. Doi, Y.; "Robots Get Smarter and More Versatile," *IEEE Spectrum* (Sep 1977).
50. Goksel, K. and Parrish, Jr., E. A.; "The Role of Microcomputers in Robotics," *Computer Design* (Oct 1975).

Communications

51. Lippman, M. D.; "Microprocessors in Data Communications," *RCA Engineer* (special issue on microprocessor technology, Feb-Mar 1977).
52. Stanzione, D. C.; "Microprocessors in Telecommunications Systems," *IEEE Proceedings* (special issue on microprocessor applications, Feb 1978).
53. Gundlach, R.; "Large-Scale Integration is Ready to Answer the Cell of Telecommunications," *Electronics* (Apr 28, 1977).
54. Henn, W.; "Microprocessor Automated Alarm System," *RCA Engineer*, future issue.
55. Melvin, D. K.; "Microcomputer Applications in Telephony," *IEEE Proceedings* (special issue on microprocessor applications, Feb 1978).
56. Mayo, J. S.; "The Role of Microelectronics in Communication," *Scientific American* (Sep 1977).

Medicine

57. Margalith, A.; Mergler, M. W.; and Primiano, Jr., F. P.; "Microprocessor-Based Blood Gas Analyzer," *IEEE Trans. on Industrial Electronics and Control Instrumentation* (Feb 1978).
58. Haue, P. and Graham, M.; "Microprocessor Monitor for the EKG and Blood Pressure," *Digest 1976 WESCON Technical Papers*, No. 22/3 (1976).
59. Klig, V.; "Biomedical Applications of Microprocessors," *IEEE Proceedings*, (special issue on microprocessor applications, Feb 1978).
60. Schoenfeld, R. L.; Kocsis, W. A.; Milkman, N.; and Silverman, G.; "The Microprocessor in the Biological Laboratory," *IEEE Computer* (May 1977).

Commercial products

61. Walker, G. M.; "Consumer/Commercial Microprocessors Go Public," *Electronics* (Jul 11, 1974) pp. 92-95.
62. "Here Comes the Second Computer Revolution," *Fortune* (Nov 1975).
63. Gray, M. T.; "Microprocessors in CRT Terminal Applications—Hardware/Software Tradeoffs," *IEEE Computer* (Oct 1975).
64. Roberson, D. A.; "A Microprocessor-Based Portable Computer: The IBM 5100," *IEEE Proceedings* (Jun 1976).

Consumer products

65. Walker, G. M.; "LSI Controls Gaining in Home Appliances," *Electronics*, (Apr 1977).
66. Walker, G. M.; "LSI Chips Taking Over More Household Chores," *Electronics* (Oct 1976).
67. Rosenblatt, A.; "Electronic Games No Longer Need TV," *Electronics*, (Aug 1977).
68. Mennie, D.; "Self-Contained Electronic Games," *IEEE Spectrum*, (Dec 1977).
69. Russo, P. M.; Wang, C. C.; Baltzer, P. K.; and Weisbecker, J. A.; "Microprocessors in Consumer Products," *IEEE Proceedings* (Feb 1978).
70. Marley, J.; "Evolving Microprocessors which Better Meet Needs of Automotive Electronics," *IEEE Proceedings* (Feb 1978).
71. Robbi, A. D. and Tuska, J. W.; "A Microcomputer-Controlled Spark Advance System," *RCA Engineer* (special issue on microprocessor technology, 1977).
72. Lile, W. R. and Tuska, J. W.; "Electronic Control of Vehicle Brakes," *RCA Engineer* (special issue on microprocessor technology, 1977).
73. Belohoubek, E. F.; Cusack, J. M.; Risko, J. J.; and Rosen, J.; "Microcomputer Control for the Car of the Future," *RCA Engineer* (special issue on microprocessor technology, 1977).
74. Key, A. C.; "Microelectronics and the Personal Computer," *Scientific American* (Sep 1977).
75. Doerr, J.; "Low-Cost Microcomputing: The Personal Computer and Single-Board Computer Revolutions," *IEEE Proc.* (Feb 1978).

Reprint RE-23-6-5|Final manuscript received April 24, 1978.

Interpreter control program simplifies automatic testing

Interpreter software makes this printed-circuit tester easy to use and modify.

A. R. Marcantonio

The COSMAC Automatic Inspection System (CAIS), shown in Fig. 1, was developed for the production-line automated testing of a populated printed-circuit (PC) board. It is a go/no-go tester that can be operated by a relatively unskilled person. CAIS uses easy-to-service modular electronics and application-oriented software to check a PC board consisting of passive components. Several of these systems have been on line at our tv manufacturing plant for over a year. They have proven to be reliable as well as extremely effective in reducing the reject escape rate for the assembly tested.

Simple, low-cost tester

The inspection system had to provide a simple method for specifying the PC board tests.

This requirement led to the use of an interpreter and relatively simple test directives, Fig. 2. Interpreters can make one computer appear to be an entirely different computer to the user by accepting user-problem-oriented instructions and converting them to the necessary machine-level instructions. Thus, the simplicity requirement was met by using an interpreter program which executes the PC board tests, or test directives, directly.

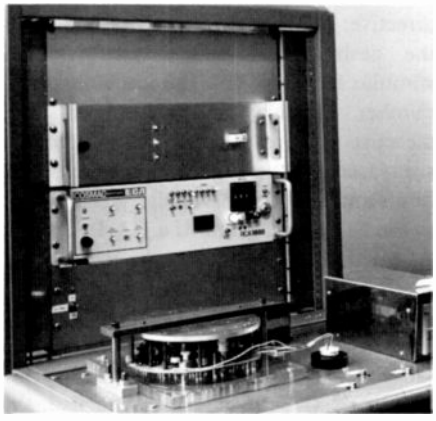


Fig. 1
Hardware for the CAIS (COSMAC Automatic Inspection System) consists of the COSMAC Development System (top), contact assembly and printed-circuit board under test (bottom left), and printer. Several systems have been in operation for over a year.

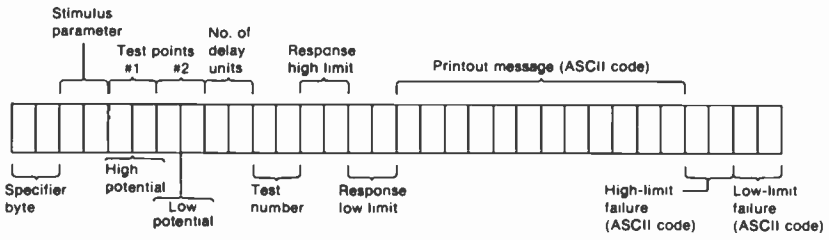


Fig. 2
Test directives for each test fit into sixteen bytes of memory. Test engineers find them powerful and easy to use. Meanings of individual directives are explained in text.

Total system hardware costs had to be low (about $5K) and test instructions had to be easy to modify.

A microprocessor (μP) was chosen for the controller function because it fulfilled the cost requirements and still offered the advantages of flexibility. Computations required during testing are well within the capabilities of most available μPs so a more powerful CPU was not justified. In particular, the total board handling time is about nine seconds. Of that nine seconds, only one second is controller execution time; and of that second about 250 ms is lost to relay contact bounce settling (2 ms/test), and about 250 ms is given up to capacitor charging prior to dc measurements (anywhere between 1/2 ms and 8 ms/test). Therefore, only about 1/2 second out of 9 seconds, or 5.6% of the total board handling time, could be improved by a faster CPU. Or, stating it differently, even if CPU execution time were zero, it would still take 8.5 seconds to check the board.

RCA's COSMAC 1802μP[1] was picked for its strong I/O architecture and familiarity among members of the project. Since the projected number of testers was low, the COSMAC Development System[2] (CDS) was selected as the fundamental building block. The CDS was used as a base for an earlier developmental system,[3] so many of the interface designs could be applied directly to this tester application. Fig. 3 is a system block diagram.

CAIS consists of the COSMAC Development System, a set of custom interfaces, a control panel, a printer, and an air-actuated contact assembly, mounted in a standard rack. Custom interfaces include a 12-bit A/D converter, relay matrix, signal multiplexer, 20-column printer interface, 2k erasable programmable read-only memory (EPROM), control panel, and signal package containing a stable dc source, a pulse generator and detector, and an ac generator and phase-measurement circuit. Details on measurement techniques and testing capacity are given in the Appendix. More specifics on the hardware system may be found in Ref. 4.

Automatic "power-on-start" was another requirement.

The "power-on-start" circuit alleviates any need for the system operator to disturb the main control panel by issuing a system reset pulse followed by a run program signal, immediately after rack power is applied.

The memory requirements are relatively small.

Fig. 4 depicts the major software sections. The interpreter control program occupies

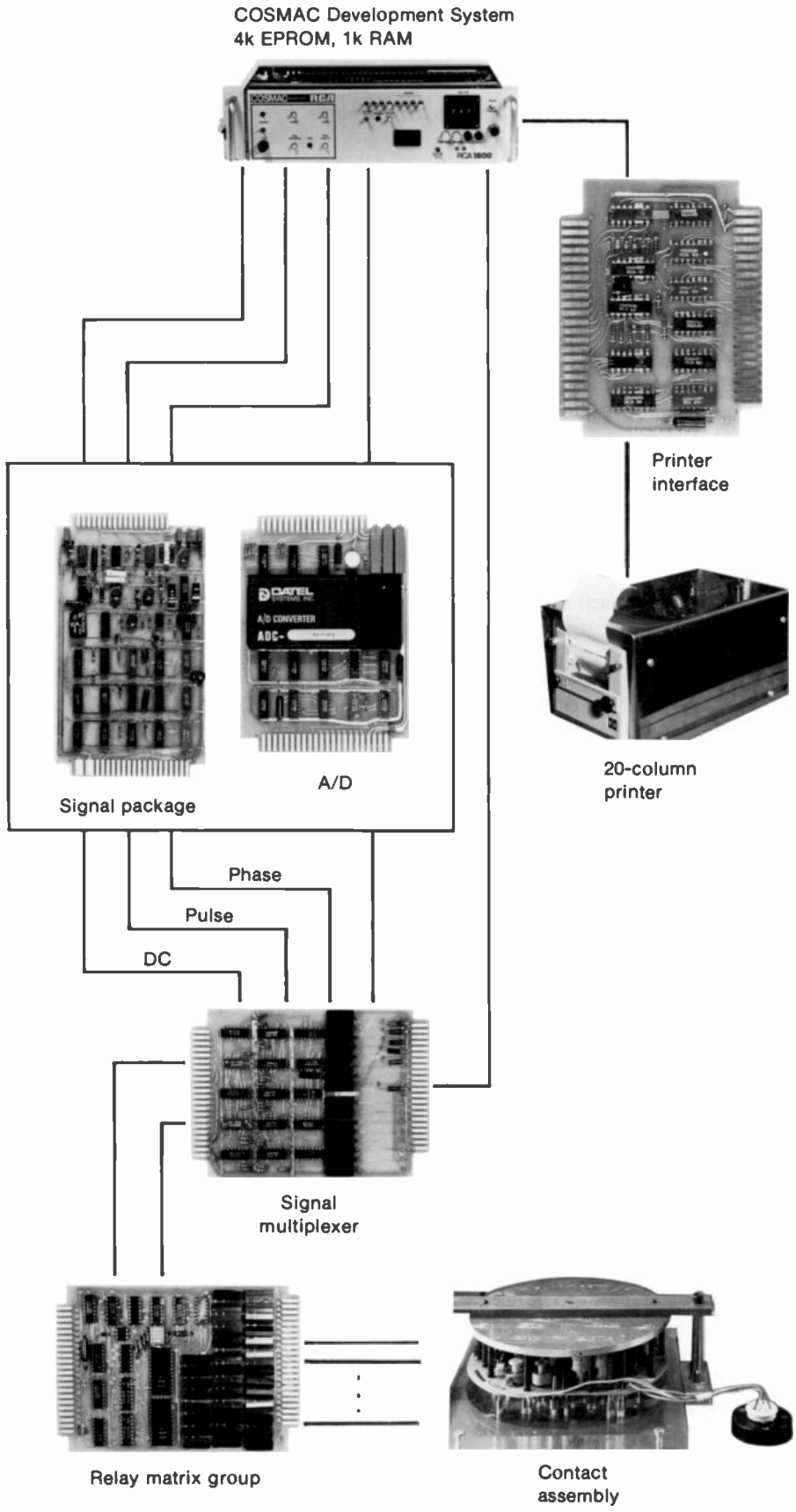


Fig. 3
Test system consists of COSMAC development system, signal circuitry, relay matrix, and contact assembly (shown contacting printed-circuit board under test).

less than 1.5k bytes of a 2k-byte EPROM memory card. Test directives completely occupy a 2k-byte EPROM memory card. This corresponds to 128 separate tests at 16 bytes/test. Thus, the interpreter program and the test instructions are completely distinct, both in address space (separate memory pages) and physical space (separate EPROM cards). The system requires about thirty bytes of RAM for buffer and stack use.

How the system works

The control program has five different operating modes that are selected from the control panel (Fig. 5). In normal production-line use, the tester operates in the automatic mode, running through the test sequence and printing output only after failed tests. The other modes—manual step, manual repeat, single-test step, and single-test repeat—are used for system testing and diagnosis, and are activated by switches on the control panel.

To use the tester, the operator first verifies that the mode switches are set to automatic and applies rack power. The power-on-start circuit then causes the control program to come up running. A PC board is placed on the contact assembly, the assembly closed, and the "start test" button depressed.

The control program jumps to the automatic-mode routine and executes the first test directive by calling the test subroutine. This subroutine uses a register pointer to step through the current test directive: to close relays corresponding to the desired test points, energize the stimulus required, hold the stimulus for the number of delay units specified, and measure the response. This reading is compared to the high and low response limits and a PASS/FAIL signal is returned to the calling program. If a FAIL signal is returned, the FAIL lamp is turned on, a specified message is printed, and the contact assembly opened. The test operator places the printed reject information with the PC board and directs it to a repair person. If a PASS signal is returned, the control program simply moves to the next test directive. The control-program interpreter continues this basic two-step process of fetching a test directive and executing it until all the PC board tests are performed. At the conclusion of the automatic test sequence, if no tests have failed, the PASS

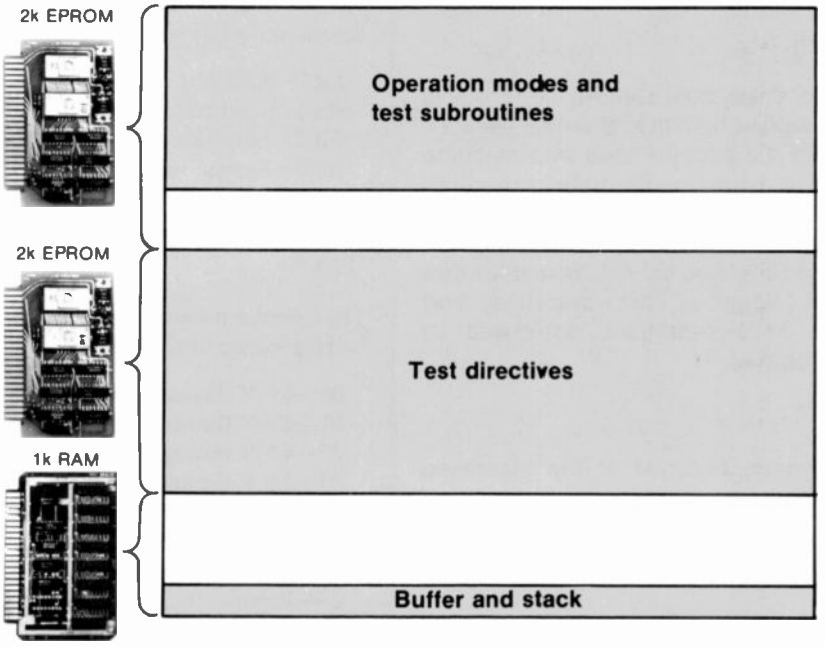


Fig. 4
Memory map shows the software needed for the tester system. At top, most of a 2k EPROM is used to contain the interpreter control program. In it, separate 256-byte "micropages" hold the entrance routine; automatic mode routine; manual/single modes; test subroutine (2 micropages); and the hex/ASCII, relay-select, time-delay, and print subroutines. Second 2k EPROM is used up completely by 128 test directives. System also uses 30 bytes of 1k RAM for print buffer and working stack.

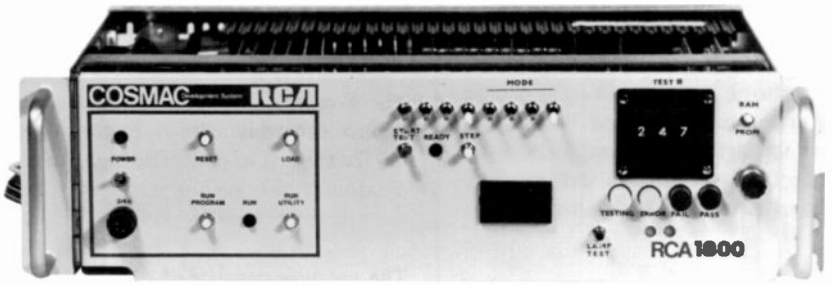


Fig. 5
Control panel has toggle switches (center) for selecting test mode, thumbwheel switches (top right) for entering test numbers in non-automatic modes, and status indicators (bottom right).

lamp is turned on and the contact assembly is opened.

How the interpreter simplifies testing

The control-program interpreter written for CAIS lets the user work with a simplified set of instructions specifically oriented to the tester.

The test engineer specifying the tests does not have to know or understand COSMAC machine-level instructions. As far as the test engineers are concerned, they are programming a machine that executes test directives, which are sixteen-byte instructions that enable one to have complete control over the selection of test parameters.

The interpretive approach has proven to be extremely effective in implementing the control program for CAIS. This effectiveness has been demonstrated by the ease with which test engineers using the system

How do users like CAIS?

The new CAIS Project has met all of the goals that were outlined for this important production improvement project. These goals were to develop an improved test/inspection system to allow fewer faulty subassemblies to reach the final production stage, to minimize operator subjective judgement, to reduce production costs, and to improve the technical skills of the support engineers by introducing new technology into the factory.

The results are that the first system was developed in time to evaluate the preproduction subassemblies and that three additional systems were constructed with no hardware design changes to meet the manufacturing production schedule. This new system, using the COSMAC microprocessor, will detect approximately 96% of all of the subassembly problems. The escape factor for tests that have been programmed into the computer PROM is approximately 0.1%. The production operator exercises minimum operational judgement, but handles the production flow into and out of the inspection fixture.

The four computer-aided production systems have saved over $100,000, compared to the normal production cost before their introduction. A second engineer has been assigned the task of modifying the software test directives, and the original project engineer has assumed a larger computer-assisted ATE assignment. Therefore, this project has successfully assisted in increasing the technical skills available to our local engineering function from the Princeton Advanced Development Group's assistance in this joint project.

John Keith
Mgr., Test Engineering
Consumer Electronics Division
Bloomington, Ind.

Why use an interpreter?

Interpreters make programming easier for the system user. An interpreter is a program that performs operations specified by a machine-like pseudo-code by using a subroutine to translate the pseudo-code into machine instructions and to execute them in turn for each pseudo-code instruction. The interpreter performs in software the same function that the CPU performs in hardware by simulating the fetch-execute hardware sequence. This allows a programmer to work with a simplified set of instructions that can be oriented toward his problem. Some of the advantages and disadvantages of interpreter-program implementations compared to machine-language programs are listed below.

Advantages:

Interpreter programs are more compact, because of the increased functional power of each statement.

Time between program design and implementation is shorter, since programming is done in a higher-level language.

Debugging is easier, because the interpreted machine can be reviewed at any time by examining the memory contents of the operating machine.

Disadvantages:

More memory capacity is required, since the interpreter software must remain in the machine during execution.

Functions implemented with an interpreter generally run slower than those implemented in machine language, because the interpreter subroutines must be called and a fetch-execute cycle simulated for each pseudo-code statement.

The interpreter itself must be designed, implemented, and debugged.

Interpreter programs have been designed to fulfill a variety of functions. COSMAC microprocessor-based implementations include an interpreter aimed at video games,[5] and the control program described here. The COSMAC 1802 microprocessor architecture is particularly well suited to implementing interpretive languages. It facilitates various addressing modes and subroutine calling schemes, and provides sixteen general-purpose registers.

are able to generate the test profile, and their complete confidence in and acceptance of the system.

Each test directive is completely specified by 16 bytes of information residing in EPROM memory.

Fig. 2 has indicated the variable names associated with each byte. The test directives enable the program to control the selection of test points, type and polarity of stimulus, high/low response limits, designation of a system self-test, and the time delay between the application of stimulus and the response measurement. This method of specifying tests fulfills three basic requirements. First, it is easy to master. No special skill is required to write the test directives other than a knowledge of the unit being tested. Second, it is simple to modify the test profile. Since the test directives reside in a separate EPROM memory, all that's required is the modification and replacement of the directive EPROM. Third, complete accurate documentation on exactly which tests are currently in the profile may be obtained easily by using the system monitor to list the EPROM.

What are the functions of the individual test-directive fields?

This is how the variables shown in Fig. 2 are identified.

The *specifier* qualifies the test directive according to bits set as follows:

Bit 0—Last test in sequence.
Bit 1—First test after open/continuity.
Bit 2—Skip this test directive.
Bit 3—System self-test.

(No bits set indicates that none of the above apply.)

The *stimulus parameter* assigns a stimulus corresponding to the byte indicated.

00—4.5 V through 50 Ω
01—4.5 V through 100 Ω
02—4.5 V through 1000 Ω
03—4.5 V through 10,000 Ω
04—4.5 V through 100,000 Ω
05—4.5 V through 500,000 Ω
16—5.0-kHz sine wave
27—Pulse

The *test points* receive the stimuli indicated by the stimulus parameter. The first test-point numbered receives the high-potential connection, the second, the low potential.

The *number of delay units* sets the amount of time that will expire between the application of the stimulus and the response measurement. Each unit equals 1/2 ms (128 ms max.).

The *test number* identifies a particular test directive (01-FF).

The *required response limits* define the region into which the A/D reading must fall for the test to pass. The high limit is the maximum pass response, the low limit is the minimum pass response.

The *message* consists of ASCII characters that are to be printed when an out-of-limit result takes place during automatic-mode testing.

The *high-limit fail character* is the ASCII character printed to indicate high-limit failure, and the *low-limit fail character* is the ASCII character printed to indicate low-limit failure.

How do the test directives work in a specific example?

Fig. 6 is a sample test directive that demonstrates the implementation of a specific PC board test. In it:

The specifier byte of '00' indicates that test directive is normal, since none of the qualifier bits are set.

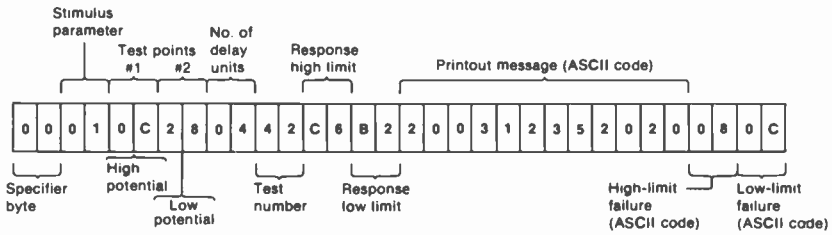


Fig. 6
Sample test directive with hexadecimal interpreter entries. Full explanation of codes is in text, but basically, for this particular dc check, test system puts 4.5 V through 100 ohms into a diode and measures voltage across diode. Reading must be between 3.14 and 3.49 V; otherwise 'L' or 'H' (for "low" and "high" results) will be printed on output.

The stimulus parameter of '01' selects 4.5 V through 100 Ω.

The test point #1 value of '0C'$_{16}$ places the more positive side of the stimulus on test point #12$_{10}$ of the printed-circuit board.

The test point #2 value of '28'$_{16}$ places the more negative side of the stimulus on test point #40$_{10}$ of the printed-circuit board.

The delay-unit value of '04' puts a 2-ms delay between stimulus application and response measurement.

The test number of 42$_{16}$ tells that the test is number 66$_{10}$ in sequence.

The required response high limit of 'C6' gives 3.49 V dc as the highest pass reading allowable. Voltages are represented in hex code by converting the analog voltage range of 0-4.5 V dc to 0-255 binary counts (00-FF$_{16}$) in the analog-to-digital converter. For example, C6$_{16}$ = (C6$_{16}$/FF$_{16}$)4.5 = (198/255)4.5 = 3.49 V.

