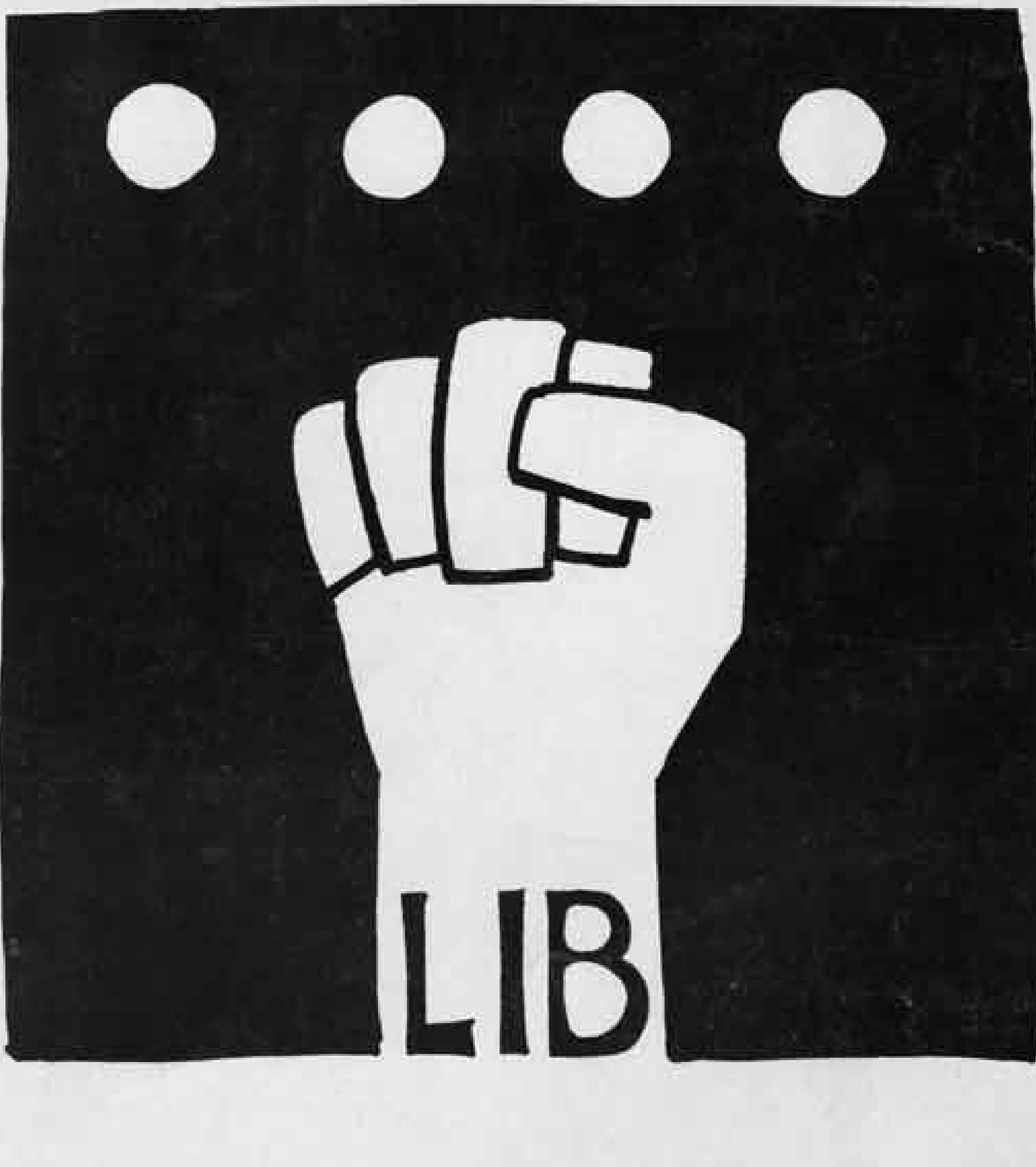
*You can and must understand computers NOW.*

# COMPUTER



# LIB

SEVEN DOLLARS.

First Edition.
ISBN 0-89347-002-3

# COMPUTER LIB



SIXTH PRINTING, MAY 1978
Additional copies available from:
the distributors
702 South Michigan
South Bend, IN 46618
Normal Trade Discounts Available

Any nitwit can understand computers, and many do. Unfortunately, due to ridiculous historical circumstances, computers have been made a mystery to most of the world. And this situation does not seem to be improving. You hear more and more about computers, but to most people it's just one big blur. The people who know about computers often seem unwilling to explain things or answer your questions. Stereotyped notions develop about computers operating in fixed ways--and so confusion increases. The chasm between laymen and computer people widens fast and dangerously.

This book is a measure of desperation, so serious and abysmal is the public sense of confusion and ignorance. Anything with buttons or lights can be palmed off on the layman as a computer. There are so many different things, and their differences are so important; yet to the lay public they are lumped together as "computer stuff," indistinct and beyond understanding or criticism. It's as if people couldn't tell apart camera from exposure meter or tripod, or car from truck or tollbooth. This book is therefore devoted to the premise that

EVERYBODY SHOULD UNDERSTAND COMPUTERS.

It is intended to fill a crying need. Lots of everyday people have asked me where they can learn about computers, and I have had to say nowhere. Most of what is written about computers for the layman is either unreadable or silly. (Some exceptions are listed nearby; you can go to them instead of this if you want.) But virtually nowhere is the big picture simply enough explained. Nowhere can one get a simple, soup-to-nuts overview of what computers are really about, without technical or mathematical mumbo-jumbo, complicated examples, or talking down. This book is an attempt.

(And nowhere have I seen a simple book explaining to the layman the fabulous wonderland of computer graphics which awaits us all, a matter which means a great deal to me personally, as well as a lot to all of us in general. That's discussed on the flip side.)

Computers are simply a necessary and enjoyable part of life, like food and books. Computers are not everything, they are just an aspect of everything, and not to know this is computer illiteracy, a silly and dangerous ignorance.

Computers are as easy to understand as cameras. I have tried to make this book like a photography magazine—breezy, forceful and as vivid as possible. This book will explain how to tell apples from oranges and which way is up. If you want to make cider, or help get things right side up, you will have to go on from here.

I am not a skillful programmer, hands-on person or eminent professional; I am just a computer fan, computer fanatic if you will. But if Dr. David Reuben can write about sex I can certainly write about computers. I have written this like a letter to a nephew, chatty and personal. This is perhaps less boring for the reader, and certainly less boring for the writer, who is doing this in a hurry. Like a photography magazine, it throws at you some rudiments in a merry setting. Other things are thrown in so you'll get the sound of them, even if the details are elusive. (We learn most everyday things by beginning with vague impressions, but somehow encouraging these is not usually felt to be respectable.) What I have chosen for inclusion here has been arbitrary, based on what might amuse and give quick insight. Any bright highschool kid, or anyone else who can stumble through the details of a photography magazine, should be able to understand this book, or get the main ideas. This will not make you a programmer or a computer person, though it may help you talk that talk, and perhaps make you feel more comfortable (or at least able to cope) when new machines encroach on your life. If you can get a chance to learn programming— see the suggestions on p. — it's an awfully good experience for anybody above fourth grade. But the main idea of this book is to help you tell apples from oranges, and which way is up. I hope you do go on from here, and have made a few suggestions.

I am "publishing" this book myself, in this first draft form, to test its viability, to see how mad the computer people get, and to see if there is as much hunger to understand computers, among all you Folks Out There, as I think. I will be interested to receive corrections and suggestions for subsequent editions, if any. (The computer field is its own exploding universe, so I'll worry about up-to-dateness at that time.)

## SUMMARY OF THIS BOOK

Man has created the myth of "the computer" in his own image, or one of them: cold, immaculate, sterile, "scientific," oppressive.

Some people flee this image. Others, drawn toward it, have joined the cold-sterile-oppressive cult, and propagate it like a faith. Many are still about this mischief, making people do things rigidly and saying it is the computer's fault.

Still others see computers for what they really are: versatile gizmos which may be turned to any purpose, in any style. And so a wealth of new styles and human purposes are being proposed and tried, each proponent propounding his own dream in his own very personal way.

This book presents a panoply of things and dreams. Perhaps some will appeal to the reader...

THE COMPUTER PRIESTHOOD

Knowledge is power and so it tends to be hoarded. Experts in any field rarely want people to understand what they do, and generally enjoy putting people down.

Thus if we say that the use of computers is dominated by a priesthood, people who spatter you with unintelligible answers and seem unwilling to give you straight ones, it is not that they are different in this respect from any other profession. Doctors, lawyers and construction engineers are the same way.

But computers are very special, and we have to deal with them everywhere, and this effectively gives the computer priesthood a stranglehold on the operation of all large organizations, of government bureaus, and anything else that they run. Members of Congress are now complaining about control of information by the computer people, that they cannot get the information even though it's on computers. Next to this it seems a small matter that in ordinary companies "untrained" personnel can't get straight questions answered by computer people; but it's the same phenomenon.

It is imperative for many reasons that the appalling gap between public and computer insider be closed. As the saying goes, war is too important to be left to the generals. Guardianship of the computer can no longer be left to a priesthood. I see this as just one example of the creeping evil of Professionalism,* the control of aspects of society by cliques of insiders. There may be some chance, though, that Professionalism can be turned around. Doctors, for example, are being told that they no longer own people's bodies.** And this book may suggest to some computer professionals that their position should not be as sacrosanct as they have thought, either.

This is not to say that computer people are trying to louse everybody up on purpose. Like anyone trying to do a complex job as he sees fit, they don't want to be bothered with idle questions and complaints. Indeed, probably any group of insiders would have hoarded computers just as much. If the computer had evolved from the telegraph (which it just might have), perhaps the librarians would have hoarded it conceptually as much as the math and engineering people have. But things have gone too far. People have legitimate complaints about the way computers are used, and legitimate ideas for ways they should be used, which should no longer be shunted aside.

In no way do I mean to condemn computer people in general. (Only the ones who don't want you to know what's going on.) The field is full of fine, imaginative people. Indeed, the number of creative and brilliant people known within the field for their clever and creative contributions is considerable. They deserve to be known as widely as, say, good photographers or writers.

"Computers are catching hell from growing multitudes who see them uniformly as the tools of the regulation and suffocation of all things warm, moist, and human. The charges, of course, are not totally unfounded, but in their most sweeping form they are ineffective and therefore actually an acquiescence to the dehumanization which they decry. We clearly need a much more discerning evaluation in order to clarify the ethics of various roles of machines in human affairs."

Ken Knowlton
in "Collaborations with Artists—
a Programmer's Reflections"
in Nake & Rosenfeld, eds.,
Graphic Languages
(North-Holland Pub. Co.),
p. 399.

* This is a side point. I see Professionalism as a spreading disease of the present-day world, a sort of poly-oligarchy by which various groups (subway conductors, social workers, bricklayers) can bring things to a halt if their particular new increased demands are not met. (Meanwhile, the irrelevance of each profession increases, in proportion to its increasing rigidity.) Such lucky groups demand more in each go-round— but meantime, the number who are permanently unemployed grows and grows.

** Ellen Frankfort, Vaginal Politics. Quadrangle Books. Boston Women's Health Collective, Our Bodies, Ourselves. Simon & Schuster.

A. Thibault Amicalement R. Nel. 8.4.87



This side of the book, Computer Lib proper (whose title is nevertheless the simplest way to refer to both halves), is an attempt to explain simply and concisely why computers are marvelous and wonderful, and what some main things are in the field.

The second half of the book, Dream Machines, is specially about fantasy and imagination, and new techniques for it. That half is related to this half, but can be read first; I wanted to separate them as distinctly as possible.

The remarks below all refer to this first half, the Computer Lib half of the book.

FANDOM

With this book I am no longer calling myself a computer professional. I'm a computer fan, and I'm out to make you one. (All computer professionals were fans once, but people get crabbier as they get older, and more professional.) A generation of computer fans and hobbyists is well on its way, but for the most part these are people who have had some sort of an In. This is meant to be an In for those who didn't get one earlier.

The computer fan is someone who appreciates the options, fun, excitement, and fiendish fascination of computers. Not only is the computer fun in itself, like electric trains; but it also extends to you a wide variety of possible personal uses. (In case you don't know it, the price of computers and of using them is going down as fast as every other price is going up. So in the next few decades we may be reduced to eating soybeans and carrots, but we'll certainly have computers.)

Somehow the idea is abroad that computer activities are uncreative, as compared, say, with rotating clay against your fingers until it becomes a pot. This is categorically false. Computers involve imagination and creation at the highest level. Computers are an involvement you can really get into, regardless of your trip or your karma. They are toys, they are tools, they are glorious abstractions. So if you like mental creation, toy trains, or abstractions, computers are for you. If you are interested in democracy and its future, you'd better understand computers. And if you are concerned about power and the way it is being used, and aren't we all right now, the same thing goes.

THE SOCIETY

Which brings us to our next topic.

There is no question of whether the computer will remake society; it has. You deal with computers perhaps many times a day— or worse, computers deal with you, though you may not know it. Computers are going into everything, are intertwined with everything, and it's going to get more and more so. The reader should have a sense of the dance of options, the remarkably different ways that computers may be used; by extension, he should come to see the extraordinary range of options which confront us as a society in our future use of them. Indeed, computers have with a swoop expanded the options of everything.

But a variety of inconvenient systems already touch on our lives, nuisances we must deal with all the time; and I fear that worse is to come. I would like to alert the reader, in no uncertain terms, that the time has come to be openly attentive and critical in observing and dealing with computer systems; and to transform criticism into action. If systems are bad, annoying and demeaning, these matters should be brought to the attention of the perpetrators. Politely at first. But just as the atmospheric pollution fostered by GM has become a matter for citizen concern and attack through legitimate channels of protest, so too should the procedural pollution of inconsiderate computer systems become a matter for the same kinds of concern. The reader should realize he can criticize and demand;

THE PUBLIC DOES NOT HAVE TO TAKE
WHAT'S BEING DISHED OUT.

There is already a backlash against computers, and the spirit of this anti-computer backlash is correct, but should be directed against very specific kinds of things. The public should stop being mad at "computers" in the abstract, and start being mad at the people who make inconvenient systems. It is not "the computer," which has no intrinsic style or character, which is at fault; it is people who use "the computer" as an excuse to inconvenience you, who are at fault. The mechanisms of legitimate public protest— sit-ins and so on— should perhaps soon be turned to complaint over bad and inhuman computer systems.

The question is, will the crummier trends continue? Or can the public learn, in time, what good and beautiful things are possible, and translate this realization into an effective demand? I do not believe this is an obscure or specialized issue. Its shadow falls across the future of mankind, if any, like a giant sequoia. Either computer systems are going to go on inconveniencing our lives, or they are going to be turned around to make life better. This is one of the directions that consumerism should turn.

I have an axe to grind: I want to see computers useful to individuals, and the sooner the better, without necessary complication or human servility being required. Anyone who agrees with these principles is on my side, and anyone who does not, is not.

THIS BOOK IS FOR PERSONAL FREEDOM,
AND AGAINST RESTRICTION AND COERCION.

That's really all it's about. Many people, for reasons of their own, enjoy and believe in restricting and coercing people; the reader may decide whether he is for or against this principle.

A chant you can take to the streets:

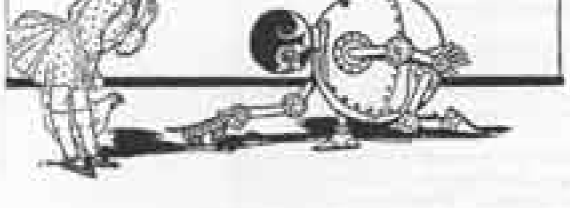
COMPUTER POWER TO THE PEOPLE!
DOWN WITH CYBERCRUD!

THE FUTURE, IF ANY

Simply as a matter of citizenship, it is essential to understand the impact and uses of computers in the world of the future, if any; and to have a sense of the issues about computers that confront us as a people— especially privacy and data banks, but also strange new additions to our economic system ("the checkless society"), our political system (half-baked vote-at-home proposals), and so on. I regret that there is not room to cover these here.

Various companies are seeking wide public support for the sorts of things they are trying to bring about. Legislation will be proposed on which the views of the public should have a bearing. It is important that these be understood sensibly by some part of the electorate before they are made too permanent, rather than made matters of dumb assent.

Finally, and most solemnly, computers are helping us understand the unprecedented danger of our future (see "The Club of Rome," p.43). The human race may have only a short time left on earth, even if there is no war. These studies must be seen and understood by as many intelligent men of good will as possible.



THEREFORE

Welcome to the computer world, the damndest and craziest thing that has ever happened. But we, the computer people, are not crazy. It is you others who are crazy to let us have all this fun and power to ourselves.

COMPUTERS BELONG TO ALL MANKIND.

AUTHOR'S CREDENTIALS

B.A., philosophy, Swarthmore; graduate study U. of Chicago; M.A., sociology, Harvard. Mostly self-taught in computers.
Member of editorial board, Computer Decisions magazine; listed in New York Times' Who's Who in Computers; member of Association for Computing Machinery since 1964.

Research assistant, Communication Research Institute, 1962-3. Instructor in sociology, Vassar College, 1964-6.
Senior staff researcher, Harcourt, Brace & World Publishers, 1966-7. Consultant to Bell Telephone Laboratories, Whippany, N.J., 1967-8.
Consultant to CBS Laboratories, Stamford, Ct., 1968-9. Proprietor of The Nelson Organization, Inc., New York City, 1968-72.
Lecturer in art, instructional resources and computer science, U. Illinois at Chicago Circle, 1973-6.
Co-founder of the Itty Bitty Machine Co. computer store, Evanston, Illinois, 1976.
Venture Fund lecturer, Swarthmore College, spring 1977.

PHOTO BY ROGER FIELD.

# WHERE IT'S AT

Computers are where it's at.

Recently a bank employee was accused of embezzling a million and a half dollars by clever computer programming. His programs shifted funds from hundreds of people's accounts to his own, but apparently kept things looking innocent by clever programming tricks. According to the papers, the program kept up appearances by redepositing the stolen amount in each account just as interest payments were about to be calculated, then withdrawing it again just after. ("Chief Teller Is Accused of Theft of $1.5 Million at a Bank Here." New York Times, 23 March 73, p. 1.) The alleged embezzlement was discovered, not by bank audit, but by records found on the premises of a raided bookmaker.

In a recent scandal that has rocked the insurance world, an insurance company appears to have generated thousands of fictitious customers and accounts by computer, then bilked other insurance companies-- those who re-insured the original fictitious policies-- by fictitious claims on the fictitious misfortunes of the fictitious policy-holders.

In April of 1973, according to the Chicago radio, a burglary ring had a "computerized" list of a thousand prospective victims.

There have been instances where dishonest university students, nevertheless able programmers, were able to change their course grades, stored on a central university computer.

It is not unheard of for ace programmers to create grand incomprehensible systems that run whole companies, systems they can personally play like a piano, and then blackmail their firms.

A friend of a friend of the author is an ace programmer at the Pentagon, supposedly a private supervising colonels. On days he is mad at his boss, he says, the army cannot find out its strength within 300,000 men. Or three million if he so chooses.

This awkward state of affairs, obviously spanning both the American continent and most realms of endeavor, has come about for various reasons.

First, the climate of uncomprehension leads men in management to treat computer matters as "mere technicalities"-- a myth as sinister as the public notion that computers are "scientific"-- and abandon the kind of scrutiny they sensibly apply to any other company activities.

Second, most of today's computer systems are inherently leaky and insecure-- and likely to stay that way awhile. Getting things to work on them involves giving people extraordinary and invisible powers. (Eventually this will change, but watch out for the meantime.)

The obvious consequence is simply for the computer people to be allowed to take over altogether. It may indeed be that computer people -- the more well-informed and visionary ones, anyway-- can see the farthest, and appreciate most deeply the better ways things can go, and the steps that have to be taken to get there. (And Boards of Managers can at least be partially assured that hanky-panky at the lower levels will be prevented, if men in charge know where the bodies are buried.)

That seems to be how it's going. Examples:

The president of Dartmouth College, John Kemeny, is a respected computerman and a developer of one of the important computing languages, BASIC (see p. [illegible]).

The new president of the Russell Sage Foundation, Hugh Cline, used to teach computing at Columbia.

It's probably the same in industry. In other words, more and more, for better and for worse, things are being run by people who know how to use computers, and this trend is probably irreversible.

In some ways, of course, this is a sinister portent. In private industry it's not so bad, since the danger is more of embezzlement and botch-up than of public menace. But then there's the problem of the government. The men who manage the information tools are more and more in charge of government, too. And if we can have a Watergate without computers, just wait. (See "Burning Issues," p. [illegible])

The way to defend ourselves against computer people is to become computer people ourselves. Which of course is the point. We must all become computer people, at least to the extent that we have already become Automobile People and Camera People-- that is, informed enough to tell when one goes by or when someone points one at you.

MANY MANSIONS

The future is going to be full of computers, for good or ill. Many computer systems are being prepared by a variety of lunatics, idealists and dreamers, as well as profit-hungry companies and unimaginative clods, all for the benefit of mankind. Which ones will work and which ones we will like is another matter. The grand and dreamy ones bid fair to reorganize drastically the lives of mankind.

For instance, Doug Engelbart at Stanford Research Institute has a beautiful system, called NLS, that will allow us to use computers as a generalized postoffice and publication system. From your computer terminal you just sign onto Engelbart's System, and you're at once in touch with lots of writings by other subscribers, which you may call to your screen and write replies to.

(These grander and dreamier applications are discussed on the other side of this book.)

But the plain computer visions are grand enough.

The great world of time-sharing, for instance. ("Time-sharing" means that the computer's time is shared by a variety of users simultaneously. See p. [illegible].) If you have an account on a time-sharing computer, you can sign on from your terminal (see p. [illegible]) over any telephone, no matter where you are, and at once do anything that particular computer allows-- calling up programs in a variety of computer languages, dipping into data on a variety of subjects as easily as one now consults a chart.

For instance, at Dartmouth College-- where time-sharing is perhaps farthest advanced as a way of life-- the user (any Dartmouth student, for instance) can just sit down at a terminal and write a simple program (in Dartmouth's BASIC language, for instance) to analyze census data. Since Dartmouth has a complete file on its time-sharing system of the detailed sample from the 1970 census, the program can buzz through that and report almost immediately the numbers of divorced Aleuts or boy millionaires in the sample, or (more significantly) the relative incomes of different ethnic groups when categorized according to the questioner's interests.

But simple time-sharing is only the beginning. Networks of computers are now coming into being. Most significant of these is the ARPANET (financed by ARPA, the Defense Department's Advanced Research Projects Agency, it is nonetheless non-military in character). Dozens of large time-sharing computers around the country are being tied into the Arpanet, and a user of any of these can reach directly into the other computers of the network-- using their programs, data or other facilities. Arpanet enthusiasts see this as the wave of the future.

MINI MANSIONS

But while computers and their combinations grow bigger and bigger, they also grow smaller and smaller. A complete computer the size of an Oreo cookie is now available, guaranteed for twentyfive years (and very expensive). But its actual heart, the Intel microprocessor, is only sixty bucks now, and just wait (see Microprocessors, p. [illegible]). By 1980 there should be as many programmed and programmable objects in your house as you now have TVs, radios and typewriters; that's a conservative estimate. But just what these devices will all be doing-- ah, there's the question that has many people talking to themselves.

OTHER COMING THINGS?

There are a lot of tall stories about what computers will do for the world. Among the most threatening, I think, are glowing reports of "scientific" politics (don't you believe it). We hear how computers will bring "science" to government, helping, for example, to redraw the lines of election districts. (See Cybercrud, p. [illegible].)

Then you may also have heard that computers are going to be our new mentors and companions, tutoring us, chatting with us and perhaps lulling us to sleep-- like Hal in 2001. Worried? Good. (See "The God-Builders," flip side.) (P. [illegible])

# CHUTZPAH DEPARTMENT

A college student broke through the security of the Pacific Telephone computer system from a terminal and, according to Computerworld (6 June 73), stole over a million dollars worth of equipment by ordering it delivered to him! (Penthouse, December 73, claims he was in highschool and it was only nine hundred thousand, but you get the idea.)

After serving a few weeks in jail, he has formed his own computer-security consulting company.

More power to him.

The new breed has got to be watched.

This is the urgency of this book. Remember that the man who writes the payroll program can write himself some pretty amazing checks-- perhaps to be mailed out to Switzerland, next year.

From here on it's computer politics, computer dirty tricks, computer wonderlands, computer everything.

For anyone concerned to be where it's at, then, this book will provide a few suggestions. Now is the time you either know or you don't.

Enough power talk. Knowledge is power. Here you go. Dig in.

# LESSON 1: GETTING THINGS STRAIGHT

The greatest hurdle for the beginner (or "layman") is making an effort to grasp particulars of that which he hears about.

A. WHAT IS ITS NAME? Every system or proposal or project has a name of some sort. Make an effort to learn it, or you're stuck trying to refer to "that computerish thing."

(And don't be a snob about acronyms, those all-cap names and terms sprung from the foreheads of other words, like ILLIAC and PLATO and CAI. There's a need for them. Short words are too general to use for names, and long phrases are too unwieldy.)

B. IN WHAT PARTICULAR WAY DOES IT EMPLOY THE COMPUTER? For record-keeping? For looking stuff up quickly or fancily? For searching out combinations? For making up combinations and testing their properties? For enacting complex phenomena? As automatic typewriters? To play music, or just to store the written notes?

It is hoped that you will become sensitive to these distinctions, and be able to understand and remember them after somebody explains them.

Otherwise you're stuck just referring to "that computer business," and you're in with the rest of the sheep.

(Incidentally —)

People ask me often where they can learn about "science." As in all fields, magazines are usually the best sources of general orientation.

Science Digest is kind of helpful for a start, although unfortunately they print summaries of every fool study that generalizes to the hearts of all humanity from two dozen Iowa State freshmen.

Scientific American is the favorite. Some stuff is hard to read but some isn't; the pictures and diagrams are terrific.

Science & Technology magazine seems to me one of the better ones-- breezy, informative, not trivial.

Science magazine is read by most actual scientists, and if you have a lively curiosity and can guess at the meanings of words, will tell you an incredible amount. (This is a main source for the science articles in the New York Times, which in turn...) Their articles on politics of science, and the future, are very interesting, important, and depressing. You have to join Am. Assn. for the Advancement of Science, Washington, D.C.

Daniel S. Greenberg's Science and Government Report (sorry-- $35 a year) is what really tells it. Greenberg is the man who knows, both what is shaping up in science and the insane governmental confusions and floundering responses and grandstanding and pork-barrel initiatives...

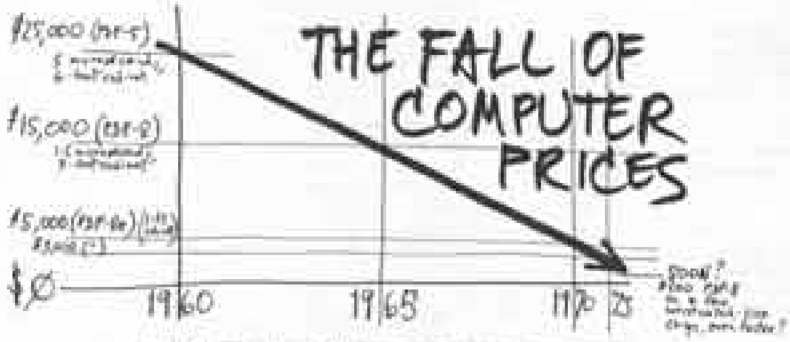
Greenberg is, incidentally, one of the finest writers of our time and a great humorist.

Science and Government Report, Kalorama Station (really?), Box 21123, Washington, D.C. 20009.

This is the wall that the handwriting is on.

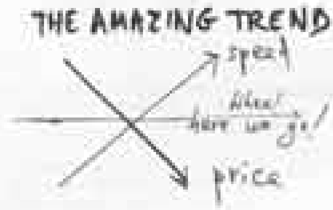
## ASPECTS OF THIS BOOK

The explanations-- not yet fully debugged-- are intended for anybody. The listings of expensive products and services are intended not only as corroborative detail, for a general sense of what's available, but also for business people who might find them helpful, for affluent individuals and clubs who want to try their hand, and finally as a box score of how the prices are coming down. Because we are all going to be able to afford these things pretty soon.




This diagram shows the amazing and unique way prices drop in the computer field. The prices shown are for the first minicomputer, the PDP-5 (and its hugely popular offspring, the PDP-8); but the principle has held throughout the field, and the downward trend will probably accelerate due to the new big integrated circuits.

Another example: an IBM 7090, a very decent million-dollar computer in 1960, was put up for sale at a modish Parke-Bernet "used computer auction" in 1970. If I remember aright, they could not get a $1000 bid, because today's machines are so much smaller, faster and more dependable.




## WHERE IT'S AT IN THIS BOOK



## WHERE IT'S AT, U.S.A.


THE BUCK STOPS HERE

Everywhere in the world people can pretend that your ignorance, or position, or credentials, or poverty, or general unworthiness, are the reasons you are being pushed around or made to feel small. And because you can't tell, you have to take it.

And of course we can do the same thing with computers. Yes, we can do it in spades. (See "Cybercrud," p. 8.) But many of us do not want to. There has to be a better way. There has to be a better world.

# YOUR INFORMATION SOURCES

There are several major places you get information in the computer field: friends, magazines, bingo cards, conferences and conference proceedings.

FRIENDS.

Friends we can't help with. But you might make some at conferences. Or join a computer club?

MAGAZINES.

The principal magazines are (first few listed roughly by degree of general interest):

Datamation. $15 a year or free. The main computer magazine, a breezy, clever monthly. Lots of ads, interesting articles the layman can read with not much effort. Twits IBM.

Subscriptions are $15 if you're not a computer person, free if you are. Datamation, 35 Mason St., Greenwich CT 06830.

Computer Decisions. Some $7 a year or free. Some nice light articles, as well as helpful review articles on different subjects. Avoids technicalities. Computer Decisions, 50 Essex St., Roselle Park NJ 07662.

Computers and Automation. Avoids technicalities but quite a bit of social-interest stuff. Nobody gets it free; something like $7.50 a year. Berkeley Enterprises, Inc., 815 Washington St., Newtonville, Mass. 02160.

Computerworld (actually a weekly tabloid paper). Not free: $9 a year. More up-to-the-minute than most people have time to be. Computerworld, Circ. Dept., 797 Washington St., Newton, Mass. 02160.

Computing Surveys. Excellent, clearly written introductory articles on a variety of subjects. Any serious beginner should definitely subscribe to Computing Surveys. (See ACM, below.)

Communications of the ACM. High-class journal about theoretical matters and events on the intellectual side of the field. (See ACM, below.) → "CACM"

Computer Design. $18/yr. or free. Concentrates on parts for computers, but also tells technical details of new computers and peripherals. Computer Design, Circulation Dept., P.O. Box A, Winchester, Mass. 01890.

Data Processing magazine. Oriented to conventional business applications of computers. $10. North American Publishing Co., 134 N. 13th St., Philadelphia, Pa. 19107.

Computer. (Formerly IEEE Computer Group News.) $12/yr. Thoughtful, clearly written articles on high-level topics. Quite a bit on Artificial Intelligence (see flip side). IEEE Computer Society, 16400 Ventura Blvd., Encino CA 91316.

Here are some other magazines that may interest you. No particular order.

PCC. Delightful educational/counterculture tabloid emphasizing computer games and fun. Oriented to BASIC language. $4/yr. from People's Computer Company, P.O. Box 310, Menlo Park, CA 94025.

Computing Reviews. Prints reviews, by individuals in the field, of most of the serious computer articles. Useful, but subject to individual biases and gaps. (See ACM, below.)

The New Educational Technology. $5/yr. Presumably concentrates on activities of its publisher: General Turtle, Inc., 545 Technology Square, Cambridge, MA 02139; wonderful computer toys for schools and the well-heeled.

The Honeywell Computer Journal. Something like $10 a year. Honeywell Information Systems, Inc., Phoenix, Arizona. Showcase magazine of miscellaneous content; readable, nicely edited. Has unusual practice of including microfiche (microfilm card) of entire issue in a pocket.

IBM Systems Journal. Showcase technical journal of miscellaneous content, especially arcana about IBM products. $5/yr. IBM, Armonk, NY 10504.

IBM Journal of Research and Development. Showcase technical journal of miscellaneous content. $7.50/year. IBM, Armonk, NY 10504.

Journal of the ACM. A highly technical, math-oriented journal. Heavy on graph theory and pattern recognition. (See ACM, below.) ("JACM")

Digital Design. $15 or free. About computer parts and designs. Digital Design, Circ. Dept., 167 Corey Road, Brookline, Mass. 02146.

Infosystems. Aspiring mag. $20 or free. Hitchcock Publications, P.O. Box 3007, Wheaton, Ill. 60187.

Think. This is the IBM house organ. Presumably free to IBM customers or prospects. IBM, Armonk, NY 10504.

There are also expensive (snob?) magazines, bought by executives.

Computer Age. $95/yr. EDP News Services Inc., 514 10th St. N.W., Washington DC 20004.

Computer Digest. $36/yr. Information Group, 1309 Cherry St., Philadelphia PA 19107.

Data Processing Digest. $51/yr. 6820 la Tijera Blvd., Los Angeles CA 90045.

Hey now, here's a magazine called Computopia. Only $15 a year. Unfortunately in Japanese. Computer Age Co. Ltd., Kasumigaseki Bldg., Box 122, Chiyoda-Ku, Tokyo, Japan.

# "COMPUTER TOYS" — A WARNING

A number of inexpensive gadgets purport to teach you computer principles. Many people have been disappointed, or worse, made to feel stupid, when they learn nothing from these. Actually the best these things really can do is give you an idea of what can be done with combinations of switches. From that to learning what computer people really think about is a long, long way.

# SOME GOOD BOOKS & ARTICLES FOR BEGINNERS

The best review of what's happening lately, by none other than Mr. Whole Earth Catalog himself: Stewart Brand, "Spacewar: Fanatic Life and Symbolic Death among the Computer Bums." Rolling Stone, 7 December 72, 50-56. He visited the most hotshot places and reports especially on the fun-and-games side of things.

Gilbert Burck and the Editors of Fortune, The Computer Age. Harper and Row. Ignore the ridiculous full title, The Computer Age and Its Potential for Management; this book has nothing to do with management, but is a nice general orientation to the field.

Thomas H. Crowley, Understanding Computers. McGraw-Hill. This is the most readable and straightforward introduction to the technicalities around.

Jeremy Bernstein, The Analytical Engine. Random House, 1964. History of computers, well told, and the way things looked in 1964, which wasn't really very different.

Donald E. Knuth, The Art of Programming. (7 vols.) A monumental series, excellently written and widely praised, for anyone who wants to dig in and be a serious programmer. Three of the seven volumes are out so far, at about twenty bucks apiece. Vol. 1: Fundamental Algorithms. Vol. 2: Seminumerical Algorithms. Vol. 3: Sorting and Searching. Addison-Wesley.

BUMMERS

This is perhaps a minority view, but I think any introduction to computers which makes them seem intrinsically mathematical is misleading. Historically they began as mathematical, but now this is simply the wrong way to think about them. Same goes for emphasizing business uses as if that were all.

We will not name here any of the various disagreeable pamphlets and books which stress these aspects and don't make things very clear.

⇨ ABOUT FREE SUBSCRIPTIONS. Many of the magazines are free to "qualified" readers, usually those willing to state on a signed form that they influence the purchase of computers, computer services, punch cards, or the like. (They ask other questions on the form, but whether you influence purchase is usually what decides whether they send you the magazine.) It is also helpful to have a good-sounding title or company affiliation.

BINGO CARDS.

These are little postcards you find in all the magazines except the ACM and company ones. Fill in your name and an attractive title ("Systems Consultant" or "consultant" is good— after all, someday someone may ask your advice) and circle the numbers corresponding to the ads that entice you. You'll be flooded with interesting, expensively printed, colorful, educational material on different people's computers and accessories. And note that senders don't lose: any company wants its products known.

However, a postoffice box is good, as it helps to avoid calls at home from salesmen, wasting their time as much as yours. If you are in a rural-type area where you can assume a company name with no legal difficulties, so much the better.

# POPULAR COMPUTERS

That the field has not been popularized by its better writers may simply come from an honest doubt that ordinary people can understand computers.

I dispute that. Through magazines, millions of Americans have learned about photography. Through the popular science-and-mechanics type magazines, and more recently the electronics magazines, various other technical subjects have become widely understood.

So far nobody has opened up computers. This is a first attempt. If this book won't do it another one will.

And you better believe that Popular Computers magazine is not very far away. Soon a fully-loaded minicomputer will cost less than the best hi-fi sets. In a couple of years, thousands of individuals will own computers, and millions more will want to. Look out, here we go.

↓

Woops, here it is. Popular Computing, $15 a year ($12 if prepaid), Box 272, Calabasas, CA 91302.

THE MAIN COMPUTER ORGANIZATIONS

ACM, the Association for Computing Machinery. This is the main computer professional society; the title only has meaning historically, as many members are concerned not with machinery itself, but with software, languages, theories and so on.

If you have any plans to stick with the subject, membership in the Association for Computing Machinery is highly recommended. ACM calls itself "The Society of the Computing Community." Thus it properly embraces both professionals and fans.

Dues for official students are $8 a year, $35 for others, which includes a subscription to Communications of the ACM, the official mag. Their address for memberships and magazines is ACM, P.O. Box 12105, Church St. Station, New York, NY 10249. (The actual ACM HQ is at 1133 Ave. of the Americas, New York, N.Y. 10036.)

They have stacked the deck so that if you want to subscribe to any ACM magazines you'd better join anyway. Here are the year prices:

| | Member | Non-Member |
|---|---|---|
| Communications of the ACM | free | $35 |
| Computing Surveys | $7 | $25 |
| Computing Reviews | $12.50 | $35 |
| Journal of the ACM | $7 | $30 |

The one drawback to joining the ACM is all the doggoned mailing lists it gets you on. It's unclear whether there's anything you can do to prevent this, but there oughta be.

SIGs and SICs. For ACM members with special interests (and we all have them), the ACM contains subdivisions-- clubs within the club, of people who keep in touch to share their interests. These are called SICs (Special Interest Committees) and SIGs (Special Interest Groups). There are such clubs-- SICs and SIGs-- in numerous areas, including Programming Languages, Computer Usage in Education, etc. Encouraging these subinterests to stay within ACM saves a lot of trouble for everybody and keeps ACM the central society.

AFIPS.

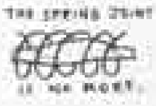
AFIPS is the UN of computing. They sponsored the Joints, and now sponsor the NCC. Just as individuals can't join the UN, they can't join AFIPS, which stands for American Federation of Information Processing Societies. Depending on your special interests, though, you can join a member society.

The constituent societies of AFIPS are, as of June 1973: (If any turn you on, write AFIPS for addresses: AFIPS, 210 Summit Ave., Montvale NJ 07645.)

☆ ACM: the Association for Computing Machinery.
IEEE, the Institute of Electrical and Electronics Engineers. This is the professional society of electronics guys.
Simulation Councils. This is the professional society for those interested in Simulation (see p.55).
Association for Computational Linguistics. (Where language and computer types gather.)
American Association of Aeronautics and Astronautics.
American Statistical Association.
Instrument Society of America.
Society for Information Display. (See flip side.)
American Institute of Certified Public Accountants.
American Society for Information Science. (This group is mainly for electronified librarians and information retrieval types-- see flip side.)
Society for Industrial and Applied Mathematics.
Special Libraries Association.
Association for Educational Data Systems.

IFIP. This is the international computer society. Like AFIPS, its members are societies, so joining ACM makes you an IFIP participant.

IFIP holds conferences around the world. Fun. Expense.




CONFERENCES.

Conferences in any field are exciting, at least till you reach a certain degree of boredom with the field. Computer conferences have their own heady atmosphere, compounded of a sense of elitism, of being in the witches' cauldron, and the sure sense of the impact everything you see will have as it all grows and grows. Plus you get to look at gadgets.

Usually to go for one day doesn't cost much, and at the bigger ones you get lots of free literature, have salesmen explain their things to you, see movies, hear fascinating (sometimes) speakers.

THE JOINTS! The principal computer conferences have always been the Spring Joint Computer Conference, held in an Eastern city in May, and the Fall Joint Computer Conference, held in a Western city in November (the infamous Spring Joint and Fall Joint, or SJCC and FJCC). In 1973, because of poor business the previous year, the two were collapsed into one National Computer Conference (NCC) in June (Universal Joint?) The Joints have always been sponsored by AFIPS (see below). The National Computer Conference will henceforth be annual, at least for a while.

The cost of attending is high-- while it's just a couple of dollars to look at the exhibits, this rises to perhaps fifteen dollars to go to the day's technical sessions or fifty for the week (not counting lodging and eats)-- but it's very much worth it. The lower age limit for attendees is something like twelve, unfortunately for those with interested children.

Other important conferences: the annual ACM conference in the summer; BEMA (Business Equipment Mfrs. Assn.) in the fall and spring (no theory, but lots of gadgets); and other conferences on special subjects, held all the time all over. Lists of conferences and their whereabouts are in most of the magazines: Communications of the ACM and Computer Design have the biggest lists.



CONFERENCE PROCEEDINGS. (such as "Proc ACM 65," "Proc SJCC 68," "Proc NCC 73.")

As you may know, conferences largely consist of separate "sessions" in which different people talk on specific topics, usually reading out loud from their notes and showing slides.

Conference proceedings are books which result from conferences. Supposedly they contain what each guy said; in practice people say one thing and publish another, more formal than the actual presentation.

This leads to a curious phenomenon at the main computer conferences (SJCC, FJCC, ACM and now NCC). When you register they give you a book (you're actually paying perhaps $15 for it), containing all the papers that are about to be given, nicely tricked out by their authors. If you rush to a corner and look at the book it may change your notion of which sessions to go to.

Anyway, the resulting volumes of conference proceedings are a treasure trove of interesting papers on an immense variety of computerish and not-so-computerish subjects. Great for browsing. Expensive but wonderful. (Horrible when you're moving, though, as they are big and heavy.)

JOINT PROCEEDINGS. Proceedings for the Spring Joint and Fall Joint, from the fifties to 1972, are available from AFIPS Press, as are proceedings of the 1973 NCC. (AFIPS Press, 210 Summit Avenue, Montvale NJ 07645.) They cost $20-26 each after the conference is over; less in microfilm. (At the Joint Conferences, AFIPS Press often gives discounts, at their booth, on back Joint proceedings.)

☞ If you want to spend money to learn about the field, Proceedings of the Joint Conferences are a fine buy.

Back ACM Proceedings. From the ACM.

Other Proceedings. Often sold at counters at conferences. Or available from various publishers. Join the ACM and you'll find out soon enough.

TRY TO GET TO THE NATIONAL JOINT. Just as every Muslim should go to Mecca, every computer fan should go to a National Joint (National Computer Conference, or NCC). The next two are (check the magazines):

May 1974, Chicago
May 1975, ~~San Francisco~~ ANAHEIM.

NO QUALIFICATIONS ARE NEEDED. Think of it as a circus for smart alecks, or, if you prefer, a Deep Educational Experience.

# WHAT HAPPENS IF YOU TAKE COMPUTER COURSES?

There is a lot of talk about "best" ways of teaching about computers, but in most places the actual alternatives open to those who want to learn are fairly dismal.

Universities. Universities and colleges tend to teach computing with a mathematical emphasis at the start. Indeed, most seem to require that to get into the introductory computer course, you must have had higher math (at least calculus, sometimes matrix algebra as well). This is preposterous, like requiring an engineering degree to drive a car. (Gradeschool kids can learn to program with no prerequisites.)

☞ It seems to be to cut down enrollment, since they're not set up to deal with all those people who want to learn about computers. (And why not?) Also it's a status thing: as if this restriction somehow should keep enrollment to students with "logical minds," whatever those are, or "mathematical sophistication," as if that were relevant.

"Computer schools," community and commercial colleges, on the other hand, tend to prepare students only for the most humdrum business applications-- keypunching (which is rapidly becoming obsolete), and programming in the COBOL language on IBM business systems. This gets you no closer to the more exciting applications of computers than you were originally.

Some experimental trends are more encouraging. Some colleges, for instance, offer "computer appreciation courses," with a wider introduction to what's available and more varied programming intended to serve as an introduction to this wider horizon.

Highschool courses seem to be cutting through the junk and offering students access to minicomputers with quickie languages, usually BASIC. Both Digital Equipment Corp. and Hewlett-Packard seem to be making inroads here.

Kiddie setups, rumored to exist in Boston and San Francisco, are geared to letting grade-school children see and play with computers. Also one company (General Turtle, see p.57) is selling computer toys intended to encourage actual programming by children.

# CYBERCRUD

A number of people have gotten mad at me for coining the term "cybercrud," which I define as "putting things over on people using computers." But as long as it goes on we'll need the word. At every corner of our society, people are issuing pronouncements and making other people do things and saying it's because of the computer. The function of cybercrud is thus to confuse, intimidate or pressure. We have all got to get wise to this if it is going to be curtailed.

Cybercrud takes numerous forms. All of them, however, share the patina of "science" that computers have for the layman.

1a) COMPUTER AS MAGIC WORD

The most delicate, and seemingly innocent, technique is the practice of naming things so as spuriously to suggest that they involve computers. Thus there is a manufacturer of pot-pipes with "Data" in its name, and apparently a pornography house with a "Cyber-".

1b) COMPUTER AS MAGIC INGREDIENT

The above seems silly, but it is no less silly than talking about "computer predictions" and "computer studies" of things. The mere fact that a computer is involved in something has no bearing on its character or validity. The way things are done with computers affects their character and validity, just like the way things are done without computers. (Indeed, merely using a computer often has no bearing on the way things are done.)

This same technique is easily magnified to suggest, not merely that something involves computers, but is wholly done by computers. The word "computerize" performs this fatal function. When used specifically, as in computerize the billing operation, it can be fairly clear; but make it vague, as in computerize the office, and it can mean anything.

"Fully computerize" is worse. Thus we hear about a "fully computerized" print shop, which turns out to be one whose computers do the typesetting; but they could also run the presses, pay the bills and work the coffee machine. For practical purposes, there is no such thing as "fully" computerized. There is always one more thing computers could do.

BY THE AID OF THE MIRROR SHE PUT ON THE HEAD

2) WHITE LIES: THE COMPUTER MADE ME DO IT

Next come all the little white lies about how such-and-such is the computer's fault and not your decision. Thus the computer is made a General Scapegoat at the same time it's covering up for what somebody wants to do anyway.

"It has to be this way."

"There's nothing we can do; this is all handled by computer."

"The computer will not allow this."

"The computer won't let us."

The translation is, of course, THE STINKY LOUSY PROGRAM DOES NOT PERMIT IT. Which means in turn: WE DO NOT CHOOSE TO PROVIDE, IN OUR PROGRAMS AND EQUIPMENT, ANY ALTERNATIVES.

Now, it is often the case that good and sufficient reason exists for the way things are done. But it is also often the case that companies and the public are inconvenienced, or worse, by decisions the computer people make and then hide with their claim of technical necessity. (See p. 56: Dealing with computer people.)

3) YAGOTTAS: COMPUTER AS COERCER

More aggressively, cybercrud is a technique for making people do what you want. "The computer requires it," you say, and so people can be made to hand over personal information, secretaries can be intimidated into scouring the files, payment schedules can be artificially enforced.

THE GENERAL STATUS TRICK

Status tricks, combining the putdown and the self-boost, date back to times immemorial. But today they take new forms. The biggest trick is to elevate yourself and demean the listener at the same time, or, more generally, the technique is making people feel stupid while acting like a big cheese. Thus someone might say,

"People must begin to get used to the objective scientific ways of doing things that computers now make necessary."

But the translation seems to be:

"People must get used to the inflexible, badly thought out, inconvenient and unkind systems that I and other self-righteous individuals and companies are inflicting on the world."

YOU DON'T ALWAYS GOTTA

The uninformed are bulldozed, and even the informed are pressured, by the foolish myths of the clever, implacable and scientific computer to which they must adapt. People are told they have to "relate to the computer." But actually they are being made to relate to systems humans have designed around it, in much the same way a sword dance is designed around the sword.

When establishment computer people say that the computer requires you to be systematic, they generally mean you have to learn their system. But anyone who tells you a method "has to be changed for the computer" is usually fibbing. He prefers to change the method for the computer. The reasons may be bad or good. Often the computer salesman or indoctrinator will present as "scientific" techniques which were doped out or whomped up by a couple of guys in the back room.

Here is an example, as told to me. A friend of mine worked in a dress factory where they had a perfectly good system for billing and bookkeeping. Customers were listed by name and kept in alphabetical order. The fast pace of the garment industry meant that companies often changed names, and so various companies had a number of different names in the file. This bothered nobody because the people understood the system.

Then management bought a small computer, never mind what brand, and hired a couple of guys to come in and put the bookkeeping system on it.

Still okay. Indeed, small programming firms can sometimes do this sort of thing very well, because they can work flexibly with the people and don't necessarily feel committed to making it work a certain way.

Well, this was a nice instance where the existing system could have been exactly transferred to the computer. The fact that some customers had several names would certainly have been no problem; a program could have been written that allowed users to type any acceptable customer name, causing the computer to look up the correct account (and if desired, print its usual name and ask for verification).

But no. The guys did not answer employees' questions comprehensibly, nor did they want suggestions. They immediately decreed that since computers only worked with numbers (a fib, but a convenience to them), every customer would thenceforth have to be referred to by number.

After that the firm had nothing but trouble, through confusion over the multiple names, and my friend predicted that this would destroy the company. I haven't heard the outcome.

This story is not necessarily very interesting; it merely happened. It's not a made-up example.

Moral: until we overthrow the myth that people always have to adapt to computers, rather than the other way around, things will never go right. Adaptations should take place on both sides, darn it.

EVERYBODY DOES IT

Cybercrud is by no means the province of computer people alone. Business manipulators and bureaucrats have quickly learned the tricks. Companies do it to the public. The press, indeed, contributes (see Suggestions for Writers and Spokesmen, p. 77). But the computer people are best at it because they have more technicalities to shuffle around magically; they can put anybody down.

Now, computer people do deserve respect. So many things that people do with computers are hard. It can be understood that they want to be appreciated, and if not for the particulars, for the machismo (machinismo?) of coping with intricacy. But that is no excuse for keeping others in controlled ignorance. No man has a right to be proud that he is preserving and manipulating the ignorance of others.

"If it can't be done in COBOL,
I just tell people it can't be done by computer.
It saves a lot of trouble."

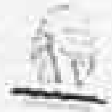
Attributed to somebody in Rochester.
(See COBOL, p. 31.)

BALDERDASH!

Dear Charter Subscriber,

Because [redacted] by electronic computer, a subscription will be maintained a common expiration date to all subscriptions. This enables us to distribute copies and mail renewal notices to all subscribers at the same time. Therefore, we are writing to inform you that your [redacted] Charter Subscription must be renewed now.

In the movie "Fail-Safe," they showed you lots of fake tape drives with the reels constantly turning in one direction. This they called a "computer." Calling any similar box "a computer" is a widespread trick. Gives people the willies. Keeps 'em in line.



Dear Depositor:

Your bank is now utilizing a computer to provide you with better banking service. This new computer requires the use of a three part deposit slip. Enclosed you will find a supply of these new deposit slips. Please compare the account number on the deposit slips with the one imprinted on your checks to be sure the numbers are identical. If they agree, please start using them immediately. We recommend that you carry a few of these deposit slips in the cover of your checkbook.

If there are any questions about this new procedure, any one of our officers will be glad to help you.

PIFFLE. (No program requires it.)

You can buy little boxes with blinking lights that do nothing else but blink. They really put people uptight. "Are you recording what I say?" people ask. "Is it a computer?" They'll believe such a box is anything you tell them.

## REASONS FOR CYBERCRUD (ALL BAD)

1) to manipulate situations.
2) to control others.
3) to fool.
4) to look like hot stuff.
5) to keep outsiders from seeing through something.
6) to sell something.
7) to put someone down.
8) to conceal.
9) general secretiveness.
10) low expectation of others' mentality.
11) seeking to be the broker and middleman for all relations with the computer.
12) vagueness sounds profound.
13) you don't have to show what you're not sure of.
14) your public image is monolithic.
15) you really don't know.

## BEAUTIFUL BUNNY BOOTIES

Cybercrud is not aimed only at layman. It can work even among insiders.

The operations manager of a national time-sharing service, for example, was fanatical about cleanliness. In order to assure a Clean Computer Room, he said, and hence no dangerous dust near the tapes or disks, he made a rule requiring that anyone entering the computer room had to wear cloth booties over his shoes.

Booties were hung outside for those who had to enter.

"And I had the greatest time making his," says his wife, laughing. "With the cutest little bunny faces on them. The buttons were the hardest part to get-- you know, the ones with eyes that roll!" She laughs very hard as she tells this.

"Of course there was no need for it," he now chortles, "but it sure kept people out of the computer room."

(That's applied logic for you.)

" COMPUTERS
AND THEIR PRIESTS

" First get it through your head that computers are big, expensive, fast, dumb adding-machine-typewriters. Then realize that most of the computer technicians that you're likely to meet or hire are complicators, not simplifiers. They're trying to make it look tough. Not easy. They're building a mystique, a priesthood, their own mumbo-jumbo ritual to keep you from knowing what they-- and you-- are doing."

-- Robert Townsend,
Up The Organization (Knopf), p. 36.

## THE CARGO-CULT ASPECT

Outsiders are often prey to cybercrud they dream up themselves. I once knew a college registrar's office where they had been getting along fine for years with paper forms. The year before the computer was slated to arrive, they started using file cards filled out by hand, instead. Why? "Well, we thought that would make it easier for the computer. Computers use cards, don't they?"

Note that referring to a computer as if it were a living creature is not cybercrud, to say that a program "looks at" a device, "tries to" effect a procedure, and "goes to sleep," are all colorful brief ways of describing what really happens. (See Guidelines for Writers and Spokesmen, p. 47.)


Cybercrud is, of course, just one branch of
THE GREAT GAME OF
TECHNOLOGICAL PRETENSE
that has the whole world in its grasp.

"Man, woman, child —
all is up against the wall
of Science."
Firesign Theater

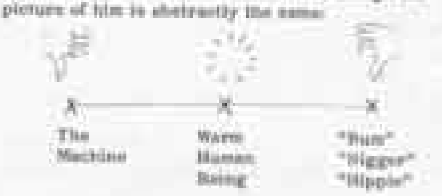
## THE MYTH OF THE MACHINE: A DEEP CULTURAL ENGRAM

Public thinking about computers is heavily tinged by a peculiar image which we may call the Myth of the Machine. It goes as follows: there is something called the Machine, which is Taking Over The World. According to this point of view The Machine is a relentless, peremptory, repetitive, invariable, monotonous, inexorable, implacable, ruthless, inhuman, dehumanizing, impersonal Juggernaut, brainlessly carrying out repetitive (and often violent) actions. Symbolic of this is of course Charlie Chaplin, dodging the relentless, repetitive, monotonous, implacable, dehumanizing gears of a machine he must deal with in the film Modern Times.

Ordinarily this view of The Machine is contrasted with an idea of a Warm Human Being, usually an idealized version of the person thinking these thoughts.

The Machine ←——→ Warm Human Being

But consider something. The model often goes further than this. The Machine is cold, the Human Being emotional and warm. Yet there is such a thing as being too emotional and warm. There is in fact a third type in the schema, the being who goes too far on the same scale. Strangely, he has at least three different names, though the picture of him is abstractly the same:




Now, "bums," "niggers" and "hippies" are not real people. The words are derogatory slang for the destitute, for persons with any African ancestry, and for people dressing in certain styles. But the remarkable thing about the slang is that all three of these derogatory terms seem to have the same connotation in our culture: someone who is dirty, lazy and lascivious. In other words, whatever distinguishes The Machine from the Warm Human Being is carried too far by the bunch at the other end.

In other words, this conceptual continuum is a single, fundamental scale in our culture; why is unclear. Since most people consider themselves-- naturally!-- to be in the middle category, it acts as a sort of reference continuum of two bad things on either side.

It also has another effect: it supplies a derogatory way of seeing. On the right-hand side, it allows many Americans not to see, or to see only with disgust, the destitute and those with African ancestry and those dressing in hippie style. But this book isn't about that.

The left side of the continuum is our present concern. There, too, people refuse to see. What people mainly refuse to see is that machines in general aren't like that, relentless, repetitive, monotonous, implacable, dehumanizing. Oh, there are some machines like that, particularly the automobile assembly line. But the assembly line was designed the way it is because it gets the most work out of people. It gets the work it does out of people by the way it exerts pressure.

So here we see the same old trick: people building a system and saying it has to work that way because it's a machine, rather than because that's how I designed it.

To make the point clearer, let's consider some other machines.

The automobile is a machine, but it is hardly the repetitive, "dehumanized" thing we usually hear about. It goes uphill, downhill, left and right, fast and slow. It may be decorated. It is the scene of many warm human activities. And most importantly, automobiles are very much the extension of their owners, exemplifying life-style, personality, and ideology. Consider the Baja Buggy Volkswagen and the ostentatious cushy Cadillac. Consider the dashboard ornament and the bumper sticker. The Machine, indeed.

The camera is a machine, but one that allows its user to freeze and preserve the views and images of the world he wants.

The bicycle is a machine, but one that brings you into personal and non-polluting contact with nature, or at least that stylized kind of nature accessible to bicycle paths.

To sum up, then. The Machine is a myth.
The bad things in our society are the products of bad systems, bad decisions and conceivably bad people, in various combinations. Machines per se are essentially neutral, though some machines can be built which are bad indeed, such as bombs, guns and death-camps.

The myth of The Machine is a curious aspect of our ideology. Is it especially American, or world-wide?

If we ignore this myth we can see each possible machine or system for what it is, and study how it ties in with human life for good or ill, fostering or louting up such things as the good life, preservation of species, love and self-respect.

## THE MYTH AND THE RORSCHACH

"The computer is the ultimate Rorschach test," Freed Bales said to me twelve years ago. Dr. Bales, a Harvard psychologist, was somewhat perturbed by the papers he was getting in his seminar on computer modelling in the social sciences. Somewhat nutty people in the seminar were writing somewhat nutty papers for him.

And truer words were never spoken. On this point I find Bales has been terribly, terribly right. The computer is an incredible projective test: what you see in the computer comes right off the back wall of your psyche. In over a decade in the field I have not ceased to marvel at the way people's personalities entwine with the computer, each making it his own-- or rejecting it-- in his own, often unique and peculiar way, deeply reflecting his concerns and what is in his heart. Yes, odd people are attracted to the computer, and the bonds that hold them are not those of casual interest.

In fact, people tend to identify with it.

In this light we may consider the often-heard remarks about computers being rigid, narrow, and inflexible. This is of course true in a sense, but the fact that some people stress it over and over is an important clue to something about them. My own impression is that the people who stress this aspect are the comparatively rigid, narrow and inflexible people.

Other computer experts, no less worthy, tell us the computer is a supertoy, the grandest play machine ever to be discovered. These people tend to be the more outgoing, generous and playful types.

In a classic study, psychiatrist Bruno Bettelheim examined a child who thought he was a machine, who talked in staccato monosyllables, walked jerkily and decorated the side of his bed with gears. We will not discuss here the probable origins and cure of this complex; but we must consider that identifying with machines is a crucial cultural theme in American society, an available theme for all of us. And it well may be that computer people are partaking of this same self-image: in a more benign form, perhaps, a shift of gears (as it were) from Bettelheim's mechanical child, but still on the same track.

Some of the computer high-school kids I've known, because of their youth, have been even more up-front about this than adults.

I know one boy, for instance, whose dream was to put a 33ASR Teletype on wheels under radio control, and alarm people at the computer conference by having it roll up to them and clatter out questions impersonally. (If you know the kid -- aloof and haughty-seeming-- you might think that's how he approaches people in real life.)

I know a high-school boy (not a computer expert) who programmed a computer to type out a love story, using the BASIC "print" command, the only one he knew. He could not bring himself to write the love story on paper.

The best example I can think of, though, took place at the kids' booth (see p. 47) at a computer conference. One of the more withdrawn girls was sitting at an off-line video terminal, idly typing things onto the screen. When she had gone a sentence remained. It said:

I love you all, but at a distance.

(On the other side of this book, Dream Machines, we will carry this matter further. The most exciting things in the computer field are coming from people trying to realize their wildest dreams by computer: artificial intelligence, computer music, computer picture-making and so on.)

# THE POWER AND THE GLORY

Forget what you've ever heard or imagined about computers. Just consider this:

The computer is the most general machine man has ever developed. Indeed, it should be called the All-Purpose Machine, but isn't, for reasons of historical accident (see nearby). Computers can control, and receive information from, virtually any other machine. The computer is not like a bomb or a gun, which can only destroy, but more like a typewriter, wholly noncommittal between good and bad in its nature. The scope of what computers can do is breathtaking. Illustrated are some examples (although having all this happen on one computer would be unusual). It can turn things on and off, ring bells, put out fires, type out on printing machines.

Computers are incredibly dogged. Computers can do things repeatedly forever, or an exact, immense number of times (like 4,981,223), doing something over and over, depending on whether it's finished or not. A computer's activities can be combined in remarkable ways. One activity, repeated over and over, can be part of another activity repeated over and over, which can be a part of still another activity, which can be repeated ad infinitum. THERE ARE DEFINITE LIMITATIONS on what computers can do, but they are not easy to describe briefly. Also, some of them are argued about among computer people.

# THE ALL-PURPOSE MACHINE

Computers are COMPLETELY GENERAL, with no fixed purpose or style of operation. In spite of this, the strange myth has evolved that computers are somehow "mathematical."

Actually von Neumann, who got the general idea about as soon as anybody (1940s), called the computer

THE ALL-PURPOSE MACHINE.

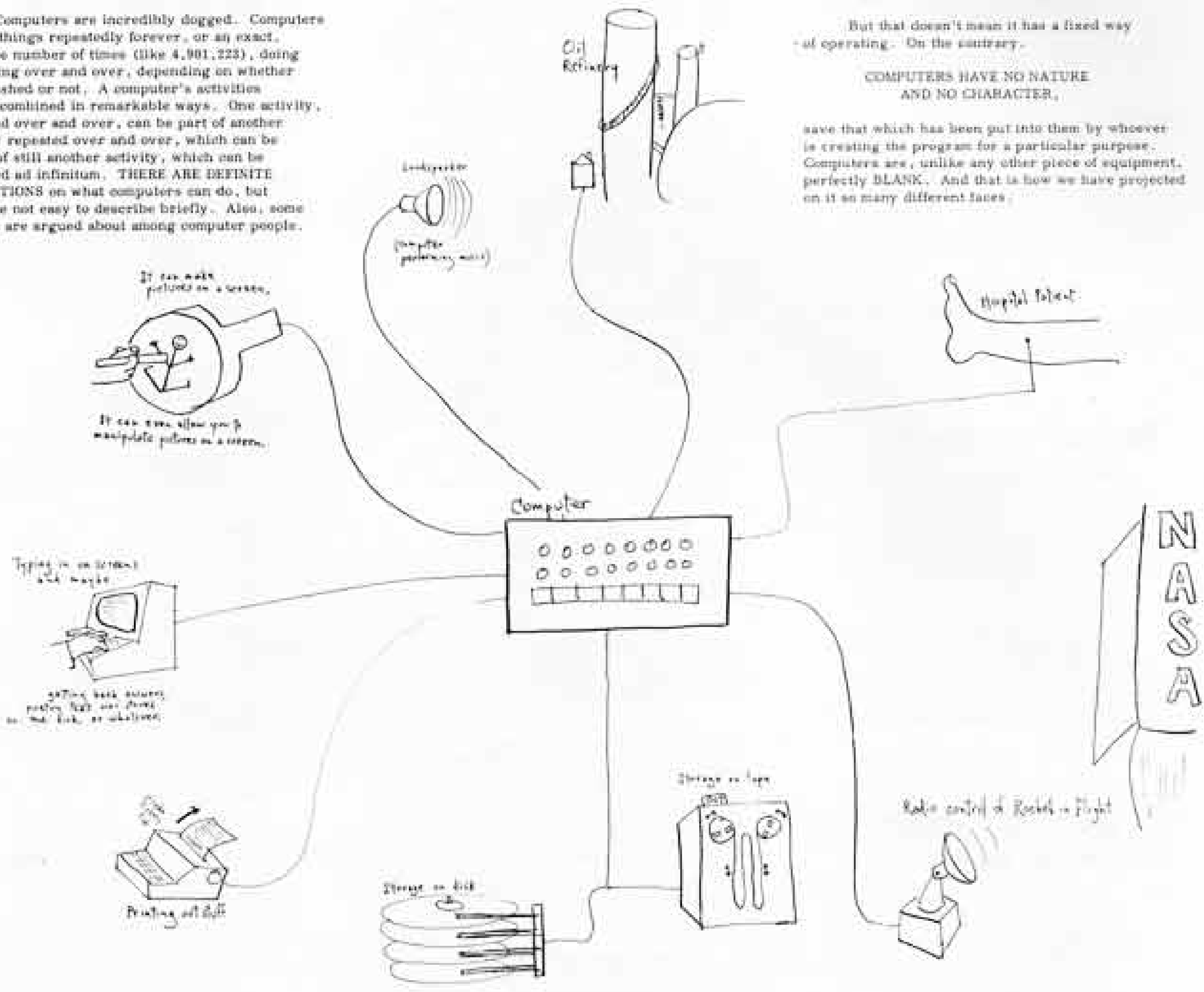
(Indeed, the first backer of computers after World War II was a maker of multi-lightbulb signs. It is an interesting possibility that if he had not been killed in an airplane crash, computers would have been seen first as text-handling and picture-making machines, and only later developed for mathematics and business.)

We would call it the All-Purpose Machine here, except that for historical reasons it has been slapped with the other name.

But that doesn't mean it has a fixed way of operating. On the contrary.

COMPUTERS HAVE NO NATURE
AND NO CHARACTER,

save that which has been put into them by whoever is creating the program for a particular purpose. Computers are, unlike any other piece of equipment, perfectly BLANK. And that is how we have projected on it so many different faces.




A HELPFUL COMPARISON

It helps sometimes to compare computers with typewriters. Both handle information according to somebody's own viewpoint.

| Nervous Question | Helpful Parallel |
|---|---|
| "Can a Computer Write a Poem?" | "Can a Typewriter Write a Poem?" (Sure. Your poem.) |
| "Can't Computers Only Behave Mechanistically?" | "Can't Typewriters Only Behave Mechanistically?" (Yes, but carrying out your intent.) |
| "Aren't Computers Completely Impersonal?" | "Aren't Typewriters Completely Impersonal?" (Well, it's not like handwriting, but it's still what you say.) |

Many ordinary people find computers intuitively obvious and understandable; only the complications elude them. Perhaps these intuitively helpful definitions may help your intuition as well.

1. Think of the computer as a WIND-UP CROSSWORD PUZZLE.

2. A COMPUTER IS A DEVICE FOR TWIDDLING INFORMATION. (So, what kinds of information are there? And what are the twiddling options? These matters are what the computer field consists of.)

3. A computer is <u>a completely general device, whose method of operation may be changed, for handling symbols in any specific way.</u>

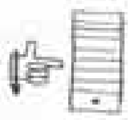
# THE DEEP DARK SECRET

THE MAGIC OF THE COMPUTER PROGRAM

The basic, central magical interior device of the computer we shall call a program follower. A program follower is an electronic device (usually) which reads symbols specifying operations, carries out the step each specifies and goes on to the next.

The program follower reads down the list of instructions in the program, taking each instruction in turn and carrying it out before it goes on to the next.

Now, there are program followers that just do that and nothing more: they have to stop when they get to the end of the list of instructions.



A true computer, however, can do several things more.


It can jump back to an earlier point in the program and go on from there. Repeating the program in this fashion is called a loop.


It can perform tests on symbols in the memory— for instance, to see if a loop has been done enough times, or if some other part of the job has been finished— and jump to some other program depending on these symbols. This is called a branch.

Finally, the computer can change the information stored in memory. For instance, it can place an answer in a specific part of memory.

WHAT, THEN, IS A (Digital) COMPUTER?

A device holding stored symbols
in a changeable memory,
performing operations on some of those symbols
in the memory,
in a sequence specified by other symbols
in the memory,
able to change the sequence
based on tests of symbols in the memory,
and able to change symbols in the memory.
(For example, do arithmetic and
store the result in the memory.)

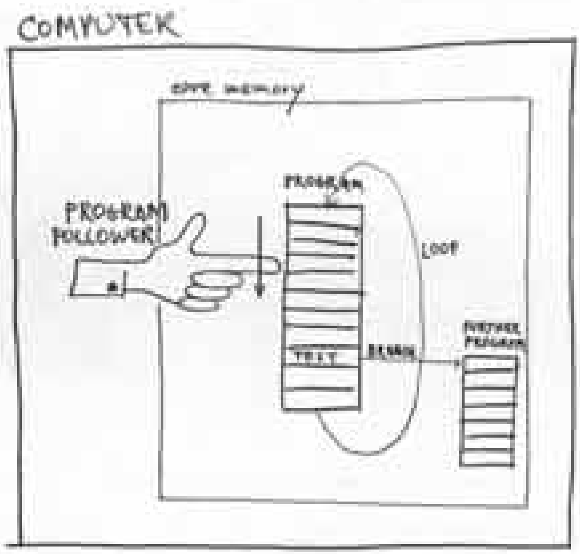
Rather than try to slip it to you or prove it in some fancy way, let's just state baldly: the power of such a machine to do almost anything surpasses all previous technical tricks in human history.

HOW CAN A COMPUTER CONTROL
SO MANY DIFFERENT THINGS?


Answer. Different as they may seem, all devices are controlled in the same way. Every device has an interface, that is, its own special connection setup, and in this interface are the device registers.

These device registers look the same to the computer; the computer program simply moves information patterns into them or moves information patterns from them to see what they contain.


The computer, being a machine, doesn't know or care that device register 17 (say) controls a hog feeder, or device register 22 (say) receives information from smog detectors. But what you choose, in your program, to put into device register 17, controls what the hogs eat, and what comes into device register 22 will tell your program, you hope, about smog conditions. Choosing how to handle these things in your program is your business.

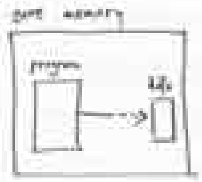



"What is an Interface?"
asked the baby machine.
"Whatever Turns You On."
said its dad.


HOW DOES THE LOOP WORK?

The computer does things over and over by changing a stored count, then testing the stored count against another number which is what the count should get to, and going to the beginning if the desired count has not been reached. This is called a loop. (If there's no way it can ever get out, that's an endless loop.) (Actually, the program loop is done the same way as a program branch: IF a certain count has not been reached, it branches BACK to the start of the loop.)

Other things besides programs may be stored in the memory. Anything besides programs are usually called data.




The instructions of programs use the data in different ways. Some programs use a lot of data, some use a little, some don't use any. It is one of the fascinating and powerful things about the computer that both the instructions of a program, and the data they work on, are stored as patterns of bits in the same memory, where they can be modified as needed. Indeed, the program can modify its own patterns of bits, a very important feature.

WHAT DO PROGRAMS LOOK LIKE?

In what forms are these programs stored, you ask? Well, they are written by people in computer languages, which are then stored in some form in the computer's fast core memory, where the program follower can act on them. But what does a computer language look like, you ask? Aha...

GO TO PAGE 16

(If you want to see what the bottom-most level looks like, with all the bits and things, skip ahead to p. 37.)

WHATEVER IT MAY DO IN THE REAL WORLD,
to the computer program
it's just another device.

ANALOG COMPUTERS DISPOSED OF

There are two kinds of computers: analog and digital. (Also hybrid, meaning a combination.) Analog computers are so unimportant compared to digital computers that we will polish them off in a couple of paragraphs.

"Analog" is a shortened form of the word "analogy." Originally an "analog" computer was one that represented something in the real world by some other sort of physical enactment— for instance, building a model of an economic system with tubes and liquids; this can demonstrate Keynesian economic principles remarkably well.

However, the term "analog" has come to mean almost exclusively pertaining to measurable electrical signals, and an "analog computer" is a device that creates or modifies measurable electric signals. Thus a hi-fi amplifier is an analog computer (it multiplies the signal), a music synthesizer is an analog computer (it generates and reshapes analog signals). Thus the term has deteriorated: almost anything with wires is an analog computer.

Analog computers cannot be truly programmed, only rewired.

Analog equipment is useful, important and indispensable. But it is simply not in the same class with digital computers, henceforth called "computers" in this book, which manipulate symbols on the basis of changeable symbolic programs.

