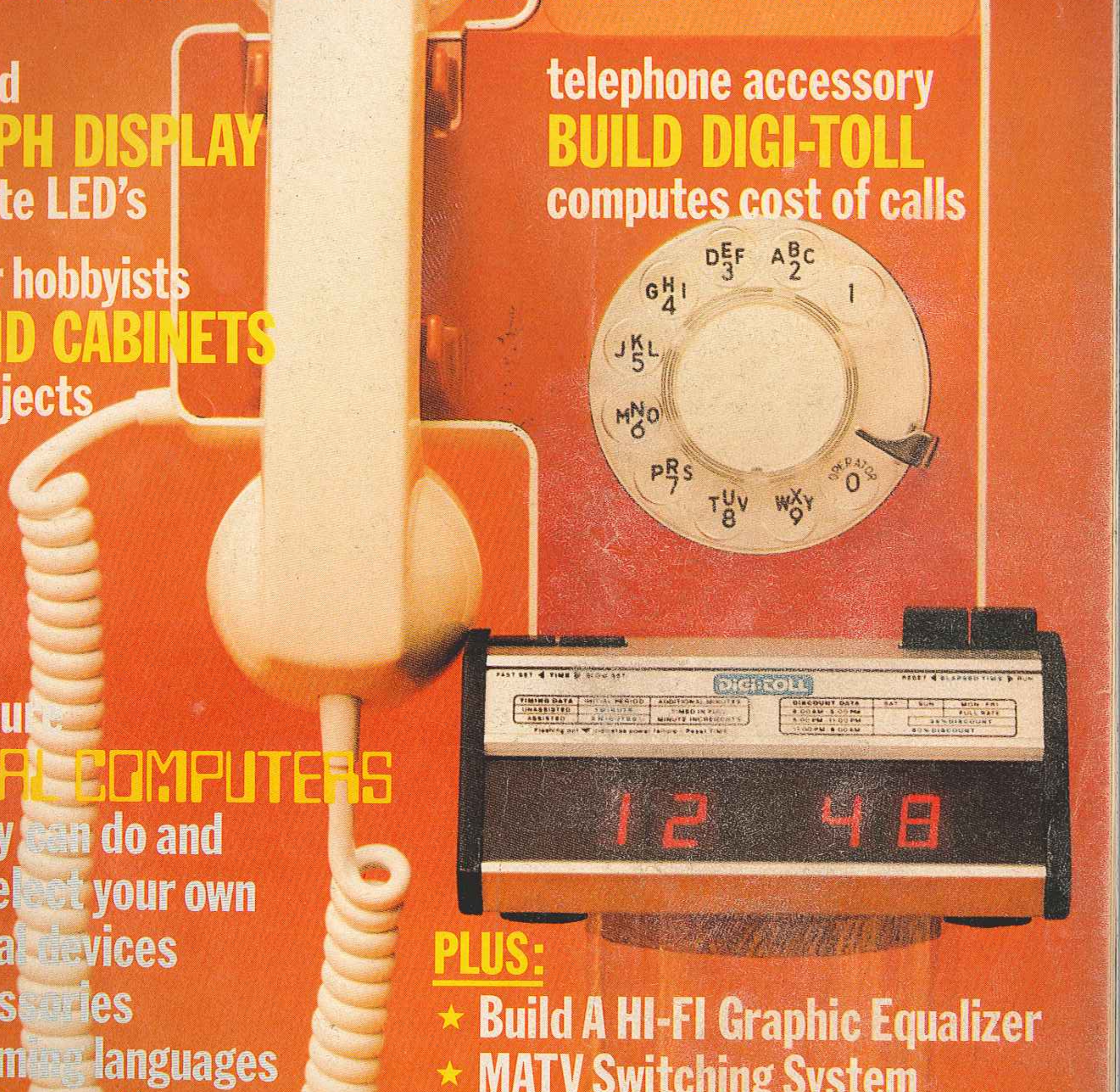
SPECIAL 32-PAGE FEATURE—PERSONAL COMPUTING

$1.00 ■ JUNE 1978

Radio-Electronics®

THE MAGAZINE FOR NEW IDEAS IN ELECTRONICS

VOL. 49 No. 6



easy to build
BAR-GRAPH DISPLAY
uses discrete LED's

roundup for hobbyists
CASES AND CABINETS
for your projects

telephone accessory
BUILD DIGI-TOLL
computes cost of calls

special feature
PERSONAL COMPUTERS

1 what they can do and how to select your own
2 peripheral devices and accessories
3 programming languages and how to use them
4 roundup of who makes what

PLUS:

★ Build A HI-FI Graphic Equalizer
★ MATV Switching System
★ All About Audio Oscillators
★ Hobby Corner
★ State-Of-Solid-State
Service Clinic
8080 Computer Corner

0 70989 34140 06

692188 JNK 1102409O 14 A DEC78
R J JENKINS 06
1102 SOUTH 45TH ST
TEMPLE TX 76501

This 8-bit machine, by itself, is as versatile as a lot of systems that include peripherals

Skeptical? For starters, because of its unique design, the H8 is the only machine in its price class that offers full system integration yet, with just 4K of optional memory and using only its "intelligent" front panel for I/O, may be operated completely without peripherals!

Memory Display

Register Display

I/O Port Display

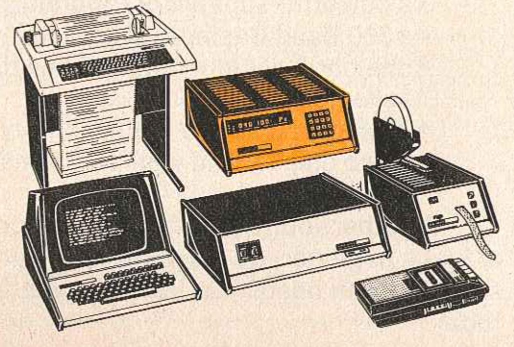
In addition, by using the features of its built-in PAM-8 ROM panel control program, the H8 actually allows you to dig in and examine machine level circuitry. Responding to simple instructions, the "intelligent" panel displays memory and register contents and lets you inspect and alter them even during operation. And for greater understanding, the front panel permits you to execute programs a single instruction at a time. The H8's memory is fully expandable, its 8080A CPU extremely versatile, and with the addition of high speed serial and parallel interfacing you gain the added flexibility of I/O operation with tape, CRT consoles, paper tape reader/punches, and soon floppy disk systems! The H8 offers superior documentation including complete step-by-step assembly and operation manuals, and comes complete with BASIC, assembler, editor, and debug software that others charge over $60 for! H8, simplicity for the beginner, sophistication for the expert and at $375* just right for you.

*Prices are mail order net FOB, Benton Harbor, Michigan. Prices and specifications subject to change without notice.

HEATHKIT COMPUTERS

System Engineered for Personal Computing

H8



Computers, peripherals and nearly 400 exciting, easy to build electronic kits all in your

FREE Heathkit Catalog

HEATH Schlumberger	Heath Company, Dept. 020-420 Benton Harbor, MI 49022

Please send me my FREE Heathkit catalog. I am not on your mailing list.