The required response low limit of 'B2' makes 3.14 V dc the lowest pass reading allowable. Note that response measured must fall between high limit (3.49 V dc) and low limit (3.14 V dc) in order to pass.

The message code of '200312352020' prints as "space CR 5 space space" if the test fails.

The high-limit fail character of '08' prints the character 'H' if the measured response is above the high limit.

The low-limit fail character of '0C' prints the character 'L' if measured response is below the low limit.

Conclusions

An interpreter control program has been implemented for a COSMAC μP-based printed-circuit-board tester. It allows tests to be specified with instructions that are oriented toward the problem. Test directives, the instructions acted on by the control program interpreter, are easily written and their implementation is as simple as programming an EPROM. In addition, complete test documentation may be obtained by listing the contents of the test-directive EPROM.

Acknowledgements

I am grateful to A. Abramovich, K. Brooks, T. F. Lenihan, P. M. Russo, and C. C. Wang, whose efforts helped make this system possible.

References

1. *User Manual for the CDP1802 COSMAC Microprocessor*; MPM-201, RCA Solid State Division, Somerville, N.J.
2. *Operator Manual for the RCA COSMAC Development System*; MPM-208, RCA Solid State Division, Somerville, N.J.
3. Marcantonio, A.R.; Russo, P.M.; and Wang, C.C.; "An automation-oriented microprocessor-based developmental system," *Proc.* of IECI '77, Philadelphia, Pa., (Mar 1977).
4. Marcantonio, A.R.; "Microprocessor-based printed circuit board tester," *Proc.* of IECI '78, Philadelphia, Pa., (Mar 1978).
5. Baltzer, P.K.; Weisbecker, J.A.; and Winder, R.O.; "Interpretive programming of small microprocessor-based systems," *Proc.* of IEEE Electro '76, Boston, Mass., (May 11-14, 1976).

Appendix— Measurement techniques and system capacity

Resistance measurements are made using a stable dc source by calculating the voltage drop across a precision resistor in series with the unknown resistor. The magnitude and polarity of the potential across diodes is checked by applying a low-amplitude dc current and measuring the response with an A/D converter. Reactive components are tested by either a fast pulse technique or by calculating the amount of phase shift the component effects on a 5.0-kHz sine wave. All measurements are referenced to the corresponding readings obtained from known valid PC boards. Variations of these readings determine the limits stored in the directive EPROM. A system self-check is performed once each complete test cycle to establish the integrity of each stimulus source (dc, 5.0-kHz sine wave, and pulse). Capacity of the system is 128 tests/2k EPROM and 32 test points/relay-card pair. In our system, 128 tests and 64 test points were defined, requiring 2 relay-card pairs.

Reprint RE-23-6-20
Final manuscript received January 6, 1978.

Angelo Marcantonio has been involved with computer hardware and software systems since he joined RCA Laboratories in 1970. As a member of the original microprocessor team, he designed and programmed applications software and interpretive languages. He is presently a member of the TV Microsystems Research group, where he is engaged in the design of microcomputer-based consumer products and automated test and control equipment for manufacturing.

Contact him at:
TV Microsystems Research
RCA Laboratories
Princeton, N.J.
Ext. 2750

Why not do it in software?

L.A. Solomon|D. Block

One reason designers choose microprocessor-based systems is the lower parts count. But by using the microprocessor's software ability to advantage, parts count can be lowered even further.

The advantages of choosing a microprocessor-based system over designing with standard ICs are, by now, well known. One of the principal reasons is the reduction in parts count that can be realized in a microprocessor system. All but the very simplest systems consist of several ICs—RAM, ROM, and I/O devices. A major design consideration involves tradeoffs between hardware and software, which reflects itself in the ROM to I/O mixture. The economics indicate that the more functions handled in software, the less expensive and more flexible the system will be. Thus the best departure point for a design is to try to do everything in software. Then functions are relegated to hardware as the speed/processing capability of the CPU becomes taxed. The following examples illustrate the approaches for handling typical I/O functions by means of software.

A classical system

Consider the simple system diagrammed in Fig. 1—a simple input device, microprocessor, and output device. The specific functions have been purposely omitted. A classical software control flowchart for such a system is shown in Fig. 2. We see the standard initialization procedure followed by an input, processing and output procedures, and a final loop back to repeat the action again. Without knowing the details of the hardware or software design nor the specific application in mind, certain predictions can be made about this system.

The software cycle time will affect the type of input and output devices chosen.

First, let's consider the software cycle time, the time to go completely through one loop of the procedure. The cycle time is the sum of the time spent in each portion of the software, including input, processing, and output. Since the program apparently waits for an input, the time spent in the input block is indeterminate. Thus the *system* cycle time is indeterminate. This has immediate impact on the selection of both input and output devices used in the system. The output device, for instance, must be capable of operating for prolonged periods without processor attention. Therefore, it must be a device that is self-refreshing or contains a latch. It certainly cannot be dynamic, since the simple software structure shown thus far has no provision for refreshing. As dynamically refreshed displays have the potential for lower cost, we should consider the static requirement as a serious drawback to the straightforward, simplistic approach.

Next, consider the input device. Perhaps it presents data to the processor at an uneven rate, which is most certainly true if the entry device is human-operated. If a keyboard is being serviced, for example, the time between keystrokes can vary from a few milliseconds to several seconds or minutes. Yet, the processor must be fast enough to respond to the fastest keystroke rate. If each keystroke results in some serious analysis on the part of the microprocessor, the minimum time between keystrokes might have to be lengthened. If the microprocessor must additionally activate some hard-copy device such as a drum printer, several hundred milliseconds might have to be dedicated to the output processing block as well. In that case, we might be designing the system such that the operator would have to be "tuned " to the system rather than the reverse. To compensate for lack of time between keystrokes, you could choose a microprocessor that can "handle the job", i.e., one that goes faster (but costs more). Another solution to the time problem could be to design in an "intelligent" keyboard controller or some sort of buffering device to smooth out the input rate. However, resorting to this additional hardware may not be necessary.

There is a third case in which the simple structured system doesn't work at all. If, for instance, the input to the system was a requirement for some lengthy output report, then while the system was busily processing the request, no new input could be presented. Any input during that time would be lost.

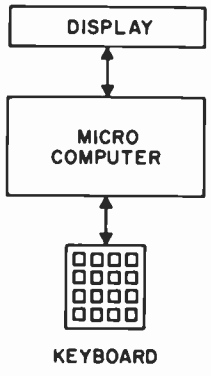


Fig. 1
Starting point. This classical system can be implemented in varying combinations of hardware and software.

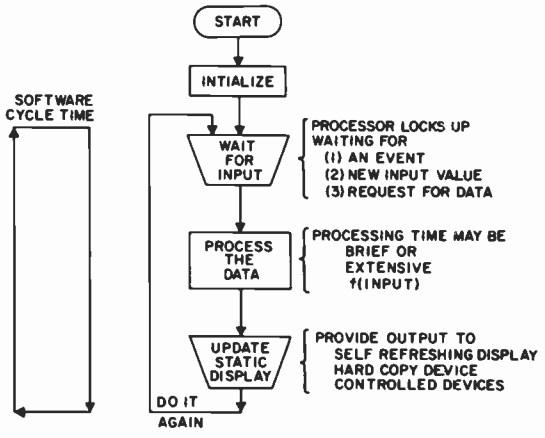


Fig. 2
Software cycle for our generalized system can be divided into three parts—input, processing, and output.

System performance can be limited by either hardware or software.

We now see that the system "specified" in Fig. 2 is a "simple" one, but one with severe limitations and pitfalls. The system will inherently go from an I/O-bound block to a processing-bound block and back to being I/O-bound. Also, since it makes no attempt to level peak loads, this system will waste the microprocessor's power. We will next examine how to minimize wasted processing time and level loads to a manageable degree.

Hardware-software tradeoffs

The microprocessor system has to have ROM, RAM, and CPU. By using them to a greater capacity, other hardware can be eliminated.

Let's take a closer look at the proposed system and see what is involved in the hardware. Fig. 3 has broken down the initial diagram into smaller blocks. The display now has some controlling circuitry, as does the keyboard. These are shown along with the indispensable elements of any microprocessor system—ROM, RAM, and CPU. The system must have these, but must it have the controllers? Can we perform the controller function by marginally increasing the cost of some other portion of the system? Perhaps!

The software approach distributes the workload.

By doing these functions in software, we may eliminate the controllers at the cost of enlarging the system ROM. But, here's an interesting fact in IC economics. Since ROMs come in fixed increments, it is often no more expensive to have a program that is 1024 bytes long than one that is 527 bytes, even though one is almost twice as long as the other. To take the software approach, we will restructure the simple-minded solution originally suggested and, through workload distribution and continuous processing, obtain minimum system hardware.

We will change our approach so that instead of waiting for an input to take place, we will simply look at the input periodically. If no input is present at that time, we will skip the input operation and go on to see if something else remains to be done. Refreshing a dynamic display, for instance, always needs to be done, and so a dynamic display meshes neatly with our revised concept. After refresh, we will again look for an input. If one should now be pending, it will be accepted. Now we must make a decision. If the input can be readily handled, we will do so immediately. If not, the input will be saved for later when we have the time to handle it. Alternately, the processing associated with the input may be broken up into smaller computational blocks interspersed with refreshes of the display. These approaches are flowcharted in Fig. 4.

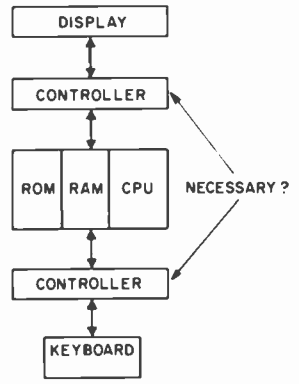


Fig. 3
System defined in smaller blocks leads to the question "How much of this hardware is necessary?" ROM, RAM, and CPU are essential, but controllers might be replaced by software.

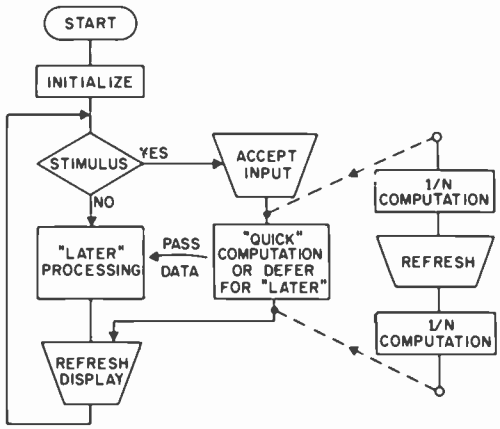


Fig. 4
Balancing the load with software. System decides whether to accept input, do computation, or refresh display.

Larry Solomon is leader of the MOS systems software group at the Solid State Division. He was the principal designer of the COSMAC resident software, and is responsible for custom microprocessor-system software designs. He came to RCA in 1974 with five years of experience in minicomputer systems design for real-time data collection and analysis systems.

Contact him at:
MOS Systems Software
Solid State Division
Somerville, N.J.
Ext. 6349

Dennis Block, one of the pioneer members of the RCA COS/MOS activity, has worked in the design of standard and custom circuits and applications for the CD4000 family. He is presently working with the CDP1800-series microprocessor line, designing hardware support products and custom systems plus providing support for the applications and marketing activities.

Contact him at:
Microprocessor Applications Engineering
Solid State Division
Somerville, N.J.
Ext. 6214

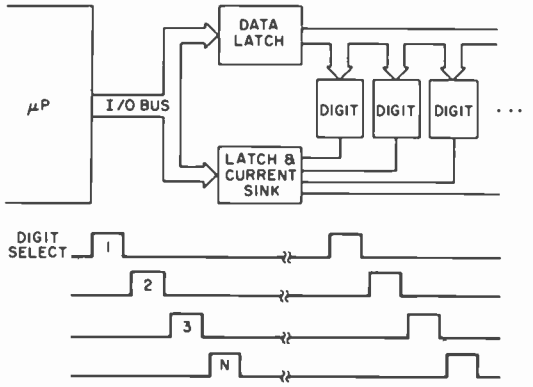


Fig. 5
Typical multiplexed display system.

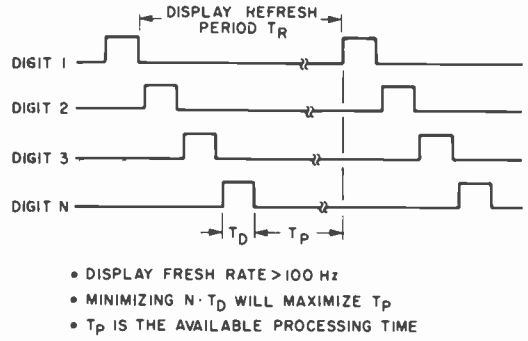


Fig. 6
Display refresh rate affects both software and hardware. Refresh rate must be fast enough to prevent flicker, but still allow time for processor to do other work.

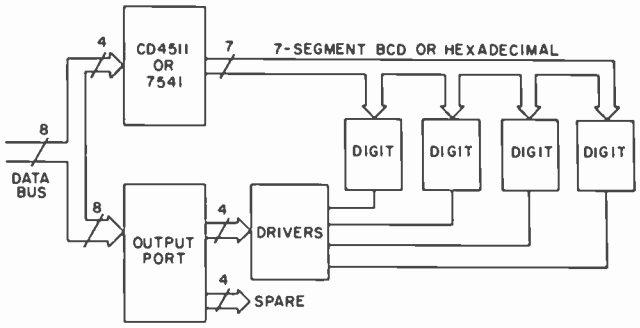


Fig. 7
Display information handled by hardware using MSI devices.

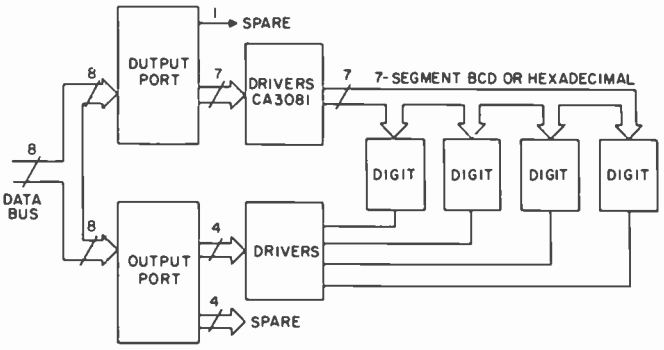


Fig. 8
Same display as Fig. 7, but handled by an output port and a software look-up table.

Handling a dynamic display

How do you divide the processor's time between refreshing the display and doing the rest of its processing load?

Fig. 5 shows a typical multiplexed display system and Fig. 6 gives the details on the display refresh rate. The minimum refresh rate for any digit should be 100 Hz. This is fast enough to prevent flicker under most stationary display conditions. The actual ON time of any digit is a compromise between the intensity of the display and the time remaining within the refresh period (T_R) for the processor to do some other work. It is desirable to minimize the display time (T_D) so that a maximum of processing time (T_P) is left for the rest of the processing load.

It is customary to overdrive multiplexed LED displays to increase their apparent brightness. The extent to which you can do this is a function of the duty cycle, T_D/T_R, of the display. This technique is not without its risks, however, for if the program crashes or hangs up someplace (because of a program bug, noise injected into the system, component failure, etc.) it is quite probable that a digit driver will be incinerated. You should be aware of this and take appropriate precautions, particularly when debugging your system.

The segment information for a 7-segment display can be handled in either of two ways. If the data is in BCD, a device such as the CD4511, which contains a latch, BCD-to-7-segment decoder, and drivers, can be used as shown in Fig. 7. A device such as the 7541 provides a similar function for hexadecimal displays.

Or, instead of doing the code conversion in hardware, do it in software with a simple look-up table.

With this software method, a less expensive, simple output port can be used (Fig. 8) instead of the MSI devices above. But, since I/O ports generally do not have enough drive to handle LEDs directly, an intermediate stage of buffering is necessary, defeating the apparent cost-effectiveness of this approach. No clear-cut distinction appears here, since factors such as the volume of the devices, cost of the I/O devices required by your processor, and type of display chosen all enter the picture.

So far we have minimized output hardware and smoothed out the processing flow by multiplexing the input and display. Next, let's take a closer look at the input block and see what we find there.

Handling single-signal inputs

Software can even debounce switch inputs.

First, consider a basic switch circuit shown in Fig. 9. We shall assume a CDP1802 microprocessor having a flag input; the mechanical switch is shown attached directly to that input. It could just as well be a software-testable bit of an input port—the principles to be discussed would be the same. To signal the processor, a change on the flag line from a logic 1 to

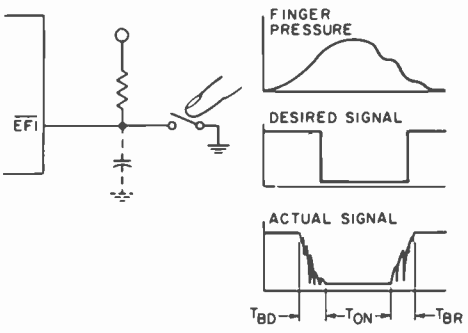


Fig. 9
Switch inputs present a problem because of switch bounce, which can be solved via hardware or software techniques.

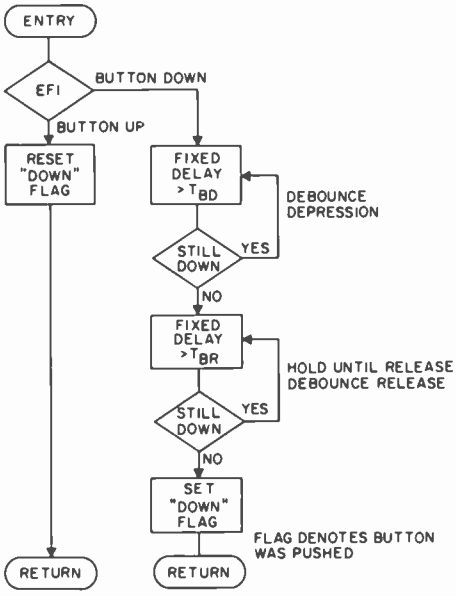


Fig. 10
Software debouncing subroutine tests to see if switch has been depressed and then later released. Method has drawbacks, though: it gives "switch down" input only upon release of switch, also wastes valuable time waiting.

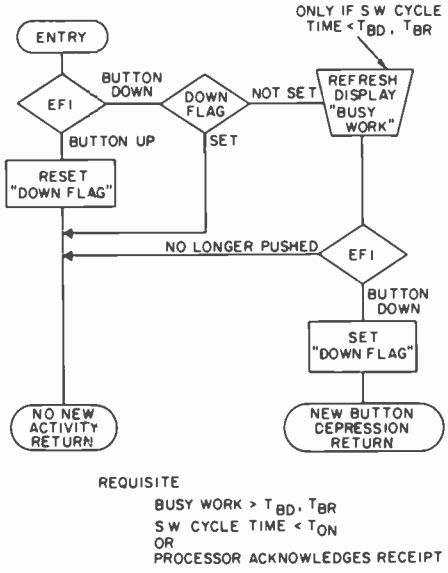


Fig. 11
Improved debouncing subroutine checks to see if switch is in same position as during previous look.

logic 0 level will be used. However, the mechanical switches tend to bounce, preventing a simplistic solution. The actual signal presented to the microprocessor will consist of three parts—an initial bounce, a stable ON period, and a release bounce. A program looking only for a simple 1 to 0 to 1 transition may sense *many* switch closures because of the bounce noise. While there are hardware solutions to this problem, software techniques may prove more cost-effective.

Fig. 10 is a flowchart of a subroutine to debounce a mechanical switch. A test is made on the input signal to test for a switch closure. If none is found, a "switch down" software flag is reset. This flag may be some convenient bit in a CPU register or a bit in a RAM status word. If the switch is down, then the software will loop, waiting for the button to be released. The wait is performed to insure that the switch is not "seen" again for the initial bounce period T_{BD}. Once the switch is released, the switch is again interrogated until it reaches a stable OFF condition. The software flag indicating a "switch down" condition is set, and the program returns to the caller. While this program is easy to understand, it is, like the earlier simple solutions, not without its problems. For instance, the processor again wastes valuable time. The execution time for this subroutine is at least $T_{BD} + T_{BR}$ and does, in fact, last as long as the button is depressed. Thus it is obviously not suitable for systems having dynamically refreshed displays. A further drawback, from the human-engineering standpoint, is that a response is made on the *release* of the switch rather than on its depression, the opposite of what one would normally expect.

Fig. 11 shows a flowchart for an improved method that overcomes both of these drawbacks. Here, the subroutine that looks at the input signal remembers what that signal was the last time it looked; it then saves this information in a software flag we will call the "down flag." The routine operates as follows. If the button is now down and was also down during the last look, then the subroutine assumes that it is seeing the same button depression seen earlier. The subroutine then returns to the caller and indicates no new activity. If the button is not now down, but was during the last look, the subroutine assumes that the switch has been released, so it resets the "down flag" and returns to the caller, again indicating no new activity. (We are assuming the processor is only interested in switch depressions and not their duration.) But, if the switch is down now and was not down during the last look, this must be a new depression. The switch must be debounced, the "down flag" set, and a message returned to the caller. Notice in the flowchart that a second test was made after the delay generated in the "busy work" block. This is a debouncing technique that assures the switch has been in the same state for two successive samples before deciding on the true state of the switch.

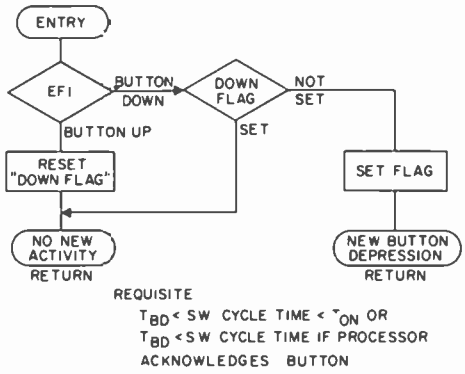


Fig. 12
Further improved debouncing subroutine has no timewasting loops. For method to work, cycle time must be greater than switch bounce time, but less than switch "on" time.

Additional constraints can lead to a better debouncing program.

This program is waiting (and therefore wasting time) during the debounce period. If some additional constraints are placed on the software cycle time, however, the program can be further optimized. If the cycle time is greater than the bounce time (T_{BD}) but less than the switch ON time (T_{ON}), then the flowchart can be simplified to Fig. 12. Here, there are no time-wasting loops, since switch bounce will effectively not be seen within the given timing restraints.

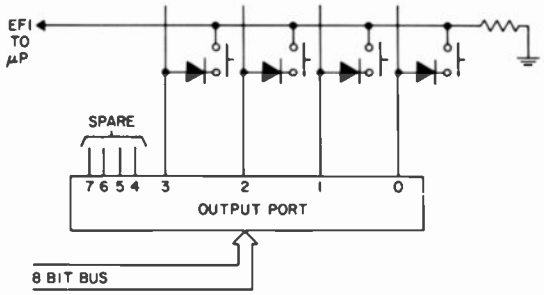


Fig. 13
Hardware for multiple-input scanning system. Fig. 14 gives software.

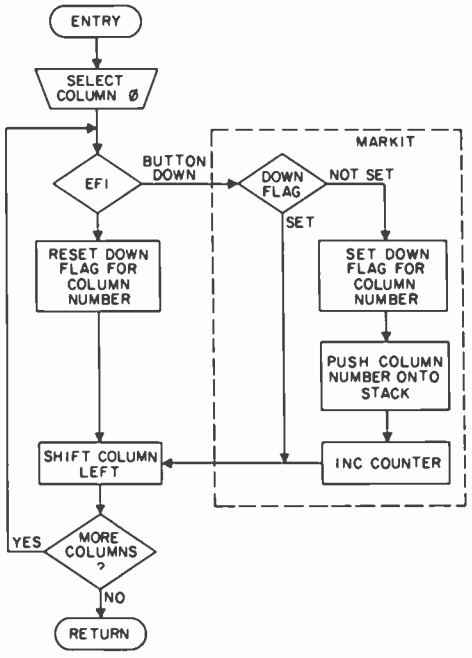


Fig. 14
Scanning subroutine looks for new switch closures and reports them to the main program by pushing the switch number onto a stack and incrementing a counter. Main program looks at counter to see if any new switch closures have occurred; if so, it gets switch number from stack and decrements counter.