"Analog computer" also means any way of calculating that involves measuring approximate readings, like a slide rule.

# LET'S CALL A SPADE A SPADE

It's awfully easy to fool people with simple words, let alone buffalo them with weird technical-sounding gab. The thing about tech talk is that it can really be applied to any area. The trick lies in the arrangement of bizarre adjective nouns, and in the vague use of windy terms that have connotations in some particular technical area-- say, the space program.

Just consider. We might call a common or garden spade--

A PERSONALIZED EARTH-MOVING EQUIPMENT MODULE

A MINERALOGICAL MINI-TRANSPORT

A PERSONALIZED STRATEGIC TELLURIAN COMMAND AND CONTROL MODULE

AN AIR-TO-GROUND INTERFACE CONTOUR ADJUSTMENT PROBE

A LEVERAGED TACTILE-FEEDBACK GROMASS DELIVERY SYSTEM

A MAN-MACHINE ENERGY-TO-STRUCTURE CONVERTER

A ONE-TO-ONE INDIVIDUALIZED GEOPHYSICAL RESTRUCTURIZER

A PORTABLE UNITIZED EARTHWORK SYNTHESIS SYSTEM

AN ENTRENCHING TOOL (Foreign Theater)

A ZERO-SUM DIRT LEVEL ADJUSTER

A FEEDBACK-ORIENTED CONTOUR MANAGEMENT PROBE AND DIGGING SYSTEM

A GRADIENT DISEQUILIBRATOR

A MASS DISTRIBUTION REDENTROPIZER

or **Hey! A DIG-IT-ALL SYSTEM**

AN EXTRA TERRESTRIAL TRANSPORT MECHANISM.

Spades, not words, should be used for shovelling. But words should help us unearth the truth.

---

In the computer field, the same things are often called by different names. (For instance, the IBM 1800, a fairly ordinary minicomputer, is called by them the "IBM 1800 Data Acquisition and Control System"). Different things are often called by the same names, and things can be inside-out and upside-down versions of each other in extraordinary variety. (Indeed, computer people may find this book inside-out, which is okay with me. Life is a Klein bottle.)

Sorting things out, then, means having a few basic concepts clear in your mind, and knowing when you see examples and variations of them.

---

*Computer people often say that to understand computers you have to have a "logical mind."*

*There's no such thing. But saying such things intimidates many, especially those who have been told they do not have "logical minds."*

*What is meant, actually, is indeed important: in working with computers you must often work out the exact ramifications of specific combinations of things, without skipping steps.*

*But the other mode of thinking, the intuitive, has its place in the computer field too. Whichever your habitual style of mind, computers offer you food-- and atmosphere-- for thought.*

---

HORRIBLE MISUNDERSTANDINGS

Some people think of computers as things that somehow mysteriously digest and assimilate all knowledge. "Just feed it to the computer," is the motto. But what you feed into the computer just sits there unless there's a program.

"How would you do that by computer?" is a question people often ask. The question should be, "how would you do that at all?" If there is a method for doing something which can be broken down into simple steps, and requires no human judgment, then maybe we can take these steps and program them on a computer. But maybe we can also think of a simpler way to get them done.

Then there is the idea that a computer is something you ask questions. This assumes, I guess, the earlier premise, that the computer has already digested and assimilated a lot of stuff and can sling it back at you in new arrangements.

Actually what must happen, to get "questions" answered, is this: there must be some program that puts input material into a data structure. (See "Data Structures.") Then you need programs that will count and trace, or whatever, through the data structure in ways you desire. Then you need a way to start these tracing-and-searching programs going through the data structure in ways you want. So you need a progress accepting input from a keyboard, or whatever, and starting the other programs in operation...

---

## COMPUTERS DEFINED JUST LIKE CAMERAS AND CARS

Just the way everyone can understand cameras, viz.:
"A camera is a device you point at something to willfully capture its appearance."

Just the way everyone can understand cars, viz.:
"A car is a device people get inside which then goes somewhere else, under the willful control of the driver."

Well, how about
"A computer is a device which manipulates information and external accessories, according to a plan willfully prepared by a planner."

# INSPIRATIONAL MISCELLANY

---

FICTIONS ABOUT WHAT COMPUTERS DO

Many people suppose there is nothing computers cannot do (see p. 41); some people, indeed, think there is nothing computers do not already do.

A couple of years ago, a leading picture magazine carried a piece about Stanford's Artificial Intelligence Laboratory, claiming that one "Shakey the Robot" had been developed to near-human intelligence and capabilities. This was pure bosh, since repudiated in the computer magazines, but a lot of people Out There in Readerland believed it. (See "The God-Builders," flip side.)

Once I had a long discussion with a somewhat wild-eyed young woman who believed that the government was monitoring her brain with computers. I think I persuaded her that even if this were feasible it would cost the government tens of thousands of dollars to do it, and that probably no existing government agency was that interested in her thoughts. I'm not sure she was persuaded.

---

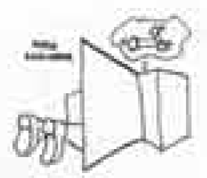
## SOME COMPUTER PEOPLE to distinguish among

Computer operators turn 'em on and off, change programs, change disks and tapes, select modes of operations for programs that can do more than one thing. (See p. 38.)

Input typists (also called keypunch operators) are clerks who copy information into the computer (on terminals) or onto something the computer can read (punch cards, magnetic disk, etc.)

NOTE: these jobs may end in a few years when nothing else has to be copied anymore because users put things in themselves.

Computer programmers create exact plans for what the computer is to do, then change them till they work.



Computer repairmen, or "field engineers," fix computers and their accessories when something goes wrong electrically or in the gears.

They always wear tie clips, at least if they wear ties, so as not to get pulled into running machinery.

A NAIVE USER (no offense) is an ordinary person who doesn't need to know any of these things in order to do something useful with the computer.

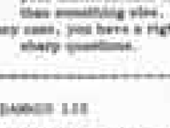
Creating programs to help him is the frontier of computing.

---

WHAT YOU'VE SEEN PROBABLY WASN'T "A COMPUTER."

Get out of your head the notion that some one system you've seen showed you what Computers Are Really Like. Computer systems can be as different externally as bats and whales. (Yet it's the same kind of heartbeat, but that's no help in dealing with them.)

Then what is it computer people know, you may ask, that leads them to understand new systems quickly? Aha. Computer people simply adjust faster to whole new worlds.



---

THE AUTOMOBILE ANALOGY (more)

"The Interstate was bumper-to-bumper, but after we had lunch at the rest stop it cleared up till we got to the tollbooth. Then Harry got lost on the interchange, and we had to double back on the service road."

How incomprehensible to someone from 1903. Yet how simple-minded when you understand it. That's how it is with computers.

Computer talk sounds so strange and incomprehensible to you folks out there-- yet to us in here it's often as simple as the lines above-- if you know the fundamental concepts.

And nothing in the normal everyday world will have prepared you for them.

It's not jargon, but the simplest way to express thoughts in these areas.

---

USING A COMPUTER
SHOULD ALWAYS BE EASIER
THAN NOT USING A COMPUTER.

If it isn't, you
(or your company, or your state)
may have been sold a bill of goods.
OR they may have decided
your inconvenience is less important
than something else.
In any case, you have a right to ask
sharp questions.

---

THE DAMNED LIE

"Computers are rigid and inhuman."

A BETTER APPROXIMATION

People are sometimes (all too often) rigid and inhuman. (Machines and animals are nonhuman-- the term "inhuman" applies only to people.)

"Rigid and inhuman" computer systems are the creation of rigid and inhuman people.

---

WHAT IS THIS SYSTEM ABOUT?

Handy questions to size up what a computer is supposed to be doing.

What data does it contain?

Where is the data stored?

What other data will it look up to?

What information do you suppose can reasonably be derived from that?

What are the key input and output devices?

In what forms does information go in and out?

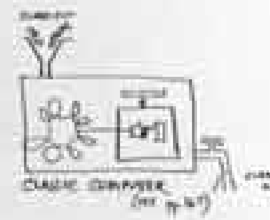
What do you suppose they might want to know?

---

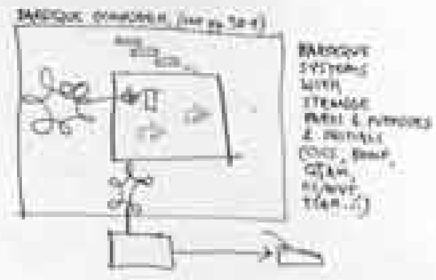
# THE NEW ERA

A new era in computers is dawning.

The first, or Classic, computer era used straightforward equipment and worked on straightforward problems.



The second, or Baroque, computer era used intricate equipment for hard-to-understand purposes, tied together with the greatest difficulty by computer professionals who couldn't or wouldn't explain very well what they were doing.



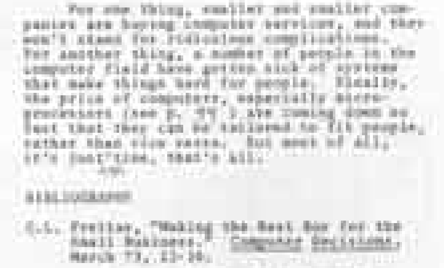
But a change is coming. No one company or faction is bringing it about, although some may feel it is not in their interest. I would like to call it here the DIAPHANOUS age of the computer.

By "diaphanous" I refer both to the transparent, understandable character of the systems to come, and to the likelihood that computers will be showing us everything (dig), across everything, phalanxes, to then). [Ed: Also?]

In the first place, COMPUTERS WILL DISAPPEAR CONCEPTUALLY, will become "transparent", in the sense of being parts of understandable wholes. Moreover, the "parts" of a computer system will have CLEAR CONCEPTUAL MEANING. In other words, COMPUTER SYSTEMS WILL BE UNDERSTANDABLE. Instead of things being complicated, they will become simple.

Now, many people think computers are by their nature incomprehensible and complicated-- unfortunately, that's because they have been MADE TO BE. Usually this is unintentional, but I fear not always. EXAMPLE. Instead of being told, "this is the mysterious XYZ computer, it has to have things just so, you have to fill out these RBQ forms to go into the YZA...", you will hear such surprisingly simple things as "This system is set up for keeping track of who owes what to the company. On the screen you can get lists of accounts and outstanding bills and who owes them; if you point at one with the light pen, the printing machine over here will print a bill all set to go in the envelope."

In other words, systems will increasingly have UNDERSTANDABLE PARTS WITH UNDERSTANDABLE INTERCONNECTIONS.



What is responsible for this remarkable change?

For one thing, smaller and smaller companies are buying computer services, and they won't stand for ridiculous complications. For another thing, a number of people in the computer field have gotten sick of systems that make things hard for people. Finally, the price of computers, especially microprocessors (see p. 59), are coming down so fast that they can be tailored to fit people, rather than vice versa. But most of all, it's just time, that's all.


BIBLIOGRAPHY

C.L. Freiling, "Making the Best Buy for the Small Business." Computer Decisions, March 73, 23-26.

Compares the relative costs of minicomputers and time-sharing; concludes that minis are the best buy.

Burton L. Katz, "Making Minicomputers Work in a Medium-sized Business," Data Processing, Winter 1971, 8-11.

Stresses the point that well-designed computer systems can be used by existing personnel of a firm, without excessive complication.

Frederic G. Withington, "Cosmetic Programming." Datamation, Mar 70, 61-65. How to make systems friendly on the outside.

# INTERACTIVE SYSTEMS

Used to be that ordinary people had to deal with computers by filling out intricate forms, which were then translated into punch cards. The forms put things in weird categories (see "Caged-Down Data," p. 29.)

No longer.

Anyway, no longer *necessary*.

Computer systems can now give you action, excitement-- and explanations.

This is done through the magic of the TERMINAL. Terminals come in two conspicuous flavors (typewriter and screen or "boob tube") and also have two less-noticeable divisions ("Teletype" or "industry" versus "IBM type.")

Anyway, a terminal is something that allows a person and a computer to type at each other.

Now, computers are merely gadgets for twiddling information. They no more understand English, or human psychology, than puppies can read music. (See "Artificial Intelligence," p. 11-15). But the computer's program can, for instance, direct the computer to type out a simple question, and compare the user's answer with a simple set of alternatives. For example, suppose the user is visiting a hospital. A computer can sign him in without the abrasiveness of a receiving nurse, and with far more patience. The following might be a sample dialogue. (Here the computer types what's in caps, and the user's replies are in lower-case.)

```
DO YOU HAVE AN ACUTE PAIN?  (Y, N, DK)
     dk
YOUR ANSWER IS: DK FOR "DON'T KNOW."
     DOES THAT MEAN YOU'RE NOT SURE
     WHAT 'ACUTE' MEANS?  (ANSWER A)
     A PAIN COMES AND GOES? (ANSWER B)
     YOU HAVE A PAIN SORT OF ON THE
     BORDER?  (ANSWER C)
     c
IS THIS PAIN IN AN EXACT PLACE YOU
     CAN IDENTIFY? (Y,N,DK)
     y
```

As interactive system of this kind is called a __conversational__ system, in that it "converses" with the user. The secret is that the alternatives in the computer program are few and carefully worked out beforehand; there are great pitfalls when there are too many alternatives, as when such conversational systems are used for teaching (see pp. DM 16-19).

Here is a straightforward example: a system I wish I had for balancing a checkbook. Note that the inner program for this conversational system could be written in any of the three languages presented later.

```
WHAT PROGRAM WOULD YOU LIKE TO RUN? chkb
CHECKBOOK PROGRAM STARTS.
DO YOU WANT TO PROOF THE NEW STATEMENT FOR
     MARCH?  y
PLEASE LIST THE CHECKS THAT HAVE COME IN.
     231, 239, 240, 241, 244, 241.
SUM OF INCOMING CHECKS IS $345.72.  DO YOU
     WANT BREAKDOWN?  n
PRESUMABLY BANK IS CHARGING YOU .60 FOR
     SIX CHECKS.  ALSO MONTHLY CHARGE OF
     FIFTY CENTS (PLEASE CONFIRM).  y
ARE THERE ANY OTHER BANK CHARGES THIS
     MONTH?  n
ARE THE FOLLOWING DEPOSITS ON THIS STATE-
     MENT-- MARCH 1 SALARY, $444.00?  y
     GIFT FROM AUNT AGATHA, 18 MARCH,
     $13.00?  n
TOTAL ON STATEMENT SHOULD BE $1711.11.
     PLEASE CONFIRM.  y
YOUR CURRENT FLOAT IS $836.73.  DO YOU WANT
     BREAKDOWN?  y
CURRENT FLOAT AS FOLLOWS--
     NO. 245   IRVING'S RECORDS 7 MARCH $   6.73
     NO. 245   SINISTER & MALAMBUIT (LEGAL
               FEES) 11 MARCH              $800.00
     NO. 246   DOGGIE HAIRDRESSERS
               12 MARCH                    $ 10.00
     NO. 247   SAM GROSS (ESTATMENT)
               13 MARCH                    $ 20.00
     TOTAL                                 $836.73
ARE YOU DONE WITH CHECKBOOK PROGRAM?  y
```

(The part shown above is easy. Thinking out the ways for the user to __correct__ his records, and/or the bank, is the tough part.)

COMPANIES THAT WILL SET UP WHOLE LITTLE BUSINESS SYSTEMS

A number of companies make minicomputers (partial list on p. 45.); however, companies who want business systems built around minicomputers may want to investigate companies that will put together __whole__ business systems for them around minis.

(It is hoped that one contribution of this book will be to give the reader a better idea of what to ask for.)

Two companies that seem to be in this business are:

Genesis One Computer Corporation, 95 Park Ave., NY 10016. Appears to use BASIC language (see pp. 11-17).
Qantel Corp. (offices in five major cities). Sells a minicomputer of their own manufacture, using a language called QIC (Qantel Interactive Code), which a salesman tells me is "just like BASIC" (see pp. 16-17). Minimum setup includes a display terminal, printer, computer and 4-million-character disk, at $31,000.

In a conversational system the computer can helpfully lead the user on.

# TERMINALS

*A terminal is simply any device by which a person and a computer can type at each other.*

*Like true terminals. This one is a video terminal or keyscope (see p. DM 15). It allows the computer to present textual or numeric information, play games with you, ask you for information in a good-guy system, or whatever -- depending on the program, of course.*

*More expensive screens for computer __displays__ allow pictorial animation under the user's control (discussed throughout flip side). THE MAIN THING TO UNDERSTAND: what they do is decided by __human beings__, not "scientific principles." Human beings talk here.*

*Types of available computer terminals are discussed in the next spread; more display terminals discussed p. DM 15.*

*Below: a "bull pen" of terminals, all hooked up to the same computer at the Chicago Circle Campus, University of Illinois. What each person does at his terminal is actually independent of what any other person does, through __time-sharing__ of the same computer. Installations more suited to time-sharing can have large numbers of terminals, all over a campus, a company or the world; see Timesharing, p. 45.*

Motto 1 for the new era:

UNDERLINE: USING A COMPUTER SHOULD ALWAYS BE EASIER THAN __NOT__ USING A COMPUTER.

Motto 2 for the new era:

__THE NEW FRONTIER IN COMPUTERS IS CONCEPTUAL SIMPLICITY AND CLARITY.__

People who delight in intricacy are going to have to learn some new tricks. Internal intricacy is fine, as long as the user doesn't have to deal with it.

Motto 3 for the new era (to computer people):

MAKING THINGS EASY IS HARD.

Motto 4 for the new era:

ANY SYSTEM FOR A SPECIFIC PURPOSE SHOULD BE TEACHABLE IN TEN MINUTES OR LESS.

Anyone who has been taught the use of some fixed-purpose computer system, such as an airline reservation system, may doubt this. But perhaps this book will clarify things somewhat.

A "GOOD-GUY SYSTEM"

is a conversational

computer system that is

CLEAR,

EASY TO USE,

AND FRIENDLY.

ANY MAN OF COMMON SENSE CAN DESIGN A COMPUTER SYSTEM FOR A PURPOSE IMPORTANT TO HIM: the data structure, forms of information, general operations, record-keeping, and responses to on-line users.

But for some reason this is generally kept a secret.

"THE TURKEY TEST"

A good friend of mine, Jordan Young, is a former R.E.S.I.S.T.O.R. (see p. 77) and now a systems programmer (see p. 4) on the mighty Dartmouth time-sharing system, DTSS. (See p. 45.)

Jordan tells me that one of the most important people at Dartmouth is a mythical individual named the Turkey User. This estimable personage knows hardly anything about computers, makes a lot of mistakes, thinks he understands when you tell him when he doesn't, leans on the big wrong keys on the terminal, and in general tends to screw up.

But the motto up there is: "If it's not simple enough for the Turkey User-- it's too complicated."

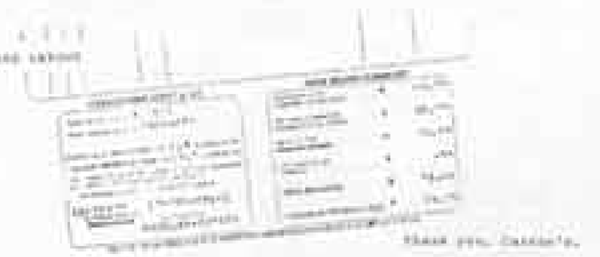
DTSS is a good-guy system.

YOUR FIRST COMPUTER CONTACT

When you first sit at a computer terminal, the feeling is one of sheer terror. Sweat and chills, limpness and sudden clammy nervous motions. Inward apprehensiveness and stammering fear and nervousness interfere with your ability to function or understand the person who is helping you.

It's perfectly normal.

A consignment catalog



Thank you, Carson's.

THE MOST IMPORTANT COMPUTER TERMS FOR THE '70s

Here are some phrases that will count in the new era of computing, when we will run into more and more computer systems set up for particular purposes.

on-line
: connected to a functioning computer. (Note that the computer may be in the typewriter or desk itself.)
(As distinct from __off-line__, setting things up for processing __later__.)

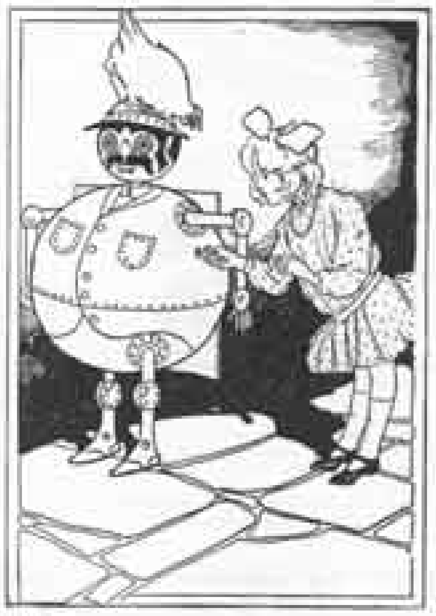
interactive
: not just connected, but responding to you. Interactive systems and programs can respond to your choices and requests, clarify what they want from you, etc.

remote
: referring to something far away, as distinct from __local__, right where you are. A computer can be either remote or local, e.g., on your desk.

front end (n.), front-end (adj.)
: whatever stands between you and a system. A front end can be the terminal in your office, for example. A __front-end program__ is one which mediates between a user and some other system or program, perhaps collecting data for it by quizzing you.

dedicated
: set up for only one use. A big computer at a computing center has to have many uses; a little computer in your office can be dedicated. Dedicated computers are now hidden in all sorts of things: cash registers, for example (see "Microprocessors," p. 44).

turnkey (adj.)
: turned on with a key. Especially, __turnkey systems__, small computer systems that can just be turned on (key or not) and are fully set up, ready to run, programmed, etc.



real-time
: responding to events in the world as needed, without delay. Computer systems that control machinery, make airline reservations, predict the weather or respond to naive users are real-time. Systems that can catch up overnight are non-real-time.

"intelligent terminal"
: stupid term referring to any object that does more than act like a plain terminal. The term is stupid because it confuses distinctions. Some "intelligent terminals" have extra circuits for various purposes; others contain their own minicomputers; still others are ordinary terminals connected to front-end programs.

user-oriented
: set up for "users"-- people who are not programmers or input typists, but who actually need something done.

user level (n.), user-level (adj.)
: "where the user is" mentally; his level of involvement. __User-level system__, system set up for people who are not thinking about computers but about the subject or activity the computer is supposed to help with.

naive user (n.), naive-user (adj.)
: person who doesn't know about computers but is going to use the system. __Naive-user systems__ are those set up to make things easy and clear for such people.
(We are all naive users at some time or other; it's nothing to be ashamed of. Though some computer people seem to think it is.)

naive-proof
: not susceptible to being loused up by a naive user.
The hostility in this term may in some cases be real. Computer people sometimes forget, or do not wish to tolerate, the degree of confusion that naive users bring to the keyboard. This attitude is not just their problem but everybody's, since they lay it on us.

good-guy system
: term to be used here for naive-user systems that are friendly, helpful, simple and clear.

stand-alone system
: system (regardless of purpose) which doesn't have to be attached to anything else. (May contain its own computer.)

## THE MIRACLE OF OVER-THE-PHONE TERMINALS

(some people go ape just to see the typewriter going by itself)

"Modem" takes the terminal's pulse code and warbles it into the phone as audible tones. The computer answers with similar warbles and tweedling; the modem converts that back into alphabetical characters.

RS-232 is the standard interface.

## YOU CAN HANG A TERMINAL EITHER ON A MINICOMPUTER (see p. 36) OR A BIG COMPUTER (see p. 78).

(What it does, of course, depends on the program, and the size or brand of computer.)

# TWO KINDS OF TERMINALS

You would think the fundamental dichotomy among computer terminals was between those that print on paper and those that show you stuff on a screen. But it isn't. (That's like the difference between people and whales-- much greater outside than inside.)

Actually the fundamental distinction between terminals is between ASCII (pronounced "Askey") and IBM terminals. ASCII is a code and scheme of organization which was adopted by "the industry," under the blessing of the National Bureau of Standards. But IBM has pointedly ignored this standard.

The principal terminal of the ASCII type, in sheer numbers, is the model 33-ASR Teletype (trademark of Teletype Corp.), so this kind of terminal is called the "33 ASR type," or "Teletype-type," or we even say a given terminal "looks to the computer like a Teletype."

IBM, however, seems to like changing its systems around a lot, for instance changing its codes when it brings out a new computer. (Fortunately, it just happens that they also sell adaptors between them. Whew.) So IBM-type terminals are different by design.

There is one main type, however, exemplified by the IBM model 2741 terminal. Thus we say a terminal is an "IBM-type" or "2741-type" terminal.

Both Teletype- and IBM-type terminals come in either video-screen or printing models from a variety of manufacturers.

Indeed, even the Selectric (IBM trademark) typing mechanism appears in some Teletype-type terminals.

There is a very important performance difference between ASCII and IBM terminals. The ASCII terminal can send each character typed by the user-- each "keystroke"-- to the computer immediately. This means that highly responsive programs can be written, which examine the user's input and can reply instantaneously, if need be, after anything the user types.

IBM-type terminals, however, require a "line feed" character or an "end of transmission" character to be typed by the user to make it the computer's turn. This locks the keyboard so the person can't use it. Then the computer must type something, ending with its own "unlock" signal that makes it the person's turn again.

Why this unwieldy design? Supposedly it results from the curious decision, in the design of IBM's 360 computer, to make all devices resemble the card reader as far as the computer is concerned. Just as the card reader reads punched cards till the last one is done, the IBM terminal is designed to send and receive characters until a "finished" condition is reached.

(It makes sense to the poor man)

# SOME TERMINALS YOU MIGHT LIKE.

All are ASCII-type unless otherwise noted. Note: there are hundreds of types and brands of terminals available. These are just some thoughts.

PRINTING TERMINALS.

BEST BUY? The model 38 ASR Teletype gives you upper and lower case, and is otherwise similar to the standard model 33. $19 a month from RCA Service Company, Data Communications Div. (offices in major cities). $15/mo. for the coupler. 30-day cancellable but costs $50 to put in, $14 to take out.

There is a cute terminal that behaves just like the 33 ASR, but is faster and uses NCR pressure paper or a ribbon, interchangeably. The Extel Series A teleprinter from Extel Corp., 310 Anthony Trail, Northbrook, Ill. 60062.

If you like Selectrics, but want to go to ASCII, there is one weird possibility.

A firm called Tycom Systems Corporation (26 Just Road, Fairfield NY 07006) offers an interesting alternative. It happens that all Selectrics (anyway, Model I and Model II) have a seam around the midriff at which the typewriter can be unscrewed into two sections. Clever Tycom! They make a device which fits between, looks to the bottom like the top of the Selectric, and looks to the top like the bottom. Also, it turns the Selectric into a terminal, receiving ASCII codes from whatever computer you attach it to and causing the computer to type them, or sending out what you type to the computer in ASCII.

Curiously, IBM has given its blessing to this arrangement, meaning you can have this sandwich deal done to a Selectric you rent from IBM, and serviced under beefed-up IBM maintenance agreements ($72 per year, or $15.38 per hour, as of 1973).

DISPLAY TERMINALS (see pp. DM 20-1) There are many brands. Some are flimsy.

The earlier video terminals came with dreadful styling, like a 1940s science-fiction movie. But as an example of how the market is developing, one of the handsomest video terminals is the $1295 Mini-Tec from TEC Incorporated, 9800 North Oracle Road, Tucson, Ariz. 85704. It comes covered with wood-grain contact paper and looks very nice. (You should have seen their early models.)

The Hazeltine 1000 video terminal rents for $49/mo. on a 2-year contract. LOWER CASE OPTION; modem and coupler apparently not included. (Hazeltine, Greenlawn, NY 11740, with offices all over.)

If you have no objection to ITT, they offer a portable video terminal with built-in modem and coupler, the Asciscope, for $95/month. Supposedly there's a long waiting list. (ITT Data Equipment and Systems Division, East Union Ave., East Rutherford, NJ 07073.)

For a snazzier terminal in your car, see Kustom Electronics, Inc. (aren't they the truck-way people?), Data Communications Division, 1010 West Chestnut, Chanute, Kansas 66720. They've already set up travelling terminals for the mobile constabulary of Kansas City (Mo.), Palm Beach and Nashville. (Communications, Jan. 73, ad p. 87.) Now, of course, you'll need a whole stationary radio setup to run that.

Classic Teletype

MISCELLANEOUS

Various firms rent terminals, some on a short-term basis. These terminal companies are bad news, keeping up their equipment badly and offering poor service, so watch it.

(The day will come, let's hope it's soon, that you can rent a terminal overnight or for a weekend like a movie camera. But till people get a sense of how far and fast things are moving, we'll continue to schlock along haphazardly.)

Unfortunately rental people are hard to find, since they are usually local, and the Yellow Pages invariably lump together every possible form of computer sales and service under "Data Processing Equipment and Supplies," and few firms further specify their business in the listing.

Here are some names (neither endorsed nor criticized):

Computer Planning & Supply, Chicago
TTY Systems, LA
Varney & Associates, Dallas

A good outfit, that rents both ASCII and IBM-type terminals of their own manufacture, is Anderson Jacobson Co. (1065 Morse Ave., Sunnyvale, Calif. 94086, and major cities). They have a Selectric terminal, for instance, which rents for about $100 a month (about the same as the standard IBM 2741) but is portable.

To provide a memory with your ASCII or IBM (IBM-type) terminal, an odd machine called the Techtran 4100 (about $1800 from Techtran Industries, 580 Jefferson Rd., Rochester, NY 14623) can be used for offline storage. It uses a magnetic cassette. Here are some things you can do with it:

type stuff into the Techtran,
later squirt it to a computer at high speed
receive stuff from a computer at high speed,
later type it back automatically on the terminal
type into the Techtran, correct it, and then have it typed back automatically-- no computer.

The question of whether the Techtran can be used with the Digi-Log has not been publicly resolved.

It happens that Anderson Jacobson (above) will rent you their 2741-type Selectric terminal with a Techtran, for about $215 a month total. But they won't rent the Techtran separately.

A 2741-type Selectric terminal with memory, offering these same capabilities, is now available from IBM! It is the Communicating Mag Card Executive (CMC). (Note the Mag Card Executive, to which they have added the communication feature, costs over $200 a month; figure the communication feature could cost another $100 or so monthly, or probably half again as much as the Anderson-Jacobson.

Honeywell (Honeywell Information Systems, Wellesley Hills, Mass.) has recently made available a Braille program to be used with "standard terminals" on their systems. (This may be the adaptation developed at MIT to do Braille on the 33 ASR.)

For those of us literary types who want upper and lower case but are stuck with 33ASRs, a LOWER-CASE CONVERSION KIT is available from Data Terminals and Communications, Campbell, California.

FURTHER POOP

If you're serious about keeping up with developments in the terminal area, you might want to subscribe to Terminals Review ($36/yr.), highly spoken of by Datamation. (GML Corp., 594 Marrett Rd., Lexington, MA 02173.)

A "CRT Survey" listing characteristics of 110 CRT displays (including both video terminals and fancier pictorial displays-- see flip side of this book) is available for ten bucks postpaid from Datapro Research Corp., One Corporate Center, Route 38, Moorestown, NJ 08057.

Standard display terminal offered with coupler from DEC (see p. 33). It's the model VT05, I guess.

VIDEO TERMINALS WITHOUT THE VIDEO

A very hot item right now is a terminal called the "Digi-Log"-- actually several different models-- available from Digi-Log Systems, Inc., 666 Davisville Rd., Willow Grove, Pa. 19090.

This device fits in a briefcase. Basically it is a keyboard with a socket for the phone, and an antenna wire. You phone the computer, drop the phone handset in the slot, and clip the wire to the antenna of a TV set. Presto! On the TV set appears what you and the computer type at each other.

This is especially good for travelling salesmen (to communicate with their offices and ordering system via time-sharing computer) and executives who do computer work from the road. Also for people who want to show off remote computer systems.

Disadvantage: only 32 characters per line, which is awkward for some things, such as programming in Fortran.

Price: $1150 to $1400. They also lease, at rates as low as $46/month (3 years).

No lower-case as yet.

Also available on rental, supposedly, from Westwood Associates, Inc., 16 Washington Terrace, East Orange, NJ 07017.

Ann Arbor Terminals, Inc. (Ann Arbor, Mich.?) is said to offer a similar unit but in very slow.

The equivalent IBM-type terminal-- keyboard, coupler and clip to the TV-- is the IPSA-100, offered by I.P. Sharp Associates, Inc. (Bridge Administration Building, Bridge Plaza, Ogdensburg, NY 13669). Unfortunately it's much larger than the Digi-Log-- it comes in a medium-size suitcase-- and more expensive ($1700 up). However, they offer the APL character-set (see APL piece "Magic Languages," p. 22) as an option-- even a model with both normal and APL character-sets as a switch-selectable option (costs even more).

Recently, of all things, plans for a do-it-yourself unit of this type were announced in a popular electronics magazine (Don Lancaster, "TV Typewriter," Radio-Electronics, Sept. 1973, 43-52). This does not include the full plans, which are available for $2 from TV TYPEWRITER, Radio Electronics, 45 E. 17th St., New York, NY 10003.

Supposedly this can be built for "around $120"-- probably a deal more-- if you are a skilled electronics builder or technician. But that looks to include a great deal of labor.

The finished unit holds up to 32 characters per line and up to 16 lines on the screen; a second memory can be added, to hold a second alternative screenful.

Typical video setup.

---

# TYPE RIGHTER: The Magic Typewriters

A number of different systems are coming on the market to aid you in error-free typing.

IBM would have you call these "word processing systems," since that makes them sound of-a-piece with their dictation equipment. Actually they're text regurgitation systems, but let's just call them Magic Typewriters.

Prices of these things tend to run between $100 and $300 a month.

Generally these are being sold as secretarial aids, partly because they tend to be too ungainly for use by writers themselves. A principal use has been in large law offices, where contracts, wills and such are stored as "boilerplate" (standard sections of Document) and then modified slightly by the lawyer to justify the legal fees.

Such systems all basically consist of three things:

A typewriter, connected to some sort of Magnetic Memory, such as a tape, coated card or disk, and
editing circuitry, which responds to various acts by the user.

WHAT THEY DO: allow you to type stuff in, which is both typed on the paper and at the same time stored on the magnetic whatever. Small errors you correct as you type along, generally by backspacing.

When you want a clean copy-- Presto Wash-o! Put in clean paper, start the magnetic whatever at the beginning, and the typewriter retypes it without a mistake.

If you're lucky.

Unfortunately some of these systems are quite badly thought out. In one or two cases I am not sure whether they are designed so they are accidentally or on purpose. Neither interpretation is flattering to the manufacturer.

I have had extensive experience with two of these systems, the IBM Mag Tape Selectric and the IBM Mag Card Executive. Suffice it to say that if I believed that these systems were as cumbersome as they are by accident, then the sections in this book on IBM and its products might have a very different slant. As it is, these systems require a training period of (two) a week, and require such continuous attention to their curious mechanics that the user is given little opportunity to think of anything else. In both cases, in my opinion, the superficial plausibility of the initial design provides basic user-tangled ramifications which verge on the preposterous. Much of this book was written on a Mag Card Executive-- and I'm damned sorry I bothered.

Some systems of this type are:

The IBM Mag Tape Selectric (MT/ST or MTST). Records on unprotected 16mm mag film of the type used for movie sound recording, and you have two different tapes to get confused between.

The IBM Mag Card Executive. Records on a plastic Hollerith card (see p. 26) coated with magnetic oxide. Variable width of characters presents fascinating difficulties.

The IBM Mag Tape Selectric Composer (MT/SC, MTSC). Produces lovely results with the Selectric Composer, a very fancy Selectric. But has complications well beyond those of the Mag Tape Selectric. Uses more variable widths than Mag Card Executive. Uses same mag-film cartridges as MTST.

(Note: for those who like the output from the above devices, but appreciate also the relative difficulty of their use, there is available a computer peripheral device which reads and writes these IBM mag tape cartridges. I don't know who makes it, unfortunately.)

IBM's latest is called the Magnetic Memory Typewriter, and seems to store up to two pages in a hidden memory. Apparently you can't set it aside, like the cards or tapes.

A firm called Redactron makes magic typewriters using either cassettes (audio-type) or Mag cards (like the Mag Card Executive).

A firm called Vasle does the same thing, using a Tycom Selectric Sandwich (see under "Printing Terminals," nearby).

Olivetti has one called the TC-16 Word Processing System. Their cartridge (a disk?) stores, they say, 330 pages of typing.

Two other outfits in the field are Trendata and Quintype.

Whoops! Here comes Sperry Remington! (Sperry Remington?) They have one too.

For those interested in this sort of thing, there is an International Word Processing Association (Maryland Road, AMS Building, Willow Grove, PA 19090.)

See also the Flip Side of the book for more high-performance text systems.

# COMPUTER LANGUAGES

are what make computers go 'round.

If your computer only did one thing, then to start it you'd only need one button to press.

If your computer only did two dozen things, without variation, then you could let each operation be started by pressing one of the keys of the terminal, and that would be that.

But that's not what it's about.

We have lots of different things that we want computers to do, and we want one command to work on different varieties of data, or on the results of a previous command, or even to chew on another command itself; and so a computer language is a contrived method of giving commands to a computer that allows the commands to be combined in a complex fashion.

This means having rules the computer can carry out and the person can remember.

This means having basic operations that can be built into bigger operations (routines, subroutines, subprograms, programs).

Thus a computer language is really a method by which a user can tie these programs together. Computer languages are built according to contrived sets of rules for tying programs together. [illegible]

Computer languages tend to look like nothing else you've ever seen. [illegible]

One of the central concepts of this book is that of a "program follower," [illegible]

So the basic secret of computer people is this: it's not that they necessarily know so much, but they can adapt to a whole new world of possibilities more quickly.

PROGRAMS VS. SYSTEMS:
A Vague Guideline to a Vague Distinction

a "program" runs on an ordinary computer, without necessarily interacting with the outside world.

a "system" involves a whole setup, of which the computer and a program in it are just the central things.

# THREE COMPUTER LANGUAGES FOR YOU

Everyone should have some brush with computer programming. [illegible]

Why three languages? [illegible]

These three languages make it possible in principle for you to learn computer[s] [illegible]

Why these three? [illegible]

A computer language is a system for tying together the fundamental operations of computers for larger tasks. Each computer language ties together according to its own principles, based in part on the personality and preoccupations of the person or people who designed it.

Modern computer languages generally can handle all the main kinds of programming: text handling, number crunching, storing files on disk memory and getting them back, and controlling whatever external devices you may have. [illegible]

In this book we will try to give you a smattering of all these.

These languages have been chosen because they are important, very different from each other, very powerful, influential and highly regarded in the field, [illegible]

Each can be used to create programs for science, business or recreation.

Because these languages can be used from a terminal, and thus learned quickly, we might call them Quickie languages.

[illegible]

Another reason for these three: they represent, in a way, several major types.

BASIC is a widespread and fairly standard language [illegible]

TRAC Language, though well-known among researchers, [illegible]

Moreover, it is a so-called "list language," [illegible] (See "Data Structures," pp. 26.)

APL is another elegant language, [illegible]

In the contemplation of these three languages you may begin to see the influence of the individual human mind in the computer field, quite contrary to the stereotype. [illegible]

Two of these languages permit the creation of interactive programs that work on a line-by-line basis; in addition, TRAC Language (pp. 18-21) permits the creation of systems that react to one character the user types in, rather than waiting for the carriage return at the end of a line. [illegible]

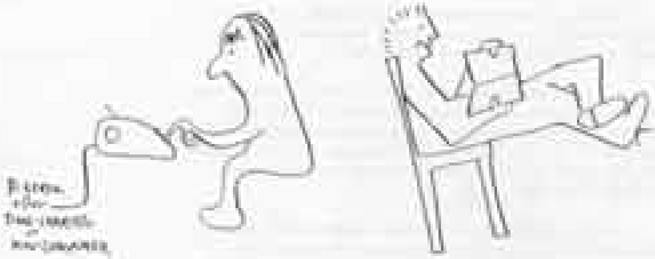
IF YOU'RE SCARED: Don't worry, it's not a test. Flip the pages and look at the examples. In particular, you might look for the same program which appears in each language: a program to cause the computer to print "HELP, I AM TRAPPED IN A LOOP" forever.)

This book is organized so you can look at it or skip it in any order, so there is no particular reason you have to fight through the next three chapters if you want to press on. But if you want to study these languages, by all means do so.

Languages that can be used from a terminal are called on-line languages. There are a number of other popular on-line languages, JOSS (the original), FOCAL, LOGO, SPEAKEASY. I'm just sorry there's no room for them here.

Some popular non-interactive languages are briefly described on pp. 30-31.

[illegible]



[illegible]

THE BEST WAY TO LEARN.

THREE Quickie Computer Languages:
BASIC (pp. 16-17)
TRAC© Language (pp. 18-21)
APL (pp. 22-5).

# "COMPUTER TEXT EDITORS"

The Moving Finger writes; and, having writ,
Moves on: nor all your Piety nor Wit
Shall lure it back to cancel half a Line,
Nor all your Tears wash out a Word of it.

Khayyam/Fitzgerald

Numerous interactive programs exist for editing text at computer terminals-- in other words, for doing what Magic Typewriters do, but using a computer instead of a small special-purpose machine.

Unfortunately most of these systems are dreadful. Dreadful, that is, for ordinary human beings. What computer people seem to think of as appropriate systems for handling text are totally unsuitable for people who care and think a lot about text, although they may be good for computer programmers.

Such systems allow you to insert text (with some difficulty), delete (with some difficulty), and rearrange (maybe).

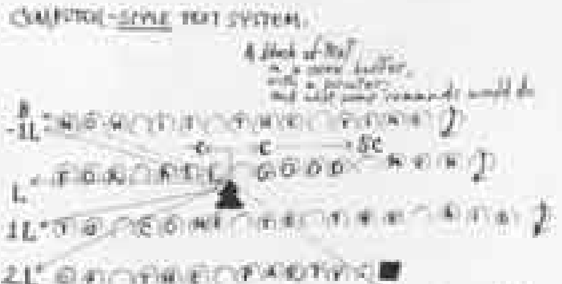
Ordinarily the user must learn an explicit command language, some system of alphabetical commands that have to be typed in to effect any change in the material. Programmers think this is good for you and toughens the mind.

The text is usually stored as a string of alphabetical and punctuation codes in the computer's core memory. The area it occupies in the core memory is called a core buffer.

The program generally gives the user an imaginary "pointer," a marker specifying what point in the text the program is currently concerned with.

What is the pointer for? It specifies where the operations are to take place. "Insert," for example. If text is inserted, it will go into the place presently pointed at.

Many of the commands are concerned with controlling the current position of the pointer, moving it backward or forward by a specific number of characters (including punctuation marks and spaces) or lines (known to the program by the carriage-return codes interspersed in the text).



In this simplified illustration, the pointer can be moved forward and backward in the text by various commands. Typing "B" moves the pointer to the beginning. "Z" takes it to the end. "L" moves it to the beginning of the line it's currently on, and the commands "C" and "L," when given with numbers, tell the pointer to move forward or back the specified number of positions. For instance:

| | |
|---|---|
| 3C | Move forward 3 characters |
| -4C | Move backward 4 characters |
| 2L | Move forward 2 lines |
| -1L | Move backward 1 lines |

and so on. Note that these operations are not god-given, but that the particulars of how they behave and work together are determined by the personal quirks of who programmed them.

Another feature many of these programs have is called a "context editor" feature. So-called context editing moves the pointer from its present position to the next occurrence of a specific string of characters: for instance, the next occurrence of the word [illegible]. Often such commands permit you, by giving the command properly, to replace any given word or phrase with any other. It was drily remarked at a recent conference that this would allow a writer to change every occurrence of "or" in his writing to "and." Yet programmers seem to think this is a feature writers want.

(For programmers' purposes this is a very good facility; indeed, a whole computer language, SNOBOL, is built around it; -- see p. 31. But it has nothing to do with normal text.)

This type of thing is totally unsuited for the literary types of people who care most about text and its characteristics (connotations, twists) which can not be found by definable structured search. And who should not be forced to deal with explicit computer languages because it tends to interfere with the thought processes they are supposed to be pursuing, if not make them physically ill.

# YOUR FIRST COMPUTER LANGUAGE: DARTMOUTH'S BASIC

The BASIC language, also called Dartmouth-Basic, was introduced in the sixties at Dartmouth College by John Kemeny and Thomas Kurtz. It was intended to be a simple and easy-to-learn introduction to computer programming, yet powerful enough to do useful things. It has grown in use, in recent years, both as the foremost beginner's language, and as a perfectly fine language for doing many simple kinds of work-- like custom business applications, statistics, and "good-guy" systems for nai,e users as discussed elsewhere in this book.

Kemeny is now president of Dartmouth, and Kurtz runs their high-power time-sharing computer center, so BASIC has a permanent home base there.

Note that the name BASIC does not refer to the bottom-level or elemental languages of computers. BASIC has been contrived specifically to make programming quicker and easier. It is not "basic" to all computers; such bottom languages are called "machine language" or "assembler language" (see pp. 33-5).

The simplicity of the language begins at the program input, or editing, level. Each command of BASIC must be on a separate line, and each line must have a separate line number. Suppose you accidentally type in

```
50    INPUGN  Y
```

when you meant "INPUT" instead of "INPUGN." You may replace that command at any time by typing the same line number and the new version of the line,

```
50    INPUT  Y
```

which automatically replaces the previously line 50. If you want to get rid of the line entirely, you type

```
50
```

and an end-of-line code, and the whole line is gone.

Example of a BASIC command:

```
120    LET X = Y
```

You can choose any line numbers you want, but the lines are automatically put in the order of their numbers. Since when you write a program you don't usually know at the outset what it will look like later, you try to leave enough gaps in the numbers at the start to fit in the instructions you might want to put between them later.

THE SETTING

To begin with, there must be a computer, and it must have a processor for the BASIC language, that is, a program for carrying out the operations of Dartmouth-BASIC. We will assume that this BASIC processor is all set up in core memory ready to go.

(Note: This is how it looks in a minicomputer. On a time-sharing system there's a lot of irrelevant other stuff going on, which we'll leave out.)

And we will assume, as previously mentioned, that you have some kind of a terminal-- that is, a device with a keyboard, some kind of place the computer can send messages to you and vice versa, and is more or less standard.

Now then: all that is needed is for you to understand the BASIC language, and you can program this computer within the confines of BASIC.

➡ It is one of the strange aspects of this field that languages can be taught independently of discussions of the machine itself.

When you type in a program, the BASIC processor will do certain things to it (actually cook it down) and store it in core memory:

Every time you change one of the lines of the program the BASIC processor will insert, delete or replace lines as you have commanded, then rearrange whatever's left accordingly, in order of the line numbers.

Then when you tell the processor to start the program, by typing (with no line number)

```
RUN
```

the processor will start the program going at the command with the earliest line number, and your instructions will be executed according to the rules of BASIC.

Now we will consider some of the commands (or statements) of BASIC.

These two boys had never seen a computer before, but I loaded it up with the BASIC language processor, showed them a few basic commands and told them to turn it off when they were through.

I got back ten hours later and they were still at it. Too bad kids have such short attention spans.

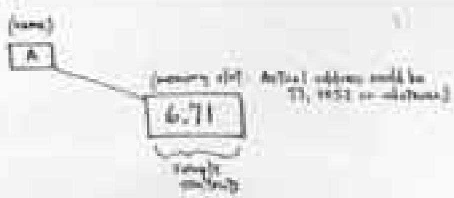
VARIABLES

The BASIC language, like a number of other languages, allows you to set aside places in core memory and give them names. These places may hold numbers. They can be used to count the number of times that things are done (or not done), to hold answers, numbers to test against, numbers to multiply by and so on.

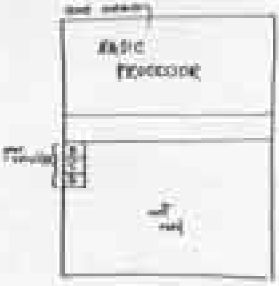
In BASIC, these places are given names of one alphabetical letter. That means you can have up to 26 of them. Examples:

A   E   I   O   U   sometimes Y   even Z

Because these named spaces in memory may be used something like the way letters are used in algebra, we call them variables. In fact, each one is a place with a name.



If you use the names B,C and D for variables in your program, the BASIC processor will automatically set up places for them to be stored.



The END command

The END command in BASIC simply consists of the word END. It must come last in the program. Therefore it must have the highest line number. Example:

```
99    END
```

The PRINT command

Whenever the program follower gets to a PRINT command, it prints out on the terminal whatever is specified. Example:

```
97    PRINT "HAIL CAESAR.  HEED THOU NEVER WENT"
```

When and if the program follower gets to this command, the terminal will print out

HAIL CAESAR.  HEED THOU NEVER WENT

The GOTO command (pronounced "Go 2")

The GOTO command tells the program follower the number of the next command for it to do, from which it will go on. Example:

```
82    GOTO    99
```

which means that when a program follower gets to command #82, it must next jump to 99 and go on from there, unless that happens to be the END statement.

A SIMPLE SAMPLE PROGRAM

These are enough commands to write a sample program.

```
43    PRINT  "HELP, I AM CAUGHT IN A LOOP"
47    GOTO   43
68    END
```

The program will start at the first instruction, which happens in this case to be instruction number 43. That one prints a message. The next command, by line number, is 47. This tells the program follower to go back to 43, which it does.

```
43    PRINT  "HELP, I AM CAUGHT IN A LOOP"
47    GOTO   43
68    END
```

The result is that your terminal will print

```
HELP, I AM CAUGHT IN A LOOP
HELP, I AM CAUGHT IN A LOOP
HELP, I AM CAUGHT IN A LOOP
```

interminably, or until you do something drastic. It never gets to the END statement. (Two strategies for doing something drastic are usually to hold down the CONTROL button and type C, or hold down both CONTROL and SHIFT buttons, if you have them, and type P. One of these usually works.)

The LET command

The LET command puts something into a variable. Example:

```
63    LET R = 2.3
```

What is on the right side of the equals sign in the last statement, in this case 2.3, is stuffed into whatever location of core memory is designated on the left side, in this case a place known to you only as R. With the result that someplace in core memory is

The LET statement is an example of an assignment statement, which most computer languages have; an assignment statement assigns a specific piece of information (often a number, but often other things) to some name (often standing for a particular place in core memory).

The LET command in BASIC can also be used to do arithmetic. Example:

```
14    LET M = 2.3 * (17*999.1)
```

(The asterisk has to be used for multiplication because traditionally terminals don't have a times-sign.) BASIC will work this out from right to left and store the result in M.

The INPUT command

The INPUT statement asks the person at the terminal for a number and then shoves it into a variable. Example:

```
61    INPUT  Z
```

which causes the terminal to type a question mark, and wait. When the user has typed in a number followed by a carriage return, the BASIC processor stuffs the number into the variable and proceeds with the program. Here is a program using the INPUT statement

```
10  PRINT "HOW OLD ARE YOU"
15  INPUT A
20  LET B = A/48.0
25  PRINT "YOUR AGE IS", B, "TIMES THE AGE
    OF THE EMPIRE STATE BUILDING."
30  END
```

This will cause the following to happen:

Program types:
HOW OLD ARE YOU? 20

Program types:
YOUR AGE IS .5 TIMES THE AGE OF THE EMPIRE STATE BUILDING.

The IF command

The IF command is a way of testing what's stored in a variable. Example:

```
50  IF M = 40 then 83
```

This tests variable M to see if it contains the number 40. If M is indeed 40, the program follower jumps to line 83. If not, it goes right on and takes the next higher instruction after 50. The IF can test other relations than equality, including "less than," "greater than," "not equal," "less than or equal to," etc. For instance,

```
90  IF Q < 7 then 102
```

will send the program follower to command 75 if variable Q contains a number less than 7. (Note that different BASICs for different computers may have slightly different rules here.)

The BASIC language, developed at Dartmouth, must not be confused with the underlying binary languages of individual computers (see "Rock Bottom," p. 31). These underlying codes are called "machine languages" (or, in a dressed-up form, easier to use for programmers, "assembler language"). These are the basic languages, different for each machine. Dartmouth BASIC, or just plain Basic, is a widely available, standardized, simple beginner's language.

ANOTHER PROFOUND EXEMPLARY PROGRAM

```
5   LET Z = 25
10  PRINT Z, " BOTTLES OF BEER IN THE WALL"
15  LET Z = Z - 1
62  IF Z = 0 GOTO 74
63  GOTO 10
74  PRINT "TIME TO GO HOME."
75  END
```

The program will start typing thusly:

25 BOTTLES OF BEER IN THE WALL
24 BOTTLES OF BEER IN THE WALL

and so on, until Z has reached 0. Then it will type

0 BOTTLES OF BEER IN THE WALL
TIME TO GO HOME.

and then it will stop.

You will note that this program, like the one that printed "HELP, I AM CAUGHT IN A LOOP," has a loop, that is, a repeated sequence of operations. The first one was an endless loop, which repeated forever. This loop, however, is more well-behaved (by some people's standards), in that it allows an escape when a certain criterion has been reached— in this case, printing a line of text 25 times with variants.

The reason we are able to escape from this loop is that we have a test instruction, IF statement number 62.

It is very important for the programmer to include tests which allow the program to get out of a loop. This may be couched as a motto, viz.:

LEAK BEFORE YOU LOOP.

AN AUTOMATIC LOOP

Indeed, for people who are big on program loops, BASIC provides a pair of instructions which handle the program loop completely. These are the FOR and NEXT instructions. We won't show them here, but they're not very hard. Using the FOR command, you can easily direct the computer to do something a million and one times, say. This can be exhilarating. You can even direct it to include that program in something to be done a billion times, resulting in a program loop that would be carried out over a trillion times. All in a short program! But of course this is just power on paper; we want our programs to be useful, and finish their jobs in the present century, and so such flights are just mental exercises.

FAST ANSWERBACK WITH BASIC (in some versions)

If you want a fast answer to a numerical question, you can do it without the line numbers, typing in

```
PRINT   3.1416 * 7124
```

will cause BASIC to print the answer right out and forget the whole thing.

TEXT STRINGS IN BASIC

The deluxe versions of the Dartmouth BASIC language have operations for handling text— or what computerfolk call "strings," that is, strings of alphabetic characters and punctuation. These operations tend to begin with $ (standing for "$tring"?) and there's no room for them here.

But what they mean is that BASIC can type letters, count the words in Gone With The Wind, or print out the nine hundred million names of God.

If you write the program.

THIS IS A SERIOUS LANGUAGE,
AND CAN SAVE SOME COMPANIES A LOT OF MONEY

BASIC is a very serious language. Advanced versions of BASIC have instructions that allow users to put in alphabetical information, and store and retrieve all kinds of information from disks or tape. In other words, BASIC can be used for the fairly simple programming of a vast range of problems and "good-guy systems" mentioned elsewhere. Complete BASIC systems allowing complex calculations can be had for perhaps $3000; a general-purpose computer running BASIC with cassette or other mass storage, for business or other purposes, can now be had for some $6000. Allowing a few thousand dollars for programming specific applications in BASIC, simple systems can be created for a variety of purposes that some companies might say you needed a hundred-thousand-dollar system for.

This is serious business. Languages like BASIC must be considered by people who want simple systems to do understandable things in direct ways that are meaningful to them, and that don't disrupt their companies or their lives.

This has been a very hasty and brief presentation in which I have tried to convey the feeling of this important language. If you have the chance to learn it, by all means do.

SOME FUN THINGS TO TRY IN BASIC

Write a program that prints calendars.

Write a program that converts an input number to Roman Numerals.

Write a dialogue system that welcomes the user to the sanitarium, asks him questions, ignores the answers and insults him. (Use the INPUT statement for receiving numerical answers. Since the answers are ignored they can all be stored in one variable.)

WHERE TO GET IT

(Features of the BASIC language vary considerably from system to system. Which ones offer the highly desirable alphabetic commands and mass storage have to be checked out individually.)

BASIC is offered on many if not most time-sharing services, so you can use it from your home on a terminal. (But note that this can be expensive and even dangerous, if you're paying yourself; there are not presently adequate cost safeguards to prevent you from running up huge bills.)

BEST BUY? Rumors persist of a time-sharing service somewhere that offers BASIC for $5 an hour, total, with disk storage thrown in. I have not been able to verify this.

DEC offers minicomputer-based systems which time-share BASIC among several terminals simultaneously. (But you have to buy the whole big system.) The ones that run on the PDP-8 are marketed mainly to schools, and for this reason are called, somewhat peculiarly, EDUSYSTEMS. Their multiterminal system for the PDP-11 is called RSTS (pronounced "Ristiss,") and is marketed mainly to businesses.

Hewlett-Packard offers BASIC, I believe, on all of its minicomputers. Of special interest is an odd computer called the Series 9800 Model 30. You're only allowed to program in BASIC. (It's actually a microprocessor; see p. 77.)

Many other minicomputer manufacturers now offer BASIC. Data General's NOVA is one.

BIBLIOGRAPHY


Kemeny and Kurtz, BASIC Programming. Wiley, 1967.

DEC's EDUsystem Handbook is a very nice introduction to BASIC, quite pleasant and whimsical; it may be a good introduction even if you're using other people's BASIC systems. It's $3 from DEC, Communications Services, Parker St., Maynard, Mass. 01754.

There is also a programmed text on BASIC by Albrecht (published by Wiley). For those of us who freeze at numerical-looking manuals, programmed texts can take away a lot of anxiety.

MY COMPUTER LIKES ME (when I speak in BASIC). This book has evidently been put together by the People's Computer Company, and has some idealistic fervor behind it. $1.19 from Dymax, Box 310, Menlo Park, Cal. 94025.


BASIC is a good example of an "algebraic" type of language, that is, one formulated more or less to look like high-school algebra and permit easy conversion of certain algebraic formulas into actual runnable programs.

The most widely-used language of this type is FORTRAN (see p. 77). Thus BASIC is often referred to as a "Fortran-type language."

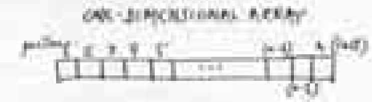
The kicker— and if you understand this it's half the battle— is that a line of BASIC or FORTRAN directs a certain event to take place, while a statement in algebra just describes relations.

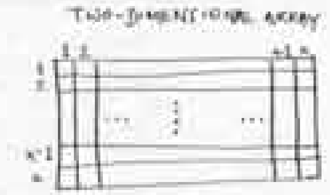
The strange resemblance between the descriptive language (algebra) and the prescriptive language (Fortran or Basic) is that algebraic operations (which are just recombinations and restatements) can be mimicked by the computer language, and this early obsession of many computerfolk led to making the computer language look like a descriptive algebra. Especially with the weird use of the equals-sign to mean "is replaced now by." In hindsight this was a ridiculous idea; some of the more recent languages (such as APL) use a left-pointing arrow instead of an equals-sign, showing that an action is being called for, rather than a relationship being described.

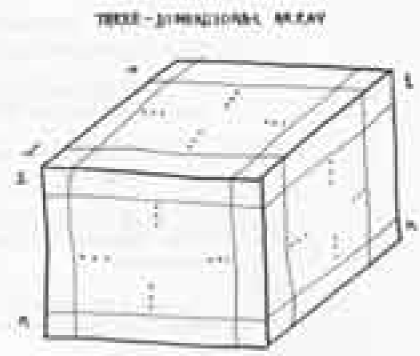
# ARRAYS, an important data structure

(available in BASIC, APL and many other languages)

Arrays are information setups with numbered positions. The positions can contain all sorts of different things, however: numbers, letters or other data, depending on the data structures allowed in the language.



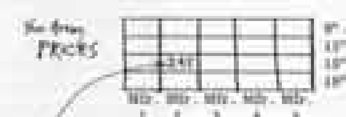




A one-dimensional array is like a row, a two-dimensional array is like a tabletop, a three-dimensional array is like a box; and for more dimensions you can't visualize.

Arrays are handy for working with a lot of different things one at a time. They can be given names just like variables.

Suppose you have a one-dimensional array named SAM. Then in a program you can actually ask for the third element in SAM by referring to SAM(3). Better than that: you can refer by turns to every element of SAM by using a counting variable and changing its value. SAM(JOE) can be any one of the elements of the array, if we set the value of JOE, the counting variable, to the number of the position we want to point to.

For arrays having more than one dimension, the principle is the same. You may refer in a program to any space in the array by giving a number in parentheses, or subscript, specifying the space's position in each dimension. Suppose you have an array named PRICES, which gives the prices of, say, various sizes and brands of TV sets.




This is PRICES(3,2)
because it's the item in row 3, column 2.

Suppose you have a two-dimensional array giving the telephone numbers, salaries and ages of several different employees of a company. You have decided to call the array WHAM.


You can refer to any single entry in this array as WHAM(IRV,JOE), where IRV and JOE are two counting variables you've decided to set up.

If you set IRV and JOE both to 1, WHAM(IRV,JOE) is really WHAM(1,1), which refers you to the telephone number of employee A. If you change JOE to 2, that gives you WHAM(1,2), giving you B's phone; while WHAM(2,1) would be A's salary.

These are just the mechanics. What you choose to do with this sort of thing is your own affair. Counting around in arrays (and core memory, where they're stored) is called indexing.

# THE SLEEPING GIANT
# TRAC* Language

A mild-mannered man in Cambridge, Massachusetts, who owns his own very small business, is the creator of one of the most extraordinary and powerful computer languages there is, though lots of people in the field don't realize it. The language is fairly well-known among professionals, but its real power is hardly suspected.

If BASIC is a fairly conventional programming language, strongly resembling FORTRAN, TRAC (Text Reckoning and Compiling) Language is fairly unusual.

The name of it is "TRAC Language," not just TRAC — because it's a registered brand name (like Kleenex Tissues). Within the rules, the word "TRAC" is an adjective and not a noun. Thus TRAC is its first name, Language is its last; so we can refer to "TRAC Language" instead of having to precede it with the.

It is included here for several reasons.

1) It is extremely easy to learn, at least for beginners. Experienced programmers often have trouble with it.

2) It is extremely powerful for non-numeric tasks. In fact, it is ideal for building your own personal language.

3) It offers perhaps the best control of mass storage, and your own style of input-output, of any language.

4) It is superbly documented and explained with the new "The Beginner's Manual for TRAC Language," which is now available.

5) It is likely to catch on one of these days. (Some large corporations have been investigating it extensively.)

---

It is not so much the basic idea of TRAC Language, but the neatness with which the idea has been elaborated, that is so nice.

As a side point, here is an important motto for thinking in general about computers (and about other things in general):

MAKING THINGS FIT TOGETHER WELL
TAKES A LOT OF WORK AND THOUGHT.

Let Calvin Mooers' TRAC Language be a shining example.

---

TRAC Language is great for creating highly interactive systems for special purposes, including turnkey systems for inexperienced users and "good-guy" systems. It combines this with good facilities for handling text, and what is needed along with that, terrific control over mass storage. It is also excellent for simulating complex on-off systems; rumor has it that TRAC Language was used for simulating a major computer before it was built.

Against these advantages we must balance TRAC Language's less fortunate characteristics. For numerical operations it is extremely slow, if not terrible, compared to the most popular languages. The same applies to handling numerical arrays and controlling loops, which are comparatively awkward in TRAC Language.

Finally, many programmers are incensed by the number of parentheses that turn up in TRAC programs; in this it resembles the language LISP. But this is an aesthetic judgement.

The TRAC Language has been thought out in great detail for total compatibility of all parts. (Moreover, by standardizing the language exactly, Mooers heroically assures that programs can be moved from computer to computer without difficulty.)

In the well-thought-out ramifications of its basic concept, the TRAC Language is so elegant as to constitute a work of art. It beautifully fulfills this rule:

> "... the facilities provided by the language should be constructed from as few basic ideas as possible, and ... these should be general-purpose and interrelated in the language in a way which avoided special cases wherever possible." (Harrison, Data-Structures and Programming, pub. Scott, Foresman, p. 251.)

The fundamental idea of TRAC Language, which has been worked out in detail with the deepest care, thought and consistency, is this:

ALL IS TEXT.

That is, all programs and data are stored as strings of characters, in the same manner. They are labelled, stored, retrieved, and otherwise treated in the same way, as strings of text characters.

Data and programs are not kept in binary form, but remain stored in character form, much the way they were originally put in. The programs are examined for execution as text strings, and they call data in the form of text strings.

This gives rise to certain interesting kinds of compatibility.

a) Complete compatibility exists in the command structure: the results of one command can become another command or can become data for another command. ALMOST NOTHING CREATES AN ERROR CONDITION. If enough information is not supplied to execute a command, the command is ignored. If too much information is supplied, the extra is ignored.

b) Complete compatibility exists in the data: letters and numbers and spaces may be freely intermixed. Special terminal characters (like carriage returns and backspaces) are handled just like other characters, giving the programmer complete control of the arrangement of output on the page.

c) Complete compatibility also exists from one computer to another, so that work on one computer can be moved to another with ease. By the trademark TRAC, Mooers guarantees it — an innovation.

COMMAND FORMAT

A TRAC command has the following form. The cross-hatch or sharp-sign is the way this language identifies a command's beginning.

#(NM, arg2, arg3, arg4, ...)

NM is the name of any TRAC command. It counts as the first "argument," or piece of information supplied. Arg2, arg3, etc. are whatever else the command needs to know to be carried out.

We will look first at examples that use the arithmetic commands of TRAC Language, not because it is particularly good at arithmetic, which it isn't, but because they're the simplest commands. The arithmetic commands are AD (add), SU (subtract, ML (multiply), DV (divide). Each arithmetic command takes three arguments, the command name and two numbers. Examples:

#(AD, 1, 2)

is a command to add the numbers 1 and 2.

#(SU, 4, 3)

is a command to subtract the number 3 from the number 4.

#(ML, 632, 521)

is a command to multiply 632 by 521.

#(DV, 100, 10)

is a command to divide 100 by 10.

Now comes the interesting part.

The way TRAC commands may be combined provides the language's extraordinary power. This is based on the way that the TRAC processor examines the program, which is a string of character codes. Watch us we combine two AD instructions:

#(AD, 3, #(AD, 2, 5))

The answer is 10. Miraculous!

How can this be?

> A comma ends an argument
> in the TRAC language?
> Ah, that all arguments
> could be ended so easily.
> —My grandfather.

---

* TRAC is a registered service mark of Rockford Research, Inc. Description of TRAC Language primitives adapted by permission from "TRAC, A Procedure-Describing Language for the Reactive Typewriter", copyright © 1966 by Rockford Research, Inc.

---

I am grateful to C.A.R. Kagan, of Western Electric Engineering Research Center, for his extensive (and finally successful) efforts to interest me in TRAC Language.

# Mild-mannered Calvin Mooers steps into a phone booth, tears open his terminal, and
# #(POW!)
# IT'S SUPERLANGUAGE!

THE MAGIC SCAN

The secret of combining TRAC commands is that every command, when executed, is replaced by its answer; and whatever may result is in turn executed.

There is an exact procedure for this:

```
SCAN FROM LEFT TO RIGHT
      UNTIL A RIGHT PARENTHESIS;
*RESOLVE THE CONTENTS OF THE
      PAIRED COMMAND PARENTHESES
      (execute and replace by the command's result);
STARTING AT THE BEGINNING OF THE RESULT,
KEEP SCANNING LEFT-TO-RIGHT
      UNTIL A RIGHT PARENTHESIS.
WHEN YOU GET TO THE END, PRINT OUT
      WHAT'S LEFT.
```

The beauty part is how it all works so good.

An arithmetic example — so you get the procedure.

```
#(AD, 2, #(AD, 3, 4))
                    -> )    first right parenthesis found.
           7                execute what's in the command parentheses
                            & replace with their answer, leaving:
#(AD, 2, 7)
           )                scan to next right parenthesis
      9                     execute & replace
                            find no more parentheses
                            print out what's left.
```

You might try this yourself on a longer example:

#(AD, #(SU, #(AD, 3, 4), #(SU, 7, 2)), 1)

Here is an interesting case:

#(AD, 1)

There's no third argument to add to the 1 — but that's okay in TRAC Language. 1 it remains.

PULLING IN OTHER STUFF

The core memory available to the use is divided into two areas, which we may call WORKSPACE and STANDBY.

| |
|---|
| #(SE, #(AD, 7, 3), #(SU, 14, 8)) — WORKSPACE |
| STANDBY — Strings with Names |

The Standby area contains strings of characters with names. Here could be some examples:

| names | strings |
|---|---|
| HAROLD | 54321 |
| SUE | 7.14 |
| PROGRAM | #(PS, HELP: I AM TRAPPED IN A LOOP)#(CL, PROGRAM) |
| GALOSHES | I MUSTN'T FORGET MY GALOSHES. |

There is an instruction that moves things from the Standby area to the Workspace. This is the CALL instruction.

#(CL, whatever)

The CALL instruction pulls in a copy of the named string to replace it, the call instruction, in the work area. The string named in the call instruction also stays in the Standby area until you want to get rid of it. Example:

#(CL, HAROLD)

would be replaced by

54321

Suppose we say in a program

#(AD, 1, #(CL, HAROLD))

Then the result is:

54322

Now let's do a program loop using the CALL. If we type in to our TRAC processor

#(CL, PROGRAM)

it should type

HELP; I AM TRAPPED IN A PROGRAM LOOP
HELP; I AM TRAPPED IN A PROGRAM LOOP
HELP; I AM TRAPPED IN A PROGRAM LOOP

indefinitely.

Why is this? Let's go through the steps.

We noted that in our Standby area we had a string named PROGRAM which consisted of

#(PS, HELP; I AM TRAPPED IN A PROGRAM LOOP)#(CL, PROGRAM)

The TRAC processor scans across it to the first right parenthesis,

#(PS, HELP; I AM TRAPPED IN A PROGRAM LOOP)#(CL, PROGRAM)

and now executes this.

It happens that PS is the PRINT STRING instruction. PRINT STRING prints out its second argument, and forgets the rest. But the only argument after PS is

HELP; I AM TRAPPED IN A PROGRAM LOOP

so it prints that. If it had said

HELP, I AM TRAPPED IN A PROGRAM LOOP

the PRINT STRING command would only have printed

HELP

since a comma ends an argument in TRAC language.

Now, the PRINT STRING command leaves no result, so it is vaporized; all we have left in the work area is

#(CL, PROGRAM)

which is now scanned. But that's another CALL, and when it is executed by fetching the object called PROGRAM, its replacement in the work area is

#(PS, HELP; I AM TRAPPED IN A PROGRAM LOOP)#(CL, PROGRAM)

and guess what. We does it again.

(Another example of TRAC Language's consistency: suppose it executes the command

#(CL, EBENEZER)

when there is no string called EBENEZER. The result is nothing; so that command disappears, leaving no residue.)

THE FORM COMMANDS

Let us be a little more precise. The Standby area is really called by Mooers "forms storage," and a string-with-name that is kept there is called a form. One reason for this terminology is that these strings can consist of programs or arrangements that we may want to fit together and combine. Thus they are "forms".

1. CREATING A FORM

To create a form, you use the DEFINE STRING command:

#(DS, formname, contents)

The arguments used by DS give a name to the form and specify what you want to have stored in it. Example:

#(DS, ELVIS, 1234)

creates a form named ELVIS with contents 1234.

ELVIS → 1234

(Note that to get a program into a form without its being executed on the way requires some preparation. For this, "protection" is used; see end of article.)

It turns out that DEFINE STRING is the closest TRAC Language has to an assignment statement (as in BASIC, LET A = WHATEVER). If you want to use a variable A, say, to store the current result of something, in TRAC Language you create a form named A.

#(DS, A, WHATEVER)

Whenever the value of A is changed, you redefine form A.

2. CALLING A FORM.

As noted already,

#(CL, ELVIS)

will then be replaced by

1234

But a wonderful extension of this, that hasn't been mentioned yet, is

2A. THE IMPLICIT CALL.

You don't even have to say CL to call a form. If the first argument of a command — that is, the first string inside the command parentheses — is not a command known to TRAC Language, why, the TRAC processor concludes that the first argument may be the name of a form. So now if you type

#(AD, #(HAROLD), #(ELVIS))

it will first note, on reaching the right-paren of the HAROLD command, that since HAROLD is 54321, you evidently wanted this:

#(AD, 54321, #(ELVIS))

rescan of result

and then will do the same with ELVIS:

#(AD, 54321, 1234)

so that pretty soon it'll type for you

55555

This language is marvelously suited to data base management, management information systems, interactive query systems, and the broad spectrum of "business" programming.

For large-scale scientific number crunching, not so good.

With one exception: "infinite precision" arithmetic, when people want things to hundreds of decimal places.