Name ____________________

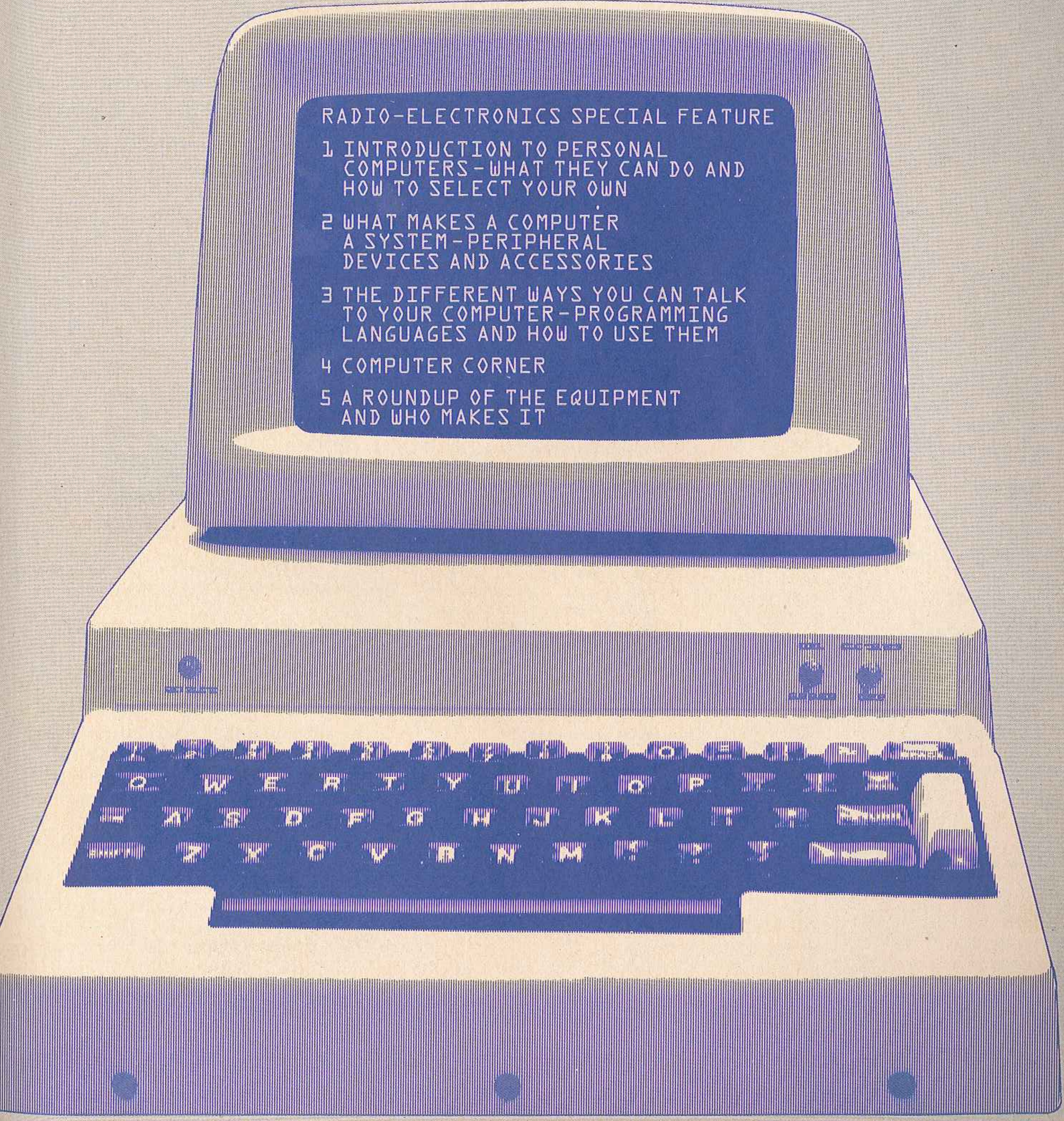
Address ____________________

City ____________________ State ________

CP-150 Zip ________

CIRCLE 100 ON FREE INFORMATION CARD

YOUR OWN Computer



RADIO-ELECTRONICS SPECIAL FEATURE

Personal Computers Are They Right For You?

Ever wonder what you would do with a computer if you had one in your home? The possibilities are limited only by your imagination. Here's a look at a few of the many possibilities, plus a look at what a computer system consists of and what you should look for when selecting your own.

WILLIAM BARDEN

Buying a personal computer system today is somewhat similar to choosing a wife. Rather than evaluating the field qualitatively and making the choice based on logic, a lot of emotion is involved. This article describes currently available microcomputers and attempts to give the reader some selection guidelines.

Why buy a microcomputer?

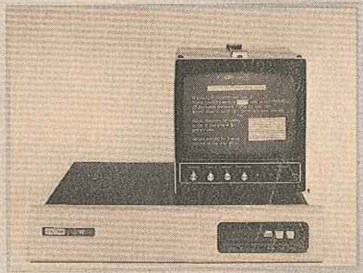
Personal computers are essentially composed of logic circuitry. Unlike complicated special-purpose circuitry

BYT-8 microcomputer from Byte Incorporated has toggle switches and LED indicators on the front panel.

that performs only one dedicated function, however, the microcomputer can be rapidly and easily changed to perform a variety of functions by reprogramming. The circuitry is "told" how to perform each specific job demanded of it by its owner. Using any of the many microcomputers available today it is possible (at reasonable cost) to:

- Play a variety of games from Star Trek to involved combat games.
- Create and play synthesized musical selections
- Understand human voice commands.
- Speak with a Scots brogue.
- Set up an energy conservation system for your home with the computer controlling the heating and cooling, and possibly even opening and closing windows automatically.
- Prepare your income tax.
- Set up a weight-loss program that is specifically tailored to the amount of weight you want to lose, how fast you want to lose it and how much work you want to put into it.
- Compute your payroll; figuring tax deductions, health insurance, social security, etc.
- Act as a bill collector; automatically preparing collection letters that are individualized for each customer and reporting back to you on the status of each customer and the total of your unpaid accounts.
- Balance your checkbook and estimate your net worth.
- Inventory parts for your small business.
- Decipher and generate Morse code for an amateur radio transceiver.
- Provide a burglar and fire alarm for home or business.
- Provide automatic telephone dialing and decode remote telephone commands.
- Control lights, sprinklers and heating.
- Tutor yourself, spouse and children.
- As more software is developed (more instructions), your computer system can grow to accomplish additional tasks. It is never limited to a specific set of tasks. Whenever you come up with a new job for your machine, a little programming will make it possible for the computer to add still another job.

Now it is possible to design individual, special-purpose circuits so that they can provide most of these functions, but what other design can perform *every* one of these functions and at such a low cost?


pdp 11/03 MINICOMPUTER from digital shows how information is displayed on a CRT.

Several years ago, a properly equipped minicomputer would have cost about $40,000. Now, all the functions described above can easily be implemented on a $2000 microcomputer system, and many can be performed on a $600 system. If you're ready to spend what you'd pay for a new TV set or stereo system, just take a look at what your money will buy.

Microcomputer components

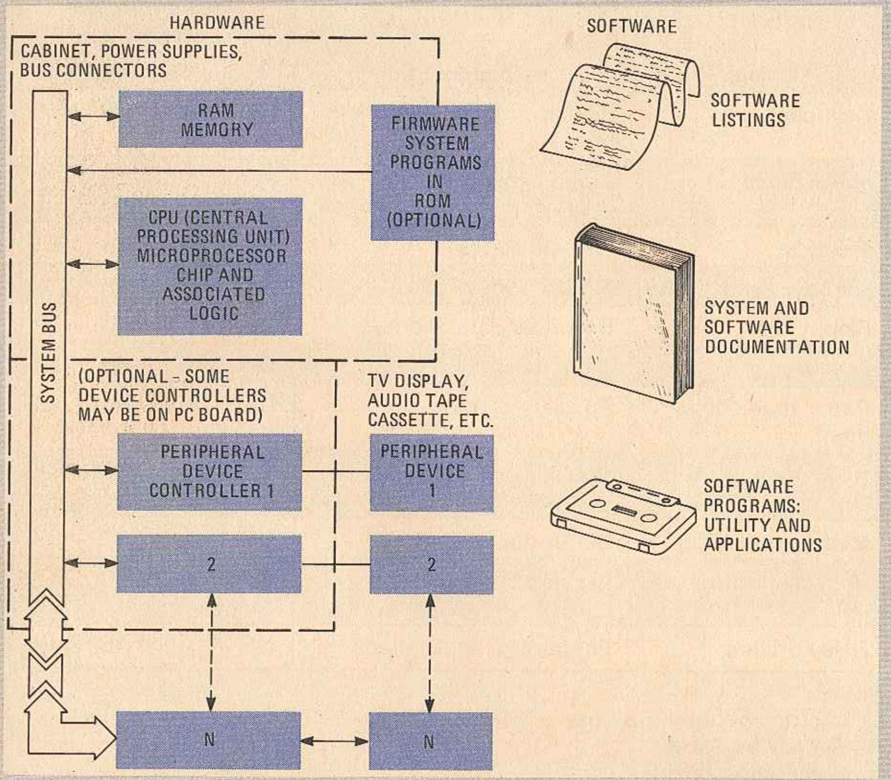
Figure 1 shows typical microcomputer components. The logic components are grouped into *hardware, software* and *firmware*.

Hardware consists of the cabinetry that contains the circuitry, necessary power supplies, a real or imaginary bus representing system logic signals, memory, a central processing unit (CPU) and firmware. The bus is the master wiring system of the computer. It can most conveniently be thought of as a wide ribbon cable consisting of as many as 100 separate wires.

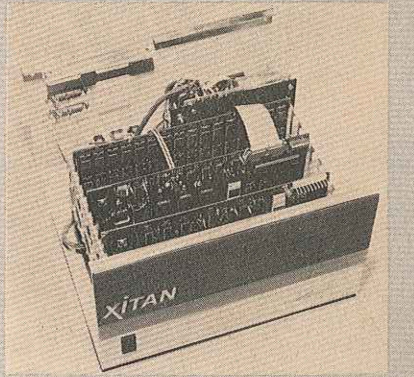
In the S-100 bus, for example, there are 100 wires. Each one carries a specific signal. Each circuit board in the computer is set up with a connector that matches the bus. Lead number 1 is always the same in that system, as is lead number 2, and 3 and so on. All bus systems are not compatible. Each one has its own set of connections. This can be both an advantage and a disadvantage.