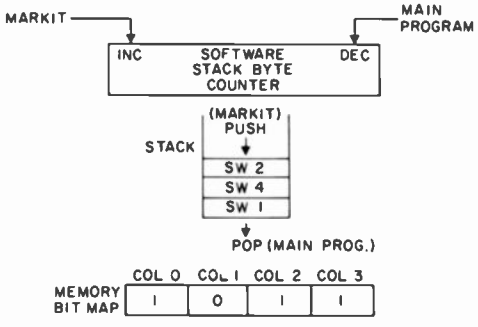


Fig. 15
Auxiliary portions of scanning subroutine. "Markit" portion of subroutine increments stack counter; main program decrements it. In example here, switches 1,4, and 2 have been depressed and so are read into memory as column inputs.

Scanning multiple inputs

A subroutine to handle four inputs by a scanning technique can be realized with the hardware of Fig. 13 and the flowchart of Fig. 14. The software looks for new switch closures and reports any to the main program by pushing the switch number onto a stack and incrementing a counter. The main program will pop switch numbers off the stack and decrement the counter whenever the count is greater than zero.

Let's take a close look at the flowchart and the auxiliary functions for the subroutine that is shown in Fig. 15. We will assume that the timing constraints of Fig. 12 are met by this subroutine also, so that Fig. 14 is an extension of the basic flowchart previously developed. Upon entry into the subroutine, the first switch column is selected by outputting a 1 in bit position 0 of the data bus and examining the switch associated with that position. If a new depression is detected, the "down flag" is set for that switch (in the memory bit map), the column number is pushed onto the stack, and the counter incremented. Next, the column is shifted and, if more columns remain to be scanned, the process is repeated. No switch closure or no new switch closure simply results in a column shift and continuation. When all columns have been scanned, the subroutine returns to the main program, which detects if any new switch closures have occurred by seeing if the counter has a value greater than zero. If so, the main program will successively get a switch number from the stack and decrement the counter until it reaches zero.

A section of the flowchart has been partitioned off and labeled "MARKIT," a common routine that can be used for an expanded keyboard scan routine discussed in the next section. Before we leave this section, notice that the approach taken above lends itself well to a multi-processor system in which one processor handles the keyboard scanning and puts key numbers in a stack accessible to other processors as well.

Keyboard scanning technique

The multiple-input techniques already described can be expanded into keyboard-reading software.

Consider the arrangement shown in Fig. 16 for scanning a 16-key matrix. It is a simple extension of the arrangement just discussed. For a microprocessor having 4 input flag lines, such as the CDP1802, the horizontal lines can go directly to the flag inputs as shown. For other microprocessors with a more limited I/O structure, those lines could be brought in through an input port, but the principle of operation remains the same.

Fig. 17 is a flowchart of the keyboard-scanning software where "MARKIT" is now responsible for handling row as well as column information. The basic interface between the main program and keyboard scan subroutine remains the same; the subroutine places new key depressions on the stack, from whence they are passed to the main program. Notice that a key number's position on the stack does not necessarily represent *when* a given key was depressed with respect to the other keys on the stack, but merely reflects the order in which the keys were scanned. Since the stack is emptied on each cycle by the main program and only *new* key depressions are entered, two key numbers on the stack tells you only that both keys were down when the scan took place. To discriminate in time between rapid key depressions, a short software cycle time is necessary. But, remember that this time must be kept *between* the constraints of T_{BR}, T_{BD}, and T_{ON}. This technique has a limitation—the software does not tell the main program when a key has been released. Thus it cannot be used in a system requiring lockout of other keys when any one key is down.

Combined display and keyboard

Our original "classical" system has now been minimized to the point that it needs only two 8-bit output ports as additional hardware.

The whole system of Fig. 1 is shown with its component blocks filled in on Fig. 18. Our original objective to minimize hardware has been realized in that only two 8-bit output ports are required in this design, besides digit drivers (not shown).

A further improvement can be made in the system by combining the keyboard scan and display multiplexing signals, as shown in Fig. 19. Here a single-byte output is used, with the upper 4 bits being BCD-coded data for the display and the lower-order 4 bits used to simultaneously select a display digit and keyboard column. This

arrangement does not reduce the parts count, but does give smaller packages if space is a consideration, and cuts down on the number of output operations and output bytes stored. A ready expansion of the system (Fig. 20) still uses only two ICs, but can scan two 16-key keyboards and an 8-digit display.

Conclusion

We have shown that the hardware needed in a microprocessor-based system can be minimized by using software to perform functions that are usually done in hardware. The lower parts count should then produce a lower system cost. Of course, software isn't free—it takes time and money to develop and debug it. However, in volume applications, the spread-out cost of software development should be less than the replaced hardware, and the high-software system should also be more reliable.

Reprint RE-23-6-6
Final manuscript received January 30, 1978.

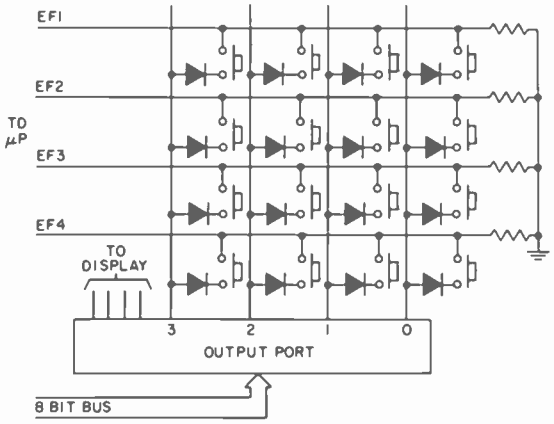


Fig. 16
Expanded scanning system for 16-key matrix.

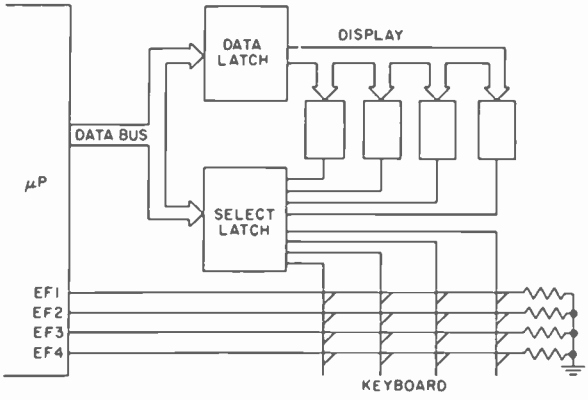


Fig. 18
Our original system with the component blocks filled. Using software has minimized the amount of hardware needed.

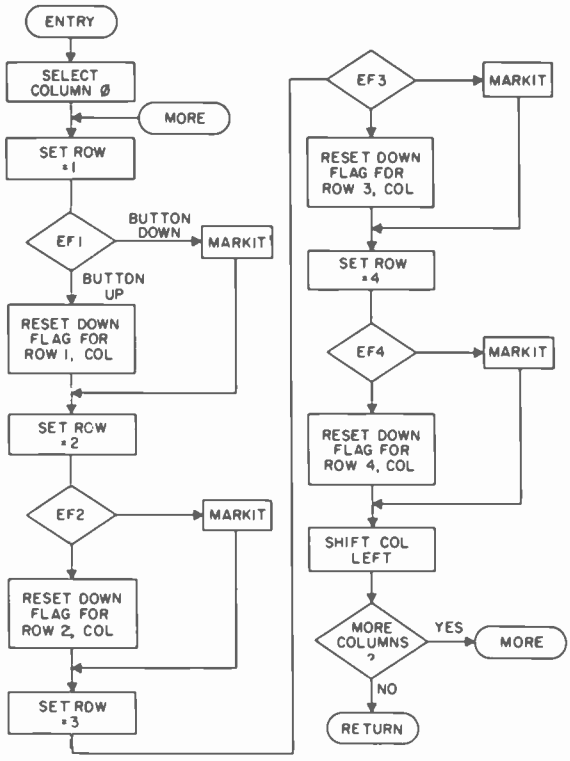


Fig. 17
Software for 16-key matrix scanning uses "Markit" subroutine for handling row as well as column information.

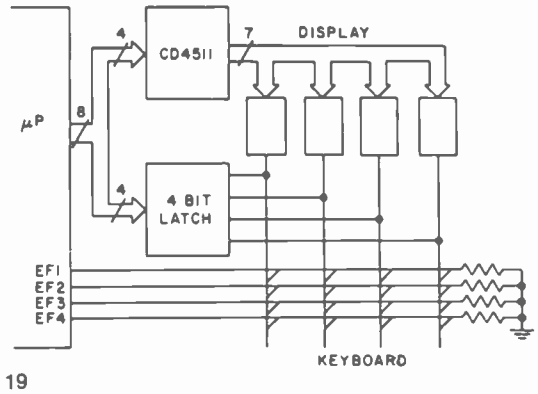


Fig. 19
Combining keyboard-scan and display-refresh signals produces smaller packages and cuts down on the number of output operations.

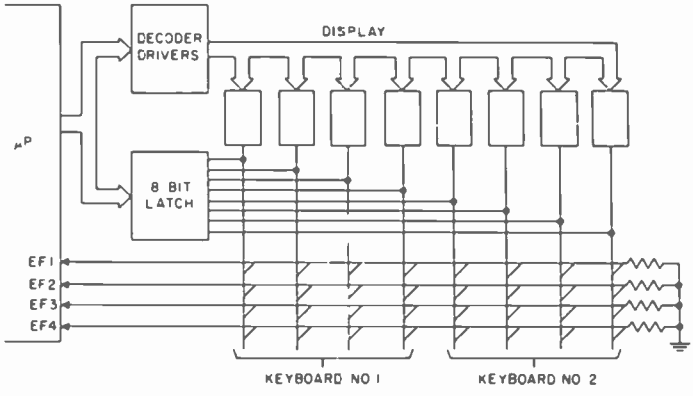


Fig. 20
Expanded system still uses only two ICs, but can scan two 16-key keyboards and an 8-digit display.

A unified approach to test-data analysis

M. A. Gianfagna

Faced with the analysis of a million nonstandard test data points per month, the Solid State Technology Center developed a highly flexible and efficient data-analysis system.

Cost-effective performance evaluation or engineering feedback from circuit-test results often require complex analyses of large volumes of non-standard data. Using a large-scale data-management system and a modular design philosophy, the Solid State Technology Center has developed a system to cope with these requirements. TDAS (Test Data Analysis System) has provided timely and economic solutions to test-data analysis problems which might have been unmanageable by other means.

Reprint RE-23-6-16
Final manuscript received January 13, 1978.

Michael Gianfagna has been involved in various aspects of test-data analysis and test-system specification since he joined RCA in 1975. He is currently doing work in the area of IC process refinement through the correlation of test data and process control data.

Contact him at:
Design Automation
Solid State Technology Center
Somerville, N.J.
Ext. 6946

Need for a workable data-base system

The development and manufacture of electronic circuits usually requires device testing for performance evaluation or engineering feedback.

Testing operations are often automated using a variety of computer-controlled test systems capable of performing hundreds of measurements in a matter of seconds. These high-speed systems generate circuit test data faster than it can be analyzed using conventional "pencil and paper" methods. Furthermore, extracting meaningful results from these measurements often requires difficult manipulations of large volumes of data, accumulated over long periods of time. The electronic testing field is also rapidly expanding, giving rise to larger and more demanding applications. Clearly, if circuit test data is to provide timely information, a computer must be used to organize the data and perform the required analyses in a cost-effective manner. Communicating this data to a computer is difficult, however, since the various test systems have virtually no standard data-log medium or format. Therefore, the problems of test-data analysis for electronic devices stem from the requirements that complex analyses be performed on large volumes of non-standard test data.

Dealing with these problems requires a highly flexible data analysis tool that is both test-system and data independent.

Flexibility accommodates a wide variety of applications both quickly and inexpensively. Test-system independence allows users of the various (non-standard) test systems to communicate their data to a common, well supported data-analysis system. Data independence implies that the internal structure used to store the test data is decoupled from the rest of the data-analysis system. This feature allows data-storage requirements to grow with the expanding needs of the user community without causing the rest of the system to become obsolete. Beyond these requirements, the system must be well documented and simple to operate, since the majority of its users will not be computer-oriented people.

What is TDAS?

TDAS (Test Data Analysis System) is a unified collection of over 100 programs and cataloged procedures that copes with the problems of test-data analysis.

By adhering to the principles of flexibility, test-system and data independence, and ease of operation, TDAS has been used successfully in a number of applications within RCA. TDAS is largely responsible for producing the test-data documentation required to certify RCA as the first manufacturer of hi-rel COS/MOS integrated circuits for aerospace and military applications.

TDAS has been used successfully in very large, long-term analyses.

To produce devices for these high-reliability applications, RCA had to demonstrate, through an extensive program of testing and documentation, conformance of its product line to the highest level of reliability standards.* To attain hi-rel qualification for a single device type, TDAS was used to validate, analyze, and produce government-format reports on over 300,000 test results. These efforts have helped to make RCA the leading supplier of COS/MOS Hi-Rel circuits in the industry, with 27 device types currently certified for production under this very strict standard.[1]

Other applications of TDAS in the high-reliability area include meeting the strict requirements of documentation and quality-control analysis for integrated circuits manufactured for the Trident missile

* Military standard MIL-M-38510, Class A.

program. The power and flexibility of TDAS have aided in the continued growth of RCA's participation in this project. Here, TDAS is used to analyze and maintain both process-related test data from integrated-circuit wafers and data generated by Reliability Verification Testing (RVT) performed on finished devices on a per shift basis.

The process-related data analysis for Trident requires measurements taken on IC wafers to be gathered on a continuing basis. In a two-week period, up to 300,000 measurements are collected and analyzed to produce over 2000 TDAS histogram reports for process evaluation purposes. (The TDAS histogram report is discussed later.) With the aid of these reports, numerous processing refinements have been accomplished. To perform RVT analysis, data is entered into TDAS 6 times per day to produce 10 tabulations, each containing approximately 18,000 test results. RVT data is accumulated in the system for one month, resulting in up to one million test results being available for individual analysis at any given time.

In addition to the routine, on-going applications, TDAS has been successfully applied to a large number of one-time, custom test-data analysis projects.

While these smaller applications do not produce the impressive high-usage statistics of the long-term projects, they represent perhaps the most important milestones of all. The majority of short-term engineering projects requiring test-data analysis can allocate only a small percentage of the available time and money to actually validating and analyzing the data; it is obviously not cost-effective to undertake a test-data analysis project which is larger and more complex than the original project that generated the test data. By taking advantage of the power and flexibility of TDAS, the data-analysis requirements of these projects may be kept well within budget constraints.

Applications of this nature include the engineering evaluation performed by the Solid State Division's Bipolar Integrated Circuits Engineering group, which often receives small engineering samples of new or experimental devices for initial testing and analysis. Many devices are tested only once. The flexibility of TDAS allows these one-time analyses to be performed without spending a disproportionate amount of resources on them.

Other one-time applications include test-data analysis for feasibility studies done under contract. One such study, performed by the Solid State Technology Center, required RCA to investigate the reliability and failure mechanisms of a logic circuit manufactured using 2 different technologies (bulk CMOS and CMOS/SOS). The preparation of the final results of this study made heavy use of the various TDAS analysis programs.[2] Other feasibility studies using TDAS include a number of contracts executed by the Solid State Division's COS/MOS Hi-Rel group on such subjects as the feasibility of producing radiation-hardened CMOS devices and the validity of high-temperature accelerated life testing.

An enhanced version of TDAS is currently being prepared for use in a high-volume, commercial product-line environment. In this application, TDAS will provide RCA engineers with daily reports of product performance and aid in process refinements through the correlation of test data and process information. This type of analysis should produce a substantial increase in yield for this product line.

The system design

The central test-data storage for TDAS is implemented using the RAMIS® data-base management system. Ramis is a software system that provides an easy-to-use language for describing a data-storage structure (or data base). Using this language, a data base was developed which efficiently stores the information generated during the device testing cycle. RAMIS also embodies a powerful, high-level programming-type language for extracting and operating on specific subsets of data from the data base. While this system is intended primarily for financial reporting

® RAMIS is a registered service mark of Mathematica, Inc.

Using TDAS in a high-reliability environment

The High Speed Bipolar Integrated Circuits (HSBIC) manufacturing line has routinely used TDAS since September 1976. Two major activities have been under control—product reliability and process monitoring. TDAS has been effective in supplying the software structure to perform these tasks, as well as the flexibility to meet the very specialized contractual requirements for product reliability.

For the product-reliability portion of the work, each month over one million test results are stored and analyzed. More than one hundred reports are generated. In addition to these reports, the database structure permits many special reports to be produced This has helped to solve specific engineering problems.

TDAS assists process monitoring by automatically displaying histograms of key electrical parameters measured on wafers during processing. Data variables can be grouped by product type, processing lot, or individual wafer using TDAS procedures. This allows either a broad or specific look at the product.

The timeliness of reports is extremely critical in this application, since TDAS analysis results are used to move product through the processing line on a 3-shift basis. Because of the extremely large volumes of data which are routinely processed in our application, special attention by the computer center has been required on occasion to ensure that our deadlines are met. This situation occurs when data-storage space or processing time is in short supply. However, in retrospect it is obvious that the production requirements of HSBIC could not have been easily met without the use of a large-scale data-management system, such as TDAS. For this reason, HSBIC will continue to use TDAS for its data-analysis needs in the future.

Robert DeMair
Test Engineering
High Speed Bipolar I.C.
Solid State Division

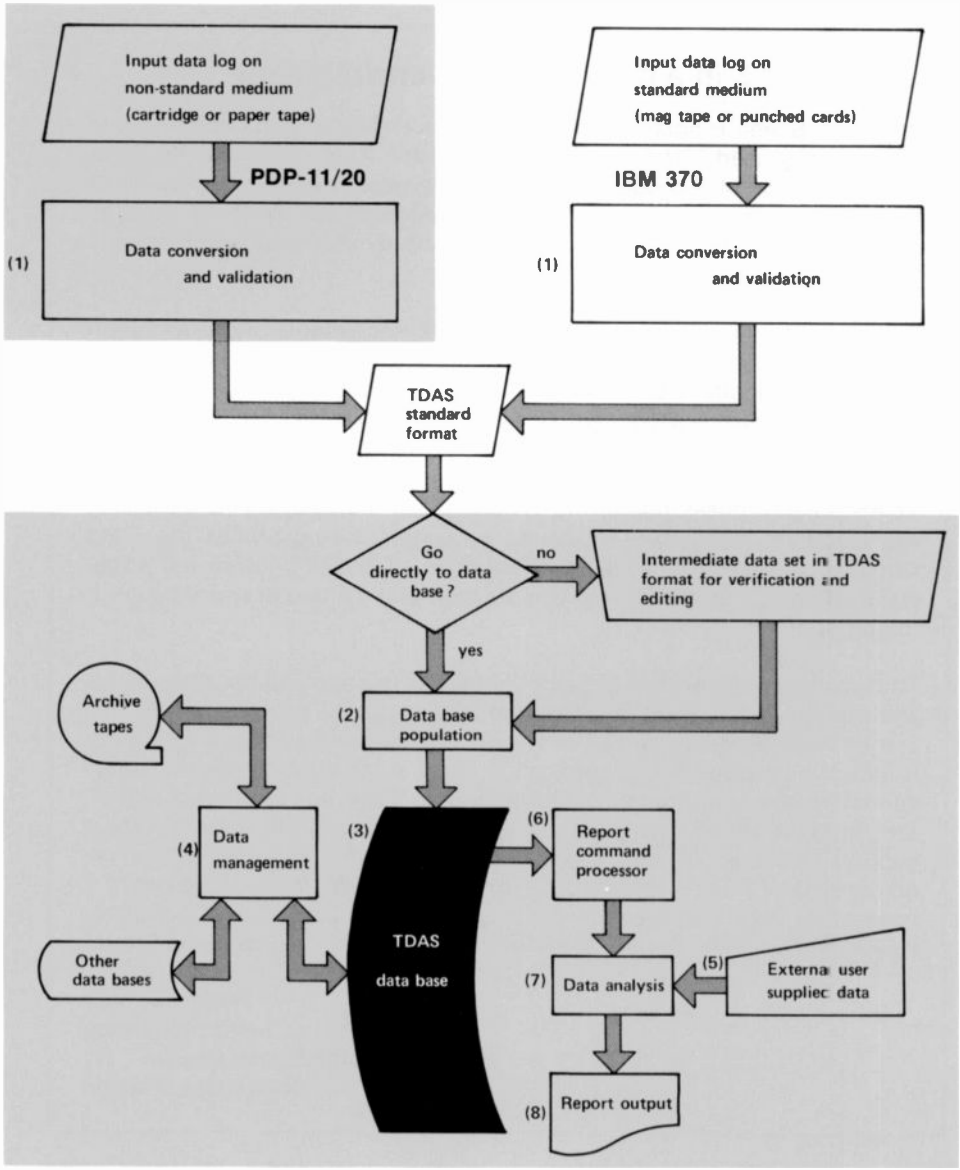


Fig. 1
TDAS outputs can be in the form of data analysis reports or inputs to other data bases or archives. System normally operates on IBM 370 computer, but uses PDP-11/20 minicomputer for nonstandard data conversion.

applications,[3] the use of a large-scale, programmable data-base management system such as RAMIS is instrumental in giving TDAS the power and flexibility needed to cope with diverse test-data analysis tasks. TDAS can best be described as a network of extremely modular operations; each operation being performed by an independent group of routines written in a variety of programming languages. This modular design philosophy has allowed TDAS to be enhanced and expanded without the (often disastrous) effects of a total system-level rewrite.

Fig. 1 illustrates the flow of data through the TDAS network. The interface between the sources of test data and TDAS is accomplished by the data-conversion and validation routines (1). [Figures within parentheses locate the subsystems on Fig. 1.] These routines verify that the incoming data contains no format or context errors and translate the information into a standard (readable) format for further processing. While the greater part of TDAS runs on the IBM 370, conversion routines, which run on an off-line PDP-11/20 minicomputer, are necessary to capture data logged on media incompatible with the IBM 370 (such as cartridge or paper tape). Note that all the operations in the lower shaded area are completely test-system independent. Therefore, interfacing a new test system to TDAS requires only the relatively simple task of writing a new data-conversion routine.

Once data-conversion and validation are accomplished, the user has the option of either entering the data directly into the TDAS data base (3), using the data-base population routines (2), or creating an intermediate dataset. The latter is useful when manual editing of the data is required. This situation can arise when data keyed in by an operator at testing time (e.g., date, unit number being tested, etc.) is incorrectly specified. This type of problem is easier to correct before the data is entered into the data base.

The TDAS data base is a hierarchical (or tree-like) structure which contains storage for all possible data items produced by the test systems it currently supports.

The data base is the only portion of TDAS built solely from the RAMIS software. Fig. 2 illustrates a typical storage configuration for test results in the TDAS data base. Note that with the hierarchical structure, frequently occurring data items need not be re-specified each time they appear in the input data. Instead, these items "point" to all other data items to which they apply (e.g., lot L501 "points" to the two wafers which comprise L501; each wafer in turn points to the chips on that wafer, etc.). This type of structure decreases the space required to store the data and increases the efficiency with which it is accessed.

Interfacing directly to the data base are the data-management routines (4) and the report-command processing routines (6). Both of these sets of routines are designed with enough flexibility (or data independence) to allow the data-base structure to be altered without making a major portion of those routines obsolete. This feature allows the data-base structure to be fine-tuned for optimum performance without affecting overall system operation. The data-management routines consist of such functions as data-base archiving and making statistical summaries of parameter test data for storage in a modified TDAS (Summary) data base.