Chugga chugga

---

This implicit call is the trick that allows people to create their own languages very quickly. In not very long, you could create your own commands — say ZAPP, MELVIN and some more; and while at first it is more convenient to type in the TRAC format

#(ZAPP, #(MELVIN))

it is very little trouble in TRAC Language to create new syntaxes of your own like

ZAPP ! MELVIN

that are interpreted by the TRAC processor as meaning the same thing.

III. FILLING IN HOLES

Another thing the CALL command in TRAC Language does is fill in holes that exist in forms. Let us represent a hole as follows:

[ ]

Now suppose there is a TRAC form with a hole in it, like this.

WORD → H[ ]T

Additional arguments in the call get plugged into holes in the form. Examples:

| call | result |
|---|---|
| #(CL, WORD) | HT |
| #(CL, WORD, O) | HOT |
| #(WORD, A) | HAT |
| #(WORD, OO) | HOOT |

Now, a form can have a number of different holes. Let us denote these by

[1] [2] [3] [4] . . .

Now suppose we have a form:

WORD → [1]H[2]T[3]

which we might call numerous ways:

| call | result |
|---|---|
| #(WORD, W, I, E) | WHITE |
| #(WORD, , OO, OWL) (Note that putting nothing between two commas made nothing the argument.) | HOOTOWL |
| #(WORD, #(WORD, , OH, O)) | HOTHOT |

Perhaps you can think of other examples.

This fill-in technique is obviously useful for programming. If a form contains a program, its holes can be made to accept varying numbers, form names, text strings, other programs. Example: Suppose we want to create a new TRAC command, ADD, that adds three numbers instead of just two. Fair enough:

ADD → #(AD, [1], #(AD, [2], [3])) and there you are.

This brings up another example of how nicely TRAC Language works out. Suppose you have the following in forms storage:

ZOWIE → #(ZIP, [1], [2])
ZIP → #(ZAP, [1], [2])
ZAP → #(AD, [1], [2])

Try acting this one out with pencil and paper. Suppose you type in

#(ZOWIE, 5, 7)

It happens that the arguments 5 and 7 will be passed neatly from ZOWIE to ZIP to ZAP to the final execution of the AD; all through the smooth plugging of holes by the implicit call and the Magic Scan procedure of the TRAC processor.

**************************************************************

TRAC Language is a so-called "list processing language" or "List Language." This term has come to mean any language for handling data having arbitrary and changing form. Two other prominent languages of this type are SNOBOL and LISP (see p. 31).

List languages are traditionally freaky.

TRAC Language is:

an interpretive language
(each step carried out directly by the processor without conversion to another form first);
an extensible language
(you can add your own commands for your own purposes);
a list-processing language
(for handling complex and amorphous forms of data that don't fit in boxes and arrays).
It is one of the few such languages that fits in little computers.

3. DRILLING THE HOLES

The holes (called by Mooers segment gaps) are created by the SEGMENT STRING instruction.

#(SS, formname, whatever1, whatever2 ...)

where "formname" is the form you want to put holes in and the whatevers are things you want to replace by holes. Example: Suppose you have a form

[INSULT] → [YOU ARE A CREEP]

You make this more general by means of the SEGMENT STRING instruction:

#(SS, INSULT, CREEP)

resulting in

[INSULT] → [YOU ARE A [ ]]

which can be filled in at a more appropriate time.

Fuller example. Suppose we type into the TRAC processor the following:

#(DS, THINGY, ONE FOR THE MONEY AND TWO FOR THE SHOW)
#(SS, THINGY, ONE, TWO, )
note space

We have now created a form THINGY and replaced parts of it with segment gaps. Since each of the later arguments of SEGMENT STRING specifies a differently numbered gap, we will have gaps numbered [1], [2], and [3]. The gap [1] will have replaced the word ONE, the gap [2] will have replaced the word TWO, and a lot of gaps numbered [3] will have replaced all the spaces in the form (since the fifth argument of SS was a space). The resulting form is:

[THINGY]
[1][3]FOR[3]THE[3]MONEY[3]AND[3][2][3]FOR[3]THE[3]SHOW

We can get it to print out interestingly by typing #(CL, THINGY, RUN, HIDE) (since after the call, the plugged-in form will still be in the forms storage.) This is printed:

RUNFORTHEMONEYANDHIDEFORTHESHOW

or perhaps, if we use a carriage return for the last argument, we can get funny results. The call

#(THINGY, NOT A FIG, THAT, [carriage return]
)

should result in

NOT A FIG
FOR
THE
MONEY
AND
THAT
FOR
THE
SHOW

In TRAC Language, every command
is replaced by its result
as the program's execution proceeds.
This is ingenious, weird and highly effective.


TEST COMMANDS IN TRAC LANGUAGE

There are test commands in TRAC Language, but like everything else they work on strings of characters. Thus they may work on numbers or text. Consider the EQ command (test if equal):

#(EQ, firstthing, secondthing, ifso, ifnot)

where "firstthing" and "secondthing" are the strings being compared, and ifso and ifnot are the alternatives. If firstthing is the same as secondthing, then ifso is what the TRAC processor does, and ifnot is forgotten. Example:

#(EQ, 3, #(SU, 5, 2), HOORAY, NUTS)

If it turns out that 3 is equal to #(SU, 5, 2), which it is, then all that would be left of the whole string would be

HOORAY

while otherwise the TRAC processor would produce NUTS.

To most computer people this looks completely inside-out, with the thing to do next appearing at the center of the test instruction. Others find this feature at-trac-tive.

DISK OPERATIONS

Now for the juicy disk operations. Storing things on disk can occur as an ordinary TRAC command.

#(SB, name, form1, form2, form3 ... )

creates a place out somewhere on disk with the name you give it, and puts in it the forms you've specified. Example:

#(SB, JUNK, TOM, DICK, HARRY)

and they're stored. If you want them later you say

#(FB, JUNK)

and they're back.

Because you can mix the disk operations in with everything else so nicely, you can chain programs and changing environments with great ease to travel smoothly among different systems, circumstances, setups.

Here is a stupid program that scans all incoming text for the word SHAZAM. If the word SHAZAM appears, it clears out everything, calls a whole nother disk block, and welcomes its new master. Otherwise nothing happens. If you have access to a TRAC system (or really want to work on it), you may be able to figure it out. (RESTART must be in the workspace to begin.)

[RESTART] → [#(DS, TEMP, #(RS))#(SS, TEMP, )#(RPT)]

[RPT] → [#(EQ, SHAZAM, #(TEST), (#(EVENT)))#(RPT)]

[TEST] → [#(CS, TEMP, (#(RESTART)))]

[EVENT] → [#(DA)#(FB, MARVEL)#(PS, WELCOME O MASTER)]

In this example, however, you may have noticed more parentheses than you expected. Now for why.

PROTECTION AND ONE-SHOT

The last thing we'll talk about is the other two syntactic layouts.

We've already told you about the main syntactic layout of TRAC Language, which is

#(          )

It turns out that two more layouts are needed, which we may call PROTECTION and ONE-SHOT. Protection is simply

(          )

which prevents the execution of anything between the parentheses. The TRAC processor strips off these plain parentheses and moves on, leaving behind what was in them but not having executed it. (But it may come back.) An obvious use is to put around a program you're designing:

#(DS, PROG, (#(AD, A, B)))
safe
stripped    stripped

but other uses turn up after you've experimented a little. The last TRAC command arrangement looks like this

##(          )

and you can put any command in it, except that its result will only be carried one level.

##(CL, ZOWIE, 3, 4)

results in (using the forms we defined earlier),


which is allowed to survive as is, because the moving finger of the TRAC scanner does not re-scan the result.

It is left to the very curious to try to figure out why this is needed.

> Whatever can be executed
> is replaced by
> its result.
> This may or may not
> yield something
> which is in turn
> executable.
> When nothing left is executable,
> what's left
> is printed out.
>
> That's the TRAC language...

## FAST ANSWERBACK IN TRAC LANGUAGE

TRAC Language can be used for fast answerback to simple problems. Typing in long executable TRAC expressions causes the result, if any, to be printed back out immediately.

For naive users, however, the special advantage is in how easily TRAC Language may be used to program fast answerback environments of any kind.

## A SERIOUS LANGUAGE; BUT BE WILLING TO BELIEVE WHAT YOU SEE

TRAC Language is, besides being an easy language to learn, very powerful for text and storage applications.

Conventional computer people don't necessarily believe or like it.

For instance, as a consultant I once had programmed, in TRAC Language, a system for a certain intricate form of business application. It worked. It ran. Anybody could be taught to use it in five minutes. The client was considering expanding it and installing a complete system. They asked another consultant.

It couldn't be done in TRAC Language, said the other consultant; that's some kind of a "university" language. End of project.

## HOW TO GET IT

There have been, until recently, certain difficulties about getting access to a TRAC processor. Over the years, Mooers has worked with his own processors in Cambridge. Experimenters here and there have tried their hands at programming it, with little compatibility in their results. Mooers has worked with several large corporations, who said said they wanted to try processors to assess the value of the the language, but these endeavors brought nothing out to the public.

FINALLY, however, TRAC Language service is publically available, in a fastidiously accurate processor and with Mooers' blessing, on Computility™ timesharing service. They run PDP-10 service in the Boston-to-Washington area. (From elsewhere you have to pay long distance.) The charge should run $12 to $15 per hour in business hours, less elsewhen. But this depends to some extent on what your program does, and is hence unpredictable. A licensed TRAC Language processor may be obtained from Mooers for your own favorite PDP-10. Processors for other computers, including minis, are in the planning stage.

TRAC Language is now nicely documented in two new books by Mooers, a beginner's manual and a standardization book (see Bibliography).

Since Mooers operates a small business, and must make a livelihood from it, he has adopted the standard business techniques of service mark and copyright to protect his interests. The service mark "TRAC" serves to identify his product in the marketplace, and is an assurance to the public that the product exactly meets the published standards. By law, the "TRAC" mark may not be used on programs or products which do not come from Rockford Research, Inc.

Following IBM, he is using copyright to protect his documentation and programs from copying and adaptation without authority.

Mooers also stands ready to accommodate academic students and experimenters who wish to try their hands at programming a TRAC processor. An experimenter's license for use of the copyright material may be obtained for a few dollars, provided you do not intend to use the resulting programs commercially.

For information of all kinds, including lists of latest literature and application notes, contact:


Calvin N. Mooers
Rockford Research, Inc.
140-1/2 Mount Auburn Street
Cambridge, Mass. 02138 Tel. (617)876-6776


# TRAC® PRIMITIVES*

OUTPUT.
- PS,string
  - PRINT STRING: prints out the second argument.

INPUT.
- RS
  - READ STRING: this command is replaced by a string of characters typed in by the user, whose end is signalled by a changeable "meta" character.
- CM,arg2
  - CHANGE META: first character of second argument becomes new meta character. May be carriage-return code.
- RC
  - READ CHARACTER: this command is replaced by the next character the user types in. Permits highly responsive interactive systems.

DISK COMMANDS.
- SB,blockname,form1,form2 . . .
  - STORE BLOCK: under block name supplied, stores forms listed.
- FB,blockname
  - FETCH BLOCK: contents of named block are quietly brought in to forms storage from disk.

MAIN FORM COMMANDS.
- DS,formname,contents
  - DEFINE STRING. Discussed in text.
- CL,formname,plug1,plug2,plug3 . . .
  - CALL: brings form from forms storage to working program. Plug1 is fitted into every hole (segment gap) numbered 1, plug2 into every hole numbered 2, and so on.
- SS,formname,punchout1,punchout2 . . .
  - SEGMENT STRING: this command replaces every occurrence of punchout1 with a hole (segment gap) numbered 1, and so on.

INTERNAL FORM COMMANDS.
(All of these use a little pointer, or form pointer, that marks a place in the form. If there is no form remaining after the pointer, these instructions act on their last argument, which is otherwise ignored.)
- IN,formname,string,default
  - Looks for specified string IN the form, starting at pointer. If not found, pointer unmoved. (NOTE: string search can also be done nicely with the SS command.)
- CC,formname,default
  - CALL CHARACTER: brings up next character in form, moves pointer to after it.
- CN,formname,no. of characters,default
  - CALL N: brings up next N characters, moves pointer to after them.
- CS,formname,default
  - CALL SEGMENT: brings up everything to next segment gap, moves pointer to it.
- CR,formname
  - CALL RESTORE: moves pointer back to beginning of form.

MANAGING FORMS STORAGE
- LN,divider
  - LIST NAMES: replaced by all form names in forms storage, with any divider between them. Divider is optional.
- DD,name1,name2 . . .
  - DELETE DEFINITION: destroys named forms in forms storage.
- DA
  - DELETE ALL: gets rid of all forms in forms storage.

TEST COMMANDS.
- EQ,firstthing,secondthing,ifso,ifnot
  - Tests if EQual: if firstthing is same as secondthing, what's left is ifso; if not equal, what's left is ifnot.
- GR,firstthing,secondthing,ifso,ifnot
  - Tests whether firstthing is numerically GReater than secondthing. If so, what's left is ifso; if not, what's left is ifnot.

OH YEAH, ARITHMETIC.
(All these are handled in decimal arithmetic, a character at a time, and defined only for two integers. Everything else you write yourself as a shorty program.)
- AD, SU, ML, DI } mentioned in text.

BOOLEAN COMMANDS.
(Several exist in the language, but could not possibly be understood from this writeup.)

---

* Description of TRAC language primitives adapted by permission from "TRAC, A Procedure-Describing Language for the Reactive Typewriter," copyright © 1966 by Rockford Research, Inc.


BIBLIOGRAPHY

Calvin N. Mooers, The Beginner's Manual for TRAC® Language, 300 pages, $10.00, from Rockford Research, Inc. (See "Where to Get It.")

Calvin N. Mooers, Definition and Standard for TRAC® T-64 Language, 86 pages, $5.00, from Rockford Research, Inc.

Calvin N. Mooers, "TRAC, A Procedure-Describing Language for the Reactive Typewriter," Communications of the ACM, v. 9, n. 3, pp. 215-219 (March 1966). Historic paper, out of print. This paper is copyrighted, and the copyright is owned by Rockford Research, Inc., through legal assignment from the Association for Computing Machinery, Inc.

And for those who want to understand the depth of the standardization problem, Mooers offers freebie reprints of:

Calvin N. Mooers, "Accommodating Standards and Identification of Programming Languages," Communications of the ACM, v. 11, n. 8, pp. 574-576 (August 1968).

# STARK & CLEVER APL

Some people call it a "scientific" language. Some people call it a "mathematical" language. Some people are most struck by its use for interactive systems, so to them it's an interactive language. But most of us just think of it as THE LANGUAGE WITH ALL THE FUNNY SYMBOLS, and here they are:

[illegible]

Enthusiasts see it as a language of inconceivable power with extraordinary uses. Cynics remark that it has all kinds of extraordinary powers for inconceivable uses-- that is, a weird elegance, much of which has no use at all, and some of which gets in the way.

This is probably wrong. APL is a terrific and beautiful triumph of the mind, and a very useful programming language. It is not for everybody, but neither is chess. It is for bright children, mathematicians, and companies who want to build interactive systems but feel they should stick with IBM.

APL is one of IBM's better products, probably because it is principally the creation of one man, Kenneth Iverson. It is mainly run on 360 and 370 computers, though implementations exist for the DEC PDP-10 and perhaps other popular machines. (Actually Iverson designed the language at Harvard and programmed it on his own initiative after moving to IBM; added to the product line by popular demand, it was not a planned product and might in fact be a hazard to the firm, should it catch on big.)

APL is a language of arrays, with a fascinating notation. The array system and the notation can be explained separately, and so they will.

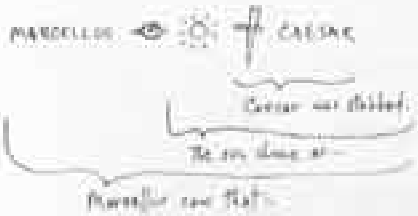
Let's just say the language works on things modified successively by operators. Their order and result is based upon those fiendish chicken scratches, Iverson notation.

THAT NIFTY NOTATION

The first thing to understand about APL is the fiendishly clever system of notation that Iverson has worked out. This system (sometimes called Iverson notation) allows extremely complex relations and computer-type events to be expressed simply, densely and consistently.

(Of course, you can't even type it without an IBM Selectric typewriter and an APL ball. Note the product-line tie-in.)

The notation is based on operators modifying things. Let's use alphabetic symbols for things and play with pictures for a minute.




In considering the successive meanings of this rebus we are proceeding from right to left, as you note, and each new symbol adds meaning. This is the general idea.

You will note, in this example, the curious arrangement whereby you can have several pictures, or operators, in a row. This is one of the fun features of the language.

TWO-SIDED OPERATORS

In old-fashioned notations, such as ordinary arithmetic, we are used to the idea of an operator between two things. Like

2 + 2

or in algebra,

x × y

These, too, occur in APL; indeed, APL can also nest two-sided operators-- that is, put them one inside the other, like the leaves of a cabbage. Old-fashioned notations nest with parentheses. But APL nests leftward. It works according to a very simple right-to-left rule.

x × y × 2 + 2

the result of this

is operated on by
the next thing and operator,
yielding another result

which is in turn operated on by
the next thing and operator,
yielding final result.

ONE-SIDED OPERATORS

We are also used to some one-sided operators in our previous life. For instance:

− 1

means the negation of 1;

− ( − 1)

means negating that.

APL can also nest one-sided operators.

first operator is applied to A.

result is worked on by second operator.

result is worked on by third operator.

result is worked on by fourth operator, yielding final result.

SAME SYMBOLS WORK BOTH WAYS

Now, one of the fascinating kickers of APL is the fact that most of the symbols have both a one-sided meaning and a two-sided meaning; but, thank goodness, they can be easily kept straight.

Here is a concrete example: the symbol ⌈ or "ceiling." Used one-sided, the result of operator ⌈ applied to something numerical is the integer just above the number it is applied to: ⌈7.2 is 8. Used two-sided, the result is whichever of the numbers it's between is larger: 10⌈8 is 10. (There is also ⌊, floor, which you can surely figure out.)

Now, when you string things out into a long APL expression, Iverson's notation determines exactly when an operator is one-sided and when it is two-sided:

As you go from right to left,

another thing? } ← OP THING
another op? }

you generally start with a thing on the right. Then comes an operator. If the next symbol is another thing, then the operator is to be treated as a two-sided operator (because it's between two things). If the object beyond the first operator is another operator, however, that means APL is supposed to stop and carry out the first operator on a one-sided basis. Example:

A − B

thing.
op.
thing.

Conclusion:
It's two-sided.
Interpretation:
"subtract B from A."

A + − B

thing.
op.
op--
stop.

Conclusion:
The first operator is one-sided.
Interpretation:
"negate B."
Then take next symbol.

---

# A WEIRD EXAMPLE, TO HELP WITH THE NOTATION.

Just for kicks, let us make up a notation having nothing to do with computers, using these Iverson principles:

1) If an operator or symbol is between two names of things, carry it out two-sidedly. If not, carry it out one-sidedly.

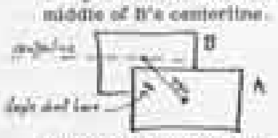
2) Go from right to left.

The best simple example I can think of involves file cards on the table (named A, B, C...) and operators looking like this:

0) 45) 90) 180) 45] 90] 180]

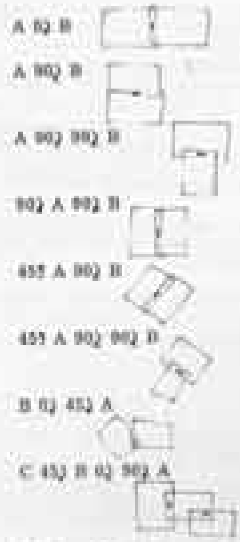
to which we may assign the following meanings:

| | |
|---|---|
| ONE-SIDED: | ROTATION OPERATORS |
| 0) A | do nothing to A |
| 45) A | rotate A clockwise 45° |
| 90) A | rotate A clockwise 90° |
| | etc. |
| TWO-SIDED: | STAPLING OPERATORS |
| B 45) A | staple A (thing named on the right) to B (thing named on the left) at a position 45° clockwise from middle of B's centerline. |




And equivalently for other angles.

Now, using these rules, and letting our things be any file cards that are handy, here are some results:




WOOPS! One of these examples is wrong! Can you find it??

It's hard to believe, but there you are. This notation seems adequate to make a whole lot of different stapled patterns.

Exercise! Use this nutty file card notation to program the making of funny patterns. Practice with a friend and see if you can communicate patterns through these programs, one person uncomprehendingly carrying out the other's program and being surprised.

The point of all this has been to show the powerful but somewhat startling way that brief scribbles in notations of this type can have all sorts of results.

Here is another example showing how we chug along the row of symbols and take it apart. Again, the alphabetical entities represent things.


Try dividing up these examples:


One more thing needs to be noted. Not only can we work out the sequences of operations, from right to left, between the symbols: the computer can carry them out in a stable fashion. Which is of course essential.

INSIDE

The truth of the matter is that APL in the computer is a continuing succession of things being operated on and replaced in the work area.


and so on.

What is effectively happening is that the APL processor is holding what it's working on in a holding area. The way it carries out the scan of the APL language, there only has to be one thing in there at a time.

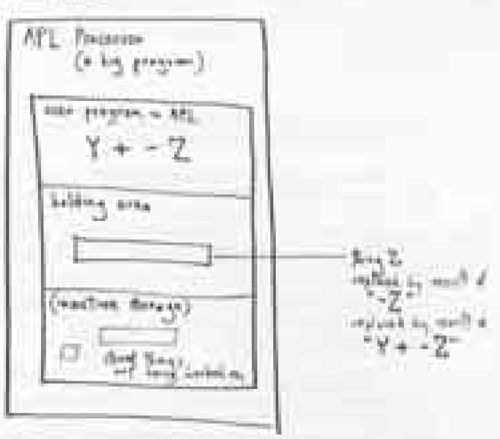
Suppose we have a simple user program,

Y + - Z

Starting at the right of this user program, the main APL program puts Z into the work area. That's the first thing. Then, stepping left in the user program, the APL processor follows the rules and discovers that the next operation makes it

- Z

which happens to mean, "the negation of Z." So it carries this out on Z and replaces Z with the result, -Z. Then, continuing to scan leftward, the APL processor continues to replace what was in the work area with the result of each operation in the successive lines of the user program, till the program is completed.




SOME APL OPERATORS

It would be insane to enumerate them all, but here is a sampling of APL's operators. They're all on the pocket cards (see Bibliography).

For old times' sake, here are our friends: (And a cousin thrown in for symmetry.)

| | |
|---|---|
| +A | plain A (whatever A should happen to be) |
| A+B | A plus B (whatever A should happen to B, heh heh) |
| -B | negation of B |
| A-B | A minus B |
| ×B | the sign of B (expressed as -1,0 or 1) |
| A×B | A times B |

And here are some groovies:

| | |
|---|---|
| !A | factorial A (1×2×3 ... up to A) |
| A!B | the number of possible combinations you can get from B, taken A at a time |
| ?A | a random integer taken from array A |
| A?B | take some integers at random from B. How many? A. |

But, of course, APL goes on and on. There are dozens more (including symbols made of more than one weird APL symbol, printed on top of each other to make a new symbol).

Consider the incredible power. Single APL symbols give you logarithms, trigonometric functions, matrix functions, number system conversions, logs to any arbitrary base, and powers of e (a mysterious number of which engineers are fond).

Other weird things. You can apply an operation to all the elements of an array using the / operator: +/A is the sum of everything in A. ×/A is the combined product of everything in A. And so on. Whew.

As you may suspect, APL programs can be incredibly concise. (This is a frequently-heard criticism: that the conciseness makes them hard to understand and hard to change.)

MAKE YOUR OWN

Finally and gloriously, the user may define his own functions, either one-sided or two-sided, with alphabetical names. For instance, you can create your own one-sided operator ZONK, as in

ZONK B

and even a two-sided ZONK,

A ZONK B

which can then go right in there with the big boys:

A ⌽ ZONK ⍀ ⍋ B

Don't ask what it means, but it's allowed.

APL THINGS, TO GO WITH YOUR OPERATORS

As we said, APL has operators (already explained) and things. The things can be plain numbers, or Arrays (already mentioned under BASIC). Think of them as rows, boxes and superboxes of numbers:

| | |
|---|---|
| 2 4 6 8 10 | a one-dimensional thing |
| 2 4<br>3 5 | a two-dimensional thing |
| 3 4<br>6 8 | a three-dimensional thing, seen from the front. Maybe we better look at the levels side by side:<br>1 2   3 4<br>5 7   6 8 |

APL can have Things with four dimensions, five and so on, but we won't trouble you here with pictures.

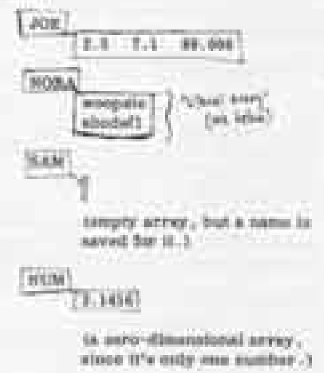
Oh yes, and finally a no-dimensional thing. Example:

73.2

It is called no-dimensional because there is only one of it, so it is not a row or a box.

Seriously, these are arrays, and Iverson's APL works them over, turns them inside out, twists and zaps through to whatever the answers are.

As in BASIC and TRAC, the arrays of APL are really stored in the computer's core memory, associated with the name you give them. The arrays may be of all different sizes and dimensionality:




Each array is really a series of memory locations with its label and boxing information-- dimensions and lengths-- stored separately. One very nice thing about APL is that arrays can keep changing their sizes freely, and this need be of no concern to the APL programmer. (The arrays can also be boxed and reboxed in different dimensions just by changing the boxing information-- with an operator called "ravel.")

# STOP THE PRESSES!

An APL machine, a mini that does nothing but APL, is now available from a Canadian firm for the mere pittance of

THREE THOUSAND FIVE HUNDRED DOLLARS,

the price of many a mere terminal. This according to Computerworld, 10 Oct 73.

Run, don't walk, to Micro Computer Machines, Inc., 4 Lansing Sq., Willowdale, M2J 1T1, Ontario, Canada. That $3500 gets you a 16K memory, the APL program, keyboard and numerical keyboard, and plasma display. Cassette (which apparently stores and retrieves arrays by name when called by the program) is $1500 extra. RUNS ON BATTERIES. Sorry, no green stamps. (Note that the APL processor takes up most of the 16K, but you can get more.)

* * * * * * * * * * *

The rumor that IBM has APL on a chip, inside a Selectric -- which therefore does all these things with no external connection to any (external) computer-- remains unsubstantiated. The rumor has been around for some time.

But it's quite possible.

The thing is, it would probably destroy IBM's entire product line-- and pricing edifice.

Few people know all of APL, or would want to. The operations are diverse and obscure, and many of them are comprehensible only to people in mathematical fields. However, if you know a dozen or so you can really get off the ground.

---

As in BASIC, you can use subscripts to get at specific elements in arrays. Referring to the examples above, if you type

JOE [3]

you get back on your typewriter its value

7.1

and if you type

NORA [2;4]

you get back

4

There are basically four kinds of information used by APL, and all of them can be put in arrays. Three of these types are numerical, and arrays of them look like this on paper:

Integer arrays: 2 4 -6 8 19 2048

Scalar arrays: 2.5 -1.1416 0.001 2795331.1
(A scalar is something that can be measured on a ruler-like scale, where there are always points in between.)

Logical arrays: 1 0 0 0 1 0 1
(these arrays of ones and zeroes are called "logical" for a variety of reasons; in this case we could call them "logical" simply because they are used for picking and choosing and deciding.)

These three numerical types of information may be freely intermixed in your arrays. One more type, however, is allowed. It's hard to figure out from the manuals, but evidently this type can't be mixed in with the others too freely. We refer to the alphabetical or "literal" array, as in

The quick brown fox jumped over the lazy dog

Now, pre-written APL programs can print out literal information, and accept it from a user at a terminal, which is why APL is good for the creation of systems for naive users (see "Good-Guy Systems," p. 5).

Literal vectors may be picked apart, rearranged and assembled by all the regular APL operators. That's how we twiddle our text.

CRASHING THE SYMBOLS TOGETHER

Now that we know about the operators and the arrays, what does APL do?

It works on arrays, singly and in pairs, according to these funny-looking symbols, as the APL processor scans right-to-left.

IVERSON'S TAFFY-PULL

A number of basic APL operators help you stretch, squish and pull apart your arrays. Consider the lowly comma (called "ravel," which means the same as "unravel").

,A — forget A's old dimensions, make it one-dimensional.
A,B — make A and B one long one-dimensional array.

Here is how we make things appear and disappear. ("Compression.")

A/B — A must be a one-dimensional array of ones and zeroes. The result is those elements of B selected by the ones. Example:
1 0 1 / 4 4 4
results in
4 4

The opposite slash has the opposite effect, inserting extra null elements where there are zeroes:

1 1 0 1 \ 3 5 9
results in
3 5 0 9

Here's another selector. This operator takes the first or last few of A, depending on size and sign of B:

B ↑ A

and B ↓ A is the opposite.

If you want to know the relative positions of numbers of different sizes in a one-dimensional array,

⍋ (name of array)

will tell you. It gives you the positions, in order of size, of the numbers. And ⍒ does it for descending order.

These are just samples. The list goes on and on.

SAMPLE PROGRAMS

Here is an APL program that types out backwards what you type in. First look at the program; then the explanation below.

```
    ∇ REV
[1]   I ← ⍞
[2]   ⍞ ← ⌽ I
    ∇
```

Explanation. The down-pointing triangles ("dels") symbolize the beginning and end of a program, which in this case we have called REV. On Line 1, the "Quote-Quad" symbol (on the right) causes the APL processor to wait for alphabetical input. Presumably the user will type something. The user's line of input is stuffed into thing or array I. The user's carriage return tells the APL processor he has finished, so it continues in the program. On the second line, APL takes array I and does a one-sided ⌽ to it, which happens to mean turning it around. Left-arrow into the quote-quad symbol means print it out.

Because of APL's compactness, indeed, this magnificent program can all go on one line:

```
    ∇ REV
[1] ⍞ ← ⌽ I ← ⍞
    ∇
```

First the input goes into I, then the processor does a ⌽ I (reversal) and puts it out.

And here is our old friend, the fortune-cookie prisoner.

```
    ∇ INF
[1]   ⍞ ← 'HELP, I AM CAUGHT IN A LOOP'
[2]   → 1
    ∇
```

On line 1 the program prints out whatever's in quotes. And line 2 causes it to go back and do line 1 again. Forever.

## THE TEST-AND-BRANCH IN APL

It should be mentioned at this point that branching tests are conducted in APL programs by specifying conditions which are either true or false, and APL's answer is 1 if true, 0 if false. (This is another thing those logical arrays are for.)

Example:


This operation leaves the number 1, because 3 is greater than 2. So you could branch on a test with something like

→ 7 × A > B

which branches to line 7 in the program if A is greater than B, and is ignored (as an unexecutable branch to line zero) if B is greater than A.

Some love it, some hate it.

THE APL ENVIRONMENT

Aside from the APL language itself, to program in APL you must learn a lot of "system" commands, alphabetical commands by which to tell the APL processor what you want to do in general -- what to store, what to bring forth from storage, and so on.

Ordinarily you have a workspace, a collection of programs and data which you may summon by name. When it comes -- that is, when the computer has fetched this material and announced to your terminal that it is ready -- you can run the programs and use the data in your workspace. You can also have passwords for your different workspaces, so others at other terminals cannot tamper with your stuff.

This is not the place to go into the system commands. If you're serious, you can learn them from the book or the APL salesman.

There are many, many different error messages that the APL processor can send you, depending on the circumstances. It is possible to make many, many mistakes in APL, and there are error messages for all of them. All of them, that is, that look to the computer like errors. If you do something permissible that's not what you intended, the computer will not tell you.

But it is a terminal language, designed to help people muddle through.

Good luck!

## IVERSON'S STRANGE AND WONDERFUL CHOICES OF SYMBOLS

Iverson's notation is built around the curious principle of having the same symbols mean two things depending on context. (Goodness knows he uses enough different symbols; doubling up at least means he doesn't need any more.) It turns out that this notation represents a consistent series of operations in astounding combinations.

The overall APL language, really, is the carrying through of this notation to create an immensely powerful programming language. The impetus obviously came from the desire to make various intricate mathematical operations easy to command. The result, however, is a programming language with great power for simpler tasks as well.

Now, the consequences of this overall idea were not determined by God. They were worked out by Iverson, very thoughtfully, so as to come out symmetrical-looking and easy to remember. What we see is the clever exploitation of apparent but inexact symmetries in the ideas. Often APL's one-sided and two-sided pairs of operations are more suggestively similar than really the same thing.

When Iverson assigns one-sided and two-sided meanings to a symbol, often the two meanings may look natural only because Iverson is such an artist. Example:

| two-sided | one-sided |
|---|---|
| A × B | × B |
| A times B | the sign of B |

This makes sense. To argue that it is inherent in "taking away half the idea of multiplication," however, is dubious.

Some symmetries Iverson has managed to come up with are truly remarkable. The arrow, for instance. The left arrow:

A ← B
Assignment statement: B (which may have been computed during the leftward scan) is assigned the name of A;

and the right arrow:

→ B
The jump statement, where B (which may have been computed during the leftward scan) is a statement number; the program now goes and executes that line.

This symmetry is mystically interesting because the assignment and jump statements are so basic to programming.

Or consider this:

⎕ ← X
print X.

X ← ⎕
take input from the user and stuff it into X.

Another weird example: supposedly the conditional branch

→ B/A

(one way of writing, "jump to A if B is true") is a special case of the "compression" operator. (Berry 360 primer, 73 and 161.) This is very hard to understand, although it seems clear while you're reading it.

On the other hand, there is every indication that APL is so deep you keep finding new truths in it. (Like the above paragraph.) The whole thing is just unbelievable. Hooray for all that.

---

APL FOR USER-LEVEL SYSTEMS
(See "Good-Guy Systems," p. 5.)

Because APL can solicit text input from a user and analyze it, the language is powerful for the creation of user-level environments and systems -- with the drawback, universal to all IBM terminals, that input lines must end with specific characters. In other words, it can't be as fully interactive as computer languages that use ASCII terminals.

Needless to say, the mathematical elegance and power of the system is completely unnecessary for most user-level systems. But it's nice to know it's there.

APL is probably best for systems with well-defined and segregated files -- "array-type problems," like payroll, accounts and so on. It is not suited for much larger amorphous and evolutionary stuff, the way list languages like TRAC are. Don't use APL if you're going to store large evolving texts or huge brokerage data bases, like what tankers are free in the Mediterranean.

The quickest payoff may lie in using APL to replace business forms and hasten the flow of information through a company. A salesman on the road with an APL terminal, for instance, can at once enter his orders in the computer from the customer's office, checking inventory directly. If the program is up

ROUND (an obscure and demolish joke)

ρ, the Greek letter "rho," is an APL operator for testing the size of arrays. When used in the one-sided format, it gives the sizes of each dimension of an array.
Thus
ρA, when A is [1 2 / 3 4]
is 2 2.
And now
ρ 'YOUR BOAT'
equals 9, since there are 9 letters in the array 'YOUR BOAT';
ρρ 'YOUR BOAT'
is 1,
since ρ 9 is 1, and
ρρρ 'YOUR BOAT'
is likewise 1.

This language is superb for "scientific" programming, including heavy number crunching and experimentation with different formulas on small data bases. (Big data bases are a problem.)
It is also not bad for a variety of simple business applications, such as payroll, accounting, billing and inventory.

FAST ANSWERBACK IN APL

If you want quick answers, the APL terminal just gives you the result of whatever you type in. For instance,

3 × 4

will cause it to print out

12

and the same goes for far less comprehensible stuff like

7 2 ⍋ ⍒ 7 1 2 3 4 (carriage return)
typed-in array

PROGRAMS IN APL

But the larger function of APL is to create programs that can be stored, named and carried out at a later time.

For this, APL allows you to define programs, a line at a time. The programs remain stored in the system as long as you want. Using the "Del" operator (∇), you tell the system that you want to put in a program. Del causes the terminal to help you along in various ways.

A nice feature is that you can lock your APL programs, that is, make them inaccessible and unreadable by others, whether they are programmers or not. In this case you define a program starting with the mystical sign del-tilde (⍫) instead of del (∇), and invoke the names of dark spirits.

APL, like BASIC, can be classed as an "algebraic" language-- but this one is built to please real mathematicians, with high-level stuff only they know about, like Inner and Outer Products.
Paradoxically, this makes APL terrific for teaching these deeper mathematical concepts, helping you see the consequences of operations and the underlying structure of mathematical things. Matrix algebra, for instance, can be visualized a lot better by working up to it with lesser concepts (like vectors and inner products) enacted on an APL terminal.
It would be really swell if someone would put together a tour-guide book of higher mathematics at the grade/highschool level for people with access to APL.
Interestingly, Alfred Bork (U. of Cal. at Irvine) is taking a similar approach to teaching physics, using APL as a fundamental language in his physics courses.

SNEAKY REPEATER STATEMENT IN APL?

One of the APL operators, "iota" (ι), seems to make its own program loop within a line. When used one-sided, it furnishes a series of ascending numbers up to the number it's operating on. This until the last one is reached.

You type: 3 × ι 7
APL replies: 3 6 9 12 15 18 21

In other words, one-sided iota looks to be doing its own little loop, increasing its starting number by 1, until it gets to the value on its right, and chugs on down the line with each.

Very sneaky way of doing a loop.

However! It isn't really looping, exactly. What the iota does is create a one-dimensional array, a row of integers from 1 up to the number on its right. This result is what then moves on leftward.

WHERE TO GET IT

IBM doesn't sell APL services. Their time-sharing APL is available, however, from various suppliers. Of course, that means you probably have to have an IBM-type terminal, unless you find a service that offers APL to the other kind-- an addition which seems to be becoming fashionable.

Usual charge is about ten bucks an hour connect charge, plus processing, which depends on what you're doing. It can easily run over $15 an hour, though, and more for heavy crunching or printout, so watch it.

The salesman will come to your house or office, verify that your terminal will work (or tell you where you can rent one), patiently show you how to sign on, teach you the language for maybe an hour if he's a nice guy, and proffer the contract.

→APL services are probably safer to sign onto, in terms of risked expenses, than most other time-sharing systems. (Though of course all time-sharing involves financial risk.) Because the system is restricted only and exactly to APL, you're not paying for capabilities you won't be using, or for massive disk storage (which you're not allowed in most APL services anyway), or for acres of core memory you might be tempted to fill.

→ In other words, APL is a comparatively straight proposition, and highly recommended if you have a lot of math or statistics you'd like to do on a fairly small number of cases. Also good for a variety of other things, though, including fun.

Different vendors offer interesting variations on IBM's basic APL\360 package, as noted below. In other words, they compete with each other in part by adding features to the basic APL\360 program, vying for your business. Each of the vendors listed also offers various programs in APL you can use interactively at an IBM-type terminal, in many cases using an ordinary typeball and not seeing the funny characters; though how clear and easy these programs are will vary.

And remember, of course, that you can do your own thing, or have others do it for you, using APL.

APL is also available on the PDP-10, and presumably other non-IBM big machines.

THE VENDORS

Scientific Time-Sharing Corporation (7316 Wisconsin Ave., Bethesda MD 20014) calls its version APL*PLUS. They'll send you a nice pocket card summarizing the commands.

APL*PLUS offers over twentyfive concentrators around the country, permitting local-call services in such metropolitan centers as Kalamazoo and Rochester. (Firms with offices in both cities, please note.)

They also have an "AUTOSTART" feature which permits the chaining of programs into grand complexes, so you don't have to call them all individually.

APL*PLUS charges the following for storage, if you can dig it: $18 PER BILLION BYTE-DAYS. (A byte is usually one character.) The census is probably taken once a day.

This firm also services ASCII terminals, which some people will consider to be a big help. That means you can have interactive users of APL programs at ASCII terminals, and that you can also program from the few APL terminals that aren't of the IBM type.

Time Sharing Resources, Inc. (777 Northern Blvd., Great Neck, N.Y. 11021) offers a lot of APL service, including text systems and various kinds of file handling, under the name TOTAL/APL.

Among the interesting features Time Sharing Resources, Inc. have added is an EXECUTE command, which allows an APL string entered at the keyboard in user on-line mode to be executed as straight APL. This is heavy.

Perhaps the most versatile-sounding APL service right now is offered by, of all people, a subsidiary of the American Can Company. American Information Services (American Lane, Greenwich CT 06830) calls their version VIRTUAL APL, meaning that it can run in "virtual memory"-- a popular misnomer for virtually unlimited memory-- and consequently the programmer is hardly subject to space limitations at all. Moreover, files on the AIS system are compatible with other IBM languages, so you can use APL to try things out quickly and then convert to Fortran, Cobol or whatever. (Or, conversely, a company may go from those other languages to APL without changing the way their files are stored on this service.) APL may indeed interface with these other languages; how is unclear.

And the prices look especially good: $8.75 an hour connect, $15 a month minimum (actually their minimum disk space rental-- 1 IBM cylinder-- so for that amount you get a lot of storage). But remember there are still core charges, and $1 per thousand characters printed or transferred to storage.

In the West, a big vendor is Proprietary Computer Systems, Inc., Van Nuys, California.

TERMINALS

For an APL terminal, you might just want a 2741 from IBM (about a hundred a month, but on a year contract).

Or see the list under "Terminals" (p. [illegible]), or ask your friendly APL company when you sign up.

Two more APL terminals, mentioned here instead of under "Terminals" for no special reason:

Tektronix offers one of its groovie graphics terminals (see flip side) for APL (the model 4013). This permits APL to draw pictures for you. It seems to be an ASCII-type unit.

Computer Devices, Inc. supposedly makes an an APL terminal using the nice NCR thermal printer, which is much faster and quieter than a mechanical typewriter. Spookier, though. And the special paper costs a lot of money.

BIBLIOGRAPHY


Iverson has a formal book. Ignore it unless you're a mathematician: Kenneth E. Iverson, A Programming Language. Wiley, 1962.
Paul Berry, APL\360 Primer, Student Text. Available "through IBM branch offices," or IBM Technical Publications Department, 112 East Post Road, White Plains, NY 10601. No IBM publication number on it, which is sort of odd. 1969.
→This is one of the most beautifully written, simple, clear computer manuals that is to be found. Such a statement may astound readers who have seen other IBM manuals, but it's true.
A.D. Falkoff and K.E. Iverson, APL\360 Users' Manual. Also available from IBM, no publication number.
POCKET CARDS (giving very compressed summaries) are available from both:
Scientific Time Sharing Corp. (see WHERE TO GET IT)
Technical Publications Dept., IBM, 112 East Post Road, White Plains, N.Y. 10601.
Ask for APL Reference Data card S210-0007-0. May cost a quarter or something.

Paul Berry, APL\1130 Primer. Adapted from 360 manual. Same pub. But for version of APL that runs on the IBM 1130 minicomputer.
Roy A. Sykes, "The Use and Misuse of APL." $2 from Scientific Time-Sharing Corp., 7316 Wisconsin Ave., Bethesda MD 20014.
A joker for you math freaks: Trenchard More, Jr., "Axioms and Theorems for a Theory of Arrays." IBM Journal of Resch. & Devt., March 73, 135-157. This is a high-level thing, a sort of massive set theory of APL intended to make APL operators apply to arrays of arrays, and lead ultimately to the provability of programs.
"Get on Target with APL." A suggestive circular sales thingy. IBM G320-1439-0.
IBM has a videotaped course in APL by A.J. Rose. (Since 1968.)


⇨ What you really need to get started is Berry's Primer, Falkoff and Iverson's manual, and a pocket card. Plus of course the system and the friend to tutor you.

Power and simplicity do not often go together.
APL is an extremely powerful language for mathematics, physics, statistics, simulation and so on.
However, it is not exactly simple. It's not easy to debug. Indeed, APL programs are hard to understand because of their density.
And the APL language does not fit very well on minis.

APL is not just a programming language. It is also used by some people as a definition or description language, that is, a form of notation for stating how things work (laws of nature, algebraic systems, computers or whatever).

For instance, when IBM's 360 computer came out, Iverson and his friends did a very high-class article describing formally in APL just what 360s do (the machine's architecture). But of course this was even less comprehensible than the 360 programming manual.


Falkoff, A.D., K.E. Iverson and E.H. Sussenguth, "A Formal Description of System/360." IBM Systems Journal, v.3 no. 2, 1964.
The formal description in APL.

IBM System/360 Operating System: Assembler Language. Document Number C28-6514-X (where X is a number signifying the latest edition). IBM Technical Publications, White Plains New York.
The Manual.

# DATA STRUCTURE:
## INFORMATION SETUPS

One of the commonest and most destructive myths about computers is the idea that they "only deal with numbers." This is TOTALLY FALSE. Not only is it a ghastly misunderstanding, but it is often an intentional misrepresentation, and as such, not only is it a misrepresentation but it is a damned lie, and anyone who tells it is using "mathematics" as a wet noodle to beat the reader with.

Computers deal with symbols and patterns.

Computers deal with symbols of any kind-- letters, musical notes, Chinese ideograms, arrows, ice cream flavors, and of course numbers. (Numbers come also in various flavors, simple and baroque. See chocolate box, p. 29.)

Data structure means any symbols and patterns set up for use in a computer. It means what things are being taken into account by a computer program, and how these things are set up-- what symbols and arrangements are used to represent them.

The problem, obviously, is Representing The Information You Want Just The Way You Want It, in all its true complexities.

(This is often forbiddingly stated as "making a mathematical model"-- but that's usually in the rhetorical, far-fetched and astral sense in which all relations are "mathematical" and letters of the alphabet are considered to be a special distorted kind of number.)

Now it happens that there are many kinds of data structure, and they are interchangeable in intricate ways.

The same data, with all its relationships and intricacies, can be set up in a vast variety of arrangements and styles which are inside-out and upside-down versions of each other. The same thing (say, the serial number, 24945, of an automobile) may be represented in one data structure by a set of symbols (such as the decimal digits 2, 4, 9, 4, 5 in that order), and in another data structure by the position of something else (such as the 24945th name in a list of automobile owners registered with the manufacturer).

Furthermore, many different forms of data may be combined or twisted together in the same overall setup.

The data structure chosen goes a long way in imposing techniques and styles of operation on the program.

On the other hand, the computer language you use has a considerable effect upon the data structures you may choose. Languages tend to impose styles of handling information. The decision to program a given problem in a specific language, such as BASIC or COBOL or APL or TRAC Language, either locks you into specific types of data structure, or exerts considerable pressure to do it a certain way. In most cases you can't set it up just any way you want, but have to adjust to the language you are using-- although today's languages tend to allow more and more types of data.

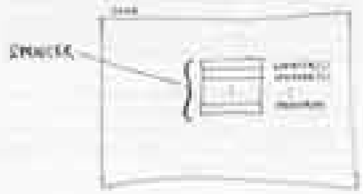
Plainly, then, it is these overall structures that we really care about; but to understand overall structures, we need an idea of all the different forms of data that may be put in them.

VARIABLES AND ARRAYS

The earliest data structures in computers, and still the predominating ones, are variables and arrays. (We met them earlier under BASIC, see pp. 4-17, and APL, see p. 22.)

A variable is a space or location in core memory. (For convenience, most programming languages allow the programmer to call a variable by a name, so that he doesn't have to keep track of its numerical address.)


An array (also called a table) is a section of core memory which the programmer cordons off for the program to put and manipulate data in. If SPENCER is the name of the array, then SPENCER(1) is the first memory slot in it, SPENCER(2) is the second, and so on up to however big it is.




(You can get a feel for how this ordinarily relates to input from outside-- see "How Data Comes, Goes, and Sits," nearby.)

The contents of a numerical field, or piece of data coming in, can simply be stuffed by the programmer into a variable.

The contents of a record, or unified set of fields, can get put into an array. The program can then pluck into it for separate variables, if desired, or just leave them there to be worked on.

Then you twiddle your variables with your program as desired.

When you've done one record, you repeat. That's how lots of business programs go. Some other routine kinds, too.

FANCY STRUCTURES

Many forms of advanced programming are based on the idea that things don't have to be stored next to each other, or in any particular order.

If things aren't next to each other, we need another way the program can tell how they belong together.

A pointer, then-- sometimes called a link-- is a piece of data that tells where another piece of data is, in some form of memory. Pointers often connect pieces of data.


A pointer can be an address in core memory; it can be an address on disk (diskpointer); it can point to a whole string of data, such as a name, when there is no way of knowing in advance how long the string may be (stringpointer).

A series of pieces of data which point to each other in a continuing sequence is called a threaded list.

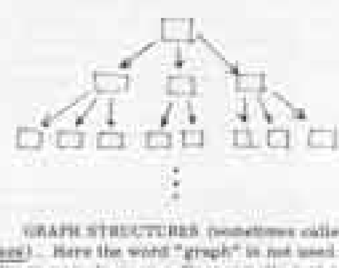

For this reason the handling of data held together by pointers-- even though it may make all sorts of different patterns-- is called list processing. (The term "list processing" might seem to go against common sense, as it might suggest something like, say, a laundry list, which is structured in a very simple blocklike form. But that's what we call it.)

Prominent list-processing languages include SNOBOL, L⁶ and LISP (see p. 31). There is argument as to whether TRAC Language is a list-processing language.

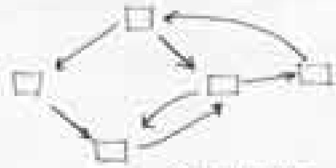
Here are some interesting structures that programmers create by list processing:

RINGS (or cycles). These are arrangements of pointers that go around in a circle to their first item again.


TREES. These are structures that fan out. (There are no rings in a tree structure, technically speaking.)



GRAPH STRUCTURES (sometimes called plexes). Here the word "graph" is not used in the ordinary way, to mean a diagrammatic sort of picture, but to mean any structure of connected points. Rings and trees are special cases of graph structures.



Graph structures can go any which way.

FAST-CHANGING DATA

One of the uses of such structures is in strange types of programs where the interconnections of information are changing quickly and unpredictably. Such operations happen fast in core memory. In this kind of programming (for which languages like LISP, SNOBOL and TRAC Language are especially convenient), the pointers are changed back and forth in core memory, every which way, all the time. Presumably according to the programmer's fiendish master plan-- if he's gotten the bugs out. (See Debugging, p. 70.)

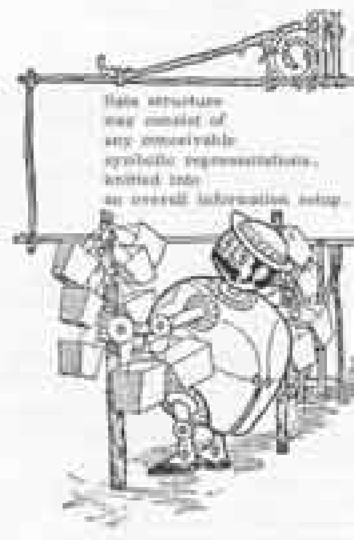
FANCY FILES

But these structures are not restricted to data in core memory. Complex and changeable files can be kept on disk in various ways by the same kind of threading (called "chaining" on mass storage).

CHAINED FILE ON DISK

Another way of handling changeable files is through a so-called directory block, which keeps track of where all the other blocks are stored.


But these techniques, you see, may be used in both fast and slow operations, and for any purpose, so trying to categorize them tends not to be helpful. (Note also that these techniques work whether you're dealing with bits, or characters, or any other form of data.)




Note: By decent standards of English, the word data should be plural, datum singular. But the matter is too far gone: data is now utterly singular, like "corn" and "information," a granular collective which may be scooped, poured or counted.

But I draw the line at media. Media are many, "media" is plural!

## A CLASSIC MISUNDERSTANDING

"Computers put everything into pigeonholes."

Wrong. People put things into pigeonholes. And designers of computer programs can set up lousy pigeonholes. If you let 'em. More sophisticated programming can often avoid pigeonholes entirely.

## A Bit Is Not A Piece

People who want to look with it occasionally use the term "bit" for any old chunk of information, like a name or address. This is Wrong. A Bit is the smallest piece of binary information, an item that can be one of two things, like heads or tails, X or O, one or zero, and all other information can be packed into a countable number of bits. (How many may depend on the data structure chosen.)

As a handy rule of thumb: every letter of the alphabet or punctuation mark is eight bits (see ASCII box); for heavy storage of everyday decimal numbers, every numerical digit can be further packed down (to four bits in BCD code).

A CONCRETE EXAMPLE. Suppose we want to represent the genealogy of the monarchs of England, as far as is known, in a computer data structure. NOTE THAT A DATA STRUCTURE IS DIFFERENT FROM A PROGRAM: if several programmers agree beforehand on a data structure, then they can go separate ways and each can write a program to do something different with it— if they have really agreed on a complete and exact layout, which they may only think they've done.

First we consider the subject matter. Genealogy is conceptually simple to us, but as data is not as trivial as it might seem at first. Every person has two parents and a specific date of birth. Each pair of parents can have more than one child, and individual parents can at different times share parenthood with different other individuals.

Presumably we would like a data structure that allows a program to find out who was a given person's parent, who were a given person's children, what brothers and sisters each person had, and similar matters (so far as is known by historians— another difficulty).

Note that just because it is simple to put this information in a wall chart, that does not mean it is simple to figure out an adequate data structure.

Note too, that any aspect of the data which is left out cannot then be handled by the program. What's not there is not there.

The easy way out is to use a language like, say, TRAC Language, and use its basic units (in this case, "forms") to make up a data structure whose individual sections would show parentage, dates, brothers and sisters and so on.

The braver approach is to try to set it up for something like FORTRAN or BASIC, languages which treat core memory more like a numerically-addressed array or block, as does rock-bottom machine language.

Let us assume that we have decided to use an array-type data structure, for instance to go with a program in the BASIC language on a 16-bit minicomputer. We do not have much room in core memory, so for each person in our data structure we are going to have to store a separate record on a disk memory, and call it into core memory as required.

After much head-scratching, we might come up with something like the following. It is not a very good data structure. It is not a very good data structure on purpose.

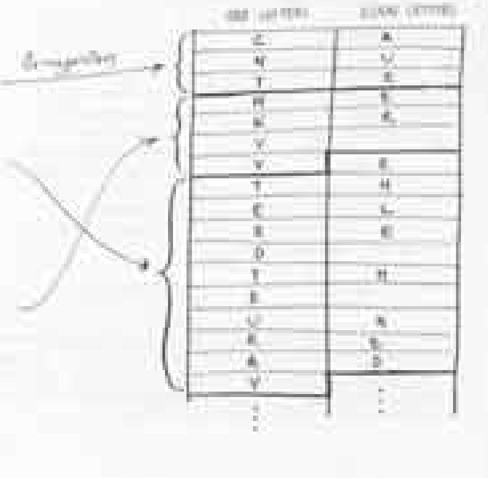
It uses a block of 28 words, or 448 bits, per individual, not counting the length of his name, which is an additional 8 bits per character or space. However, this in itself is not that good nor bad. It's more than you might expect, but less than you might need.

(Incidentally, out of concern for storage space, some data fields are packed more than one to a 16-bit computer word. This is scornfully called bit-fiddling by computerfolk who work on big machines and don't have to worry about such matters.)

| | | | | |
|---|---|---|---|---|
| | 1 | monarch no. (if any) | sex — (1 bit) | |
| (individual's own | 2 | serial no. | | |
| (name) | 3 | stringpointer | | → "name area" |
| | 4 | (two 16-bit words long) | | |
| mother | 5 | serial no. | | |
| father | 6 | serial no. | | |
| brothers | 7 | serial no. | | |
| (up to five) | 8 | : | | |
| | 9 | : | | |
| | 10 | | | |
| | 11 | | | |
| sisters | 12 | serial no. | | |
| (up to five) | 13 | : | | |
| | 14 | : | | |
| | 15 | | | |
| | 16 | | | |
| date of 1st reign, if any | 17 | start (11 bits) | no. months | |
| date of 2d reign, if any | 18 | start (11 bits) | no. months | |
| female children, | 19 | serial no. | | |
| up to five | 20 | : | | |
| | 21 | : | | |
| | 22 | | | |
| | 23 | | | |
| male children, | 24 | serial no. | | |
| up to five | 25 | : | | |
| | 26 | : | | |
| | 27 | | | |
| | 28 | | | |

As explained already, that was the basic block. We still have to keep the names somewhere, in a string area. Whether to keep this in core all the time, or on disk, is a decision we needn't go into here.

NAME AREA (packed 2 letters to a 16-bit word)




Here are some assumptions I have embodied in this data structure. That is, I had them in mind. (The parts you didn't have in mind are what get you later.)

Parents and children of monarchs are included, as well as monarchs.

All monarchs have a separate monarch number.

No monarch reigned more than twice. (?)

No monarch or parent of a monarch had more than five children of one sex. (Note the danger of these assumptions.)

We are not interested in grandchildren of monarchs unless they are also monarchs, or siblings, or parents of monarchs.

The information about the different people can be input in any order, as the years of reign can be stepped through by a program to find the order of reign.

If this seems like too much bother, that is in a way the point. Data structures must be thought out. Since computers have no intrinsic way of operating or of handling data (though particular languages will restrict you in particular ways), you will have to work all this out, and a carelessly chosen data structure will leave something out, or fail to distinguish among important differences, or otherwise have its revenge.

(For instance, if you haven't noticed yet, we left out legitimacy. For many purposes we want to know which kings were bastards.)

(Self-test: is five bits long enough to express the greatest number of months any English monarch reigned? — see "Binary Patterns." Or do we have to fix this data structure on that score also?)

To give you a sense of the sort of program this data structure allows:

A program to ascertain how many kings were the sons of kings would look at each entry that had a monarch number, test whether the monarch was male, and if male, would look at the male parent's serial number. Then it would look up that parent's entry, and see whether it in turn had a monarch number, and if so, add one to the count it was making. Then it would go back to the entry it had been looking at, and step on to the one after that.

This is actually a pretty lousy data structure. The clumsiness of this approach to such data— and you are welcome to think of a better one— shows some of the difficulties of handling complex data about the real world. Things like lengths of names and numbers of relatives produce great irregularities, but make these kinds of data no less worth of our attention.

We could add lots of things to our data structure (and so make it more unwieldy). For instance, we might want to mark each serial number specially if it referred to someone who was the offspring of a monarch. We could simply set a particular bit to 1 in the serial number for them (called a flag or tag). We could also flag dates and genealogies that are regarded as uncertain. There is no limit to the exactness and complexity with which information may be represented. But doing it right can, as always, be troublesome.

A lot of computer people want to avoid dealing with complex data; perhaps you can begin to see why. But we must deal with the true complexities of information; therefore languages and systems that allow complex information structures must become better known and easier to use.

THE FRONTIER: COMPLEX FILE STRUCTURE

The arrangements of whole files— groups of records or other info chunks— are up to the programmer. The structure of files is called, not surprisingly, file structure, and it is up to the programmer to decide how his files should be arranged.

Habits die hard. The notion of sequence— even false, imposed sequence— is deep in the racial unconscious of computer people. An interesting concrete term shows this clearly. Because computer people often think any file should have a basic sequence, they use the term inverted file for a file that has been changed from its basic sequence to another sequence. But increasingly, all the sequences are false and artificial. Where now are inverted files? All files are inverted if they're anything.

Fortunately, the final frontier of data structure is now increasingly recognized as the control of complex storage of files on disk memory. The latest fancy term for this is data base system, meaning planned-out overall storage that you can send your programs to like messengers.

The fact that IBM now has moved into this area (with its intricate "access methods" and all their initials) means complex storage control has finally arrived, although the pioneering work was done by Bachman at GE some years ago (see bibliography). Till the last few years, external storage, with pointers and everything, has not been conveniently under the programmer's control except in crude ways. Finally we are seeing systems beginning to get around that automatically handle complex file structures in versatile ways that programmers can use more easily.

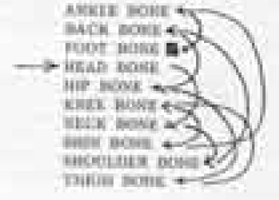
data
damyata
dhayadvam
— T.S. Eliot,
The Waste Land

"There is a growing feeling that data processing people would benefit if they were to accept a radically new point of view, one that would liberate the application programmer's thinking from the centralism of core storage and allow him the freedom to act as a navigator within a database. ... This reorientation will cause as much anguish among programmers as the heliocentric theory did among ancient astronomers and theologians."

Charles W. Bachman
(piece cited in Bibliography)

Remember the song that had a pointer data structure?

(in alphabetical order)




BIBLIOGRAPHY


Malcolm C. Harrison, Data-Structures and Programming. Scott, Foresman, 1973.

→This book can be recommended to ambitious beginners. It has useful summaries of different languages, as well as fundamental treatment of data structures as they intertwine with specific languages.

An obscure and intricate study of the interchangeability of data structures— how they fundamentally interconvert— has been the longtime research of one Anatol Holt, who calls his work Mem-Theory. Mem is from memory, and also, conveniently, a Hebrew letter.

This is an extremely ambitious study, as it in principle embraces not just much or all of computer science, but perhaps mathematics itself. Math freaks attention: Holt has said he intended to derive all of symbolic logic and mathematics from relations and pointer structures. Let's hear it for turning Russell on his head.

I don't know if Holt has published anything on it in the open literature or not.

However, he does have a game available which seems weirdly to embody these principles. The game of Mem is available for $8.35 postpaid ($8.86 to Pennsylvanians) from Mellanbar, Inc., 1700 Walnut St., Phila. PA 19103. It has beautifully colored pieces, looks deceptively simple, and is unlike anything, except discrete abstractive thinking itself. Recommended.

Charles W. Bachman, "The Programmer as Navigator." CACM Nov 1973.

Bachman was the prime mover in the development of large linked disk data systems at General Electric; he is the Pioneer. This is about big n-dimensional stuff.

David Lefkovitz, File Structures for On-Line Systems. Spartan-Hayden Books, 1969.

Alfonso F. Cardenas, "Evaluation of File Organization— a Model and System." CACM Sep 73, 540-548. Not surprisingly, it turns out that different file organizations have different advantages.

Edgar H. Sibley and Robert W. Taylor, "A Data Definition and Mapping Language." CACM Dec 73, 750-759.

Example of current sophisticated approaches: a whole language for calling the data just the way it should be. Has helpful further citations.

"ASCII and ye shall receive."
— the Industry

# SOMETIMES IT JUST SITS THERE SOMETIMES IT COMES AND GOES.

Data usually has to be marshalled into rows, or even regiments and battalions, before it can go into a computer.

(Some people just get their data into a computer by sitting at a terminal and typing it in, perhaps answering questions typed to them by a front-end program. But they're the lucky ones. Most of us have to get the data set up on some kind of holding surface before it gets fed in. That's an input medium.)

## DATA MEDIA

A data medium ("medium" is the singular of "media") is anything that holds the marks of data outside the core memory of a computer. Thus punched cards and punched paper tape may be used as input media, used for putting information into a computer. (Each medium needs a corresponding input or output device, to which across the surface and translate its marks or holes into the corresponding electronic pulses.)

There are three types of data media: input, output and storage media. An input medium carries the data in. An output medium receives the results of a program; for instance, a sheet of paper coming out of a printing device is an output medium, as is a punched card or punched paper tape.

Storage media are output media that may be used as input media later on. Thus punched cards and punched paper tape can be storage media. But the better storage media use magnetic recording (which is faster and less bulky), like magnetic tape and disks, or just plain "disks" as we generally call them. (See fuller list of mag media under "Peripherals," p. 37.)

The units and arrangements of data used for input, output and storage are in principle not necessarily the true ones of the data structure used by the program. The blocks and records of storage, for instance, may have irregular data with pointers sitting in them. (Unfortunately there is some carryover, in that programmers are tempted to use data structures which are easy to store and run in and out, rather than handling the true complexities of the subject. This is always a temptation.)

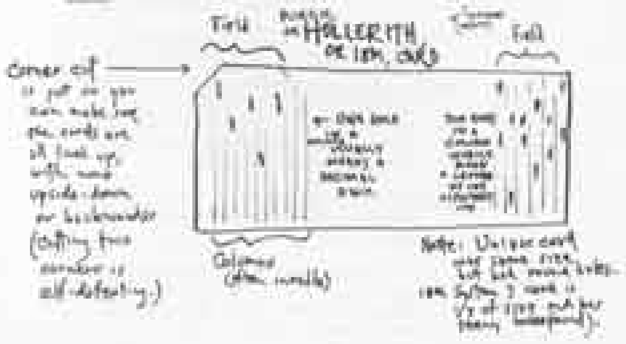
Let us consider the units and arrangements of data used for input and output and storage. These are, respectively, fields, records, files and blocks.

## THE PUNCH CARD

Let's begin with a fun example: that hoary old medium for input and output, the punched (or "punch") card. The punch card will show us what a field is.

The punch card is generally believed to have been invented by Herman Hollerith (although the author's in-laws had bitter recollections to the contrary). It was first used on a broad scale to count up the census of 1890, and later became an early cornerstone of IBM, but that's another story.

The punches on a card represent a row of information (such as a row of typed letters); this is not obvious because the card is a rectangle rather than a line. However, the length of the card is actually divided into eighty positions, each of which may hold one number, alphabetic character or punctuation mark. These positions are actually narrow columns, eighty of them, with different positions in which holes may be punched. One hole in a column represents a numeral; which position in the column specifies what number. Two holes in a column generally mean a letter of the alphabet, three holes in a column mean a punctuation mark.




Data is punched into cards according to some plan associated with the program.

Beyond these simple matters there is no preordained arrangement for information on a punch card; it all depends on what the program calls for. But each separate piece or section of information — each bunch of consecutive characters that together have a specific meaning — are called a field.

A field can be a name, a number, an amount of money, an alphabetical code representing something, a numerical code representing something, or other stuff. When the cards go into the program, the program can pick off the information it needs one field at a time — putting the field in columns 1 to 17 into one program variable, the field from columns nine to ten into another program variable, and so on.

The punch card is an important example of an input medium influencing the structure of computer programs. It is convenient to use fields on a punch card as the basic data structure of a program and say, "That's the way it has to be for the computer." In the worst cases we see the workings of the "punch card mentality" or "80-column mind" (see box).

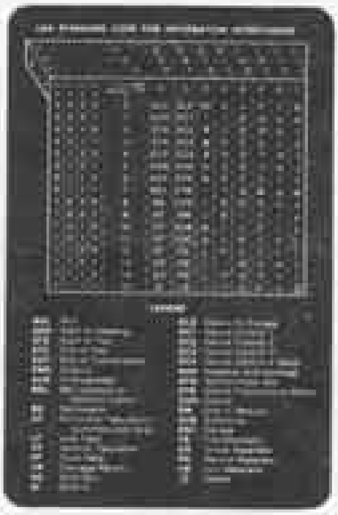
→ People will often thrust a punched data card at you and ask, "What does this mean?" Who knows? It may have lettering banged along the top, showing what characters the holes represent, but if these characters don't show anything understandable, such as the person's name, you're in the dark. The card may have preprinted section lines dividing it up, but these are rarely self-explanatory. It's often impossible just to look at a punched card and tell by eye what the individual fields are for, or even where they begin and end; all that depends on the program. Only someone who understands the program, or at least knows what fields the card is divided into and what the characters represent there, can help.

Sometimes, in dismal systems we encounter day-to-day — like for university registration — a punch card will have a person's name in the first few columns, or worse, a personal serial number. Other information continues from there. These may or may not be recognizable, either from reading the holes by eye, or from designations pre-printed on the card.

"ASCII not,
what your computer
will do for you."
— IBM

ASCII code. You can figure out from the table the bit pattern for any letter, or what any given combination of seven bits means.

Example. Find the capital letter G in the table. For the first three bits of the code, look at the top of the column: 100. For the next four, look sideways to the left: 0111. So G is: 1000111.



(An eighth bit is used as a check on the number of ones in the code; this is called the parity bit, and either rounds to an even number of bits (even parity) or an odd number of bits (odd parity). Thus if a code comes through to the computer with a wrong number of ones, the computer can take remedial action.)

These funny multiletter codes are for controlling terminals and like that.

Pocket card courtesy of Computer Transceiver Systems, Inc.

## MAGNETIC STORAGE

The same principle of fields applies to other data media, especially magnetic tape and disk. We may extend the notion of a field to explain records and files.

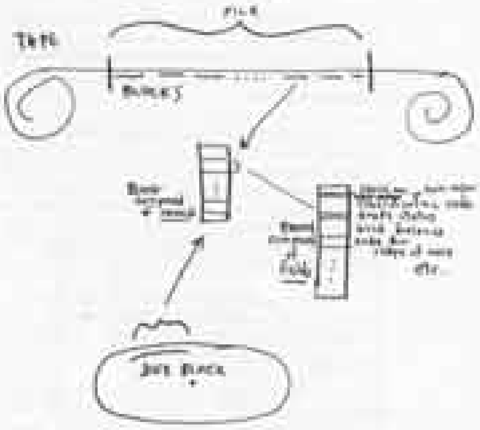
A field, generally speaking, is a section of positions on some medium reserved for one particular piece of information, or the data in it.

A record is a bunch of fields stored on some medium which have some organized use. (For instance, the accounting information held by an electric utility company about a particular customer is likely to be stored as a record with at least these fields: account number; last name, initials; address; amount currently owed.)

A file is a whole big complete bunch of information that is stored someplace. In many applications a file is composed of numerous similar, consecutive records. For instance, an electric company may well store the records for all of its customers on a magnetic tape, ordered by account number (account number field).

Storing sequences of similar records in long files is typical of business programs, though perhaps this should begin to change. It's especially suited to batch processing, that is, handling many records in the same way at the same time. (See "System Programs.")

Now, the divisions of field, record and file are conceptual: they are what the programmer thinks about, based on the information needs of a specific computer program.



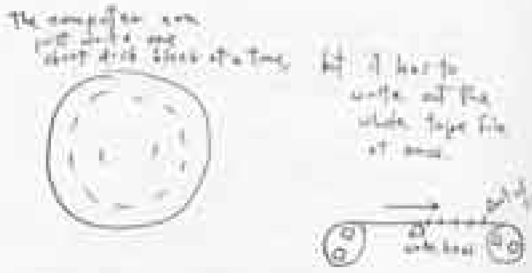

## BLOCKS

A block is something else, which may be related only to quirks of the situation.

A block is a section of stored material, divided either according to the divisions of the data or peculiarities of the device holding it, such as a disk drive. Short records may be stored many to a block. If records are long they may be made up of many blocks.

→ In particular, tape blocks can be almost any size, while disk blocks often have a certain fixed size (number of characters or bits) based on the peculiarities of the individual device. (This can be a pain in the neck.)

On the other hand, due to the quirks of magnetic recording, your program usually can't just change something in the middle of a block; the whole disk block or tape file has to be replaced. This is less trouble with a short disk block than a long tape file.



## TRADITIONAL CONVEYER-BELT PROGRAMS

Many traditional business programs are of this type, reading in one data record at a time, doing something to it (such as noting that an individual has paid the exact amount of his gas) and writing out a new record for that customer on the current month's tape.

## THE PROBLEM

Standardized fields, blocks and records are often necessary or convenient. But, on the other hand, the kinds of computer programs people find oppressive often have their roots in this kind of data storage and its associated styles of programming, especially the use of fixed-field records as the be-all and end-all. The more interesting uses of the computer (interactive, obliging, artistic, etc.) use a greater variety of data structures.

People's naive idea of "programming" is often a reasonable approximation to the notion of "data structure." Data structure is how information is set up. After it's set up, programs can twiddle it; but the twiddling options are based on how the information is set up to begin with.

# THE MAGIC OF DATA

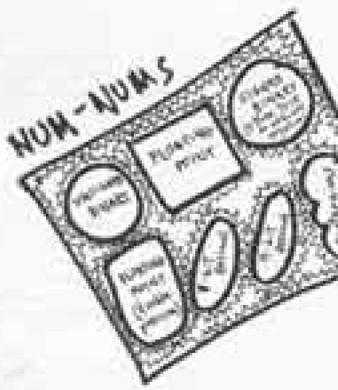
How does a computer program print something out on a printing machine? It sends the code for each letter out to the printing machine.

How does a computer program respond to something a user types in? It compares the codes that come in from the letters he types with a series of codes in memory, and when it finds a match between letters, numbers, words or phrases, branches to the corresponding action.

How does a computer program measure something? It takes in numerical codes from a device which has already made the measurements and converted them to codes.

DOES NOT COMPUTE!

Some TV writer's idea of a computer announces this when data are insufficient or contradictory. Ho hum.