The good side is that several companies can make products that fit a particular bus. As the computer owner you can then buy plug-ins from a variety of suppliers. However, in practice, there are multiple suppliers for only a small number of the available bus systems. For the others, the only source of add-ons is the original system manufacturer. Don't forget this point. It could cost you dollars when you want to expand your system.

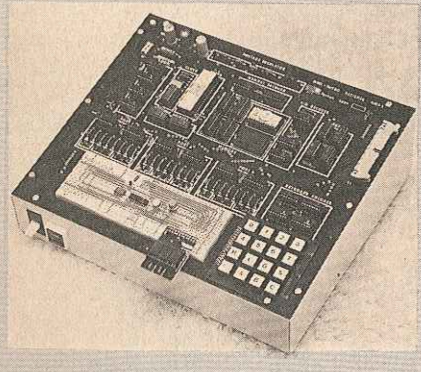
Software is the instructions that enable the computer to follow your directions and accomplish its task. Again, there are options, mostly in terms of the "language" you use. (For more information on language and programming, see the following article in this special section.) There is also available for almost all personal computers, prepared software on magnetic tape or paper tape that can be used to tell the computer how to do a specific job. Firmware is hardwired software and cannot be easily changed.


HARDWARE
CABINET, POWER SUPPLIES, BUS CONNECTORS
RAM MEMORY
FIRMWARE SYSTEM PROGRAMS IN ROM (OPTIONAL)
CPU (CENTRAL PROCESSING UNIT) MICROPROCESSOR CHIP AND ASSOCIATED LOGIC
SYSTEM BUS
(OPTIONAL - SOME DEVICE CONTROLLERS MAY BE ON PC BOARD)
TV DISPLAY, AUDIO TAPE CASSETTE, ETC.
PERIPHERAL DEVICE CONTROLLER 1
PERIPHERAL DEVICE 1
2
2
N
N
SOFTWARE
SOFTWARE LISTINGS
SYSTEM AND SOFTWARE DOCUMENTATION
SOFTWARE PROGRAMS: UTILITY AND APPLICATIONS
FIG. 1—A MICROCOMPUTER SYSTEM consists of RAM memory, a central processing unit plus system programs stored in ROM memory. Peripheral devices, such as a CRT terminal, printer, or additional external memory are connected to the system via peripheral device controllers. Signals to the various parts of the system are carried along the system bus.

The cabinet in many cases is quite unimpressive. The Apple II and Radio Shack TRS-80 systems are of this type; they contain a minimum of controls and switches. Other microcomputers have a cabinet with a control panel, as is the case with the MITS 8800b and IMSAI 8080 microcomputers; however, a control panel's usefulness is debatable.

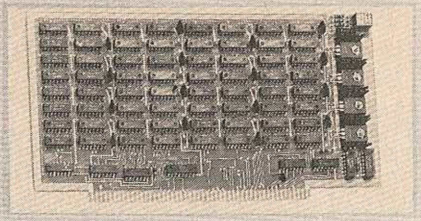

XITAN MAINFRAME from Technical Design Labs has an S100 bus structure.

Within the cabinet are the power supplies and system bus. If the microcomputer is similar to the Byte Incorporated BYT-8, it contains a discrete physical bus. You can look into the cabinet and observe a series of 100-pin connectors mounted on a PC board with 100 etched lines representing the bus. If the microcomputer is a Commodore PET, the bus may be represented only by external connectors and various etched lines on a single large PC board. The bus of a microcomputer is usually similar to the pinout (the diagram that shows where the individual pins on the microprocessor connect and shows their functions) of the microprocessor itself. A microcomputer built around the 8080A microprocessor


MINI-MICRO DESIGNER from E & L Instruments is a breadboarding system for prototyping computer circuits.

has 8080-type signals, one constructed around the 6800 microprocessor has 6800-type signals, etc.

If the microcomputer has a bus with plug-in connectors (as in the MITS 8800b), the microprocessor and associated circuitry will be on a plug-in board called the CPU board. (The CPU is the central processor unit. It is the brain, the control center, of the computer.) If the microcomputer is an integrated single-board type (PET), the CPU forms part of the logic IC's on the board itself.


STATIC RAM MEMORY from Vector Graphic is designed for the S100 bus structure.

The CPU consists of the *microprocessor* IC and any TTL or MOS IC's required to interface the microprocessor to the rest of the system. Almost all personal computers use either the 8080, 6800, 6502, or Z-80 microprocessors as a base around which the microcomputer is built. A comparison of the four types is beyond the scope of this article, but, in general, the current versions of these microprocessors are approximately equal in speed, instruction set and efficiency—much more so than, for instance, the differences between an 8080 microprocessor and its predecessor, the 8008.

(Time out for some definitions. The instruction set is the set of built-in instructions that the microprocessor can inherently perform. Every microprocessor type has its own specific instruction set. Speed relates to how quickly the microprocessor can complete a step in its operation. This time is usually measured in microseconds.)

In many applications, the actual microprocessor IC used will be *transparent* (transparent means that the user will not be aware of the microprocessor and will not know what type is in the machine unless he checks the manual to find out) to the

H9 CRT TERMINAL from Heathkit has CRT display plus alphanumeric keyboard.

system user. The differences in speeds and efficiency between the microprocessors themselves will be secondary to other system attributes such as the efficiency of the software, the design and the bus structure.

Memory will also be contained on separate plug-in modules in a bus-oriented microcomputer (8800b), or on a portion of the motherboard (PET). All current microcomputers can address up to 65,536 bytes of

RADIO-SHACK'S TRS-80 microcomputer system has memory and keyboard.

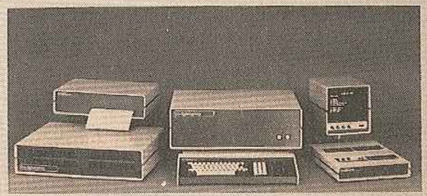
memory directly (sixty-four 1024-byte segments, or 64*K*). In practice, this memory range is divided between user-accessible memory and firmware memory. If 10K bytes are used for firmware, for example, only 54K bytes are available to the system user.

COMPUCOLOR offers microcomputer and full-color graphics terminal in single enclosure.

The firmware portion of memory consists of read-only memory, physically represented by ROM, PROM, or EPROM (*R*ead-*O*nly *M*emory, *P*rogrammable *R*ead-*O*nly *M*emory, or *E*rasable PROM). Generally, the firmware includes the manufacturer's systems software programs to provide file manage, an operating system and a BASIC interpreter. Firmware is included with such microcomputers as Radio Shack's TRS-80 and Commodore's PET, and can be added to virtually any other microcomputer. Firmware programs are nonvolatile, that is, turning off the computer will not destroy the program burned into the PROM.

Firmware is an easy way to add specific functions to a computer. Unlike entering a program into memory, firmware does not consume any memory space and it cannot be erased (with the exception of EPROM's which can be erased and reprogrammed after being removed from the computer). One way to think of firmware is as a plug-in program—the program is in the form of an integrated circuit.

The remaining portion of memory is RAM (*R*andom *A*ccess *M*emory or *R*ead-*W*rite *M*emory) and is user-accessible for program storage. Naturally, the more memory, the more flexible and powerful the system. A small system can have 4K of RAM, while a larger configuration might have 32K or more.


COMPUTER SYSTEM from The Digital Group shows the various peripheral devices.

Peripheral devices and controllers for the devices are connected to the microcomputer either directly through the bus or through input/output (I/O) ports and connectors.

The peripheral devices shown in Table I are available for use with current microcomputers. Not every device, however, is available for every microcomputer. Many devices can be connected to virtually any bus by simple interfacing-design work.

A minimum workable system consists of a keyboard (to enter alphanumeric characters), a video display (to display characters or graphic lines), a CPU and memory. Each peripheral device needs a controller, essentially an interface between the bus and device. Since the peripheral devices operate much more slowly than the CPU's hundreds of thousands of instructions per second, the controller buffers the data to match the CPU and device speeds and per-

TABLE I—Peripheral Devices For Current Microcomputers

Device	Explanation	Availability On System Type			
		I	II	III	IV
Keyboard	Similar to typewriter keyboard	BC	A	AB	A
Video display	Alphanumeric and/or graphic	BD	A	AB	A
Teletype	ASR-33 series	BD	A	AB	CD
Audio cassette tape	Secondary storage	BD	A	AB	A
Floppy disc	Secondary fast storage	DE	A	AC	AB
Printer	High-speed hard copy	CE	A	AC	BD
Paper tape equipment	Auxiliary storage	CE	A	BC	CE
A/D, D/A	Analog-to-digital input, D/A output	CE	A	AC	CE
Music synthesizer	Better than a MOOG	CE	A	AC	CE
Speech input/output	Actually sound input/output	CE	A	BC	CE
Relay drivers	For control applications	CE	A	AC	CE

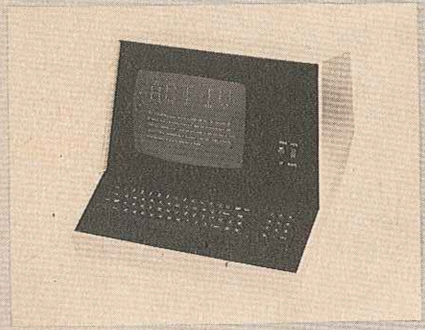
Notes:
I. Microcomputer on a board; II, S-100; III, non-S-100; IV, turnkey system.
A. Readily available.
B. Sometimes available.
C. May be connected with some difficulty or design work.
D. May be connected with more difficulty or design work.
E. A major effort.