The report-command processor accepts analysis requests issued by the user and, using the services of RAMIS, extracts the required subset of data from the data base. The data-analysis routines (7) accept the selected data along with auxiliary information not contained in the data base (5) (such as descriptive names for certain data items and display limits used by certain TDAS plotting routines). It is here that the actual reports are generated. Since the report-

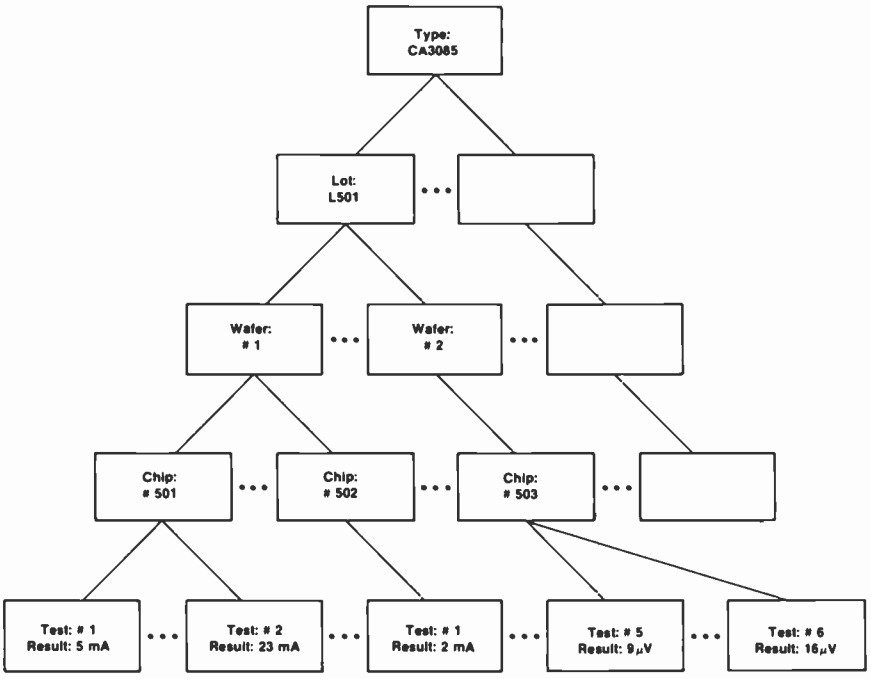


Fig. 2
Hierarchical setup of TDAS data base decreases memory space required to store data and increases efficiency of access. Each level "points" to the one below. Here, for example, wafer 1 refers to chips 501 and 502; similar storage configurations are possible with other test-data items.

generation task is so localized, it is possible to add new analysis features to TDAS without affecting any other part of the system. The final step in the report generation cycle is the output routines (8). TDAS currently produces reports on line printers, terminals, or microfiche.

The modular design of TDAS allows a wide variety of rather specific requirements to be easily accommodated within the framework of a generalized system.

For example, transistor beta measurements at 5 levels of collector current for 300 devices are logged on magnetic tape in a special binary test-system format. The data analysis requires histograms of the beta distributions at each level of collector current; the mean value of beta at each level of current must also be stored for future use. To accomplish these tasks, the data tape is first loaded onto the IBM 370 and converted to TDAS standard format using the data-conversion and validation routines. Next, the data is loaded into a TDAS data base that has been created for this particular run. Once the measurements have been loaded into the data base, two requests can be issued in parallel: one, sent to the report command processor, generates the required histograms; the other, sent to the data management routines, makes a statistical summary of the data and stores the results in another data base, which will be archived on magnetic tape for future use. This archive tape may be reloaded into an active data base for continued analysis at any time.

Features of TDAS

The following is a brief description of the major features of TDAS. For a more in-depth discussion of these and other features, refer to the "TDAS User Guide."[4]

The following test-system formats are currently supported by TDAS: Teradyne, Datatron, C-CAT, ADULT, and hand-taken data in arbitrary format on keypunched cards. The provision to input arbitrary format data has greatly aided in accommodating the non-computer-oriented testing environment, such as gathering hand-taken measurements done on the bench. This type of data is entered into the system by providing the user with a data-description language capable of defining the particular format of the keypunched data.

The data-management facilities of TDAS include such features as: data-base loading and unloading from magnetic tape to facilitate long-term data storage; the ability to create and access multiple data bases; the ability to merge the contents of several data bases; the ability to change the size of a data base; and the ability to reduce parameter test data to a compact set of statistical quantities. This last feature uses the TDAS Summary data base to store statistical information calculated from the original test data. It is extremely useful in lowering storage requirements when only the essential statistics of the data need to be maintained on-line.

The various TDAS reports allow a high degree of flexibility in the way the data is presented. The format, data content, and options used to generate the reports are all defined by a consistent, user-oriented report-command language. Thus, to drastically change the format of a particular report, one need only change the options specified in the report command. For convenience, TDAS lists the options used to generate each report in the heading. Reports currently produced are: Histogram, TABLE, Wafer Map, Bin Map, and TREND diagram reports. See Figs. 3-7 for examples of these outputs.

In addition to the above capabilities, the user may define a mathematical function which will be applied to all the data values selected before they are included in any of the reports mentioned above. By this means measurements can, for example, be converted to a decibel scale and then be analyzed in this format.

The majority of features mentioned in this section may be performed either in the interactive (time-sharing) or batch mode.

The applications

TDAS has been used in a wide variety of situations. Here are some details about a few of the more ambitious and unique applications:

CMOS Hi-Rel—The work by CMOS Hi-Rel to qualify parts for high-reliability applications has already been mentioned. The effort to attain hi-rel qualification status for new CMOS device types is an on-going project. Initial work to qualify the first RCA CMOS IC (a quad AND-OR select gate) began in 1972. At that time, the Design Automation group began experimenting with test-data analysis on its PDP-11 minicomputer. While these efforts ultimately produced valuable results,[5] they found that the PDP-11 was insufficient for managing the tremendous volumes of data

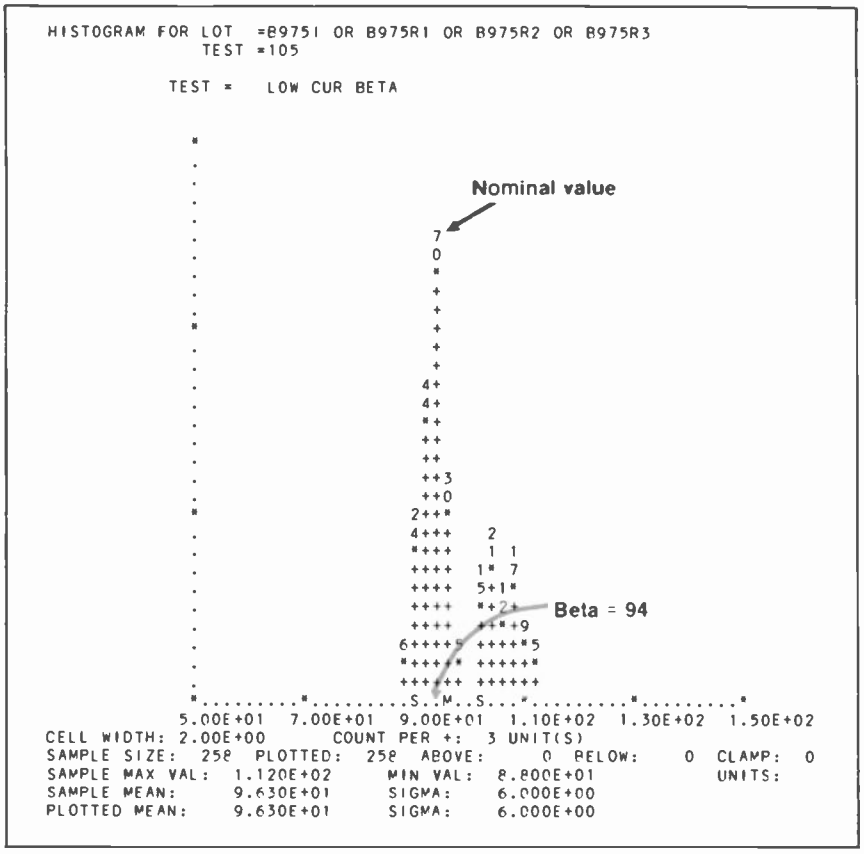


Fig. 3
The Histogram report provides a bar-chart representation of the data by listing test values along the horizontal axis, and accumulating the number of devices tested that possess a certain test value along the vertical axis. Using this technique, the distribution of values for a particular parameter may be determined. This example illustrates the distribution of low-current transistor beta for a sample of 258 devices. If the nominal value of beta for this device is 94, we can use this report to determine that 70 devices in the sample possessed the nominal value. Shifts in device performance are easily spotted using histograms with constant display limits. TDAS can plot the data within a user-specified range of values, or it can analyze the existing range to produce a histogram that optimally displays the information. It is also possible to selectively focus on a small portion of the data.

```
LOT  =B975I OR B975R1
CHIP =FROM 1 TO 20
TEST =110
STATS=COUNT OR MEAN OR SIGMA
DELTA OPTION
  ROW=CHIP
  COL=LOT
  Q  =TVAL AND RANGE AND UNITS

TEST    =  ZENER VOLTAGE

CHIP      FIRST          SECOND         SECOND
                                        -FIRST
       1  5.960E+00 V    5.960E+00 V    0.0          V
       2  5.980E+00 V    5.960E+00 V   -2.000E-02    V
       3  5.970E+00 V    5.970E+00 V    0.0          V
       4  6.010E+00 V    6.020E+00 V    1.000E-02    V
       5  6.030E+00 V    6.030E+00 V    0.0          V
       6  5.990E+00 V    5.990E+00 V    0.0          V
       7  5.950E+00 V    5.950E+00 V    0.0          V
       8                 5.960E+00 V
       9  6.020E+00 V    5.970E+00 V   -5.000E-02    V
      10  6.030E+00 V    6.030E+00 V    0.0          V
      11  6.020E+00 V    6.020E+00 V    0.0          V
      12  5.990E+00 V    5.990E+00 V    0.0          V
      13  5.960E+00 V    5.960E+00 V    0.0          V
      14  5.960E+00 V    5.960E+00 V    0.0          V
      15                 5.980E+00 V
      16  5.980E+00 V    6.000E+00 V    2.000E-02    V
      17  5.990E+00 V    5.990E+00 V    0.0          V
      18  5.970E+00 V    5.980E+00 V    1.000E-02    V
      19  6.000E+00 V    6.000E+00 V    0.0          V
      20  5.990E+00 V    5.990E+00 V    0.0          V

COUNT     1.800E+01      2.000E+01      1.800E+01
MEAN      5.990E+00 V    5.990E+00 V   -1.670E-03    V
SIGMA     2.520E-02 V    2.500E-02 V    1.420E-02    V
```

DELTA column

Fig. 4
The TABLE report produces a tabular representation of the data by listing any group of data items against any other 2 data items, which act to form the row and column headings of the table. In this example, Zener voltage is listed for chips (or devices) numbered between 1 and 20 contained in lots FIRST and SECOND. Here, chip numbers form the row headings and lot names form the column headings. A set of statistical information for each column of the report is also listed. The DELTA option of the TABLE report has been specified for this example, producing an additional column listing the arithmetic difference between the Zener voltages for devices in the two lots. This type of information is useful if a shift in device performance is of interest (e.g., performance before and after thermal shock). Either statistical summaries or raw parameter measurements may be listed using TABLE.

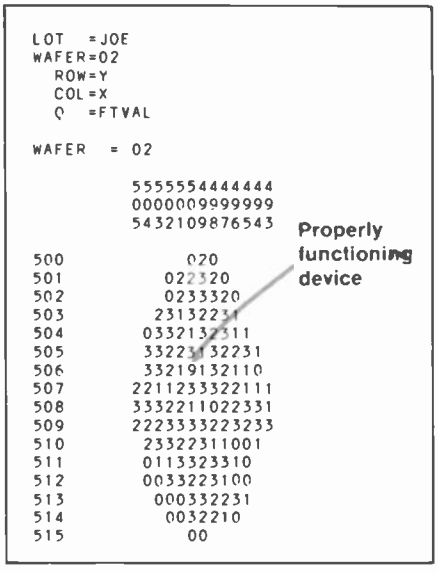


Fig. 5
For IC applications, the Wafer Map report lists device performance as a function of position on the wafer. This type of information is useful to determine the ways a representative IC fabrication process affects yield for certain areas of the wafer. A common use of the Wafer Map is to tabulate device performance grade (or bin) numbers, which are logged by many test systems as a function of position on the wafer. These bin numbers usually reflect the degree to which a particular device is functioning as designed. In this example, the *x* coordinates are listed vertically above the Wafer Map and the *y* coordinates are listed horizontally and to the left of the Map. If a bin number of 9 indicates a correctly functioning device, we can see, using the information in this report, that only the chip at coordinates *x*=500, *y*=506 was functioning properly for wafer number 2.

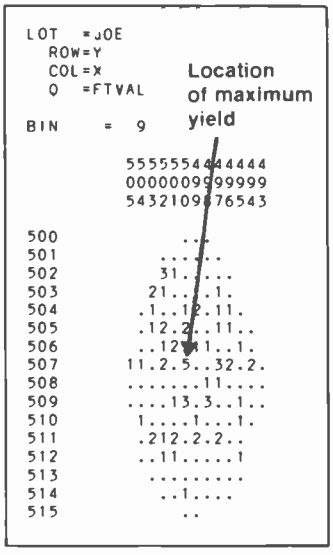


Fig. 6
The Bin Map report lists the accumulated number of occurrences of a particular performance grade (or bin) number over a group of wafers as a function of position. This type of information greatly aids in spotting trends in yield for particular areas of the wafer. These trends may be caused by mask defects or processing variations. For example, this Bin Map tabulates the number of chips with a bin number of 9 as a function of position over a group of wafers. If we know that a bin number of 9 indicates a properly functioning device, we can use the information in this Bin Map to determine that coordinates *x*=500, *y*=507 produced the highest yield, with 5 wafers containing a good device in that location.

being produced to demonstrate conformance.

Data-analysis efforts were then moved to the large-scale IBM equipment, where development of the first TDAS capabilities began. These efforts paid off late in 1975, when RCA became the first semiconductor manufacturer to qualify a CMOS integrated circuit to the highest applicable military standard. To demonstrate conformance to this standard, the TABLE report of TDAS was used to produce over 2000 pages of test-data listings of device performance under a wide variety of stresses. Once the initial bugs were out of the system, remarkable progress was made; over the 13-month period following initial qualification, RCA certified 23 device types for production under the government standard. Devices conforming to this standard often sell for over 50 times the price of their commercial equivalents.

High-Speed Bipolar IC—TDAS is used by the High Speed Bipolar group of RCA's Solid State Division primarily in conjunction with manufacturing done under contract to Lockheed for the Trident missile program. What is unique about this application is that the user has taken advantage of the modular design of TDAS to add a number of new features to the system to accommodate special contractual requirements. The result of this relatively small programming effort is a system that is performing special, custom data-analysis functions which would normally have taken months, and possibly years, to implement "from scratch."

This modified version of TDAS produces engineering reports automatically by generating its own report commands based on the content of the input data. A summary listing of each new lot of incoming data is also automatically produced for verification purposes. A procedure has been developed to automatically tag and remove from on-line storage all data which has already been reported upon. This has resulted in a substantial reduction in storage costs. In addition, a new analysis program has been implemented to gate the flow of product through the processing line.

TDAS was also used by this group to demonstrate the immunity to radiation damage of devices manufactured by the Trident production line. To accomplish this, the DELTA option of the TABLE report was used to list the shift in device performance before and after exposure to radiation. The work in this area resulted in a contract to supply radiation-hardened devices to General Electric for military applications.

Personnel Department—This application falls into the class of a one-time analysis requirement dealing with a very different kind of test data. RCA's Somerville personnel department was confronted with the task of analyzing the responses to an RCA personnel questionnaire. What would normally have been a very lengthy and tedious job was submitted to the computer center for keypunching. The hand-taken data input path of TDAS was then used to populate a TDAS data base from some 2000 keypunched cards containing the data. TDAS's Histogram report was used to produce the required statistical and distribution analyses.

Conclusions

To deal effectively with the problems of gathering and analyzing large volumes of nonstandard circuit test data requires a system that is flexible, test-system and data independent, and easy to use. Flexibility is necessary to allow the system to adapt quickly and inexpensively to the varied requirements of the user community. Test-system and data independence insure that the system will survive an expanding and changing environment. Ease of use insures that the labor costs of doing test-data analysis are reasonable.

Through the use of a large-scale data base management system and a modular design philosophy, a user-oriented system which possesses the above properties has been developed. This system is being used as an effective tool to provide timely and economic analyses in a variety of applications within the Solid State Division and Laboratories of RCA.

Acknowledgments

The author wishes to thank Vicki Fowler, Hugh Lambert, and Cherie Lewin for their work on the design and development of TDAS, and Chris Davis for his patient efforts to focus all of the good ideas into a cohesive system.

References

1. *High-Reliability Integrated Circuits*, RCA Solid State Division, Somerville, N.J., (Feb 1975).
2. Caswell, G. and Cohen, S., *A reliability study and investigation of complementary MOS/SOS integrated-circuit technology*, Final Technical Report (Jul 1976).
3. Roche, Bob; "RAMIS at Citibank," *Datamation* (Dec 1976).
4. *TDAS User Guide*, (Design Automation Publication D-3000), RCA Solid State Technology Center, Somerville, N.J., Second Edition (May 1977).
5. DeMair, R.F.; "Test data analysis for device and process characterization," *RCA Engineer*, Vol. 21, No. 2 (Aug/Sep 1975).

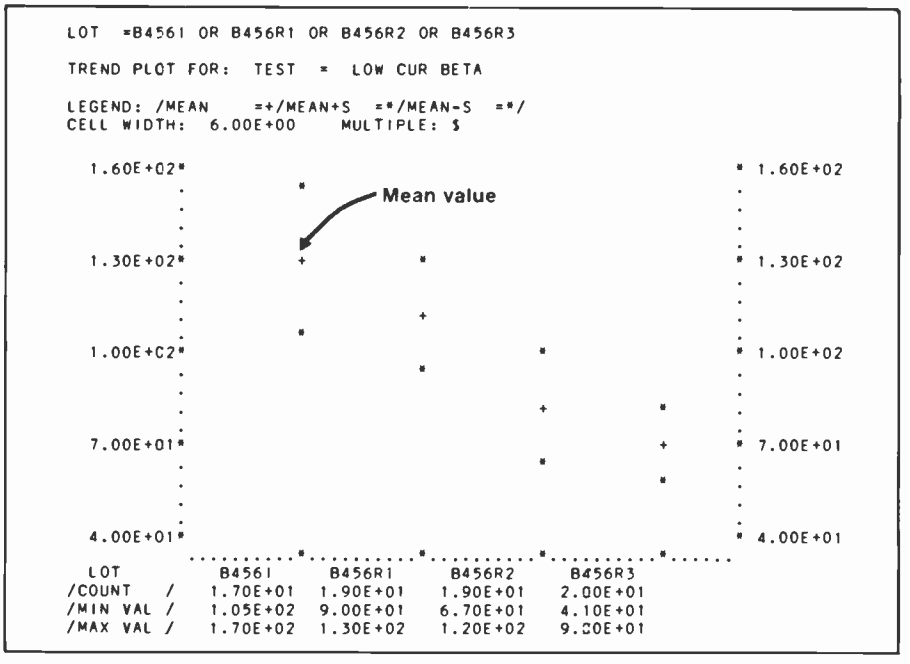


Fig. 7
The TREND report provides a point plot of a set of summary statistics. Statistical values that cannot be plotted on a common axis (because of their unit of measurement and/or magnitude) are automatically tabulated below the plot. This example plots the average value and one-sigma points for low-current beta associated with four lots. The sample size (number of devices in the lot), minimum beta, and maximum beta for each lot are also shown below the plot. Using the information in this report, we can see that the average low-current beta steadily decreased from 130 to 70 for the four lots shown. This information might be useful in isolating a processing problem, such as an incorrect base diffusion time for some of the lots shown.

Microcomputers in picture-tube manufacturing

T.F. Simpson|J.P. Wittke

Microprocessors, replacing complex hardwired systems, provide a flexible, accurate, and low-cost way to control production and testing operations.

Microcomputers are beginning to find extensive application in the production of picture tubes. The introduction of these software-controlled LSI chip computers as a replacement for dedicated, hardwired control and testing circuits not only brings about flexibility, permitting rapid changes in operations, but also eliminates operator-induced variability in the product, and can significantly increase the speed of operations. Moreover, with such systems, automatic record keeping for inventory and quality control is easily implemented.

Automatic testing system

The Accutrak picture tube emission tester, developed at the Picture Tube Division in Lancaster, is a good example of such a system. During the early '70s, the Color Applications group there was developing an automatic tester with several goals in mind:

It should be versatile enough to perform several different tests.

It should operate automatically.

It should print a record of test results.

The existing hardware-based system was adequate, but lacked flexibility.

The design was originally implemented in TTL, with one MOS arithmetic chip for calculations. As test requirements grew, including a gun-to-gun tracking test that gave the instrument its name, the unit became more and more complex. Although programmable (using TTL PROMs), the complexity of the unit (over 200 ICs) and the constantly changing nature of the test requirements led to dissatisfaction with both the hardware and the difficulty of programming the "homegrown" design. Subtle architectural deficiencies, such as not being able to use test data as part of test conditions, became serious problems.

For these reasons, alternatives were considered. For example, minicomputers were a possibility, but at the time were bulky and costly. At about this time Intel announced their 8080 microprocessor and the Intellec 8 system based on it. In short order, it became the obvious answer to the problem. The microprocessor could replace large chunks of the TTL controller, do all the calculations itself, program the power supplies, and even operate a strip printer.

The microprocessor-based tester can perform a number of electrical tests in sequence.

A block diagram of the unit is shown in Fig. 1. Basically, it must set up several test voltage conditions for the picture tube, such as heater voltage, grid 1 bias, and the voltages on grids 2 and 3, and measure the resulting effects on some parameter such as cathode current. The tester must often vary one or more potentials in search of a condition which sets up a predefined condition, such as "cutoff." This is complicated by the nonlinear relationships between the voltages and currents. Typical test sequences determine and set cutoff, measure maximum cathode current by shorting the grid 1 to the cathode, and record both values for each of the three electron guns in a color picture tube.

The hardware changes were relatively easy.

In switching over to the microcomputer approach, the hardware design consisted of interfacing the microcomputer to the already existing relays, analog-to-digital converters (ADCs), and digital-to-analog converters (DACs) that control the programmable power supplies. The control was via input and output ports of 8 bits each on commercial I/O cards that simply plugged in to the mother board. By using such assembled, pretested cards, much engineering time and grief was saved. In fact, the only hardware surprise in the whole transition was a very late realization that Intel ports invert!

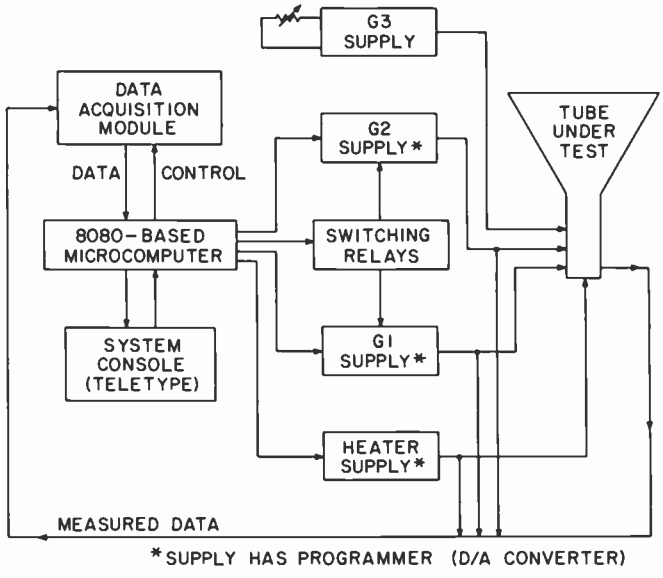


Fig. 1
Accutrak tester can change test-voltage inputs to two of the tube's three grids, as well as change heater voltage, to obtain test results on output parameters such as "cutoff" and maximum cathode current. System tests each of the three electron guns in color tubes.

The tester software

The software, however, held many surprises—some pleasant, some not so pleasant. For example, early in the debugging stages, sensible data could not be obtained from the test of the maximum cathode current. Everything seemed to be operating correctly, but the computer didn't "see" the data. The problem, it turned out, was in not programming a delay between the relay closure and the data measurement—the microcomputer was so fast it was trying to read data before the relay had closed!

The main effort in getting the system operating satisfactorily was in developing appropriate software.

For this application, where changes in the tests to be run can be frequent, it was important to use a system with software as "obvious" and self-documenting as possible. Thus, the choice of software language was very important; the closer the programming language was to the task at hand, the easier the programming. The initial choice was PL/M,™ a high-level language developed by Intel. It is a procedure-oriented, block-structured language that is a subset of the more powerful PL/1. It allows you to use long, descriptive variable names, like "GRID ONE" or "RED CATHODE," a real aid in documentation. Its statements are formula-like, and it has built-in 16-bit multiplication and division routines.

Cross-compiler vs. interpreter.

A PL/M cross-compiler was installed on the Univac VS/9 computer in Cherry Hill, and arranged to punch object code tapes on a Teletype terminal in Lancaster for loading into the microcomputer. This got the system "on the air," but had its drawbacks. We soon found that the microcomputer, in combination with a data acquisition module, can be a valuable "window" into the system that can be a real help in exploring and debugging the system. But to be of maximum value, one needs the capability of modifying and rerunning the program right from the system console. This was not possible with the cross-compiler.