# CODED-DOWN DATA: AN IDEA WHOSE TIME HAS PASSED

Codes are patterns or symbols which are assigned meanings. Sometimes we make up special codes to cut down the amount of information that has to be stored. On your driver's license, for instance, they may reduce your hair color to one decimal digit (four bits of information), since there are less than nine possibilities for quick identification of hair-color anyway.

Obviously, codes can be any darn thing: any set of symbols that is less than what you started with. But by compressing information they lose information, as the subtleties disappear (consider the use of letters A to F to grade students). When you divide a continuum into categories, not just the fineness of the categories, but the places you draw the line— called "breaks" or "cutting-points"— present problems. Such chopping frequently blurs out important distinctions. Coding is always arbitrary, frequently destructive and stupid.

Lots of ways now exist to handle written information by computer. These often present better ways to operate than by using codes of this type. But many computer programmers prefer to make you use codes.

(NOTE: there are two other senses of "code" used hereabouts: 1) the binary patterns made to stand for any information, especially on input and output; 2) what computer programs consist of, that is, lines of commands.)

# SOME POINTS

"Logical deduction" really consists of techniques for finding out what's already in a data structure.

"Logical inconsistency" means a data structure contradicts itself. Rarely does it happen that a computer helps you discover something new about a subject that you didn't suspect or see coming without the computer; after all, you have to set up a study in such a way as to make room to find things out, and you can only make room to find some things out.

# THE PUNCH CARD MENTALITY

Punch cards are not intrinsically evil. They have served many useful purposes. But the punch-card mentality is still around. This will be seen in the programmer who habitually sets things up so we have to use punch cards (when other media, or interactive terminals, would be better); who insists on the user or victim putting down numbers (when with a little more effort the program could handle text, which is easier for the human, or even look up the information in data it has already); who insists that people's last names be cut down to eleven letters because he doesn't feel like leaving a longer field or handling exceptions in his program; who insists on the outsider cutting his information into snarfy little codes, when such digestion, if needed at all, could be better done by the program; and so on.

The punch card mentality is responsible for many of the woes that have been blamed on "computers."

# IF YOU WANT NUMBERS, WE GOT 'EM

The basic kinds of number operations wired into all computers are few: just add (and sometimes subtract) binary numbers. However, up above the minicomputer range, a computer may have multiply, divide, and more. Fancier computers offer more types and operations on them.

PLAIN BINARY— Very important for counting. Represents numbers as patterns of 1's and 0's (or X's and Ohs, if you prefer). How to handle negative numbers? Two ways:

TRUE NEGATIVE— binary number with a sign bit at the beginning, followed by the number.

Trouble is, the arithmetic is harder to wire for this kind, because there are two zeroes (plus and minus) between 1 and -1.

ADDABLE NEGATIVE— this system does a sort of flip and begins a negative number with all ones. It means that the machine doesn't have to have subtraction circuitry: you just add the flipped negative version of a number, and that actually subtracts it. This has now caught on generally. (It's usually called "twos complement negative," which has some obscure mathematical meaning.)

BCD (Binary-Coded Decimal)— the accountant's numbering system. Used by COBOL (see p. 31). It's plain old decimal, with every numeral stored in four bits; the machine or language has to add them one numeral at a time, instead of crunching together full binary words.

FLOATING POINT— the scientist's number technique for anything that may not come out even. Expresses any quantity as an amount and a size.


The "amount" part contains the actual binary numerals, the "size" is the number of places in front of or after the decimal point that the number starts. Very important for astronomical and infinitesimal matters, since a floating-point number can be bigger, say, than

9,876,543,210,000

or smaller than

.00000001234567

For some people even this isn't precise enough, so they program up "infinite precision arithmetic," which carries out arithmetic to as many places as they want. It takes much longer, though.

WHAT'S AVAILABLE IN MACHINES AND LANGUAGES

Some machines, like the 360, are more-or-less wired up to handle several number types: binary, floating point, BCD. Little machines usually only have plain binary, so other types have to be handled by programs built up from that fundamental binary.

Languages make up for this by providing programs to handle numbers in some or all of these formats. There are languages that offer even more kinds of numbers—

IMAGINARY numbers
(two-part numbers following certain rules)
QUATERNIONS
(like imaginary numbers but worse)
and goodness knows what else.

On the other hand, some languages restrict what number facilities are available for simplicity's sake. BASIC, for instance, doesn't distinguish between integers (counting numbers) and those with decimal points; all numbers may have decimal points. TRAC Language only gives you integers to start, since it's easy enough to program other kinds of number behavior in (like infinite precision).

For historical reasons computers have been used mostly with numbers up to now; but that is going to be thoroughly turned around. Within a few years there may be more text— written prose and poetry— stored on computers than numbers.

During the recent massive lawsuit by Control Data against IBM, it was revealed that IBM had an awesome number of letters and communications stored on magnetic memory.

When I lived in New York, I had a driver's license with the staggering serial number

NO 3443 13866 3-4121-37

Now it may very well be, as in some serial numbers, that information is hidden in the number that insiders can dope out, like my criminal record or automobile accidents, if any. (N is my initial, and two of the digits show my date of birth, a handy check against alteration by thirsty minors. But the rest of it is ridiculous.) The fact that that leaves 15 more decimal digits means (if no other codes are hidden) that New York State has provision in their license numbering for up to 999,999,999,999,999 inhabitants. It is doubtful that there will ever be that many New Yorkers, or indeed that many human beings while the species endures.

In other words, either New York State is planning on having many, many more occupants, or an awfully inefficient code has been adopted, meaning a lot of memory space is wasted holding those silly big numbers for millions of drivers. However, that doesn't represent a lot of money. 15 million decimal spaces these days fits on a couple of disk drives. But it's an awful pain in the neck when you want to cash a check.

# INPUT AND OUTPUT CODES

Data has to get inside the machine somehow, and results have to get back out. Two main types of codes— that is, standardized patterns— exist, although what forms of data programs work on inside varies considerably. (The input data can be completely transformed before internal work starts.)

1. ASCII (pronounced "Askey," American Standard Code for Information Exchange). This allows all the kinds of numbers and alphabets you could possibly want (for instance, Swahili) for getting information in and out of computers.

ASCII is used to and from most Teletype terminals and keyscopes.

However, ASCII is also used for internal storage of alphabetical data in many non-IBM systems, and it is also the running form of a number of programming languages, such as TRAC language (see p. 18), TECO (see p. [illegible]), and GRASS (see p. 51).

IBM's deliberate undermining of the ASCII code is a source of widespread anger. (See IBM, p. 52.)

2. EBCDIC (pronounced "Ebsedick,") Extended Binary Coded Decimal. This was the code IBM brought out with the 360, passing ASCII by. (IBM seems to think of compatibility as a privilege that must be earned, i.e., paid for.) EBCDIC also allows numbers, the English alphabet, and various punctuation marks. This is used to and from most IBM terminals ("2741 type").

HOLLERITH, meaning the column patterns that go in on punched cards. (They can also come out that way, if you want them to.)

CARD-IMAGE BINARY. If for some reason you want exact binary patterns from your program, they can be punched out as rows or columns on punch cards.

STERLING. Just to show you how nominal things can get, the original PL/I specifications (see p. 31) allowed numbers to be input and output in terms of Pounds, Shillings and Pence (12 pence to the shilling, 20 shillings to the pound). No provision was made for Guineas (the 21-shilling unit), or farthings, unfortunately.

# MAGIC LANGUAGES

A computer language is a system for casting spells. This is not a metaphor but an exactly true statement. Each language has a vocabulary of commands, that is, different orders you can give that are fundamental to the language, and a syntax, that is, rules about how to give the commands right, and how you may fit them together and entwine them.

Learning to work with one language doesn't mean you've learned another. You learn them one at a time, but after some experience it gets easier.

There are computer languages for testing rocketships and controlling oil refineries and making pictures. There are computer languages for sociological statistics and designing automobiles. And there are computer languages which will do any of these things, and more, but with more difficulty because they have no purpose built in. (But each of these general-purpose languages tends to have its own outlook.)

Most programmers have a favorite language or two, and this is not a rational matter. There are many different computer languages— in fact thousands— but what they all have in common is acting on series of instructions. Beyond that, every language is different. So for each language, the questions are

WHAT ARE THE INSTRUCTIONS?
and
HOW DO THEY FIT TOGETHER?

Most computer languages involve somebody typing in the commands of your spell to a computer set up for that language. (The computer is set up by putting in a bigger program, called the processor for that language.)


(Same computer with different language processors loaded in its core memory)

Then, after various steps, you get to try your program.

Once you know a language you can cast spells in it; but that doesn't mean it's easy. A spell cast in a computer language will make the computer do what you want—

IF it's possible to do it
with that computer;
IF it's possible to do it
in that language;
IF you used the vocabulary
and rules of the language
correctly,
and IF you laid out in the spell
a plan that would effectively
do what you had in mind.

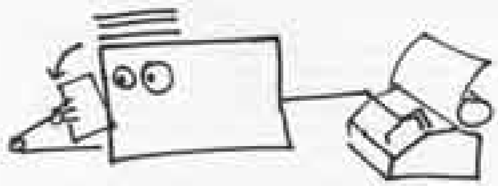
BUT if you make a mistake in casting your spell, that is a BUG. (As you see from the IFs above, many types of bug are possible.) Program bugs can cause unfortunate results. (Supposedly a big NASA rocket failed on takeoff once because of a misplaced dollar sign in a program.) Getting the bugs out of a program is called debugging. It's very hard.

DESIGNING COMPUTER LANGUAGES

Every programmer who's designed a language, and created a processor for it, had certain typical uses in mind. If you want to create your own language, you figure out what sorts of operations you would like to have be basic in it, and how you would like it all to fit together so as to allow the variations you have in mind. Then you program your processor (which is usually very hard).

## ★AN INTERPRETER★

carries out each instruction as it's encountered.



## ★A COMPILER★

changes the instructions of the language into another form to be processed later.

An Interpreter carries out,

A Compiler sets up.

# HOW DO COMPUTER LANGUAGES WORK?

Basically there are two different methods.

A compiling language, such as FORTRAN or COBOL, has a compiler program, which sits in the computer, and receives the input program, or "source program," the way the assembler does. It analyzes the source program and substitutes for it an object program, in machine language, which is a translation of the source program, and can actually be run on the computer. The relation of the higher language is not one-to-one to machine language: many instructions in machine language are often needed to compile a single instruction of the source program. (A source program of 100 lines can easily come out a thousand lines long in its output version.) Moreover, because of the interdependency of the instructions in the source program, the compiler usually has to check various arrangements all over the program before it can generate the final code.

Most compilers come in several stages. You have to put the first stage of the compiler into the computer, then run in the source program, and the first stage puts out a first intermediate version of the program. Then you put this version into a second stage, which puts out a second intermediate version; and so on through various stages. This is done fairly automatically on big computers, but on little machines it's a pain.

(In fact, compilers tend to be very slow programs; but that depends on the amount of "optimizing" they do, that is, how efficient they try to make the object program.)

An interpretive language works differently. There sits in core a processor for the language called an interpreter; this goes through the program one step at a time, actually carrying out each operation in the list and going on to the next. TRAC and APL are interpretive; it's a good way to do quickie languages.

Interpreters are perhaps the easier method of the two to grasp, since they seem to correspond a little better to the way many people think of computers. That doesn't mean they're better. For programs that have to be run over and over, compiling is usually more economical in the long run; but for programs that have to be repeatedly changed, interpreters are often simpler to work with.

A BLACK ART

Making language processors, especially compilers, is widely regarded as a black art. Some people have tricks that are virtual trademarks (see below).

Actually, the design of a language— especially the syntax, how its commands fit together— strongly influences the design of its processor. BASIC and APL, for instance, work left-to-right on each line, and top-to-bottom on a program. Both act on something stored in a work area. TRAC, on the other hand, works left-to-right on a text string that changes size like a rubber band. Other languages exhibit comparable differences.

MIXED CASES AND VARIATIONS (for the whimsical)

There are a lot of mixed cases. A load-and-go compiler (such as WATFOR) is put into the computer with the program, compiles it, and then starts it going immediately. An interpretive compiler looks up what to do with a given instruction by interpreting it into a series of steps, but compiling them instead of carrying them out. (A firm called Digitek is well known for making very good compilers of this type.) An incremental compiler just runs along compiling a command at a time. This can be a lot faster but has drawbacks.

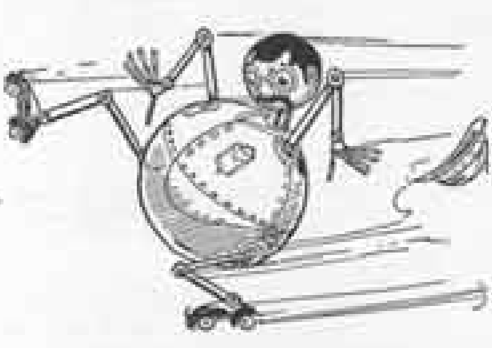
BIBLIOGRAPHY

David Gries, Compiler Construction for Digital Computers.
Not for beginners, but a beautiful book. Good on abstract theory of languages, too.

# DEBUGGING

A program is like a nose:
Sometimes it runs, sometimes it blows.

Attributed to Howard Rose
(Datamation, 1 Sep 71, 11.)



According to the grapevine...

a prestigious Southern university
had a program
where the number of months
was carelessly set to 12
(as a dimension in an array).
In November,
nobody got their checks
till this error was found.

candid photos

*Debugging means changing and fixing your program till it works the way you want it to.*

*This is the part of programming people like the least.*

*You run your program and then try to find out what went wrong. It could be a mistake in the basic thinking ("logic error"), or a clerical error in the particular choice of commands to carry out a well-thought-out process ("coding error").*

*Some systems allow you to debug interactively, from a terminal. This helps a lot. You can run parts of your program, get it to stop at certain points to let you look around, and so on.*

No program is ever fully debugged.

— folk saying

For every bug that goes out,
two more bugs go in.

— folk saying

# THE GREAT COMPUTER LANGUAGES

A certain number of computer languages are very widely accepted and used; I list them here. If you want to learn any of them, I believe that Daniel McCracken has written a manual on every one of them. (Not the variants listed, though.)

Why their names are always spelled with capital letters I don't know. (Generally they get let down in longer articles, though.)

## Good Old FORTRAN

FORTRAN was created in the late fifties, largely by John Backus, as an algebraic programming system for the old IBM 704. (However, the usual story is that it stands for FORmula TRANslator.)

Fortran is "algebraic," that is, it uses an algebraic sort of notation and was mostly suited, in the beginning, to writing programs that carried out the sorts of formulas that you use in highschool algebra. It's strong on numbers carried to a lot of decimal places ("scientific" numbers) and the handling of arrays, which is something else mathematicians and engineers do a lot (see Arrays under BASIC).

Fortran has grown and grown, however, after Fortran I came Fortran II, Fortran III and Fortran IV; as well as a lot of variants like Fortran FI ("irrational, and somewhere between III and IV"), WATFOR and WATFIV.

The larger Fortrans— that is, language processors that run on the bigger computers— now have many operations not contemplated in the original Fortran, including operations for handling text and so on.

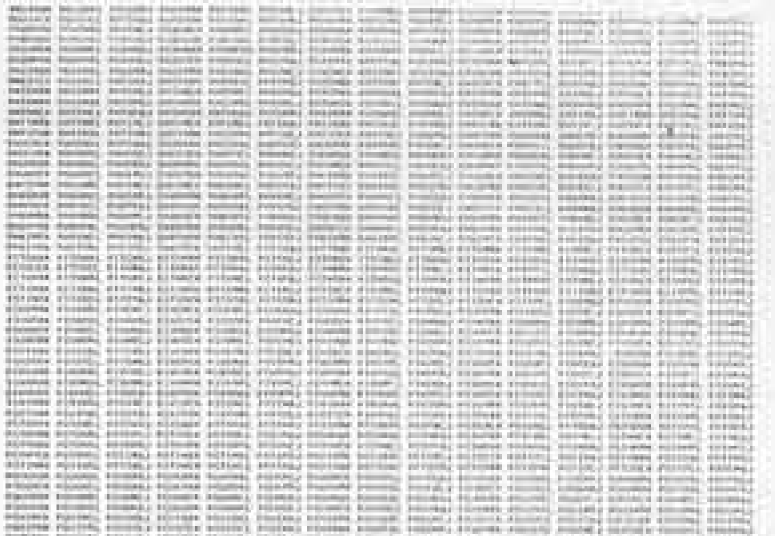
BASIC, presented earlier, is in some respects a simplified version of Fortran.

*This program was a surprise from Alan Bellas, a student at Chicago Circle. He was amazed by my practice of alphabetizing phone numbers, and wrote a program to do it automatically.*

*Premises of the program: you supply it with your phone number, and it prints out all the alphabetical combinations that could also be dialled to reach your telephone.*

Language: Fortran.

*Behold some of the combinations. The recipient picks out the one he likes from 26 pages of them.*



## ALGOL LOST, AND PL/I

ALGOL is considered by many to be one of the best "scientific" languages. It has been widely accepted in Europe, and is the standard "publication language" in which procedures for doing things are published in this country. It is different from FORTRAN in many ways, but a key respect is this: while in FORTRAN the programmer must lay out at the beginning of his program exactly what spaces of core memory are to have what names, in ALGOL the spaces in core memory are not given names except within subsections of the program, or "procedures." When the program follows gets to a specific procedure, then the language processor names the spaces in core memory.

This has several advantages. One is that it can be used for so-called "recursive" programs, or programs that call new versions of themselves into operation. I guess we better not get into that. But mathematicians like it.

Originally this language was called IAL, for International Algebraic Language, but then as it grew and got polished by various international committees it was given its new name. (I don't know if anyone consciously named it after Algol, the star.)

It has gone through several versions. Algol 60, the publication language, is one thing; Algol 70, the 1970 version, is much more complicated and strange.

Several versions of ALGOL have gotten popular in this country. One, developed at the University of Michigan, is called MAD (Michigan Algorithm Decoder); its symbol is of course Alfred E. Newman. Another favorite (for its name, anyway) is JOVIAL (Jules' Own Version of the International Algebraic Language), developed under Jules Schwartz (and supposedly named without his consultation) at System Development Corporation.

When IBM announced its System 360 back in 1964, there had been hope that they would support the international language committees and make Algol the basic language of their new computer line. No such luck. Instead they announced PL/I (Programming Language I), a computer language that was going to be all things to all men.

In programming style it resembled COBOL, but had facilities for varieties of "scientific" numbers and some good data structure systems. It is available for the 360 and for certain big Honeywell computers; indeed, the operating system for MULTICS (see p. 45) was written in PL/I. Whether there are people who love the language I don't know; there are certainly people who hate it.

*Below: Bellas' program to calculate the date of Easter. The language is Algol.*

## YECCCH, IT'S COBOL

Research and hobby types hate COBOL or ignore it, but it's the main business programming language. Your income tax, your checking account, your automobile license— all are presumably handled by programs in the COBOL language.

COBOL, or COmmon Business Oriented Language, was more or less demanded by the Department of Defense, and brought into being by a committee called CODASYL, which is apparently still going. COBOL uses mostly decimal numbers, is designed basically for batch processing (described elsewhere), and uses verbose and plonking command formats.

Just because it's standard for business programming doesn't mean it's the best or most efficient language for business programming; I've talked to people who advocate business programming in FORTRAN, BASIC, TRAC and even APL. But then you get into those endless arguments... and it turns out that a large proportion of business programmers only know Cobol, which pragmatically settles the argument.

There are people who say they've discovered hidden beauties in COBOL, for instance, that it's a splendid language for complex pointer manipulation (see Data Structures, p. 26). That's what makes horse racing.

## JCL

Some call it Despicable, Some call it 'Home'

*"After you study it for six months, it makes perfect sense."* —An IBM enthusiast.

JCL is a language with which you submit programs to an IBM 360 or 370 computer. "Submit" is right. Its complications, which many call unnecessary, symbolize the career of submission to IBM upon which the 360 programmer embarks. (See IBM, pp. 52-3, and 360, p. 41.)

## SNOBOL

SNOBOL is the favorite computing language of a lot of my friends. It is a list-processing language, meaning it's good for amorphous data. (It derives from several previous list-processing languages, especially IPL-V and COMIT.)

SNOBOL is a big language, and only runs on big computers. The main concept of it is the "pattern match," whereby a string of symbols is examined to see if it has certain characteristics, including any particular contents, relations between contents, or other variations the programmer can specify; and the string substitution, where some specified string of symbols is replaced by another that the programmer contrives.

## LISP

is probably the favorite language of the artificial-intelligence freaks (see p. M(2)). A fondness for LISP, incidentally, is not considered to reflect on your masculinity.

LISP is a "cult" language, and its adherents are sometimes called Lispians. They see computer activities in a somewhat different light, as composed of ever-changing chains of things called "cars" and "cudders," which will not be explained here.

LISP was developed by John McCarthy at MIT, based largely on the Lambda-notation of Alonzo Church. It allows the chaining of operations and data in deeply intermingled forms. While it runs on elegant principles, most people object to its innumerable parentheses (a feature shared to some extent by TRAC Language).

Joseph Weizenbaum, also of MIT, has created a language called SLIP, somewhat resembling LISP, which runs in FORTRAN. That means you can run LISP-like programs without having access to a LISP processor, which is helpful.

## THEN, THERE'S ALWAYS MACHINE LANGUAGE

If you feel like making programs run fast, and not take up very much core memory, you go to machine language, the computer's very own wired-up deep-down system of commands (see p. 32). It takes longer, usually, but many people consider it very satisfying.

Then, of course, if you have a particular style and approach and set of interests, you will probably start building up a collection of individual programs for your own purposes.

Then you'll work out simplified ways of calling these into operation and tying their results and data together.

Which means you'll have a language of your own.

# ROCK BOTTOM
## THE WORLD BENEATH THE HIGHER LANGUAGES

Every computer is wired to accept a specific system of commands. When these commands are stored in the computer's memory, and the computer's program follower gets to them, they cause it to respond directly by electronic reflex. This is called machine language, the very language of the machine itself.

In most available computers the machine languages are binary, meaning composed of only two alternative symbols. Binary because it's a sensible way of organizing the machine's structure: it permits programs to be reduced to a single common form of information, and permits programs to be stored in binary memory. Each individual instruction or command ordinarily occupies one memory slot, though some computers have commands of varying length.

Different computers have different machine languages, but the instructions of all computers are basically similar. Big computers have more commands, with more variations, and carry them out faster; but these variations are just extra ways of saving steps, not qualitatively different features.

These deep-down operations ARE ALL THE THINGS THE COMPUTER EVER DOES. However, in their combinations these instructions can be woven into chains and diadems of complex actions.

ALL COMPUTER PROGRAMS ARE EVENTUALLY WRITTEN OR ENACTED IN THE MACHINE'S PARTICULAR BINARY LANGUAGE.

Now, it is entirely possible to write your programs at this level, considering and arranging rock-bottom commands. This is called machine-language programming (and assembly programming: see examples a little later on). Indeed, working at this level is very highly respected in some quarters. Others avoid it. This is a very serious matter of taste and what you're working on.

Higher-level languages, seen on earlier pages, have more convenient forms for people, but must be translated, either ahead of time or on a running basis, to the bottom-most codes that make things happen in the machine. All of them are built out of machine language. Writing the language processors, programs that accept or translate these higher-level languages, is considered a black art. (See p. 3•.)

---

Every programmable device has a "machine language," or rock bottom code system that activates the thing directly; its program follower responds electrically to these codes, and enacts them one instruction at a time.

True computers are programmable devices that can modify their own instructions, change their sequence of operations and do other versatile stuff.

# What the Computer Really Is
# COMPUTER ARCHITECTURE
## The Nuts and Bolts

Computers are basically alike. Ignore their appearances: a roomful of roaring cabinets may have a great deal in common with a small blinking box. Indeed, they may have the same architecture, or structure, and therefore be the same computer.

The structure of computers, in their glorious similarities and fascinating differences, is called computer architecture.

(For the architecture of a beginner's computer, see p. 33; for the architecture of some famous computers, see p. 4•-•.)

Computer architecture covers three main things: registers (places where something happens to information); memories (places where nothing happens to information); their interconnections; and machine language, all the bottom-level instructions (for this last see "Rock Bottom," p. 32).

REGISTERS AND MEMORIES

Computers are made, basically, of two things: registers and memories. A register is where something happens to information; a memory is where nothing happens to information. Let's go over that slowly.

A register is a place where something happens to information: the information can be flipped around, tested, changed by arithmetic, or whatever. (We noted earlier that registers are what connect a computer to its accessories. They are also principal parts of the computer itself.)

A memory is a place where nothing happens to information. A program puts the information there, and there it stays till some program pulls it out again or replaces it.

A main or general register (often called the accumulator, for no good reason) is where the program brings things to be worked on, tested, compared, added to and so on. There can be several of them in a computer.

Other registers perform other functions in the computer; a given computer's design, or architecture, is largely the arrangement of registers and the operations that take place between them.

The reason we don't just have all registers—and no memories at all— is that registers traditionally cost more than memories. (However, some machines are being tried that have all working registers instead of memory. See STARAN, p. 43.)

Memories come in all sizes and speeds. So lots of computers have big slow memories, such as disk memories, along with their small fast memories.

A memory consists of numerous holding places or storage locations, each holding one standard piece of information for the computer, a word having a specific number of bits (see p. ). We must stress: a "COMPUTER WORD" HAS NOTHING TO DO WITH ENGLISH WORDS OR ALPHABETICAL CHARACTERS. The term refers to a specific machine's standard memory slot, having a fixed number of bit positions.

One important reason for this standardization is that each holding place, or memory location, can be given a number or address. If every slot in the memory has an address, information can be stored in specific places

and gotten back out of specific places

A core memory has a definite rhythm or cycle, into which it divides the passing time. The memory cycle of a core memory is so important that its duration is often called the cycle time of the computer. A request to the core memory made at the beginning of the cycle is honored at the end of the cycle. Core cycles are very fast, being these days about one microsecond, or millionth of a second.

A core memory can only perform one action (store or fetch) during one memory cycle.

Core cycles during which nothing is requested of the memory simply go by.

One last point about core memories. The number which specifies an address to the memory is a binary pattern— just like all the other information (see "Binary Patterns," p. 33). (Or more exactly, whatever binary pattern is supplied to the memory as the address to store or from which to fetch, that pattern will be treated as the address to store or from which to fetch; that pattern will be treated as a binary number whether it was supposed to be or not. It could be the alphabetic word GRINCH which got there by mistake (see "Debugging," p. 3•), but the memory will treat it as an address number and go to the address specified by that pattern.

THEN WHAT ARE THE DIFFERENCES BETWEEN COMPUTERS?

- The word length (number of bit-spaces in a main register and memory slot)
- The number of main registers and what they can do, i.e., how they are set up and what operations can take place in and among them, i.e., the instruction set (see nearby);
- The amount of memory;
- The accessories or peripherals;
- The cycle time.

Here's the computer, then, in all its glory: a device with a symbolic program, stored in a memory, being stepped through by a program follower.

The commands of the program cause the program follower to carry out the individual steps requested by each command of the program.

THE ROCK BOTTOM PROGRAM FOLLOWER

How, you ask desperately, does this innermost program follower work? The one that is built into the computer?

Aha.

Basically it consists of two specific registers, the Program Counter (usually abbreviated PC) and the Instruction Register (usually abbreviated IR), and other electronic stuff, loosely termed "decoding logic."

(Since we are already visualizing the program follower as a little hand, let's think of the index finger as the program counter and imagine that the thumb can flip an instruction into a little cup, the Instruction Register or IR. What the heck.)

WHEN a program is set into operation, the binary pattern specifying its first address in memory is put into the program counter.

Then the instruction at that address is fetched to the program follower (that is, put in the instruction register), decoded and carried out.

THEN THE PROGRAM COUNTER AUTOMATICALLY HAS ONE ADDED TO IT, SO IT POINTS TO THE NEXT INSTRUCTION.

The instruction pulled from memory is held in the command or instruction register and there decoded by the system's electronics.

It is of no concern to the programmer how this is done electronically. (And indeed electronics is generally of little concern to computer people, unless they are trying to design or up-grade computers or other devices themselves. Indeed, the electronic techniques are constantly changing.)

All we need to know is that an electrical decoding system (called the logic circuitry) carries out the specific instruction— for instance, by shutting off the path to the memory, turning on the adding circuit, and opening paths through the adding circuit and back to the main register.

Now that the program counter holds the number of the next instruction it in turn is accordingly fetched and executed.

And so it continues.

When an instruction calls for a jump or branch in the program, what happens?

The jump command causes a new number to be stuffed into the program counter. That's what, and so that's where the program goes next.

ALTERNATING CYCLES

Many instructions tell the program follower to take a data word (also a binary pattern) from memory and put it in a main register or vice versa.

Such an instruction is translated by the decoding logic into a request to the memory.

Since a core memory can only do one thing during one of its cycles, the next instruction in the program cannot be fetched until the data has moved in or from the memory.

Thus in many types of program the cycles alternate:

> Instruction cycle (fetch the next)
> Data cycle
>   (data goes to or from memory),
> Instruction cycle,
> Data cycle,
>   and so on.

---

# FUNDAMENTAL OPERATIONS OF COMPUTERS
## A GREAT MYSTERY IS ABOUT TO UNFOLD.

YOUR BASIC COMMANDS, NOW

(Computers exist which do little more than these, and yet they can in principle do anything fancier computers can do.)

TO BE SHOWN: The following are the rock-bottom basic operations of computers, available as specific instructions in all computers (with some variation):

The first seven listed below will be used in the extended example in the next spread.

LOAD a binary pattern from core memory to a main register.

STORE a binary pattern in core memory from a main register.

SEND OUT ("OUTPUT") a binary pattern to an external device.

BRING IN ("INPUT") a binary pattern from an external device.

ADD TWO binary patterns together. (This causes them to be treated as numbers, whether they were to begin with or not.)

JUMP--
Go to another part of the program and forget you were here.

TEST TWO binary patterns against each other, and branch or not in the program depending on the result.

NOT TO BE SHOWN: Here are the rest of the utterly fundamental commands of computers. (These are not used in the forthcoming example.)

TEST ONE SPECIFIC binary pattern, and branch in the program depending on the result.

SET AN ACCESSORY IN OPERATION/TURN IT OFF.

REVERSE (or "COMPLEMENT") a binary pattern—changing all the X's to 0's and vice versa.

SLIDE (or "SHIFT") a binary pattern sidelong through a register.

FLIPPER (or "LOGICAL") operations between two binary patterns, especially--

> OR (or "INCLUSIVE OR" or "IOR")-- result is an X where either original pattern was an X.
> AND (or "MASK")-- result is an X only where both original patterns had an X.

FANCY OPERATIONS

The following operations are desirable but not strictly necessary, and many computers, especially minicomputers, don't have them all.

SUBTRACT. (Can also be done if necessary with combination of adds and flips.)

MULTIPLY. (Can also be done if necessary with combination of adds, shifts and tests.)

DIVIDE. (Can also be done if necessary with combination of subtracts, shifts and tests.)

MORE FLIPPER ("LOGICAL") operations:

> XOR (or "EXCLUSIVE OR")-- result is an X only where one pattern had an X, but not both.
> NAND-- reversed AND.
> NOR-- reversed OR.

SUBROUTINE JUMP--
"Go to another part of the program but remember this place because you'll be coming back on your own."

RETURN FROM SUBROUTINE--
"Go back to wherever it was in the program that you last came from."

PUSH (on Stack machines only, see p. )-- take a binary pattern and put it on top of the Stack.

POP (on Stack machines only, see p. )-- take whatever binary pattern is now on the top of the Stack.

ADD ONE (or "INCREMENT")-- (Useful when you're counting the number of times something has been done.)

SUBTRACT ONE (or "DECREMENT," not "excrement")-- (Also useful when you're counting the number of times something has been done.)

ASTRONOMICAL/INFINITESIMAL ARITHMETIC (or "FLOATING POINT" arithmetic)-- operates on a certain number of Significant Digits and keeps separate track of the decimal point-- actually a Binary Point, since it's rarely if ever done decimally.

→Very important in the physical sciences.

Almost any operations can be "built in". The sky is of course the limit, since any electronic operation can be added to a computer's instruction-set if desired-- say, "turn on the electric blender" or "multiply quaternions"-- but the former is more easily done as an output instruction, and the latter as part of a program.

Somehow
LOADING, STORING,
MODIFYING
AND TESTING
BINARY PATTERNS
DOESN'T SEEM
TERRIBLY FRAUGHT
WITH POSSIBILITIES,
but the endless variations and ramifications make chess look like tic-tac-toe.

And part of the power, of course, is in the great speed, the teeny fraction of a second each step takes; five hundred operations yet take only about a thousandth of a second. So no matter how intricate the assortment to which these tiny steps are built, it will happen awfully fast.

A computer, then, internally just consists of certain places to work on information (main registers), certain places to keep it the rest of the time (memories); certain pathways and interconnections between them; an instruction-set having certain powers whose instructions can be operated on out of memory; and a program follower that carries out the instructions of that instruction-set.

INSTRUCTION-SET:

The system of command patterns designed and wired into a particular computer, each with its exact results.

(The instructions in the set are the vocabulary of a machine language.)

# A WIND-UP CROSSWORD PUZZLE

We look at last at what really happens inside a given computer. It must be a specific computer because there is no single basic language for all computers. For simplicity's sake (like most introductory texts) we hereby present a fictitious machine.

## THE ★ FIDO ★

(Faithful Instrument, Domesticated and Obliging).

The FIDO is a twelve-bit machine. The main register (it has only one) is twelve bits long, and every memory slot is twelve bits long.

Every instruction is twelve bits long; every data word is twelve bits long, though of course much longer pieces of data can be put together by taking more than one twelve-bit word.

Some rudimentary instructions of the FIDO are listed in a nearby box. The instructions of the FIDO are of two types: plain ones that just use the main register (like CLEAR), and the divided ones, which select a memory slot or output device. On the FIDO these are divided into an operation code (opcode) of five bits— the bits that tell the program follower what the operation is to be, and an address of seven bits, specifying which memory slot (or external device) is to be operated on.

These seven bits allow exactly 128 different patterns, (from OOOOOOO to XXXXXXX), which means we can select among exactly 128 different memory slots. (See Binary Patterns, p. 33.) [p. 33.]

The Fido comes with one row of lights and switches; the row of lights can show the contents of any specific working register or memory slot. When the computer is stopped, this is helpful for debugging programs (see p. 36.)

Ah, if only we could tell you all about the FIDO here! Its many more instructions. The option bits in the commands that allow fancy variations, or the option bits in the interfaces, spoken of earlier, which allow the program to give different commands to external devices.

But let's get on with a program for the FIDO. Thrill to the pulsating rhythms of...

BUCKY'S WRISTWATCH!
(for next page)

# BASIC INSTRUCTIONS OF THE FIDO COMPUTER.

For a revelation of its secret identity, see below.

| OPCODE (binary pattern selecting operation) — 5 bits | ADDRESS (binary pattern selecting where to perform operation) — 7 bits | OPERATION CALLED FOR |
|---|---|---|
| XXXXX | ooooooo (don't matter) | CLEAR AC: This instruction causes the AC to be filled with zeroes. |
| OOXOO | ooooooo (address goes here) | ADD (from memory to AC): This adds the contents of the specified memory location to the contents of the AC. Result remains in the AC. Whatever was in the memory before is still there. This instruction is also used to bring a new pattern to the AC, copying it from the specified memory location; but you have to CLEAR the AC first, so you're adding it to zero. |
| OXXOO | ooooooo (address goes here) | STORE: This instruction copies the contents of the AC to the specified memory location. Whatever was in the memory location is destroyed. Whatever was in the AC is still there, too. |
| XXOOO | ooooooo (address goes here) | INPUT*: This instruction copies the contents of a specified device register to the AC. |
| XXOOX | ooooooo (address goes here) | OUTPUT*: This instruction copies the contents of the AC to a specified device register. |
| XOXOO | ooooooo (address goes here) | JUMP: This instruction makes the program follower take its next instruction at the specified address and go on from there. |
| OOOOX | ooooooo (address goes here) | TEST, SKIP IF EQUAL**: This is a common test instruction, permitting the program to branch depending on various conditions. The contents of the AC are compared with the specified core memory location. If they are not the same, the program continues and takes the next instruction in the normal fashion. If the two patterns are the same, the program follower SKIPS the next instruction and goes on to the one after. |

Whatever the next instruction is, then, determines the course of events if the two patterns turn out to be the same.

For instance, that middle instruction can be a JUMP instruction, taking the program to a whole other part of core memory and a new series of events.

FIDO POCKET CARD

| | |
|---|---|
| XXXXXooooooo | CLEAR |
| OOXOOooooooo | ADD from mem. |
| OXXOOooooooo | STORE in mem. |
| XXOOOooooooo | INPUT |
| XXOOXooooooo | OUTPUT |
| XOXOOooooooo | JUMP |
| OOOOXooooooo | TEST, SKIP IF EQUAL |

* Note: these instructions have been changed slightly to protect the innocent (true).

** This instruction does not exist on the PDP-8. Actually it offers a wider choice, which we can't go into here. Sophisticated instruction-packing makes the PDP-8 remarkably efficient considering its small 12-bit word length.

If you want information on the machine language and assembly language of any given machine, write the manufacturer for the programming manual. There may also be a pocket card.

# INSTRUCTION LAYOUT

An overall aspect of computer design is the matter of how to pack into the so-many bits of an instruction word all the options the programmer should have.

For no particular reason the instruction select bits are usually on the left, the address bits on the right, and option bits (no room for them in this book, unfortunately) in the middle.

The number of bits in the address determines the number of places in the memory that the programmer can choose among. 15 bits in the address means a choice of 32,768 memory locations; 7 bits means a choice of only 128. (See "Binary Patterns," p. 33.)

Generally a specific computer has more than one instruction layout.

Deciding what the instruction layouts are to be hinges on the architectural design of the computer (see p. 33) and the instruction-set. It all gets worked out together.

It's ultimately a matter of design elegance, but the consequences are very concrete. An elegant instruction-set is easy to use and therefore saves a lot of time and money. (Anyone interested in studying the matter might want to compare the PDP-11, a 16-bit computer with a brilliantly designed instruction-set, with some other 16-bit computers.)

GUESS WHAT?

The FIDO is nothing but a stripped-down version of that beloved family pooch of computerdom,

## The PDP-8.

(Described p. 41.)

If you buy a PDP-8 from Digital Equipment Corporation, you get all this and more. (Except for the external devices.) And the PDP-8, of course, allows much bigger memories than 128 slots, but that's too complicated for here.) Arf.

# BINARY PATTERNS

are what the computer operates on deep down. "Binary" just means that only two symbols are used (just as "decimal" means that ten symbols are used). Patterns of binary symbols happen to be electrically convenient, so that's how computers are built, but that would change if some more convenient set of symbols came along.

Binary patterns are very systematic and easy to deal with. Consider the number of binary symbols you can have in just four spaces. ✱LET'S USE THE LETTERS X AND O, AND PUT THEM IN ALPHABETICAL ORDER, SO YOU'LL SEE THAT WE'RE TALKING ABOUT PATTERNS, RATHER THAN NUMBERS.

| | | | |
|---|---|---|---|
| O | O | O | O |
| O | O | O | X |
| O | O | X | O |
| O | O | X | X |
| O | X | O | O |
| O | X | O | X |
| O | X | X | O |
| O | X | X | X |
| X | O | O | O |
| X | O | O | X |
| X | O | X | O |
| X | O | X | X |
| X | X | O | O |
| X | X | O | X |
| X | X | X | O |
| X | X | X | X |

You can see that the pattern repeats in certain interesting ways. Each column repeats itself as you read down; adding a new position to the left doubles the number of possible patterns you can have in the row.

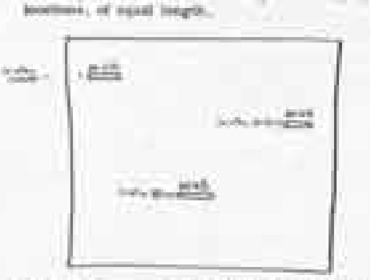
These are the infamous "bits" you have heard of. As you can see, there is nothing hard or complicated about them. The number of bits in a thing are the number of spaces which can be either X or O.

Now, the most basic fact about any computer is its word length: that is, the number of spaces in a standard memory slot of that computer.

A "12-bit computer" (like the PDP-8) has memory words that are all twelve bits long. A "16-bit computer" (like the PDP-11) has memory words that are all 16 bits long.

Actually computers with small word lengths like these are called minicomputers. Big computers have much bigger word lengths. The IBM 360 has a 32-bit word length. The Control Data 6600 has a 60-bit word.

Now, it is an interesting fact that not only are computer memories divided up into slots, or locations, of equal length,



but each of those locations has an address, that is, a number by which the contents of the location can be found. And these numbers are binary.

Many forms of information are kept in binary patterns which are not numbers. For instance, letters of the alphabet are usually stored as 8-bit patterns.

| X | X | O | X | O | O | O | X |
|---|---|---|---|---|---|---|---|

THE LETTER "Q"
(IN ASCII CODE)

All computers can in principle do the same things, some faster. However, some are too slow or too small ever to do what others can, though the types of their operations are similar.

Some computers (and their languages and facilities) are much more convenient for programmers than others, because their instruction-sets are better.

This is no small matter.

(But it's a big matter of taste and argument among computer people.)

The big point is:

AT THE BOTTOM PROGRAMS ARE BINARY

AND DATA IS BINARY,

since it's all stored in binary memory.

But since that suits few people's individual purposes, we build up HIGHER LANGUAGES AND DATA STRUCTURES, so that different users deal with different mechanisms corresponding better and more conveniently to the structures that interest them.

However, we will have to stop using these X's and O's. It's not really done, so we will switch to the more usual way of writing binary patterns with 1's and zeroes. (Apologies to readers who hate numbers; but remember that these patterns, while we may write them out as 1's and zeroes, may represent wholly non-numerical kinds of information.) That means the letter Q is

| 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |
|---|---|---|---|---|---|---|---|

but it's still the letter Q.

Of course, bits may also represent numerical information. And so we pass on to

BINARY NUMBERS.

These are the same old binary patterns, but when we decide to treat them as numbers, they are binary numbers.

Let's count. Note that these are the same combinations of bits as before, merely put in the more usual notation.

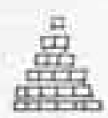
| decimal number | binary number |
|---|---|
| 0 | 000 |
| 1 | 001 |
| 2 | 010 |
| 3 | 011 |
| 4 | 100 |
| 5 | 101 |
| 6 | 110 |
| 7 | 111 |
| 8 | 1000 |
| 9 | 1001 |
| 10 | 1010 |
| 11 | 1011 |
| 12 | 1100 |
| 13 | 1101 |
| 14 | 1110 |
| 15 | 1111 |

As you observe, the higher numbers need more and more bits to hold them.

This brings up some interesting facts.

CERTAIN NUMBERS ARE SPECIAL because they are the number of things that can be specified by a certain number of bits.

| Special number | |
|---|---|
| 2 | one bit |
| 4 | two bits |
| 8 | three bits |
| 16 | four bits |
| 32 | five bits |
| 64 | six bits |
| 128 | seven bits |
| 256 | eight bits |
| 512 | nine bits |
| 1024 | ten bits |



("ONE K" is 1024; memories and everything else count in K's, or multiples of 1024.)

Actually the term "K," standing for "kilo-," should mean one thousand, and the term BK, or Binary K, is used by fussy people to stand for the very important nearby number 1024. But computer people generally use expressions ending in K for the following special numbers:

| NUMBER | THAT'S HOW MANY COMBINATIONS FIT IN |
|---|---|
| 2048 ("2K") | eleven bits |
| 4096 ("4K") | twelve bits |
| 8192 ("8K") | thirteen bits |
| 16,384 ("16K") | fourteen bits |
| 32,768 ("32K") | fifteen bits |

Above this number they increase very fast, and we generally have to look them up. But the idea is this: the number of bits used to select something limits the number of things you can select among. For instance, if you have a computer memory with 32K different locations, you need fifteen bits exactly to specify a location in memory.

Here are some ramifications.

• The word length of a computer determines how large a number it can hold. A computer with a twelve-bit word can only hold a number up to 4095 in one memory location (since we use 000 000 000 000, the first combination, to stand for zero). If we want to use larger numbers we have to set aside two or more word locations per number. (A 16-bit computer can hold a number up to 65,535 in one memory location.)

• In designing data structures, if you use binary codes (rather than, say, alphabetical characters), you have to allow enough bits for all the alternatives that might turn up.

• In the design of the wired-in instructions for a computer, therefore, the number of bits set aside to specify an address in core determines whether that instruction can select from the whole memory, or just a part of it.

# ★BUCKY'S WRISTWATCH★

There is a certain folk hero whom the people all call Bucky. It is said that he wears three wristwatches: one for where he is now, one for where he will be next, and one that tells what time it is at his home.

Well now. Here's an example of a little problem on which to try our FIDO computer.

Let's wire up a magic wristwatch for Bucky the Folk Hero, one that will use a teeny FIDO on a chip (the coming thing), attached to three rows of numerical readouts (like those on pocket calculators).

This application is not so absurd as you might think.

It is obviously quite simple in principle.

It will let us see some of the ways that the rock-bottom machine languages of computers are used.

## ABOUT THIS WONDERFUL PROGRAM.

Naturally this got saved for last, and what is presented here shows it.

The example was meant to be a case of not-very-numerical programming that would show the abstractness of it all. The program itself has no intrinsic quality related to the problem; that much should be visible.

Anyhow, I programmed this myself a few weeks ago in the FIDO language, and was very pleased with it, but then discovered a couple of appalling bugs. As time closed in on this project I asked my friend Mike O'Brien to code the program, and he kindly consented, taking time out of his previous weekend plans. Here is Mike's program, for which I am grateful.

HowEVer, after it was set in type, Mike realized that it too has some gross flaws and would not work as here presented. We thought of having a chocolate chip cookie contest for corrections, sending out chocolate chip cookies to entrants fixing it up, but we don't have such a computer and we wouldn't run the program if we had one anyway, so see if you can get the basic idea of it, and if you are a real wise guy fix the program for your own satisfaction, and that will be that.

The basic idea is that we have a FIDO, presumably on a single integrated circuit chip, attached to thirteen external devices (or peripherals, or input-output devices, or I/O devices or whatever). These devices are a timer or clock, which reaches zero once per minute— this is a computer clock, meaning a timer, not something that people can read— and the three rows of numerical readouts that are the desired Superwatch.

For simplicity's sake we assume here that each numeral is interfaced to do either input or output; thus the FIDO computer can ask any given numeral what it says, and change its contents.

The Bucky Wristwatch is going to give time on a twentyfour-hour basis, not twelve, like at NASA and suchlike places. After 12:59 comes 13:00. After 23:59 comes 01:00.

WOOPS;

01:00 should come on the clock after 24:59. The days will be a little short for Bucky, since we test the second digit for a 3 rather than a 4. This is called a LOGIC ERROR, meaning it was thought out incorrectly, rather than a "coding error" (meaning the program fails to carry out the steps that have been thought out).

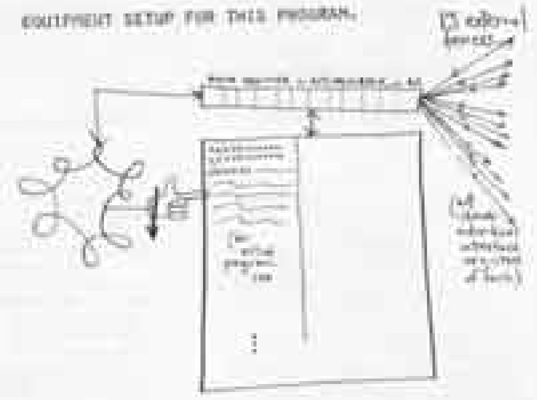
So you begin to see, uh, how bad, some of the, uh...

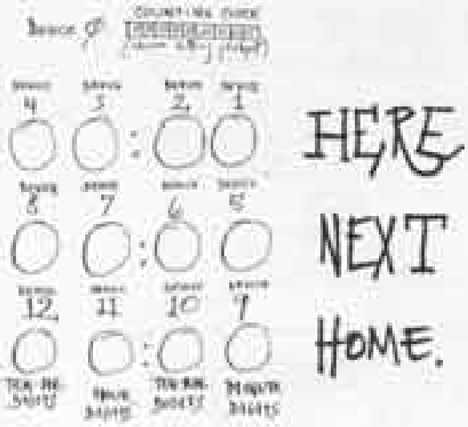
The bulk of the program is occupied with testing the numerals and changing them. However, in proportions of activity, the poor thing is going to spend most of its time saying, "Is it time yet? Is it time yet? Is it time yet?" (That's the second, third and fourth instruction.)

Because the FIDO selects the particular input-output device with the last seven bits of an input or output instruction, this has been done with "address modification" arithmetic: creating an output instruction to address a particular device by adding the instruction to the name of the device. This is an ancient and honorable programming trick.

In several cases, the program chooses a device to examine, or fill, by taking a blank input or output instruction (kept at locations X OXO XOX and X OXO XXO, respectively) and adds N, in the AC, to a counting number that is being used to step around in the array of numerals. (This counting number is "N," stored in location X OXO XXX.) (These instructions were put into the slots in octal form, as "4ØØØB" and "5ØØØB" respectively. The slashes are meant to distinguish zeroes from Ohs. The "B" at the end (in the assembly listing) means that the assembler is supposed to translate these numbers to Binary, taking them three bits at a time: 4 Ø Ø Ø comes out to XOO OOO OOO OOO.)

EQUIPMENT SETUP FOR THIS PROGRAM.




Note that in this flowchart

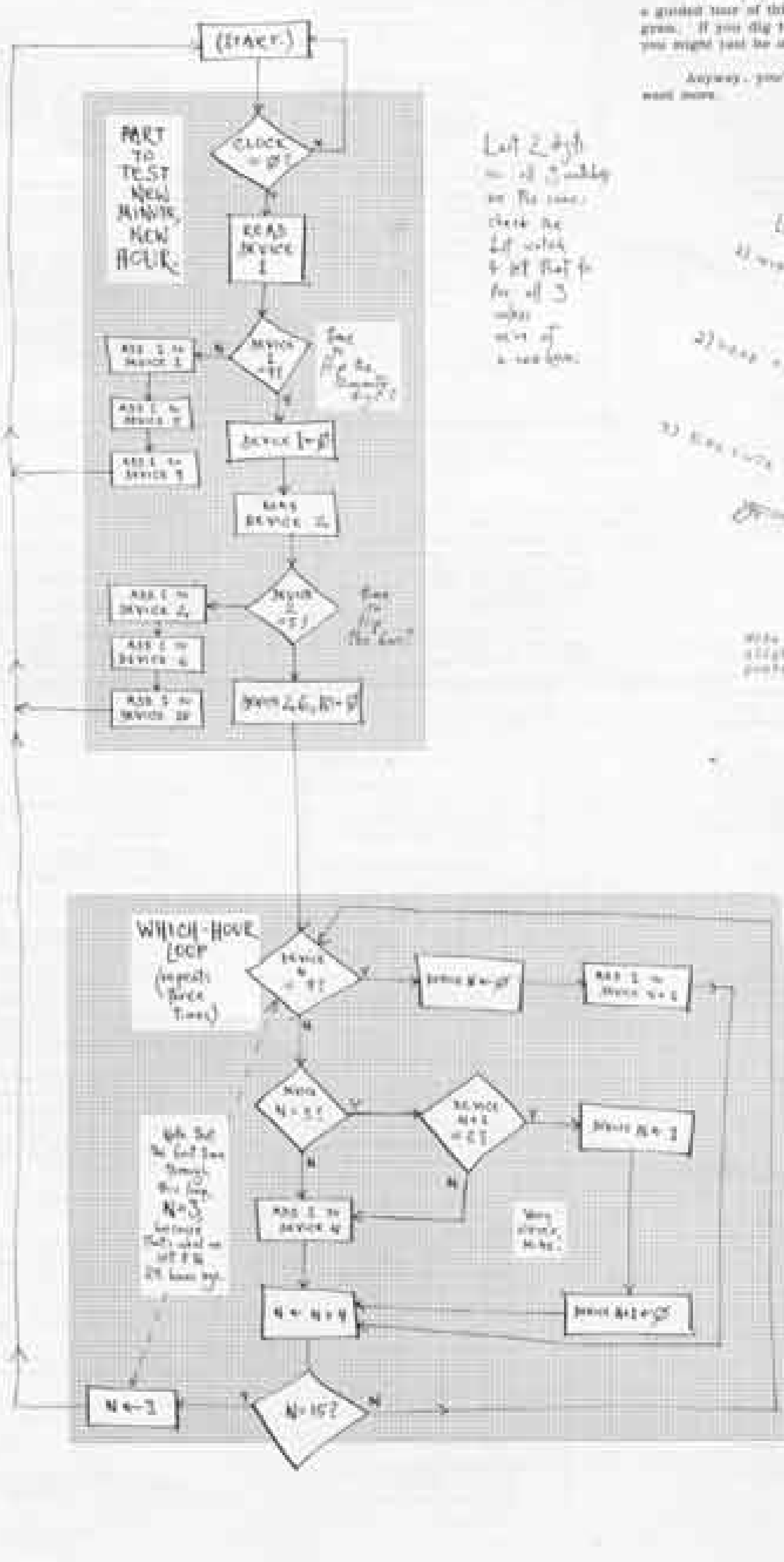
A ← 3

means, "stuff the number 3 into the variable A." A variable is a named location in core memory.

Anyhow, what the program is really doing when it finds the timer has reached zero, is testing whether the rightmost digit is a nine. (It only has to test one, since minutes are the same round the world.) If it's not nine, it just adds one to each— a part of the program called ADDIN, starting at XXO OXO. If it's nine, however, it sets the final digits all to zero, and then tests the tens digit to see if it's a five, meaning the end of an hour. (The number five has been ingeniously stored in a location which Mike has called FIVE, which assembled to slot number X OXO OXO. If you look there, you will see that the slot does, indeed, contain the binary pattern for the number 5.)

What a pity there is no time to take you on a guided tour of this profound, magnificent program. If you dig this sort of thing, however, you might just be able to dope it out.

Anyway, you've had your taste. Hope you want more.




Mike O'Brien's slightly disgruntled postscript to the program.

This is what the program looks like in the computer's core memory. (A printout like the following is called a machine-language listing.)

Since all the addresses are filled in, this program is said to be in absolute binary. If they weren't filled in, it would be called relocatable binary.

Machine-language listings come in different flavors. A binary listing (or dump) is generally in ones and zeroes. An octal listing groups the bits by threes and substitutes the numbers zero through seven for the different combinations of three bits. The other main kind, the hexadecimal listing or dump (on IBM things), groups the bits by fours and substitutes the numbers 0-9 and the letters A to F, for the sixteen different combinations of four bits.

This is what the program looks like when you set it up for the Assembler, which is the easier way.

A program laid out like this is called an Assembly Listing. Studying it may help you debug (see p. 37).

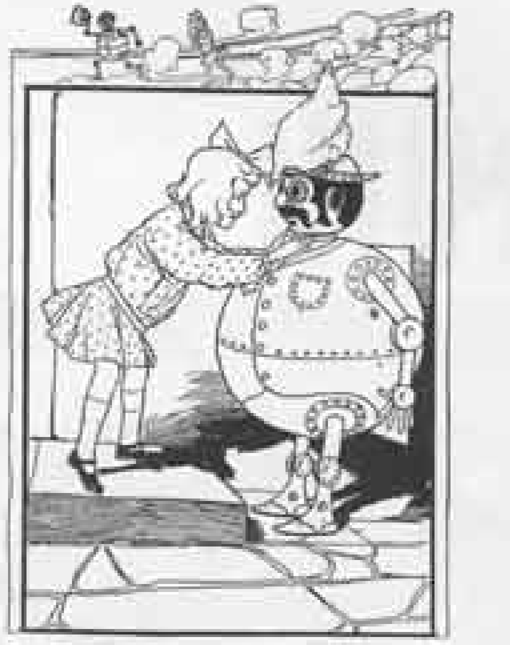
An easy-to-remember alphabetical code is used to represent each fixed instruction desired. Such an abbreviation is called a mnemonic; usually they're more cryptic. The mnemonics are turned by the assembler into the binary opcode.

You don't have to know the actual addresses in core memory; you just use alphabetical names or labels, and the Assembler figures out where they really go and puts in the binary addresses.

Desired numbers, such as 5, are plugged into the address parts of instructions.

YOUR OWN COMMENTS (here set off with slashes) can stay here too.

In this PDP8 example, the Assembler follows two common practices: it recognizes a label because it ends in a comma, and recognizes a comment because it begins with a slash.



"THIS COPPER MAN IS NOT ALIVE AT ALL."

## Bucky's Wristwatch in BINARY

## Bucky's Wristwatch in ASSEMBLY LANGUAGE

ADDRESS (that is, in core memory) — CONTENTS (actual switchable BITS …) — LABELS (names programmer gave) — OP NAMES (Mnemonics) — PROGRAMMER'S COMMENTS (so he doesn't forget, or the next guy can tell)

CORE MEMORY

```
      OOO   XXXXXOOOOOOO   START,    CLEAR
      OOX   XXOOOOOOOOOO   CHKCL,    INPUT 0        /CLOCK IS I/O SLOT #0000000
      OXO   OOOOXXOOXXOX             TEST ZERO      /A NEW MINUTE?
      OXX   XOXOOOOOOOOX             JUMP CHKCL     /NO. CHECK CLOCK AGAIN.
      XOO   XXOOOOOOOOOX             INPUT 1        /YES. READ MINUTE SLOT OF 1ST WATCH
      XOX   OOOOXXOOXOXX             TEST NINE      /IS IT A 9?
      XXO   XOXOOXXXOOOO             JUMP ADMIN     /NO, GO TO MINUTE INCREMENTER.
      XXX   XXXXXOOOOOOO             CLEAR          /YES, SET EACH
  OOX OOO   XXOOOXOOOOOX             OUTPUT 1       /TEN-MINUTE DIGIT
  OOX OOX   XXOOXOOOOXOO             OUTPUT 4       /TO ZERO.
  OOX OXO   XXOOXOOXXOOX             OUTPUT 8
  OOX OXX   XXOOOOOOOOXO             INPUT 2        /CHECK TEN-MINUTE DIGIT.
  OOX XOO   OOOOXXOXOOOO             TEST FIVE      /NEW HOUR?
  OOX XOX   XOXOOOXXOOXX             JUMP ADSTEN    /NO, GO TO TEN-MINUTE INCREMENTER.
  OOX XXO   XXXXXOOOOOOO             CLEAR          /YES, SET EACH
  OOX XXX   XXOOXOOOOOOO             OUTPUT 2       /TEN-MINUTE DIGIT
  OXO OOO   XXOOXOOOOOXO             OUTPUT 6       /TO ZERO.
  OXO OOX   XXOOXOOOOXOX             OUTPUT 10
  OXO OXO   OOXOOOXOOXXX   ROUND,    ADD N          /GET CLOCK-NUMBER COUNTER
  OXO OXX   OOXOOOXOOXOX             ADD INPUT      /AND FORM INPUT INSTRUCTION
  OXO XOO   OXXOOOXOOOOX             STORE IN1      /PUT IT WHERE IT BELONGS.
  OXO XOX   OOXOOOOOOXXO             ADD ONE        /FORM OTHER INPUT INSTRUCTION
  OXO XXO   OXXOOOXOOXOO             STORE IN2      /PUT IT WHERE IT BELONGS.
  OXO XXX   OXXOOOXOXXXO             STORE IN2P1    /HERE TOO.
  OXX OOO   XXXXXOOOOOOO             CLEAR
  OXX OOX   OOXOOOXOOXXX             ADD N          /GET COUNTER AGAIN.
  OXX OXO   OOXOOOXOOXXO             ADD OUTPUT     /AND FORM OUTPUT INSTRUCTION
  OXX OXX   OXXOOOXOXXOO             STORE OUT1     /PUT IT HERE WHERE IT BELONGS
  OXX XOO   OXXOOOXXOOOO             STORE OUT1P1   /AND HERE.
  OXX XOX   OXXOOOXXXXOX             STORE OUT1P2   /HERE TOO.
  OXX XXO   OOXOOOXOOXXO             ADD ONE        /FORM OTHER OUTPUT INSTRUCTION
  OXX XXX   OXXOOOXOOOXX             STORE OUT2     /PUT IT WHERE IT BELONGS.
  XOO OOO   OXXOOOXOOOOO             STORE OUT2P1   /HERE TOO.
  XOO OOX   OOOOOOOOOOOO   IN1,0                    /BECOMES "INPUT N"
  XOO OXO   OOOOXXOOXOOX             TEST NINE      /IS HOUR DIGIT A 9?
  XOO OXX   XOXOOOXOOXOX             JUMP FAST      /NO. TEST AGAIN
  XOO XOO   XOXOOOXXXXOO             JUMP ADJHR     /YES. GO FLIP 10-HOUR DIGIT
  XOO XOX   OOOOXXOXXOOO   FAST,     TEST THREE     /IS HOUR DIGIT A 3?
  XOO XXO   XOXOOOXOXXXX             JUMP INCHR     /NO. GO INCREMENT HOUR.
  XOO XXX   OOOOOOOOOOOO   IN2,0                    /BECOMES "INPUT N+1."
  XOX OOO   OOOOXXOOXXXX             TEST TWO       /IS TEN-HOUR COUNTER A TWO?
  XOX OOX   XOXOOOXOXXXX             JUMP INCHR     /NO, INCREMENT HOUR NORMALLY
  XOX OXO   XXXXXOOOOOOO             CLEAR          /YES; IT WAS 23:59, SO SET
  XOX OXX   OOOOOOOOOOOO   OUT2,0                   /TIME TO 01:00. "OUTPUT N+1" IS HERE
  XOX XOO   OOXOOOXOOXXO             ADD ONE        /SET AC TO 1.
  XOX XOX   OOOOOOOOOOOO   OUT1,0                   /AND "OUTPUT N" HERE.
  XOX XXO   XOXOOXOOOOOX             JUMP INCN      /GO INCREMENT CLOCK-NUMBER COUNTER
  XOX XXX   OOXOOOXOOXXO   INCHR,    ADD ONE        /ADD 1 TO HOUR
  XXO OOO   OOOOOOOOOOOO   OUT1P1,0                 /BECOMES "OUTPUT N".
  XXO OOX   XOXOOXOOOOOX             JUMP INCN      /GO INCREMENT CLOCK-NUMBER COUNTER
  XXO OXO   OOXOOOXOOXXO   ADMIN,    ADD ONE        /ADD 1 TO MINUTE DIGIT
  XXO OXX   XXOOXOOOOOOX             OUTPUT 1       /AND PUT IT
  XXO XOO   XXOOXOOOOXOX             OUTPUT 5       /IN ALL
  XXO XOX   XXOOXOOOXOOX             OUTPUT 9       /THE MINUTE DIGITS.
  XXO XXO   XOXOOOOOOOOX             JUMP CHKCL     /THEN GO BACK TO CLOCK-WATCHING.
  XXO XXX   OOXOOOXOOXXO   ADSTEN,   ADD ONE        /ADD 1 TO TEN-MINUTE DIGIT
  XXX OOO   XXOOXOOOOOXO             OUTPUT 2       /AND PUT IT
  XXX OOX   XXOOXOOOOXXO             OUTPUT 6       /IN ALL
  XXX OXO   XXOOXOOOXOXO             OUTPUT 10      /THE TEN-MINUTE DIGITS.
  XXX OXX   XOXOOOOOOOOX             JUMP CHKCL     /THEN GO BACK TO CLOCK-WATCHING
  XXX XOO   XXXXXOOOOOOO   ADJHR,    CLEAR          /FIRST CLEAR
  XXX XOX   OOOOOOOOOOOO   OUT1P2,0                 /HOUR DIGIT (BECOMES "OUTPUT N")
  XXX XXO   OOOOOOOOOOOO   IN2P1,0                  /THEN GET TEN-HOUR DIGIT
  XXX XXX   OOXOOOXOOXXO             ADD ONE        /AND ADD 1 TO IT.
X OOO OOO   OOOOOOOOOOOO   OUT2P1,0                 /BECOMES "OUTPUT N+1".
X OOO OOX   XXXXXOOOOOOO   INCN,     CLEAR          /ROUTINE TO GET NEXT CLOCK NUMBER.
X OOO OXO   OOXOOOXOXOXX             ADD N          /ADDING FOUR TO CLOCK NUMBER
X OOO OXX   OOXOOOXOXOOX             ADD FOUR       /TAKES US TO NEXT CLOCK.
X OOO XOO   OOOOXXOOXOOO             TEST FTEEN     /HAVE WE RUN OUT OF CLOCKS (N=15)?
X OOO XOX   XOXOOOXOXOXX             JUMP STORN     /NO, GO STORE N AND RETURN
X OOO XXO   XXXXXOOOOOOO             CLEAR          /YES, SET
X OOO XXX   OOXOOOXOXOXX             ADD 0          /N=3
X OOX OOO   OOXOOOXOXOOO             ADD THREE      /AND RETURN
X OOX OOX   OXXOOOXOXOXX             STORE N        /TO START OF PROGRAM
X OOX OXO   XOXOOOOOOOOX             JUMP CHKCL     /(WE'VE DONE CHECKING CLOCKS).
X OOX OXX   OXXOOOXOXOXX   STORN,    STORE N        /STORE NEW CLOCK-NUMBER COUNTER
X OOX XOO   XOXOOOOOXOXO             JUMP ROUND     /AND SERVICE NEXT CLOCK.  END OF MAIN PROGRAM
X OOX XOX   OOOOOOOOOOOO   ZERO, 0                  /    THESE ARE CONSTANTS.
X OOX XXO   OOOOOOOOOOOX   ONE, 1
X OOX XXX   OOOOOOOOOOXO   TWO, 2
X OXO OOO   OOOOOOOOOOXX   THREE, 3
X OXO OOX   OOOOOOOOOXOO   FOUR, 4
X OXO OXO   OOOOOOOOOXOX   FIVE, 5
X OXO OXX   OOOOOOOOXOOX   NINE, 9
X OXO XOO   OOOOOOOOXXXX   FTEEN, 15
X OXO XOX   XXOOOOOOOOOO   INPUT, 6000              /RAW INPUT INSTRUCTION.  (OCTAL)
X OXO XXO   XXOOXOOOOOOO   OUTPUT, 6000             /RAW OUTPUT INSTRUCTION.  (OCTAL)
X OXO XXX   OOOOOOOOOOOO   N, 0                     /COUNTER FOR WHICH CLOCK WE'RE ON.
                 :
```

IF THIS LOOKS FORMIDABLE, TRY OVER HERE.

# Thank God for THE ASSEMBLER

Ten minutes after starting to program in Machine Language you will probably want Assembly Language.

It's a pain trying to get all the ones and zeroes right. (Xes and Os in the example. Same thing.)

It's a pain trying to keep track of binary numbers for where things are stored.

SO: let's give them alphabetical names. That's assembly language. (And the conversion program we put our alphabeticals into, to turn them back into the binary patterns that really run the machine-- that conversion program is called the Assembler.)

An assembler is a direct and non-tricky translator, intended mainly to handle the details of exact transposition between instruction code-words and the exactly corresponding machine-language program that you intend.

IT WORKS LIKE THIS: The assembler scans through the assembly-language program, testing the successive alphabetical characters. After finding the key punctuation marks or delimiters (shown as commas and slash for the PDP8 assembler), it scans for the alphabetical instruction mnemonics, and translates them by a table in core memory into the corresponding binary codes. (It ignores everything on a line after a slash /, which is lucky, since in the comments you may use words which are the same as instruction mnemonics.)

The assembler also counts the instructions, and (starting wherever you say) figures where in core memory the instructions (and any data or spaces you put in) go. Then it makes a list of these addresses, called a symbol table (also called a name list at less elegant places).

An assembler is the simplest form of compiler (see p. 30). Basically it translates an assembly-language program, which cannot be run directly, into a binary program which can.

Then from this symbol table it fills the resulting binary addresses into the binary commands of the program.

Aren't you glad you don't have to?

Generally the assembler then sends out the binary program to some external device, such as a disk memory or paper tape punch. Then it can be put into core memory when you want to run it.

(You can put a program into core memory one bit at a time through the front-panel switches but nobody likes doing this except for teeny programs.)

(Note: an assembler for one computer (say the PDP-8) that runs on a different computer (say, the 360) is called a cross assembler.)

NOW YOU SEE WHY WE USE HIGHER COMPUTER LANGUAGES.

Most people don't like this stuff.

"Assembly language programming is good for the soul."

Folk saying

# the MINI

© Walt Disney Productions

*This is a PDP-11, one of the world's best-designed minicomputers (see p. 42.). The PDP-11 is a 16-bit machine. Shown is Model 45, the fastest PDP-11, which has various special features. Stripped, with 4K of core memory (that's 4096 locations), it costs about $18 grand. A smaller PDP-11 goes for some $6000.*

A minicomputer simply means a small computer, no different in principle from the big ones (see next spread), and it can do all the same things except as limited by speed and memory capacity.

(Mind, we are talking about real computers, not the little calculators you hold in your hand that just do arithmetic. A real computer is one which works on stored programs and all kinds of data, working not merely on numbers but on such other things as text, music and pictures if supplied with appropriate programs; see flip side.)

There is some argument over what constitutes a minicomputer; basically we will say it's any computer with a word length of 18 bits or less (see "Binary Patterns," p. 27). (Some companies, like Datacraft and Interdata, are trying to peddle their worthy computers as "minicomputers" even though they're 24 and 32 bits, respectively, but that's very odd. Interdata says any computer under ten thousand is a mini-- which means all computers will be minis by and by; a vexing thing to do to the term.)

Traditionally minicomputers come with much less. In the old days pretty much all the programs you got with it were an assembler (see p. 35) and a debugger (see p. 30) and a Fortran compiler (see p. 31) if you were lucky. Today, though, with minis having highly built-up software like (see pp. 40-42 for descriptions) the PDP-8, the PDP-11 and the Nova, you can get a lot of different assemblers, together with Fortran, BASIC, and a little disk or cassette operating system (see p. 45) to make your life a little easier.

The idea of owning a computer may seem strange to some people, but with prices falling as they are it makes perfect sense. Numerous individuals own minis, and as the price continues to drop the number will shoot up. For several families with children to pool together and buy one for the kids makes a lot of sense. One friend of mine has an 8, another is contemplating an 11. (I've been trying to get my own for years; perhaps this book...) Anyhow, the general price range is now $3000 to $6000 plus accessories, and that's dropping fast. Rental is usually a great mistake: prices are very high and after six months or so you'll have paid for it without owning it. (But names of rental places will be found in this book, and some of them may offer good arrangements.) Minis may now be had in quantity for $1000 each-- price of the PDP-8A in May 1974-- and soon that will be the consumer price.

Unfortunately, the price of the computer itself is dropping faster than that of the accessories, such as the basic terminal you'll need, which still weighs in at $1000-5000. Moreover, as soon as you want to do anything serious you'll need a disk (starting around $4500) or at least a cassette memory (starting around $1500). But these prices too will come way down as the consumer market opens.

Some of us minicomputer freaks see little real need for big computers. Minicomputers are splendid for interactive and "good-guy" systems (see p. 15); as personal machines, to handle typing and bookkeeping; even for business systems, if you recognize the value of working out your own in BASIC or, say, TRAC Language.

Minicomputers are being put inside all manner of other equipment to handle complex control. (However, for repetitive simple tasks, the latest thing is microprocessors (see p. 44), which cost less but are harder to program.)

Minicomputers are now being found in highschools; active marketing to highschools is now being done by both DEC and Hewlett-Packard.