TABLE II—Current Microcomputer Types

Company	Computer	Type	Microprocessor
Apple Computer	Apple II	I	6502
Central Data	2650 Computer	I	2650 (Signetics)
E&L Instruments	MMD-1	I	8080A
Iasis	ia7301	I	8080A
IMSAI	IMSAI 8048	I	8048 (Intel)
MOS Technology	KIM-1	I	6502
Alpha Digital Systems	Alpha Z-80	II	Z-80
Byte Inc.	BYT-8	II	8080A
Cromemco	Various	II	Z-80
Equinox	Equinox System	II	8080A
IMSAI	8080	II	8080A
IMSAI	80/30	II	8080A
MITS	Altair 8800b	II	8080A
North Star	Horizon	II	Z-80A
PolyMorphic Systems	POLY 88	II	8080A
Processor Technology	Various	II/IV	8080A
Vector Graphic	Vector 1	II	8080A
Vector Graphic	Vector 1+	II	8080A
Digital Group	Various	III	Z-80, 8080, 6800, 6502
Heathkit	H8	III	8080A
Heathkit	H11	III	LSI-11 (DEC PDP-11)
Intelligent Systems	Intecolor 8001	III	8080A
Midwest Sci. Instr.	MSI-6800	III	6800
MITS	Altair 680B	III	6800
Ohio Scientific	Challenger II	III	6502A
Southwest Technical	SWTP 6800	III	6800
Apple Computer	Apple II	IV	6502
Commodore	PET 2001	IV	6502
Ohio Scientific	Challenger IIP	IV	6502A
Radio Shack	TRS-80	IV	Z-80

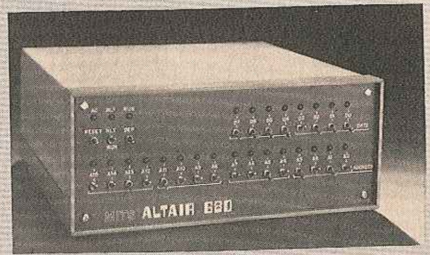
I—Microcomputer on a board.
II—S-100 type.
III—Non-S-100 type.
IV—Turnkey system.

forms handshaking (a constant back-and-forth verification of the transmission of data) between the CPU and the device. The controller logic again may be contained on a portion of the single-board microcomputer (represented as dashed lines in Fig. 1), or it may be a separate module. Generally, for each new peripheral device that is added to the system, a new controller board must be attached to the bus or to an I/O port.


ACT-IV CRT TERMINAL from Micro-Term has alphanumeric keyboard.

Two general types of software are available: *system* (or *utility*) *software* and applications *software*. System software includes programs to help develop other programs, debug or troubleshoot other programs, or manage data in memory or in secondary memory storage, such as a floppy disc or audio cassette tape. System software can be supplied in firmware or on an easily *loadable* program medium, such as audio cassette tape or paper tape. Applications programs range from game packages to accounts-receivable programs, in fact, any program for which the microcomputer system will be used.

Although system software may be supplied in firmware as part of the system, application programs generally come separately in the form of audio cassette tapes, listings and documentation. Not all microcomputer manufacturers supply huge quantities of applications programs. For this reason and for the fun involved, you can write your own applications (or systems!) programs. There are three types of programming available to the user—*machine language, assembly language* and *higher-level languages*.


ALTAIR 680 FROM MITS has toggle switches and discrete LED indicators on front panel.

Machine language is the most basic, tedious and (although some will disagree) least enjoyable of the three methods of programming. Rarely necessary on today's systems, it requires you to enter strings of binary data to program the computer. *Assembly language* allows you to automatically assemble the instructions for the microprocessor being used in the microcomputer, thus eliminating the manual machine language method. Although machine language and assembly language programs can be executed much faster than other higher-level languages, they are more time-consuming to write. *Higher-level languages* such as BASIC let you program statements that resemble the English language. BASIC is by far the easiest language to learn and the most readily available for microcomputers. Other languages that are offered are FORTRAN, COBOL or APL. Figure 2 compares the three programming types used for a simple program. For more details on the three major programming methods, see "How To Program A Computer" elsewhere in this special section.

```
MACHINE LANGUAGE
As input on hexadecimal key pad:
            A8
            0E
            0A
            81
            OD
            C2
            00
            01
ASSEMBLY LANGUAGE
As input on keyboard:
                  XRA  A
                  MVI  B, 10
          LOOP    ADD  B
                  DCR  B
                  JNZ  LOOP
HIGHER-LEVEL (BASIC) LANGUAGE
As input on keyboard:
            350  LET S=0
            360  FOR I=1 to 10
            370  LET S=S+I
            380  NEXT I
```

FIG. 2. ADDING 1+2+3 . . . +10 in three languages

Available microcomputers

Currently, four basic personal microcomputer types are available: The *microcomputer on a board*, the *non-S-100 modular microcomputer*, and the *turnkey* system. Table II lists the lower-cost systems of the four microcomputer types.

The *microcomputer on a board* format is exemplified by such systems as MOS Technology's KIM-1. Other microcomputers of this type

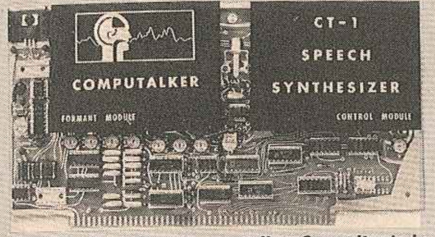
SYSTEM 8813 from Polymorphic has three floppy disk drives.

are offered by microprocessor manufacturers as single-board evaluation modules. These are designed primarily for the engineer who will either evaluate the microprocessor or integrate the module in a production system.

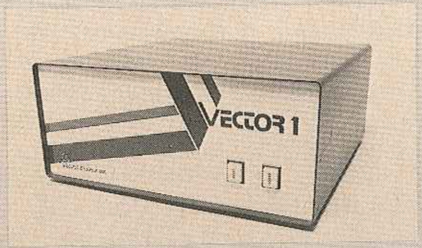
This type of module (available either as a kit or fully assembled) may include an LED display, a small numeric keyboard, a small systems software program in the firmware and a small user-accessible RAM memory. In some cases, additional memory or I/O devices are added to the basic system. Programming is generally possible only by using machine language.

The advantages of this system are its low cost, hardware simplicity, small size and low power drain. Its disadvantages are its lack of expandability (there are few add-ons available) and its less-sophisticated software. This type of microcomputer is probably ideal for an experimenter who wants to obtain the most basic hardware and software in order to learn and understand how both aspects of a personal computer function.

The *S-100 modular microcomputers* started with the MITS 8800.


MODEL CT-1 from Computalker Consultants is a speech synthesizer board.

MITS defined the MITS (or S-100) bus, and many other manufacturers copied it in their microcomputer designs. There are literally dozens of memory boards, I/O devices and special-purpose interfaces that plug directly into the S-100 bus. Most are compatible with each other, even if they are produced by different manufacturers. A control panel may or may not be offered. Firmware is not stressed but may be provided. In general, the S-100 microcomputers cannot simply be plugged in and operated. Kits or fully assembled systems are available.


VECTOR 1 MICROCOMPUTER from Vector Graphic.

This system's advantages are its low cost, a wide range of add-on peripherals and sophisticated hardware and software. The chief disadvantage is the lack of good general-purpose firmware (preprogrammed plug-in PROM's to take care of things like programming language).

The third group of microcomputers do not use the S-100 bus and provide a modular, rather than a turnkey, system. Since the S-100 bus is not used, additional modules or peripheral devices that are compatible with the system are probably provided only by the manufacturer of the system—for example, the Southwest Technical Products SWTP-6800. As with S-100-type microcomputers, systems in the non-S-100 group are available as kits or fully assembled. The primary advantage of this group is their somewhat lower cost. The greatest disadvantage is their reliance upon the manufacturers' own add-on devices.