The solution was to write an interpreter for a simplified subset of BASIC. This interpreter can reside in the microcomputer's memory. It can both edit and run programs written in the easily learned BASIC-like language. This new system really speeded up system development: devices under computer control were programmed through their paces and checked out completely from the Teletype terminal with a short test program, with no separate translation step needed. Yet programs could still be written with the simplicity and power of a high-level language that keeps the busy-work to a minimum.

As the system (and our thinking) developed, a few hidden benefits in the microcomputer-based system were discovered.

Since the Teletype was needed for system programming anyway, the numerical strip printer that had been used for data logging was no longer required. And with the Teletype's alphanumeric printing capabilities, data can be labeled, annotated, and made self-explanatory and self-documenting—a real time-saver when looking at six-month-old data. Now, an operator who knows nothing about the nature of the test need only enter the tube serial number and a test-type number, and the machine takes over test selection and performance. Since the machine completely takes over, the operator can, in effect, do two jobs at once, a great productivity improvement. With a high-level language available to program and run the system interactively, it is possible to respond rapidly to new test requirements and to make modifications to old tests. And since the language is simple and has built-in mnemonics, technicians can learn to program the machine in a reasonable length of time.

The resulting instrument has fulfilled the initial goals satisfactorily. Three of these units are in daily use testing engineering and manufacturing samples. (See Fig. 2.) A "menu" of five different tests is available for selection by the operator at any time. By using properly titled and annotated test results with carbon paper in the Teletype, complete test results can be both delivered to the test engineer and filed for reference at a later time. In addition, the unit measures and prints most critical test parameter voltages as part of the report, serving as a self-check on the validity of the test.

Fig. 2
Three such picture-tube testers are in daily use at Picture Tube Division plants, testing engineering and manufacturing samples.

As a final software step, a source-code-compatible compiler generates compact, fast-executing object code for the more permanent tests. This code is stored in ROM for permanence during power outages, while freeing up more RAM for experimental programs.

Laser welding system

The emission tester, as mentioned, is in routine use. We now discuss a microcomputer-controlled system that is still in the developmental stage: an automated laser welder for picture tube gun parts. The nearly one hundred welds found in modern picture tube mounts are presently made with conventional resistive welding techniques. However, high-power laser welding can offer important advantages for many of these welds, in higher, more uniform weld quality and in greater welding rates and lower costs. In such a high-speed, automated laser welder, a microcomputer provides the required rapid control signals in a versatile, flexible way. In this system under development, it is used to control the flow of parts under the laser beam as well as the laser firing times and durations.

The laser is a large carbon dioxide one, capable of either continuous or pulsed operation. A numerically-controlled x-y table moves the parts to be welded under the focused laser beam. The laser optics provide a small region where the optical power is high enough to melt even refractory materials such as tungsten, and can vaporize a wide variety of materials—metals, ceramics, glass, etc. There are several advantages in using laser processing. The energy is deposited on the target without mechanical contact that can deform the parts, and it can be focused onto hard-to-reach areas. The heat is delivered rapidly to a very localized area, and thus adjacent regions are not thermally stressed. Laser welding is therefore useful for joining parts of picture tube guns, such as tungsten heater wires to stainless steel

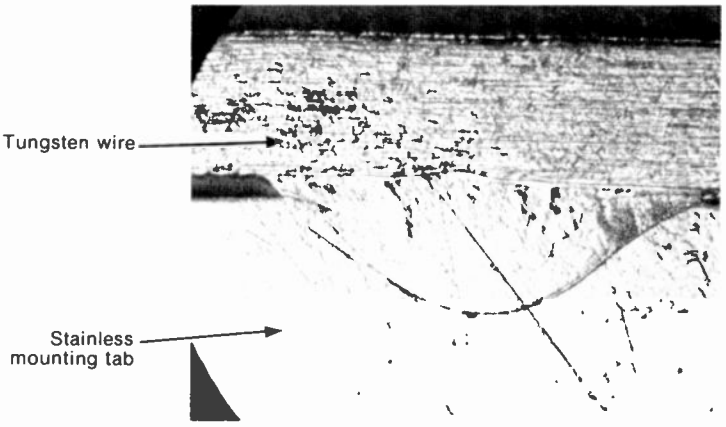


Fig. 3
Laser weld (360x cross section) of a tungsten heater wire to a stainless-steel mounting tab.

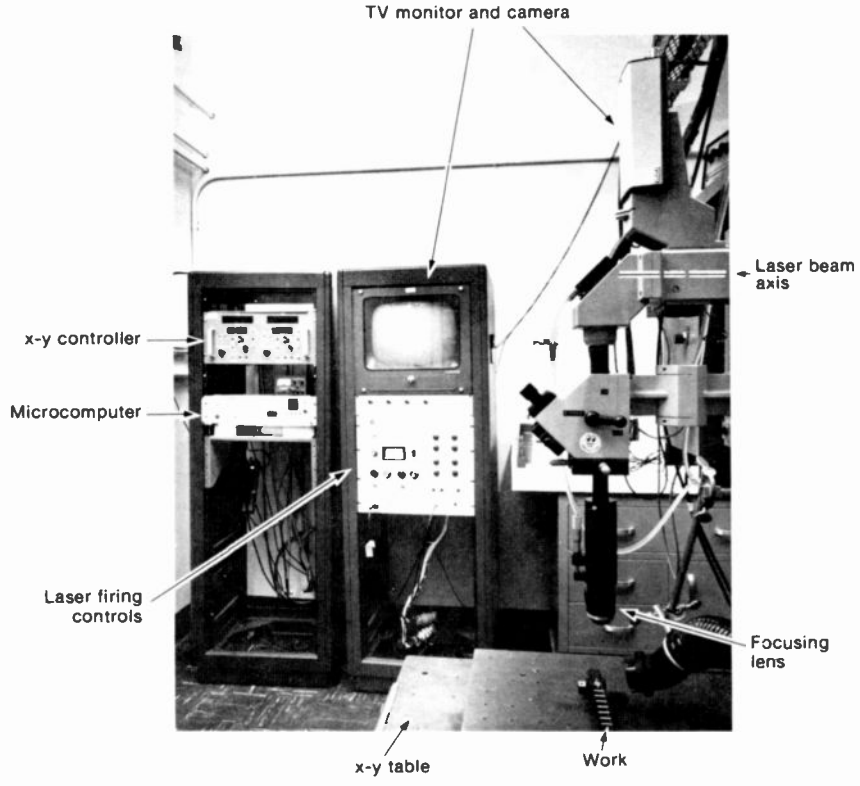


Fig. 4
Laser welding facility uses microcomputer to control time and duration of each laser pulse. Closed-circuit tv system monitors the welding operation through the laser optics.

contact tabs, grids to grid supports, and feedthrough wires to other tube elements. Fig. 3 is a microphotograph of a section through a typical laser weld, and Fig. 4 shows our welding system.

The welding system operates with very little information feedback to the microcomputer.

The computer sends a 4-byte output signal to the x-y table controller, specifying the desired location of the work table. (This controller, which is supplied by the vendor of the x-y table, uses CMOS logic.)

Positions can be specified in either an absolute or incremental mode. Other control words, requiring additional microcomputer output ports, set the direction of the table motion, its speed, and the signal for table motion to begin. The only signal that the microprocessor requires from the system is a flag indicating when the table has reached its desired position. The laser can then be fired, forming the weld.

At present, only the time and duration of the laser firing are under microcomputer control; the laser power level is set by a potentiometer on the laser control panel. However, this function should be soon under computer control as well.

The system uses RCA's COSMAC 1802 microprocessor, and is based on the COSMAC Evaluation Kit.

In many ways, this is an excellent choice of microcomputer. However, factory-built I/O cards are not available for the COSMAC system, so all the necessary interface circuits had to be custom designed and built. This may be a good way to learn about microcomputer hardware, but it has not proved to be a quick way to get a system going!

The welder software

As mentioned, only one flag line is used as input, to indicate that the x-y table is in its desired position. However, the number of necessary output lines can not readily be accommodated by the seven COSMAC output commands. Therefore, a simple two-level output command sequence is used to transfer position data to the table controller, specify absolute or incremental positions, control speed, and open and close the laser shutter. Other commands are implemented via the basic (one-level) output commands. For convenience, the COSMAC Q-line is used to turn the laser on and off. The laser firing, and some table motions, entail timing loops. These can be implemented either in hardware or in software. At present, all timing is done with programmed delay loops in software.

The system had some surprises in store for us when it was turned on, of the usual, unpleasant variety. Despite apparently proper interfaces and what seemed to be an error-free microprocessor program, the table wouldn't behave properly. It would either just sit there (or perhaps oscillate) when told to move, or would dash off in some unexpected direction unrelated to its programmed path. Interestingly enough, the cause was finally traced to the same source as that of the emission-tester problem mentioned earlier; the microprocessor was *too fast*. It was telling the table to move, and then, before the controller and drive motors could actually get the table going, was asking the table if it

had reached its desired target point. Since the flag from the *previous* move was often still set at this time, the one bit of feedback information being requested of the system was often wrong! As before, judicious insertion of programmed delays eliminated the problem.

The software requirements for the laser welding facility are rather different than for the emission tester.

With the tester, frequent program changes, associated with changes in the desired testing schedule, were the rule. With the laser welder, only a relatively few programs are needed to provide all the versatility required. One program, for example, moves the parts to be welded successively into position at the laser focus, and makes the welds as each part stops in the proper place. However, to speed things up, where welding parameters are not too critical, the parts can be placed in a line and scanned past the focus position while the laser fires "on the fly." This "skeet-shooting" mode of operation gives, of course, target motion during the welding laser pulse, and can result in an egg-shaped weld region. But since the table is not stopped and started up again for each weld, the time needed for a given number of welds can be significantly reduced. Indeed, for some seam welds, one may want to have a continuous burst of laser power during the whole time the part is under the focal point. A second microcomputer program provides for this mode of operation.

A third program lets cutting (or welding) operations be carried out along arbitrary contours on the work piece. Now the computer is programmed to move the table, at constant speed, along a specified curve, while controlling the laser firing in an appropriate way. In addition to these modes of operation, the work may consist of a pattern of (perhaps different) welds to be performed on one workpiece, with the same pattern to be repeated on an array of similar pieces. Such a "step-and-repeat" mode of operation has also been programmed into the system.

Assembly language was adequate for the relatively small program set.

Because this handful of programs seems likely to provide all the flexibility needed for foreseeable jobs, the need for a high-level programming language was not as great as with the emission tester; the assembly language available with COSMAC was adequate. The programs, written in this assembly language, can be fast and efficient. They are stored in cassettes on magnetic tape, and are read into the microcomputer before running. PROMs would provide an alternative non-volatile storage medium. Of course, as welds are made on different gun parts, the array of components on the work table will change and the data table of desired weld locations must be changed, even though the same control program may be used.

Although the system hardware is still being completed, preliminary tests have been run using a COSMAC Development System with partial interfacing. It has welded a series of grids to their supports, the assemblies being mounted on a line on a jig plate. Two passes of the parts under the laser beam, one in each direction, were required.

Conclusions

We have seen that microprocessors can provide a very flexible way to control production and testing operations. They can replace complex hardwired systems that can only be altered with difficulty, and do the job with a speed and accuracy far beyond that attainable with direct human control. Use of large, pre-tested blocks of hardware where possible can greatly shorten system development time.

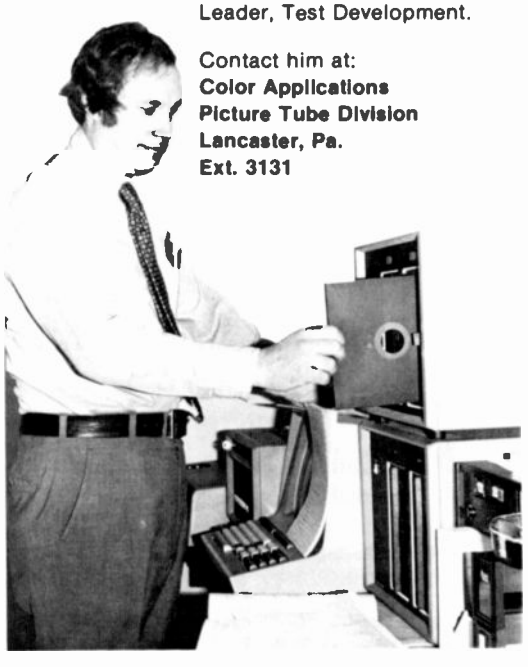
Acknowledgements

The authors had a great deal of help in setting up the systems described in this paper. In Lancaster, T. Boyer, G. Good, and P. Klugh had a big role in the work, while in Princeton, W. Mitchell, J. Csercsevits, J. Knox, and D. Fairbanks provided much help.

Reprint RE-23-6-1
Final manuscript received January 16, 1978

Ted Simpson joined the Color Applications Department of PTD in 1965. His work has ranged from investigating new color systems to automating lab and factory test systems. He was recently designated Leader, Test Development.

Contact him at:
Color Applications
Picture Tube Division
Lancaster, Pa.
Ext. 3131



Jim Wittke has worked in many areas of electro-optics since coming to RCA in 1955. His earlier research was on masers, laser spectroscopy, GaAs luminescence, injection lasers, and optical fiber communications. His current work, on materials processing using high-powered lasers, includes the use of microprocessors.

Contact him at:
Optical Electronics and
Process Control Research
Manufacturing Systems and
Technology Laboratory
RCA Laboratories
Princeton, N.J.
Ext. 3261

Using PRICE S to estimate software costs

F. R. Freiman

This computer model uses an organization's past experience to estimate software development costs rapidly.

The PRICE Software Model (PRICE S) is designed to estimate the development cost of a wide variety of computer software projects and applications. With minimal information, PRICE S can determine the software costs for business, communication, telemetry, fire control, and many other types of systems. PRICE S eliminates the problem of differences between programming languages. Moreover, the model's universality provides ways for tailoring its methods and regressions to fit the varying skills, experiences, and costs of specific projects and organizations.

Frank Freiman, as Director, PRICE Systems, is responsible for developing and using parametric modeling systems for engineering and management planning and decision-making purposes. He invented the basic PRICE methodology and the PRICE software model described here.

Contact him at:
PRICE Systems
Government Systems Division
Cherry Hill, N.J.
Ext. 5212

PRICE S is an interactive computer model which is accessed on a time-shared computer via standard telephone lines from local terminals. Within a few minutes after the input of several principal project descriptors, the model will output the computed costs for each of three overlapping development phases: engineering design, implementation, and test and integration. Fig. 1 is an example of this output.

Typical schedules are computed for the size, type, and difficulty of the project described. The user may, if he wishes, specify alternative schedules. Input schedules are examined internally, and costs are adjusted to account for apparent accelerations, stretch-outs and phase-transition inefficiencies.

Three modes of operation are available: normal, ECIRP, and design-to-cost. Normal operation computes cost directly from user inputs. The ECIRP (PRICE backwards) mode enables PRICE S to be run "in reverse" to calculate PRICE S empirical factors from known project costs—a feature extremely useful for model calibration. The design-to-cost mode uses specified costs to compute typical program sizes and project schedules. This mode permits PRICE S to investigate feasibilities and set scope-of-work goals for design-to-cost efforts.

Effective use of PRICE S requires professional instruction—authorized users are required to attend a four-day training program. In addition to covering operation and control of the time-shared computer using a local terminal, the program explains the organization and computative procedures of the PRICE S model so users can understand how costs, schedules and inefficiency penalties are calculated. The preparation and structuring of PRICE S inputs, methods of examining the reasonableness of the scope of work descriptors, and procedures for completing input data sets when all necessary information may not be known or available, are covered, as are calibration techniques for customizing the model to reflect an organization's methods, experiences, skills, costs, and other appropriate factors. Students learn the sensitivity techniques to rapidly and economically examine ranges of uncertain input information and work on several life-like software cost problems designed to test their proficiencies.

How does PRICE S work?

The underlying principle of PRICE S is that all estimates involve comparative evaluation of new requirements in light of analogous histories. PRICE methodology provides a convenient way of reducing empirical data to a few parameters of principal variables, each of which can be simply adjusted to account for technological and economic differences between individual projects and organizations.

What are the input variables?

Because the interactive procedure of PRICE S permits immediate sensitivity analysis, more than forty alternative conditions or uncertainties of various input data can be quickly assessed. The principal PRICE S inputs fall into the following seven categories:

Project magnitude (How big?)

Program application (What character?)

Level of new design (What exists now?)

Resources (Who will do the work?)

Project difficulty (What time is needed?)

Project specifications and reliability (Where and how used?)

Utilization (What hardware constraints?)

Reprint RE-23-6-23
Final manuscript received February 8, 1978.

```
SAMPLE CASE                                                    MOBILE RADAR

                              INPUT DATA
FILENAME: SAMPLE                                          DATED: 07/22/77

DESCRIPTORS
  INSTRUCTIONS    36000       APPLICATION    0.0          RESOURCE      3.500
  FUNCTIONS           0       STRUCTURE      0.0          LEVEL         2.600
                                                          INTEGRATION   0.500

APPLICATION CATEGORIES          NEW DEVELOPMENT           SYSTEM CONFIGURATION
                  MIX           DESIGN        CODE        TYPES       QUANTITY
  DATA S/R        0.0           1.00          1.00          0             0
  ONLINE COMM     0.08          1.00          1.00          1             1
  REALTIME C&C    0.08          1.00          1.00          2             2
  INTERACTIVE     0.23          1.00          1.00          1             2
  MATHEMATICAL    0.28          0.50          0.70        ***           ***
  STRING MANIP    0.26          1.00          1.00        ***           ***
  OPR SYSTEMS     0.07          1.00          1.00        ***           ***

SCHEDULE
  COMPLEXITY      1.250
  DESIGN START    OCT 77        IMPL START    JUL 78      T&I START     DEC 78
  DESIGN END      DEC 78        IMPL END      AUG 79      T&I END       JUL 80

SUPPLEMENTAL INFORMATION
  YEAR            1977          ESCALATION    0.0         TECH IMP      1.00
  MULTIPLIER      1.000         PLATFORM      1.4         UTILIZATION   0.80

                             PROGRAM COSTS

COST ELEMENTS                   DESIGN          IMPL        T & I         TOTAL
  SYSTEMS ENGINEERING             392.           16.         290.          698.
  PROGRAMMING                      51.           76.         119.          246.
  CONFIGURATION CONTROL            90.           23.         179.          292.
  DOCUMENTATION                    66.            7.          72.          145.
  PROGRAM MANAGEMENT               37.            7.          36.           80.
     TOTAL                        636.          129.         695.         1461.

                           ADDITIONAL DATA

DESCRIPTORS
  INSTRUCTIONS    36000       APPLICATION    5.299        RESOURCE      3.500
  FUNCTIONS         400       STRUCTURE      4.961        LEVEL         2.600

SCHEDULE
  COMPLEXITY      1.250
  DESIGN START    OCT 77        IMPL START    JUL 78      T&I START     DEC 78
  DESIGN END      DEC 78        IMPL END      AUG 79      T&I END       JUL 80

                           SCHEDULE GRAPH

OCT 77                                                                 JUL 80
************ DESIGN ************
                  ********** IMPLEMENT **********
                               ************ TEST & INTEGRATE ***************
```

Fig. 1
Typical PRICE S output lists the inputs (number and type of instructions, application, resources available, amount of new design, etc.), along with the proposed schedule at top. Bottom half of output gives the program cost and graphs the program schedule for the three development phases—design, implementation, and test and integration.

The universal concept of PRICE S was attained by the careful structuring of these required inputs. Its inputs were designed to permit their rapid calibration and orientation. Most organizations having computer software development experience will find familiar terms and descriptors. The values or parameters associated with the variables are relatively simple to determine. PRICE S input data is less extensive than the information usually needed to estimate software costs by other methods. Moreover, PRICE S has the capability of testing the credibility and consistency of input data to minimize problems of faulty information.

The following is a more detailed description of the principal input groups.

How big is the project?

The size or amount of work to be done is described by the number of executable machine instructions or assembly-level code. Using the number of assembly-level code or machine-oriented language instructions in lieu of source code or higher-order language statements avoids the problem of language differences. Of the many higher-order languages available, FORTRAN, COBOL, BASIC, and APL are among the most popular types. PRICE S is a universal model, capable of accommodating all programming methods. Factors are provided to convert language statements to executable code. The conversion factors will vary from organization to organization because of differing programming and compiler efficiencies.

In situations when the number of instructions is not known, PRICE S can compute the amount from other information, such as a functional flowchart. PRICE S has built-in cross-checks that measure the reasonableness of the input instruction amount. Instruction information which may be faulty results in warning messages that are printed prior to processing.

What's the end application?

This defines the type of project, such as MIS, command and control, telemetry, communication, etc. The almost infinite variety of software programs ranges from business systems through vastly complex space communications projects. The organization, associated peripheral equip-

ment, and real-time aspects of these programs vary. The variations have significant software-development cost effects because they require different resources, skills, and project schedule times. Therefore, the description of the program type is a significant parametric input.

PRICE S gives a numerical "application" value to all types of software programs. It was determined that every software project type can be identified by its mix of instructions. There are basically seven different types of instructions; these range from mathematical to operating systems and interactive code. PRICE S computes an application value from an indicated mix of instruction types. Application values can range from 1 to 11—MIS projects normally have application values of about 2 to 3, whereas real-time telemetry projects have values of about 6 to 7. Whenever PRICE S calculates an application value from an instruction-mix input, the model cross-checks the credibility of the data.

How much of the design will be new?

Software projects may not require a totally new design effort. Some of the required subroutines, algorithms, procedures, peripheral-device control instructions, etc., may be available and so lower development costs. PRICE S addresses this requirement by allowing the user to specify the amount of new design.

New design levels may be assigned to a total project or to specific instruction types. In some cases, existing designs may not mean that the application code is available. PRICE S procedures provide a separate indication for the level of code inventory.

PRICE S considers the effort necessary to integrate the available design with the new design. For example, the final test and integration of the software program will involve the total system, whether or not some design or code may have existed before the work was started.

Who will do the work?

One of the most significant cost-estimating considerations concerns the effect of the capability, experience, and talents of the activity that will be doing the work. In addition, such skills have associated labor and overhead costs. The "resource" variable of PRICE S was designed to reflect, compositely, all such factors. Resource values may range from 2.5 to 5.0, with an average of 3.5. Higher values indicate more costly activities.

Resource values are determined empirically. A PRICE S calibration procedure lets the user describe an experienced software effort and so produces a calculated resource value that can be used to apply the same (or essentially the same) performing organization to the project being measured or costed. This mode is called ECIRP, which is the ability to run PRICE backwards for calibration against actual costs and schedules of completed projects. PRICE S procedures provide adjustments to the resource value to reflect changing technologies and variations in labor and overhead rates or efficiencies.

What about complexity and time?

This factor establishes the scheduled time, in calendar months, required to complete the job with respect to the organization resources, program application, and project size. The PRICE S variable "complexity" establishes the relative degree of engineering difficulty. A value of 1.0 is average. Values less than 1.0 indicate more simple and routine efforts. They also may mean that there has been some prior experience in doing the same or comparable software design. Parameters greater than 1.0 imply that the effort is more demanding and perhaps new to the organization doing the work. As complexity values approach 2.0, the efforts are pushing the state of the art.

Complexity values are directly relatable to schedule time; they are vital because project costs are very sensitive to performance schedules. Greater complexity values mean more calendar time. Whenever one of the two values—complexity or schedule—is entered into PRICE S, the correlated value of the other is computed. To permit the user to indicate an accelerated or protracted project to PRICE S, both the directed schedule and appropriate complexity value would be input. PRICE S generates a schedule based on the complexity factor, and compares the computed schedule with the input schedule to ascertain appropriate cost penalties. PRICE S can print out the schedule-variation penalties upon command.

Where and how will the software be used?

Software programs have many and varied uses. Some are used for business-management purposes, others are involved in highly complex real-time applications such as fire control, communications, and telemetry. They may be also used aboard a manned spacecraft or in weapons systems aboard aircraft. In some cases, software programs are connected with life-support systems. The program's usage can significantly affect the software development cost. As conditions become more demanding or sophisticated, they will require more reliability testing, configuration control, documentation, and more time to complete the job.

The variable called "platform" in PRICE S indicates the varied usage requirements in terms of specifications and reliability. It plays a major role in the computation of costs, indirect cost ratios, and performance schedules.