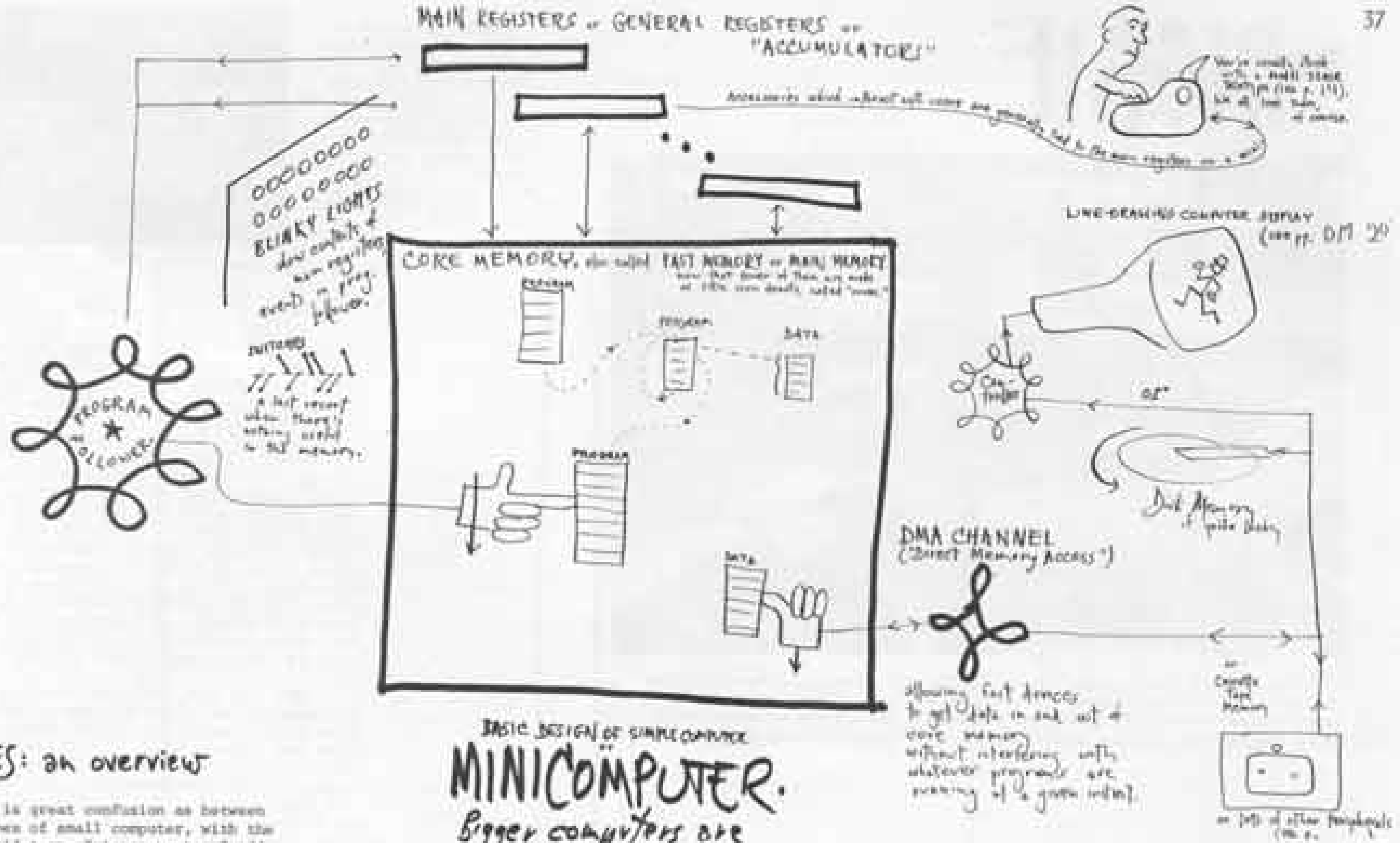
Children's museums in Brooklyn and Boston have recently obtained PDP-11s for the kids to interact with. In the Brooklyn case, the computer will even demonstrate the exhibit and help the child discover things about it, in ways worked out by Gordon Pask (see p. DM 15).

In the future, networks of minis may be the systems to offer low-cost information services to the home (for speculations, see p. DM 57).
But minis will also start to make bigger and bigger incursions on the territory of the big machines. For instance, one group proposes a time-sharing system which will simply consist of Novas interconnected in a ring, the so-called STAR-RING, which will supposedly compete with big time-sharing.

*Here's that selfsame PDP-11 in its overall setting. With peripherals shown, plus the magnificent Vector General display (shown later on in book, p. DM 31 & elsewhere), this setup cost well over a hundred grand. (This is the Circle Graphics Habitat, otherwise known as the Chemistry Department Computer, U. Illinois at Chicago Circle. Why do chemists need such things? See p. DM 31.)*

*The good ol' PDP-8, perhaps the most popular minicomputer (12 bits). Full PDP-8e now cost about $3000, "kits" less. Shown here with a Sykes cassette tape deck-- a nice, rather reliable unit-- and a screen display (see pp. DM 22-3). Courtesy Princeton University & R.E.S.I.S.T.O.R.S. (see p. 47)*

*Kids love computers. They belong together. This lad flips panel switches on a Nova, perhaps the third most popular mini after the 8 and 11 (16 bits; see p. 41).*




BASIC DESIGN OF SIMPLE COMPUTER

# MINICOMPUTER.

*Bigger computers are the same but more so.*

## DINKIES: an overview

There is great confusion as between various types of small computer, with the latest stupid term, "microcomputer," adding to the confusion. We have:

<u>minicomputer</u> or <u>mini</u>
: Traditionally, any computer having an architecture (memory and main registers) of 16 bits or less. Lately, unfortunately, some people have been advertising their 24-bit and even 32-bit computers as minis. This is just confusing.
(They base this on the fact that "minicomputer" has also referred to a machine sold without a lot of programs. But that's really a separate issue.)

<u>microprocessor</u>
: Two-level computer (see p. 44).

<u>microcomputer</u>
: Crummy term apparently being used to mean any tiny computer, regardless of its structure. Thus all computers will be "microcomputers" in a few years. This clarifies nothing as to their structure or use.

<u>midi computer</u>
: Remember midi skirts? Well, this term has been used for computers larger than 16 bits or faster than usual, by people seeking to give the impression that their machines are bigger than minis and less than biggies. Even the PDP-10 (a genuine biggie) has sometimes been called a midi.

A product called Cling Free -- comes scented in a spray can, for preventing static in your laundry-- is said to eliminate static electricity in carpeted computer rooms. Spray it all over the rug, especially near the computer, and you won't zapp the computer with sparks from your fingers.

**HEY, SOME MINI RENTALS MAY BE REASONABLE**

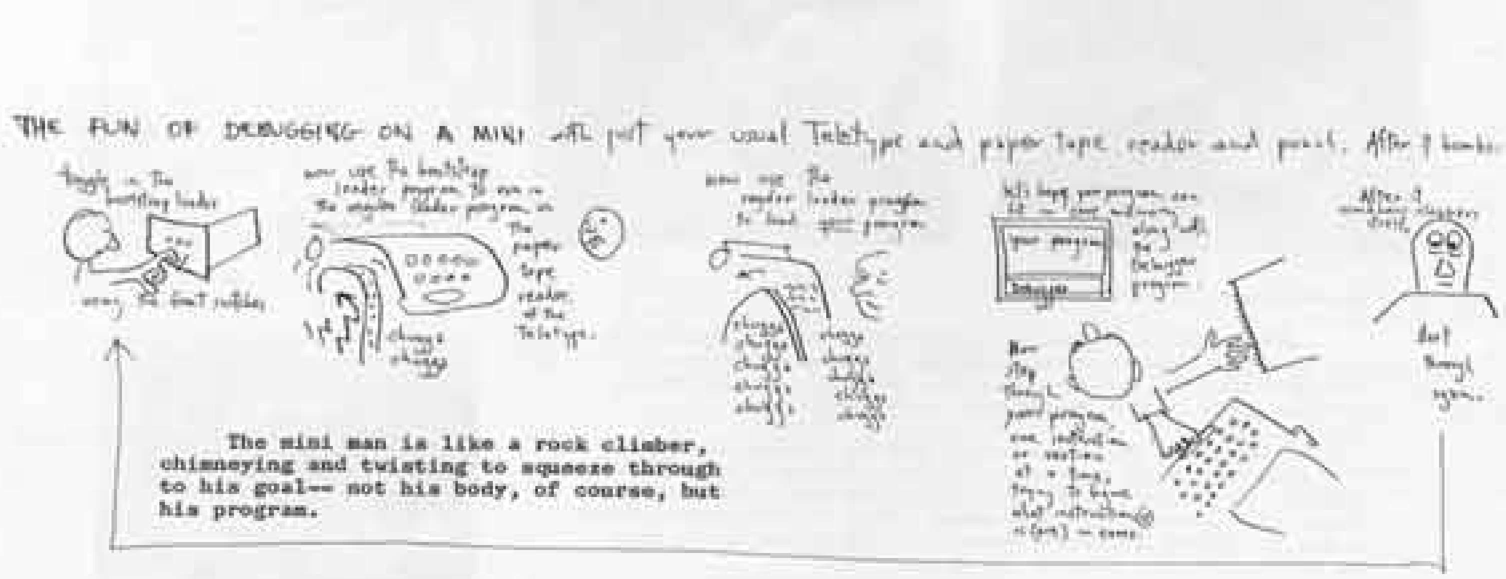
Nova minicomputers are leasable from:

Rental Electronics, Inc.
(a subsidiary of Pepsico)
99 Hartwell Ave.
Lexington, MA 02173

for as little as $250/mo., <u>long-term</u>.

## WHERE TO GET 'EM

*A long but incomplete list of minicomputer manufacturers is at the bottom of p. 43.*




**The mini man is like a rock climber, chimneying and twisting to squeeze through to his goal-- not his body, of course, but his program.**

# THE BIGGIE

Operator's console of this particular setup. The operator may use the keyboard or light-pen (see p. DM 25) to select among waiting programs, submitted by various programmers and departments.

© Walt Disney Prod.

The operator muses at the console of the main computer at the University of Illinois at Chicago Circle. It is an IBM 370 model 158, which rents for about $50,000 a month, including all accessories and a dozen or so terminals -- in the parlance of big-computer people, a "medium-sized installation."

This is a big computer.

In principle it's no different from a small one; but it has bigger memories, more registers, more program followers. There are more specialized parts and more things happening at once. (Thus the term "digital computer complex" is sometimes used for a big computer.) It comes supplied with a monitor program or operating system (see p. 45) and a variety of other utility programs and language processors.

Biggies have many ominous and seemingly incomprehensible things to scare the layman.

For one thing, <u>where is the computer</u>? All you see is a lot of roaring cabinets. <u>Which is it?</u>

Answer: all of them. "The computer" is divided among the different cabinets (note diagram and cluster of pictures locating the operator among them, below). The external devices or peripherals (see p. 57) are usually in separate housings. Usually there is one single box or "mainframe" containing core memory, main registers, program-following circuitry, etc., as in the machine illustrated, but these things don't <u>have</u> to be in one box, and sometimes aren't.

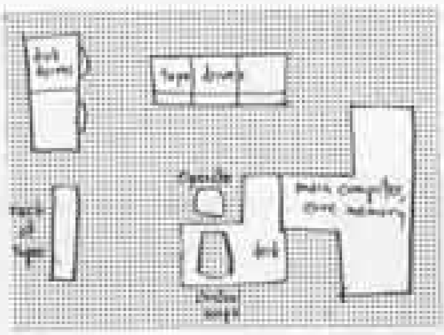
The parts of a computer are set up to be <u>gotten at</u>, to be refilled and repaired. Their innards swing open like refrigerators. Similarly, the wiring of computers is in separate sections or modules ("module" merely being today's stylish term for "unit"), having very orderly connections among them. Individual circuits are on circuit sheets or "cards" which plug in sideways and may be replaced easily. There's nothing really computerish about this, it's merely sensible construction; but it is traditional in other fields to build something as a tangle of wires. (When TV makers follow these rational practices, they call it "space age construction.")

Why are the different parts so far apart? So there's room to swing them open, refill or change them, sit down and repair them. Refrigerators could, and perhaps should, also be built in separate sections, but it's not traditional. Automobiles can't be spread out because they have to endure the jostles of the road. But computers like this baby aren't going anywhere.

Also intimidating is the fact that you have to <u>step up</u> as you enter a computer room. That's because computer rooms ordinarily have raised floors, permitting cables to be run around among the pieces of equipment without your tripping.

Computer rooms are generally lit by millions of fluorescent bulbs, making them garishly bright. This is simply tradition.

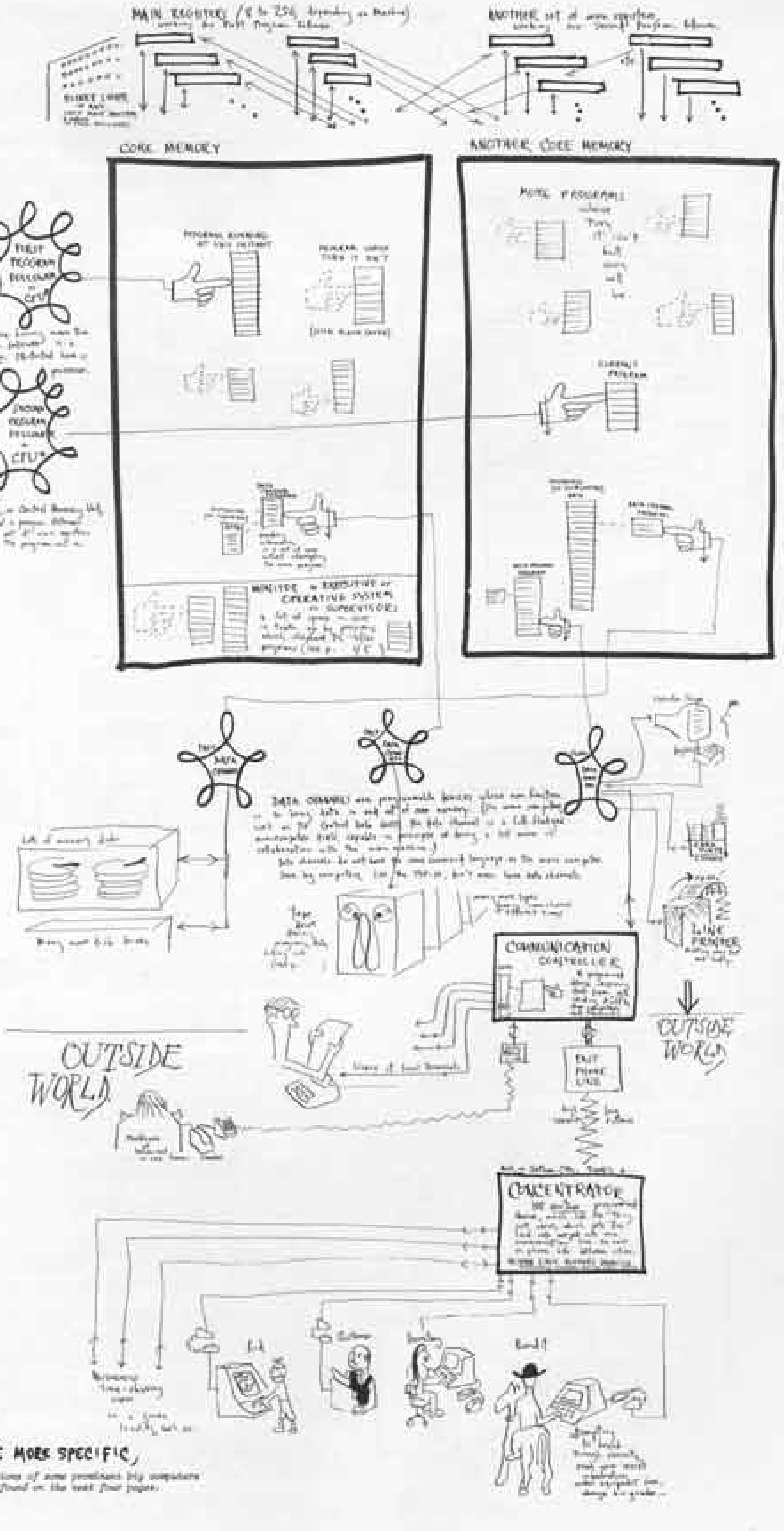
Big computers can have millions of words of core memory. Moreover, there are usually several disk drives and tape drives, as seen in the pictures, used to hold data and programs. (Some of the programs are the system programs, especially the language processors and the operating system-- see p. 45-- but other programs and most of the data belong to the users.)




AN OPERATOR IS NOT A PROGRAMMER

Cindy Woelfer is the day-shift operator of Circle's big computer. The job mainly consists of changing disks and tapes, starting and stopping different jobs listed on the scope, and restarting the computer when the system crashes (gratuitously ceases operation).

Ms. Woelfer, a thoughtful person, says she does not find her job very stimulating. She <u>can</u> program, but the job doesn't involve programming. It's also a lonely job. Non-systems people, except Mayor Daley, aren't ordinarily allowed around. About the only people to talk to are the systems programmers who stop through to look at the scope and see whether their programs are up next.




**TO BE MORE SPECIFIC,**

*descriptions of some prominent big computers will be found on the next four pages.*

It used to be traditional for machines like this to have many many rows of blinking lights, showing what was in all the main registers at any fraction of a second. But there's really no point in seeing all that, since about all you can tell from it is whether the computer is going or not (if it's not, the lights are stopped) and other high-level impressions. For that reason some big computers, beginning with the CDC 6600, started doing away with the fancy lights and bringing written messages to the operator on a CRT scope instead (for lots more on the glories of CRTs, see the flip side, pp. DM22.

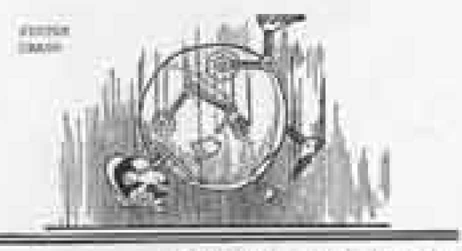
Big computers can have multiple program followers and sets of registers (a program follower and its main registers are together called a CPU, Central Processing Unit). A computer with two CPUs, i.e., two sets of program followers and registers to carry the programs out, is called a dual processor; a computer with more than two CPUs is called a multi-processor.

Separate independent sections of core memory may be put in one computer, allowing separate program followers and data channels to work at the same time. (Note: a "bank" of core memory is an independent section. Except in this sense of "core memory bank" or "core bank," there is no other correct usage of the layman's vague term "memory bank." Computer people only say "memories," and distinguish further among core, disk, tape, etc. Note that "data banks" are a separate issue-- see "Issues," p. 58.)

DINOSAURS?

Many computer people, the author included, entertain certain doubts about the long-term usefulness of big computers, since minicomputers are cheaper, especially in the long run, and can actually be in the offices and homes where people create and use the information. Big computers are necessary for time-sharing (see p. 45) and huge "number-crunching" jobs (see "Grosch's Law," nearby). However, it will soon be cheaper to put standardized number-crunching jobs in stand-alone or accessory hardware; see "Microprocessors," p. 44.

Fans of big computers also argue that they are necessary for business programming, but that only means traditional business programming-- non-interactive and batch-oriented. For tomorrow's friendly and clear business systems, networks of minis may be preferable. But makers of big computers may be unwilling to admit this possibility.




Tends to happen several times a day.

## GROSCH'S LAW

Minicomputers are so nifty that we may ask why have big computers at all. The answer is that there are considerable economies, especially in applications that require many repetitive operations and don't need interaction with users.

A hypothesis about the economy of big computers was formulated a long time ago by Herbert J.R. Grosch, onetime director of IBM's Watson Lab and now a heavy detractor of IBM. Thus it is called Grosch's Law. The idea is basically that there is a square-law relationship between a machine's size and its power (narrowly defined in terms of the cost of millions of operations, and without considering the advantages of interactive systems or other features which may be of more ultimate value). Anyway, when I asked him recently for his formulation of Grosch's Law, I got the following:

"Grosch's Law (formal): Economy in computing is as the square root of the speed.

(informal): If you want to do it ten times as cheap, you have to do it a hundred times as fast.

(interpretive): No matter how clever the hardware boys are, the software boys piss it away!"

# SOME GREAT COMPUTERS

Here, then, are some thumbnail descriptions of some great, classic or popular computers, expanding our basic diagrams as needed.

Individual computers represent variations of the patterns shown so far.

The particular structure of registers, memories and pathways among them is called the architecture of a computer (see p. 52). The binary instructions available to the programmer are called the instruction-set of the particular computer (see p. 33). (The word "architecture" is often used to cover both, including the instruction-set as well.)

The principal variations among computers are the word length (in bits-- see "binary patterns," p. 33) and the number and arrangement of main registers. Then come the details of the instruction-set, especially the ways in which items are selected from core memory -- the addressing structure. Then the instruction-set, whose complications and subtleties can be considerable indeed.

The individual computer is the complex result of all of these. If they fit together well, it is a good design. If they fit together poorly, it is a bad design. A bad design is usually not so much a matter of overt stinky features as of ramifications which fit together disappointingly. (Glitch is a term often used for such stinky features or relationships.)

The possible ways of organizing computing hardware are vast, and only partly explored. (An aside to computer guys: on the Intel chip debugging consoles they have an address trap (trapping on a presettable effective address) and a pass counter (trapping after n passes). How come we haven't seen these sooner?)

The machines mentioned here are an arbitrary selection. Some of them are the Great Numbers, computers so important that folks use their numbers as proper nouns, with no brand name:

"Do you have a 360 up there?"

"No, but there's a 6600, a 10 and a bunch of 8s."

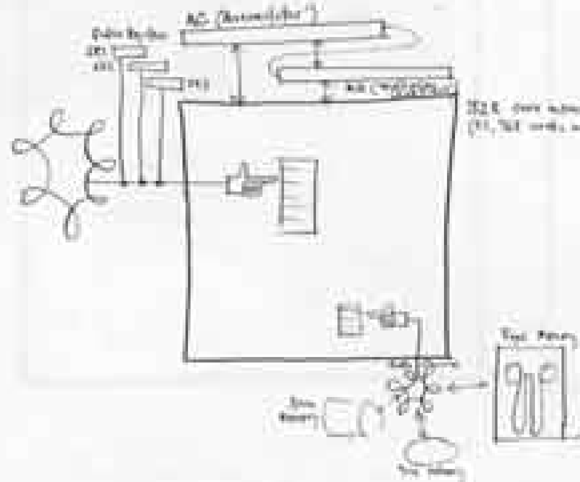
"Personally, I'd rather work on a 5500."

Here is what they are talking about.

## the 90/94 (36 bits)

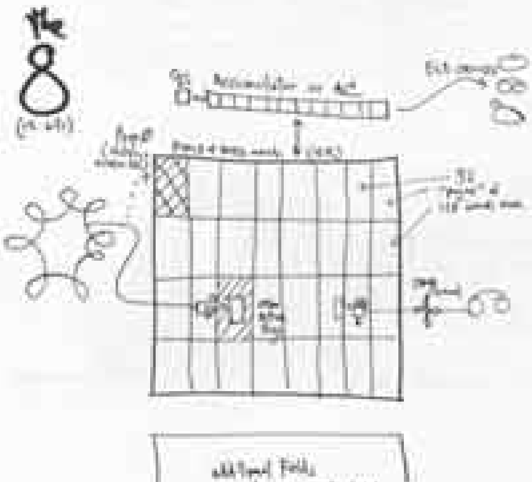
**The IBM 7090 was the classic computer. Introduced about 1960 and mostly gone by '66, it was simple and powerful, with clean and decent instructions. With its daughter the 7094, it became virtually standard at universities, research institutions and scientific establishments. At many installations they went on to 360s they long for those clearminded days.**

**The 90 had three index registers and fifteen bits to specify core addresses. (This meant, of course, that core memory could ordinarily be no longer than 32,768 words ("32K"-- see "Binary patterns," p. 33).) A later model, the 94, went up to 7 index registers, since there were three bits to select them with.**



**Though those were million-dollar machines ten years ago, you now hear of them being offered free to anyone who'll cart them away; partly because they needed a lot of power, airconditioning and soo on. But they were great number crunchers. (If you want a 90, I believe that 90 lookalikes are still available from Standard Machines in California.)**

## the 8 (12 bits)



**The PDP-8 was designed by Gordon Bell (in its original version, the PDP-5) about 1960. Originally it cost about $25,000; as of May 1974 that price is down to about $3000, or less than a thousand dollars if you want to buy the circuits and wire it all up yourself. Yup, here comes that Heathkit.**

**The PDP-8 has been DEC's hottest seller; you'll find them in industrial plants and museums, or even hidden in the weirdest equipment, from typesetting devices to big disk drives. At universities all over there are kids who know them inside out.**

**Today the PDP-8 seems archaic, with its one accumulator and awkward addressing schemes: you can only get to 256 different addresses in core memory directly, and it's chopped up into pages. But for its time it was a brilliant design, packed like a parachute, and even today there are people who swear by it. (But look at what Bell's done lately: the PDP-11.)**

**So many programs exist for the PDP-8, though, and so much sentimental fondness, that it will be with us for the foreseeable future. Thus the "Bucky's Wristwatch" example (see p. 31-3) is not totally frivolous: we may assume that a PDP-8 on one or two wristwatch-sized chips is only a year or so away. But let's hope they do the 11 first.**

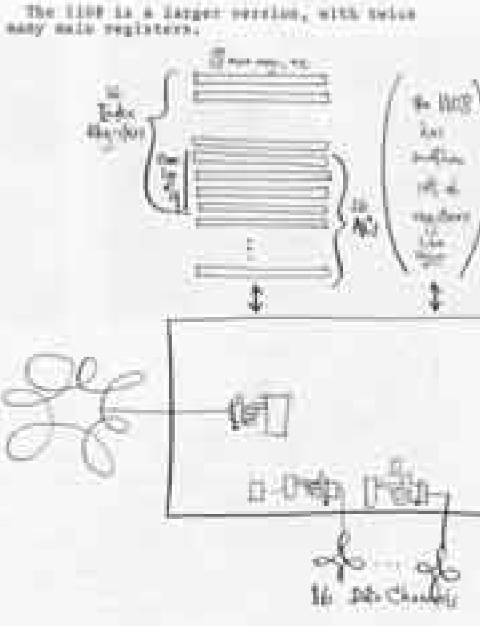
**(Lookalikes available from Digital Computer Controls and Fabri-Tek.)**

## the 1106 (then, or 36 bits) & 1108.

Univac's 1106 and 1108 are fast, highly regarded machines. In designing the computer Univac did a clever thing: they built an upgraded 7094. This meant (as I understand it) that all the programs from the old 7094 will run on it. But instead of two main registers they have 18.

(Where they found the bits in the instruction word to select among all those registers I can't tell you.)

The 1108 is a larger version, with twice as many main registers.



## THE 10, formerly the 6 (36 bits)

DEC's PDP-10 is in some ways the standard scientific computer that the IBM 7094 was in the sixties.

The PDP-10 is excellent for making highly interactive systems, since it can respond to every input character typed by the user.

It is a favorite big computer among research people and the well-informed. The ARPANET, which connects big computers at some of the hottest research establishments, is largely built with PDP-10s. There are PDP-10s at MIT, U. of Utah, Stanford, Yale, Princeton and Engelbart's shop (see p. DM 46). The Watkins Box (see p. DM 53) hooks to a 10.

Digital Equipment Corporation, aware that its computer trademark "PDP" connotes minicomputers to the uninformed, now wants the 10 to be called DECsystem-10 rather than PDP. We'll see if that catches on.

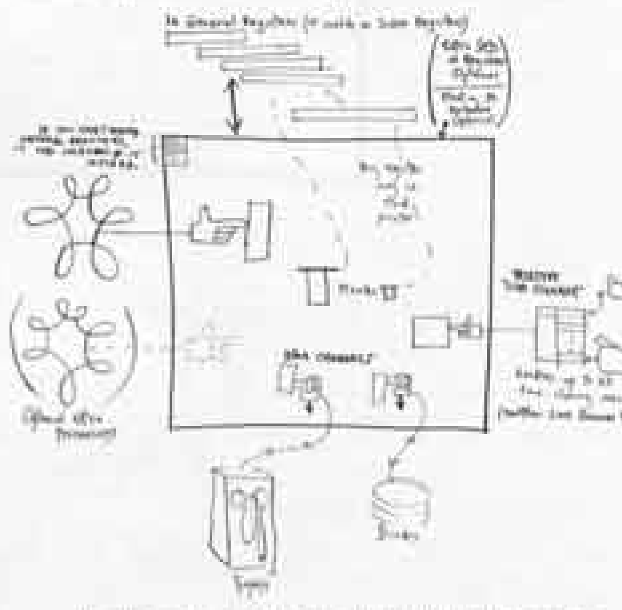
Who designed it is not entirely clear. I've heard people attribute it variously to the Model Railroading Club at MIT, to Gordon Bell, and one Alan Kotok.

Originally it was the PDP-6, which appeared about 1964, and was the first computer to be supplied with a time-sharing system, which worked from the beginning, if feebly. Now it's good and solid. DEC's operating system for it (see p. 45) is called TOPS, but BBN sells one called TENEX, also highly regarded. The 10 does time-sharing, real-time programming and batch processing simultaneously, swapping to changeable areas of core memory. (This feature should soon be available, at last, on IBM computers ("VS2-2").)

PDP-10 time-sharing works even if you don't have a disk, using DECtape (DEC's cute little tapes). Of course, without disk it's really hobbling, but this capacity is nevertheless noteworthy.

The PDP-10 has debugging commands which work under time-sharing and with all languages, and hugely simplify programming.

Unlike the IBM 360, whose hardware protection comes in options, the 10 has seven levels of protection: the user can specify who may read his files, run them, change them, and do four other things. The PDP-10 does have job control commands, but they are not even comparable in cumbersomity to IBM's JCL language (see p. 51), and they are the same for all three modes of operation: time-sharing, real-time and batch.



The PDP-10 has 36 bits but has instructions to operate on chunks, or bytes, of any length. It has sixteen main registers, as does the 360, but uses them more efficiently.

The PDP-10 also has unlimited indirect addressing: an instruction can take its effective address from another location, which can in turn say to take its effective address elsewhere, ad infinitum. For your heavy tight elegant stuff.

Perhaps most important, the 10 has a full set of stack instructions (see "The Magic of the Stack," p. 42), allowing programmers to use multiple stacks for purposes of their own. (The operating system's own stacks are protected.) Programmers do not have to save each other's registers, as on the 360. Programmers are relatively safe from each other.

Some think of the PDP-6 and 10 as a glorified 7094 (with 18 addressing bits, instead of 15). In this case we might consider the 360 a stripped-down version of the 6, since IBM threw out the stack and in most models the memory mapping.

PDP-10s are ordinarily sold where the views of scientists and engineers are considered important, and comptrollers do not have first choice. Nevertheless, some say that its business-programming facilities (i.e., COBOL, duh) are just as good as those of companies who claim to have designed computers "for all purposes." First National City Bank of New York has found that the PDP-10 makes a splendid banking computer for internal use, profitable at an internal charge of $1.75 an hour plus processing charges. Prices for a PDP-10 system with disk start start about $500,000, or $15 grand a month, and go up into the millions.

However, DEC salesmen are not like IBM's, who can reputedly sell Eskimos to iceboxes. For one thing, DEC salesmen are on salary. That fits DEC's demure, aw-shucks image, but it doesn't exactly sell big computers.

(For you Firesign Theater fans, the mutterings of the dying computer on the "Bozos" album are various PDP-10 system thingies, artistically juxtaposed.)

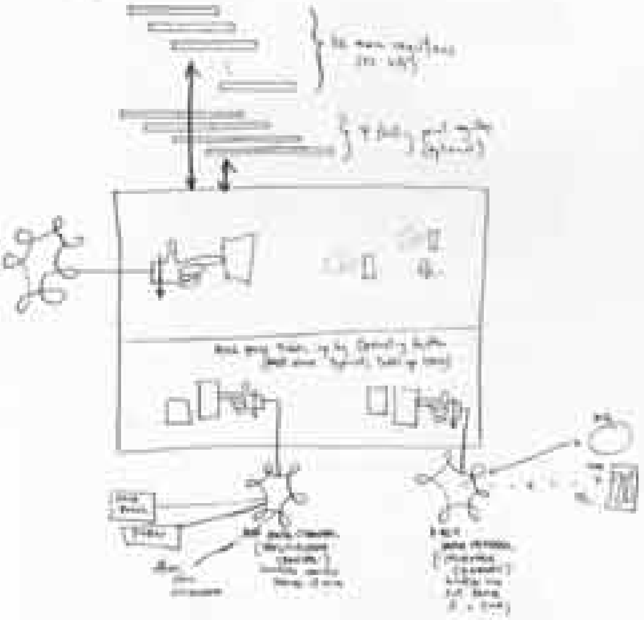
# the 360 & 370 (32 bit, 16 bit, 8 bit, 4 bit)

"No programmer ~~~~~ IBM would sell a computer like this." -- A friend.

The IBM 360 (now called 370 because we're in the 70s) is the commonest and most successful line of computer in the world. This does not necessarily mean it is the best. There are those who appreciate IBM typewriters but not their computers.

360s are bought because the repair service is great; because IBM has very tough salesmen; and possibly for other reasons (see pp. 52-4).

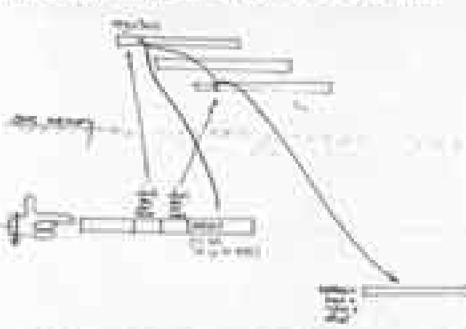
A strange unseen curse seems to haunt the 360 series; indeed, some cynics even think it results from deliberate policies of IBM? Yet the 360 (and its software) seem somehow organized to make programs inefficient and slow; to make programs big, needing lots of core memory (with numerous enticements for the programmer to take up more); to prevent the compatibilities that are so widely advertised, except through expensive options; to make things excessively complicated, thus locking in both its customers and the employees of its customers in practices and intricacies that are somehow unnecessary on other brands of computer.



The design of the 360, which was basically decent, is generally attributed to Amdahl, Blaauw and Brooks. Those who hate it, and there are many, base their complaints largely on the restrictions and complications associated with its operating system OS, which is notoriously inefficient (see p. 51).

The architecture of the 360 was quite similar to the PDP-6 (now the PDP-10), designed about the same time: sixteen main general-purpose registers of over thirty bits, and using the 16 main registers as either accumulators or index registers.

A curious form of addressing was adopted, called "base-register addressing." This had certain advantages for the operating system that was planned, and was thought to be sufficiently powerful that you wouldn't need Indirect Addressing. Two main registers were required, one holding a "base" more or less equal to the program's starting address, and an "index register," whose contents are added to the base to specify an address. Often a third number, or "offset," is added as well.



The idea of this technique is that programs can be "relocatable," operating anywhere in core memory. A few instructions at the beginning of each program can ascertain where it is running from, and establish the Base accordingly.

The basic idea of the 360 seems to have been doped out for multiprogramming, or the simultaneous running of several programs in core, a feature IBM has pushed heavily with this computer.

WHAT'S WRONG WITH THE 360?

The main differences between the 360 and the PDP-6 and 10 represent conscious and legitimate and arguable design decisions. To fans of the PDP-6 and 10, here are the 360's main drawbacks:

NO INDIRECT ADDRESSING. This was because, within the addressing scheme adopted, indirect addresses could not be adjusted automatically. (But it also makes programs more inefficient, thus more profitable to IBM.)

NO STACK. Why? Too expensive, said Amdahl, Blaauw and Brooks in the IBM Systems Journal. Today, they have stacks on $5000 PDP-11s. And it would have saved everybody a lot of money on programming.

NO MEMORY MAPPING (except on certain models). Where the PDP-6's successor, the PDP-10, automatically takes care of redistributing addresses in core to service every program as if it were operating from location zero on up, the 360 left this general problem to local programmers and (on certain levels) to operating systems.

Handling this automatically in the PDP-10's hardware shortens the complications of base-index addressing and makes possible the efficiencies of indirect addressing.

LOOKALIKES

360 lookalikes were sold by RCA and Univac. Now that RCA no longer makes computers, Univac is servicing the ones they made.

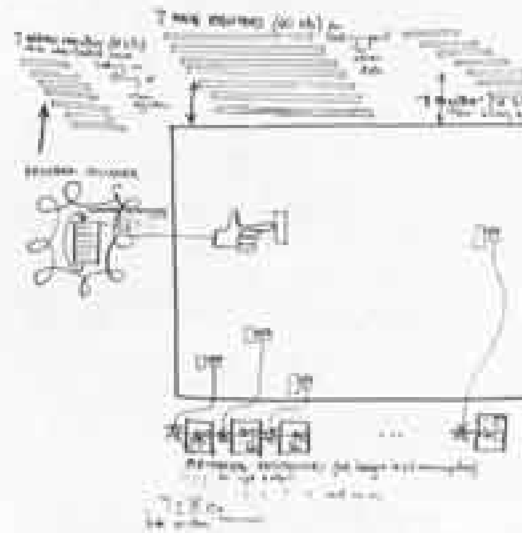
Gad Amdahl, no longer with IBM and now head of the Amdahl Corp., is coming down the pike with a super-360 of his own, in part backed by Japanese money. It will be bigger than IBM's biggest-- and cheaper. (See Hank Wiener, "Outdoing IBM: the Amdahl Challenge," Computer Decisions, March 73, 18-22.)

# THE 6600 ("Sixty-six hundred") (60 bit)

FIRST OF THE SUPERCOMPUTERS. Also 6500, 6800.

Control Data's 6600 computer was the first really big computer. The first one was delivered around 1963. The machine and its operating system, CHIPPEWA, were created by Seymour Cray and his team in hinterland Minnesota.

Extreme speed was designed into the computer in a number of ways. The main computer has no input or output at all; this is handled by data channels which have been built up into full-scale minicomputers or "peripheral processors" of 12 bits.

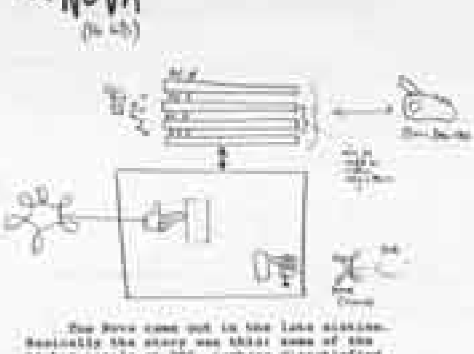


Instructions can be executed at lightning speed, much faster than the usual microsecond or so. However, since core memory is much slower than the main registers, a trick is used: program instructions are drawn from core into a superfast instruction list (often called a cache), and any jumps or loops within this streamlined cache can be executed at unthinkable speeds-- perhaps tens of millions of times per second.

The machine is especially geared for floating-point numbers (see p. 39). Because of the intense speed of the fast instruction cache, many instructions (such as multiplication and division of integers) can be accomplished faster by a short program than if they had actually been wired into the computer.

They 6600 became the start of a whole line, including the 6400, 6800 and others. The 6600 is used by PLATO (see p.DM 28).

# the NOVA (16 bit)



The Nova came out in the late sixties. Basically the story was this: some of the higher people at DEC, perhaps dissatisfied with DEC's soft sell, perhaps out for their own personal share of things, broke out and started their own corporation. They had in hand the design for a hot, solid minicomputer -- some say it was the rejected design for the as-yet-unannounced PDP-11-- and since then they have built it reliable and sold it hard.

The basic design of the Nova is clean and simple: four main registers, no stack, well-designed instructions. Moreover, it was (I think) the first computer to be built around a Grand Bus (see 50), a design which has caught on rather widely.

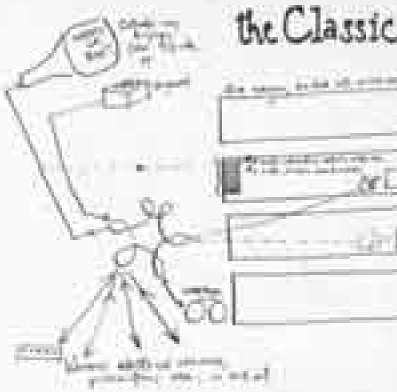
Data General (the company mentioned) has used a very interesting marketing strategy. Instead of bringing out a variety of new computers as time goes on, they concentrate on making the Nova faster and smaller. They began by competing against DEC-- especially in "the OEM market," purchasers who are burying minicomputers in larger equipment they in turn make-- but more recently they have actually started to market against IBM with business systems. In recent months, Data General ads have ridiculed the complexity and mystery of IBM systems, arguing quite rightly that minicomputers programmed in BASIC are a reasonable alternative for a wide variety of business applications.

The Nova's instruction-set is clean and straightforward. Key examples (first bits only):

| | |
|---|---|
| 00000 | Jump (thus an all-zero instruction jumps to loc 0) |
| 0000X | Subroutine Jump |
| 000XX | Increment, skip if zero |
| 00XXX | Decrement, skip if zero |
| 0XX | Load AC |
| 0XX | Store AC |
| 1 | Instructions among registers. |

One competitor, Digital Computer Controls, sells a Nova lookalike. Whether Data General will sell you its programs to run on it is another question.

# the Classic LINC (12 bit)



A computer named the LINC, now usually referred to as "the Classic LINC," was perhaps the first minicomputer. It was an important forerunner of our highly interactive systems of today, notably including today's graphic displays with double program followers (see p. DM 15), which offer the highest interaction capabilities.

Perhaps most importantly, it was designed with none of the biases that creep in from the traditions of business computing.

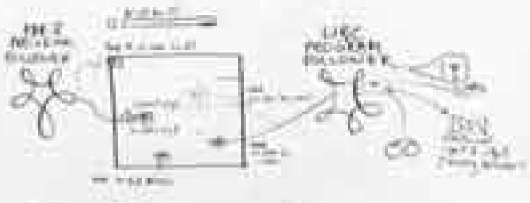
It was called the LINC because it was designed at Lincoln Laboratories (about 1963), for "biomedical research"-- actually it was the sort of computer you'd want for hooking up to all sorts of inputs and outputs, to make music, to run your darkroom, but only medical scientists could afford it, as that's what they said it was for.

The LINC had two interesting innovations. It was probably the first computer to be designed with a built-in CRT display (see flip side). It also came with a funny little tape drive, designed for reliability and high response, that was supposed to perform almost as conveniently as a disk and be reliable even in dusty or messy environments. This was the LINCtape, still offered as an accessory by one company. DEC adapted it somewhat and made it the DECtape, handy pocket tape unit of the PDP computer line.

It was never sold commercially. A dozen or so were made up specially out of DEC modules and dealt out to various scientists, and the general hope was that DEC would take the machine up as part of its product line, but that's not what happened. DEC instead pushed its PDP-8 and gave us instead, by and by,

# the LINC-8

(12 bit, a marriage of the PDP-8 with the LINC)



DEC was offered the option of building Lincoln Laboratories' classic LINC, but decided instead to combine it, in the mid-sixties, with the already-successful PDP-8. That way all the PDP-8 programs and most of the LINC programs would work on it. The result is kind of strange, but very popular in biomedical research: two computers in one, handing control back and forth as needed. You can write programs on the LINC with sections for the 8, and vice versa. Hmm. A more recent and slicker version is called the PDP-12.

While you might half-think that both sides of the computer could work simultaneously, giving you double speed, it doesn't work that way. There's only one core memory, and that sets the basic speed; either a PDP-8 instruction or a LINC instruction can be underway at once, but not both.

Nevertheless, we see here the double structure that plays such an important part in highly interactive computer displays (see p. DM 25). Indeed, LINC programmers often use the machine just that way: the PDP-8 running an actual program, the LINC part running the CRT display in conjunction with it.

A horrifying and weird picture of an experimental monkey sitting on a PDP-12 and making like the Creature from the Black Lagoon is to be seen in Time, 19 Jan 70, p. 54. It looks very scientific.


BIBLIOGRAPHY

The classic book: C. Gordon Bell and Allen Newell, Computer Structures: Readings and Examples. McGraw-Hill, 1971.

Note that Bell designed various of the PDPs, and Newell pioneered in list processing (see p. 26).

Computer Characteristics Review keeps you in touch with the traits of available computers and peripherals. $25/year (3 issues). GML Corp., 594 Marrett Rd., Lexington, MA 02173.

Other firms, such as Auerbach, offer more expensive services of the same nature.

G. Bezier, The Architecture and Engineering of Digital Computer Complexes. Plenum Press, 2 vols., 1971.

Heavier than Bell and Newell. A catalog of thousands of structures and tricks, emphasizing the tradeoffs among them.

# the Very Great 5000

& 5500
(B5? things, obscure)

I have heard no computer more widely praised among computer people than the Burroughs 5000 (replaced by the 5500). The 5000 was designed about 1960 by Edward Glaser and Bob Barton. It was designed to be used only with higher languages, not allowing programmers access to the binary instructions themselves. Indeed, it was particularly designed to be used with ALGOL, which would have been the standard language if IBM had allowed it (see p. 31 ) and is still the "international" language.

Because of this approach, its main registers were to be hidden from the programmer, and attention centered instead upon the stack, a high-level programming device (see box on Stacks). However, index registers were added to make it better for Fortran.

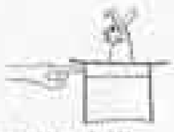
The 5000 was marketed as an "all-purpose" computer with an operating system, anticipating IBM's 360 of a few years later. Indeed, after the 360 was announced, Burroughs sales picked up, because IBM salesmen were at last promoting the concepts that customers hadn't understood when they heard about them from Burroughs salesmen years before.

Bigger machines in the line are now the 6500, 6700...

The Burroughs Corporation continues to be an acknowledged leader in computer design. Apparently their sales force is something else, unfortunately. I once spent some time with a Burroughs salesman who not only knew nothing about the magnificent structure of the machine he represented, but would not get me further information unless I demonstrated that the company I represented (a large corporation) was seriously interested. He wore very fancy clothes.

EVERYTHING IS DEEPLY INTERTWINGLED.

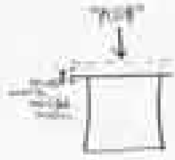
# THE MAGIC OF THE STACK



The Stack is a mechanism-- either built into the computer ("hardware") or incorporated in a program ("software") which allows a computer to keep track of a vast number of different activities, interruptions and complications at the same time.

Basically, it is a mechanism which allows a program to throw something over its shoulder in order to do something else, then reach back over its shoulder to get back what it was previously working on. But no matter how many things it throws over its shoulder, everything stays orderly and continues to work smoothly, till it has resumed everything and finished them all.

It goes like this: if the program has to set aside one thing, it puts that one thing in core memory at a place specified by a number called a stack pointer. Then it adds one to the stack pointer, to be ready in case something else has to go on the stack. This is called a PUSH.

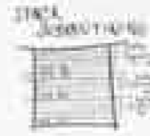



When a program is ready to resume a previous activity, it subtracts one from the stack pointer and fetches whatever that stack pointer points to. This is called a POP.

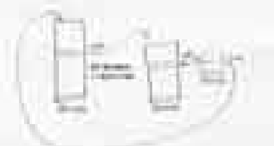

It may not be immediately obvious, but this trick has enormous power. For instance, we may stack any number of things together-- the addresses of programs, data we are moving between programs, intermediate results, and codes that show what the computer was doing previously.

Using stacks, programs may use each other very freely. It is possible, for instance, to jump among subroutines-- independent little programs-- arbitrarily, using a stack to keep track of where you've been.

In this case the stack holds the previous locations and intermediate data, so that the program followed can go back where it came from at the end of each subroutine.




This even makes possible "re-entrant" programs, meaning subroutines that can be used simultaneously by different programs without mixup, and "recursive" programs, meaning programs that manage to call themselves when they themselves are in progress.

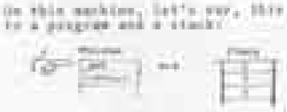


Stacks are also used for handling "interrupts" -- signals from outside that require the computer to set aside one job for another. Having a built-in hardware stack enables the interrupts to pile up without confusion:

Finally, stack arithmetic, like that done on the Burroughs 5000, enables arithmetic (and other algebraic types of activity) to be handled without setting aside registers or space in core memory. As a simple-minded example on a hypothetical machine, suppose we wanted to handle

A + B + C

On this machine, let's say, this gets compiled to a program and a stack:



Then the operations are carried out on the stack itself:

Stack programming tends to be efficient, particularly in its use of core memory.

Some languages, such as Algol and TRAC Language, require stacks.

Some computer companies, such as IBM, resolutely ignore stack architecture, though hardware stacks have become widely adopted in the field.

# THE GRAND BUS

In electronics, a "bus" is a common connector that supplies power or signals to and from several destinations. In computers, a "bus" is a common connection among several points, using carrying a complex parallel signal.

The Grand Bus, a new idea among computers, is catching on. (The term is used here because the colloquial term, "Unibus," is a DEC trademark.)

Basically the Grand Bus is a connector of multiple wires that goes among several pieces of equipment. So far that's just a bus. But a Grand Bus is one that allows the different pieces of equipment to be changed and replaced easily, because signals to any common piece of equipment just go out on the bus.

This means that the interface problem is deeply simplified, because any device with a proper bus interface can simply be plugged onto the bus.

It does mean a lot more complexity of signals. The Unibus, for example, has about fifty parallel strands. But that means various tricky electrical dialogues can rapidly give instructions to devices and consider replies about their status, in quick and standardized ways.

Prominent grand buses include:

The Nova bus (nameless; the first?)

PDP-11's Unibus

Lockheed SUE's Infibus

PDP-8's Omnibus.

The idea is great in general. For your home audio equipment, for instance, Grand Bus architecture would simplify everything.

Not only that, but Detroit is supposedly going to put your car's electrical system on a Grand Bus. This will mean you can tell at once what is and isn't working, and hook up new goodies easily.

# The 11

(a hit)

The PDP-11 is not a beginner's computer. But the power and elegance of its architecture have established it, since its introduction in 1970, as perhaps the foremost small computer in the world.

Actually, though, we can't be too sure about the word "small." Because as successive parts of the line are unveiled, it becomes increasingly clear that this line of "small" computers has been designed to include some very powerful machines and coupling techniques among them, and it would seem that we haven't seen everything yet.

In other words, DEC's PDP-11, which has already cut into sales of their PDP-8 12-bit series and PDP-15 18-bit series, may soon cut into its PDP-10 36-bit series-- as designer Bell unveils (perhaps) monster PDP-11s in arrays or double word-length or whatever.

The PDP-11 was designed by C. Gordon Bell and his associates at Carnegie-Mellon University. In designing the architecture, and especially the instruction-set, they simulated a wide variety of possibilities before the final design was decided. The resulting architecture is extremely efficient and powerful (see box, "The 11's Modes").

Basically it is a 16-bit machine, with most instructions operating on 8-bit data as well.

There are eight main registers. Two, though, function specially: the program counter (that part of the program follower that holds the number of the next instruction), and the hardware stack pointer. Both follow the same programming rules as the main registers-- an unusual technique. Then a jump in the program is simply a "move" instruction, in which the next program address is "moved" into main register R7, the program counter.

In addition, all external devices seem to the program to be stored in core memory. That is, the interface registers of accessories have "addresses" numerically similar to core locations-- so the program just "moves" data, with MOVE instructions, to devices in core. (This is facilitated by the automatic handling of previously bothersome stuff, like Ready, Wait and Done bits.)

Physically all devices are simply attached to a great hank of wires called a Unibus. (See Grand Bus box.)

BIBLIOGRAPHY

A.W. Southern, PDP-11 Programming Fundamentals. (Programmed workbook. No price listed.) Algonquin College Bookstore, 1385 Woodroffe Avenue, Ottawa, Ontario, Canada K2G 1V8.

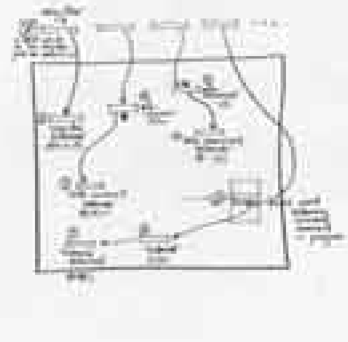
PDP-11 lookalikes are sold by Cal Data. Other firms have been scared off by DEC's patent, but Cal Data say they have a patent too.

# 3rd part of THE 11's MAGIC MODES

Minicomputers are cramped, and so the basic problem in mini architecture is how to cram into the instruction enough choices for getting around in core memory.

In designing the PDP-11, Gordon Bell and his co-workers systematically sought a powerful solution, simulating various possible structures by computer program, trying out a variety of different combinations and structures.

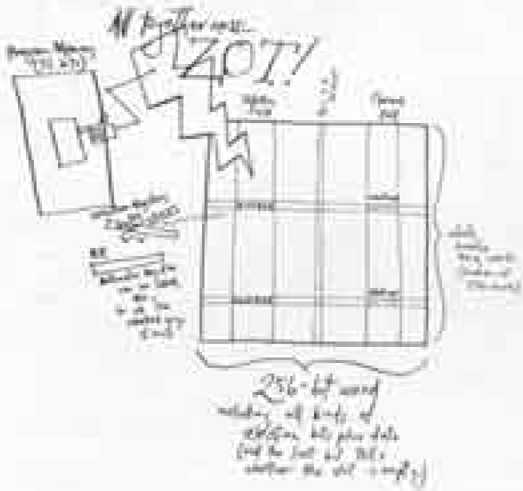
The elegance and power of the solution are little short of breathtaking. Basically the PDP-11, the final design, provides seven different types of indirect addressing. The computer's main registers may be used both to operate on information (the usual technique, here called mode zero), or to point to locations to be operated on (indirect modes 1 through 7). These provide extremely efficient means for stepping through tables, PUSH and POP, dispatch tables, and various other programming techniques. The following diagram is meant for handy reference.



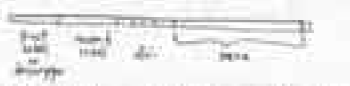
# an Exciting New Weirdie the STARAN

There are a lot of strange computers being designed-- it's a traditional occupation of electronics professors and a great way to soak the Defense Department-- but this one is commercially available. Now if we just knew what to do with it.

Goodyear's STARAN is the first available computer with a Content-Addressable Memory, which is actually very hot stuff. Instead of having to search for a particular item of information in core, or having to make lists of where in core things are being put, or creating linked data structures (see p. 26 ), the program can simply ask all items of data having particular properties to step forward.




It works like this. Having an immense 256-bit word to play with, the programmer uses different parts or "fields" of the word (see p. 28, col.3) to specify what other information is in it:

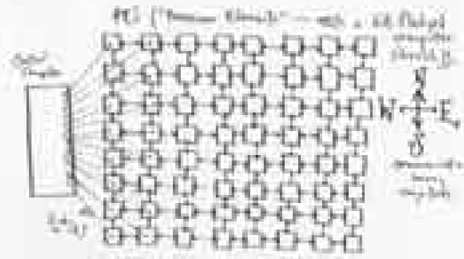


With a single command, the program may ask all words in memory to clear a particular field, or set a particular bit. Then with another command it can tell all memory locations with particular identifiers to add a certain number to their data, and this occurs in a couple of microseconds. Or it can direct all memory locations having particular identifiers to multiply one section of their data by another-- which takes rather longer.

This is an entirely different kind of programming, and considering how much thought computer people have given to doing things one at a time, it kind of sets you back a little. The brochure lists these possible applications: "ballistic missile defense," "intelligence data processing," "electronic warfare," "airborne command and control," as well as more peaceful applications like weather prediction, data management, transportation reservations, air traffic control. Truth is, most computer people would have to scratch their heads quite a while to figure out how to start using this fascinating machine for any of these things; the reason the military applications seem to be so easy is simply that the military computer types have been scratching their heads longer. We might as well start too, and find some of the nicer things to do for humanity with it.

Bibliography: Jack A. Rudolph, "A Production Implementation of an Associative Array Processor-- STARAN," Proc. FJCC 72, 229-241.
Contact: Computer Division Marketing, Goodyear Aerospace Corp. Akron, O. 44315.

# the ILLIAC 4



The Illiac IV is the biggest and most extraordinary computer in the world, knock wood. To most computer people it's as big as anything they want to think about.

The Illiac 4 consists of sixty-four biggish computers, all going at once under the supervision of yet another big computer, typically all working on a single problem. It is the brainchild of Daniel Slotnick, who worked on the theory of array computers and pressed for its creation for years; eventually built by Burroughs, it sits at an airbase but is available to outside users through the ARPA network.

In principle the idea is this: certain classes of problems, especially those involving very large arrays and matrices, can be run only rather slowly on ordinary computers. If, however, a computer is built which itself is an array, certain operations can take place very much faster because they happen in parallel units simultaneously. Matrices, particular formal kinds of array, are used in a great variety of mathematical-type applications. For instance, weather prediction. It seems that the theory of weather prediction has been well worked out for decades, but because the swirly behavior of the atmosphere is so intricate, actually calculating out everything involves billions of operations. At one conference session I believe it was explained that it used to take twenty-five hours to predict the weather twenty-four hours in advance, whi which means you get the answer an hour after it's happened already; now it is possible, using Illiac IV, to do the whole planet's weather in an hour and a half, said the speaker.

Some say that may be its only use and the whole project was inadequately thought out. Others suspect it's really intended as a radar-watcher for the ABM system.

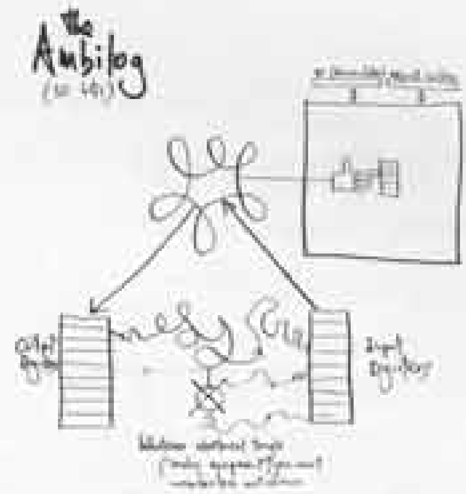
Anyway, there it is. And the individual breadcase-sized Burroughs machines, if they're ever marketed, may provide a new price breakthrough for small highpower systems.

Incidentally, "Illiac" is the traditional name for computers built at the University of Illinois. Will the series end with this one?

BIBLIOGRAPHY

Daniel J. Slotnick, "Unconventional Systems," Proc. SJCC 1967, 477-481.

# the Ambilog (ac 10)



An interesting but little-known computer was the Ambilog, made by Adage, Inc. of Boston, a most innovative machine first marketed in the mid-sixties.

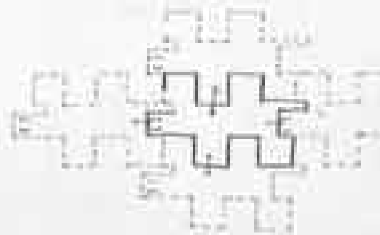
The Ambilog is a hybrid computer, i.e., both digital and analog (see Analog Computers, p. 33) . It was mentioned that "analog computers" are any electrical circuits set up to produce a result according to some formula. For certain types of repetitive functions, analog makes a lot of sense. Thus the Adage people put this machine together for highly efficient hybrid computing.

The essential idea was to have a highly ventilated machine that could take in and put out measurable electric signals at high rates. What they created was a rather straightforward digital computer with a lot of registers and converters to send analog information out and bring it back in. This meant that problems suited to repetitive electrical twisting and measurement could push out through special analog circuits, and the "answers" or doctored signals could push back in.

The instruction-set was designed for this high-speed management of input and output.

The principal applications this equipment has been used for are three-dimensional display (see Adage Display, p. DM 37) and Fourier analysis for sound and other applications (see p. DM 32, + DM 11).

# CELLULAR SYSTEMS



Now that integrated circuits are getting cheap, the distinction between registers (where things happen to information) and memory (where nothing happens to information) can be reconsidered. Storing information in cells that can themselves perform actions, or having numerous subsystems in which computation takes place, leads to a fascinating variety of possible architectures. These are generically called "cellular" computers; this is slightly ironic considering that the living cell itself is now known to be at least a digital memory, and probably more (see p. 40 ).

Examples of cellular computers more or less include STARAN, ILLIAC IV and the author's own hypothetical FANTASM (see p. 32). But this type of architecture has barely begun.

# WHERE TO GET MINICOMPUTERS.

being an incomplete, but not bad, list of manufacturers.

| Company | Some Popular Minis | Comments |
|---|---|---|
| Digital Equipment Corp. Maynard, Mass. | PDP-11 (16 bits) | Beautiful, splendid architecture. Current world favorite among sophisticates. |
| | PDP-8 (12 bits) | Beloved family dog of computer world. |
| | PDP-15 (18 bits) PDP-12 (12 bits) | Lots of goodies for measuring, dispensing info, switching. |
| Data General Corp. Southboro, Mass. | Nova (16 bits) & variants Supernova, Nova 1200, Nova 800 | Reliable. All on one large circuit card. Sleek. |
| Hewlett-Packard 1100 Wolfe Rd. Cupertino, Cal. | 2100 series (16 bits) Variety of other machines, variety of architectures | |
| Varian Data Machines 2722 Michelson Drive Irvine, Cal. 92664 | Varian 620 (16 bits) Varian 520 (16 bits) Varian 73 | |
| General Automation, Inc. 706 West Katella Orange, Cal. 92667 | SPC-16 (16 bits) | They claim marvelous architecture, purport to show this in free book, The Value of Power. |
| | 18/30 (16 bits) | Copy of IBM 1800/1130. |
| Texas Instruments, Inc. P.O. Box 1444 Houston, Texas 77001 | TI 960, 960A, 980 (16 bits) | Model 980A is fast (750 ns) and under $3000. |
| Interdata, Inc. 2 Crescent Place Oceanport, NJ 07757 | Model 70 (variable word length). Lots of others | Instructions vaguely copy IBM 360. |
| Digital Computer Controls 12 Industrial Road Fairfield, NJ 07006 | DCC-116 (16 bits) DCC-112 (12 bits) | Nova copy PDP-8 copy |
| Systems Engineering Laboratories 6901 W. Sunrise Blvd. Ft. Lauderdale, Fla. 33313 | 810 "Systems 72" (16 bits) | Offers virtual memory (making disk seemingly an extension of core memory). 50-ns mapping registers. |
| Lockheed Electronics Co., Inc. Data Products Division 6201 East Randolph St. Los Angeles, Cal. 90040 | MAC 16 (16 bits) SUE | (See "Microprocessors," p. ) |
| Electronic Processors, Inc. 5050 S. Federal Blvd. Englewood, Colorado 80110 | 16-bitter | |
| GRI Computer Corp. 320 Needham St. Newton, Mass. 02164 | GRI-99 (16 bits) | |
| Computer Automation, Inc. 18651 Von Karman Irvine, Cal. 92664 | Naked/Mini LSI | $2400 w/cabinet! (But no accessories, and roughly 7 μsec cycle time.) |
| Xerox Data Systems, Inc. Los Angeles, California | Sigma 3 (16 bits) | |
| Modular Computer Systems 2709 North Dixie Highway Ft. Lauderdale, Fla. 33308 | MODCOMP line (16 bits) | "240" general-purpose registers (presumably in core). Multi-programming. |
| Computer Terminal Corp. 9725 Datapoint Drive San Antonio, Texas 78284 | Datapoint 2200 (8 bits) | Built-in keyboard, video character scope, cassettes. Model 1: 8 μsec; Model 2: 1.6 μsec. No standard languages. |
| Standard Logic, Inc. 2215 S. Standard Ave. Santa Ana, CA 92707 | CASH-8 (16 bits) | 400 nsec, starts at $500 (Whee!) and offer a whole setup (4K, TTY, Floppy Disk) for $5000. |
| Datacraft 1200 Northwest 70th St. Fort Lauderdale, Fla. 33307 | Model 6024 (24 bits-- not really a mini) | $11,000. |
| International Business Machines Armonk, NY | IBM 1130 (16 bits) IBM 1800 (16 bits; beefed-up 1130) System 3 System 7 | Both very expensive, hardly minis in price. Sold principally for canned business systems. |
| Honeywell | H-316 (16 bits) DDP-516 (16 bits) DDP-716 (16 bits) | |
| General Electric | GEPAC 4010 GEPAC 4020 | |
| Westinghouse | W2500 | |
| Raytheon, Inc. | 704 706 707 | |

TWO USED-COMPUTER COMPANIES

American Used Computer Corporation
P.O. Box 68, Kenmore Station
Boston, Ma 02215

Computer Marketing Corp.
487 Sylvan Ave.
Englewood Cliffs, NJ 07632

# HERE THEY COME — the MICROPROCESSORS!

## COMPUTERS INSIDE COMPUTERS

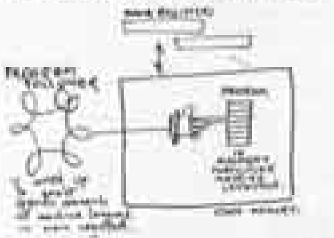
[illegible]

Microprocessors are what's happening.

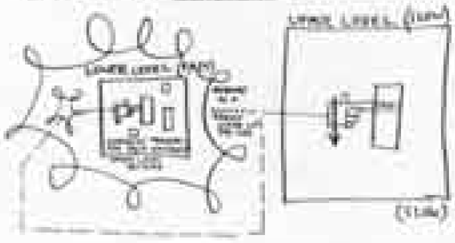
[illegible]

A microprocessor is a two-level computer.

[illegible]

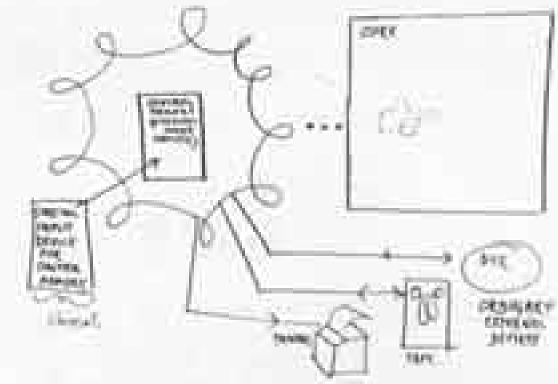


[illegible]



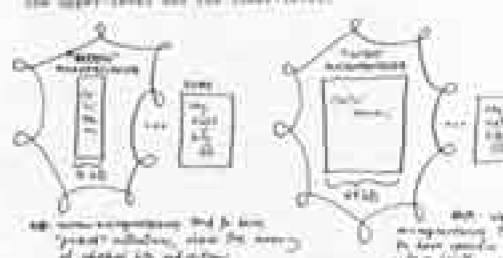
This has some extraordinary ramifications.

[illegible]



TWO LEVELS, TWO SPEEDS

[illegible]



[illegible]

BIBLIOGRAPHY

[illegible] "Current Microcomputer Architecture." Computer Design, Feb 74, [illegible]

[illegible]

## SOME MICROPROGRAMMABLE COMPUTERS.*

| | | | |
|---|---|---|---|
| Standard Computer Meta 4 | 18 bits<br>36 bits<br>80 or 35 nsec | 36 bits<br>36 bits<br>900 nsec | Big & expensive. Up to 32 hard-wire registers. |
| Burroughs 1700 | 16 bits<br>60 nsec | 24 bits<br>900 nsec | Comes with cassette holding various emulators. |
| Lockheed SUE | 16 bits | 16 bits | $450 stripped. |
| Hewlett-Packard 2100 | ? | 16 bits | Already microprogrammed to be like other HP computers -- but there's space for yours as well. $7500. |
| Microdata 3200 | 32 bits<br>135 nsec | 16 bits | $9000 up ($10,000 for model 32/S, stack-oriented). |
| Varian 73 | 64 bits<br>165 nsec<br>(198 read-write) | 16 bits<br>660 nsec | $11,000 to $100,000 (heavy upgrade of Varian 620). |
| IBM 360 model 25 | ? | 16? | |
| Prime 200 | 64 bits<br>160 nsec | 16 bits<br>750 nsec | |
| Interdata 85 | 32 bits<br>160 nsec | 16 bits<br>320 nsec | $21,000 |

## SOME MICROPROCESSORS TO BE BUILT INTO THINGS.*

(read after micro[illegible], etc., [illegible])

| | | | |
|---|---|---|---|
| Intel MCS-8 | 8 to 24 bits<br>800 nsec | 8 bits<br>12.5 usec | Stack-oriented (now faster model). |
| Intel MCS-4 | 8 or 16 bits<br>500 nsec | 4 bits<br>10.8 usec | Basic chip $60. |
| GTE 500 | (Weird but interesting wide microprocessor: [illegible] among many separate addresses, rather than branching.) | | |
| Microdata Micro 800<br>Micro 1600 | 16 bits<br>220 nsec<br>200 nsec<br>(read-write) | 8 bits<br>1.1 usec<br>1 usec | |
| AES-80 (Auto. Electric Systems, Montreal) | 12 bits<br>240 nsec | 8 bits<br>160 nsec or 1 usec | $930 w/o memory |
| National Semiconductor IMP-16C (8 1/2 x 11-- odd size for computer, convenient for notebook.) | | | $1500 stripped |
| DEC PDP-16M | 8 bits | 16 bits | $1000. (Compatible w. PDP-11 family.) |
| Atron 501 | 16 bits<br>200 nsec | 16 bits<br>1 usec | |

*(abbreviations: nsec (nanoseconds, or billionths); usec (microseconds, millionths; usual metric abbreviation).)

**HARDWARE:** equipment itself

**SOFTWARE:** computer programs

**FIRMWARE:** underprograms for microprocessors. (Also called Microprograms. Should be called Underware.)

A microprocessor simply means
a computer which has
under the binary language
you want to use
another binary language
that's cheaper to wire up.

# SUPERCHIP!

[illegible]

# the META 4

[illegible]

Lower memory: [illegible]

Upper memory: [illegible]

[illegible]

# the LOCKHEED SUE

[illegible]

# THE STANDARD COMPUTER

[illegible]

# ADVANCED PROGRAMS

In the early throes of computer enthusiasm, it is easy to suppose that anything can be done by computer-- that is, anything involving the chewing or diddling of information. This is decidedly not so.

For instance, it is easy enough, and often practical, to have a computer do something a few million times. But it is almost never practical to have a computer do something a trillion times. Why? Well, let's say (for the sake of simplicity) that a certain program loop takes 1/1000 of a second. To do it a thousand times, then, would take one second, and to do it a million times would take a thousand seconds, or about seventeen minutes. But to do it a trillion times, now, would mean doing it 17,000,000 minutes, or over thirty years.

Now, you will note that even if you speed up that loop to 1/1,000,000 of a second, a trillion repetitions will take almost twelve days, which is obviously going to need some justifying, even assuming that it is otherwise feasible.

(For problems of this type people begin thinking about building special hardware, anyway. It will be noted, for instance, that the PDP-11-- see p. [illegible]-- lets you compile your own special equipment for problems that need eternal repetitions.)

COMBINATORIAL EXPLOSIONS

One kind of thing that's too much to do is generally called a combinatorial explosion-- that is, a problem that "explodes" into too many things to do. For instance, consider the game of chess. Just because you can write a program to look ahead at all the possible outcomes of, say, tic-tac-toe, that doesn't mean you can consider all the possibilities of chess. To look at "all" the possibilities just a few moves ahead involves you in trillions of calculations. (Breather about "trillions".) And it turns out that there are a lot of problems like that.

METHODS FOR DOING THINGS

There are really no firm bounds to "what computers can do."

The problem is always to think up methods for doing things by computer. (Also called algorithms.)

Basically what can be done by computer is what can be done in a [illegible] with slips of paper-- [illegible], [illegible], sorting, [illegible], [illegible] arithmetic-- and handing slips of paper out to [illegible].

So the question should never be, "how would you do that by computer?" -- but "can you think of a method for accomplishing that?" The "computer" is really irrelevant; for it has to [illegible] and [illegible] information as needed.

Then there is the problem of "Turing impossibility." Turing was a mathematician who discovered that some things can be done in [illegible] in a finite amount of time, and some things can't, such as proving certain types of mathematical theorem. In other words, anything that has to do things in sequence-- whether a computer or a mind or God, if any-- cannot possibly know anything which is not Turing-computable. Another important limitation.

On a more practical level, though, there are just lots of things which nobody has figured out how to do in any feasible way, or are just now figuring out different systematic ways of doing. (For a favorite such area of mine, compare the different computer half-tone image synthesis systems described on pp. DM [illegible] to DM [illegible].)

Thus you see that figuring out ways of doing stuff is still one of the principal aspects of the computer field. (Whole journals are devoted to it, such as CACM, JACM and so on.)

But then of course, every few years there comes a new movement in the field that looks to make us start all over.

One such trend is called structured programming, being promulgated by a Dutch theoretician named Dijkstra, among others. The idea of structured programming is to restrict computing languages in certain ways and "eliminate the GO TO," i.e., no longer have jumps to labels placed in programs. By dividing computer programs up only in certain ways, goes this school of thought, the programs can perhaps be proven workable, in the mathematical sense, rather than just demonstrated to work, as they are now. [illegible] If the Dijkstra school is correct, we may have to start all over again with a new bunch of programming languages.

These remarks give you the flavor of some restrictions and ideas of development. The rest of this page is devoted to The Great Software Problem: The Operating System.

## OPERATING SYSTEM/360

or OS/360, or OS

We have no space here to discuss OS, the operating system of the IBM 360 and 370, which is just as well: it is a notoriously heavy-handed system, elaborated with what some would call devastating excesses. Kinds of conveniences taken for granted by users of such computer systems as the Burroughs 5500, the PDP-10, DTSS and others aren't there.