The last group of microcomputers are the *turnkey* (plug it in and turn it on) systems, as for example, the Radio Shack TRS-80. These systems are fully assembled, warrantied and ready to operate. Firmware includes an operating system and a BASIC interpreter; a keyboard, video display and audio cassette interface are usually provided.

The advantages of this group are the low cost, the "plug it in and turn it on" configuration, the integrated systems software, and a wide range of user-developed application programs. The disadvantages are the possible limited expandability and the reliance upon manufacturers' own add-on modules and peripheral devices.

This represents a rather brief summary of the current personal computer marketplace. Although there are many factors involved in selecting a microcomputer, unlike choosing a wife, the purchaser won't be rejected if he has the cash! **R-E**

Peripherals-arms and legs

To perform useful functions, a computer needs peripheral devices. These devices interface between man and machine and greatly expand a computer's capability.

KARL SAVON
SEMICONDUCTOR EDITOR

PERIPHERALS ARE THE ARMS, LEGS, mouth and ears of the microcomputer. Without them, the computer is no more than a theoretical curiosity. Peripherals perform two basic functions: First, they interface between man and machine. They appear in the form of terminals, graphic displays and printers. Second, they expand the computer's capability from theoretical to highly practical applications through the use of such devices as cassette recorders, floppy-disk drives and analog interfaces.

Terminals

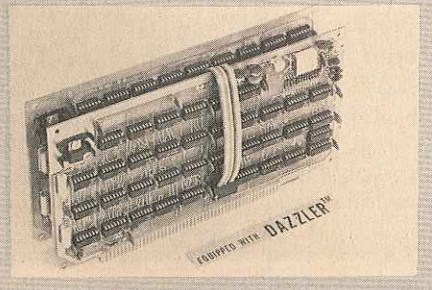
The most basic peripheral needed for a microcomputer is a terminal that lets you talk to the machine, program it, enter data into it and receive answers from it. Toggle switches and LED displays are adequate for a learning system, but once past basics it is vital to be able to move information in and out at conversational and at higher rates. Program-development cycles require an efficient link between you and the assemblers, compilers, interpreters and file management software. Even though there are diverse forms of data terminals, the most generally used is a keyboard-display combination.

A terminal, unlike a typewriter, consists of two electrically separate parts that are actually *physically* separated in some systems. The keyboard-driven transmitter usually has a standard typewriter key arrangement, plus several control keys. Often, only upper case letters are implemented and separate numeric keypads are built-in for convenience. On the receive end of the terminal, the most common devices are impact ribbon printers and video display.

For the computer and terminal to understand each other, they must speak the same language. Several standard formats have been composed so that system components from different manufacturers can work together:

- The predominant format in use is the **American Standard Code for Information Interchange (ASCII).** It is a 7-bit code plus a parity bit with a particular combination of bits corresponding to each character in its vocabulary.
- Some older equipment use a five-level **Baudot code** that was developed for telegraphic purposes. Its basic set of 2^5 or 32 codes is nearly doubled in keyboard characters, using letter and number keys that precede characters in one of two sets. Although similar in function to a typewriter shift key, these keys are not depressed simultaneously with other keys.
- The **EBCDIC code** is an eight-level language used by IBM and others that has twice the characters of ASCII ($2^8 = 256$ vs. $2^7 = 128$), and is suited for more complex control and graphic applications.

Diode or IC encoders convert key closures into ASCII or other codes.

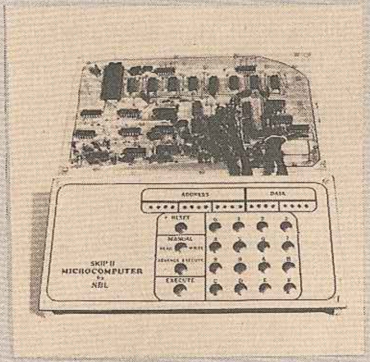


DAZZLER from Cromemco interfaces between S100 bus and TV set to add graphics.

These relatively slow, operator-limited terminals use serial data flow over one or two wires (plus ground). Parallel-to-serial conversion in the transmitter converts the parallel output of the keyboard-encoder into the serial bit stream, and corresponding serial-to-parallel conversion and decoding are performed in the receiver. UART integrated circuits have been designed that perform the conversion and the related tasks of parity-checking and start-and-stop bit generation.

When the ASCII code is combined with start-and-stop bits, sequences of 10 or 11 bits represent each transmitted or received character. Transmission is asynchronous, which means it does not require a transmitted clock signal.

The ones and zeroes of the data stream on the wires that connect the terminal and computer are distinguished by two DC levels. Several



SKIP II MICROCOMPUTER by NBL has hexadecimal keyboard and LED indicators.

different voltage or current levels are used for compatibility with TTL, RS232C (bipolar) or 20- or 60-mA current-loop equipment.

The terminal receive-and-transmit sections are independent of each other. It is only by electrical interfaces in the terminal or through the computer that the printer mechanism or CRT screen displays the keyboard data entries. In a full-duplex system, the computer and terminal communicate in both directions simultaneously. The computer is programmed to echo back what is transmitted to it. Control characters would normally produce nonprinting codes, but the computer can return intelligent responses; for example, "control C" or "↑C" when the control-C combination is typed on the keyboard. In half-duplex systems, communication takes place in one direction at a time. In most cases the terminal is wired so that the printer or display responds directly to the keyboard because the computer does not have time to echo the response. Otherwise, after you typed a letter you would then have to wait for the computer to return the character before you could type a second letter. Inadvertently hooking a half-duplex terminal to a full-duplex system produces strange results—typically, the double-printing of pressed keys. Many terminals can be switched between half-duplex and full-duplex modes.

IBK-1 MICROCOMPUTER from IMSAI has built-in card cage, CRT terminal and power supply

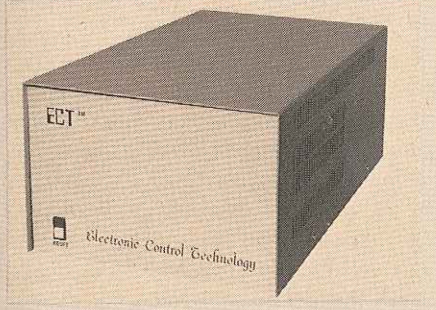
Terminal data rates are from 10 to 30 characters-per-second for strictly manual keyboard devices. Those keyboards having built-in paper tape or magnetic tape storage have data rates of 30 to 480 characters-per-second and even higher.

Keyboards may have 64 or 128 ASCII characters, and may have such features as numeric pads, tactile feedback and rollover. During an entry sequence, a fast typist, or a slower one with an uneven technique, may depress more than one key at any given instant. N-key rollover (NKRO) recognizes the key closures in the time order in which they are entered, and does not become confused when more than one key remains depressed. Two-key rollover (2KRO) is a less-sophisticated method that only recognizes two key closures simultaneously. The third and subsequent closures are ignored.

Keyboards are classified as *contacting* and *noncontacting*, depending on how the electrical current path is interrupted. Key pressure sensing may be performed mechanically, capacitively, photoelectrically, or by using reed switches, saturating cores, Hall-effect devices or elastomers.

Hard-copy printers are classified as *impact* and *nonimpact*, each with its own particular advantages and drawbacks. Impact printers use mechanical hammers, wheels or dot-matrix pins that strike the paper through an inked ribbon, and one column or one line is printed at a time. Impact printers are low-cost and can make copies, but they are generally noisy, may have poor print quality and are not as reliable as well-designed nonimpact printers.

Nonimpact printers are fast and quiet. One type uses ink jets that are sprayed onto the paper and controlled electrostatically. *Electrolytic printers* pass an electric current through chemically treated paper with wires or pins. *Electrostatic printers* use precharged dielectric-coated paper that is passed through a toner. The toner has ink particles that are oppositely charged to the printed charges on the paper. *Thermal printers* use transistor dot-matrix printheads that move across temperature-sensitive paper. Character fonts are electrically stored in read-only-memories (ROM's).



ECT MICROCOMPUTER FROM Electric Control Technology has S100 bus structure.

Some stand-alone computer systems have a keyboard and video display or printer built right into the main housing. Sometimes a video or RF output is provided so a video monitor or TV receiver can be used as an economical display.

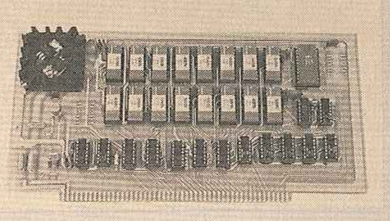
The video terminal is expected to become the main home-computer peripheral because of its low cost and graphic capability. Video terminals have editing options that allow corrections before the characters are fed to the computer. Other options include page and scrolling formats, character accents, selectable baud rates and cursor controls. Video terminals do not make hard copies. A preferred arrangement is a video terminal backed up by a printer that can generate a permanent record of the data that must be saved.