The platform variable is also used to consider the transportability requirements of a software project. There may be instances when it is necessary to develop and test a project on one computer system, knowing that the program will be installed on other processors. Because of operating-system differences and other possible variations, the basic program may have to be organized and documented to permit less troublesome tailoring.

What are the hardware constraints?

The effect of load conditions of the processor relative to its speed (operations per second) and memory (for example, is overlaying necessary?) determine the "utilization" value. Compiler inefficiencies may be a problem.

The effort necessary to "fit" a software program into a computer processor that is somewhat limited can be significant. Programs may require severe timing control in terms of operations per second. Core memory may necessitate effective management to assure adequacy for the time requirements. Virtual memories may have timing problems. Inefficient compilers may load excessive machine code that may affect the program's timing and capacity. Because of these problems, the development of a software effort can become very costly.

Utilization values usually vary with program end usages. For example, shipborne or mobile applications systems normally have utilization values of about 0.7, airborne applications about 0.8, and space

systems about 0.9. Utilization values less than 0.5 have no effect. However, a value of 0.95 (suggesting 95% utilization) can increase costs several hundred percent.

What about economic factors?

PRICE S necessarily considers the impact of economic conditions by using two economic factors within the model. One factor establishes the fundamental or baseline cost based on the year that was input to the model. The second factor applies to the escalation rate appropriate to the time during which the work is to be done. For example, the user input of 1970 has the PRICE S model establish its base cost (as a function of the resource variable) as of January 1, 1970. (Note: All resource values are normalized January 1, 1976 rates.)

SENSITIVITY DATA

RESOURCE	COMPLEXITY 1.150	COMPLEXITY 1.250	COMPLEXITY 1.350
3.400	COST 1442. MONTHS 33.0	COST 1406. MONTHS 33.0	COST 1397. MONTHS 33.0
3.500	COST 1497. MONTHS 33.0	COST 1461. MONTHS 33.0	COST 1453. MONTHS 33.0
3.600	COST 1552. MONTHS 33.0	COST 1515. MONTHS 33.0	COST 1509. MONTHS 33.0

Fig. 2
Sensitivity matrix saves extra runs by automatically varying resource and complexity values about their initial inputs.

To de-escalate or escalate the resource rate, PRICE S uses a built-in economic table of yearly economic change rates covering the years 1946 to 2001. Adjustments based on these rates are automatically processed to reflect the input year. Users may replace any or all of the escalation rates built into the model with values of their own choosing. For example, one set of rates may be used for government projects and a different set for commercial programs.

Does the model account for technological changes?

The skill and science associated with the development of software projects are continuously being improved through experience, as are programming methods and compilers. As a result, development productivity is increasing. Ignoring economic factors, the average cost per instruction is improving. The model automatically sets its technology base to the reference year defined by the user.

The improvement trend varies as a function of time, resources, and project type. PRICE S accounts for the character of the productivity change as a result of such factors. The control variable used to modify the relative rate of productivity change subsequent to the reference year is called "technology improvement."

What about the time for system integration and testing?

Many large software-development situations involve the merging of two or more related software projects into a single unified operational system. The individual projects often have widely varying characteristics, and they may even be performed by different development organizations or companies. They may also be installed on separate computers. Examples include report generators, operating systems, system simulators, preprocessors, and peripheral devices designed to support the baseline software.

Resources and time are required to accomplish total system integration. PRICE S develops cost and schedule estimates for this activity, just as it does for the individual subsystems. It does this by relating the level of integration required for each individual subsystem to the effective amount of engineering and programming effort needed to bring the subsystems together into a total unified operation.

The technical and management problems associated with verification and validation are in many ways analogous to those of system integration. Experience has shown that very credible cost and schedule estimates can be obtained with PRICE S by treating validification and verification as if it were a moderately difficult system integration. If additional software, such as a system simulator, is required to support the validification and verification, it may be accounted for directly as another subsystem to be developed and incorporated in the equivalent system integration effort.

Multiple test beds are needed when software is to be installed into several different computer systems that may have varying operating systems and configurations. Since these conditions can have significant cost implications, PRICE S provides procedures to take them into account also.

PRICE S outputs

Fig. 1 shows a typical basic PRICE S output. It has four sections arranged as follows:

The *input data* section lists all the parameters associated with the descriptive variables. It sets forth the total scope of work, resources, schedules, and other significant factors.

The *program costs,* by work element, for each of the three development phases (design, implementation, and test and integration) are shown in thousands of dollars.

The *additional data* section provides information to be used as reference values for the purpose of measuring the credibility and consistency of the input data. Proper use of such data minimizes the probability of using costs resulting from faulty inputs. For example, if schedules are not specified, the calculated schedules are shown in this section. Any such inputs entered as zero will be calculated and printed out in this section.

The *schedule graph* depicts the workload phasing among the three development activities.

In addition to the basic PRICE S output, three optional output sections are shown in Figs. 2, 3, and 4.

The *sensitivity matrix* (Fig. 2) is a table of costs and schedules resulting from varying the engineering difficulty and resource inputs about their initial input values. Its use precludes the need to reprocess PRICE S to examine the cost and schedule sensitivity of these two significant factors.

The *schedule effect summary* (Fig. 3) consists of two tables. The first table compares the typical schedule calculated by PRICE S to the one specified by the user, which was 33 months overall in this example. The typical schedule results if the user only inputs the schedule start date instead of a specific schedule and lets the model calculate the schedule. Here, PRICE S selects a 22-month schedule and overall program costs are lowered by $240K or 16.4%. The second table shows the estimated cost penalties associated with inefficiencies induced by the directed schedule.

The *cost distribution table* (Fig. 4) shows the distribution of the work and associated costs throughout the duration of the development project. The distributions are based on the user-specified resource allocation profile assigned to each phase.

ACTIVITY LENGTH IN MONTHS

COMPLEXITY = 1.250	DESIGN	IMPL	T & I	TOTAL
SPECIFIED SCHEDULE	14.0	13.0	19.0	33.0
(OVERLAP)		(5.0)	(8.0)	
TYPICAL SCHEDULE	9.6	10.0	13.9	21.9
(OVERLAP)		(5.7)	(5.9)	

DEVELOPMENT COSTS

COMPLEXITY = 1.250	DESIGN	IMPL	T & I	TOTAL
SPECIFIED SCHEDULE	636.	129.	695.	1461.
TYPICAL SCHEDULE	500.	110.	611.	1222.
ESTIMATED PENALTY	136.	19.	84.	239.

Fig. 3
Schedule effect summary compares user-specified schedule and schedule generated by PRICE S (top), also lists cost penalties associated with user-specified schedule (bottom).

SAMPLE CASE MOBILE RADAR

MONTH	% COMPLETED DESIGN	% COMPLETED IMPL	% COMPLETED T & I	$ EXPENDED THIS MONTH	$ EXPENDED TOTAL	% EXPENDED THIS MONTH	% EXPENDED TOTAL
OCT 77	1.0	0.0	0.0	6.2	6.2	0.4	0.4
NOV 77	7.7	0.0	0.0	43.0	49.2	2.9	3.4
DEC 77	18.8	0.0	0.0	70.1	119.4	4.8	8.2
JAN 78	32.0	0.0	0.0	83.9	203.3	5.7	13.9
FEB 78	45.7	0.0	0.0	87.5	290.8	6.0	19.9
MAR 78	58.8	0.0	0.0	83.4	374.1	5.7	25.6
APR 78	70.5	0.0	0.0	74.1	448.3	5.1	30.7
MAY 78	80.2	0.0	0.0	61.8	510.1	4.2	34.9
JUN 78	87.8	0.0	0.0	48.2	558.4	3.3	38.2
JUL 78	93.3	0.1	0.0	34.9	593.2	2.4	40.6
AUG 78	96.9	1.3	0.0	24.3	617.5	1.7	42.3
SEP 78	98.9	5.2	0.0	17.9	635.4	1.2	43.5
OCT 78	99.8	12.5	0.0	15.1	650.5	1.0	44.5
NOV 78	100.0	22.9	0.0	15.0	665.5	1.0	45.6
DEC 78	100.0	35.8	0.0	16.8	682.2	1.1	46.7
JAN 79	100.0	50.0	0.1	19.0	701.3	1.3	48.0
FEB 79	100.0	64.2	0.4	20.8	722.1	1.4	49.4
MAR 79	100.0	77.1	1.2	22.2	744.3	1.5	51.0
APR 79	100.0	87.5	2.7	23.4	767.7	1.6	52.6
MAY 79	100.0	94.8	4.9	24.9	792.7	1.7	54.3
JUN 79	100.0	98.7	8.1	27.3	820.0	1.9	56.1
JUL 79	100.0	99.9	12.4	31.3	851.2	2.1	58.3
AUG 79	100.0	100.0	17.8	37.8	889.0	2.6	60.9
SEP 79	100.0	100.0	24.4	45.8	934.8	3.1	64.0
OCT 79	100.0	100.0	32.1	53.7	988.5	3.7	67.7
NOV 79	100.0	100.0	40.8	60.7	1049.2	4.2	71.8
DEC 79	100.0	100.0	50.4	66.3	1115.5	4.5	76.4
JAN 80	100.0	100.0	60.4	69.8	1185.3	4.8	81.2
FEB 80	100.0	100.0	70.5	70.5	1255.8	4.8	86.0
MAR 80	100.0	100.0	80.2	67.4	1323.2	4.6	90.6
APR 80	100.0	100.0	88.9	59.8	1383.1	4.1	94.7
MAY 80	100.0	100.0	95.6	46.7	1429.8	3.2	97.9
JUN 80	100.0	100.0	99.5	27.0	1456.8	1.9	99.7
JUL 80	100.0	100.0	100.0	3.8	1460.6	0.3	100.0
ALPHA =	0.82	0.0	0.0	FOR PROFILE GRAPHS			
BETA =	0.18	1.00	0.18	RESPOND OK = 7			
PEAK/AV =	1.93	1.88	1.93				

Fig. 4
Cost distribution shows when and where money will be spent throughout the duration of the software project.

The PRICE S example shown in Fig. 1 illustrates a mobile-radar software program requiring 36,000 machine-level instructions. This number was obtained by taking the program's approximately 8000 high-order-language (source) statements and converting them to the number of machine-level instructions by using empirically derived factors. As listed in the figure, approximately three-quarters of the machine code involves mathematical, interactive, and string-manipulation types of instructions. The software project is relatively new and will require, because of circumstances, thirty-three months to complete. The organization that will be doing the work and its costs (labor and overhead) are represented by an empirically derived resource value of 3.5. The computer utilization is 80%.

Conclusion

PRICE S has been under development for several years. From May through August 1977, a number of major software development organizations, including RCA, participated in field-testing the model. Their studies covered a variety of projects, ranging in size from a few thousand to several million instructions and in applications from payroll to space programs. Positive results and feedback from the field-testing encouraged the commercial sale of the model in September 1977, when the first formal PRICE S training class was held. By the spring of 1978, all the organizations participating in the field test had contracted for use of the model.

Dates and Deadlines

Upcoming meetings

Ed. Note: Meetings are listed chronologically. Listed after the meeting title (in bold type) are the sponsor(s), the location, and the person to contact for more information.

JUN 21-23, 1978—**Machine Processing of Remotely Sensed Data** (IEEE) West Lafayette, IN **Prog Info:** D. Morrison, Purdue Univ. LARS, 1220 Potter Drive, West Lafayette, IN 47906

JUN 26-28, 1978—**Device Research Conf.** (IEEE) Univ. of Calif., Santa Barbara, CA **Prog Info:** Dr. James McGroddy, IBM, T.J. Watson Research Ctr., Yorktown Heights, NY 10598

JUN 26-29, 1978—**Conf. on Precision Electromagnetic Measure** (IEEE, NBS, URSI/USNC) Conf. Ctr., Ottawa, Ont. **Prog Info:** Dr. Andrew F. Dunn, Natl. Research Council, Montreal Road, Ottawa, Ont.

JUN 27-29, 1978—**Intl. Microwave Symp.** (IEEE, et al) Chateau Laurier, Ottawa, Ont. **Prog Info:** A.L. VanKoughnett, Communications Research Ctr., POB 11490, Station H, Ottawa, Ont. K2H 8S2

JUL 10-13, 1978—**Intersoc. Environmental Systems** (ASME et al) Town & Country, San Diego, CA **Prog Info:** Tech. Affairs Dept., ASME, 345 E. 47th St., New York, NY 10017

JUL 16-20, 1978—**Fourth American Conf. on Crystal Growth** (NBS, Amer. Assn. for Crystal Growth) Natl. Bureau of Standards Gaithersburg, MD **Prog Info:** Dr. Robert L. Parker, Materials Building, Rm. B164, Natl. Bureau of Standards, Washington, DC 20234

JUL 18-21, 1978—**Nuclear & Space Radiation Effects** (IEEE) Univ. of New Mexico, Albuquerque, NM **Prog Info:** B.L. Gregory, Sandia Labs., Dept. 2140, Albuquerque, NM 87115

AUG 20-25, 1978—**Intersociety Energy Conversion Engr. Conf.** (IEEE) Town & Country, San Diego, CA **Prog Info:** George P. Townsend, Hamilton Standard Div., United Technologies Corp., Windsor Locks, CT 06096

AUG 22-25, 1978—**Intl. Conf. on Parallel Processing** (IEEE) Shanty Creek Lodge, Bellaire, MI **Prog Info:** Prof. T.Y. Feng, Dept. of Elect. & Comp. Engr., Wayne State U., Detroit, MI 48202

AUG 28-31, 1978—**Laser Applications and Optical Communication** (Soc. of Photo-Optical Instrumentation Engineers) Town and Country, San Diego, CA **Prog Info:** SPIE, PO Box 10, Bellingham, WA 98225

SEP 5-8, 1978—**COMPCON FALL** (IEEE) Washington, DC **Prog Info:** COMPCON FALL, P.O. Box 639, Silver Spring, MD 20901

SEP 5-7, 1978—**Intl. Optical Computing Conf.** (IEEE) Imperial College, London, Eng. **Prog Info:** S. Horvitz, Box 274, Waterford, CT 06385

SEP 6-8, 1978—**Fiberoptic Communication Conf. & Exhibit** (Info. Gatekeepers Inc.) Hyatt Regency O'Hare, Chicago, IL **Prog Info:** Information Gatekeepers, 167 Corey Rd., Suite 212, Brookline, MA 02146

SEP 6-8, 1978—**OCEANS '78** (IEEE, OEC, MTS) Sheraton Park, Washington, DC **Prog Info:** Myra Binns, Marine Tech. Society, 1730 "M" Street NW, Washington, DC 20036

SEP 12-14, 1978—**Western Electronic Show & Conv.—WESCON** (IEEE) Los Angeles Convention Ctr., Los Angeles, CA **Prog Info:** W.C. Weber, Jr., 999 N. Sepulveda Blvd., El Segundo, CA 90245

SEP 12-14, 1978—**Automatic Support Systems for Advanced Maintainability (AUTOTESTCON)** (IEEE) San Diego, CA **Prog Info:** Bob Aquais, General Dynamics Electronic Div., Mail Stop 7-98, PO Box 81127, San Diego, CA 92138

SEP 17-20, 1978—**Joint Fall Mtg. American Ceramic Soc. and IEEE Subcommittee on Ferroelectricity** (Am. Cer. Soc., IEEE) Dallas Hilton Inn, Dallas, TX **Prog Info:** Relva C. Buchanan, Dept. of Ceramic Engrg., 208 Ceramics Bldg., U. of Illinois, Urbana, IL 61801

SEP 21-23, 1978—**Interactive Techniques in Computer Aided Design** (IEEE) Palazzo dei Congressi, Fiera di Bologna, Italy **Prog Info:** Dr. Bertram Herzog, Computer Ctr., U. of Colorado, Boulder, CO 80303

SEP 24-27, 1978—**Electronic and Aerospace Systems Conv. (EASCON)** (IEEE) Sheraton Intl., Arlington, VA **Prog Info:** Bette English, At-Your Service, Inc., 821 15th Street NW, Washington, DC 20005

SEP 25-27, 1978—**Ultrasonics Symp.** (IEEE) Cherry Hill Hyatt House, Cherry Hill, NJ **Prog Info:** F.S. Welsh, Bell Telephone Labs, 555 Union Blvd., Allentown, PA 18103

OCT 1-5, 1978—**Industry Application Society Annual Meeting** (IEEE) Royal York Hotel, Toronto, Ont. **Prog Info:** W. Harry Prevey, 4141 Yonge Street, Willowdale, Ont., MSP 1N6

OCT 16-17, 1978—**Joint Engineering Management Conf.** (IEEE) Regency, Denver, CO **Prog Info:** Henry Bachman, Hazeltine Corp., Greenlawn, NY 11740

OCT 18-20, 1978—**Joint Automatic Control Conf.,** (IEEE) Civic Center, Philadelphia, PA **Prog Info:** Dr. Harlan J. Perlis, New Jersey Institute of Tech., 323 High Street, Newark, NJ 07102

OCT 18-20, 1978—**Canadian Communications and Power Conf.** (IEEE) Queen Elizabeth Hotel, Montreal, P.Q. **Prog Info:** Jean Jacques Archambault, CP/PO 757, Montreal, Quebec H2L 4L6

OCT 21-25, 1978—**Engineering in Medicine and Biology** (IEEE) Marriott Hotel, Atlanta, GA **Prog Info:** Walter L. Bloom, M.D., Georgia Institute of Tech., Atlanta, GA 30302

OCT 23-25, 1978—**Digital Satellite Communications** (IEEE) Hotel Reine Elizabeth, Montreal, Quebec **Prog Info:** Marcel Perras, Teleglobe Canada, 680 Sherbrooke Street W., Montreal, Que. H3A 2S4

OCT 24-26, 1978—**Biennial Display Research Conf.** (IEEE,SID) Cherry Hill Inn, Cherry Hill, NJ **Prog Info:** Lawrence Goodman, RCA Laboratories, Princeton, NJ 08540

OCT 25-27, 1978—**Intelec (Intl. Telephone Energy Conf.)** Sheraton Park, Washington, DC **Prog Info:** J.J. Suizzi, Bell Laboratories, Room 5D-178, Whippany, NJ 07981

Calls for papers

Ed. Note: Calls are listed chronologically by meeting date. Listed after the meeting (in bold type) are the sponsor(s), the location, and deadline information for submittals.

OCT 9-11, 1978—**Semiconductor Laser Conf. (6th)** (IEEE) Hyatt Regency, San Francisco, CA **Deadline Info: 6/15/78 to** T.L. Paoli, Bell Laboratories, 600 Mountain Ave., Murray Hill, NJ 07974

SEP 23-26, 1979—**3rd World Telecommunication Forum** (ITU) Geneva, Switzerland **Deadline Info: 9/30/78 100-200 word abs. to** A.E. Joel, Jr. Bell Telephone Laboratories, Room 2C-632, Holmdel, NJ 07733

Patents

Advanced Technology Laboratories

S.E. Ozga
Microprocessor architecture—4079455 (assigned to U.S. government)

Astro Electronics

D.S. Binge
Component rotation means—4076191

L. Muhlfelder
Roll/yaw body steering for momentum biased spacecraft—4084772

G.E. Schmidt|L. Muhlfelder
Magnetic control of spacecraft roll disturbance torques—4084773

Avionics Systems

F.C. Easter
Video digital display device with analog input—4086579

Broadcast Systems

R.N. Hurst|F.W. Huffman
Discontinuous motion special effects generator for television—4080626

Consumer Electronics

J. Avins
Automatic control of free wheeling—4084672

A.L. Baker
Video disc player mechanism control system—4086617

J.C. Peer
Circuit for correcting setup error in a color television receiver—4084115

J.J. Serafini
Automatic transient beam current limiter—4079424

T.D. Smith
Switch mechanism for a calculator type keyboard—4084071

S.A. Steckler
Low distortion signal amplifier arrangement—4081758

Government Communications Systems

R.H. Brader
Scrambler and unscrambler for serial data—4087626 (assigned to U.S. government)

L.N. Reed|J.R. Khalifeh
Overcurrent protection circuit—4084070

Laboratories

C.H. Anderson
Flat display device with beam guide—4076994

F. Aschwanden|T.E. Bart
SECAM subcarrier generator—4084177

V.S. Ban|S.L. Gilbert
Method for chemical vapor deposition—4082865

A. Bloom|D.L. Ross
R.A. Bartolini|L.K. Hung
Organic volume phase holographic recording media using sucrose benzoate—4084970

R.S. Crandall|B.W. Faughnan
Method of storing optical information—4075610

W. Denhollander
Power supply arrangement with minimum interaction between plural loads—4079295

A.R. Dholakia
Video disc player employing a spring loaded stylus apparatus—4077050

A.G. Dingwall
Method of making a substrate contact for an integrated circuit—4081896

J.G. Endriz
System for modulating a flat panel image display device—4077054

R.S. Engelbrecht|K. Knop
Diffractive phase filter having near field wavelength dependent focusing properties—4079411

I. Gorog|B.F. Williams
Optical filter—4084188

P.E. Haferl
Switched mode vertical amplifier with elimination of feedback ringing—4079293

P.E. Haferl
Conduction overlap control circuit for switched output stages—4081721

P.E. Haferl
Signal processor for switched vertical deflection system—4081722

W.E. Ham
Apparatus and method for cleaning and drying semiconductors—4079522

J.M. Hammer
On line electro-optic modulator—4076381 (assigned to U.S. government)

J.N. Hewitt|V. Christiano
Method of forming a metal pattern on an insulating substrate—4083710

W. Hinn
Video amplifier including an ac-coupled voltage follower output—4082996

H. Kawamoto|E.J. Denlinger
Pickup circuitry for a video disc player with printed circuit—4080625

J.B. Klatskin|A. Rosen
Method of manufacturing semiconductor devices having a copper heat capacitor and/or copper heat sink—4080722 (assigned to U.S. government)

K. Knop
Fabrication of rectangular relief profiles in photoresist—4082453

K. Knop
Zero-order diffractive subtractive filter projector—4082438

M.A. Leedom
Video disc package—4084691

F.J. Marlowe
Line scan converter for an image display device—4080630

A. Miller
Optical coupler—4082425

D.D. Mawhinney|Z. Turski
Microwave frequency discrimination—4079325 (assigned to U.S. government)

J.R. Oberman|R.G. Stewart
Write enhancement circuit—4075690

R.M. Rast
Dual mode frequency systhesizer for a television tuning apparatus—4078212

R.M. Rast|J. Tults
Phase locked loop tuning system with station scanning provisions—4077008

J.M. Shaw|K.H. Zaininger
Method of making planar silicon-on-sapphire composite—4076573 (assigned to U.S. government)

R.G. Stewart
Level shift circuit—4080539

M. Toda|S. Osaka
Method of producing optical image on chromium or aluminum film with high-energy light beam—4087281

J. Tults
Digital frequency deviation detector useful in a television receiving system—4084127

Z. Turski|A. Rosen
Microwave frequency discriminator comprising a one port active device—4085377 (assigned to U.S. government)

L.C. Upadhyayula
Planar transferred electron logic device with improved biasing means—4086501 (assigned to U.S. government)

J.L. Vossen, Jr.
Video disc with a conductive layer having an oxygen content gradient—4077051

J.L. Vossen, Jr.
Video disc capacitive recording means with a conductive bilayer—4077052

S. Weisbrod
Apparatus for measuring the current-voltage characteristics of a TRAPATT diode—4080571

Missile and Surface Radar

J.A. Lunsford|L.W. Martinson
Output buffer synchronizing circuit having selectively variable delay means—4079456 (assigned to U.S. government)

J.C. Williams|W.B. Sisco
Method and detection of phase and frequency modulation—4085367 (assigned to U.S. government)

O.M. Woodward
Circularly-polarized antenna system using tilted dipoles—4083051

Picture Tube Division

J. Evans, Jr.
Plural gun cathode ray tube having parallel plates adjacent grid apertures—4086513

H.A. Gross|R.F. Walters
H.R. Frey|J.D. Messner
Method for inspecting cathode-ray-tube window for objectionable cord—4076426

J.M. Ratay
Slurry process for coating particles upon the viewing-window surface of a cathode-ray tube—4078095

M.H. Wardell, Jr.|B.G. Marks
High voltage electron tube base with separate dielectric fill-hole—4076366

RCA Ltd., England

B. Crowle
Differential amplifier—4078206

SelectaVision Project

M.L. McNeely|H. Rees
Method for producing injection molded and centrally apertured disc records—4085178

C.F. Pulse
Video disc package having a center post—4084690

Solid State Division

G.W. Albrecht
Inverter circuit control circuit for precluding simultaneous conducting of thyristors—4078247

A.A. Ahmed
Self-starting amplifier circuit—4085359

R.D. Faulkner|R.E. McHose
Phototube having apertured electrode recessed in cup-shaped electrode—4079282

A.J. Leidich
Push-pull transistor amplifier with driver circuitry providing over-current protection—4078207

J.T. Mark
Vacuum tube grid structures of phosmic bronze having copper and copper alloy conical supports—4076992

H. Popp
Grid having reduced secondary emission characteristics and electron discharge device including same—4079286

R.J. Robe
Input stage for fast-slewing amplifier—4075575 (assigned to U.S. government)

O.H. Schade, Jr.
Ground fault detector—4080641

L.W. Varettoni
Voltage limiter circuit—4085432

H.A. Wittlinger
Voltage monitoring circuit—4084156

Pen and Podium

Recent RCA technical papers and presentations

To obtain copies of papers, check your library or contact the author or his divisional Technical Publications Administrator (listed on back cover) for a reprint. For additional assistance in locating RCA technical literature, contact RCA Technical Communications, Bldg. 204-2, Cherry Hill, N.J., extension 4256.