The programmer has to concern himself with intricacies having names like ACCOMs, VCBs, DCBs, ECBs, and the complications of JCL. (While these other systems may have equivalent complications, the programmer need not mess with them.) In certain efficient programs, as the 360 demands, the programmer must even set aside the previous programmer's information in "SAVE AREAS," which is like a restaurant guest having to clear the dirty dishes on sitting down-- and return them when he leaves. Several of the 360's sixteen general registers are confiscated. Time-sharing requires its own JCL-type language. And so on.

IBM says its forthcoming operating system, OS/VS2-2, will be better.

BIBLIOGRAPHY


A.L. Scherr, "The Design of IBM OS/VS2 Release 2." Proc. NCC 73, 387-394.


# OPERATING SYSTEMS & TIME-SHARING

Basically, an operating system is a program that supervises all the other programs in a computer. For this reason it is also called a supervisor or a monitor. Because the operating system is supposed to be in charge, many computers now offer special wired-in instructions that only the operating system can use. This prevents other programs from taking complete control of the machine.

Operating systems come in all sizes. The bigger ones take up a lot of computer time because they have to do a lot. The smallest kind, which are really kind of different, are just to help a single programmer move quickly between his basic programs. (A typical such system is DEC's DOS, or Disk Operating System, which you can get with the PDP-11.) This system is really a kind of butler that keeps track of where your basic programs are stored on disk and brings them in for you quickly.

A step up is the Batch Monitor, or operating system set up for Batch Processing (see p. [illegible]). In batch processing, programs go through the computer as if on a conveyor belt, one at a time (or in some systems several at a time). The operating system shepherds them.

Batch processing is used when programs don't need any interaction with human users (Or, and this is more common, when human users want time-sharing but can't get it; see below.) A multiprogramming operating system is one that allows several different programs (or conceivably sequences of batch programs) to operate at one time. (This is how most IBM 360s are used.)

SYSTEMS PEOPLE
are the folks who bring you the computer.

That is, they're the ones who try to keep the operating system working. And make the changes it needs to adapt to new equipment and working rules and schedules and software. And change the parts through which [illegible] users crash the system.

Systems people often look like every nerd in every computer system. To each other they often look like harried, overworked, unsung heroes, their fingers [illegible] wherever they can in the dike, trying to hold back the tide of Disorder.

Systems people deserve more thanks than they get.

Thank you, systems people.

Then there is time-sharing.

Time-sharing means the simultaneous use of one computer by several different users at once. It's basically a complex form of multiprogramming.

In principle this is like a Lazy Susan. The central computer works on one user's program for a while, then on another's... until it is back to the first user.

There are basically two kinds of time-sharing: time-sharing where you can only use certain facilities or languages, and time-sharing where you can use all the facilities of the computer (including programming in the computer's assembly language).

Examples of restricted time-sharing are the various mini-computer systems that are available which time-share the BASIC language. (Wang and PDP-11 and Hewlett-Packard, for instance.)

Some examples of unrestricted time-sharing are the PDP-10 (see p. [illegible]), Dartmouth's DTSS, Honeywell's MULTICS, IBM's TSO, and General Electric's MARK III.

Bigger is not necessarily better. For instance, there are time-shared versions of BASIC that run on big IBM computers. However, it may very well be that big IBM installations can save money by eliminating this function and buying instead a small Hewlett-Packard minicomputer to run their BASIC on, thereby supplying BASIC to more users at less cost and freeing the 360 for whatever it is IBM systems do better.

Restricted time-sharing, with only one or a few languages offered, is much easier to provide for than full time-sharing.

Full time-sharing is always shared with batch. In other words, the computer, darting among users, still finds some time to devote to the batch stream.

Time-sharing is self-limiting. That is, the more users are signed onto a time-sharing system at a given moment, the more slowly the system responds to all of them.

Operating systems are big and hard to program. They take a lot of the computer's time. For instance, Dartmouth's time-sharing operating system, taking as much as 23% of the computer's time, is considered efficient.

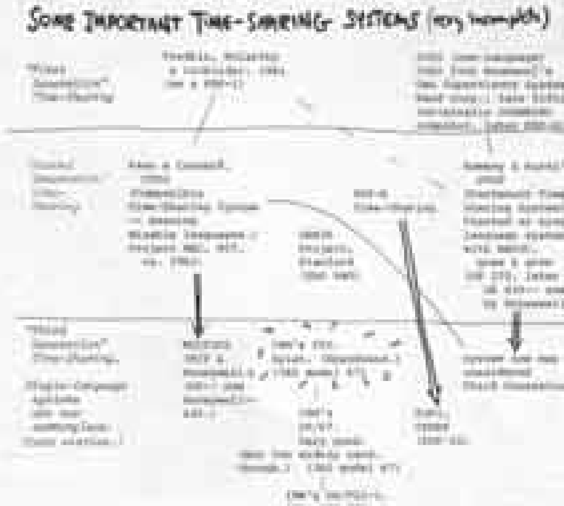
The importance of time-sharing is not in terms of "raw" efficiency, that is, the cost of each million operations, but in terms of human efficiency, the ability of each user to get as much more out of the computer by using interactive programs and languages.

OPERATING SYSTEMS TRICKERY

Swapping means transferring one user's program out of core memory in order to move in somebody else's program. This can happen very rapidly, and even when it's done to you every [illegible], your terminal may seem to respond as though you are in continuous possession of the entire computer.

Paging is one of the Great Abstruse Problems of modern operating systems. The problem is this: you've always got fast expensive memory and cheap slow memory. How can the operating system store most of your program in cheap slow memory and still predict which parts you'll need soon enough to get them in there for you? In the fancier systems, indeed, the operating system tries to predict what's most important and move it to a fast little memory called a cache. This area is so bizarre and complicated I prefer not to think about it. "Works for me," says Mr. Natural.

## SOME IMPORTANT TIME-SHARING SYSTEMS (very incomplete)



BIBLIOGRAPHY


H.T. Wilkes, Time-Sharing Computer Systems. MacDonald/American Elsevier Publishing Co.

All About Timesharing Service Companies. Datapro Research 21 Corporate Center, Moorestown, NJ 08057, $18.


# *DTSS*

DTSS is the Dartmouth Time-Sharing System, and let it be an example to us all.

It was created by Kemeny and Kurtz, who created the BASIC language to be used on it (see p. [illegible]).

Their computer arrived in fall '63. Their time-sharing system went into operation in spring '64, programmed mostly by Dartmouth students, and has grown and improved continuously since then. On that basis: programmed by students.

It's great.

The Dartmouth computer philosophy-- i.e., the idea carried through by John Kemeny and Tom Kurtz-- was that a computer is like a library: its services should be free to all in a community, paid for through some general fund.

Students and faculty at Dartmouth use it free. (Unless they have grants.) You can use it too, if you pay.

The result: everybody at Dartmouth uses the computer. It's always running, [illegible] six days a week. There are almost two hundred terminals around the campus; peak afternoon usage is about a hundred and fifty. Freshmen learn BASIC first thing, after which the computer is a standing facility, to be used in courses, in music, sociology, bibliographic research, engineering or whatever, for independent research, or just for fun and games and showing off to visitors.

The entire Dartmouth system is built for simplicity and clarity, with copious lines of all the facilities available at [illegible]. [illegible] you like, [illegible] the terminal to type out [illegible] of [illegible].

Those fuddy-duddies insist that computer usage should be billed, as it is in most college computer. That is essentially the Dartmouth view. But what if we treated it [illegible] like that? It would probably cost [illegible]. [illegible] But the point is that if we believe that software conditions are a social good, then we should be flexible about how to implement them. (See Arthur W. Luehrmann and John M. Nevison, "Computer Use under a Free-Access Policy," Science, [illegible], pp. [illegible]. This article describes this kind of approach, and at length describes the Dartmouth billing system.)

Anyway, Dartmouth will sell you its time-sharing system for about $7000 a month (and you'll need a computer setup that begins at $17,000 a month). That's not to mention a bigger setup with cost more. But that gets you [illegible], COBOL, [illegible], [illegible], the best BASIC in the whole world, games, financial systems, and myriad other programs that've built at Dartmouth. Furthermore, Mr. Administrator, it means the system will be available to users with a minimum of complication and bother.

A number of companies have bought. Including the U.S. Naval Academy at Annapolis, which offers Dartmouth-style computing to its midshipmen.

Connect charge is $7 to $9 an hour depending on what [illegible] [illegible] plus [illegible] charges.
Contact: DTSS, Inc., Hanover NH [illegible]. (Several commercial firms also offer DTSS to users, including Computer Sharing Services, Inc., Denver; [illegible] Data Systems, Woodbury, NY; Firstline Systems Ltd., Toronto.)

The most enjoyable session at the [illegible] National Computer Conference was the Nostalgia session on the Dartmouth System, DTSS. The [illegible] speakers were those guys who as kids worked on the original time-sharing system, and have now become pioneers of the more [illegible] section.

An alarming statement was made at that session by Jerome [illegible] (who said he had been the liaison man between the Dartmouth effort and the computer manufacturer (not GE)). He stated that he had been ordered by his company to drop the Dartmouth "experiment" any way he could, to leave the GE [illegible] there [illegible] [illegible]. He did no such thing, and the next after being fired continued to help the Dartmouth effort, holding weekend meetings with others from the company in his home. He deserves the Prussian [illegible] Order-of-[illegible]-[illegible] medal.



Time-sharing prices are a mix of three stuff:

1. Connect time. This you pay by the hour. Good prices: $2 (DTSS), $1.50 (Monmouth County (NJ) Community College -- but they have no communications and want no beginners).
2. "CPU charges"-- essentially the price of processing itself; depends on amount of number crunching. PDP-10 bills this in kilocore-seconds, i.e., how many thousand words of core memory your program really takes out to hand, for how many seconds.
3. Storage, which costs much. Example: 1000 characters for a month for a buck. (Typical.) That adds up fast. You might do better with a cassette memory on your terminal, such as the TermiNet (see p. [illegible]).

Five bucks an hour overall is a pretty good rate.

Note that time-sharing usually costs less in non-business hours -- but some exceptions charge more.

WHERE TO GET IT

No way can we here get into the pros and cons (both senses) of the myriad time-sharing services that are available. An excellent summary of fifty-six different time-sharing services (variously using computers by Honeywell, IBM, DEC, Univac, CDC, Xerox and Burroughs), appeared in the February, 1973 Computer Decisions ("Sizing Out the Timesharing Field" by John E. Hillegass, pp. [illegible]). This summarizes information available from Datapro Research Corp., Moorestown, NJ. The article cautions against the potential high cost of time-sharing services, and urges you to get all the advice you can before committing to a time-sharing service.

# MULTICS!

MULTICS was announced in 1965 as the Time-Sharing System of All Time, to be created jointly by MIT, General Electric, and Bell Labs.

It took a lot longer to get going than they expected. I have a 1968 (?) button that says, YOU BUILD-- HAVE YOUR [illegible] FOR MULTICS-- but now it's available from Honeywell. People say it's the greatest, all right-- its fascinating facilities include the ability to execute parts of other people's programs, if you have permission-- but it's also said to be awfully expensive.

Interestingly, the MULTICS operating system is largely programmed in the PL/I language (see p. [illegible]).

Contact: Honeywell Information Systems, 200 Smith Street, MS 050, Waltham, Mass. 02154.

## THE LOGIN HEARD 'ROUND THE WORLD: GE's MARK III

Some time-sharing systems are local, others have "concentrators" allowing users in other cities to log into them with local telephone calls.

Perhaps the most far-reaching time-sharing system, though, is General Electric's MARK III, with concentrators in many of the major cities of the world. Location Center: The main computer is in Ohio, but the overall system may be thought of as an octopus around the globe. Besides hundreds of cities in the US, the GE system offers local access in Australia, Austria, Belgium, Canada, Denmark, Finland, France, Italy, Japan, Netherlands, Norway, Puerto Rico, Sweden, Switzerland, United Kingdom and West Germany.

What this basically means is that if a company has offices in these places, it can do its internal communication through General Electric's computer system.

This presents obvious merits and difficulties, which there is no room to discuss here. The service is said to be expensive.

They also offer a toll-free number for program consultation.

Contact:

General Electric Information Services
Business Division,
401 North Washington St.,
Rockville, Md. 20850.

# TSO

IBM's "TSO", for Time-Shared Operating System, is an odd sort of time-sharing they have come up with for the 370.

It is a sort of interactive batch programming. That is, it allows the user at a terminal to communicate with programs running in batch mode.

While this is a form of true time-sharing, (though its detractors tend to compare it with what they call "true" time-sharing, such as that on the PDP-10), it has a number of drawbacks.

For instance, on the model 158, a fairly large machine (i.e. $30,000 a month-- see p. [illegible]), TSO normally allows only twenty users.

The bad feature of TSO most often mentioned is its slow response time. That is, response may be sometimes good, sometimes miserable.

IBM is urging its fans to believe that its next operating system, called OS/VS2-2, will be much better.

# THE HEARTS AND MINDS OF COMPUTER PEOPLE

Computer people are a mystery to others, who see them as somewhat frightening, somewhat ridiculous. Their concerns seem so peculiar, their hours so bizarre, their language so incomprehensible.

Computer people may best be thought of as a new ethnic group, very much unto themselves. Now, it is very hard to characterize ethnic groups in words, and certain to give offense, but if I had to choose one word for them it would be elfin. We are like those little people down among the mushrooms, skittering around completely preoccupied with unfathomable concerns and seemingly indifferent to normal humanity. In the moonlight (i.e., pretty late, with snacks around the equipment) you may hear our music.

Most importantly, the first rule in dealing with leprechauns applies as hypothesis to computer people: when one promises to do you a magical favor, keep your eyes fixed on him until he has delivered. Or you will get what you deserve. Programmers' promises are notoriously unkept.

But the dippy glories of this world, the earnestness and whimsy, are something else. A real computer freak, if you ask him for a program to print calendars, will write a program that gives you your choice of Gregorian, Julian, Old Russian and French Revolutionary, in either small reference printouts or big ones you can write in.

Computer people have many ordinary traits that show up in extraordinary ways-- loyalty, pride, temper, vengefulness and so on. They have particular qualities, as well, of doggedness and constrained fantasy that enable them to produce in their work. (Once at lunch I asked a tableful of programmers what plane figures they could get out of one cut through a cube. I got about three times as many answers as I thought there were.)

Unfortunately there is no room or time to go on about all these things-- see Bibliography-- but in this particular area of fantasy and emotion I have observed some interesting things.

One common trait of our times-- the technique of obscuring oneself-- may be more common among computer people than others (see "The Myth of the Machine," p. 8 , and also "Cybercrud," p. 8 ). Perhaps a certain disgruntlement with the world of people fuses with fascination for (and envy of?) machines. Anyway, many of us who have gotten along badly with people find here a realm of abstractions to invent and choreograph, privately and with continuing control. A strange haven for the emotions, this. Like Hegel, who became most eloquent and ardent when he was lecturing at his most theoretical, it is interesting to be among computer freaks boisterously explaining the cross-tangled ramifications of some system they have seen or would like to build.

(A syndrome to ponder. I have seen it more than once: the technical person who, with someone he cares about, cannot stop talking about his ideas for a project. A poignant type of Freudian displacement.)

A sad aspect of this, incidentally, is by no means obvious. This is that the same computer folks who chatter eloquently about systems that fascinate them tend to fall dark and silent while someone else is expounding his own fascinations. You would expect that the person with effulgent technical enthusiasms would really click with kindred spirits. In my experience this only happens briefly; hostilities and disagreements boil out of nowhere to cut the good mood. My only conclusion is that the same spirit that originally drives us muttering into the clockwork feels threatened when others start monkeying with what has been controlled and private fantasy.

This can be summed up as follows: NOBODY WANTS TO HEAR ABOUT ANOTHER GUY'S SYSTEM. Here as elsewhere, things fuse to block human communication: envy, dislike of being dominated, refusal to relate emotionally, and whatever else. Whatever computer people hear about, it seems they immediately try to top.

Which is not to say that computer people are mere clockwork lemons or Bettelheimian robot-children. But the tendencies are there.


BIBLIOGRAPHY

Gerald M. Weinberg, The Psychology of Computer Programming. Van Nostrand Reinhold.

Systematic treatment in a related vein.


This case is so classic it's almost a Punch and Judy show.

One of the nastiest people I have ever met was the head of security for a big-computer installation. Several people agree with me that he delights in telling people they can't do specific things on the computer, mainly for the sake of restricting them.

Anyway, at this same installation there was a programmer, let's call him A, who disliked authority; and disliked this director of security, let's call him B, with a wordy passion.

A spent much of his time intensely, obsessively contemplating possible ways that users might break into the system, and elaborately programming defenses and countermeasures into the machine. How do I know this? I know this from B, who constantly went through A's wastebasket. B still plans incessantly for the day B will get a big taunting printout, coming unexpectedly to him off the machine, that shows him all his secrets are known.


# COMPUTER PUTDOWNS

Practice saying them loudly and firmly to yourself. That way you won't freeze when they're pulled on you.

THAT'S NOT HOW YOU DO IT
THAT'S NOT HOW YOU USE COMPUTERS
THAT'S NOT WHAT YOU DO WITH COMPUTERS
THAT'S NOT HOW IT'S DONE
THAT'S NOT PRACTICAL
HOW MUCH DO YOU KNOW ABOUT COMPUTERS?
WITH YOUR BACKGROUND,
YOU COULDN'T UNDERSTAND IT
LET'S CALL IN SOMEONE WHO KNOWS THIS
APPLICATION (generally a shill)
IT ISN'T DONE
(you know the answer to that one)
and the one I've been waiting to hear,
IF GOD HAD INTENDED COMPUTERS TO BE USED
THAT WAY, HE WOULD HAVE DESIGNED
THEM DIFFERENTLY.

Unfortunately there is no room here to coach you on how to reply to all these. Be assured that there is always a reply. The brute-force brazen comeback, equally dirty, is just to say something like

DIDN'T YOU SEE THE LAST JOINT PROCEEDINGS?
or
OH YEAH? WHAT ABOUT THE x WORK
USING A y?

(where x is anyplace on the map on p. 5 , and y is any current computer, such as a PDP-10.)

"... programmers, in my experience, tend to be painstaking, logical, inhibited, cautious, restrained, defensive, methodical, and ritualistic."

Ken Knowlton,
"Collaborations with Artists--
A Programmer's Reflections,"
in Nake & Rosenfeld (eds.),
Graphic Languages
(North-Holland Pub. Co.), p. 399.

USEFUL, AND POSSIBLY EMBARRASSING QUESTIONS

If the Computer Priests start to pick on you, here are some helpful phrases that will give you strength.

I do not want to give the impression that the Guardians of the Machine are always bad guys. Nevertheless, sad to relate, they are not always good guys. Like everyone out to bolster his position, including the plumber and the electrician, the computer-man has learned how easy it is to intimidate the layman.

Now, these people are often right. But if you have reason to question the way things are done-- whether you're a member of the same corporation, a consumer advocate or whatever-- you are probably entitled to straight answers that will help settle the matter honestly, without pettiness. Any honest man will agree.

Now, these helpful questions, honestly answered, may elicit long mysterious answers. Be patient and confident. Write down what's said and sit down with the glossary in this book until you understand the answer. Then you can ask more questions.

I am not inviting the reader to make trouble flippantly. I am suggesting that many people have a right to know which has not been exercised, and there may be some discomfort at first.

HOW DOES IT WORK?
(This question may have to be backed up as follows: "There are no computer systems whose workings cannot be clearly described to someone who understands the basics. I INSIST THAT YOU MAKE A SINCERE ATTEMPT.")
WHY DO YOU CLAIM IT HAS TO BE THIS WAY?
(SPEAK MORE SLOWLY, PLEASE.)
WHAT IS THE DATA STRUCTURE?
COULD YOU EXPLAIN THAT IN TERMS OF THE DATA STRUCTURE?
WHO DESIGNED THIS DATA STRUCTURE?
And can I talk to him?
WHAT IS THE ALGORITHM?
WHO IS THE PROGRAMMER?
And can I talk to him?
WHY DO WE HAVE TO USE A CANNED PROGRAM FOR THIS?
WHY IS THE INPUT LANGUAGE SO COMPLICATED?
WHY DO WE NEED CARDS? WHY CAN'T PEOPLE TYPE IN THEIR OWN INPUT?
WHY NOT HAVE A SIMPLE-MINDED FRONT END THAT LETS USERS CONTROL IT THEMSELVES?
WHY HAVE FORMS TO FILL OUT? WHY NOT HAVE A DIALOGUE FRONT-END ON A MINI?
WHY CAN'T IT BE ON-LINE? *And on CRT terminals (see pp. DM 22)?*
WHY DOES IT HAVE TO BE THAT BRAND OF COMPUTER?
WHY NOT GET A SYSTEM WITH LESS OVERHEAD?
WHY SHOULD ALL COMPUTER OPERATIONS BE CENTRALIZED? DON'T THEY GET IN EACH OTHER'S WAY?
WHY DOES IT ALL HAVE TO BE ON ONE COMPUTER?
WHY NOT PUT PART OF IT ON A DEDICATED MINI?
WHY CAN'T WE DO THIS PARTICULAR THING ALL ON A MINI?
WOULDN'T IT COST LESS IF WE GOT A MINICOMPUTER FOR THIS TASK?
WHY CAN'T THIS BE PROGRAMMED IN SOME LANGUAGE LIKE BASIC?
YOU KNOW AND I KNOW THAT COMPUTERS DON'T HAVE TO WORK THAT WAY. WHY DO YOU CHOOSE TO DO IT THAT WAY?

If these suggestions seem unnecessarily contentious, it is because some of these guys like to pick on people and you may have to be ready. And you may need all the support you can get, if, say, you take a stand like one of these:

"If the information is in there, I don't see why we can't get it out."

"You have no right to ask questions like this, and if the program requires it, change the program."

Remember, ILLEGITIMIS NON CARBORUNDUM
(don't let the bastards grind you down)

"For me it always comes down to a personal challenge: not just to create a program that meets the specifications, but to do it in a way that I find aesthetically pleasing."

Robert H. Jones IV,
a heavy programmer at Chrysler

# PROGRAM NEGOTIATION

A very important kind of discussion takes place between people who want computer programs but can't write them, and people who can write them, but don't want to. Or, that is, who don't want to get caught having to do a lot of unnecessary work if it could be done more simply.

Program negotiation, then, is where the "customer"-- he may actually be the boss-- says, "I want a program that will do so-and-so," and the programmer says, "I'd rather do it this way."

In a series of requests and counter-offers the customer explains what he wants and the programmer explains why he would rather do it a different way. It is essential for both sides to make themselves completely clear. Often the customer thinks he wants one thing but would be quite satisfied with another that is much easier to program. Often the programmer can make helpful suggestions of better ways to do it that will be easier for him.

Very bad things can happen if program negotiation is not done carefully and honestly enough. The programmer can misunderstand and create something that was not wanted. Or the customer can carelessly misstate himself and ask for the wrong thing. Or worst of all-- the programmer can deliberately mishear and do something different, saying, "There, that's what you wanted," as he hands over something that isn't what was really asked for. And the poor customer may even believe it (see "Cybercrud," p. 8 ).

Program negotiation should be more widely acknowledged as a difficult and painful business. It is exhausting and fraught with stress; people (on both sides) get all kinds of psychosomatic symptoms (like abdominal pains, flus and chills). The fact that people's careers often depend on the outcome makes the atmosphere worse, rather than fostering the thorough and sympathetic conversation which is essential.

If there is one thing that laymen in business should be taught about computing, this is it.

"I CAN'T BEAR HEAT," REMARKED LANGUEDERE.

THE MEETING OF THE MINDS

| The Customer, Naive Advocate of Change | The "Expert" |
|---|---|
| I don't see why, since it's a computer... These are not details that concern me... These are just technical issues... I mean a computer can do all these things, can't it? | What you've gotta understand is that there are problems involved... It can't be that way... Leave it to me, it'll be just what you want... |

Consequence: the customer will get what he deserves.
Moral: if you want something, you'd better damn well negotiate it at the detailed level.

The strange language of computer people makes more sense than laymen necessarily realize. It's a generalized analytical way of looking at time, space and activity. Consider the following:

"THERE IS INSIGNIFICANT BUFFER SPACE IN THE FRONT HALL." (Buffer: place to put something temporarily.)

"BEFORE I ACKNOWLEDGE YOUR INTERRUPT, LET ME TAKE THIS PROCESS TO TERMINATION."

"COOKING IS AN ART OF INTERLEAVING TIME-BOUND OPERATIONS." (i.e., doing parts of separate jobs in the right order with an eye on the clock.)

# THOSE ADORABLE INFURIATING R.E.S.I.S.T.O.R.S.

[illegible]

Now here's my plan...

[illegible]

[illegible]

[illegible]

What's a group like you doing in a joint like this?

## R.E.S.I.S.T.O.R.S. Anecdotes.

[illegible]

[illegible]

---

# GUIDELINES FOR WRITERS AND SPOKESMEN

The public is thoroughly confused about computers, and the press and publicists are scarcely free from blame. IT'S TIME FOR EXPLANATIONS. People want to know what computer systems really do-- no more of this "latest space-age technology" garbage. Mr. Businessman, Mr. Writer, are you man enough to start telling it straight?

The computer priesthood, unfortunately, often wants to awe people with, or unduly stress, the notion of the computer being involved in a particular thing at all. It is time for everybody to stop being impressed by this and get on with things. Don't just copy-edit what they give you. Nose around and really find out, then write it loud and clear.

These simple rules are my suggestions for bringing on more intelligent descriptions that will help enlighten the public by osmosis.

1. FIND OUT AND DESCRIBE THE FUNDAMENTAL APPROACH AND PHILOSOPHY OF THE PROGRAM. This can invariably be stated in three clear English sentences or less, but not necessarily by the person who created it. THIS IS WHAT WRITERS ARE FOR: it is your duty to probe until the matter has become clear.

Examples.

"This chess-playing program evaluates possible moves in terms of various criteria for partial success, and makes the move which has the highest merit according to these ratings."

"This music-composing program operates on a semi-random basis, screening possible notes for various kinds of attractiveness..."

"This archaeological cataloguing system keeps track of a variety of objective features of each artifact, plus information on where it was, including linkages indicating what other artifacts were near it."

What or whose computer is used to do a thing is of almost no concern (unless it is one of unusual design, of which there are comparatively few). Not the make of the computer, but the GENERAL IDEA OF HOW THE PROGRAM OPERATES, is the most important thing.

Of course, if you are being paid by a hardware manufacturer, you'll have to name the equipment over and over; b t recognize that your real duty is public understanding, and put the facts across. (If you think it can't be done, read the splendid Kodak ads in the Scientific American.)

2. Keep gee-whizzing restricted to the description of a system's psychological effect on real people. (What impresses you may turn out to be old hat.)

3. Look for angles special to what you're reporting. Pursuing details is likely to bring up better story pegs and more human interest. Instead of saying "computer scientists" have done something, you might find something more interesting for your lead; how about "The unlikely team of a biophysicist and a teen-age art student..." or-- finding what's special-- "Never before has this been done on a computer so small, the size of a portable typewriter (and having only some 4000 words of memory)..."

4. Attempt to find out how else computers are used in the particular area, and mention these to help orient the reader.

This goes against the exclusivist tendencies we all have when we want to ballyhoo something. It is a matter of conscience, an important one.

5. Questions to ask:

What are the premises of your program?

What if people turn out to need something else?

What could go wrong?

And most important: What is that?

IMPORTANT DISTINCTIONS

It is only by clarifying distinctions that people are ever going to get anything straight.

6. Do not say "the computer" when you mean "the system" or "the program."

7. Don't say "a malfunctioning computer" (hardware error) if the computer functioned as it was directed by an incorrect program (software error). (And remember that the best programmers make mistakes, so that a catastrophic bug in a system is no sign that it was programmed by an incompetent, only that it isn't finished.)

8. (A particular point about graphics. See flip side.) Don't say "TV screen" if a computer screen is not TV, i.e., 525 horizontal lines that you can see on the screen if you look for them. (See p. DM 4 versus p. DM 77.) HOW ABOUT: "visual display screen"? -- you can add, "on which the computer can draw moving lines," or whatever else the particular system does.

9. Don't assume that your audience is computer-illiterate.

10. Don't assume that it can't all be said simply. Only lazy or hard-pressed writers are unclear.

11. Do not use cutesy-talk, particular that which suggests that computers have an individualistic character. By "cutesy" I mean sentences like "Scientists have recently taught a computer to play chess," mis-leads like "What does a computer sound like?" (when talking about music constructed by a particular program in a particular way), and awe-struck descriptions like, "At last the Space Age has come to the real estate business..."

12. Do not use the garbage term "computerized," unless there is a clear statement of where the computer is in the system, what the computer is doing and how. A "computerized traffic system," for instance, could be any damn thing, but a "system of traffic lights under computer control, using various timing techniques still under development," says something.

13. Don't put in clichés as fact, for example by the use of such terms as "mathematical" or "computer scientist" unless they really apply. Do not imply any mathematical character unless you know the system possesses it; many programs contain no operations that can fairly be called mathematical. Similarly, a "computer scientist" is someone widely or deeply versed in computers or software, not just a programmer. (Anyway, if something has been programmed by an entomologist, it is probably more interesting to refer to him as an entomologist than as a "computer scientist.")

14. Do not refer to apparent intelligence of the computer (unless that is an intended feature of the program). Credit rather the ingenuity of the system's creator. Do not say "the clever computer." If anybody is clever it is the programmer or program designer, and if you think so, say so. These guys don't get the recognition they deserve.

15. Never, never say "teach the computer" as an elliptical way of saying "write computer programs." Programming means creating exact and specific plans that can be automatically followed by the equipment. To say "teach" when you mean "program" is like "persuading" a car instead of driving it, or making a toilet "cry" instead of flushing it.

(There are systems, described on the flip side, which simulate intelligent processes and may thus be said to "learn" or "be taught." But neither programming nor simulated learning should be described in a slipshod fashion that suggests the computer is some sort of trainable baby, puppy or demon.)

16. Do not imply that something is "the last word," unless you have checked that it is.

BIBLIOGRAPHY

Ernest Gowers, Plain Words.

This wonderful little book showed English civil servants "bureaucratic writing" was totally unnecessary. Its precepts-- mainly concerned with calling a spade a spade (see p. 13)-- transpose exactly to the computer world.

"You blew it, kid"-- [illegible]

# COMPUTER FUN & MISCHIEF

All kinds of dumb jokes and cartoons circulate among the public about computers. Then our friends regale us computerfolk with these jokes and cartoons, and because we don't laugh they say we have no sense of humor.

Oh we do, we do. But what we laugh at is rather more complicated, and relates to what we think of as the real structure of things.

Some of the best humor in the field is run in Datamation; an anthology called Faith, Hope and Parity reran a lot of their best pieces from the early sixties. Classic was the Kludge series, a romp describing various activities and products of the Kludge Komputer Korporation, whose foibles distilled many of the more idiotic things that have been done in the field. ("Kludge," pronounced "klooj," is a computerman's term for a ridiculous machine.) Datamation's humorous tradition has continued in a ponderous but extremely funny serial that ran in '72 called Alan Sprach von Neumann, which in mellifluous and elliptical euphemisms described the author's adventures at the "airship foundry" and other confused companies that had him doing one preposterous thing with computers after another.

# COMPUTER PRANKS

Pranks are an important branch of humor in the field. Here are some that will give you a sense of it.

ZAP THE 94

One of the meaner pranks was a program that ran on the old 7094. It could fit on one card (in binary), and put the computer in an inescapable loop. Unfortunately the usual "STOP" button was disabled by this program, so to stop the program one would eventually have to pull the big emergency button. This burnt out all the main registers.

TIMES SQUARE LIGHTS

One of the weirder programs was the operator-waker-upper somebody wrote for the 7094. It was a big program, and what it did was DISPLAY ALPHABETICAL MESSAGES ON THE CONSOLE LIGHTS, sliding past like the news in Times Square. You put in this program and followed it with the message; the computer's console board would light up and the news would go by. Since the lights usually blink in uninteresting patterns, this was very startling.

This program was extremely complex. Since the 94 displayed the contents of all main registers and trap, arithmetic and overflow lights, it was necessary to do very weird things in the program to turn these lights on and off at the right times.

THE TIME-WASTER

In one company, for some reason, it was arranged that large and long-running programs had priority over short quick ones. Very well: someone wrote a counterattack program occupying several boxes of punch cards, to which you added the short program you really wanted run, and a card specifying how long you wanted the first part of the program to grind before your real one actually started.

This would blink lights and spin tapes impressively and lengthen the run of your program to whatever you wanted.

BOMBING THE TIME-SHARE

One of the classic bad-boy pranks is to bomb time-sharing systems— that is, foul them up and bring them to a halt. Many programmers have done this; one has told me it's a wonderful way to get rid of your aggressions.

Of course, it can damage other people's work (especially if disks are bombed); and it always gets the system programmers hopping mad, because it means you've defied their authority and maybe found a hole they don't know about. Here are a couple of examples.

1. THE PHANTOM STRIKES

The way this story is told, one of the time-sharing systems at MIT would go down at completely mysterious times, with all of core and disk being wiped out, and the lineprinter printing out THE PHANTOM STRIKES.

For a long time the guilty program could not be found. Finally it was discovered that the bomb was hidden in an old and venerable statistics program previously believed to be completely reliable. The reason the phantom didn't always strike was that the Bomb part queried the system clock and made a pseudo-random decision whether to bomb the system depending on the instantaneous setting of the clock. This is why it took so long to discover; the program usually bided its time and behaved properly.

Apparently this was the revenge of a disgruntled programmer, long since departed. Not only that, but his revenge was thorough: the Bomb part of the program was totally knitted into the rest of it, it was a very important program that had to be run a lot with different data, and no documentation existed, making it for practical purposes impossible to change.

The final solution, so the story goes, was this: whenever the rowdy program had to be run, the rest of the machine was cleared or put on protect, so it ran and had its fits in majestic solitude.

2. RHBOMB

The time-share at the Labs, never mind which Labs, kept going down. Mischief was suspected. Mischief was verified: a program called RHBOMB, submitted by a certain programmer with the initials R.H., was responsible, and turned out always to be present when the terminals printed TSS HAS GONE DOWN. It was verified by the systems people that the program called RHBOMB was in fact a Bomb program, with no other purpose than to take down the time-sharing system.

R.H. was spoken to sternly and it did not happen again.

However, some months later a snoopy systems programmer noted that a file called RHBOMB had been stored on disk. Rather than have R.H. scalped prematurely, he thought he would check the contents.

He sat down at the terminal and typed in the command, PRINT RHBOMB. But before he could see its contents, the terminal typed instead

TSS HAS GONE DOWN

But this was incredible! A program so virulent that if you just tried to read its contents, without running it, it still bombed the system! The systems man rushed from the room to see what had gone wrong.

He did so prematurely. The contents of the new file RHBOMB were simply

TSS HAS GONE DOWN

followed by thousands of null codes, which were silently being fed to the Teletype, 10 per second, preventing it from signalling that it was ready for the next thing.

# GAMES

Games with computer programs are universally enjoyed in the computer community. Wherever there are graphic displays there is usually a version of the game Spacewar. (see Steward Brand's Spacewar piece in Rolling Stone, mentioned elsewhere.) Spacewar, like many other computer-based games, is played between people, using the computer as an animated board which can work out the results of complex rules.

Some installations have computer games you can play against; you are effectively "playing against the house," trying to outfox a program. This is rarely easy. A variety of techniques, hidden from you, can be used.

When "a computer" plays a game, actually somebody's program is carrying out a set of rules that the programmer has laid out in advance. The program has a natural edge: it can check a much longer series of possibilities in looking for the best move (according to the criteria in the program).

There is a more complicated approach: the computer can be programmed to look for the best strategy in a game. This is much more complicated, and is ordinarily considered an example of "artificial intelligence" (see "The God-Builders," elsewhere in this book).

# CONWAY'S GAME OF LIFE

A Grand Fad among computerfolk in the last couple of years has been the game of "Life," invented by John Horton Conway.

The rules appeared in the Scientific American in October 1970, in Martin Gardner's games column, and the whole country went wild. Gardner was swamped with results (many published in Feb. '71); after a couple more issues Gardner washed his hands of it, and it goes on in its own magazine.

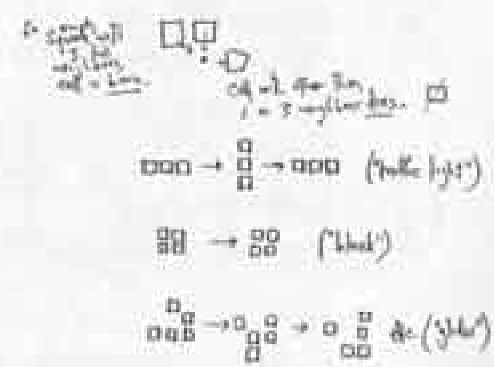
The game is a strange model of evolution, natural selection, quantum mechanics or pretty much whatever else you want to see in it. Part of its initial fascination was that Conway didn't know its long-term outcomes, and held a contest (eventually won by a group from MIT).

The rules are deceptively simple: suppose you have a big checkerboard. Each cell has eight neighbors: the cells next to it up, down and diagonally.

Time flows in the game by "generations." The pattern on the board in each generation determines the pattern on the board in the next generation. The game part simply consists of trying out new patterns and seeing what things result in the generations after it. Each cell is either OCCUPIED or EMPTY. A cell becomes occupied (or "is born") if exactly three of its neighbors were full in the previous generation. A cell stays occupied if either two or three of its neighbors were occupied in the previous generation. All other cells become empty ("die").

These rules have the following general effect: patterns you make will change, repeat, grow, disappear in wild combinations. Some patterns move across the screen in succeeding generations ("gliders"). Other patterns pulsate strangely and eject gliders repetitively (glider guns). Some patterns crash together in ways that produce moving glider guns. Weird.

While the game of Life, as you can see from the rules, has nothing to do with computers intrinsically, obviously computers are the only way to try out complex patterns in a reasonable length of time.




NON-OBVIOUS RESULTS OF SOME SIMPLE PATTERNS: some die, one blinks back and forth, others become stable. (Conway's Game of Life programmed for PLATO by Danny Sleator.)

BIBLIOGRAPHY

Donald D. Spencer, Game Playing with Computers. (Spartan/Hayden, $13.) This includes flowcharts, programs and what-have-you for some 33 games, and suggestions for more.

A continuing series of game programs (mostly or all in BASIC) appears in PCC, a newspaper mentioned earlier.

Stewart Brand's marvelous Spacewar piece, also mentioned earlier, is highly recommended.

Robert C. Gammill, "An examination of Tic-Tac-Toe-like Games," Proc. NCC 74, 349-355. Examines structure of simple games (e.g. 3D tic-tac-toe or QUBIC) where forced wins are possible, and program structures to play them.

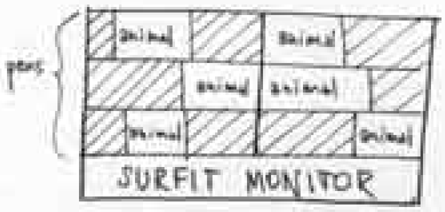
"The Game of Life," TIME, 21 Jan 74, 66-7.

(Lifeline, said to be published by Robert T. Wainwright of Milton, Connecticut.)

# SURVIVAL OF THE FITTEST

One of the stranger projects of the sixties was a game played by the most illustrious programmers at a well-known place of research; the place cannot be named here, nor the true name of the project, because funds were obtained through sober channels, and those who approved were unaware of the true nature of the project, a game we shall call SURFIT ("SURvival of the FITtest".) Every day after lunch the guys would solemnly deliver their programs and see who won. It was a sort of analogy to biological evolution. The programs would attack each other, and the survivors would multiply until only one was left.

It worked like this. Core memory was divided up into "pens," one for each programmer, plus an area for the monitor.



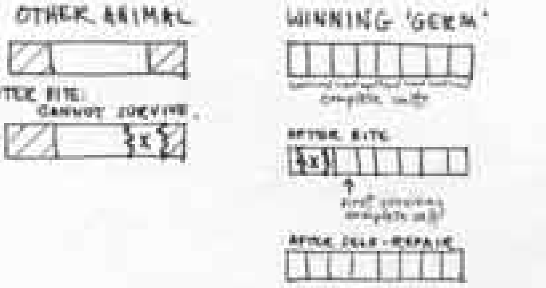

Each program, or "animal," could be loaded anywhere in its pen. The other programs knew the size of the pen but not where the animal was in it. Under supervision of the special monitor, the animals could by turns bite into the other pens, meaning that the contents of core at several consecutive locations in the other pen was brought back, and changed to zero in its original pen.

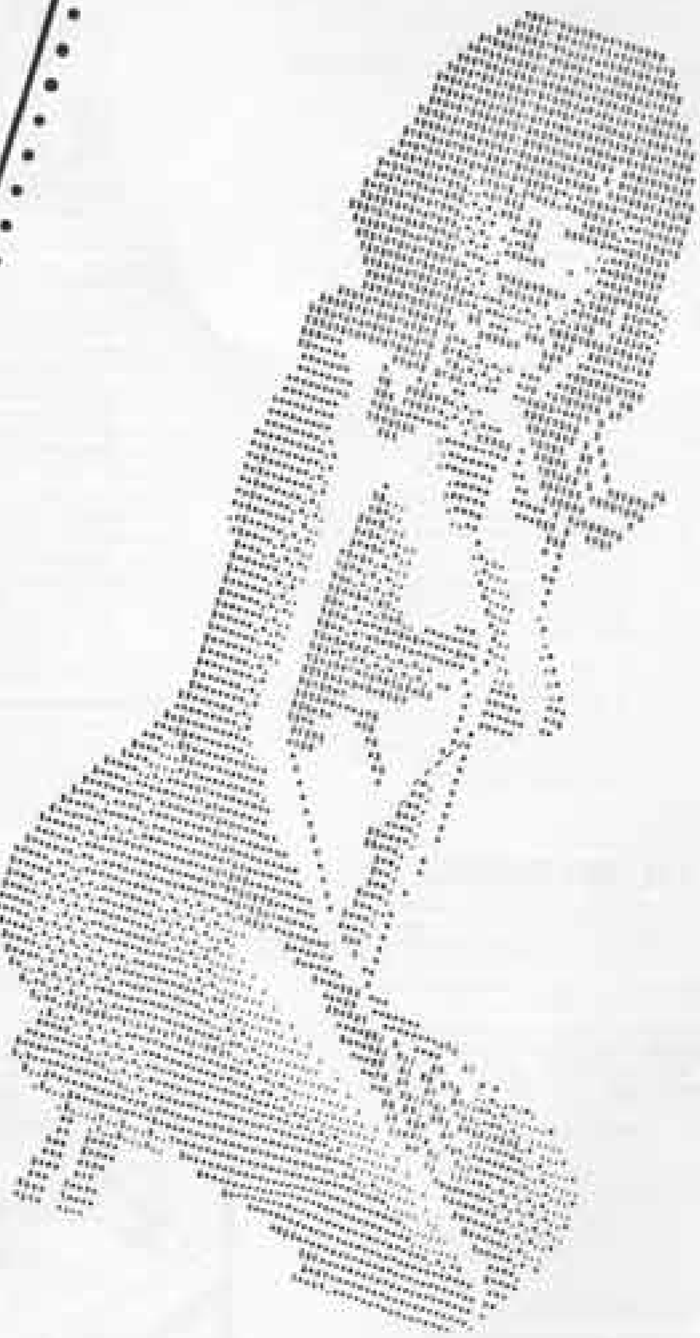
Your animal could then "digest"-- that is, analyze-- the contents bitten. Then the other animal got his turn. If he was still alive-- that is, if the program could still function-- it could stay in play; otherwise the animal who had bitten it to death could multiply itself into the other pen.

The winner was the guy whose animal occupied all pens at the end of the run. If he won several times in a row he had to reveal how his program worked.

As the game went on, more and more sophistication was poured into the analytic routines, whereby the animal analyzed the program that was its victim; so the programmer could attack better next time. The programs got bigger and bigger.

Finally the game came to a close. A creature emerged who could not be beaten. The programmer had reinvented the germ. His winning creature was all teeth, with no diagnostic routines; and the first thing it did was multiply itself through the entirety of its own pen, assuring that no matter where it might just have been bitten, it would survive.



When word got around that this nude was in a public file on the time-sharing system, my office-mates scrambled to get printouts of her. The cleverest, though, had a deck punched. As he predicted, she was thrown off by the systems people within an hour or so-- leaving the other guys with their printouts, but he had the deck. Now he can put her back in the computer any time, but they can't.

Twitting a program within its own premises is a jolly aspect of computer fun. This game of three-dimensional tic-tac-toe was played with a program running on a minicomputer at the Spring Joint, 1969. CAUTION-- ADULTS ONLY. Ⓧ While this example may offend some people, it vividly shows how programs may be toyed with -- in this case, by the mischievous sign-on-- to make them behave humorously.

# HOW COMPUTER STUFF IS BOUGHT AND SOLD

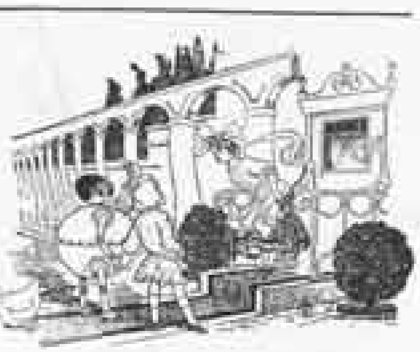
For the most part, big computers have always been rented or leased, rather than bought outright. There are various reasons for this. From the customer's point of view, it makes the whole thing tax-deductible without amortization problems, and means that it's possible to change part of the package-- the model of computer or the accessories-- more easily. And big amounts of money don't have to be shelled out at once.

From the manufacturer's point of view (and of course we are speaking mostly of IBM), it is advantageous to work the leasing game for several reasons. Cash inflow is steady. The manufacturer is in continuous communication with the customer, and has his ear for changes and improvements costing more. Competitors are at a disadvantage because the immense capital base needed to get into the selling-and-leasing game makes competition impossible.

Basically, leasing really may be thought of as having two parts: the sale of the computer, and banking a loan on it, essentially the lease payments are installment payments, and the real profits come after the customer has effectively paid the real purchase price and is still forking over.

Many firms other than IBM prefer to sell their computers outright. Minicomputers are almost always sold rather than rented. However, for those who believe in renting or leasing, the so-called "leasing firms" have appeared, effectively performing a banking function. They buy the computer, you rent or lease it from them, and they make the money you would've saved if you'd bought.

IBM, now required to sell its computers as well as lease them, keeps making changes in its systems which some think are done partly to scare companies away from leasing, since if you've bought the computer you can't catch up. (Large computers bought from companies that like to sell them, such as DEC and CDC, do not seem to have this problem.)



SOFTWARE

Computer programs, or "software," used to come free with the computer. But IBM turned around and "unbundled," meaning you had to buy it separately, and there has been some following of this example. However, for users who are buying a computer with some canned program for a particular purpose, prices are obviously for the whole package; it's people who use the same computer for a lot of different things that have to pay for individual programs.

There are many small software companies. For the cost of a letterhead anyone can start one; the question is whether he has anything special to sell. Some people whomp up programs on their own which turn out to be quite useful. (For instance, one Benjamin Fitzroy offers a magnificent program in Fortran to generate textual garbage. It's so good it can be used to expand proposals by hundreds of pages. He calls it Simplified Integrated Modular Prose (SIMP) and it sells for $10. His address is: Computer Center, University of Georgia, Athens GA 30602*.)

Obviously, to create big systems for intricate management purposes requires a great deal more effort. Traditionally these are done by vast programmer teams working in COBOL or the like, constantly fighting with monster programs and chewing up millions of dollars. However, the new Quickie Languages (three shown pp. 14-17) may offer great simplification of such programming tasks.

Programs are protected by copyright-- that's the only way there can be a software industry at all-- but since there has been no court litigation in the field, nobody knows what the law really is or what it covers. Everybody agrees that traditional copyright precedent covers a lot of ground-- "derivative works" definitely violate copyright, even study guides to textbooks-- but no one knows how far this goes.

Same for patents. The Patent Office has granted program patents, notably the one on the sorting program of Applied Data Research, Inc., but The Patent Office has a profound distaste for this potential extension of its duties and is telling everyone that programs aren't patentable, even though they clearly fall within the boundary as unique, original processes.

People who only read the headlines think that the Supreme Court struck down the patentability of programs. No such thing.

In this light the patents that the University of Utah has gotten on the halftone image synthesis programs of Romney and Wylie and Romney (see p. ) are of considerable interest. These patents use the "software as hardware" ruse: the program is described in detail as taking place in a fictitious machine shown in many detailed drawings whose individual character is not readily seen by the uninitiated; events vaguely taking place in "microprogrammable microprocessors" have been neatly foisted on the Patent Office as detailed technical disclosure. It's a great game. The idea is that the claims are so drawn as to cover not just the fictitious machine, but any program that should happen to work the same way. But such approaches, though common in previous patent practice, have not yet been litigated in this field.

USED COMPUTERS

While in principle there would seem to be every advantage in buying used computers, there are certain drawbacks. Service is the main one: the manufacturer is not very hopeful about fixing secondhand machines, and you may have to prove how to do it yourself. Even with machines still available, (you may have trouble getting a service contract from the manufacturer). (Some of these have been unavailable? [illegible] Computer, in Boston, with [illegible] that its merchandise will be accepted back into manufacturer's contract service.) A final drawback is price: a popular machine may cost as much used as new, since they're saving you the waiting period.

It's kind of unfortunate: obviously usable machines get wasted. (Not here's waste for you: certain well-known laboratories, owned by a profit-making monopoly, smash their used computers if nobody wants them within the lab. They claim they can't resell them because they would then be "competing" with the manufacturers. I wish the conservationists would get on that one.)

(Aside from all that, it seems that all the surviving members of the Philco computer, a nice machine but very much discontinued, have either gone to the state of Israel or to Pratt Institute in Brooklyn. When I spoke at Pratt they showed me their Philco machines, chugging faultily, and said they had (I think) some four more Philcos in crates, donated by their original owners.)

ANNOUNCEMENTS

An eccentric aspect of the computer field is the Announcement, the statement by a company (or even individual) that he is planning to make or sell a certain computer or program. Some very odd things happen with announcements in this field. (Some of this is unique to computerdom, but it goes to unusual extremes here.)

Under our system it is permissible for any person or firm to announce that he will make or sell any particular thing, and even if he's lying through his teeth, it's not ordinarily considered fraud unless money changes hands. Talk is cheap. Thus it is common practice in American industry for people to say that they will soon be selling hundred-mile-an-hour automobiles, lighter-powered [illegible], [illegible] [illegible].

Okay. In the computer world the same thing happens. The strategy depends on the announcer's market position. The little guys are often bluffing wistfully, hoping someone will get interested enough to put up the money to finish the project, or the like; the big companies are often "testing the water," looking to see whether there are potential customers for what they haven't even attempted to develop. Announcements by big companies also have strategic value: if they announce something a smaller guy has already announced, they may cut him off at the pass, even though they have no intention of delivering. That's just one example. The whole role of IBM's announcements in a poker game in the field. It has been alleged, for instance, that IBM announced its 360 computer long before it was ready to cut off incursions on its customers by other firms. Control Data, in a recent suit, alleged that the Model 90 numbers of the 360 were announced, and then developed, simply to destroy Control Data and its own big fast machines. These are just examples.

In other words, caveat emptor.

Datamation ran several good articles on buying computer stuff in its September, 15, 1970 issue.

"Software Buying" by Howard Bromberg (35-40) and "Contract Caveats" by Robert P. Bigelow (41-44) are very helpful warnings about not getting burned.

Another, "Project Management Games," by Werner W. Leutert (24-34) is an absolutely brilliant, blood-curdling strategic analysis of the ploys and dangers involved in buying and selling very expensive things, such as computers and software. ANYONE INVOLVED IN COMPUTER MANAGEMENT SHOULD READ THIS MACHIAVELLIAN PIECE WITH THE GREATEST CARE. Anyone interested in the theory of showdowns and negotiation can read it with a different zest.

# UH OH, MAINTENANCE

A practical problem of immense importance is "maintenance," meaning repair and upkeep of computers and their accessories. Lots of guys in Boston and L.A. are raving for making computers, but here you are stuck in Squeedunk and it doesn't work anymore.

Trying to find people who will fix these things on a steady basis is a great problem.

You can sign a "maintenance contract" with the manufacturer, which is sort of like breakdown insurance: whatever happens he'll fix. Eventually. If you own equipment from different manufacturers, though, it's worse: each manufacturer will only contract to fix his own equipment. (And remember, interfaces have to be maintained too.)

This is the biggest point in favor of IBM. Their maintenance is superb.

There's also something called third-party maintenance: companies who'll contract to keep all your hardware working. RCA and Raytheon are into that.

THE SEVEN DWARVES AND THEIR FRIENDS

The computer companies are often called "Snow White and the Seven Dwarves," even though the seven keep changing. Here are some main ones beside IBM. I hope I haven't left anyone out.

- Sperry Rand Univac
- Honeywell
- Burroughs
- Control Data Corporation (CDC)
- National Cash Register (NCR)
- Digital Equipment Corporation (DEC)
- Xerox Data Systems (XDS, formerly Scientific Data Systems (SDS))
- Hewlett-Packard (HP)
- Data General
- Interdata, Inc.
- Varian Data Machines
- Lockheed

Requiescant in Pace:

- General Electric (sold out to Honeywell)
- RCA (sold out to Univac)
- Philco
- General Foods
- & others beyond recollection.

Some modestly program offered by Ben Fitzroy, Esq.* (see text)

Additional Programs

Calendar
This FORTRAN program can produce a calendar for any year from 1583 through 2100. The calendars are printed at the bottom of a Playboy bunny who is lovelier than a [illegible]. $25.

Birth Announcement
This PL/I program produces a picture of a baby in the fetal position with the details of a birth announcement in the interior. The details can be up to 160 characters. See [illegible]. $15.

Simplified Integrated Modular Prose (SIMP)
This FORTRAN program produces page after page of double talk. The input is a permanent data deck of phrases followed by one or more [illegible] little cards. Each little card is placed at the top of a new page. The program uses 4 letters from the title to prime the random number generator and selects phrases from the table and combines them into sentences and paragraphs to produce the "report". The program can easily be expanded by increasing the size of the phrases and the data in the phrase deck. See attached example. $10.

Louis and Holiday
The deck when listed produces a Santa in his sleigh being pulled over several pine tops and [illegible] by a reindeer. Above and [illegible] the words MERRY CHRISTMAS and HAPPY NEW YEAR. $5.

Snoopy
The deck when listed produces a Snoopy. $5.

* More recent address: c/o Computech Systems Inc., 1819 Peachtree Rd., Atlanta GA 30309.

STATUS REPORT NUMBER 1

[illegible]

SIMP Example

# HOW (SOME) COMPUTER COMPANIES ARE FINANCED — A PERSPECTIVE

Those of us who were around will never forget the Days of Madness (1968-9). Computer stocks were booming, and their buyers didn't know what it was about, but everywhere there were financial people trying to back new computer companies, and everywhere the smart computer people who'd missed out on Getting There were looking for a deal.

Datamation for November 1969 was an inch thick, there were that many ads for computers and accessories.

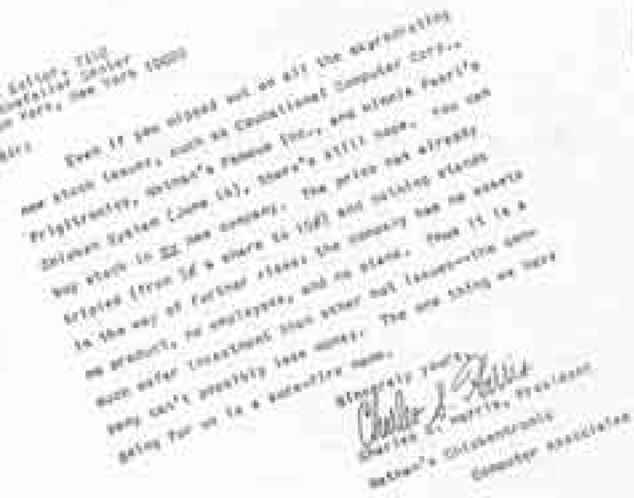
At the Fall Joint Computer Conference that year in Las Vegas, I had to cover the highlights of the exhibits in a hurry, and it took me all afternoon, much of it practically at a trot. Then, after closing time, I found out there had been a whole other building.

It is important to look at how a lot of these companies were backed, the better to understand how irrationality bloomed in the system, and made the collapse of the speculative stocks in 1970 quite inevitable.

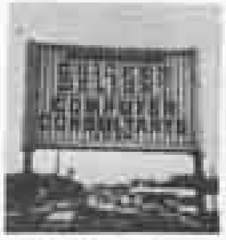
A number of companies were started at the initiative of people who knew what they were doing and had a clear idea, a new technique or a good marketing slant. These were in the minority, I fear.

More common were companies started at the initiative of somebody who wanted to start "another X"-- another minicomputer company, another terminal company, expecting the product somehow to be satisfactory when thrown together by hired help. Perhaps these people saw computer companies as something like gold mines, putting out a common product with interchangeable commodity value.

The deal, as some of these Wall St. hangers-on would explain it, was most intriguing. Their idea was to create a computer company on low capital, "bring it public" (get clearance from the SEC to sell stock publicly), and then make a killing as the sheep bought it and the price went up. Then, if you could get a "track record" based on a few fast sales, the increasing price of your stock (these are the days of madness, remember) makes it possible to buy up other companies and become a conglomerate.


Even if you missed out on all the skyrocketing new stock issues, such as Canonizational Computer Corp., Frightronics, Shrinke's Famous Inc., and Minnie Pearl's Chicken System (June 14), there's still hope. You can buy stock in ?? new company. The price has already tripled (from 5¢ a share to 15¢) and nothing stands in the way of further rises; the company has no assets, no product, no employees, and no plans. Thus it is a much safer investment than other hot issues--the company can't possibly lose money. The one thing we have going for us is a seductive name.

Sincerely yours,
Charles S. Harris, President
Nathan's Chickentronic Computer Associates



Yes, it's real. Life imitates art on Route 46, N.J.

It was very difficult to talk to these people, particularly if you were trying to get support for a legitimate enterprise built around unusual ideas. (Everybody wants to be second.) And what's worse, they tended to have that most reprehensible quality: they wouldn't listen. Did they want to hear what your idea actually was? "I'll get my technical people to evaluate it"-- and they send over Joe who once took COBOL. I finally figured out that such people are impossible to talk to if you're sincere-- it's a quality they find unfamiliar and threatening. I don't think there's any way a person with a genuine idea can communicate with such Wheeler-Dealers; they just fix you with a piercing glance and say "Yeah, but are we talking about hardware or software?" (the two words they know in the field).


"IT'S A WHEELER!"

The joker is that if you missed out on all this you were much better off. Anyone with a genuine idea is being set up for two fleecings: the first big one, when they tell you your ideas, skills and long-term indenture are worth 2½% (if you're lucky) compared to their immense contributions of "business knowhow," and the second, when you go public and the underwriter gets vast rakeoffs for his incomparable services. What is most likely to get lost in all this is any original or structured contribution to the world that the company was intended, in your mind, to achieve.

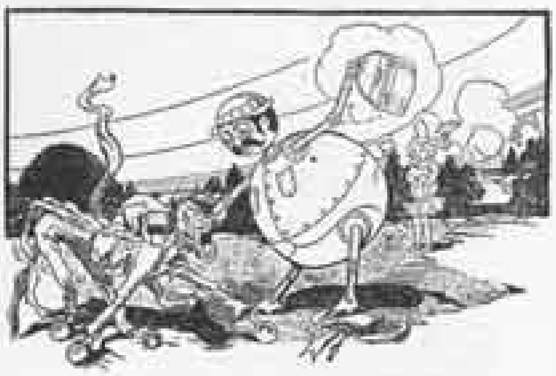
In part this is because anyone with technical knowledge is apparently labelled Silly Technician in the financial community, or Impossible Dreamer; it is entrenched doctrine among many people there that the man with the original idea cannot be allowed to control the direction of the resulting company. In one case known to me, a man had a beautiful invention (not electronic) that could have deeply improved American industry. It was inexpensive, simple to manufacture, profoundly effective. He made his deal and the company was started, under his direction. But it was a trick. When the second installment of financing came due (not the second round, mind you), the backers called for a new deal, and he was skewered. Result: no sales, no effect on the world, no nothing to speak of.


This is all the sadder because the companies that achieve important things in this field, as far as I can see, are those with a unifying idea, carried out unstintingly by the man or men who believe in it. I think of Olsen's Digital Equipment Corporation, Data General, Evans and Sutherland Computer Corporation, Vector General. This is not to say that a good idea succeeds without good management or good breaks: for instance, Viatron, a firm which was the darling of the computer high-flying stocks, had a perfectly sound idea, if not a deep one: to produce a video terminal that could be sold for as little as $100 a month. But they got overextended, and had manufacturing troubles, and that was that. (You can now get a video terminal for $49 a month, the Hazeltine.) Of course, a lot of ideas are hard to evaluate. A man named Ovshinsky, for instance, named a whole new branch of electronics after himself ("ovonics"), and claimed it would make integrated circuits cheaper or better than anybody else's. Scoff, scoff. Now Ovshinsky has had the last laugh: what he discovered some now call "amorphous semiconductor technology," and his circuits are being used by manufacturers of computer equipment. Another example is one Frank Marchuk, whose "laser computer" was announced several years ago but hasn't been seen yet. Many computer people are understandably skeptical.

This is still a field where individuals can have a profound influence. But the wrong way to try it is through conventional corporate financing. Get your own computer, do it in a garret, and then talk about ways of getting it out to the world.

BIBLIOGRAPHY


John Brooks, The Go-Go Years. Weybright & Talley. $10.

# THE BEHEMOTH IBM

also known affectionately in the field as
International Big Mother
Itty-Bitty Machine Co.
International Brotherhood of Magicians
"I'm Being Moved"
Institute of Black Magic
In Bleakest Mordor
It's Better Manually

as well as
Mother of Us All
The Grim Gray Giant
Big Mama Crass
Security Blanket
Snow White
Grey Menace
and
Big Brother.

"IBM," as everyone knows, is the trade mark of the International Business Machines Corporation, an immense company centered in Armonk, N.Y., but extending to over a hundred countries and employing well over a quarter of a million people.

IBM dominates two industries, computers and electric typewriters.

To many people, IBM is synonymous with computers. Some of the public, indeed, believe them to be the only computer manufacturer.

In cameras and film, there is Kodak. In automobiles, there is General Motors. And in the computer field there is IBM.

IBM sells some 65 to 75% of all the computers and programs that are sold. In this respect, the balanced near-monopoly, they are like Kodak and GM.

But there are important differences. Everybody knows what a camera is, or an automobile. But to many, if not most, people, a computer is what IBM says it is.

The importance of this firm, for good or ill, cannot be overstated: whose legend is so thick; whose stock prices have doubled and redoubled, ten times over, to its multibillion-dollar mass; whose seeming infallibility-- at least, as seen by outsiders-- have been the stuff of legend; whose style has proliferated across the world, a style which has in a way itself become synonymous with "computers," whose name symbolizes for many people-- remarkably, both those who love it and those who hate it-- the New Age.

The rigidity associated in the public mind with "the computer" may be related in some deep way to this organization. As a corporation they are used to designing systems that people have to use in their jobs by fiat, and thus there are few external limitations on the complications in our lives that IBM can create.

Many people mistake IBM for "just another big company," and here lies the danger. IBM's position in the world is so extraordinary, so carefully poised (as a result of various antitrust proceedings and precautions) just outside of total monopoly of a vitally important and all-penetrating field, that much of what they do has implications for all of us. Ralph Nader's contention that General Motors is too powerful to function as an independent government surely applies even more to IBM. General Motors is not in a position to persuade the public that every car has to have ten wheels and a snowplow. IBM seems in some ways to have molded computers in its own image, and then persuaded the world that this is the way they have to be.

But IBM is deeply sensitive, in its way, to public relations, and has woven an extensive system of political ties and legends (if not mythology) which have kept it almost completely exempt from the critical attention of concerned citizens.

Thus it is necessary here, simply as a matter of covering the field at an introductory level, to raise some questions and criticisms that occur to people who are concerned about IBM. IBM presumably will not mind having these matters raised; their public-spirited concern in so many areas assures that when something so publicly important as the character of their own power is concerned, occasional scrutiny should be welcome.

## A FINE PROGRESSIVE CORPORATE CITIZEN AND A WONDERFUL EMPLOYER

It is important to note first of all that IBM is in many respects the very model of a generous and dutiful corporate citizen. In "community relations," in donations to colleges and universities, in generous release of the time of its employees for charitable and civic undertakings, it is almost certainly the most public-spirited corporation in America, and perhaps on the face of the earth.

They have been generous about many public interest projects, from Braille transcription to donating photographers and facilities for films on child development.

The corporation sponsors worthwhile cultural events. "Don Quixote" with Rex Harrison on TV was terrific; Katherine Hepburn's "Glass Menagerie" was marvelous.

They treat their small suppliers honorably and with great solicitude.

IBM's enlightenment and benevolence toward its employees is perhaps beyond that of any company anywhere. They have vigorously upgraded the position of women and other minority employees; the opportunities for women may be greater there than anywhere else. They have upgraded repair of their systems, at any level, to white-collar status, and tool kits are disguised as briefcases. This innovation, making a repairman into a "field engineer," is one of the cleverest public-relations and employment policies ever instituted.

They are openhanded to employees who want to run for office, evidently regardless of platform. In the sixties there were young candidates who worked for IBM, and evidently got time off for it. More recently, Fran Youngstein, an IBM marketing instructor, was a 1973 candidate for Mayor of New York on the ticket of the Free Libertarian Party, opposing all laws against victimless crimes (e.g. prostitution and odd sex) as well as Day Care and welfare.

They also rarely fire people. Once you're in, and within certain broad outlines, it's extremely safe employment. For those who turn out not to fit in well, they have a tradition of certain gentle pressure-practices like moving you around the country repeatedly at IBM expense. This encourages leaving, but also exposes the less-wanted employee to a variety of opportunities he might not otherwise see, without the trauma and anxiety of dismissal.

(It is said that there are IBM firings, but they are rare and formidable. Heywood Gould's description of an IBM firing (Corporation Freak, pp. 115-118), for which he does not claim authenticity, is nevertheless bloodcurdling.)

IBM's international manners (in its 113 countries) are likewise praiseworthy. Compared to the perfidious behavior of some of our other multinational corporations, they are sweetness and light and highschool civics. Sensitive to the feelings of people abroad, they are said to operate carefully within arrangements made to satisfy each country. They train nationals for real corporate responsibility rather than bringing in only outside people. And they are sensitive to issues: for instance, they recently refused to set up an Apartheid computer in South Africa.

> ONE THING IS PERFECTLY CLEAR
>
> IBM has no monopoly on understanding or sophistication.

## THEN WHY SUCH A RANGE OF FEELINGS TOWARD IBM?

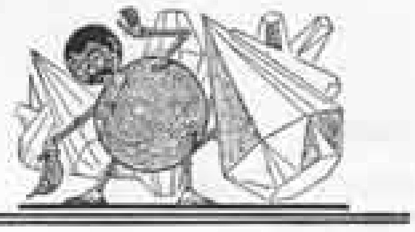
Among computer people, feelings toward IBM range from worship to furious hate (depending only in part on whether you work there).

Many, many are of course employed by IBM, and the devotion with which they embrace the corporation and its spirit is a wonder of the world.

But the spiritual community of IBM extends further. Upper-management types, especially Chairmen of Boards and comptrollers, seem to have a reverence for IBM that is out of this world, some amalgamated vision which entwines images of eternal stock and dividend growth with an idealized notion of management efficiency. Many others use and live with IBM's equipment, and view IBM as anything from "the greatest company in the world" to "a fact of life" or even "a necessary evil." In some places whole colonies of users mold themselves in its image, so that around IBM computers there are many "little IBMs," full of people who imitate the personalities and style of IBM people. (RCA, before its computer operation fell to pieces, imitated not just the design of IBM's 360 computer, but a whole range of titles and departmental names from out of IBM. The sincerest form of flattery.)

But outside this pale-- beyond the spiritual community of IBM-- there are quite a few other computer people. Some simply ignore IBM, being concerned with their own stuff. Some like IBM but happen to be elsewhere. Others dislike or hate IBM for a variety of reasons, business and social. And this smoldering hatred is surely far different in character from anybody's attitude toward Kodak or GM.

While it is not the intent here to do any kind of an anti-IBM number, it is nevertheless necessary to attempt to round out the one-sided picture that is projected outside the computer world. In what follows there is no room to try to give a balanced picture. Because IBM can speak for itself, and does so with many voices, it is more important to indicate here the kinds of criticisms which are commonly made of IBM by sophisticated people within the industry, so that IBM-worshippers will have some idea of what bothers people. But of course no attempt can be made here to judge these matters; this is just intended as source material for concerned citizens.



## THE GOOD NEWS AND BAD NEWS ABOUT IBM

| First, the good news. | Now for the bad news . . . |
|---|---|
| They offer many computer programs for a variety of purposes. | These programs are not necessarily set up the way you would want them. (But if you take the trouble to adapt to them, you'll probably never get back.) The programs force card or card-like input and, to date, strongly discourage time-sharing and widespread convenient terminal use by untrained people. IBM programs are also notoriously inefficient. (That way you have to use bigger machines for longer.) |
| A company or governmental agency can get immense amounts of "help" and "information" from IBM, which offers free courses, even IBM people on "released time" to look over the problems or the programs. | The courses indoctrinate with the IBM outlook, and the planted people spread it. Moreover, both mechanisms help IBM spot the people they can work with to make a big sale-- and (it is alleged) to sweep those who stand in the way. |
| IBM offers various kinds of compatibility among its systems. | It always seems to cost extra. |
| IBM equipment is rugged and durable, and their repairmen or "field engineers" struggle with great diligence and alacrity to keep it running. | You may not like the way it runs. |

1. SOCIAL ASPECTS OF IBM.

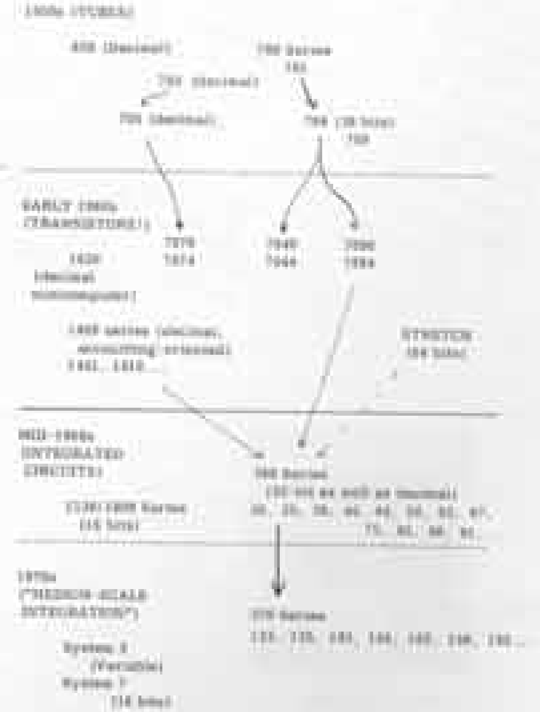
It is perhaps in the social realm, including its ideological character, that a lot of people are turned off by IBM.

IBM has traditionally been the paternalistic corporation. (Paternalistic corporations were some kind of big philosophical issue to people in the fifties, but nobody cares anymore. Anyway, the rest were perhaps inconsequential compared to IBM.) Big IBM towns not only have a Country Club (no booze), but a Homestead for the comfort of important corporate guests. There are dress codes (although non-white shirts and below-the-collar hair are now allowed), and yes, codes of private behavior (now outdated). These irritate people with libertarian concerns. They do not bother employees, evidently, because employees knew what they were getting into.

Generalizations about IBM people obviously cannot be very strong. Obviously there is going to be immense variation among 261,000 people, half of whom have college degrees; but of course one of the great truths of sociology is that any non-random group has tendencies.

More than that in this case. In a way IBM people are an ethnic group. Impressive indeed are the general energy and singlemindedness of the people, galvanized by their certainty that IBM is true, good and right, and that the IBM way is the way. This righteousness is of course a big turn-off for a lot of people. Perhaps it leads in turn to the most-heard slurs about IBM people, that they are brainwashed or provincial.