There is an intermediate class of terminals that use a TV display and a low-cost hex keypad. Primarily used in learning systems, they are adequate and even preferred for special applications where the input is mainly numeric.

The final choice of terminal must be made on the basis of cost, multiple-copy capability, speed, interface requirements, printing-vs.-plotting characteristics, font, print quality and color, noise, ease of operation, serviceability, type of paper, and reliability.

Tape cassettes and cartridges

As system and user programs grow in length and number, you must find a sensible way to store them. Virtually all microcomputer manufacturers have recognized the advantages of using standard audio cassette recorders for program storage. In-



RAM MEMORY BOARD from Electronic Control Technology mates with the S100 bus.

terfacing hardware and software are important resident or optional parts of their systems. Frequency shift keying (FSK), pulse-width modulation (PWM), phase encoding (PE), nonreturn-to-zero (NRZ), biphase encoding, group-coded recording (GCR), Manchester coding and the Kansas City Standard (KCS) are some methods used to both store data on audio tape and retrieve it. Phase encoding is popular because it is self-clocking, making it tolerant of speed variations and has good noise performance. Manchester coding is a type of phase-transitional encoding in which positive signal transitions represent ones and negative transitions represent zeroes. The KCS method is a variation of Manchester code that represents zeroes as 1200-Hz tone bursts and ones as 2400-Hz bursts.

Although at one time paper tape was very popular, it is being gradually displaced by digital magnetic-tape cartridges and cassettes that offer much greater reliability than either paper tape or audio cassettes. Lower-performance machines use drives that have evolved from audio recorders. In these systems the tape is caught and held between a capstan drive shaft and a free-wheeling pinch roller. At higher tape speeds and data densities, direct servomotor drives are used. Two motors drive the cassette hubs in a very clean mechanical arrangement.

Digital recorders use phase encoding with typical tape densities of 800 bits-per-inch. They have versatile control functions including write, stop, reverse one block or line, read one block or line, read continuously, erase, rewind for cassette removal and wind past tape leader. Extensive error checking is performed for dropouts, bit timing and character parity. The unformated capacity of a cassette is about 720 kilobits and data transfer rates are 24 kilobits-per-second.

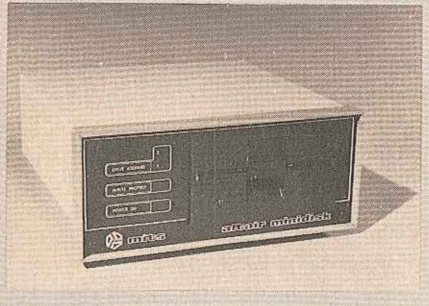
Cartridge drives provide improved performance and are used wherever higher operating speeds (48 kilobit-per-second transfer rate) and greater storage capacity (2.9 kilobits) are needed. Quarter-inch tape is recorded with densities of 1600 bits-per-inch on up to four tracks. Cartridge-drive manufacturers plan to compete with floppy disks by reducing access times. But for the present the tape heads and system mechanics limit search speeds to about 30 inches-per-second.

Floppy disks

Given enough financial resources, there are very few computer enthusiasts who would not invest in a large-capacity, floating-head, magnetic disk system. Random-access addressing of large amounts of stored data makes such a system look almost like a huge chunk of RAM. Some ad-

RADIO SHACK TRS-80 microcomputer shown with video monitor.

vanced computer systems exchange segments of disk memory with main computer memory to give the computer the *virtual* appearance of a machine with a very large main memory. Real-life budgets have forced computer manufacturers alike to find an economical compromise. The floppy disk does not have the data capacity of cassettes and cartridges, but is a random-access device with almost 500 times faster access. The *Mylar* disk is 9 inches in diameter and 0.003-inch thick, coated with a magnetic oxide and protected in an 8-inch-square jacket. The jacket has holes through which the spindle rotates the disk at 360 RPM, the head contacts the disk, and (on hard-sectored disks) the index holes are exposed. Hard-sectored disks use photoelectrically sensed index holes to locate data; soft-sectored disks rely on reading the block headings recorded on the disk.


FLOPPY DISC DRIVE from Mits adds additional memory to microcomputer system.

On IBM-compatible disks, data is recorded on 77 concentric tracks that are numbered 00 to 76 from the outside of the disk in. Track 00 holds labels and system and disk information, two tracks are reserved as alternates, leaving 73 tracks for data. Each track has 26 sectors, and each record or sector stores 128 bytes. This calculates to a total of 73 × 26 × 128 = 242,944 bytes. There are double-sided, higher-density floppy disks and other non-IBM-compatible disks with greater capacity.

In a step to reduce costs even further, the *minifloppy* disk was created, packaged in a 5¼-inch-square jacket. Hard-sectored *minifloppy* disks have 16 records-per-track and soft-sectored 18 records-per-track. The record length remains 128 bytes, with either 35 or 40 tracks. Maximum storage capacity is then 18 × 40 × 128 = 92,160 bytes (one byte = 8 bits). Data transfer rates are about 125 kilobits-per-second. It is rumored that a $100 *mini-floppy* drive will soon appear on the market.

Floppy disks also have their difficulties, the main problem being much higher wear than their floating-head big brothers. Disk life is typically specified in the millions of passes-per-track.

Other equipment

Having deliberately skipped over terminals different than the keyboard-printer, it is time to backtrack a little. A terminal is any device that reads in and writes out information; it can be equipment that handles punched paper tape or cards, magnetic cards and tape, bar codes, and so forth.

Paper tape is useful if you own an ASR *Teletype* machine with its built-in paper-tape reader and punch. With no additional expense except for the paper tape, you have a readily available storage medium. Unfortunately, this system is also slow and noisy. More sophisticated photoelectric readers and high-speed punches are available but they are fairly expensive. Low-cost optical readers eliminate some of the complexity and expense: The tape is pulled manually through the reader. This type of equipment is attractive if you have quite a lot of paper-tape software available.

Card readers and punches are not very popular. They are most familiar in the 80-column by 12-row format. One of their peculiar advantages is the ease with which a single program statement or data line can be modified without upsetting the rest of the program.

The line printer is an extension of the terminal printer or display, except it tends to be a high-speed device (for example, 125 lines-per-minute with each line having 132 characters) used when there are many pages of data to be printed.

Plotters use servocontrolled coordinate positioning arms to produce permanent graphic records. They differ from chart recorders in that the stylus can draw complex patterns back and forth over the stationary paper.

If a terminal is situated far away and telephone lines or radio waves are used as the connecting medium, some type of communications interface or modem (modulator-demodulator) is needed. Tone and phase-modulation schemes are used, and error-correcting codes are added to enhance reliability. Acoustic couplers and modems are either built into terminals or are separate entities. The coupler connects to a telephone handset and converts the received telephone tones into an electrical output signal. It also converts an electrical input signal from the modem into sound that drives the phone's transmitter element. A portable terminal can be taken anywhere and the computer called up for field access. The modem performs the phase or frequency encoding to condition the digital bit stream for the communications channel.

Charged-coupled mass memories may become competitive with cassettes, cartridges and disks in the near future. Access is completely electronic and about 50 times faster than a high-speed disk. There are some devices already on the market with capacities of 1 million bytes.

If your application is highly scientific or mathematic and you perform such operations as evaluating matrices, you may find that your BASIC interpreter or other software is much too slow. Peripheral floating-point hardware solves such problems with their high-speed parallel computation methods.

Interfaces

Interfacing between the computer and the peripheral device is an essential consideration. The interface can be as expensive as the peripheral itself. Unless the peripheral is directly compatible with the computer's voltage levels and timing, a controller will be necessary to match up the two. A widely adopted technique minimizes hardware modification by using software drivers stored in ROM or RAM to produce the signals needed by the peripheral. Software or firmware (ROM) approaches may adversely affect the performance of the peripheral or the entire system. Before rushing out to purchase any peripheral device, it is best to know exactly what other hardware and software components are needed to make it work (and at what performance level) with a particular computer system. **R-E**

how to Program A Computer

To get a computer to do what you want it to, you must communicate with it. Here's a look at three different levels of programming and how to use each one.

ART KLEIMAN
MANAGING EDITOR

THERE'S NOTHING DUMBER ON THE face of this earth than a computer. Oh boy, are they ever stupid! Go ahead, make one perfectly human mistake when writing or entering a program, hit "go" and. . . . . nothing! A pathetic 20th-century staring match between man and machine. So, instead of finding out the winner of that match, let's grab the latest computer catalog.