Advanced Technology Laboratories

M. Corrington
Two-dimensional image processing—IEEE CAS/ASSP Joint Meeting, U. of Penn., Philadelphia, PA (3/28/78)

A. Feller|T. Lombardi
A VLSI test chip, interpretation of its data and on-chip diagnostic techniques—IEEE Solid State Circuit Conf., San Francisco, CA (2/15-17/78)

R. Kenville
The optical video disc for digital mass memory applications—COMPCON Spring '78, San Francisco, CA (2/28-3/3/78)

J. Saultz|A. Feller
Approaches to developing and utilizing LSI in military equipment—IEEE SOS Talk, Wayland, MA (1/16/78)

Americom

J. Lewin
Ground control system for Satcom satellites—Seventh AIAA Communications Satellite Systems Conf., San Diego, CA (4/24/78)

Automated Systems

G.T. Burton
Laser beam recording of tactical imagery—SPIE Airborne Reconnaissance III Seminar, Washington, DC (3/29/78)

H.L. Fischer|T.E. Fitzpatrick
Simplified automatic test equipment—Industry/Joint Services ATE Workshop, San Diego, CA (4/3/78)

T.E. Kupfrian
Reliability test results of a commercial integrated circuit-based system — IEEE Spring Reliability Seminar, Boston, MA (4/27/78)

N.L. Laschever
Some considerations of the missile altimeter—*Proc.*, Strategic Penetration Technology Study 1977 (3/2/78)

J.N. Ostis
A transportable VFR terminal—*J. of Air Traffic Control* (3/78)

P.M. Toscano
Cost estimating relationships for ATE Test Program Set (TPS) development—Industry/Joint Services ATE Workshop, San Diego, CA (4/3/78)

Broadcast Systems

L.J. Bazin
Adapting a small camera for production applications—NAB, Las Vegas, NV (3/28/78)

R.N. Clark
The fan-vee circularly polarized tv antenna—*IEEE Trans. on Broadcasting*, Vol. BC-24, No. 1 (3/78)

M.S. Siukola
Evaluation of circularly polarized tv antenna systems—*IEEE Trans. on Broadcasting*, Vol. BC-24, No. 1 (3/78)

M.S. Siukola
RCA circularly polarized antennas—NAB Convention Workshop, Las Vegas, NV (4/10/78)

Consumer Electronics

B. Beyers|H. Blatter|J. George
J. Henderson|R. Rast|J. Tults
Frequency synthesis tuning systems with automatic offset tuning—IEEE Chicago Spring Conf. (6/5/78)

H. Blatter|R. Rast|J. Tults
Frequency synthesis custom LSI: the inside story—IEEE Chicago Spring Conf. (6/5/78)

Government Communications Systems

A. Birnbaum|R. Howe|P. Patterson
A parametric cost analysis model for EMSS—*Proc.*, 9th Annual Pittsburgh Conf. on Modeling and Simulation, Pittsburgh, PA (4/27-28/78)

R.M. Gould
Provisioning by reliability and maintainability factors—9th Annual Reliability Clinic, Holiday Inn, Philadelphia, PA (4/27/78)

M. Paskman
Evolution of an EMSS from model to operation—*Proc.*, 9th Annual Pittsburgh Conf. on Modeling and Simulation, Pittsburgh, PA (4/27-28/78)

P. Patterson
Electronic message service system definition & evaluation—the role of modeling—*Proc.*, 9th Annual Pittsburgh Conf. on Modeling and Simulation, Pittsburgh, PA (4/27-28/78)

P. Scott
An economic analysis of system growth—*Proc.*, 9th Annual Pittsburgh Conf. on Modeling and Simulation, Pittsburgh, PA (4/27-28/78)

G. Spector
Formulation of EMSS error objectives and error budgets—*Proc.*, 9th Annual Pittsburgh Conf. on Modeling and Simulation, Pittsburgh, PA (4/27-28/78)

R. Turner
EMSS network design and costing model—*Proc.*, 9th Annual Pittsburgh Conf. on Modeling and Simulation, Pittsburgh, PA (4/27-28/78)

B.E. Tyree|J.F. Bailey|V.C. Chewey
Ground mobile forces tactical satellite SHF ground terminals—7th AIAA Communications Satellite Systems Conf., San Diego, CA (4/23-27/78)

Government Systems Division Staff

J. Hilibrand
Recent advances in custom CMOS LSI—AFCEA Symp., Moorestown, NJ (4/13/78)

Laboratories

M.S. Bae
Defect free shallow P+/N junction and associated photovoltaic effect—13th IEEE Photovoltaic Specialists Conf., Washington, DC (6/5-8/78)

M.S. Bae
Schottky effect and photovoltaic devices on the texturized surfaces—13th IEEE Photovoltaic Specialists Conf., Washington, DC (6/5-8/78)

J. Blanc
Defects at the Si/SiO_2 interface: fact, fiction and fancy—Electrochemical Soc. Mtg., Seattle, WA (5/23/78)

C.A. Catanese|J.G. Endriz
The physical mechanism of feedback electron sources—1978 Chicago Spring Conf. on Consumer Electronics (6/5-6/78)

L.S. Cosentino|V. Christiano|J.G. Endriz
J. Dresner|G.F. Stockdale|J.L. Cooper
J.N. Hewitt|J.B. Harrison, Jr.
Feedback multiplier flat panel technologies—1978 Chicago Spring Conf. on Consumer Electronics (6/5-6/78)

J.K. Clemens
Capacitive pickup and the buried subcarrier encoding system for the RCA VideoDisc—*RCA Review*, Vol. 39, No. 1 (3/78)

L.P. Fox
The conductive VideoDisc—*RCA Review*, Vol. 39, No. 1 (3/78)

R.A. Geshner
Microlithography overview—IGC Symp. on Photolithography, Amsterdam, Netherlands (4/27/78)

W.C. Hittinger
New directions for 1983—NEWCOM '78, Las Vegas, NV (5/2-4/78)

E.O. Keizer
VideoDisc mastering—*RCA Review*, Vol. 39, No. 1 (3/78)

E.O. Keizer|D.S. McCoy
The evolution of the RCA "SelectaVision" VideoDisc system—a historical perspective—*RCA Review*, Vol. 39, No. 1 (3/78)

S.A. Keneman|J.G. Endriz
C.A. Catanese|L.B. Johnston
Flat tv display using feedback multipliers—1978 Chicago Spring Conf. on Consumer Electronics (6/5-6/78)

S.A. Keneman
Evolution of a flat television display using feedback multipliers—Colloquium, U. of Penna., Philadelphia (4/27/78)

H.W. Lehmann
Pattern transfer in the micron range by reactive sputter etching—IBM Research Laboratory, San Jose, CA (5/26/78)

A. Levine
Determination of water content of molded plastics—36th Annual Mtg. of SPE, Washington, DC (4/25/78)

D.S. McCoy
The RCA "SelectaVision" VideoDisc System—*RCA Review*, Vol. 39, No. 1 (3/78)

C.J. Nuese
Advances in heterojunction lasers for fiber optics applications—Soc. of Photo-Optical Instrumentation Engineers, Washington, DC (3/28-31/78)

D. Redfield
Cost criterion for low efficiency solar cells system power cost competitive with that of high efficiency cells—*Proc.*, 13th Photovoltaic Spec. Conf., Washington, DC (6/5-8/78)

D. Redfield
Reinterpretation of heavy-doping limitations on solar cell performance—*Proc.*, Photovoltaic Spec. Conf., Washington, DC (6/5-8/78)

R.N. Rhodes
The VideoDisc player—*RCA Review*, Vol. 39, No. 1 (3/78)

D.L. Ross
Coatings for VideoDisc—*RCA Review*, Vol. 39, No. 1 (3/78)

L.D. Ryan
Trends in programmable video game design—SID, Phila. Chapter (5/15/78)

N.D. Winarsky
Optimal numerical differentiation—SIAM Conf., Madison, WI (5/78)

Missile and Surface Radar

J.A. Bauer
Application of CMOS to hybrid design—ISHM, Hybrid Design Symposium, NY Chapter, (4/78)

M.W. Buckley
Project management—Instr., Cont. Engrg. Educ. Prog. (George Washington U.), Brussels, Belgium (4/78); Lect.—EBI Seminar, London, England (4/78)

W.T. Patton
Very low sidelobe electronic scanning reflector antenna—IEEE Conv., Boston, MA (4/78)

H. Urkowitz
A personal view of mathematics in the electronics industry—Rutgers University, Camden, NJ (4/78)

SelectaVision Project

W.J. Gordon
VideoDisc testing philosophy and techniques—*RCA Review*, Vol. 39, No. 1 (3/78)

R.J. Ryan
Materials and process development for VideoDisc replication—*RCA Review*, Vol. 39, No. 1 (3/78)

Engineering News and Highlights

New engineering management in CCSD

Arch C. Luther was appointed Chief Engineer, Commercial Communications Systems Division by **Neil Vander Dussen,** Division Vice President and General Manager. In his new position Mr. Luther will be responsible engineering activities at Broadcast Systems, Mobile Communication Systems, Avionics Systems, and Cablevision Systems. He was Chief Engineer of Broadcast Systems since 1973. During his 28 years with RCA, Mr. Luther has worked on color-tv studio equipment, many video tape recorders (including the Emmy-winning TCR-100 video cartridge recorder) and other studio and transmitting equipment for radio and television. In 1975, he received the David Sarnoff Award "for his many outstanding technical contributions enhancing RCA's reputation as a leading supplier of television systems."

Anthony H. Lind was appointed Chief Engineer, Engineering, for Broadcast Systems by **J. Edgar Hill,** Division Vice President and General Manager, Broadcast Systems. Mr. Lind has been with Broadcast Systems since 1946, and an Engineering Manager since 1951. His engineering organizations have designed and developed numerous broadcast products—cameras, tv tape recorders, projectors, switches, and terminal equipment. He has participated in many industry technical committees and is presently on the Board of Governors of the SMPTE. Last year he was elected a Fellow of the IEEE in recognition of his "technical leadership in the design and product development of video tape recorders and color television cameras."

Richard L. Rocamora was recently appointed Manager, Meadow Lands Broadcast Engineering, for RCA Broadcast Systems. He is responsible for the engineering design and development of the company's line of a.m. and fm radio and tv broadcast transmitters and audio products. Previously, Mr. Rocamora was Manager, Antenna Engineering, at the antenna development and test center in Gibbsboro. Mr. Rocamora joined RCA in 1952 as an electrical engineer and served in design engineering and managerial positions in the fields of microwave equipment, digital terminals, computers, and communications systems, as well as radio and tv broadcast equipment.

Degrees granted

RCA Laboratories

Ronald E. Daniel,—MS, Industrial Engineering/Operations Research; Rutgers University.

Nils O. Ny,—BS, Computer Science, Monmouth College.

Still more Fellows, PEs

Two issues ago, we published two lists—one of RCA engineers who had attained the grade of Fellow of the IEEE, the other of the Licensed Professional Engineers at RCA. Last issue, we published additions to both lists; the updating continues here.

IEEE Fellows

K.A. Chittick 1953

Licensed engineers

When you receive a professional license, send your name, PE number (and state in which registered), RCA division, location, and telephone number to *RCA Engineer,* Bldg. 204-2, RCA, Cherry Hill, N.J. New listings (and corrections or changes to previous listings) will be published in each issue.

Consumer Electronics

E.J. Bynum, Prescott, Ont., ONT-

L.M. Krugman, Bloomington, Ind.; IN-9460, NJ-7700.

RCA Service Co.

Danton, K.K., AUTEC; AL-103-28, FL-16851

Harvey, L.R., AUTEC; FL-16787

Jackson, C.L., AUTEC; TX-16531

Klein, E.L., AUTEC; DC-3563

Schmidt, A.J., AUTEC; PA-11152E

Scott, R.E., AUTEC; MI-8075, FL-15118

Missile and Surface Radar

Kisko, R.P., Moorestown, N.J.; NJ-21840

Landry, R.J. Moorestown, N.J.; CT-9634

Government Communications Systems

Cochrane, S.A., Camden, N.J.; CA-Q2139

Picture Tube Division

Stanton, G.W., Scranton, Pa.; PA-023735E

Promotions

Missile and Surface Radar

H. Anderson from Unit Manager, Design and Development to Manager, TRADEX Site.

Bruno F. Melchionni was recently appointed Manager, Antenna Engineering, for Broadcast Systems. He is headquartered at the Gibbsboro, N.J. facility where broadcast antennas for radio and television are designed and built. Previously, Mr. Melchionni was Manager, Systems Engineering and Custom, Repair and Engineering Shop, a position he had held for four years. Originally joining RCA in 1941, he has served in various engineering management capacities, including positions in video tape engineering, tv film systems, and drafting and engineering standards. Mr. Melchionni is a member of the Society of Motion Picture and Television Engineers.

Burris Heads Corporate Engineering Education

Frank E. Burris was recently appointed Manager, Engineering Education, reporting to Dr. William J. Underwood, Director, Engineering Professional Programs. Dr. Burris will guide and direct the Corporate Engineering Education (CEE) unit located at Cherry Hill, N.J. The CEE unit is responsible for addressing the technical education needs of RCA's technical staff as one means of enhancing RCA's technical viability. The unit is perhaps best known for its video tape education courses.

Dr. Burris was previously located at Bradley University, Peoria, Ill., as Assistant Professor of Electrical Engineering and Electrical Engineering Technology. He received his undergraduate and graduate education in electrical engineering from the Univ. of Cincinnati. Dr. Burris has been extremely active in engineering professional education and is currently Vice President—Geographic Councils and a Member of the Executive Committee of the American Society for Engineering Education.

G. Boose from Senior Member, Engineering Staff to Unit Manager, Design and Development Engineers.

G. Brady Senior Member, Engineering Staff to Unit Manager, Engineering Systems Project.

Peter Fletcher from Senior Member, Engineering Staff to Unit Manager, Engineering Systems Project.

Barry Galman from Principal Member, Engineering Staff to Unit Manager, Engineering Systems Project.

J. Gross from Senior Member, Engineering Staff to Unit Manager, Engineering Systems Project.

R. Kingsley from Senior Member, Engineering Staff to Unit Manager, Engineering Systems Project.

Robert Main from Senior Member, Engineering Staff to Unit Manager, D&D Engineers.

Stephen McCammon from Principal Member, Engineering Staff to Unit Manager, Engineering Systems Project.

G. Mondle from Senior Member, Engineering Staff to Unit Manager, Engineering Systems Project.

A. Whitley from Unit Manager, ESP to Manager, Engineering.

Solid State Division

Michael J. Meehan from Leader, Technical Staff, to Manager, Wafer Fabrication Module II, Findlay.

Consumer Electronics

Jack S. Fuhrer from Senior Member, Engineering Staff to Manager, Signal Circuits.

M.W. Garlotte from Senior Member, Engineer Staff to Manager, Instrument Mechanical Engineering.

Staff Announcements

Solid State Division

Carl R. Turner, Division Vice President, Integrated Circuits, announced the organization as follows: **Marvin B. Alexander,** Manager, Operations Control—IC; **Donald R. Carley,** Manager, Automotive Programs; **Larry J. Gallace,** Project Manager, Quality and Reliability Programs—IC; **Harry B. Gould,** Manager, International IC Manufacturing; **Ronald J. Costlow,** Manager, Solid State—RCA Taiwan Ltd.; **Stanley Rosenberg,** Director, Government and Hi Reliability IC Products; **Richard L. Sanquini,** Director, IC Products; **John E. Schaefer,** Manager, Domestic IC Manufacturing; **John E. Schaefer,** Acting, Palm Beach Gardens Operations; **Joseph H. Scott,** Director, Integrated Circuit Technology; **Norman C. Turner,** Manager, Operations Planning—IC; and **Robert A. Donnelly,** Administrator, Operations Planning—IC.

Mr. Gould, while continuing his present responsibility as Managing Director, RCA Senderian Berhad (Malaysia), is appointed Manager, International IC Manufacturing. Messrs. Gould and Costlow will report administratively to the appropriate subsidiary management Vice President, Integrated Circuits, and the Manager, International IC Manufacturing, respectively.

Mr. Scott, while continuing his present responsibilities under Gerald B. Herzog, Staff Vice President, Technology Centers, will report to the Division Vice President, Integrated Circuits, for the Silicon on Sapphire project.

Richard L. Sanquini, Director, IC Products, announced the organization as follows: **Gerald K. Beckmann,** Manager, MOS IC Product Marketing; **Michael S. Fisher,** Manager, IC Applications Engineering and Test; **Heshmat Khajezadeh,** Manager, IC Engineering; **Seymour Reich,** Manager, Bipolar IC Marketing; and **John R. Steiner,** Administrator, IC Products.

Michael S. Fisher, Manager, IC Applications, Engineering and Test, announced the organization as follows: **Wayne M. Austin,** Leader, Technical Staff—Consumer Bipolar IC's; **Richard E. Funk,** Leader, Technical Staff—MOS Logic; **Al A. Key,** Leader, Technical Staff—Memories, Microprocessors and Software; **George I. Morton,** Leader, Technical Staff—MOS Custom and Timekeeping; **Bruno J. Walmsley,** Manager, Test Technology; **Charles Engelberg,** Leader, Technical Staff—Test Technology—MOS; and **Harold Wittlinger,** Leader, Technical Staff—Industrial Bipolar IC's.

Heshmat Khajezadeh, Manager, IC Engineering, announced the organization as follows: **Orest Harasymowych,** Administrator, Design Engineering; **Raymond A. McFarlane,** Manager, Equipment Technology; **Henry S. Miller,** Manager, IC Technology Support; **Arnold S. Rose,** Project Manager, Advanced Processing; **Alfredo S. Sheng,** Manager, Bipolar IC Design Engineering; and **Alexander W. Young,** Manager, MOS Design Engineering.

Raymond A. McFarlane, Manager, Equipment Technology, announced the organization as follows: **Alva B. Hom,** Leader, Technical Staff—Assembly; **Dieter G. Krawitz,** Manager, Equipment Technology Shop; and **Robert C. Shambelan,** Leader, Technical Staff, Wafer Processing.

Henry S. Miller, Manager, IC Technology Support, announced the organization as follows: **Martin A. Blumenfeld,** Leader, Technical Staff—Palm Beach Gardens Reliability; **Robert E. Kleppinger,** Leader, Technical Staff—Findlay Reliability; **Kenneth J. Orlowsky,** Leader, Technical Staff—IC Model Shop; **Murray A. Polinsky,** Leader, Technical Staff—BIC Wafer Processing; and **Maurice I. Rosenfield,** Leader, Technical Staff—IC Packaging and Assembly.

Alfredo S. Sheng, Manager, Bipolar Integrated Circuits, Design Engineering, announced the organization as follows: **Thomas H. Campbell,** Leader, Technical Staff—Product Development—Industrial BIC; **Merle V. Hoover,** Leader, Technical Staff—Circuit Design; **Lewis A. Jacobus,** Manager, BIC Reliability; **Stephen C. Ahrens,** Leader, Technical Staff—Reliability Programs Coordination; **Max Malchow,** Leader, Technical Staff—Circuit Design—Communications; and **Sterling H. Middings,** Leader, Technical Staff—Product Development, Consumer BIC.

Alexander W. Young, Manager, MOS Design Engineering, announced the organization as follows: **Richard P. Fillmore,** Leader, Technical Staff—Memory Projects; **Edwin M. Fulcher,** Leader, Technical Staff—Support Systems Design; **Mark D. Holbrook,** Leader, Technical Staff—1800 Design; **Joel R. Oberman,** Leader, Technical Staff—8085 Design; **George J. Waas,** Manager, CMOS Design Engineering and Support; and **Nicholas Kucharewski,** Leader, Technical Staff—CMOS Circuit Design.

John E. Schaefer, Manager, Domestic Integrated Circuit Manufacturing, announced the organization as follows: **Philip C. Baumann,** Manager, Plant Engineering—Findlay; **Alco D. Brown,** Manager, Wafer Fabrication—IC's; **George W. Barclay,** Manager, Wafer Fabrication—LD²; **Michael J. Meehan,** Manager, Wafer Fabrication—MOS; **Raymond C. Reutter,** Manager, Wafer Fabrication—Bipolar; **Richard E. Davey,** Manager, IC Operations Support and Stem Manufacturing; **John E. Schaefer,** Acting, Palm Beach Gardens Operations; **John C. LaBerge,** Manager, Assembly/Test and Warehouse—IC's; **John G. Nussear,** Manager, Engineering and Test—MOS; **David D. Pfeiffer,** Manager, Industrial Relations—Findlay; **John A. Schramm,** Manager, Quality and Reliability Assurance; and **John H. Sundburg,** Manager, Engineering and Test—Bipolar.

RCA Records Division

Robert D. Summer, President, RCA Records Division, announced the organization as follows: **Herb S. Helman,** Division Vice President, Public Affairs; **David A. Heneberry,** Division Vice President, Music Service; **Melvin Ilberman,** Division Vice President, Business Affairs and Associated Labels; **Arthur C. Martinez,** Division Vice President, Finance and International; **William M. O'Grady,** Division Vice President, Industrial Relations; **Ernest P. Ruggieri,** Division Vice President, Manufacturing; and **Robert D. Summer,** Acting, RCA Records—U.S.A.

RCA Laboratories

Richard E. Quinn, Staff Vice President, Administration, announced the formation of the Thin Film Technology activity. **John L. Vossen, Jr.** is appointed Manager, Thin Film Technology.

Peter Friederich, Staff Vice President, Industrial Relations, announced the appointment of **Henry J. Liszewski** as Manager, Organization Development and Training.

Robert H. Dawson has been appointed Manager, New Technology Applications Research, Television Research Laboratory.

Distributor and Special Products Division

J.J. Badaracco announced the appointment of **Paul R. Slaninka,** Division Vice President, Marketing and Consumer Products Management.

Forty-seven RCA engineers and scientists honored for 1977 research achievements

Dr. William M. Webster, Vice President, RCA Laboratories, Princeton, announced that forty-seven scientists have been given RCA Laboratories Outstanding Achievement Awards for contributions to electronics research and engineering during 1977. Recipients of individual awards are:

Liston Abbott, for development of techniques that double the television transmission capacity of satellites.

Vladimir S. Ban, for fundamental studies and novel concepts leading to the design of improved chemical-vapor-deposition reactors.

John C. Hartmann, for improved techniques in the fabrication of aperture masks.

Ronald L. Hess, for leadership and technical contributions resulting in significant improvements in television chassis design.

Richard J. Hollingsworth, for the design and development of advanced CMOS/SOS memories.

Alfred C. Ipri, for continuing innovative contributions to the development of silicon-on-sapphire devices, circuits and process technology.

Ramon U. Martinelli, for important contributions to the design of fast, high power switching transistors.

Samuel H. McFarlane, III, for developing a comprehensive system of automatic control and data acquisition of materials characterization.

Joseph C. Toner, for resourceful and innovative application of financial techniques to analysis and control of semiconductor processing.

Recipients of team awards are:

Abe Abramovich, Thomas F. Lenihan, Angelo R. Marcantonio, Paul M. Russo, and **Chih C. Wang,** for contributions leading to imaginative applications of microprocessors to automated manufacturing systems.

Richard A. Auerback, Frederick E. Brehn, Donald F. Fraipont, Evelyn Jetter, and **Richard A. Sunshine,** for contributions leading to applications of semiconductor processing, monitoring and control systems.