## MAJOR IBM COMPUTERS AT A GLANCE




The same slick marketing could be applied to any other industry. But it wouldn't be IBM. Nowhere else could the mystery of the subject be met and enhanced with so many more mysteries.

PROVINCIAL?

There would seem to be no question that IBM people are comparatively conservative and conventional. This partly because that's who IBM hires (though they reportedly urge tolerance of the unusual employee in a training film, "The Wild Duck"). A huge number of IBM people never worked for anybody else; obviously this affects the perspective, like staying at one university all your life, or in one city.

[illegible]

BRAINWASHED?

[illegible]

[illegible]

It's still possible.

[illegible]

1. SALES TECHNIQUES

[illegible]

"When we went from IBM to National Cash Register, it was like the difference between night and day."

Retired hardware executive, talking about inventory programs

[illegible]

2. TECHNICAL DECISIONS AND DESIGNS

A. Prestige

[illegible]

# IBM'S CONTROL
## THE VIRTUAL MECHANICS

IBM controls the industry principally by controlling its customers. Through various mechanisms, it seems to enforce the principle that "Once an IBM customer, always an IBM customer." [illegible]

[illegible]

A. Interconnection and compatibilities

[illegible]

B. Control and guidance of what the customer can get

[illegible]

C. Having to do things just their way

[illegible]

D. Captive bureaucracies running in place?

[illegible]

PROSPECTS

[illegible]

"... IBM is an infinite window, has learned that this is the way we must go."

Typical computer installation manager, quoted in Computerworld, [illegible]

An interesting example of an IBM non-breakthrough was the dramatic announcement in 1964 of the 360 computer, portrayed as a machine which would at last combine the functions of both "business" computers and "scientific" computers. But other companies, such as Burroughs (with the 5500) had been doing this for some time. The quaint separation of powers between scientific computers (with all-binary storage of numbers) and business computers (decimal storage) was based only on tradition and marketing considerations, and was otherwise undesirable. In amalgamating the "two types," IBM was only rescinding their own previous unnecessary distinction. The drama of the announcement derived in large measure from the stress they had previously laid on the division. (Fortune has an interesting piece on the decision struggles preceding the introduction of the 360 computer, and the internal arguments as to whether there should be one line of computers or two. See the five-billion-dollar gamble piece, Bibliography.)

This ties in closely with another interesting aspect of the IBM image, the public notion that IBM is a great innovator, bringing out novel technologies all the time. It is well known in the field that they are not. IBM usually does not bring out a new type of product until some other company has pioneered it. (Again remember the earlier point, that the product offering is a strategic maneuver.) But of course such facts do not appear in the promotional literature, nor are they volunteered by the salesman.

The expression for this in the field is that IBM "makes things respectable." That is, customers get that reassured feeling, when IBM adds other people's innovations to their product line, and decide it's okay to go ahead and rent or buy such a product. (This also sometimes kicks business back to the original manufacturer.)

A few examples of things that were already on the market when IBM brought them out, when making them sound completely new: transistorized computers (first offered by Philco), virtual memory (Burroughs), microprogramming (introduced commercially by Bunker-Ramo).

This is not to say that IBM is incapable of innovation; merely that they are never in a hurry about it. The introduction of IBM products is orchestrated like a military campaign, and what IBM brings out is always a carefully planned, profit-oriented step intended not to threaten its product base. This is not to say that they don't have new stuff in the back room, a potential arsenal of surprises of many types. But it is probable that most of them will never be seen. This is because of IBM's "impact" problem.

Unique in IBM's position is the problem of fitting new products into the market alongside its old ones. Its problem is much worse, say, than that of Procter & Gamble. The problem is not merely the size and the diversity of the products, but the fact that they may interfere with each other ("impact" each other, they say) in very complicated ways. A program like their Datatext, for example, which allows certain kinds of text input and revision from terminals, may affect its typewriter line. These are no small matters: the danger is that some new combination of products will save the customers money IBM would otherwise be getting. Innovations must exceed the amount IBM is taking in, or IBM loses by making them.

These complications of the product line in a way provide a counterbalance to IBM's fearsome power. The corporation has an immense inertia based on its existing product line and customer base, and on ways of thinking which have been carefully promulgated and explained throughout its huge ranks, that cannot be reversed quickly or flippantly.

Nevertheless it is remarkable how at every turn— notably when people think IBM will be set back— they manage to make policy decisions or strategic moves which further consolidate their position. (When these seem to involve restricting the way their computers will be used, see box, "IBM's Control.")

(The most ironic such countermove by IBM occurred a few years ago with the so-called "unbundling" decision. IBM at last agreed (on complaint from other software firms) to stop giving its programs away to people renting the hardware. Glee was widespread in the industry, which expected IBM to lower computer prices in proportion to what it would now charge for the software. Not at all. IBM lowered its computer prices by a minuscule amount and slapped heavy new prices on the software— often charges of thousands of dollars per month.)

A persistent rumor is that IBM fires all its salesmen in a geographic area if a key or prestige sale is "lost," as when M.I.T.'s Project MAC switched over to General Electric computers in the sixties, or when Western Electric Engineering Research Center passed over IBM computers to get a big PDP-10.

Much as some people would like to believe these stories, there seems to be no documentation. You would think one such victim would write an article about it if it were true.

Finally, there is the popular doctrine of IBM's infallibility. This, too, is a ways from the truth. The most conspicuous example was something called TSS/360.

TSS/360 was a time-sharing system— that is, the control program to govern one model of the 360 as a time-sharing computer. According to Datamation ("IBM Phases Out Work on Showcase TSS Effort," Sept. 1, 1971, 59-61), over 400 people worked on it at once for a total of some 2000 man-years of effort. And it was scrapped, a writeoff of some 100 million dollars in lost development costs. The system never worked well enough. Reputedly users had to wait much too long for the computer's responses, and the system could not really compete with those offered elsewhere.

The failure and abandonment of this program is thus responsible for IBM's present noncompetitive position in time-sharing; customers are now assured by IBM that other things are more important. IBM-haters thank their stars that this happened. Cynics think it conceivable that high-power time-sharing was dropped by IBM in order to steer its customer base toward areas it controlled more completely.

Two other conspicuous IBM catastrophes have been specific computers: the 360 model 90 in the late sixties, and a machine called the STRETCH somewhat earlier. Both of these machines worked and were delivered to customers. (Indeed, the STRETCH is said by some to have been one of the best machines ever.) But they were discontinued by IBM as not sufficiently profitable. Thereby is said to have been the "failure." (However, it has been alleged in court cases that these were "knockout" machines designed to clobber the competition at a planned loss.)

B. Negative views of IBM systems.

In the technical realm, IBM is widely unloved because many people think some or all of their computers and programs are either poor, or far from what they should be. The reasons vary.

Some of the people feeling this way are IBM customers, and for a time they had an organized lobby, called SHARE (which also facilitated sharing of programs). Recently, however, SHARE has become IBM-dominated, a sort of company union, according to my sources.

The design of the 360, while widely accepted as a fact of life, is sharply criticized by many. (See "What's wrong with the 360?", p. 71.)

IBM's programs, while they are available for a broad variety of purposes, are often notoriously cumbersome, awkward and inefficient, and sometimes document very badly. However, the less efficient a program is, the more money they make from it. A program that has to be run for an hour generates twice as much revenue than if it did its work in thirty minutes; a program that has to be run on a computer with, say, a million spaces of core memory generates ten times the revenue it would in two hundred thousand.

IBM programs are often thought to be rigid and restrictive.

The complex training and restrictions that go with IBM programs seem to have interesting functions. (See box, "IBM's Control.")

C. Theories of IBM design.

The question is, how could a company like IBM create anything like the 360 (with its severe deficiencies) and its operating system or control program OS (with its sprawling complications, not present in competitors' systems)? Three answers are widely proposed: On Purpose (the conspiracy theory), By Accident (the blunder theory), and That's How They're Set Up (the Management Science theory). These views are by no means mutually exclusive.

The Management Science theory of IBM design is the only one of these we need take up.

The extensive use of group discussion and committee decisions may tend to create awkward design compromises with a certain intrinsic aimlessness, rather than incisively distinct and simple structures. (See Gould's marvelous chapter, "The Meeting," 56-95.)

Their use of immense teams to do big programming jobs, rather than highly motivated and especially talented groups, is widely viewed as counterproductive. For instance, Herbert A. Wolff, in a letter to Datamation (Sept. 1, 1971, p. 13) says a particular program

> "remains inefficient, probably because of IBM's unfortunate habit of using trainees fresh out of school to write their systems code."

There may also be something in the way that projects are initiated and laid out from the top down, rather than acquiring direction from knowledgeable people at the technical level, that creates a tendency toward perfunctoriness and clunky structure.

Thus there may very well be an intentional policy of unnecessary complication (see box, "IBM's Control"). But the way in which goals are set out technical decisions delegated may generate this unnecessary complication.

## THE CATOLO INSIDE STORY

It is unfortunate that Rodgers' remarkable book does not follow the details of IBM's computer designs and politics in the computer age, i.e., since 1955. Later work, perhaps helped by some Pentagon Papers, will have to relate the decision procedures that occurred in this unique national institution to the systems it has produced and the stamp it has put on the world.

## QUICKIE HISTORY OF IBM

IBM appeared in 1911 as the consolidation of a number of small companies making light equipment, under the name C-T-R Company (Computer-Tabulating-Record). This was prophetic, considering how aptly it described the company's future business, and especially prophetic considering that today's stored-program computer was undreamed of at that time.

According to William Rodgers' definitive company biography Think, the company's creator was a shrewd operator named Charles R. Flint, dashing entrepreneur and former gun runner to the South American republics, who in his shrewdness brought in to run the company an incredibly talented, fire-breathing and self-righteous individual named Thomas J. Watson, even though Watson at that time was under prison sentence for his sales practices at another well-known company. The sentence was never served, and Watson went on to preside for many years over a corporation to which he gave his unique stamp.

Watson arises from the pages of Think as a sanctimonious tyrant, hard as nails yet reverently principled in his words; the pillar of fervid, aggressive corporate goals.

IBM was totally Watson's creation. The company became what he admired in others, a mechanism totally obedient to his will and implementing his forceful and inspiringly rationalized convictions with alacrity. As the Church is said to be the bride of Christ, IBM might be characterized as the Bride of Watson, molded to the styles of demandingness, pressure, efficiency and pomp which so characterized that man. But the ideas flowed from Watson alone, except for a few confidantes who received his nod. The company is vastly bigger now, and slightly more colorful, in a muted sort of way; but it is still the stiff and deadly earnest battalion of his dream.

Because of Watson's background as salesman, he made Sales the apex of the corporation. The salesman had the most prestige within the company and could make the most money; below that was administration, below that, technical staff.

Watson eliminated the meat-slicing machines, and pushed the product line based on punched cards developed by IBM's first chief engineer, Herman Hollerith. According to Rodgers, it was impetus from the Depression, and the new bookkeeping requirements of Roosevelt's remedies, that skyrocketed the firm uniquely during the depths of general economic catastrophe, till Watson came to draw the highest salary of any man in the nation. In 1934 his income was $364,432 (says Rodgers, not the author of Think), was second with $331,214). Watson had neatly arranged to get 5% of IBM's net profit.

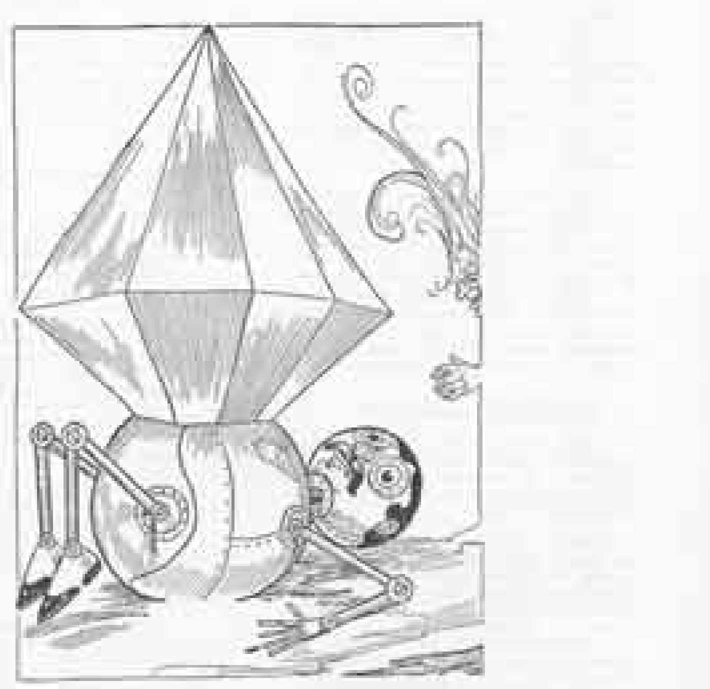
While IBM participated in the creation of certain early computers, it is interesting that Watson dismissed Eckert and Mauchly when they came around after World War II trying to get backing for their ENIAC design, in certain ways the first true electronic computer. Eckert and Mauchly went to Remington Rand, and the resulting Univac was the first commercial computer.

However, IBM bounced back very well. If there was one thing they knew how to do it was sell, and when they brought out their computers it was practically clear sailing. (The Univac I was the first of many computers to be delayed and bogged in the completion of its software, and this considerable setback helped IBM get the lead very quickly; they have never lost it since.)

In the early sixties the IBM 7090 and 7094 were virtually unchallenged as the leading scientific computers of the country. But IBM in the late sixties almost relinquished the fields of very big computers and time-sharing to other companies, and their computers are not regarded as innovative. Nevertheless, IBM's Systems 360 and 370, despite various criticisms, have been very successful; thousands of them are in operation around the globe, far more than all their rivals' big computers all put together. This despite the fact that some of these systems have failed, including the big Model 91 (an economic failure) and the TSS/360 time-sharing program, a technical catastrophe.

They have from time to time been accused of unfair tactics, and various antitrust and other actions (see "Legal Milestones" box) have required IBM to change its arrangements in various ways. One decree required them to sell the computers that before they had only rented; another decision, to "unbundle," or sell computers separately from their programs (previously "given" away with the computers they rented), is widely believed to have prevented government action on the same matter. Showing characteristic finesse, IBM thereupon lowered the computer prices almost imperceptibly, then slapped heavy price-tags on the programs that had previously been free.

Recent moves by the government have suggested an especially serious and far-reaching anti-trust suit against IBM, possibly one that might break the company up, with its separate divisions going various ways. However, in today's climate of cozy relations between business and government, it is hard to imagine that such matters would not be settled in IBM's liking. This lends a curious feel to a remark one IBM person has made to the author, to wit, that maybe IBM wants to be broken up. That might be one way of reducing the unwieldiness and interdependency of its product line, in addition to reducing its vast, underutilized personnel base. (Another angle: Acting Attorney General Bork has expressed the view that IBM is big only because its products and management are wonderful, so the antitrust case may simply evaporate during the rump days of the Nixon incumbency.)



An interesting aspect of IBM publicity is its stress on status. Publicity photographs often show a subordinate seeking advice from a superior. IBM ads appeal to the corporation president in all of us— either Doing It Alone (taking a long walk over an Executive Decision) or soberly directing a lesser employee. In one extraordinary case, we see worshipful minions at the feet of a Teacher implausibly situated in the corner of a prison yard.

# The 360 and Standardization

IBM announced a number of worthy objectives when the 360 line was announced in 1964. IBM should certainly be thanked for at least their lip service to these noble goals.

1. "One machine for all purposes, business and scientific." (Thus the name "360," for the "full circle" of applications.) By "business" this mainly meant decimal, at four bits a digit. Actually this meant grafting 4-bit decimal hardware to an otherwise normal binary computer, and making both types of users share the same facility.

2. "Information storage and transmission will be standardized." The 360 was set up to handle information 4 bits at a time, 8 bits at a time, 16, 32, and 64 bits at a time. (The preceding standard had been 6, 18 and 36 bits at a time.)

In their 360 line, IBM also replaced the industry's standard ASCII code with a strange alphabetical code called EBCDIC ("Extended Binary Coded Decimal Information Code"), ostensibly built up from the 4-bit decimal code (BCD), but believed by cynics to have been created chiefly to make the 360 incompatible with other systems and terminals.

3. "360s will all look alike to the program; thus programs can be moved freely from machine to machine."

Unfortunately this compatibility has been undermined by numerous factors, especially the variety of operating systems, including half a dozen major types, and the language processors, intricately graded according to computer size. Both these factors tend to make changes necessary to move programs between computers. While one effect of this "standardization" has indeed been to facilitate the moving of programs from small computers to big ones, a more important effect has perhaps been to make it hard to move from a big computer to a smaller one. Note the usefulness of this apparent paradox to IBM's marketing.

The secret of it all, of course, lies in IBM's keen understanding of how to sell big computers. The comptroller, or somebody like him, generally makes the final decision; and if he is told that the one computer will run "all kinds" of programs, that naturally sounds like a saving. Shades of the F-111. (Businessmen's trust and respect for IBM is discussed elsewhere in this article.)

# THE BIG QUESTIONS

Between the trade press and dozens of acquaintances in the field, almost everything I hear about IBM and its products is negative (say five or ten to one)— except from people who work or have relatives there.

Perhaps it's just sour grapes. Or the authority-hating character of research types. Or selective reading.

Or perhaps there really is something sinister.

The major questions are these.

1. How clean is their salesmanship?
2. Are their systems unnecessarily difficult or cumbersome on purpose?
3. How deep is their system of entrapment and forced commitment of the customer? How necessary are the de-standardizations and the constant changes?
4. Do they have a final liberating vision? Do they really, after all, intend to bring about a day when life is easier for people? When the difficulties of present-day computer systems, especially theirs, wither away? I think that history's judgment on IBM in our time may narrow down to that simple question.

(In this light it is not hard to understand IBM's stand on software copyrights vs. patents. IBM is against programs being patentable, which would cover abstracted properties, but argues in favor of copyright, whose protection is probably more limited to the particulars of a given program. If they have their way, it would be assured that IBM could use any ingenious new programming tricks without compensation, whereas all unnecessary complications of bulky, cumbersome software would be covered in entirety by copyright.)

Finally, it has not been demonstrated that IBM has any general ability to make systems conceptually simple and easy to use. (Two good examples of hard systems are the Mag Tape Selectric and Datatext— easy for programmers, but hardly for secretaries.) There seems to be no emphasis on elegance or conceptual simplicity at IBM. Those who adopt such a philosophy (such as Kenneth Iverson) do so on their own.

As mentioned earlier, this has something to do with the fact that individuals generally use IBM's systems because they have to, being employees or clients of the firms that rent IBM equipment, so there is no impetus to design programs or systems to run on simple or clear-minded principles, or draw out intricate systems so they can be used easily.

4. THE IMAGE.

It is hard to analyse images, corporate or personal. They are often received in such different ways by different populations. But there may be a commonality to the IBM image as generally seen. The image of IBM involves some kind of cold magic, a brooding sense of sterile efficiency. But other things are percolating in there. If we slice that connotation of efficiency aside, the IBM image seems to have two other principal components: authoritarianism and complacency. It is this mixture that longhairs will naturally find revolting. This same combination, however, may be exactly what it is that appeals to business-management types.

IF YOU REALLY WANT IT...

you can get character-by-character responding systems on IBM computers. The new Stock Exchange system uses a "Telecommunications Access Method" permitting non-IBM terminals to respond character-by-character, just as systems for non-computer people should.

Trying to use this input-output program on your local IBM computer is another problem, though. Aside from program rental costs, there is the problem of its compatibility with the whole line of IBM software. Adaptations and reprogramming would probably be necessary up and down the line.

THE FUTURE

What will IBM do next?

Speculation is almost futile, but necessary anyhow. The prospects are fascinating if not terrifying.

No one can ever predict what IBM will do, but trying to predict IBM's actions— IBM-watching is something like Kremlin-watching— is everybody's hobby in the field. And its consequences affect everybody. With so many things possible, and determined only in the vaguest way by technical considerations, the question of what IBM chooses to do next is pretty scary. Because whatever they do we'll be stuck with. They can design our lives for the foreseeable future.

We know that in the future IBM will announce new machines and systems, price changes (both up and down) in fascinating patterns, rearrangements of what they will "support," and changes in the contracts they offer (see box, "IBM's Control"). Occasional high-publicity speeches by IBM high officers will continue to be watched with great care. But mainly we don't know.

IBM's slick manufacturing capabilities mean that practically any machine they wanted to make, and put on a single chip, they could, and in a very short time. (The grapevine has it that the Components Division, which makes the computer parts, has bragged within the company that it doesn't really need the other divisions any more— it could just put whole computers on teeny chips if it wanted to.)

In this time of the 370, things are for the moment stable. The 370 computer line is still their main marketing thrust. Having sold a lot of 370 computers (basically sped-up 360s), their idea is at the moment to sell conversion jobs to adapt the 370 to run the new "Virtual System" control program (VS or OS/VS or various other names). This system (which is, incidentally, widely respected) makes core memory effectively much larger to programs that run on it. This effectively encourages programmers to use tons of core, by means of virtual memory, essentially getting people in the habit of programming as if core were infinite. This extension of apparent memory size distracts from any inefficiencies of both locally written programs and IBM programs, thus tending to increase use and rental charges.

When that marketing impetus runs out we'll see the next thing.

The other new IBM initiative is with smaller machines, the System 3 and System 7, being pushed for relatively small businesses. That is where they see another new market. How easy and useful their programs are in this area will be an important question.

With the System 7, a 16-bit minicomputer for $17,000, IBM has at last genuinely entered the minicomputer market. (Balancing its speed and cost against comparable machines, we can figure the IBM markup as being about 30%, which is typical.)

In addition, it is rumored that IBM might put out a tiny business mini, to sell out of OPD (Datamation, Dec 72, 139.) But really, who knows.

In addition to this huge-memory strategy for its big machines, and the starting foray into specialized mini systems, there is the office strategy and "word processing."

IBM has conceptually consolidated its various magic-typewriter and text services under the name of "word processing," which means any handling of text that goes through their machines. This superficially unites their OPD efforts (typewriters and dictation machines) with things going on in DPD, such as Datatext, and allays interdivisional rivalries for awhile. Also, by stressing the unity of the subject matter, it leaves the door open for later and more glamorous initiatives, such as hypertext systems (see "Carmody's System," flip side).

In other words, the foot is in the door. Mr. Businessman has the idea that automatic typing and things like that are IBM's special province.

Few firms anywhere have the confidence to advertise generically a product which is made by others as well, as in IBM's "Think of the computer as energy" series.

## SHOULD INDIVIDUALS FEAR IBM?

Even if it is true, as Anonymous says (see Bibliography) that IBM intimidates people and keeps its enemies from getting jobs at IBM-oriented establishments, that's not the end of the world.
Grosch, Gould, Rodgers and McClark are alive and working.
Extramural harassment like that employed by GM against Nader, for example, has not been reported.

END OF THE DINOSAURS?

To a very great extent, IBM's computer market is based on big computers run in batch mode, under a very obtrusive operating system.

Many people are beginning to notice, though, that many things are more sensibly done on small computers than on big ones, even in companies that have big computers. That way they can be done right away rather than having to wait in line. Is this the mammal that will eat the dinosaur eggs?

On the other hand, a very unfortunate trend is beginning to appear, an implicit feud within large organizations, which may benefit IBM's big computer approach. Those who advocate minicomputers are being opposed by managers of the big computing installations, who see the minis as threatening their own power and budgets. This may for a long time hold the minis back, perhaps with the help and advice of computer salesmen who feel likewise threatened. But there will be no holding back the minis and their myriad offspring, the microprocessors (see p. 44). And the inroads should begin soon.

(Others are growing to know and love true high-capacity time-sharing as a way of life, like that offered for DEC, GE and Honeywell machines. This, too, may begin to have derogatory effects on IBM's markets.)

Finally, it must be noted that almost all big companies have computers, usually IBM computers, and so an era of marketing may well have ended. It may be possible for IBM to go on selling bigger and bigger computers to the customers who already have them, but obviously this growth can no longer be exponential.


# A GROSCH IRONY

Herb Grosch, now editorial director of Computerworld, is perhaps IBM's worst enemy. Once he worked for old man Watson, and was the only IBM employee allowed to have a beard. Now, among other things, he gives speeches and testimony wherever possible about the Menace of IBM, at conferences, at governmental hearings, and in letters to editors.

Yet IBM's main computer sales strategy today is to stress the advantages of big computers with lots of core memory (and persuade you you don't want highly interactive systems or independent minicomputers).

And the fundamental rule stating the advantages of big computers is called Grosch's Law, formulated years ago by none other. See p.

A LITTLE GEM FROM THE IBM SONGBOOK
(Who says IBM doesn't encourage individualism?
To the tune of "Pack Up Your Troubles
in Your Old Kit Bag.")

"TO THOMAS J. WATSON, President, IBM"

Pack up your troubles— Mr. Watson's here!
And smile, smile, smile.
He is the genius in our IBM
He's the man worth while.
He's inspiring all the time,
And very versatile— oh!
He is our strong and able President!
His smile's worth while.

"Great organizer and a friend so true."
Say all we boys.
Ever he thinks of things to say and do
To increase our joys.
He is building every day
In his outstanding style— so
Pack up your troubles, Mr. Watson's here
And Smile— Smile— Smile.

(As a nostalgic public service Advanced Computer Techniques, Inc., of Boston, gave away LPs of IBM songs at the '68 SJCC. They might just have some left...)

---

NEW CHIPS...

IBM can put pretty much anything on a single chip, to make a functioning machine the size of a postage stamp; but so can a lot of other companies.

The question really becomes whether what goes on that chip is a worthwhile machine that does what people want.

...BUT THE SAME OLD BLOCK?

It is by no means clear that IBM has any general ability to make computer systems easy to use.

This is a psychological problem.

As a corporation they are used to designing systems that people have to use by fiat, and must be trained to use, contributing to the captivity and inertia of the customer base. Thus the notion of making things deeply and conceptually straightforward, without special jargon or training, may not be a concept the company is ready for.

---

SOME DIVISIONS OF IBM you may hear about

| | |
|---|---|
| OPD | Office Products Division. Typewriters, copiers. |
| DPD | Data Processing Division. Computers and accessories. |
| FSD | Federal Systems Division. Big government contracts, NASA stuff, and who knows what. |
| ASDD | Advanced Systems Development Division. Very secret. |
| Components Division. | Makes parts for the other guys, including integrated circuits. |
| SRA | Science Research Associates, Chicago. Publishes textbooks and learning kits. |
| Watson Lab | T.J. Watson Research Laboratory, Westchester County, north of New York City. Theoretical and lookahead research. |

"THERE IS A WORLD ELSEWHERE."
— Coriolanus

There is no way to escape IBM entirely. IBM mediates our contacts with government and medicine, with libraries, bookkeeping systems, and bank balances. But these intrusions are still limited, and most of us don't have to live there.

There are many computer people who refuse to have anything to do with IBM systems. Others, not so emphatic, will tell you pointedly that they prefer to stay as far away from IBM computers as possible. If you ask why, they may tell you they don't care to be bothered with restrictive, unwieldy and unnecessary complications (the JCL language is usually mentioned). This is one reason that quite a few people stick with minicomputers, or with firms using large computers of other brands.

It is possible to work productively in the computer field and completely avoid having to work with IBM-style systems. Many people do.

---

# IBM LEGAL MILESTONES

The famous Consent Decree of January 1956. (In a consent decree, an accused party admits no guilt but agrees to behave in certain ways thereafter.) In response to a federal anti-trust suit, IBM agreed to:
- sell as well as lease its computers, and repair those owned by others;
- permit attachments to its leased computers;
- not require certain package deals;
- license various patents;
- not buy up used machines;
- and get out of the business of supplying computer services, i.e., programming and hourly rentals.

Unbundling decision, late sixties. While this was not a government action but a an internal policy decision by the company, it somehow had a public-relations appearance of official compulsion. Beset by pressures from makers of look-alike machines, users of competitive equipment, and the threat of anti-trust action, IBM decided to change its policy and sell programs without computers and computers without programs. Delight amongst the industry turned to chagrin as this became recognized as a price hike.

The Telex Decision, September '73. Telex Corp. of Tulsa was awarded $352,500,000 in triple damages (since reduced) for losses attributed to IBM's "predatory" pricing and other marketing practices.
Much more important, IBM was required to disclose the detailed electronics required to hook things to their computers and accessories within sixty days of announcing any. This was a great relief for the whole industry. Essentially it meant IBM could no longer dictate what you attach to their machines. Unfortunately, it is not clear whether this will stand.

But what we're waiting to hear about is whether the Nixon Justice Department is, or is not, going to press the big anti-trust suit which has been long brewing, at the persistent request of other firms in the industry.

---

"THINK OF THE COMPUTER AS ENERGY,"
says a recent series of IBM ads.
But in terms of monopoly, price, and the world's convenience, there would seem only one way to complete the analogy, viz.:

"THINK OF THE COMPUTER AS ENERGY."

"Think of IBM as King Faisal."

# FOLDING OF THE IBM UMBRELLA

For a long time, during the sixties, IBM's high prices provided an environment that made it easy for other companies to come into the field and sell computers and peripherals. These high prices were referred to as "the IBM umbrella."

However, this era has ended. IBM now cuts prices in whatever areas it's threatened. A brief flourishing of companies making add-on disk and core memories for IBM computers has become precarious; not only will IBM now cut prices, but they have shown themselves still disposed to invent new restrictive arrangements (the recent "virtual memory" announcement for the 370 claimed that the program will only work on IBM disk and core).


BIBLIOGRAPHY

Harvey D. Shapiro, "I.B.M. and all the dwarfs," New York Times Magazine, July 29, 1973, 19-36.

An objective, factual article, sympathetic to IBM— although it drew at least one irate letter from an reader who didn't think it sympathetic enough.

"IBM: Time to THINK Small?" Newsweek, October 1, 1973, 85-86.

Frank T. Cary, letter to the editor, Newsweek, Oct. 15 73, p. 4. A snappish reply to the above by the IBM Board Chairman, who evidently didn't like the article very much.

Robert Samuelson, "IBM's Methods," New York Times Sunday financial section, June 3, 1973, p. 1.

→ This article gives a unique glimpse of some of the interesting things that came to light in the Control Data suit against IBM— citing trial documents never publicly released.

• William Rodgers, Think. Stein and Day, 1969. Subtitled A Biography of the Watsons and IBM.

→ Concentrates on the days before computers. Fascinating profile of Watson, a business tiger, but the view of the corporation in an evolving nation is general Americana that transcends fiction.

Would you believe Rodgers says Watson was the kingmaker who put General Ike in the White House?

Unfortunately, the book has relatively less on the computer era, so the inside story of many of their momentous decisions since then remains to be told.

Haywood Gould, Corporation Freak. Tower (paperback.)

Marvelous, hard to get; Gould thinks IBM quietly bought up all the copies.

The musings of a sophisticated, clever and observant cynic who began knowing nothing about IBM, Gould's wide-eyed observation of its corporate style and atmosphere is a jolt to those of us who've gotten used to it. And he thought it was just another big company!

Anonymous, "Anti-Trust: A New Perspective," Datamation, Oct 73, 185-186.

Richard A. McLaughlin, "Monopoly Is Not a Game," Datamation, Sept. 1973, 73-77.

→ Questionnaire survey intended to test truth of common accusations against IBM. (Discussed in text above.)

W. David Gardner, "The Government's Four Years and Four Months in Pursuit of IBM," Datamation, June 1973, 114-115.

Almost any issue of Computerworld or Datamation, the two main industry news publications, carries articles mentioning complaints about IBM from various quarters on various issues. Datamation's letters are also sometimes juicy on the topic.

Any issue of On Line, a news sheet of the Computer Industry Association, ten bucks a year. (CIA— no relation to the intelligence agency — 16255 Ventura Blvd., Encino, CA 91316.)

T.A. Wise, "I.B.M.'s $5,000,000,000 Gamble," Fortune, Oct 1966.

Daniel J. Sleznick, "Unconventional Systems," Proc. SJCC 1967, 477-481.
Interesting, among other reasons, for the nastiness of the sarcasm directed at IBM and its larger computers.

• William Rodgers, "IBM on Trial," Harper's, May 1974, 79-84.
Continues where Think left off; examines some of the dirt that came out in the Telex case, and other things.

The author regrets not being able to list more articles and books favorable to IBM, but these do not seem to turn up so much. However, here are a few.

A Computer Perspective, by the office of Charles and Ray Eames, Harvard U. Press, $13.

Angeline Pantages, "IBM Abroad," Datamation, December 1972, 54-57.

For an example of the kind of adulation of IBM based on faith, see Henry C. Wallich, "Trust-Busting the U.S.A.," Newsweek, 1 Oct 73, p. 98.

The IBM Songbook, any year— they haven't been issued since the fifties— is definitely a collectible.

# the Computer Fan's Computer Company
# DEC
## The PDPeople

The computer companies are often referred to in the field as "Snow White and the Seven Dwarfs"-- a phrase that stays the same even as the lesser ones (like RCA and General Electric) get out of the business one by one. The phrase suggests that they're all alike. To an extent; but there is one company sufficiently different, and important enough both in its history and in continuing existence, to require exposition here. This is Digital Equipment Corporation, usually pronounced "Deck," the people who first brought out the minicomputer and continue to make fine stuff for people who know what they are doing.

Other computer companies have mimicked IBM. They have built big computers and tried to sell them to big corporations for their business data processing, or big "scientific" machines and tried to sell them to scientists.

DEC went about it differently, always designing for the people who know what they were doing, and always going to great lengths to tell you exactly what their equipment did.

First they made circuits for people who wanted to tie digital equipment together. Then, since they had the circuits anyway, they manufactured a computer (the PDP-1). Then more computers, increasing the line slowly, but always telling potential users as much as they could possibly want to know.

The same for its manuals. People who wrote for information from Digital would often get, not a summary sheet referring you to a local sales office, but a complete manual (say, for the PDP-8), including chapters on programming, how to build interfaces to it, and the exact timing and distribution of the main internal pulses. The effect of this was that sophisticated users-- especially in universities and research establishments-- started building their own. Their own interfaces, their own modifications to DEC computers, their own original systems around DEC computers.

Digital Equipment Corporation, in response to the "Energy Crisis" of 1973, didn't turn out their Christmas tree. Instead they hooked it up to a water wheel they happened to have. Typical.

This policy has made for slow but steady growth. In effect, Digital built a national customer base among the most sophisticated clients. The kids who as undergraduates and hangers-on built interfaces and kludgy arrangements, now as project heads build big fancy systems around DEC equipment. The places that know computers usually have a variety of DEC equipment around, usually drastically modified.

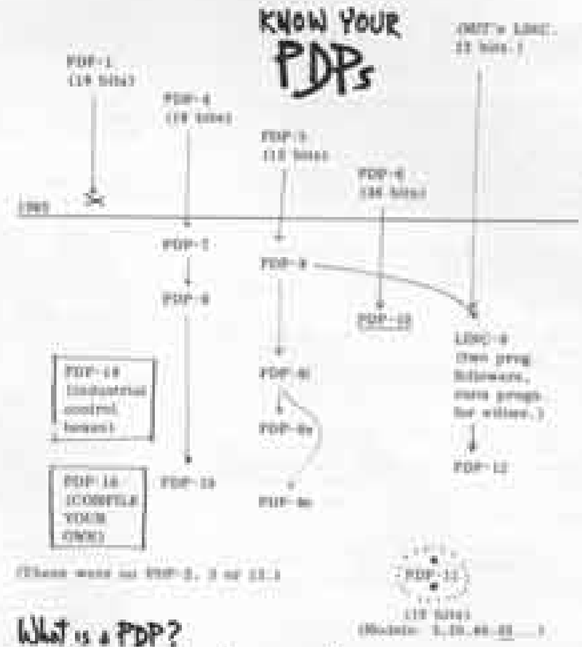
Because of the great success of its small computers, especially the PDP-8, even many computer people think they only make small computers. In fact their big computer, the PDP-10, is one of the most successful time-sharing computers. An example of its general esteem in the field: it is the host computer of ARPANET, the national computer network among scientific installations funded by the Department of Defense; basically this means ARPANET is a network of PDP-10s.

DEC's computers have always been designed by programmers, for programmers. This made for considerable suspense when the PDP-11 did not appear, even though the higher numbers did, and the grapevine had it that the 11 would be a sixteen-bit machine. It proved to be well waiting for (see p. 12.), and has since become the standard sophisticated 16-bit machine in the industry.

An area DEC has emphasized from the first has been computer display (discussed at length on the flip side). Thus it is no surprise that their interactive animated computer display, the GT40 (see p. DM[illegible]) is an outstanding design and success. (And the University of Utah, currently the mother church of computer display, runs its graphic systems from PDP-10s.)

In this plucky, homespun company, whose own president (Olsen) is known by his first name (Ken), it is understandable that marketing pizzazz takes a back seat. This apparently was the view of a group of rebels, led by vice president Ed deCastro, who broke off in the late sixties to start a new computer company around a 16-bit computer design called the Nova-- rumored to have been a rejected design for the PDP-11. The company they started, Data General, has not been afraid to use the hard sell, and between their hard sell and sound machine line they've seriously challenged the parent company.

But Digital marches on, the Computer Fan's computer company. If IBM is computerdom's Kodak, whose overpriced but quite reliable goods have various drawbacks, DEC is Nikon, with a mix-and-match assortment of what the hobbyist want. That's pluralism for you.




(There were no PDP-2, 3 or 13.)

**What is a PDP?**
**DEC's trade name for a computer.**

I'm not getting any favors from DEC. I'm just saying about them what people ought to know.

However, I do have grateful recollections of the warmth and courtesy with which people from Digital Equipment Corporation have taken pains to explain things to me, hour after hour, conference after conference.

In the early sixties they had one man in one small office to service and sell all of New Jersey and New York City. But that one guy, Dave Denniston, spent considerable time responding to my questions and requests over a period of a couple of years, and in the nicest possible way, even though there was no way I could buy anything. You don't forget treatment like that.

---

# PERIPHERALS FOR YOUR MINI

Some kinds of peripheral devices, or computer accessories, are always necessary. Only through peripherals can you look at or hear results of what the computer does, store quantities of information, print stuff out and whatnot.

Trying to print lists of available stuff here is hopeless. There are thousands of peripherals from hundreds of manufacturers. If you buy a mini, figure that your peripherals will cost $1500 (Teletype) on up. But maintenance (see p. 55) is the biggest problem. If you buy peripherals from the manufacturer of the computer, at least you can be sure someone will be willing to maintain the whole thing. (Independent peripheral manufacturers will often repair their own equipment, but nobody wants to be responsible for the interface.)

If you want a list see "Table of Mini-peripheral Suppliers," Computer Decisions, Dec 72, 35-6; more thorough poop is offered by Datapro Research Corp., 1 Corporate Center, Route 38, Moorestown NJ 08057.

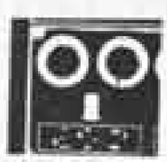
As to the serious matter of disks, an excellent review article is "Disc Storage for Minicomputer Applications," Computer Design, June 1973, 55-66. This reviews both principles of different types of disk drives, and what various manufacturers offer.

Also helpful on disks and tapes: "Taking a Go of Ministorage," by Linda Dermer, Computer Decisions, Feb 74, 32-38. Good recent survey.

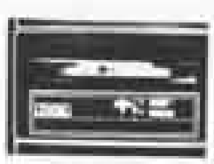
"... a sophisticated electronic computer can store and recall some 100 million 'bits' of information..."
TIME, 14 Jan 74, 56.

Piffle. That's the overall size of the memory, which is utterly independent of the sophistication or general power of the computer itself.

--Billion-bit memories are available, and you could put one on a machine as small as a PDP-8.



Cassette? It's just a DECtape drive, upside down.



Disk Drive for the 11. Most such devices go at 30 spins a second, or 1800 rpm. The heads that read and write information are on moving arms that have to be positioned on the different tracks. (Some disks have a head for every track, which costs more.)

If you have disk drives (which cost) you need a controller ([illegible]). Sigh.

Disk cartridges for this model disk drive.

The brown-coated disk itself is hidden in the plastic case. Nevertheless, they sometimes get scratched or [illegible].

A disk costs $[illegible] and holds up to 5,000,000 characters of information (2.5 million PDP-11 words, which are 16 bits each).

A small line printer. Prints some 300 lines a minute (justice if the lines are narrow). Price around $13,000.

(Of course, terminals are peripherals too.)

A card reader. ... passes to the computer based on the holes punched in the cards.

**TYPICAL PERIPHERALS** (for computer shown on p. 56).

# MAGNETIC RECORDING MEDIA

Any number of different magnetic devices are used for mass storage of symbolic (digital) information; each has its own medium, or form of storage.

The ones which are removable (called "removable media") are of all sorts.

EFFECTIVELY STANDARDIZED BY IBM

1/2-inch magnetic tape.
   Pre-1965: 6 tracks data, 1 track parity.
   Post-1965: 8 tracks data, 1 track parity.
2311 disk
   Stack of removable platters size of a layer cake.
2314 disk
   Same but bigger cake.
disk cartridge
   Plastic case; size of cookie tin, enclosing disk.
floppy disk
   Flexible, card-thin disk enclosed in square 8" envelope.
data cell (not very common)
   Plastic strips pulled out of wedge-shaped tubes arranged in a rotating cylinder. Strip is pulled out of its [illegible], whipped around a drum to make temporary drum memory, returned to case.

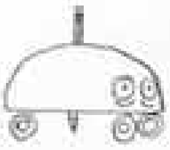
EFFECTIVELY STANDARDIZED BY OTHERS

LINCtape
   1/2-inch tape on a 4-inch reel (fits in pocket), specially coated against friction, developed at Lincoln Labs for LINC computer (see p. 41).
DECtape
   Same size and reel but differently formatted for DEC machines (varies with model). Very reliable. A personal favorite of many programmers.
3M CARTRIDGE
   The Scotch-tape people saw the cassette is unreliable, and offer as an alternative a belt-driven quarter-inch baby costing maybe $1000 without interface.
CRAM (Card Random Access Memory)-- rare. Big pieces of plastic (about four inches by two feet) pulled by notches out of a cartridge and whipped around a drum. National Cash Register.

BARELY STANDARDIZED AT ALL

"Cassettes"-- Philips-type audio-type cassette. Used by various manufacturers in various ways. Sykes, Sycor, DEC, Data General and others have separate, and usually incompatible, systems.

# YOUR TURTLE AND MUSIC BOX



Surely nobody can resist the peripherals offered by General Turtle, Inc., 545 Technology Square, Cambridge, Massachusetts 02139.

The Turtle is a sort of cannister on wheels that takes a pencil down the middle. Attached to your computer, it can be programmed to ramble around drawing pictures, or just do wheelies on the parquetry. $800.

Then the Music Box is $800. It sings in four voices, enough for a lot of Vivaldi, does five octaves and looks to the computer like a Teletype. They will play you samples on the phone (617)661-3773.

For either of these you need a Controller ($1300).

BRAILLE

No joke here. People are still making Braille copies of things by hand. But the way to do it is by computer: the machine can punch out new copies of whatever's stored in it, repeatedly.

A Braille-punching adapter kit is available for the plain 33 Teletype, I believe from Honeywell.

A similar adapter kit for IBM's System 3 is available from IBM.

(It is of interest that an early use of Bemers' COMIT language was with Braille conversion.)

You never know what you'll see next. In 1969 one firm announced a "high-density read-only memory device" which anyone could see was a plain 45 RPM phonograph-- but with digital electronics. And it made sense. But it doesn't seem to have caught on.

# SIMULATION

is an imposing term which means almost anything. Basically, "simulation" means any activity that represents or resembles something. Computer simulation is using the computer to mimic something real, or something that might be, for any purpose: to understand an ongoing process better, or to see how something might come out in the future.

Here again, though, the Science myth steps in to mystify this process, as though the mere use of the computer conferred validity or some kind of truth.

(On TV shows the Space Voyagers stand in front of the "computer" and ask in firm, unnaturally loud voices what will be the results of so-and-so. The computer's oracular reply is infallible. On TV.)

Let there be no mystery about this. Any use of a data structure on a what-if basis is Simulation. You can simulate in detail or crudely; your simulation can embody any theories, sensible or stupid, and your results may or may not correspond to reality.

A "computer prediction" is the outcome of a simulation that someone, evidently, is willing to stand behind. (See "computer election predictions," p. 47.)

These points have to be stressed because if there is one computer activity which is promotionally presented and stressed, it is simulation. Especially to naive clients. There is nothing wrong with simulation but there is nothing supernatural about it either.

Another term which means more or less the same is modelling.

In the loose sense, simulation or modelling consists of calculations about any describable phenomenon-- for instance, optical equations. In optical modelling (and this is how they design today's great lenses), a data structure is created which represents the curvature, spacing, etc. of the separate glasses in a lens. Then "simulating" the paths of individual rays of light through that lens, the computer program tests that lens design for how well the rays come together, and so on. Then the design is changed and tried again.

Another type of simulation, an important and quite distinct one-- is that which represents the complex interplay of myriad units, fanning out the upshots and consequences of intricate premises. In traffic simulations, for instance, it is easy enough to represent thousands of cars in a data structure, and have them "react" like drivers-- creating very convincing traffic jams, again represented somehow within the data structure.

Basically simulation requires two things: a representation, or data structure, that somehow represents the things you're simulating in the aspects that concern you, and then a program does something to those data, that is in some way like the process you're concerned about acting on the things you're modelling. And each event of significance enacted by the program must somehow leave its trace in the data structure.

The line between simulation and other programming is not always clear. Thus the calculation of the future orbits of the planets could be called "simulation."

The most intricate cases, though, don't particularly resemble any other kinds of programs. The intricate movements of physical movements, especially swarms and myriads with mixed and colliding populations, are especially interesting. (In a recent Scientific American article, simulation helped to understand peculiar distensions of stars between galaxies as resulting from normal considerations of inertia and gravitation. (Alar and Juri Toomre, "Violent Tides between Galaxies," Sci. Am. Dec 73, 38-48.))

Models of complex and changing rates are another interesting type. Standing complex things, whose amounts are constantly changing in terms of percentage multipliers of each other, sound easy in principle, but their consequences can be quite surprising. (See "The Club of Rome," p. 48.)

To imagine the kinds of mixed-case myriad models now possible, we could on today's big computers model entire societies, with a separate record describing each individual out of millions, and specifying the probabilities of action and different preferences according to various theories -- then follow through whole societies' behavior in terms of education, income, marriage, sex, poverty, death, and anything else. Talk about the politics and needs in the balance.

Any computer language can be used for some kind of simulation. For simulations involving relatively few entities, but lots of rates or formulas, good old BASIC or FORTRAN is fine. (MAGI's "Synthevision" system, which could be said to "simulate" complex figures in a three-dimensional space, is done in Fortran; see p. DM 31.) For simulations involving a lot of separate objects, special cases and discrete events, TRAC language (see p. 23) is great. If numerous mathematical formulas are involved, and you want to change them around considerably in an experimental sort of way, APL is well suited (see pp. 32).

There are a number of special "simulation" languages, notably SIMSCRIPT and GPSS. These have additional features useful, for instance, in simulating events over time, such as "EVENT" commands which synchronize or dove-tail functions in time (the simulated time). Simulation languages generally allow a great variety of data types and operations on them.

The list-processing fanatics, of course, insist that their own languages (such as LISP and SNOBOL) are best. And then there's PLATO (see p. DM 26), whose TUTOR language is splendid for both formulas and discrete work-- but allows you only 1500 variables, total (60 bits each).

The thing is, any set of assumptions, no matter how intricate, can be enacted by a computer model. Anything you can express exactly can be carried out, and you can see its consequences in the computer's readout-- a printout, a screen display, or some other view into the resulting data structure.

Obviously these simulations (or sometimes "predictions") are wholly fallible, deriving any validity they may have from the soundness of the initial data or model.

However, they have another important function, one which is going to be very important in education and, I hope, general public understanding, as computers get spread about more widely and become more usable.

The availability of simulation models can make things easier to understand. Well-set-up simulation programs, available easily through terminals, can be used as Staged Exploratory Structures and Theoretical Exploration Tools. The user can build his own wars, his own societies, his own economic conditions, and see what follows from the ways he sets them up. Importantly, different theories can be applied to the same setups, to make more vivid the consequences of this or the other point of view.

(Indeed, similar facilities ought to be available for Congress, to allow them to propel a new tax through the population and see who suffers, who gains....)

I should point out here that for this purpose-- Insightful Simulation-- you don't always need a computer. I have in mind the so-called "simulation games," which if well designed give extraordinary insights to the players. Allan Calhamer's brilliant game of Diplomacy, for instance (Games Research, Boston, available from Brentano's, NYC) teaches more about international politics than you could suppose possible. I am also intrigued by a game called "Dinner," worked out by a sociologist to demonstrate the development of social structures from a state of random creation, but I haven't played it. (Clark C. Abt, of Abt Associates, Boston, has also done a lot of interesting design here.)

A last point, a very "practical" application: Simulation makes it possible to check things without trying them out in concrete reality. For instance, in the lens-design systems mentioned earlier, the lenses don't have to be actually built to find out their detailed characteristics. Nor is it necessary to build electronic circuitry, now, to find out whether it will work-- at least that's what the salesmen say. You can simulate any circuit from a terminal, and "execute" what it does at any time or in any part with simulated meters. Similarly, when any computer is designed now, it's simulated before it's built; and programs are run on the simulated computer, as enacted within a real computer, to see if it behaves as intended. (Actually there are some hot-wire types who insist on building things first, but one assumes that the more sensible computer designers do this.)

With automobiles it's harder; but GM, for instance, simulates the handling characteristics of its cars before they're ever built-- so that designers can redistribute weight, change steering characteristics and so on, till the handling characteristics come out the way the Consumers seem to like.


BIBLIOGRAPHY

Simulation magazine is the official journal of Simulation Councils, Inc., the curiously-named society of the Simulators. It costs $16 a year from Simulation Councils, Inc., Box 2228, La Jolla CA 92037.

For all I know you get annual membership free with that. I've always wanted to join but it was always too one thing too many; but their conference programs are sensational. Where else can you hear papers on traffic, biology, military hardware, weather prediction and electronic design without changing your seat?


YXYXYXYXYXYXYXYXYXY

## THAT'S WHAT MAKES HORSE RACING

"Simulation" means almost anything that in any way represents or resembles something. Which is not to say it's a useless or improper term. Just a slippery one.

Example: Here are ways we could "simulate" a horse race:

Show dots moving around an oval track on a completely random basis, and declare the first to complete the circuit The Winner.

Assign odds to individual horses, and then use a randomizer to choose the winner, taking into account those odds. (This is how the PLATO "horserace" game works; see p. DM 17.)

Give conditional odds to the different horses, based on possible "weather conditions." Then flip a coin (or the computer equivalent, weighted randomization) to set the "weather conditions," and assign the horse's performance accordingly.

Program an enactment of a horse race, in which the winner is selected on the basis of the interaction of the horoscopes of horse and rider.

Create a data structure representing the three-dimensional hinging of horse's bones, and the interlaced timing of the the horse's gait. (This has been done at U. of Pennsylvania on a DEC 338.) Then have three stick figures run around a track (or the data structure equivalent).

Using a synthetic-photography system such as MAGI's Synthevision (see p. DM 31), create the 3D data structure for the entire surface of a running horse over time, then make several copies of this horse run around a track, and make simulated photographs of it.

And so on.

So don't be snowed by the term "simulation." It means much, little or nothing, depending

# OPERATIONS RESEARCH

is an extension of Simulation in a fairly obvious direction.

If simulation means the Enactment of some event by computer, Operations Research means doing these enactments to try out different strategies, and find the most effective ones.

Operations research really began during World War II with such problems as submarine hunting. Given so-and-so many planes, what pattern should they fly in to make their catching submarines most likely? Building from certain types of known probability, (but in areas where "true" mathematical answers were not easily found), operations researchers could sometimes find the best ("optimal") strategies for many different kinds of activity.

Basically what they do is play the situation out hundreds or thousands of times, enacting it by computer, and using dice-throwing techniques to determine the outcomes of all the unpredictable parts. Then, after all outfits have done their thing, the program can report on what strategies turned out to be most effective.

Example. In 1973 the Saturday Review of something-or-other printed a piece on the solution, by OR techniques, of the game of Monopoly. Effectively the game had been played thousands of times, the dice thrown perhaps millions, and the different "players" had employed various different strategies against each other in a varying mix. Always Buy, Buy Light Green, Utilities and Boardwalk, etc.

A complete solution was found, the strategy which made (over many plays) to work best. I forgot what it was.

Using another technique, the game of football was analyzed by Robert E. Machol of Northwestern and Virgil Carter, a football personage. Their idea was to test various maxims of the game, to find out which common rules about beneficial plays were true. What they did was replay fifty-six big-league football games on a play-by-play basis, see the outcomes, and see which circumstances proved most advantageous on the average. (You should the reprint (Operations Research, a recent year), and being totally ignorant of football can comment none of the findings. Anyhow, that's where to look. [illegible])

The earlier explanation of Operations Research wasn't quite right. It's any systematic study of what works best. Computers can help.


BIBLIOGRAPHY

Irvin R. Hentzel, "How to Win at Monopoly," Saturday Review of Science, Apr 73, 44-8.

Virgil Carter and Robert E. Machol, "Operations Research on Football." Operations Research, March 1971, 541-544.


# GREAT ISSUES

Until now, the obscurity of computers has kept the public from understanding that anything like political issues were involved in their use. But now a lot of things are going to break. For instance--

## WHITHER THE FBI?

J. Edgar Hoover's recent death raised a very serious problem. What about all those files he had been keeping? Responsible critics of the FBI, such as Fred J. Cook, have claimed that Hoover's policy basically consisted of chasing lone punks (like Dillinger, Bonnie and Clyde), harassing political dissenters, and keeping vast unnecessary records on innocent citizens-- thus virtually creating the vast network of organized crime in America, which stays off the police blotters. Thus the question of the FBI Succession was an important one.

The question has been answered. In July 1973 Nixon appointed Clarence Kelley, police chief of Kansas City. After the previous groupe-- for instance, Nixon's seeming to offer the post to Judge Byrne while he was presiding over the Ellsberg trial-- this looked to the press like a staid and uncontroversial resolution. But was it?

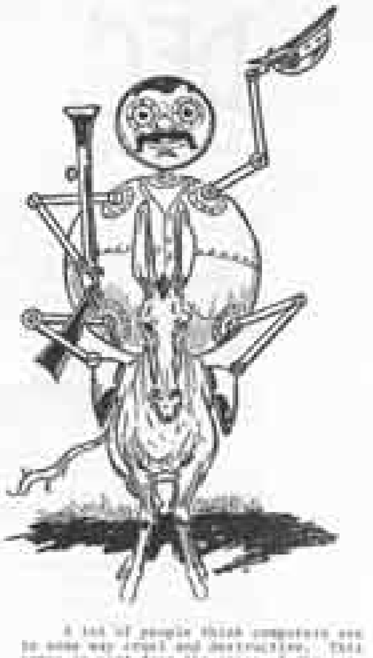
Kelley certainly is aware of technology. It seems to be he that put display screens in Kansas City police cars, created the ALERT system (Automated Law Enforcement Response Team) and CIPPS (Computerized Police Planning System), which for your amusement ties into MULES (Missouri Uniform Law Enforcement System). (See Melvin R. Rohrbach, "On-Line Computers Keeping Things Straight," which describes the Kansas City computer setup. Communications, June 72, 12-28.) In a more interesting vein, supposedly the Kansas City department kept computer files on "militants, mentals and activists." (Schwartz article, p. 15.)

What Kelley does is thus of concern to us all. The big question is whether, for all his concern with police automation, he is also concerned with the freedom this country used to be about.

"NECESSITY HAS BEEN THE EXCUSE FOR EVERY INFRINGEMENT OF HUMAN FREEDOM. IT IS THE ARGUMENT OF TYRANTS; IT IS THE CREED OF SLAVES."

EDMUND BURKE

# MILITARY USES OF COMPUTERS



A lot of people think computers are in some way cruel and destructive. This comes in part from the image of the computer as "rigid" (see "The Myth of the Computer," p. 9), and partly because the military use so many of them.

But it's not the nature of a computer, any more than the nature of a typewriter is to type poems or death warrants.

The point is that the military people are going to go technology, and soon on change, and Congress backs it for them.

No way is there room to cover this subject decently. But we'll mention a few things.

The Pentagon, first of all, with its payroll of millions, with its stupendous inventories of blankets and bombs and toilet paper, was the prime mover behind the development of the Cobol business computing language. So a vast amount is spent just on computers to run the military establishment from a business point of view.

Of course that's not the interesting stuff.

The really interesting stuff in computers all came out of the military. The Department of Defense has a branch called ARPA, or Advanced Research and Development Agency, which finances all kinds of technical developments with vaguely military possibilities.

It is thus a supreme irony that ARPA paid for the development of: COMPUTER DISPLAY (the Sketchpad studies at Lincoln Labs; see p. DM 32); TIME-SHARING (e.g., the CTSS system, see p. 45); HALFTONE IMAGE SYNTHESIS (the Utah algorithms; but see all of pp. DM 30-33); and lots more. Some folks might say that proves it's all evil. I can't buy that at all. While they have military applications, that's simply because they have applications in every field, and the military are just where the money is.

Just to enumerate a few more military things--

Command and control-- the problem of keeping track of who's done what to whom, and what's left on both sides, and so on.

It is a solemn irony that the great "NORAD Command and Control Center"-- a grand room with many projectors driven by computer, only something like those in "Dr. Strangelove" and "Fail-Safe"-- may be a prototype for offices and conference rooms of the future.

"Avionics"-- all the electronic gadgets in airplanes, including those for navigation. (A recent magazine piece described how wonderful it felt to fly the F-111-- which has a computer managing the feel of the controls for you.)

"Tactical systems"-- computers to manage battlefield problems, aim guns and missiles, scramble your voice among various air frequencies or whatever they do.

"Intelligence"-- computers are used to collate information coming in from various sources. This is no simple problem-- how to find out what is so from a tangle of contradictory information. Think about it. Don't think about how we got that information.

"Surveillance"-- it can't all be automatic, but various techniques of pattern recognition (see p. DM 11) are no doubt being applied to the immense quantities of satellite pictures that come back. (Did you know our Big Bird satellite either drops back its film pictures by radio, or parachutes them to Droppings?)

Of course, the joke is that all this obsession with gadgets does not seem to have helped us militarily at all. The army seems demoralized, and the navy losing ground to a country that hardly uses bad computers.

[illegible]

Boston welfare recipients have been automatically short-changed for at least 18 years, according to Computerworld (10 Oct 73, p. 3).

A systems analyst recently discovered that the welfare program was not calculating cost-of-living increases on a compound basis, as it should have been, but on a simple increase based each year on an obsolete original figure.

However, it's too late to ask for refunds, and anyway not many welfare recipients take Computerworld.

## A PREVIOUSLY UNPUBLISHED STORY

Not all kids who play with computers are quite as law-abiding as the R.E.S.I.S.T.O.R.S. And the temptations are very strong.

One such youngster went on a high-school field-trip to a suburban Philadelphia police station, and saw a demonstration of the police remote information system.

The police who were demonstrating it, not being computer freaks, didn't realize how simple it was to observe the dial-in numbers, passwords and protocol.

When this lad got home, he merrily went to his computer terminal in the basement and proceeded to enter into Philadelphia's list of most wanted criminals the names of all his teachers.

A few days later a man came to the house from the FBI. He was evidently not a regular operative but a technical type. He asked very nicely if the boy had a terminal. Then the FBI man asked very nicely if he had put in these names. The boy admitted, grinning, that he had. (Everyone in the school knew it had to be he.)

The FBI man asked him very, very nicely not to do it again.

"Of course it didn't do any harm," says the culprit. "I had them down for crimes like 'intellectual murder.' What could happen to them for that?"

Does that make you feel better?

* * * * *

PHILADELPHIANS AND CROOKS PLEASE NOTE:

This happened five or six years ago, and without a doubt the system is by now totally secure and impenetrable. Let's hope.

## LOUSED-UP RECORDS: A CASE IN POINT

The question of "privacy" in the abstract isn't really an issue. Who cares if God sees under your clothes? The problem is what happens to you on the basis of people's access to your records.

Margo St. James is a case in point.

Ms. St. James is a celebrated and most presentable, once well known for her activities with Paul Krassner as "The Realist Nun;" she is now Chairmadam of an organization called COYOTE, campaigning for the decriminalization of prostitution.

She originally had no intention of becoming a prostitute. Rather, she learned that there was a false record of her arrest for prostitution, and despite her efforts to clear her name, the record followed her wherever she tried to get a job. Finally she said the hell with it and did become a prostitute.

(Membership is $5 a year. COYOTE, Box 26354, San Francisco CA 94126.)

## BLACK AND BLUE AND RED ALL OVER

The phone system is bruised and bleeding from the depredations of people who have found out how to cheat the phone company electronically. Such people are called Phone Freaks (or Phreaks); articles on them have appeared in such places as Ramparts, The Realist and Oui. For no clear reason, the electronic devices they use have been given various colorful names:

Black box: device which, attached to a local telephone, permits it to receive an incoming call without billing the calling party; it "looks like" the phone is still ringing, as far as the billing mechanism is concerned.

Blue box: device that generates the magical "inside" tones that open up the phone network and stop the billing mechanism. Possession of a blue box can get you in prison.

As with so many things, the phone system was not designed under the assumption that there would be thousands of electronic whiz-guys capable of fooling around with it. Thus the phone system is tragically vulnerable to such messing around. The only thing they can do is get ferocious laws passed and really try to catch people, both of which are apparently happening. Supposedly it is illegal to possess a tone generator, or to inform anyone as to what the magical frequencies are-- even though a cereal whistle is such a tone generator, and any engineering library is said to have the information.

Red box: device that simulates the signals made by falling coins.

The fact that the names of these devices are given here is not to be construed as in any sense approving of them, and anybody who messes around with them is a fool, playing with region.

Even if people were entitled to steal back excess profits from the phone company-- the so-called "people's discount"-- the trouble is that they louse things up for everyone. We have a beautiful and delicate phone system, one that stands ready to do wonderful things for you, including bring computer service to your home; even if, for the sake of argument, it is run by dirty rats, messing around with it is like poisoning the reservoir for everybody.

## "DATA BANKS"

The term "data bank" doesn't have any particular technical meaning. It just refers to any large store of information, especially something attached to a computer.

For instance, at Dartmouth College, where the social scientists have been working hand-in-hand with their big time-sharing project, an awesome amount of data is already available on-line to the social sciences. The last census, for instance, in detailed and undigested form. Suppose you're at Dartmouth and you get into an argument over whether, say, divorced women earn as much on the average as women the same age who have never been married. To solve: you just go to the nearest terminal, bat in a quick program in BASIC, and the system actually re-analyzes the census data to answer your question. If only Congress had this!

The usefulness should be evident.

Because of the way census data is handled, now, it is not possible to ask for the records of a specific individual. But this kind of capability leads to some real dangers.

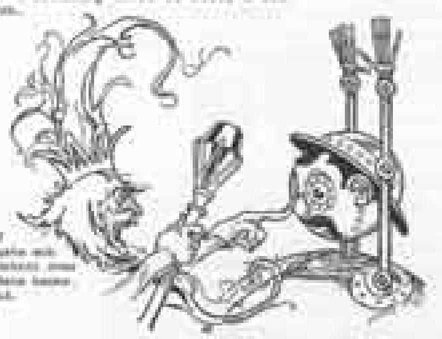
There is a lot of information stored about most individuals in this country. Credit information, arrest records, medical and psychiatric files, drivers' licenses, military service records, and so on.

Now, it is not hard to find out about an individual. A few phone calls from an official-sounding person can ascertain his credit rating, for instance. But that is very different from putting all those records together in one place.

The potential for mischief lies in danger to individuals. Persons up to no good could carefully investigate someone through the computer and then burglarize or kidnap. Someone unscrupulous could look for rich widows with 38-year-old unmarried daughters. Organized crime could search for patsies and strong-arm victims.

In the face of this sort of possibility, computer people have been worrying for years; noteworthy is the study by Alan Westin that originally sounded the alarm, and his two-reasoning follow-up study of some data-gathering organizations (see bibliography). But the scary data banks, the ones that evidently keep track of political dissenters, aren't talking about what they do (see Roberts piece).

Basically, the two greatest dangers from data banks are organized crime and the Executive branch of the Federal Government-- assuming there is still a distinction.



Imagine if the Watergate mob had had control over national data banks.

It may seem odd, but Nixon has said he is concerned about computers and the privacy problem. Cynics may joke about what his concern actually is; but a more credible stand was taken by vice-president Ford at the 1974 National Computer Conference. Ford expressed personal concern over privacy, particularly considering a proposed system called FEDNET, which would supposedly centralize government records of a broad variety.

Not mentioned by Ford was the matter of NCIC, the National Crime Information Center. This will be a system, run by the FBI, to give police anywhere in the country access to centralized records. THE QUESTION IS WHAT GETS STORED. Arrest records? Anonymous tips? (It would be possible to frame individuals rather nicely if a lot of loose stuff could be slipped into the file.)

Many people seem to be concerned with preserving some "right to privacy," which is certainly a very nice idea, but it isn't in the Constitution; getting such a "right" formalized and agreed upon is going to be no small matter.

But that isn't what bothers me. Considering recent events, and the character of certain elected officials whose devotion to, and conception of, democracy is lately in doubt, things are scarcely so abstract as all that. Considering how helpful our government has been to brutal regimes abroad-- notably the Chile overthrow, which some say was run from here (and which could openly arrange for detention just as John Mitchell did)-- we can no longer know what use our information may find in this government. Tomorrow's Data Bank may be next week's Enemies List, next month's Protective Custodial Advisory-- and next year's Termination List. (I don't know if you saw Robert Mardian's face on the Watergate hearings, but that chilled my blood.)


Heather M. David, "Computers, Privacy, and Security." Computer Decisions, May 74, 44-48.
Alan F. Westin, Privacy and Freedom, 1967.
Alan F. Westin and Michael A. Baker, Databanks in a Free Society: Computers, Record-Keeping and Privacy. Quadrangle, $12.50.
"Landmark Study of Computer-Privacy Problems Completed." CACM, Dec 72, 1096-7.
Condensed review of Westin & Baker.
Herman Schwartz, review of Westin & Baker book. NYTimes Book Review, 8 July 73, 18-20.
Notes that the opinions of Westin and Baker is based on their ignorance of various "much-feared information centers" already maintained by the government.
Stanton Wheeler (ed.), On Record: Files and Dossiers in American Life. Russell Sage Foundation (NYC), $15.
"Tax Records: First the Farmers; Then?" Datamation, Dec 73, 125-126.
"How Safe Are Those Fair Credit Dossiers?" Datamation May 73, 120-121.
Paul Armer, "Computer Systems and the Issue of Privacy: How Far Away is 1984?" Datamation, Dec 71, 34-35.


"And the rocket's red glare,
The bombs bursting in air,
Gave proof through the night
That our flag was still there.

"Oh, say, does that star-spangled banner yet wave
O'er the land of the free and the home of the brave?"

— F.S. Key

## THE ABM

THE ABM

Its name has kept changing, possibly to lull the public, possibly to gull the Congress. Anyhow, would you believe a system, totally controlled by computers, designed to shoot down oncoming missiles? If you would, read on.

It's been called Nike-X, Safeguard and goodness knows what. It's even been called a "thin shield"-- Sentinel, huh? Perhaps Congress would pay more if they called it the Turtle Mk.1. But generally we refer to it as the ABM (Anti-Ballistic Missile). It's the anti-missile missile people have talked about, and in it lie many interesting morals, possible comparisons, etc., for which there is no space here.

Western Electric is the prime contractor. They're the manufacturing arm of the telephone company, remember, the same people who make the Princess phone. Of the hundreds of millions of dollars they are taking in on this project, much of it has to go back out-- to Univac, which makes the computers; to Bell Labs, which guides the project. . . Where Whippany, N.J. facility is totally given over to it; to the rocket-builders and so on.

The system is a horror.

Note that in telling you this I am drawing only on information that is publicly available, and drawing conclusions from it the way one usually draws conclusions.

Here is how the great ABM is supposed to work.

Immense radars slow-scan the horizon looking for possible reflections that might be intercontinental missiles.

The radar images are forever constantly analyzed by computers, using every trick of Pattern Recognition (see p. DM/[illegible]).

Aha! Something is coming.

Yes, yes, I'm quite sure now, says the computer. We have fifteen minutes.

Great doors swing open, and a long phallic shape arises. It has jagged outside fins, inherited from the smaller anti-aircraft Nike (we say Nikey) rockets that preceded it. This missile is called the Spartan.

It takes off.

The computer system is tracking the incoming missile. Here it comes-- it's coming now-- the Spartan is turning, going faster and faster-- they're coming together--

Incoming missile speed: maybe 15,000 miles an hour. Spartan speed: maybe 10,000, who knows. In these few minutes the Spartan has gone 400 miles.

How's your tennis?

Can you hit a tennis ball fired out of a cannon?

But now comes the good part.

The Spartan goes off. Yes! It has contains an atomic bomb.

If it goes off within five miles of the attacking missile, the hope is that the attacking missile's thermonuclear warhead will get cooked on one side and misfire. So it lands in Times Square, just breaks a few buildings and spreads radioactive contamination.

But wait.

What if Spartan missed.

Oops, sorry, Montreal.

Never fear! Have you forgotten Sergeant York? Have you forgotten the Alamo?

There is another missile. It is called Sprint. It is shaped like the point of a pencil. It is almost all propellant. When the great computers realize that the bad guy has gotten through, up goes Sprint! Sprint is eloquently called the "terminal defense system." It only has a couple of minutes.

Brighter than a thousand suns! Sorry, Scarsdale. Can't win 'em all.

If you find this description mind-boggling, that's because it is. Anybody who imagines that this project, in which billions of your dollars have already been spent, can work, is a mindful thinker indeed.

Even if missiles stayed like they were in the good old days of 1960, the helpless clunkers they had to fuel up just before the shoot, the likelihood of the h-sets ABM enterprise they ought to was pretty low. (Supposedly DARPA was hoping that Spartan and Sprint could be replaced with phlogiston, Fry-in-the-sky laser beams, mopping down all comers with ray-pointing plate under computer control-- but that is said to have been abandoned.)

But even given, and only for the sake of argument, the feasibility of Spartan-Sprint for first-intercontinental shots, look what's happening now.

MIRVs and FOBs.

MIRV (Multiple Independently Targeted Re-entry Vehicle) basically means Multiple Warheads. One rocket can carry all these little guys, see, that fan out when it gets near the target, and each one goes to its own target city or installation. FOB, or Fractional Orbital Bombardment System, just means that they send the thing into an orbit around the world, and the warheads come in from the opposite side. Any side. Meaning that all those radars pointed at Russia would make good drive-in movie screens.

ABM is sort of a dead duck: the one face-saving installation is in North Dakota, and there won't be any others. But one wonders how such things could ever be funded. But then again I remember once hearing Eric Sevareid, whom some call a liberal, pontificate on this subject. "They describe it as a 'thin shield,'" he said. "Why can't we just spend a few billion more and get complete protection?" Otherwise taking people, if fooled by the technologists, will believe anything.

But the ABM is a beautiful example of top-down planning-- like the Vietnam war. I imagine that the Sprint came about something like this:

"Garfield, our people in Operations Research have concluded that Spartan won't work."

"Yes, yes, sir."

"Garfield, I want your team to get on it and find something additional that will make it work."

Now move Garfield to his cubicle and calls meetings; and it becomes clear. "Looks now, I can't just say it'll never work, they want something additional, well, I guess it would have to be..." Same as Vietnam: "Now this, they say to search and destroy, I guess that must mean..." Something new, this: the top-down project of the worst sort, where the orders go down, and only news of partial success goes up, rather than the facts of total hopelessness. As in Vietnam.

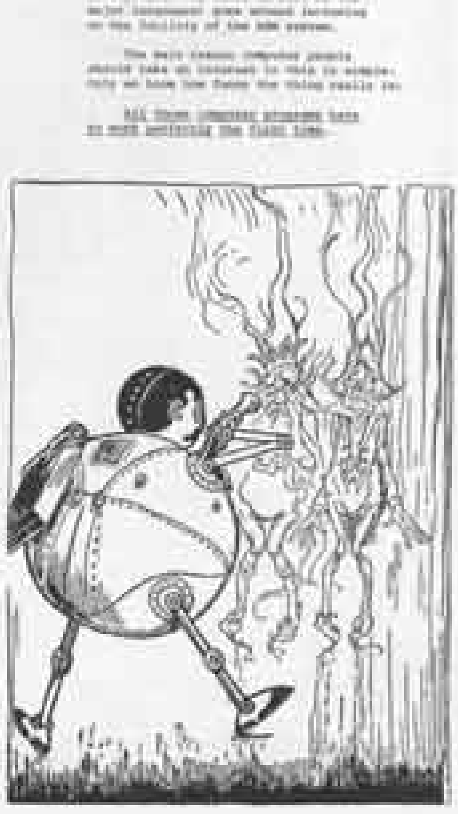
The sophisticated argument is that the ABM effort lets our nation "keep its hand in," "sharpen skills," in case something vaguely like this is ever really needed-- and possible. But this overlooks the overall strategic problem. All this foolishness leads away from the stability of the deterrent; and then may be what keeps everybody alive.