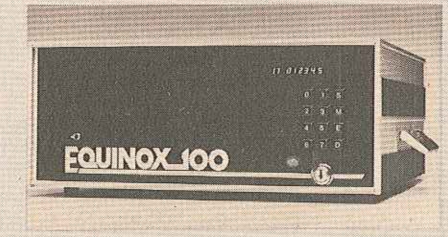
What are we looking for in the catalog? Don't be silly, we're looking for help. Help comes in the form of a higher-level language. So, we scan past the paragraphs describing the CPU boards, memory, I/O, peripherals, etc., and go right to the software section. Here, we're bombarded with terms like machine code, assembly language, interpreter, compiler, BASIC, FORTRAN, APL, EMPL. . . . . . . .whew! Before you blow a human fuse, let's take a look at what these terms mean, and what they mean to the computer. To get a better understanding, however, let's first get back to basics.

How a computer is programmed

Digital computers are digital because they're built using digital logic IC's. And digital logic IC's eat, chomp and spit out binary numbers (zeroes and ones). That's it! It doesn't matter whether we're discussing the data base at the Pentagon or the most simplistic single-board microcomputer trainer. The only things flowing through those circuits are zeroes and ones.

The heart of the computer is the CPU, or microprocessor. This nifty device performs many functions, such as adding, subtracting, comparing, etc., in response to instructions. In other words, it can do many things, but it must be told what to do by giving it an instruction. Remember, this is a digital device, so when it adds or subtracts, it adds and subtracts binary numbers (zeroes and ones) and the instruction given to it must be in the form of a binary number.

If the microprocessor has one thing going for it, it's speed. It can perform many operations in a very short time. If we wanted to perform just one operation, we wouldn't need a computer. So, to perform many operations (hundreds, thousands, or even hundreds of thousands) very fast, we must feed the instructions to the microprocessor just as fast. To do this, we connect the microprocessor



EQUINOX 100 computer from Parasitic Engineering has an octal keyboard.

to a memory. The instructions and the necessary data are stored in the memory and called a program.

Inside the microprocessor is a register (a device for the temporary storage of a binary number) called a program counter. After storing the program in the memory, the program counter is set to the beginning address (location) of the program. Then, you hit GO (instruct the microprocessor to begin execution). The program counter addresses the first memory location and the memory responds by feeding the first instruction to the microprocessor. After the microprocessor does its thing by executing the instruction, the program counter is incremented (advances) by one and addresses the next memory location. Again, the memory responds by feeding the next instruction to the microprocessor. The microprocessor executes this instruction and again increments the program counter by one. This process continues until the entire program is executed.

So, programming a computer means that you have to store the program in the memory. This is done by writing and then entering the program in the computer's memory. This can be done at three different levels—machine language, assembly language and higher-level language.

Machine-language programming

You could write the program in digital form and then enter the program into the memory one digit at a time. This is machine-language programming. Figure 1 shows a simple program for adding two numbers together. The program is written for the 8080 microprocessor, one of the more popular 8-bit microprocessors. (A bit is a binary digit and a byte is 8 bits.) The 8080 is an 8-bit microprocessor because it physically has eight data lines. These data lines are used to input instructions and data and also to output the results. The 8080 also has address lines for addressing memory. However, eight address lines aren't enough. Eight bits would enable the 8080 to address only 256 memory locations, hardly enough program space to do anything except add two numbers together. So, internally, the 8080 gangs two bytes together and feeds them out on 16 address lines. Sixteen address lines enable the 8080 to address 65,536 memory locations—quite a lot! To feed a memory location to the microprocessor on the eight data lines, however, requires two operations. First, the low-order address byte (the eight least significant bits of the memory address) is input and then the high-order address byte (eight most significant bits of the memory address) are input. This idea is implemented in the simple addition program shown in Fig. 1. Let's take a look at it and see what happens at each step of execution.

The addition program occupies 15 memory locations starting at location 0 (represented by the 16-bit binary number 0000000000000000). Basically, the program takes the binary number stored in a specific memory location, adds it to the binary number stored in another memory location and stores the result in a third memory location. For discussion purposes, the left-hand column in Fig. 1 numbers each program step the way you're used to seeing numbers—in decimal form. This will make it easier for us to refer to each program step.

The very first memory location (program step 1) contains an instruction that tells the microprocessor to load the HL register pair with the immediately following two data bytes. Both the H and the L registers are 8-bit registers located inside the microprocessor IC. When told to do so by the instruction, the microprocessor gangs these two registers together to form a single 16-bit register, called the HL register pair. The 16-bit HL register pair is used to specify a 16-bit memory location. The microprocessor instruction is also called the *op-code*.

Now, back to the program. The next memory location (program step 2) contains the low-order 8-bits of the memory address, which is to be loaded into Register L of the HL

SOL TERMINAL COMPUTER has computer and alphanumeric keyboard in a single enclosure.

register pair (by program step 1). Program step 3 contains the high-order 8-bits of the memory address, which is to be loaded into Register H of the HL register pair. After the microprocessor completes program step 3, the HL register pair contains the memory address 0000000000001100. This memory location, coincidentally, contains the first number of our addition problem. Program steps 4, 5 and 6 do the same things that program steps 1, 2 and 3 do but with the DE register pair. Here, the DE register pair is loaded with memory address 0000000000001101. This memory location contains the second number of our addition problem. So far, we've got the HL and DE register pairs loaded with two memory addresses, each one of the memory locations containing one number of our addition problem.

Program step 7 contains an instruction that tells the microprocessor to load the accumulator with the contents of the memory location specified by the DE register pair. The accumulator is another internal register within the microprocessor. This register is the working register. It is used to manipulate data, perform math and logic operations, store results, etc. The accumulator is involved with just about every operation the microprocessor performs. So far, we have the accumulator loaded with the first number of our addition problem.

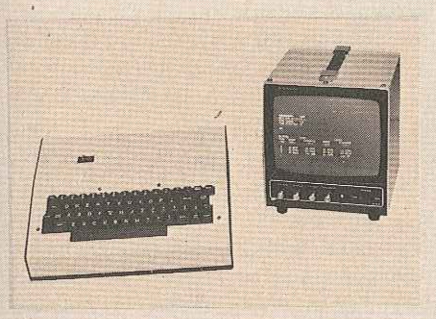
Program step 8 is where the two numbers are actually added. This instruction tells the microprocessor to add the contents of the accumulator with the contents of the memory location specified by the HL register pair. The result of the addition operation is stored in the accumulator. We could leave the result of the addition in the accumulator if we wanted to, but in actual practice this is hardly ever done. The result is either fed to an output port that is connected to a CRT terminal, a printer or something similar, or it can be stored in another memory location. Since we don't have any peripheral equipment connected to our computer, we'd simply store the result in another memory location. Program step 9 does just this.

Program Step	Memory Location		Instruction Data	Comment (Meaning of Instruction)
	High-Order	Low-Order		
1	00000000	00000000	00100001	Load the HL register pair with the two immediately following data bytes
2	00000000	00000001	00001100	Memory address, low-order byte
3	00000000	00000010	00000000	Memory address, high-order byte
4	00000000	00000011	00010001	Load DE register pair with the two immediately following data bytes
5	00000000	00000100	00001101	Memory address, low-order byte
6	00000000	00000101	00000000	Memory address, high-order byte
7	00000000	00000110	00011010	Load accumulator with contents of memory location specified by DE
8	00000000	00000111	10000110	Add accumulator contents of memory location specified by DE. Store results in accumulator.
9	00000000	00001000	00110010	Store accumulator in memory location specified by two following data bytes
10	00000000	00001001	00001110	Memory address, low-order byte
11	00000000	00001010	00000000	Memory address, high-order byte
12	00000000	00001011	01110110	Halt
13	00000000	00001100	XXXXXXXX	First number of addition problem
14	00000000	00001101	XXXXXXXX	Second number of addition problem
15	00000000	00001110	XXXXXXXX	Result of addition

FIG. 1—MACHINE-LANGUAGE PROGRAM for adding two numbers together. The actual program consists of the instruction/data that is stored in the corresponding memory locations.

Program step 9 contains an instruction that tells the microprocessor to store the contents of the accumulator in the memory location specified by the immediately following two data bytes. Program step 10 contains the low-order byte of the memory address, and program step 11 contains the high-order data byte of the memory location. Now, we have the result of the addition operation stored in memory location 0000000000001110.