Phyllis B. Branin, Simon Larach, and **Gerald S. Lozier,** for contributions to the development of cost-effective techniques for the fabrication of color kinescope screens of low reflectivity.

Jon K. Clemens, John H. Reisner, and **Howard G. Scheible,** for contributions leading to a two-hour RCA VideoDisc.

Seymour H. Cohen, Joseph J. Fabula, Charles D. Mulford, Stephen G. Policastro, and **Dae Shik Woo,** for contributions to long-term studies and implementation of radiation-hardened technologies for complementary MOS arrays.

Michael A. Colacello, Ralph DeStephanis, and **Harvey O. Hook,** for contributions to the development of a unique welding system for kinescope guns.

Pabitra Datta, Leonard O. Fox, and **Hiro-Hisa Kawamoto,** for contributions to the development and implementation of a novel capacitive VideoDisc that eliminates the need for coatings.

John W. Gaylord, Burnett H. Sams, James A. Goodman, Arthur Kalman, and **Thomas M. Stiller,** for contributions to the development of a real-time computer system for semiconductor process monitoring and control.

Steven J. Hallermeier, William J. Maddox, Raymond T. Marsland, and **Hemmige V. Rangachar,** for contributions to the development of a new automatic shadow-mask-production process control system.

Hans P. Kleinknecht and **Heinrich Meier,** for contributions to the novel application of optical technologies to the processing of semiconductors.

Twelve at Burlington get TE award

The electro-optic multiplexer is a small, militarized tv camera designed to convert the output of a FLIR (Forward-Looking Infra-Red) sensor to the serial format required for tv display. Designed by engineers at Automated Systems, it brought them a TE award and an additional $1M contract for Burlington. Pictured are (from left) **H.J. Woll,** Div. VP and General Manager; team members **R.C. DeMeo, D.J. Morand, R.E. Rooney, J.J. Klein, A.P. Arntsen, R.A. Dalrymple, L. Arlan; F.F. Martin,** acting Chief Engineer; **R.G. Spiecker, J.C. Tranfaglia, B.T. Joyce,** and **R.P. Sharland.**

Hurst gets writing award

Wins writing award—Robert N. Hurst (center), Administrator, Broadcast Technical Training, for Broadcast Systems, Camden, is congratulated upon winning a 1978 Neal Editorial Achievement Award for a series of articles on digital television that appeared in a leading trade magazine. The Neal awards, given by American Business Press, Inc., are the highest in the business publication journalism field. Offering congratulations are (left) **J.E. Hill,** Division Vice President and General Manager, and **John W. Wentworth,** Manager, Broadcast Technical Training, Broadcast Systems.

Automated Systems cited for contributions to the National Military Information Center

The Defense Intelligence Agency recently presented an award to RCA Automated Systems for its performance in support of the National Military Information Center (NMIC).

Automated Systems has been working on two successive programs to modernize the NMIC since October of 1974. Specifically, Automated Systems developed a real-time computer-based system, called DIPOLES, that is used to process time-sensitive intelligence data; then they applied techniques developed on the DIPOLES project to configuration management, system testing, and baseline documentation for the rest of the NMIC support system.

In a ceremony at the Burlington plant, several NMIC team members presented their award to **H.J. Woll,** Division Vice President and General Manager. Team members present for the ceremony were **M. Coyne, W. Cleveland, F. Chaples** (Program Manager, holding award plaque), **R. Coulter, R. Lindley,** and **R. Bitteker.** Team members **J. Kryzanowsky, C. McKusick,** and **R. Monat** were not at the presentation.

ATL recognizes authors/inventors

Advanced Technology Laboratories, Camden held its annual authors/inventors reception in April. Members of ATL who published a paper, gave a presentation, or got a patent during 1977 were honored at the event.

Scranton Engineer wins cost reduction award

William A. Parker, a member of the Technical Staff of the Process/Production Engineering department of Picture Tube Division's Scranton plant, was named CRS (Cost Reduction Savings) Man of the Year by that plant's CRS committee. A spokesman for the committee said Parker was "responsible for major savings in his immediate work area," and that "his involvement in expediting the operation of sophisticated test equipment resulted in increased capabilities and improved tube quality." Parker's prize was a trip to Hawaii.

Professional activities

Rittenhouse a chairman for Electronics Industries Association

John D. Rittenhouse, Division Vice President and General Manager, has been elected Chairman, Government Communications Council, for the Electronics Industries Association. The Electronics Industries Association is a national industrial organization of US electronics manufacturers. As Chairman, Mr. Rittenhouse is responsible for leading discussions among senior industry and government leaders on defense industry needs.

Abramovich receives Best Paper Award

Abe Abramovich, Systems Research, RCA Laboratories, Princeton, recently received a Best Paper Award at the IEEE's Industrial Electronics and Control Instrumentation Society annual conference in Philadelphia. Written with Tom Crawford, an MIT-co-op student, the paper is entitled "An Interpolating Algorithm for Control Applications on Microprocessors."

Society of Women Engineers

Ed. note: Two engineers from RCA Laboratories, Princeton, have been elected officers for the New Jersey Section of the Society of Women Engineers. **Daryl A. Doane,** Member, Technical Staff, Integrated Circuit Technology will be President and **Karen A. Pitts,** Member, Technical Staff, Systems Analysis Research, will be Secretary.

Yvonne C. Brill, recently appointed Manager of NOVA Propulsion for Astro Electronics, Princeton, will be stepping down as President of the New Jersey Section. We asked her to tell us more about the Society.

The Society of Women Engineers (SWE) was founded in 1949-50 when small groups of women engineers started meeting in New York, Boston, Philadelphia, and Washington, D.C. SWE was incorporated in 1952 as a professional, non-profit, educational service organization of graduate women engineers and women with equivalent engineering experience.

In 1961, the Society established its headquarters office in the United Engineering Center in New York City. There SWE joined many of the engineering societies representing specific engineering disciplines, as well as the Engineers' Joint Council and the Engineers' Council for Professional Development. SWE encourages each member to belong to and actively support the society representing her particular field of engineering.

The specific objectives of the Society are:

- To inform young women, their parents, counselors, and the general public of the qualifications and achievements of women engineers and the opportunities open to them.
- To assist women engineers in readying themselves for a return to active work after temporary retirement.
- To serve as a center of information on women in engineering.
- To encourage women engineers to attain high levels of education and professional achievement.

SWE presently has an international membership of 5200 women. There are SWE sections in 24 areas of the United States and Student sections have been chartered in 114 colleges, universities, and engineering institutes.

The New Jersey Section of SWE has approximately 100 members, including three chartered student sections at Princeton, Rutgers, and Stevens Institute. In keeping with SWE objectives, the section provides speakers to assist high school career days and sponsors open houses with local industries for area high school students. Section members representing all engineering disciplines are available on a one-to-one basis for counseling young women interested in pursuing engineering as a career.

More staff announcements

Patent Operations

John V. Regan, Vice President, Patent Operations, Princeton, announced the appointment of **Birgit E. Morris,** Senior Managing Patent Attorney; **Paul J. Rasmussen,** Senior Managing Patent Attorney; **Joseph S. Tripoli,** Managing Patent Attorney; and **Robert A. Hays** as Resident Patent Counsel, Cherry Hill.

Government Communications Systems

Joe Terry Swaim has been appointed Manager of the TENLEY/SEELEY programs.

Advanced Technology Laboratories

James A. Colligan has been appointed Manager of Marketing.

Engineering Professional Programs

William J. Underwood, Director, Engineering Professional Programs, announced the organization as follows: **Frank E. Burris,** Manager, Engineering Education; **Hans K. Jenny,** Manager, Technical Information Programs, and **Mary Ann Keating,** Administrator, Engineering Professional Programs.

Astro-Electronics

P.J. Martin, Director, Marketing and Advanced Planning, announced the appointment of **Frank C. Weaver** as Manager, Marketing Administration.

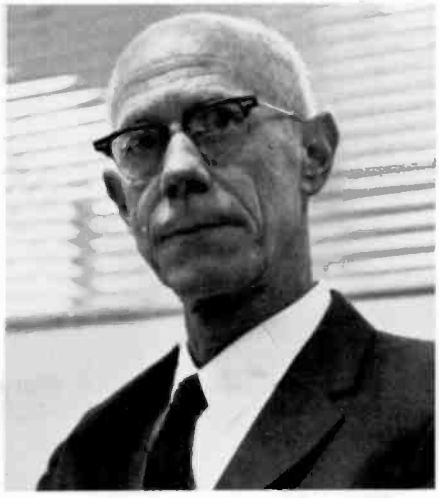
Service Company

Donald M. Cook, Division Vice President, Government Services, Marketing, announced the appointment of **Joseph Kingan** as Manager of Planning and Development for Navy Contracts.

Obituaries

Albert C. Grimm, a retired engineer who held several key responsibilities during his 34 years with the Industrial Tube Division of Electronic Components, Lancaster, Pa., died April 28.

Mr. Grimm joined the RCA Tube Division in 1940 and was assigned to the development of receiving and small power tubes. In this assignment from 1940 to 1951, he advanced to engineering leader. From 1951 to 1953, Mr. Grimm directed the advanced development effort on the RCA color kinescope. He was promoted to Product Manager for the Color Kinescope Product Group in 1954. In 1955, he was named Manager of Advanced Development in the Power Tube Group. In this capacity he was responsible, among other programs, for the development of the RCA 8568 klystron used to power the Stanford two-mile linear accelerator. From 1960 to 1967, he was manager of engineering and operations manager for RCA super power tubes, and then was appointed Administrator of Industrial Tube Market Development. He had six patents on electron tubes and associated devices, and he published several papers on the developmental aspects of vacuum tubes. He was a member of Tau Beta Pi and Eta Kappa Nu, and was a senior member of the IEEE. He was a registered professional engineer in the State of Pennsylvania.



Bertram A. Trevor, a well-known and respected RCA engineer for 43 years, died on January 11, 1978.

He first worked at the original RCA Laboratory at Riverhead, N.Y., in research, design and development for the worldwide short-wave system and also pioneered in the application of UHF waves and early wideband TV relay systems. During World War II Mr. Trevor headed a project resulting in a pulse position modulated, eight channel microwave radio relay system for the U.S. Signal Corps. In 1951, he transferred to the RCA Laboratories at Princeton, where he was Project Manager of the COSMOS Project, a study for naval communications in the 70s. For his work on this project he received the "RCA Inventive" Award. In 1958 he went to RCA Surfcomm at Vail, Arizona as staff consulting engineer and IR&D coordinator. In 1965 he became Senior Member, Technical Staff, for projects at Camden and Moorestown. He worked on the MALLARD study and received an award for Excellence in Performance on that assignment. In 1969 Mr. Trevor joined the Astro-Electronics Division as a senior scientist and engineer in the Systems Engineering area. One of his last efforts before retiring in 1971 was the system study and design for the remove control of the Apolla Camera left on the moon by the last LEM flight. He was a Fellow of the IEEE and a member of Sigma Xi and the IEEE Professional Group on Communications Technology.

Arthur W. Vance, former head of the RCA Princeton Systems Laboratory, and the first Chief Engineer of Astro Electronics Products Division, died on March 20, 1978.

While at RCA, Vance's achievement and behavior were legendary. It was out of his Systems Lab that the pioneer designs and subsystems configurations came, which demonstrated the usefulness and feasibility of the communications for SAMOS-type reconnaissance satellites. His group also was the wellspring for the first RCA operational satellites. In the early 1950s, for the "Black Cat" project, he and his Systems group, together with MIT's Stark-Draper Instrumentation Laboratory, developed the complete subsystem configuration that is the essence for the weapons system whose implementation is currently in national debate today. Under Dr. V.K. Zworykin, Vance made vital contributions in the development of the electron microscope and to the engineering success of present-day television. Vance's wartime activities were in the development of analog computers for gun detectors, the first of which was hurriedly sent to the ETO for usage against the Buzz Bombs and V-2s. Established were design principles found in most of the subsequent fire control systems. His group designed and built what is believed to be the longest precision analog computer ever made in this country: the Typhoon Guided Missile Flight Simulator.

Retirements

Dr. Otis D. Black retires at the end of June, after 36 years with RCA. Since he started with RCA in 1942, he has worked in the Central Engineering Materials Laboratory, Government Communications Systems, and predecessor organizations. "Doc" Black is a recognized solder expert in RCA. In the early days of printed wiring he made contributions in the areas of laminates, photoresists, and etchants, as well as on fluxes, solder, and cleaning solvents. This variety of involvement continued with the advent of multilayer boards, until his retirement. "Doc" Black has patents in the printed wiring field, and has contributed to or authored various technical publications. He has been a member of solderability committees of the Institute of Printed Circuits and the American National Standards Institute, Sigma Xi, Phi Beta Kappa, and Phi Lambda Upsilon. He is listed in American Men of Science.

Ken Weaber will retire at the end of July after 37 years with RCA. For the past seventeen years he has been with the Central Engineering Parts Application activity, Government Communications Systems. Prior to that he had been with the (then) Home Instruments Division and the Advanced Development Section. Since mid-1965, he has had primary responsibility for developing specifications, conducting qualification tests, and screening integrated circuits for High-Reliability programs. At the Home Instruments Division Ken was involved in the evaluation and qualification of a wide range of magnetic, inductive, and capacitive components. He did basic work in developing a widely used line of chemically activated fuses. In Advanced Development, he was an optical engineer, having primary responsibility for the development of a white room facility and for the production of lenses and mirrors.

Index to Volume 23

This index covers issues 23-1 (Jun|Jul 1977), 23-2 (Aug|Sep 1977), 23-3 (Oct|Nov 1977), 23-4 (Dec 1977|Jan 1978), 23-5 (Feb|Mar 1978), and 23-6 (Apr|May 1978). The RE number following the article identifies the issue in which the article appears.

Subject Index

Microcomputers

ATMAC MICROPROCESSOR, the—A.D. Feigenbaum|W.A. Helbig|S.E. Ozga (ATL, Cmdn.) RE-23-1-18

BINARY-TO-DECIMAL CONVERSION program for a programmable calculator—A.R. Campbell (MSR, Mrstn.) RE-23-2-18

CVM—A MICROPROCESSOR-BASED intelligent instrument—C.T. Wu (Labs, Som.) RE-23-1-5

MICROCOMPUTER CONTROL for the car of the future—E.F. Belohoubek|J.M. Cusack|J.J. Risko|J. Rosen (Labs, Pr.) RE-23-1-10

SINGLE-WIRE ALARM SYSTEM using the COSMAC Microtutor—T.F. Lenihan (Labs, Pr.) RE-23-3-19

Patents

AMERICAN PATENT SYSTEM, development of the—B.E. Morris (Pat. Ops., Pr.) RE-23-1-19

Radar

ADVANCED ANTENNA DESIGN reduces electronic countermeasures threat—R.M. Scudder (MSR, Mrstn.) RE-23-5-12

AIRBORNE WEATHER RADAR, color displays for—R.H. Aires|G.A. Lucchi (Av., Van Nuys) RE-23-5-13

DIGITAL COMPUTER SIMULATION of radar systems—J. Liston|G.M. Sparks (MSR, Mrstn.) RE-23-5-15

EARLY DAYS OF RADAR, the—T.G. Greene (GSD, Ch.Hl.) RE-23-5-1

ENHANCING ANTENNA PERFORMANCE through multimode feeds and distribution networks—C.E. Profera (MSR, Mrstn.) RE-23-5-11

MICROCOMPUTERS FOR RADAR SYSTEMS—B.D. Buch|S.L. Clapper|R.J. Smith (MSR, Mrstn.) RE-23-5-7

MICROWAVE POWER AMPLIFIERS, pulsed GaAs FET, for phased-array radars—R.L. Camisa|J. Goel|H.J. Wolkstein|R.L. Ernst (Labs, Pr.) RE-23-5-9

PROGRAMMABLE PROCESSORS and radar signal processing—an applications overview—M.G. Herold|M.C. Timken (MSR, Mrstn.) RE-23-5-8

RADAR CONCEPTS, an introduction to—B. Fell (MSR, Mrstn.) RE-23-5-10

RADAR PROCESSING ARCHITECTURES—W.W. Weinstock (MSR, Mrstn.) RE-23-5-5

RADAR WEAPON SYSTEM SENSORS—a system/technology perspective—A.S. Robinson (MSR, Mrstn.) RE-23-5-3

RADIO VISION—the early days of radar at RCA—I. Wolff (ret.) RE-23-5-2

RANGE INSTRUMENTATION, economical approaches to updating—R.G. Higbee (MSR, Mrstn.) RE-23-1-13

SHORT-RANGE RADAR for measuring blast-furnace burden height—H.C. Johnson (Labs, Pr.)|R.W. Paglione (Labs, Pr.)|J.P. Hoffman (Beth. Steel) RE-23-5-14

SUPER-POWER RADARS, solid state for—D.L. Pruitt (MSR, Mrstn.) RE-23-5-10

Software

COMPUTER SCIENCE, starting yourself out in—P. Anderson (cons.) RE-23-6-11

FLECS—A structured programming language for minicomputers—T. Stiller (Labs, Pr.) RE-23-6-17

INTERPRETER CONTROL PROGRAM simplifies automatic testing—A. Marcantonio (Labs, Pr.) RE-23-6-20

MICROCOMPUTERS in picture tube manufacturing—T.F. Simpson (PTD, Lanc.)|J.P. Wittke (Labs, Pr.) RE-23-6-1

MINI VS MICROPROCESSOR SOFTWARE—K. Schroeder (Labs, Pr.) RE-23-6-22

PERVASIVE COMPUTER, the—P.M. Russo|C.C. Wang (Labs, Pr.) RE-23-6-5

PROGRAMMING in CHIP-8—H. Kleinberg (R&E, Ch.Hl.) RE-23-6-12

RESOURCES AVAILABLE for learning computer science—L. Shapiro (R&E, Ch.Hl.) RE-23-6-10

SOFTWARE COSTS, using PRICE S to estimate—F. Freiman (GSD, Ch.Hl.) RE-23-6-23

SOFTWARE, what is?—S.A. Steele (MSR, Mrstn.) RE-23-6-4

SOFTWARE, why every engineer should be interested in—J.C. Volpe (MSR, Mrstn.) RE-23-6-2

SOFTWARE, why not do it with—L.A. Solomon|D. Block (SSD, Som.) RE-23-6-6

TEST DATA ANALYSIS, a unified approach to—M. Gianfagna (Labs, Som.) RE-23-6-16

Solid State Circuits and Devices

BI-MOS TECHNOLOGY produces a universal op amp—H. Khajezadeh|B.J. Walmsley (SSD, Som) RE-23-3-12

CMOS RELIABILITY—L.J. Gallace (SSD, Som)|H.L. Pujol (SSD, Som.)|E.M. Reiss (SSD, Som)|G.L. Schnable (Labs, Pr.)|M.N. Vincoff (SSD, Som.) RE-23-2-18

CMOS/SIS—a planar process that may improve on SOS—C.E. Weitzel (Labs, Pr.) RE-23-2-9

FUTURE SHOCK in electronics—D. Shore (GSD, Mrstn.) RE-23-1-1

HYBRID TECHNOLOGY—best supporting actor—J.G. Bouchard (AS, Burl.) RE-23-2-12

HYBRID TECHNOLOGY, a circuit designer meets—B.T. Joyce (AS, Burl.) RE-23-2-10

HYBRICS—a look at the total cost—B.T. Joyce (AS, Burl.) RE-23-2-11

HYBRIDS IN MODERN ELECTRONICS SYSTEMS, the changing role of—J.A. Bauer (MSR, Mrstn.) RE-23-2-14

LASER DIODES, current modulation of, gives direct fiber-optic baseband tv—D.R. Patterson (Labs, Pr.) RE-23-2-2

LSI TECHNOLOGY choices—J. Hilibrand (GSD, Mrstn.) RE-23-1-15

MESFET LINEAR AMPLIFIER, 4-GHz, computer optimization produces a high-efficiency—F.N. Sechi|H.C. Huang (Labs, Pr.) RE-23-3-7

SEMICONDUCTORS 1971-1981—ten-year perspective—B.A. Jacoby (SSD, Som.) RE-23-1-3

SUPPLYING MICROCIRCUITS for government end-use: risks and benefits—J. Handen (SSD, Som.) RE-23-1-2

Television Receiver

ELECTRONIC DISPLAYS—E.O. Johnson (Labs, Tokyo) RE-23-1-11

U.S. COLOR TELEVISION FUNDAMENTALS—a review—D.H. Pritchard (Labs, Pr.) RE-23-1-21

XL-100 XTENDEDLIFE CHASSIS, the—P.C. Wilmarth (CE, Ipls.) RE-23-1-17

Author Index (Authors may have more than one article per category.)

Editorial Representatives

Contact your Editorial Representative, at the extensions listed here, to schedule technical papers and announce your professional activities.

Commercial Communications Systems Division

Broadcast Systems
BILL SEPICH* Camden, N.J. Ext. 2156
KRISHNA PRABA Gibbsboro, N.J. Ext. 3605
ANDREW BILLIE Meadow Lands, Pa. Ext. 6231

Mobile Communications Systems
FRED BARTON* Meadow Lands, Pa. Ext. 6428

Avionics Systems
STEWART METCHETTE* Van Nuys, Cal. Ext. 3806
JOHN McDONOUGH Van Nuys, Cal. Ext. 3353

Cablevision Systems
JOHN OVNICK* N. Hollywood, Cal. Ext. 241

Government Systems Division

Astro-Electronics
ED GOLDBERG* Hightstown, N.J. Ext. 2544

Automated Systems
KEN PALM* Burlington, Mass. Ext. 3797
AL SKAVICUS Burlington, Mass. Ext. 2582
LARRY SMITH Burlington, Mass. Ext. 2010

Government Communications Systems
DAN TANNENBAUM* Camden, N.J. Ext. 3081
HARRY KETCHAM Camden, N.J. Ext. 3913

Government Engineering
MERLE PIETZ* Camden, N.J. Ext. 2161

Missile and Surface Radar
DON HIGGS* Moorestown, N.J. Ext. 2836
JACK FRIEDMAN Moorestown, N.J. Ext. 2112

Solid State Division

JOHN SCHOEN* Somerville, N.J. Ext. 6467

Power Devices
HAROLD RONAN Mountaintop, Pa. Ext. 633-827
SY SILVERSTEIN Somerville, N.J. Ext. 6168

Integrated Circuits
FRED FOERSTER Somerville, N.J. Ext. 7452
JOHN YOUNG Findlay, Ohio Ext. 307

Electro-Optics and Devices
RALPH ENGSTROM Lancaster, Pa. Ext. 2503

Consumer Electronics

CLYDE HOYT*Indianapolis, Ind. Ext. 5208
FRANCIS HOLT Indianapolis, Ind. Ext. 5217
PAUL CROOKSHANKS Indianapolis, Ind. Ext. 5080
DON WILLIS Indianapolis, Ind. Ext. 5883

SelectaVision Project

ROBERT MOORE Indianapolis, Ind. Ext. 3313

RCA Service Company

JOE STEOGER* Cherry Hill, N.J. Ext. 5547
RAY MacWILLIAMS Cherry Hill, N.J. Ext. 5986
DICK DOMBROSKY Cherry Hill, N.J. Ext. 4414

Distributor and Special Products Division

CHARLES REARICK* Deptford, N.J. Ext. 2513

Picture Tube Division

ED MADENFORD* Lancaster, Pa. Ext. 3657
NICK MEENA Circleville, Ohio Ext. 228
JACK NUBANI Scranton, Pa. Ext. 499
J.R. REECE Marion, Ind. Ext. 566

Alascom

PETE WEST* Anchorage, Alaska Ext. 7657

Americom

MURRAY ROSENTHAL* Kingsbridge Campus, N.J. Ext. 4363

Globcom

WALT LEIS* New York, N.Y. Ext. 3655

RCA Records

JOSEPH WELLS* Indianapolis, Ind. Ext. 6146

NBC

BILL HOWARD* New York, N.Y. Ext. 4385

Patent Operations

JOSEPH TRIPOLI Princeton, N.J. Ext. 2992

Research and Engineering

Corporate Engineering
HANS JENNY* Cherry Hill, N.J. Ext. 4251

Laboratories
MAUCIE MILLER Princeton, N.J. Ext. 2321
LESLIE ADAMS Somerville, N.J. Ext. 7357

*Technical Publications Administrator, responsible for review and approval of papers and presentations.

RCA Engineer

A technical journal published by Corporate Technical Communications
"by and for the RCA engineer"

Printed in USA

Form No RE-23-6