One interesting point to note: a biologist and population geneticist named Sternglass claims it doesn't matter, that human reproduction is so susceptible to radiation poisoning that just the fallout from the ABM defense itself-- a few dozen bombs, say-- would end human reproduction around the planet. But nobody listened to Sternglass.)

Incidentally, an illustrious computer person, Prof. Dan McCracken (author of good programming books on most of the major languages) goes around lecturing on the futility of the ABM system.

The main reason computer people should take an interest in this is simple. Only we know how funny the thing really is:

All those computer programs have to work perfectly the first time.



# THE MITIEST COMPUTER?

The focus of attention in genetics and organic chemistry has for a decade now been the remarkable systems and structures of the molecules of life, DNA and RNA.

DNA is the basic molecule of life, a long and tiny strand of encoded information. Actually it is a digital memory, a stored representation of codes necessary to sustain, reproduce, and even duplicate the creature around it.

It is literally and exactly a digital memory. Its symbols are not binary but quaternary, as each position contains one of four code molecules; however, as it takes three molecules in a row to make up one individual codon, or functioning symbol, the actual number of possible symbols is 64-- the number of possible combinations of four different symbols in a row of three. (I don't know the adjective for sixtyfourishness, and it's just as well.)

The basic mechanism of the system was worked out by Francis Crick and James Watson, who understandably got the Nobel Prize for it. The problem was this: how could living cells transmit their overall plans to the cells they split into? -- and how could these plans be carried out by a mechanical process?

The mechanism is astonishingly elegant. Basically there is one long molecule, the DNA molecule, which is really a long tape recording of all the information required to perpetuate the organism and reproduce it. This is a long helix (or corkscrew), as Linus Pauling had guessed years before. The chemical processes permit the helix to be duplicated, to become two stitched-together corkscrews, and then for them to come apart, unwinding to go their separate ways in daughter cells.


As a tape recording, the molecule directs the creation of chemicals and other cells by an intricate series of processes, not well understood. Basically, though, the information on the basic DNA tape is transferred to a new tape, an active copy called "messenger RNA," which becomes an actual playback device for the creation of new molecules according to the plan stored on the original.

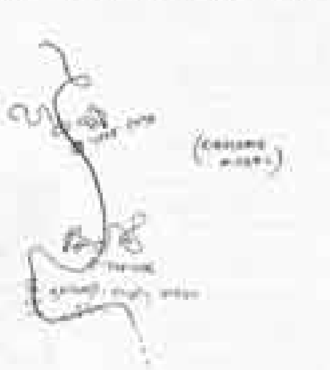
Some things are known about this process and some aren't, and I may have this wrong, but basically the DNA-- and its converted copy, the RNA-- contain plans for making all the basic protein molecules of the body, and anything else that can be made with amino acids. (Those molecules of the body which are not proteins or built of amino acids are later made in chemical processes brought about by those kinds.)

Now well may you ask how this long tape recording makes chemical molecules. The answer, so far as is known, is extremely puzzling.

As already mentioned, the basic code molecules (or nitrogenous bases) are arranged in groups of three. When the RNA is turned on, these triples latch onto the molecules of amino acid that happen to be floating by in the soupy interior of the cell. (There are twenty-seven amino acids, and sixty-four possible combinations of three bases; this is fine, because several different codons of three bases can glom onto the same passing amino acid.)

Now, the tape recording is divided into separate sections or templates; and each template does its own thing. When a template is filled, the string of amino acids in that section separates, and the long chain that results is a particular molecule of significance in some aspect of the critter's life processes-- often a grand long thing that folds up in a certain way, exposing only certain active surfaces to the ongoing chemistry of the cell.

One theory about the mechanics of this is that a sort of zipper slide, called the ribosome, chugs down the tape, attaching the called-for amino acids and peeling off the ever-longer result.



Now, here are some of the funny things that are known about this. One is that there is a particular codon of three bases that is a stop code, just like a period in ordinary punctuation. This signals the end of a template. Another is that the templates on the tape are in no particular order, but distributed higgledy-piggledy. (Geneticists engaged in mapping the genes of a particular species of creature find that the gene for eye color may turn out to be right next to the gene for length of tail-- but where those are really, and what the particular molecules do that determine it, are still mysterious sorts of question.)

Here is some more weird stuff about this.

Large sections of the DNA strand are "dark," it turns out, just meaningless stretches of random combinations of bases that don't mean anything-- or ever get used. This ties in, of course, with the notion that genetic change is random and blind: the general supposition is that genetic mutation takes place a base or two at a time, and then something else activates a chance combination in a dry stretch that turns out to be useful, and this is somehow perfected through successive 1-base changes during the process of successive mutation and evolution.

Amazing use is made of these mechanisms by some viruses. Now, viruses are often thought of as the most basic form of life, but actually they are usually dependent on some other form and hence more streamlined than elemental. Well, some viruses (but not all) have the capacity for inserting themselves in the genetic material: breaking up to the DNA or RNA, unhooking it in a certain place and lying down there, then being duplicated as part of the template, then unhooking themselves and toddling away-- both parent virus and copy. I can't for the life of me think of an analogy to this, but I keep visualizing it as happening somehow in a Bugs Bunny cartoon.

## CONTROL MECHANISMS

Now, all cells are not alike. From the first beginning cell of the organism (the zygote), various splits create more and more specialized, differentiated cells. A liver cell is extremely different from a brain cell, but they both date back by successive splitting from that first zygote. Yet they have different structures and manufacture different chemicals.

One simplification may be possible: the "structure" of a cell may really be its chemical composition, since cell walls and other structures are thought to be special knittings of certain tricky molecules. Okay, so that may reduce the question slightly. How then does the cell change from being an Original (undifferentiated, zygotic) cell to the Specialized cells that manufacture particular other complex chemicals?

One hypothesis was that these other cells have different plans in them, different tapes. But this theory was discarded when John Gurdon at Oxford produced a fresh frog zygote from the intestinal cell of a frog (which accordingly, in due time, became a frog de facto). This proved, most think, that the whole tape is in every cell.

Thus there must be something-or-other that blocks the different templates at different times. (You there, now you're a full-fledged epithelial cell, never mind what you did before) and selects among all the subprograms on the tape.

Much pressing research in molecular biology, then, is concerned with searching for whatever it is that switches different things on and off at different times in the careers of the ever-splitting cells of our bodies. Not to mention those of all other living creatures, including turnips.

## COMPUTERISH CONJECTURES

The guys who specialize in this are usually chemists, and presumably know what they're doing, so the following remarks are not intended as butting into chemistry. However, new perspectives often give fresh insight; and the matters we've covered so far might seem to have a certain relevance.

DNA and RNA, as already remarked, may without distortion be thought of as a tape. Indeed, on this tape is a data structure, and indeed it is a data structure which seems to be involved with the execution of a program-- the program that occurs as the organism's cells differentiate.

There is evidently some sort of program follower which is capable of branching to different selections of (or subprograms) in the overall program, depending on various factors in the cell's environment-- or perhaps its age.

Now, it is one thing to look for the particular chemical mechanisms that handle this. That's fine. On the other hand, we can also consider (from the top down) what sort of a program follower it must be to behave like this. (This is like the difference between tracing out particular circuitry and trying to figure out the structure of a program from how it behaves.)

At any rate, the following interesting conjectures arise:

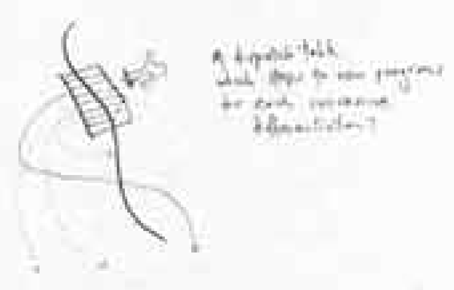
1. The mechanism of somatic reproduction is a subroutining program follower-- not unlike the second program follower of the subroutining display (see p.     That is, it steps very slowly through a master program somewhere, and with each new step directs the blocking or unblocking of particular stretches of the tape.

As the program is in each cell, presumably it is being separately followed in each cell. (This is sometimes called distributed computing.)

2. In each cell, the master program is directing certain tests, whose results may or may not command program branching-- successive steps to new states of the overall program. It may be testing for particular chemical secretions in its environment; it could even be testing a counter.

3. (This is the steep one.) If this were so, we might suppose that this program too was stored on the DNA, in one or more program areas; and it would therefore be necessary to postulate some addressing mechanism by which the program follower can find the templates to open and close. (And perhaps further sections of the program.)

4. Indeed, it makes sense to suppose that such a program has the form of a dispatch table -- a list of addresses in the tape, perhaps associated with specifications of the tests which are to cause the branching.



These wild speculations are offered in the spirit of interdisciplinary good fellowship and good clean fun. Whether (1) and (2) have any actual content, or are merely paraphrases of what is already known or disproven, I don't know; somebody may find the rest suggestive.

Two more observations, though. These are not particularly deep, and may indeed be obvious, but they suggest an approach.

5. There is definitely a Program Restart: to wit, whatever it is that turns an old differentiated intestine cell into a fresh zygote.

6. Cancer is a runaway subroutine.

BIBLIOGRAPHY (for note on left)

Har Gobind Khorana, Willard Gibbs lecture, May 1974, "Progress in the Total Synthesis of the Tyrosine tRNA Gene and Its Control Elements."

---

The above remarks seem to be obsolete. The genetic mechanism really seems to be a list processor (see p.     ) using associative, rather than numerical addressing. The gene is now thought to be divided into four segments, called Promoter, Initiator, gene proper, and Terminator. As I understand it, the promoter and terminator zones contain codes which mean, simply, Start and Stop. The initiator zone, however, is a coded segment which effectively labels the gene. This initiator area contains a chemical code unique for every gene. As suggested in the above article, we may consider both its logical structure-- its mechanisms and effects, considered from a computerman's point of view-- and its chemical structure, or what is really happening. The genes are turned off by grabbing molecules, or repressors, which glom onto the initiator sections of the genes which they have been specifically coded to repress. Research in this area must now find the specific coding of molecules which block and unblock specific genes, and how these fit in the overall graph of metabolism, immunology, development, and so on. If there is anything to make an old atheist uneasy, it is the extraordinary beauty of this clockwork.

From all this, one last speculation creeps forward.

Ivan Sutherland, in considering the structure of subroutining display processors, has noted that as you get more and more sophisticated in the design of a display program follower, you come full circle and make it a full-fledged computer, with branch, test, and arithmetic operations.

If the somatic mechanism should turn out to have a program follower as described, it is not much of a step to suppose that it might have the traits of an actual computer, i.e., the ability to follow programs, branch, and perform manipulations on data bearing on those operations.

In other words, the digital computer may actually have been invented long before von Neumann, and we may have billions of them on our persons already.

It may sound far-fetched, but the mechanisms elucidated at this level are so far-fetched already that this hardly seems ridiculous.

THE COMPUTER FRONTIER

Regardless of what's actually in the cell, it is clear that being able to adapt molecular chemistry, especially DNA and RNA, to computer storage is a beckoning computer frontier.

This would make possible computer memories which are far larger and cheaper than any we now have.

Basically we can separate this into two aspects:

The DNA Readout. This part of the system would create long molecules holding digital information.

The DNA Readin. This would convert it back to electrical form again.

Weird possibilities follow. One is that (if chemical memory is generic, rather than idiosyncratic to an individual's neural pathways) knowledge could be set up somehow in "learned" DNA form, whatever that might turn out to be, and injected or implanted rather than taught. Weird.

As our ability to create clones improves, we could clone new creatures, or genetic "improvements"— which, considering the racehorse and the Pekinese, means "those sorts of non-viable modifications supported in human society." And of course that ghastly stuff about building humans, or semi-humans, having traits that somebody or some organization, ulp, thinks is desirable...

But the real zinger is this one. It might just be a small accidental printout meant to test the facility, or maybe just a program bug—

— but the system could output a virus that would destroy mankind.

BIBLIOGRAPHY


James D. Watson, Molecular Biology of the Gene. Beautifully written; meant for highschool science teachers. But potentially formidable; if so, start with his autobiographical The Double Helix, which is a gas.

Mark Ptashne and Walter Gilbert, "Genetic Repressors," Scientific American, June 1970, 36-44.

S.E. Luria, Life: The Unfinished Experiment. Scribner's.

Lewis Thomas, The Lives of a Cell. Viking, 87. Eloquent writing to popularize, among other things, the New Genetic view that your modern animal cells, and mine, actually contain various fungi and other stray ding-a-lings that slid into one of our ancestors and found useful work, joining the basic genetic program.


# BRAINS & COMPUTERS

It used to be fashionable to say, "The brain is a computer."

But now people say, "The brain is a hologram."

Fashions change.

# THE BRAIN

Almost nothing is known about the brain. Oh, there are lots of picture-books showing cross-sections of brains... Maybe you thought it was just a big cauliflower, but it's full of strings and straps and lumps and hardly anything is known about any of it.

Clinical evidence, of course, tells us that if this or that part is cut out, the patient can't talk, or walk, or smell, or whatever. But that doesn't come close to telling us how the thing works when it does work. The histologists, the perceptual psychologists, the anatomists, are all working at it— with no convergence. Beautiful example: the split-brain stuff, which I just better not even bring up here (see new Maya Pines book, Harcourt Brace).

We used to dissect brains when I worked down in Dr. Lilly's dolphin lab. Dolphin brains are about 1.2 times the size of ours, and Lilly quite reasonably pointed out that this might mean dolphins were smarter than us.

And, of course, the bigger whales even smarter. We had a killer-whale brain in the deepfreeze that was about 2½ feet across. And whales come much bigger than that; the Killer's maybe a quarter the length of the Blue.

(I should point out here that Lilly's publicity on the intelligence of dolphins was a little too good: it somehow didn't get mentioned that dolphins are just very small whales, the only ones you can feasibly keep in a lab. So think of whales as the possible super-smarties, not just dolphins.)

What's that you say? That "brain size isn't what counts"? That's an interesting point.

People with small heads are by and large just as smart as people with big heads. That's one argument.

However, people have much bigger brains than almost any other animals. That indicates something too.

I believe that the only other animals with very big brains are elephants and whales. (An anatomical explanation: the weight is supported on the man by balancing it, on the elephant by a heavy and comparatively inflexible neck offset by a grappling tool, and in the whale by putting it in the front of a torpedo. But most other anatomies couldn't manage a big brain, so they can't evolve one.)

Anyhow, so the scientific question is whether big-brained species are smart. Well, dogs are smarter than rats...

But about these other guys in our league and beyond. How do we know scientifically that "the size of the brain isn't what counts"? Because obviously they're not as smart as we are, people say. Therefore it isn't brain size that counts. The depth of this logic should be evident. (I've even heard people say, "Of course they're not as smart. They don't have guns.")

Pay close attention to an elephant sometime.

Working elephants in India respond to some 500 different oral commands.

Can you think of a 501st thing to ask an elephant to do? (I rather suppose it could oblige.)

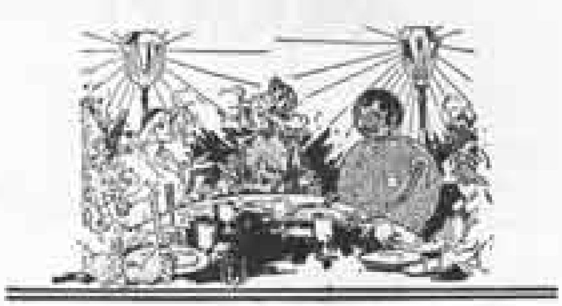
Anyway, the dozen whales I've known personally were smart as hell.

It used to be believed that memory was exclusively a matter of synaptic connections— the gradual closing of little switches between nerve cells with practice.

It is now known that temporary or short-term memory is synaptic, but something else takes place after that. It's believed that after a certain period, and it has something to do with rest and sleep, memories are transferred to some other form, presumably chemical. But how?

My friend Andrew J. Singer has a beautiful hypothesis that wraps it up. His guess is that memories are moved from synaptic storage to DNA (!) storage during dreaming, or more specifically REM sleep. I like that one.

# WHAT NEXT?



By browsing this book you may have more sense of what computers are doing, can do, should do.

What will you do now?

By reading this book in some detail, especially that difficult machine-language stuff (see "Rock Bottom" and "Bucky's Wristwatch," pp. 32-3), or the pieces on specific computer languages (pp. 6-28, 31), you really should be mentally prepared to get into programming, if you dig it.

Maybe you should consider buying your own minicomputer, for a couple of thousand. Or (if you're a parent), chipping in with several families to get one. Or a terminal, and buying (or cadging as cadge can) time on a time-sharing system. Maybe you should start a computer club, which makes it easier to get cast-off equipment; if you're kids, write the R.E.S.I.S.T.O.R.S. (p. 57). If you have a chance, maybe you should take computer courses, but remember the slant these are likely to have. Or perhaps you prefer just to sit and wait, and be prepared to speak up sharply if the computer people arrive ready to push you around. Remember:

COMPUTER POWER TO THE PEOPLE!
DOWN WITH CYBERCRUD!

Computers could do all kinds of things for individuals, if only the programs were available. For instance: help you calculate your tax interactively till it comes out best; help the harried credit-card holder with bill-paying by allowing him to try out different payments to different creditors till he settles on the month's best mix, then typing the checks; WRITING ANGRY LETTERS BACK to those companies that write you nasty letters by computer; helping with letter-writing in general. You'll have to write the programs.

How do you think computers can help
the world?
What are you waiting for?

THE COPPER MAN WALKED OUT OF THE ROCKY CAVERN

# DAMN THAT COMPUTER!

Everybody blames the computer.

People are encouraged to blame the computer. The employees of a firm, by telling outside people that it's the computer's fault, are encouraging public apathy through private deceit. The pretense is that this thing, the computer, is rigid and inhuman (see "The Myth of the Computer," p. 7 ) and makes all kinds of stupid mistakes.

Computers rarely make mistakes. If the computing hardware makes a hardware error in a billion operations, it may be noticed and a repairman called. (Of course, once in a billion operations is once in a thousand seconds, or perhaps every ten minutes. That ought to be mentioned.) Anyhow, innocent gadgetry is not what forces you to make stupid multiple choices on bureaucratic forms; mere equipment isn't what loses your subscription records;

IT'S

THE

SYSTEM.

By system we mean the whole setup: the computer, the accessories that have been chosen for it, its plan of operation or program, and the way files are kept and complaints handled.

Don't blame the computer.

Blame the system; blame the programmer; blame the procedures; best of all, blame the company. Let them know you will take your business to wherever they have human beings. Same for governmental agencies: write your congressman. And so on.

# A Basic Rejoinder

we should all practice and have ready at the tip of our tongues:

WHY THE HELL NOT? YOU'RE THE ONES WITH THE COMPUTERS, NOT ME!

Let's froth up a little citizen indignation here.

# ACCOUNT NUMBERS

In principle we no longer need account numbers.

Now that text processing facilities are available in most (if not all) major computer languages, the only excuse for not using these features is the programmer's notion of his own convenience— not that of the outside customer or victim.

Example. Someone I know got brand new [redacted] and [redacted] credit cards. He made no note of their numbers. Then he lost them both. Duly he reported the losses. Neither service could look him up, they said, without the numbers. Not having used them, he had no bills to check. Even though he was the only person at that address with anything like that name. And why not, pray tell? Either because they were fibbing, or because they had not seen fit to create a simple straightforward program for the purpose. (See Basic Rejoinder, nearby.)

I have heard of similar cases involving major life insurance companies. Don't lose the numbers. Let's all dance to it:

When anything is issued to you,
Write the number down.

# "COMPUTERS" THAT DON'T ANSWER

Few of us can help feeling outrage at the book clubs, or subscription offices, or billing departments, that don't reply to our letters. Or reply inappropriately, with a form printout that doesn't match the problem.

First let's understand how this happens.

These outfits are based on using the computer to handle all correspondence and transactions. The "office" may not have any people in it at all— that is, people whose job it is to understand and deal sensibly with the problems of customers. Instead, there may just be keypunch operators staffing a Batch System, set up by someone who has long since moved on.

The point of a batch system (see p. 95) is to save money and bother by handling everything in a controlled flow. This does not mean in principle that things have to be rigid and restrictive, but it usually seems it in practice. (See "The Punch Card Mentality," p. 29.) The system is set up with only a fixed number of event types, and so only those events are recognized as occurring. Most important, your problem is assumed to be one that will be straightened out in the course of the system's flow. While there may be provision for exceptions— one clerk, perhaps— your problem has not seemed to him worthy of making an exception for.

Here is my solution. It has worked several times, particularly on book clubs that ignored typed letters and kept billing me incorrectly.

Get a roll of white shelf paper, two or three feet wide and twenty or more feet long.

Write a letter on the shelf paper in magic marker. Make it big, perhaps six inches to a word. Legibility is necessary, but don't make it too easy to read.

Explain the problem clearly.

Now take your punch card— you did get one, didn't you, a bill or something?— and mutilate it carefully. Tear it in quarters, or cut it into lace, or something. But make sure the serial number is still legible. Staple it lovingly to your nice big letter.

Now fold your letter, and find an envelope big enough for it to fit in, and send it, registered or certified mail, to ANY HUMAN BEING, ACCOUNTING DEPARTMENT, or whatever, and the company's address.

This really works quite well.

I am assuming here, now, that your problem has merit, and you have been denied the attention required to settle it. If we want justice we must ourselves be just.

There is one further step, but, again, to be used only in proportion to the offense. This step is to be used only if a meritorious communication, like that already described, has not been properly responded to in a decent interval.

We assume that this unjust firm has sent you a reply envelope or card on which they must pay postage. Now carefully drafting a follow-up letter, explain once again, in civil language, the original problem, your efforts at attention, and so on. Now put it in a package with a ten or twelve-pound rock, affix the reply envelope to the outside, and send it off.

The problem, you see, has been to get out of the batch stream and be treated as an exception. Flagrantly destroying the punch card serves to remove you from the flow in that fashion. (However, just tearing it a little bit probably won't: a card that is intact but torn can simply be put in a certain slot of the card-punch and duplicated. Destroy it good and plenty.)

In all these cases remember: the problem is not that you are "being treated as a number," whatever that means, but that your case does not correctly fall in the categories that have been set up for it. By forcing attention to your case as an exception, you are making them realize that more categories are needed, or more people to handle exceptions. If more people do this when they have a just complaint, service will improve rapidly.

# JUNK MAIL

The people who send it out like to call it personalized advertising and the like. But most of us call it Junk Mail. And its vagaries are NOT THE POOR COMPUTER'S FAULT. What gets people angry derives from the system built around the poor computer.

You may wonder why you get more and more seed catalogs, or gift-house catalogs, as time goes on, even though you never order anything from them. Or why a deceased member of the household goes on getting mail year in and year out, regardless of your angry postcards.

How does it keep coming?

Through the magic of something called the Mailing List.

And especially the peculiar way that mailing lists are bought and sold.

DIRECT MAIL

Now, a mailing list is a series of names and addresses of possible customers, stored on computer tape or disk.

You can buy the use of a mailing list.

But you cannot buy the mailing list itself.

Suppose you have a brochure advertising pumpkin-seed relish, which you suggest has rejuvenating powers. You want this brochure to go out to rich college graduates.

You go to a mailing-list house.

"I cannot sell you this mailing list outright," says the jolly proprietor, "for it is my business to sell its use again and again, so I do not want anybody else to have a copy of it." So you leave 2500 pumpkin-seed relish brochures with the mailing list company, and pay them a lot of money. And they swear on a stack of bibles that they have mailed the brochures to their special list of rich college graduates.

Well, let's say you get 250 sales from that mailing. (10% is fantastically good.) But out of curiosity you go to another mailing-list house and have another mailing sent out— this one to people who have low incomes and little education.

This time you get 15% orders.

Now guess what you are acquiring.

A mailing list of your very own. Of people who eat pumpkin-seed relish.

Mailing lists are, you see, generally rented blind, with no chance to see the addresses or check as to whether they've already been mailed to.

And that explains all the duplications.

If an advertiser is going after a certain type of customer, and goes to several mailing-list houses asking for mailings to that particular type of customer, chances are some people will be on several of the lists. And since there's no way to intercompare the lists, these poor guys get several copies of the mailing.

(Another way this can happen is if some cheapskate has his own mailing list and doesn't check it for repeats of the same name. But writing the computer program to check for repeats of the same name is not easy— there might just be a Robert Jones and a Bob Jones at the same address— and these things are not usually checked manually. They're big.)

Another possibility exists for eliminating duplications when you rent mailing lists. You can bring in a magnetic tape with your mailing list on it, and they can send out the mailing only to the members of their list who are not already on your list. That way you still can't steal their list, since the tape is on their premises. The trouble is, they can steal your list, by making a copy of the tape. Oh dear.

One possibility, nice and expensive, is to rent a number of mailing lists from a single mailing-list house, with them guaranteeing that they'll compare all the lists you choose and not send to any person more than once.

But as you may be suspecting, this costs money. All this screening and intercomparing requires computer time, and so, even though you are getting a more and more perfect mailing, you are paying more and more and more money for it. So you can see why reasonable businessmen are willing to send out ads even when they know some recipients will get several duplicates.

Another interesting point. There are mailing lists for all kinds of different possible customers. The possibilities are endless. Minority-group doctors. People interested in both stamp collecting and flowers (you'd have to get a company with both lists, and have them go through them for the duplicates... you get the idea).

Note that mailing lists are priced according to their desirability. Weeded mailing lists, featuring only Live Ones, people who've ordered big in recent times, are more expensive. Lists of doctors, who buy a lot, are more expensive than lists of social workers. And so on.

Then there's the matter of the pitch.

The ad's phrasing may be built around the mailing plan. Some circulars come right out and tell the recipient he's going to get several copies because he's such a wonderful person.

THEN there are those advertisements that are actually printed by the computer, or at least certain lines are filled in with the recipient's name and possibly some snazzy phrases to make him think it's a personal letter. Who responds to such things I don't know. My favorite was the one-- I wish I could find it to include here -- that went something like

> You'll really look swell, Mr.     Nelson
> walking down Main Street of     New York
> in your sharp-looking new slacks...

I don't know whether I enjoyed the spaces or the Main Street more.

But you see how this works. There's this batch-processing program, see, and the names and addresses are on one long tape, and the tape goes through, and the program takes one record (a name and address), and decides whether to call the addressee "Mr.," "Ms." or whatever, and then plugs his name into the printout lines that give it That Personal Touch; and then the mailing envelope or sticker is printed; and the tape moves on to the next record.

We may look forward to increasing encroachments on our time and trust by the direct mail industry: especially in better and better quack letters that look as though they've really been personally typed to you by a real human being. (It is apparently legal for letters to be signed by a fictitious person within a company.) In the future we may expect such letters to be sent on fine paper, typed individually on good typewriters, and convincingly phrased to make us think a real personal pitch is being tendered.

There is, however, a final solution.

YOU CAN GET OFF ALL MAILING LISTS — that is, the ones "participating" in the Association— by writing to

Direct Mail Advertising Association
Public Relations Department
230 Park Avenue
New York, NY 10017

They will send a blank. If you fill it in they'll process it and delete your name from mailing lists of all participating companies.

Presumably this won't help with X-rated or stamp-collecting lists, but it ought to keep you from getting semiannual gift catalogs from places like The House of Go-Go Creative, Inc. and those million solicitations from Consumer Reports and that File Box company.

[redacted] Trust

Metropolitan Division
P.O. Box [illegible] Church Street Station, New York, N.Y. 1000_

[redacted]
Senior Vice President

Theodor H Nelson
458 W 20Th St
New York,Ny 10011

Branch 014

FILL-INS

Great news for the Nelson family!

Wouldn't you like your money to work for you full time... even when you're asleep?

Now the Nelson family can save...right at their own bank. [redacted]

-- ** Passbook Savings Plan which
** quarterly or even

DO NOT RETURN THIS FORM IF YOU ARE ALREADY A MEMBER

(whoever you are)

FILL-INS

Dear Reader:

If the list upon which I found your name is any indication, this is not the first -- nor will it be the last -- subscription letter you receive. Quite frankly, your education and income set you apart from the general population and make you a highly-rated prospect for everything from magazines to mutual funds.

You've undoubtedly "heard everything" by now in the way of promises and premiums. I won't try to top any of them.

If you subscribe to [redacted], you won't get rich quick. You won't bowl over friends and business ... clever ...

[redacted]
1255 PORTLAND PLACE, BOULDER, COLORADO 80302

Dear Mr. Nelson:

LOUSY MAILING LIST — I HAVEN'T LIVED IN POUGHKEEPSIE FOR SIX YEARS.

Let's get straight to the point. This is not an ordinary letter. It's a subscription offer, from a magazine. It can save you money off the regular price. ... won't appeal to ... ...And if it ... will appeal to you, Mr. Nelson ... Poughkeepsie, we think it ...

Because it
gives you

You call up the bank and ask your balance and they say, "I'm afraid I can't get that information. You see, it's on a computer."

(See Basic Rejoinder, nearby.)

Well, the reason it's this way is that they're handling things in Batch (see p. 45 ) and they aren't storing your account on disk, or if they are they don't have a terminal they can query it with.

But to say that they can't get the information because it's on a computer is a typical use of the computer as an excuse (see Cybercrud, p. 8 ); and second, if the person believes this to be an explanation, it's a sign of the intimidation and obfuscation that have been sown among the clerks who don't understand computers.

Write them a letter. Change banks. Let's get the banks to put on more and more citizen services. Rah!

# THINGS YOU MAY RUN INTO

Everywhere you go computers lurk. Yet they wear so many faces it's impossible to figure what's going on.

Guidelines are hard to lay down here, but if you look for examples of things you've already run into in this book, it may help some.

Terminals you can presumably recognize.

Microprocessors are harder, because you don't see them. Good rule-of-thumb: any device which acts with complexity or apparent discretion presumably incorporates a terminal, minicomputer or microprocessor.

Two other things to watch for: transaction systems and data base systems.

A transaction system is any system that takes note of, and perhaps requires verification of, transactions. Example: the new point-of-sale systems (POS). This is what's about to replace the cash register.

In the supermarket of the future, every package will have a bar code on a sticker, or printed on the wrapper. Instead of the checkout clerk looking at the label and punching the amount of the sale into the cash register-- an error-prone and cheat-prone technique which requires considerable training-- your New Improved Checkout Clerk will wave a wand over the bar code. The bar code will be sensed by the wand, and transmitted to a control computer, which will ring it up by amount and category (for tax purposes), and even keep track of inventory, noting each object as it is removed from stock.

Here is what your bar code will look like. (A circular code, which was already turning up on some TV dinners, has been eliminated by the bar code. This is unfortunate, since the scanner necessary to read the bar code is electronically more complicated, but there we are.)

(different versions)

(Incidentally, while this does arrest the classic cashier's cheat-- ringing up excessive purchases on the customers, then having a confederate walk through equivalent amounts-- the consumer is still entirely prone to cheating by the store in the computer program. Remember, it's 1974. So you still may have to check your tapes, folks.)

Data base systems are any systems which keep track of a whole lot of stuff, often with complex pointer techniques (see "Data Structures," p. 26). A cute example is the message service now offered by Stuckey's snack/souvenir stands all over the country. You may leave messages for your friends or loved ones on the road; they can stop at any Stuckey's and ask for their messages, just as if it was a telephone answering service. (You're listed by your phone number-- is this to avoid pranks? And what about people with no phones?) It's free and a neat idea. (Obviously, the messages are stored on the disk of a big central computer, and queried from terminals at the individual stands.)

Now, most of the big systems you run into tend to be a combination of transaction and data-base system. For instance, suppose you make an airline reservation. The airline has a large data base to keep track of: the inventory of all those armchairs it's flying around the country, and the list of who so far have announced plans to sit in them, and in some cases what they intend to eat. When you buy your ticket, that transaction then gets you put in the listing. Same for car rentals and so on.

The potential dangers of transaction systems are fairly obvious from the supermarket example, but they fan out in greater complexity as the systems get more complex. Credit cards, for instance, were only made possible by computers and computerized credit verification; but it is only now, fifteen or so years into the credit-card era, that laws protect the cardholder against unlimited liability if he loses it.

Yet we plunge ahead, and it is obvious why. Transaction systems managed in, and by, computers allow more flexible and (in principle) reliable operations. For instance, in the securities business, thousands of stock certificates are lost and mislaid, and the transaction paper must be typed, shuffled, put in envelopes, sent, opened, shuffled again, compared... all by hand. Little wonder they're working on an Automated Stock Exchange System. But if it's taken fifteen years to get the implicit bugs out of credit cards ... not to mention the frequent allegations that much Wall Street "inefficiency" is actually the disguised marauding of Organized Crime... uh-oh. (If they can buy the best lawyers, they can probably buy the best programmers.)

Then there is the Checkless Society. This is a catchphrase for an oft-advocated system that allows you to transfer money instantly by computer; supposedly some such thing is working already in France. Again, they better get it pretty safe before a sane man will go up in it.

The safety of such systems is of course a matter of immense general concern. IBM portentiously (sic) announced its intent to spend millions of dollars on "computer security" a few years ago. However, a few million dollars is not going to plug the security holes in the IBM 360, and evidently the 370 is just about as vulnerable.

(In this light, even the greatest IBM-haters will have to admit that there may be a proper motive behind IBM's current refusal to let others use its new operating system language: that way they may be able to prevent special holes in the system from becoming known to programmers.)

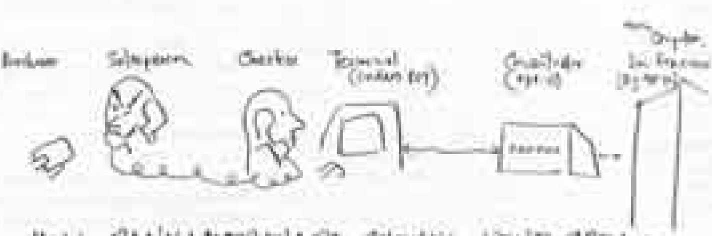
It is interesting that one profession seems to be stepping forward to try to improve this situation: the auditing profession, devoted to verification of financial situations of companies, seems to be branching into the verification of computer programs and the performance of complex systems. This will be great, if it works. Cynics, however, may note that auditors have permitted some remarkable practices in the "creative" accounting of recent years. (Obviously the way to check out the safety of big systems is to offer bounty to those who can break its security. But who is willing to subject a system to a test like that?)

Hereabouts are a few other computerish things you may run into which more or less defy categorization.

## THE COMPUTER GRAVEYARD

In the mid-sixties there was a junkyard in Kingston, N.Y. that was like an automobile graveyard-- except piled high with dead computers.

They were from various manufacturers. The guys would smash them with sledgehammers, or other awful things, to make sure they could never work again. Then you could buy the circuit cards. I saw 1401s five high, Univac File Computers, tape drives... it was an electronic nut's paradise. You could decorate your den with huge old control panels, mag disks and whatnot. It seems to be gone now. They forbade pictures.




HOW BANKAMERICARD CHECKS YOUR CREDIT

# "COMPUTER DATING"

should of course be called MATCHUP DATING, since there is nothing particularly computerish about either the process or its intended result. But there we go again: word-magic, the implicit authority of invoking the word Computer. (See "Cybercrud," p. 8.)

In the early sixties, a perky young fella at the Harvard B-School, I believe, one Jeff Tarr, came up with the notion of a computerized dating service. The result was Operation Match, an immense financial success, which sort of came and went. No followup studies were ever done or success statistics gathered, unfortunately, but they certainly had their fun.

The basic principle of "computer dating" is perfectly straightforward. Applicants send in descriptions of themselves and the prospective dates they would like to meet. The computer program simply does automatically the sorts of thing you would do if you did this by hand: it attempts to find the "best" match between what everybody wants and what's on hand.

Obviously this could be a matter for serious operations research: attempting to discover the best matchup techniques among things that never really fit together, detail for detail; trying to find out, by followup questionnaires, what trait-matchings seemed to produce the best result, etc. But such serious matchup-function research remains, so far as I know, to be even begun.

Obviously there are several problems. Demographically it is almost never true that "for every man there's a woman"-- in every age-bracket there's almost always an imbalance of the opposite sex in the corresponding eligible age-bracket, either too many or too few. But more than that, there is little likelihood that the traits women want are adequately represented among the available males, or vice versa. For introduction services it's obviously worse: there is no balance likely between what comes in one door and what comes in the other. The service can only do its best with the available pool of people-- and make believe it's somehow made ideal by the use of the computer. It's like an employment office: applicants don't match openings.

Numerous other dating services have appeared, some of which don't even pretend to use the computer (and others which claim to be a registry for nonstandard sexual appetites), but none that's gotten the attention of the original Project Match.

But there's no question who got the best dates out of that one. Jeff Tarr.

## DO YOU GOT RHYTHM?

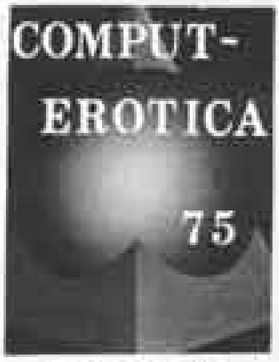
A device called the BIO-COMPUTER (trade mark) purportedly helps you predict your "body beats," telling you what days are the right sort of time to do particular things in terms of your own biological energies. The object costs $15 postpaid from BIO-COMPUTER, Dept. CLB/DM (why not?), 964 Third Ave., NY NY 10022.

The question with all such special purpose devices-- "fishing computers," horse-racing computers, etc., is always whether the theory and formulas which are built into them are correct. There is no ready way to tell.

ASTROFLASH, etc.

There are various computerized astrology services. Given your date of birth, and hour if known, they'll type out your signs, explanations, etc. Presumably there is a text network which the system selects among according to "reinforcing tendencies," etc., among the entities thought to be influential.

Conceivably this could do nine-tenths of what a talented human astrologer does, and with the same validity, whatever that may be. In any case it's probably a lot cheaper.




Is it too soon for a computer pornography contest?

(Is it too late?)

See p. DM 75.

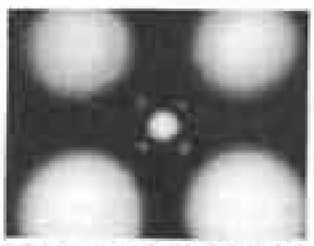
# SUPER-CUSTOMIZATION

People think computers are rigid and invariant. This (as stated elsewhere in this book) is due to the systems which people have imposed, and then blamed, on the computer.

The fact is that computers are now being set up to give new flexibility to manufacturing processes. Computers, directly connected to milling machines, grind metal into any conceivable shape much faster than a human craftsman. To change the result, change the program-- in a fraction of a second. Fabric design has been done on computer screens; the obvious next step is to have the computer control the loom or knitting machine and immediately produce whatever's been designed.

Custom clothing: soon we may look forward to tailoring services that store your measurements and can custom-tailor a suit for you to any new fashion, in minutes. (But will the price beat Hong Kong?) Customized printed matter is already here (see "Me-Books," p. 67). Wherever people want individual variations of a basic manufacturing process, computers can do it.

The Telephone Company (at least in Illinois and Indiana) offers a speaker on "The Shadowy World of Electronic Snooping" to interested groups.



Modern menage, also 29, interested in recursive relations and reverse Polish culture. Phone a must. Contact box 80-212 (& see p. DM 57).

BETCHA DIDN'T KNOW...

that the IRS hasn't been able to do instant matching of W-2 forms to tax returns. That'll be fixed in fiscal '74, and interest and dividend payments in '75. (TIME, 31 Dec 73, 17.)

# "COMPUTER ELECTION PREDICTIONS"

This is an outrageous misnomer. The computer is only carrying out, most speedily, what hardened politicos have always done: FACTIONAL ANALYSIS, now possible with new-found precision on the basis of certain election returns.

This is based on the cynical, and fairly reliable, view that people vote according to what faction of the greater populace they belong to-- middle-class white liberals, blue-collar non-union members, and so on. The factions change slowly over time, and people move among them, but the fact of factionalism remains unchanged.

Well. By the close of a major election campaign, most factions can be pretty well predicted, especially as to presidential choice, or what proportion of that faction will go for a given candidate.

But some factions' reactions are not certain up to the day of the ballot.

So. "Computer predictions" of elections basically break the country into its factional divisions, state by state and district by district, and then tabulate who can be predicted to vote for whom on a factional basis.

Then what's the suspense?

The suspense comes from the uncertain factions-- groups whose final reactions aren't known as the election starts.

Certain election districts are known to be chock full of the types of people whose reaction isn't known.

The final "computer prediction" simply consists of checking out how those districts voted, concluding how those factions are going in the present election, and extending this proportion through the rest of the country.

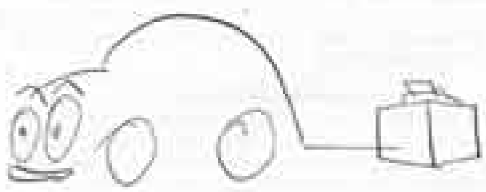
It's often painfully accurate-- but, thank god, not always. When it isn't don't blame "the computer." Thank human cantankerosity.

# THE VW CHECKOUT COUPLER

may or may not be a real computer-- friends have told me it isn't-- but it's certainly a good idea.

When you pull your late-model Volkswagen into a dealer's service area, the guys can just pull out a cable and plug it into the corresponding socket in your vehicle. At the other end of the cable is some sort of device which tests a series of special circuits throughout the car for Good Condition. These circuits indicate that things are working properly-- lights, plugs, points, brakes and so on.

This is the same technique used by NASA up to the final moment of COMMIT LAUNCH-- a system of circuits monitors the conditions of whatever can be monitored, to make sure all's functioning well. It's more expensive to wire it up that way, but it makes checking out the rocket-- or the car-- that much easier.



SSC TRANSIT

Some of the snappier new Urban Transit Systems give you a ticket with a magnetic stripe on the back. Each time you ride you must push the card into an Entrance Machine, which presumably does something to the stripe, till finally the ticket runs out and you have to pay more money.

Secrecy of the recording code is an important aspect of the thing. Indeed, waggish gossip claims that some such systems start with a blank magnetic stripe and just add stuff to it, meaning the card can be washed clean with a magnet by larcenous commuters. But this seems unlikely.

YOUR AUTOMOBILE COMPUTER

Didja know, huh, we're going to have computers in our cars? We refer here to two things--

anti-skid controllers, which are really just special circuits-- you know, "analog computers"-- to compensate among skidding wheels. Turns out that this is apparently more sensitive and reliable than even your good drivers who enjoy controlling skids. Already advertised for some imports.

grand bus electronics (see p. 42). Since the electrical part of the automobile is getting so blamed complicated, the Detroit Ironmongers have decided to switch to a grand bus structure instead of having all those switches and things separate anymore. Should make the whole thing far easier to service and customize.

Presumably this will all be under the control of a microprocessor. (See p. 44.) This means that the car can have things like a Cold-Weather Startup Sequence-- a program that starts the car, turns on the heater, monitors the engine and cabin temperature, and blasts the horn, twice, politely when it's all ready-- all at a time preset by the dashboard clock.

Presumably Detroit is not yet planning to go this far. But because of the auto industry's anomalously huge influence in America, some have expressed the fear that this move -- toward the integrated-circuit, digitally-controlled grand bus-- would effectively put Detroit in control of the entire electronics industry.

The ever-clever Japanese are computerizing faster, better and more deeply than we are.

They now have a prototype taxi operating under computer control. They're calling it, at least for export, Computer-controlled Vehicle System (CVS).

Basically it's like an Elevated Railway-- you climb up and wait-- but when you get in, you punch a button for your destination. According to Hideyuki Hayashi of the Ministry of Industry and International Trade, the system will be operational in Tokyo within the decade, and is the "cleanest, safest, quickest transport system ever devised by man." Think fast, Detroit.

(A nice point: one of the most important features of such a system is that the vehicles don't react to each other, as do vehicles in the existing Human-controlled Vehicle System (HVS). A whole line of the cars can be accelerated or slowed simultaneously, a crucial aspect of their flexibility and safety. Nothing can possibly go long.)

(Leo Clancy, "Now-- Computer-Controlled, Driverless Cars," National Enquirer 3 Mar 74, 28-5.)

THOSE THINGS ON THE RAILROAD CARS

As we lean on the fence a-chawin' an' a-watchin' the trains go by, we note strange insignia on their sides, in highly reflective Scotch-Lite all begrimed by travel.

Basically it's a stack of horizontal stripes in red, blue and other colors. This is ACI, for Automatic Car Identification. It may yet straighten out the railroads.

In this neolithic industry, it is not known at any given time where a railroad company's cars are, and some peculiar etiquette governs their unrequested use by other firms in the industry. Yet the obvious solution may come about: a running inventory of where all the cars are, where each one is going, what's in it, and who that belongs to. But, of course, that's still in the works. Revolutionary ideas take time.

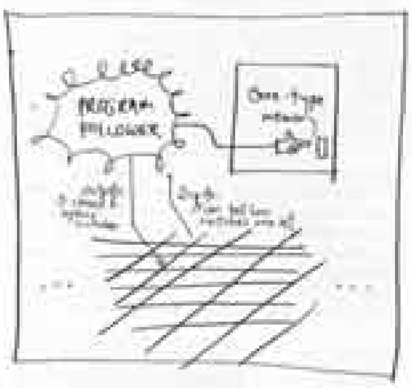
# THE ESS

The national phone company (usually called affectionately, "Ma Bell") has drastically changed its switching methods in the last few years. They are replacing the old electromechanical switches, or "crossbars," with a new device called the ESS, or Electronic Switching System. If there's one in your area you may hear about it in their jolly news sheet that you get with the bill.

In the old crossbar days, a phone connection was a phone connection and that was that. Now, with the ESS, all sorts of new combinations are possible: the ESS has stored programs that determine its operation. If you dialled a non-working number, it jumps to a program to take care of that. It does all sorts of things by special program, and new programs can be created for special purposes. Now the phone company is trying to find the services that people will pay for. Having calls rerouted temporarily to other numbers? Linking up several people in a conference call? Storing your most-called numbers, so you can reach them with a single or double digit?

These particular services are now being offered experimentally.

The way it works is this: there are a number of programs stored in a core memory; the only "output device" of the system consists of its field of reed switches, arranged to close circuits of the telephone network.



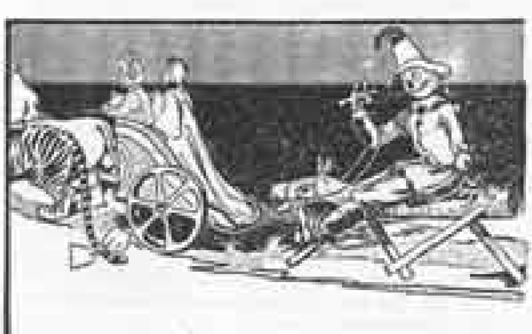

Depending on the numbers that have been dialled, and whatnot, the ESS jumps to a specific program, and that tells it to connect an incoming call to particular other circuits, or to ring other lines, or whatever.

It's really neat.

There are only a couple of things to worry about.

One is that it makes wiretapping, not a complex bother involving clipped wires and men hunched over in cramped spaces, but a simple program.

Another is that some people think that blue-boxers (see nearby) may be able to program it, from the comfort of their own homes. Meaning that not just court-authorized wiretaps, but Joe Schmoe wiretaps, would be possible. Let's hope not.



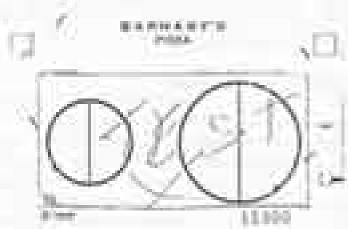
# TELAUTOGRAPH

This has been around for decades, and has nothing to do with computers, but isn't it nice?

You write with a pen attached by rods to a transmitter; somewhere else, a pen attached by rods to a receiver duplicates what you have written.

What is being transmitted consists of the measured sideways motion ("change in x"), the measured up-and-down motion ("change in y"), and the condition of the pen ("up" or "down"). What would these days be called "three analog channels, multiplexed on a single line."

These only cost a couple of hundred dollars. Why has nobody been using them for computer input?




Sugar Creek, Texas will have 3000 homes with a minicomputer-based alarm system. Evidently various automatic sensors around each house sniff for fires and burglars, as well as providing panic buttons for medical emergencies.

The system uses dual Novas (one a backup), and prints out the news to fire and police dispatchers on a good old 33ASR Teletype. (Digital Design, May 73, 16.)

# ONE OF THOSE MYTHS

"Overpay your phone bill by one cent. It drives the computer crazy."

Nope. The amount of payment gets punched in and goes through the gears quite normally.

If you want to put together your own computer-on-a-chip, or any other complex integrated circuit, a complete simulation-verification-layout-and-fabrication service is available from Motorola, Semiconductor Products Div., P.O. Box 20924, Phoenix, Arizona. Presumably it costs a mint, but after that you can roll out your circuits like cookies.

Your circuit is overlaid on their beehive-chip of logical subcircuits, called a Polycell. You use their MAGIC language (Motorola Automatically Generated Integrated Circuits), which then feeds a resulting circuit data structure to a program called SIMULA (yuk yuk) to try out the circuit without building it. That way you can supposedly be sure before they make the final masks.

I always figured that the day of Computer Hobbyism would arrive when the folks at Heathkit offered a build-it-yourself computer. But you know what they came out with instead last year? A general interface for hooking things to the PDP-8.

Minicomputers handle various control functions in our mighty new Aeroplanes and Ships of the Ocean.

It was a truly stellar group that reported to Judge Sirica on 15 Jan 1974 that the 18-minute Watergate tape buzz had at least five starts and stops.

The six panelists included:

Richard Bolt, a founder of Bolt, Beranek and Newman, Inc.
Franklin Cooper, head of Haskins Laboratories, (Comp. )
Thomas Stockham, audio resynthesizer extraordinary (see p. 34 ?)

The news, however, generally referred to them as "technicians."

# QUADRAPONG,

a swell video game now in bars, probably controls the four-player pingpong on the screen with a minicomputer or microprocessor.

Especially exciting is the social possibility of horizontal screens for other fun interpersonal stuff. As well as collaborative work. (But boy, let's hope the radiation shielding is good.)

The Computer Diet by Vincent Antonetti (Evans Pub.) shows the author sitting on the deskplate of a 360 console.

The inside consists principally of charts he recommends for weight loss. "The power of a modern digital computer" interpolated the tables. A slide rule might have have been simpler.

The thing is, he presents a paper on the thermodynamics of weight loss which may be important; in this he states the difference equations which are the heart of his diet. And these may indeed be perfectly valid. So why not call it what it is, The Thermodynamic Diet?

Kirk Brainerd, of L.A., is using computers for a registry of people with something to teach. He hopes that if people are mutually available to each other at a deep enough level, people can begin to act out of altruism in general.

# *ME-BOOKS™*

Would you believe that the greatest available computer service is for the kiddies?

For four bucks and a half, an outfit called Me-Books will send, to a child you designate, a story of which he is the hero, in which his friends and siblings appear, and whose action involves his address and birthday.

Kids adore it. Children who don't like reading treasure the volumes; children who do like reading love them just as much.

I can personally report, at least on the basis of the one I ordered (My Friendly Giraffe) that the story is beautifully thought out, warm, loving, and cleverly plotted. In other words, far from being a fast-buck scheme, this thing has been done right. It's a splendid children's story. (I won't reveal the plot, but the Giraffe's birthday, name and home address are related to those of the protagonist.)

Moreover, it has three-color illustrations, is on extra-heavy paper and is bound in hard covers.

(In case you're interested, any of the three programming languages expounded earlier in the book would be suitable for creating a Me-Book: depending on the language chosen, the holes left for the child's own name would be alphabetic variables, segment gaps or null arrays — anyhow, you could do it.)

Astute readers of the Me-Book will note that while it's not readily obvious, only the lines on which personalized information appear have been printed in the computer's lineprinter. The others have all been pre-printed on a press. Indeed, the personalizations appear on only one side of each page, the whole book being one long web of paper that's run through the lineprinter just once before being cut and bound. But it's so cleverly written and laid out that the story moves on beautifully even on the pages that don't mention the child's name.

As an experiment, the author tried sending for a copy of My Friendly Giraffe as told about a little boy named Tricky Dick Nixon, residing at 1600 Pennsylvania Avenue in Washington, D.C. The result was extremely gratifying, and well worth the $4.50. Herewith some excerpts.

Once upon a time, in a place called Washington, there lived a little boy named Tricky Dick Nixon.

Now, Tricky Dick wasn't just an ordinary little boy. He had adventures that other little boys and girls just dream of.

This is the story of one of his adventures.

It's the story of the day that Tricky Dick met a giraffe.

> As the giraffe came closer and closer, Tricky Dick started to wonder how in the world he was going to look him in the eye.

Tricky Dick knew there were no jungles in Washington. Especially on Pennsylvania Ave.

But Tricky Dick wasn't even a little bit worried.

First, because he was a very brave little boy.

And second, because he knew that his friend, the giraffe, would never take him anyplace bad.

Tricky Dick Nixon was home.

Back in Washington.

Back on Pennsylvania Ave.

And with a story to tell his friends, that they wouldn't have believed if they hadn't seen Tricky Dick riding off on the giraffe's back.

Tricky Dick would long be a hero to those who had seen him that day.

There would be many other exciting adventures for Tricky Dick and his friends.

And maybe, just maybe, if you're a very good boy, someday we'll tell you about those, too.


COMPUTER QUALITY CONTROL SHEET
*ACCOUNT NUMBER: 1344563003

DATA SUMMARY

| | |
|---|---|
| CHILD'S NAME (S): | Tricky Dick Nixon |
| ADDRESS: | 1600 Pennsylvania Ave |
| CITY, ST ZIP: | Washington DC 20050 |
| BIRTHDATE: | July 4, 1976 |
| ADDITIONAL NAMES (S): | Spiro |
| | Mitchell |
| | Pence |
| | Checkers |
| DOG'S NAME: | |
| CAT'S NAME: | |
| BOOK PLATE: | The Founding Fathers |
| GROWN-UP'S NAME: | T Nelson |
| ADDRESS: | |

I said 1776.

BOOK SHIPPED TO GROWN-UP



# About those funny numbers on your checks.

You will note that all bank checks now have funny-looking numbers along their bottoms. They go like this:

0123456789 ⑆ ⑇ ⑈ ⑉

The numbers are odd but recognizable. The last four thingies are punctuation marks, which presumably can mean anything the programmer wants them to. (In other words, frankly, I don't know their names or standard functions.)

The name of these numbers is **MICR**, which stands for Magnetic Ink Character Recording. They are printed in magnetic ink— not magnetic so's you could record on it, like magnetic tape, but check-full of iron and vitamins so that as its blobs whiz past a special read head, they cause a specific sequence of pulses in the parallel circuits of the read head that can be decoded as the specific number or mark.

The MICR system was designed in the late fifties, with the technology convenient at that time, and would certainly not be designed that way now. Nevertheless, these weird-looking symbols have inspired various

RIDICULOUS TYPE-FACES,

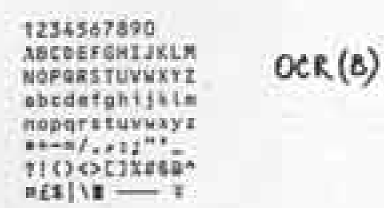
which apparently look to the public like the latest hotcha whizbang zippity up-to-date futuristic stuff, even though to the knowledgeable person they bring back the late fifties. (In fact there are no letters in the MICR character-set.)

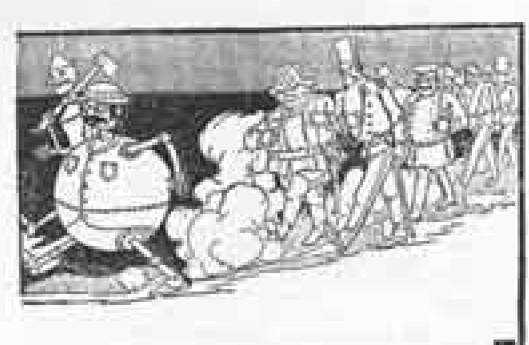
What, then (you may ask) would symbols designed for computers look like if they had been designed more recently?

We were just getting to that. In fact, there are two such alphabets, called OCR (for Optical Character Recognition). They have been standardized so everybody can design equipment and/or programs to work with them. They are called the A and B optical fonts, or, for completeness, OCR(A) and OCR(B).

They are very disappointing.

OCR(A) is a little sexier. At least it looks like something. (Evidently it's slightly easier to deal with and design for.) But the other one, OCR(B), just looks like the alphabet next door. Here they are.

# THE CLUB OF ROME

One of the world's most exclusive clubs is also one of its most dismal. It is The Club of Rome, founded by Italian businessman Aurelio Peccei, having (as of 1973) some seventy members from twenty-five countries.

Their concern they call The Predicament of Mankind, or the "problematique." It is the problem of growth, pollution, population, and What's Happening in general.

On funds from Volkswagen, they have sponsored studies which thinking men can only regard as the most dismal in portent of anything we've seen in years. Or ever.

Basically the prediction is that mankind has perhaps forty or fifty years left.

Not because of war, or bombs, or dirty movies, or Divine retribution, but for simple economic reasons. However, the studies are often called "computer studies," because computers are the viewing mechanism by which we have come to see these coming events.

MALTHUS AGAIN

In the nineteenth century, a pessimistic economist named Thomas Malthus predicted that there would always be starving people, because people increased geometrically-- expanding at compound interest, with a fixed rate of increase creating an ever-steeper growth-- while agricultural production, which must feed us all, expands arithmetically, not as a rate but a few acres or improvements at a time.

This meant, Malthus thought, that there would always be the starving poor. For various reasons this did not happen in Europe. But the regrettable soundness of the general principle persists: when rates of food production can't nearly keep up with rates of population growth, people are going to starve.

This is basically the prediction.

DYNAMIC MODELLING

Basically what has happened is this. One Jay Forrester, of MIT, has for some years been studying "Dynamic Models" of things, a new breed of simulation which couldn't have been done without computers. And now dynamic models of the world's entire economic system can be created and tried out.

Basically dynamic models are mathematical complexes where things change at rates that change themselves over time. For instance, the more you eat, the fatter you get, and the fatter you get, the hungrier you are going to be. Now, just because this is simple to say in words, and sounds as though mathematicians would have had solved the whole class of problems centuries ago, that's not how it is. The intricacy of such models, even for just a few variables, made it impossible to foresee what happens in such complexes except by techniques of computer simulation. Forrester, who has studied such systems since the fifties, has become alert to their problems and surprises. The culmination of his work has been a model of the entire world's economic growth, agriculture, population, industrialization and pollution; this is described in his book World Dynamics (Wright-Allen, 1971).

The insidious portent of Forrester's work did not go unnoticed. The dangers of population increasing at compound interest on a planet of unchanging size, and further derivatives of these changes, suggested that things might be getting worse than anybody thought. An alert Italian businessman brought together a group of scholars from all over the world to study these problems, and called the group The Club of Rome. Their first work is out now, and it is very scary and all too real. The book is called The Limits to Growth.

Basically what they have done is a very elaborate computer simulation, modelling the entire economy of the planet in the years to come as a structure of rates. They have taken into account population, food-growing capacity, industrial growth, pollution, and a lot of other things. The model is precise and elaborate.

Unfortunately the findings are precise and simple.

They tried all kinds of alternative futures using the model-- what would happen if the birth rates were different? What if there were no pollution? What if resources were infinite?

The results of the simulations are always the same.

According to all the simulations, the human race will be wiped out-- mostly or completely-- by the year 2100.

Let's go briefly through the model. Note that it can't be exact, and we can't know what years things are going to happen. The curves themselves-- the shape of things to come-- tell the story all too clearly. (For those who would like a little more drama with their numbers, finding these matters too abstract, I strongly recommend the very beautiful Indian film "Distant Thunder," a sort of "Mr. Smith Starves to Death." Or just stick around awhile.)

HOW?

The model assumes that birth rates stay relatively constant in particular parts of the world, and that new land and agricultural techniques increase food production in relatively well-understood ways.

Of course, population continues to go up, on the familiar but deadly curve.

Civilization, and the bulk of mankind, have about forty years to live, according to certain studies (see p. [illegible]). The studies are depressingly good, although unfinished.

There are four possible things to do.

1. Ignore it.

2. Deny it.

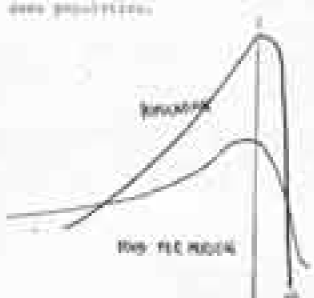
3. Seek individual salvation somehow. Hide in a remote retreat. Lay in stores.

4. The glorious Decameron. Eat, drink and be merry, for tomorrow we die. Or apocalyptic occultism, or whatever.

5. Work starting now. In whatever directions might, just might, point or contribute to a way out.


Now for the good news. Food production also tends to increase:


Now for the bad news. The running ratio of food to people, Food per Capita, takes a sudden nose-dive. And then so does population.




It is not any individual prediction that is frightening, since the numbers plugged into the equations runs are merely hypotheses, to show the shape of the consequences. It is the overall set of runs that is so ghastly, because they always come out the same.

PAY CLOSE ATTENTION

Now, it is important to clarify what is happening here and what is not. What is not happening: an oracular pronouncement by "the computer," showing some transcendental prediction by a superhuman intelligence. What is happening: people are trying out separate possible assumptions to see what their consequences are, enacted by the computer according to the economic rules they set up. Result: always the same. Any set of rules, played out in the unstable exploding-population world beyond the seventies, appears to have similarly dire results.

WHAT HOPE IS THERE?

The original model is only an approximation, and the basic results, as published in The Limits to Growth (see box) reflect these approximations. One of the things that can be done is to fill in and expand the model more, to see whether any hopes can be found in the details and fine cracks which don't appear from the gross results. And, of course, to study and re-study the basic findings. (For instance, a small error was recently found: a decimal point was misplaced in the "pollution" calculation, leading to an overstatement of the pollution in some of the runs. (But pollution, remember, is only part of the problem.)

So there you are. This is a study of the greatest importance. We may, just may, be getting wind of things in time to change the outcome. (If only we knew how. But again, this study is where serious discussion must begin.)

IBM IS BULLISH ON THE FUTURE

Lewis M. Branscomb, who has the awesome title of Chief Scientist of IBM, has been giving numerous talks recently that seem to be directed against pessimism resulting from the Club of Rome studies.

> "On the shoulders of the information processing community rests the responsibility for convincing the public that we have the tools, if it has the will, to address the complex systems management problems of the future," Branscomb said.
>
> "More than any other profession our community can restore the public's confidence that from the limited resources of the world can be fashioned a life of well-managed abundance for all," he concluded."
>
> (Keynote speech, ACM 73, quoted in Computerworld, 5 Sep 73, p. 4.)

# ENDGAME.

Now begins the winter of the world.

We are poisoning everything.

With so little time left, we are of course expanding and accelerating every form of pollution and destruction.

We are killing the last of our beautiful brothers, the whales, just to provide marginal amortization of the whale-ships that are going to be scrapped anyway.

Now: supposedly the Sahara Desert was man-made. It is growing fast.

Set down upon this beautiful planet, a garden spot of the universe, we are turning it into a poisoned pigsty.

You and I may starve to death, dear reader. In some year fairly soon now, around the turn of the century, there will no longer be nearly enough food for the teeming billions.

That, anyway, is what the predictions say. The predictions are compelling, not because a computer made them -- anybody can make a computer predict anything-- but because the premises from which the predictions grow were very well thought out.

It is now up to us to make the predictions come out wrong.

Not by killing the bearer of bad tidings, or by pretending they were not clearly stated-- but by seeing what possible alternatives remain in the few moments of free choice we may yet have-- scant years at best.

To haggle now about ideology is like arguing about who is driving while we are headed toward a brick wall with the gas pedal jammed to the floor.

The public thinks, "science will save us," a view at which many scientists snicker bitterly. Perhaps we will be shrunk to an inch's height, or fed on rocks, or grown gills and super-kidneys to live in the ever-more-poisoned soil. Or perhaps we will do what science says others have done: die out.

This science-will-save-us ostrich-position is nicely exemplified by Albert Rosenfeld, Science Editor of Saturday Review/World.

Since "science" has given us the flying [illegible] and the neutron, neither of which could once, he thinks, have been imagined possible, obviously (to him) science can do anything else we think is impossible! He fully imagines that science will come up with something to take care of geometrically increasing numbers of people. In perpetuity?

"Take a lesson from the neutron," he says. "We can solve our problems." ("Look to the Neutron, Thou Doomsayer," Saturday Review/World, Feb 23 1974, 47.)

OTHER FUN

The growing diffusion of weapons and [illegible], and the great vulnerability of almost everything, ensure that terrorism and political extortion will will increase dramatically for the foreseeable future. On the other hand, [illegible] have lately occurred and demonstrated by example how to hold the country at bay in order to get their wishes; as everybody can see what's happening, and learn from it, the number of [illegible] accomplishments created by terrorism and [illegible] will [illegible].

All these were [illegible] foreseen by Thomas C. Schelling in his masterly 1960 work, The Strategy of Conflict. Schelling formalizes a theory of intimidation as part of his study of communicating adversaries. (His is a structural rather than a psychological study, examining the properties of situations whether or not they are psychologically perceived. Regrettably, perception of situations is improving all the time.)

Countless says the oceans are dying a lot faster than he anticipated -- and gives mankind fifty years after life ends there.

But even if everything else were all right, the Breeder Reactors are sure to get us. I refer to those wonder machines that the electric companies are calling Clean Energy for the Future. What is not explained with such slogans is that breeder reactors not only create energy, they create atomic waste, breeding new fissionable material-- including plutonium. Plutonium is well named for the god of hell. Chemically a poison and radioactively a horror, it does not go away; whatever we put it, it will get back to us.

The mere radiation from the atomic crap is hardly the problem. The radioactive poisons are getting into the oceans. They are getting into the clean waters of the land. (A December 1973 news report, for instance, revealed that a 1968 leak of radioactive chemicals was into the water supply of Broomfield, Colorado.) Now, atomic enthusiasts call it a Disposal Problem, like the question of where to bury the garbage. But it's a very different problem. Wherever we put it, it will come back. The sea? No, that'll be poisoned after the containers go. Deep wells? The mountains? But there is no place that cannot be guaranteed against earthquake and recycling. It will come back. Though dozens of generations might survive it, it will be waiting.

But the breeder reactors multiply this output. Perhaps we could survive the the waste for a few hundred years, till it comes back out. But the other part of it is the fissionable material which can be made into backyard bombs.

That's the kicker. With more and more fissionable crud being [illegible], its availability for terrorists who want to build their own increases. Ralph Lapp pointed out last year that the stuff was shipped in unguarded trucks, and one or two good hijackings would enable any bright kid to build his own dirty A-bomb. By the year 2000 it is not inconceivable that homing atomic weapons will be as widespread as handguns in Detroit-- and as much used.

But now, with the breeder reactors-- in lots of countries-- grinding the stuff out, the era of atomic plenty is here. The smaller countries who want them are getting their atomic weapons -- though holding back assembly of the parts, for various reasons. It is generally believed among bomb-watchers, for instance, that India and Israel have theirs anytime they want.

And this is the great arsenals of missiles, fat and funny in their silos, ready to go on two, take three or four, some. (The U.S. official arsenal now stands at the explosive equivalent of 5 billion tons of TNT, a ton of TNT for every human being. And that's just the explosive part, not the fallout, a fraction of those bombs would destroy all life on earth by its sterilizing residue.) And now, because of the SALT talks, we may expect a new and drastic increase of this Doomsday Product. Hee hee. What is there to say.

So there it is, folks, merry fears ahead. Humanity may end with a bang (thermonuclear exchanges, or just [illegible] things until we're all poisoned or sterile), or a whimper (unfunded starvation), or, I would anticipate, some spastic combination of the two, and all within the [illegible] lifetime of the average reader. This is, at any rate, what I think most likely.

Except of course we won't see it happen that way. We'll watch the starvations on TV (as we did Biafra, Bangladesh, now West Africa, what next ... India?), and talk about the poor foreigners who can't take care of themselves. And as the problems increase and move toward our heartland, it'll be blamed on environmentalists and on the news media. Till bang.

Or maybe not. Just maybe.

But we've all got to get access to the Club of Rome models, and look for holes or strengths. If computer modelling systems doing this kind of work are made widely enough available, perhaps some precocious grade-schooler or switch-[illegible] will find some way out that the others haven't hit on.

We've got to think hard about everything.

BIBLIOGRAPHY


Frederick Pohl and C.M. Kornbluth. The Space Merchants. Ballantine, paper.

Thomas C. Schelling. The Strategy of Conflict. Paper.

The Great American Bomb Machine (citation not handy). Paperback.

A book called Cold Dawn, citation not handy, originally published in the New Yorker, presents a most discouraging view of this country's actions in the SALT talks.

One Access Catalog, not to be named here, gives a recipe for an atomic bomb. Very funny, he is. "The U-235 we are using, (although Plutonium will work just as well) is a radioactive isotope and deserved some care in handling. It is NOT radioactive enough to kill with limited exposure, but don't sleep with it or anything." And so on. Thanks a lot, folks.

Ralph Lapp had a piece in the New York Times Magazine last year, pointing out that plutonium is shipped in unguarded trucks and it's only a matter of time before [illegible] get their hands on it.

A piece in a recent Esquire, "Did There Ever Come a Point in Time When There Were Forty-Three Different Theories about Watergate? Yes, to the Best of Our Recollection," is a very helpful general source, especially for those who suspect a connection between "Watergate" and the assassinations of the Kennedys, Malcolm X, Martin Luther King, etc. But for a real chill see "Mae Brussell's Conspiracy Newsletter" in the March (?) 1974 Realist, as well as "Who Is Organized Crime and Why Are They Saying Such Awful Things About It," same issue.

Glen A. Love and Rhoda M. Love, Ecological Crisis: Readings for Survival. Harcourt, $4 (paper). A quick way to catch up on some bad stuff. Four bucks well spent.

William Leiss, The Domination of Nature. Braziller, $7.

For a dazzling, romantic and optimistic view of the future, see Dimensions of Change by Don Fabun (Glencoe Press, $5 in paper).

The Futurist magazine goes out to members of the World Future Society, An Association for The Study of Alternative Futures. Post Office Box 30369, Bethesda Branch, Washington, DC 20014. The magazine used to be pretty sappy and optimistic, but seems to be acquiring sophistication.

Ronald Kotulak, "The Lifeboat Ethic," Chicago Tribune Magazine, 28 Apr 74, 16-22.

Perhaps the Club of Rome study should be called:-

# THE HOLE EARTH CATALOG

Tim DeForest (neg. me)

## "I have a dream..."

P. My feeling frankly is this: That you know I was just thinking tonight as I was making up my notes for this little talk, you know, what the hell, it is a little melodramatic, but it is totally true that what happens in this office in the next four years will probably determine whether there is a chance, and it's never been done, that you could have some sort of an uneasy peace for the next 25 years.

E. Uh huh.

(Nixon to Ehrlichman. Apr 73.)

*Thank you, Mr. President.*

## READ IT AND WEEP

Donnella H. Meadows, Dennis L. Meadows, Jørgen Randers and William W. Behrens III, The Limits to Growth: A Report for THE CLUB OF ROME'S Project on the Predicament of Mankind. Universe Books, paper, $2.75.

"Things are going to get worse and worse and never get any better again."

— attributed to Kurt Vonnegut, Jr.

"FOLKS DON'T NEED THESE LI'L SHMOOS!!— THEY ALREADY GOT ONE— TH' BIGGEST SHMOO OF ALL— TH' EARTH, ITSELF! JEST LIKE THESE LI'L SHMOOS, IT'S READY T'GIVE EV'RYBODY EV'RYTHING THEY NEED!! EF ONLY FOLKS STOPPED A-FIGHTIN', AN' A-GRABBIN'— THEY'D SEE-LIKE THEY THIS SHMOO— TH' EARTH— GOT PLENTY O' EVERYTHING— FO' EV'RYBODY!!"

— Li'l Abner

(Al Capp, The Life and Times of The Shmoo, Pocket Books, 1948, pp. 121-122.)