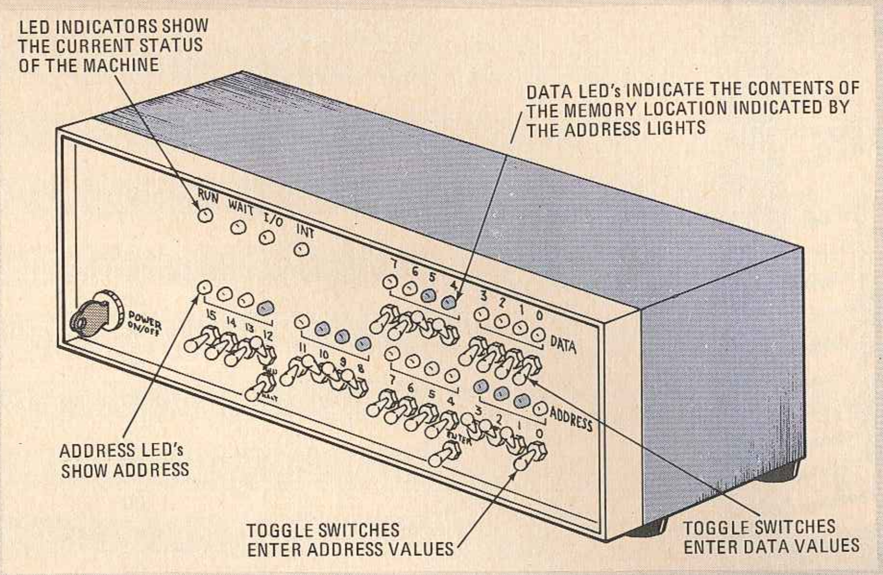
Program step 12 is a HALT instruction. This tells the microprocessor that the program is finished and to stop execution. Program step 13 is the memory location containing the first number of the addition problem, program step 14 is the second number in the addition problem and program step 15 is the result of the addition operation.


ACT-I TERMINAL has alphanumeric keyboard. Monitor shows how information is displayed.

To run the program, we would simply load program steps 13 and 14 with the two numbers and set the program counter to 0000000000000000. Then hit GO. You could examine the contents of memory location 0000000000001110 to obtain the result.

Entering a machine-language program

Now that you know what a machine-language program is and what it looks like, let's take a look at how we load it into the machine. The program isn't fed to the microprocessor; it's stored in the memory. This means that we would have to address the right memory location and store the data/instruction at that location. We could do this by using toggle switches on the front panel of the computer—16 toggle switches connected to the address lines and eight toggle switches connected to the data lines (see Fig. 2). We would also need another toggle switch to instruct the memory IC's to store the data on the data lines. This switch could be labeled STORE, WRITE, ENTER, or anything appropriate. An example of how one such toggle switch is connected is shown in Fig. 3. One position of the toggle switch would be labeled 0 and would ground the line, the other position would be labeled 1 and connect the line to +5 volts.


LED INDICATORS SHOW THE CURRENT STATUS OF THE MACHINE
DATA LED's INDICATE THE CONTENTS OF THE MEMORY LOCATION INDICATED BY THE ADDRESS LIGHTS
RUN
WAIT
I/O
INT
POWER ON/OFF
15 14 13 12
11 10 9 8
7 6 5 4
3 2 1 0
DATA
ADDRESS
ENTER
ADDRESS LED's SHOW ADDRESS
TOGGLE SWITCHES ENTER ADDRESS VALUES
TOGGLE SWITCHES ENTER DATA VALUES
FIG. 2—TYPICAL FRONT PANEL of computer showing toggle switches and LED indicators.

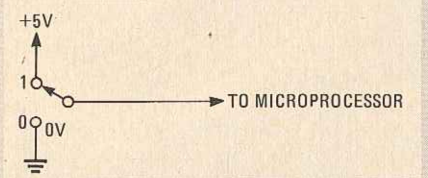


FIG. 3—FRONT PANEL TOGGLE SWITCHES apply +5 volts to microprocessor in one position, and ground in the other position.

Now, we're set. Well, almost. The toggle switches permit us to enter the program, but there is no way we can check the data stored in the memory to see if we entered it correctly. So, we connect eight LED's to the data lines and install them on the front panel along with the toggle switches; a lit LED signifies 1. Now we are set.

To load the program into memory, set the address switches to the first memory location, set the data switches to the first instruction and then hit ENTER. Then, set the 16 address switches to the next memory location, set the eight data switches and again hit ENTER. Continue this procedure until the entire program is loaded. Be careful, though, don't make any errors. If you do, the program won't run. Make a mistake entering just one bit, just one, and the program won't run.

As you see, entering a program this way can be quite tedious. Of course, with short programs such as the one shown in Fig. 1, the procedure isn't too bad. But with longer programs that really do something, the situation becomes more complicated. Some programs can run 2000, 5000, 10,000 program steps or longer, and the situation gets a lot worse. With patience and diligence, however, you can become quite good at entering programs this way. With a lot of practice, you can whiz your fingers across those toggle switches so fast that you will dazzle your family and friends. However, those toggle switches may grow limp with fatigue, not to mention your fingers.

This type of front panel was common in the early microcomputers. One of the very first hobbyist computers was the Mark 8 and it had a similar front panel. It was a construction project published in the July 1974 issue of **Radio-Electronics.** The first commercially available microcomputer was the Altair, which also had a similar front panel. Another early microcomputer was the Im-

Program Step	Memory Location		Instruction Data
	High Order	Low Order	
1	00	00	21
2	00	01	0C
3	00	02	00
4	00	03	11
5	00	04	0D
6	00	05	00
7	00	06	1A
8	00	07	86
9	00	08	32
10	00	09	0E
11	00	0A	00
12	00	0B	76
13	00	0C	XX
14	00	0D	XX
15	00	0E	XX

FIG. 5—HEXADECIMAL MACHINE-LANGUAGE program is identical to the program shown in Fig. 1, except the binary numbers are represented by their hexadecimal equivalents.

sai, again with the same type of front panel. Both the Altair and the Imsai are popular microcomputers, still going strong in the marketplace. That's because they can be programmed using higher-level languages. You'll discover what these languages are later on in this article, but now let's get back to the front panel.

It's obvious that using toggle switches is no way to enter a machine-language program. There must be a better way, and there is. Let's sit back for a moment and take a look at that 8-bit data word we are entering. Is there another, shorter way to represent those same 8 bits? If we split these 8 bits up into two groups of 4 bits, we could represent each 4-bit group by a *hexadecimal* digit. A group of 4 bits provides 16 possible combinations of zeroes and ones. If we want to represent each of those 16 combinations with a single digit number, we need a numbering system to the base 16. Figure 4 shows the 4-bit group and the corresponding hexadecimal digit. Don't be confused—here A, B, C, D, E and F represent numbers, not letters. Now, if we recombine those two 4-bit groups back into a single 8-bit group, we can represent those 8 bits with 2 hexadecimal digits. For example, 00000000 would be represented as 00, and 11111111 would be represented as FF.

Binary	Hexadecimal
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

FIG. 4—FOUR-BIT BINARY WORD and corresponding hexadecimal value.

CONNECT A TV SET to the Apple II computer and you have a computer and CRT terminal.

Now we can construct a front panel using a hexidecimal keypad rather than toggle switches. Instead of the discrete LED's, we can use a 2-digit LED alphanumeric display for the data. (Remember in hexidecimal we count 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, and so on. So we need 2 digits of readout, each one capable of counting from 0 through F.) We'll also have to include some circuitry to decode the hexadecimal digits into the corresponding 8-bit word. Since the memory address requires 16 bits (2 bytes), we can represent it by using 4 hexadecimal digits.

To use this type of front panel, first switch to the MEMORY mode and enter the starting memory address. Figure 5 shows the simple addition program shown in Fig. 1 but in hexadecimal format. If you wanted to enter the program shown in Fig. 5, you would select the MEMORY mode and enter 0000. You would then select the ALTER mode and enter the first instruction, which is 21. Memory location 0000 would automatically be loaded with 21, and the memory address would be advanced by 1. The next data byte would then be entered by simply hitting 0 followed by a C. This process is continued until the entire program is entered. If while you were entering the program, you came across a memory location that you did not want to alter, you would simply hit the MEMORY INCREMENT switch and the contents at the memory location would remain intact. Or you could re-enter the contents at that memory location. In any event, entering machine-language programs in hexadecimal form is easier, both on the eyes and on the fingers, and there is much less of a chance of making a mistake.

Are binary and hexadecimal the only two ways to enter a machine-language program? No. There is a third way being used by a few manufacturers. The 8-bit data word is again split up but this time into three groups instead of two. The first group contains 2 bits and the last two groups contain 3 bits each. Since the maximum number of bits in any group is 3, we could use the octal number system to represent the binary number contained in each group. Figure 6 shows the 3-bit groups and the corresponding octal number. Since the first group contains only 2 bits instead of 3 bits, this group will have a maximum octal number of 3. Since the other groups contain 3 bits, they will have a maximum possible octal number of 7. We then recombine the groups to form a single 8-bit data word, just like we did with the hexadecimal system. The octal numbers will range from 000, corresponding to the data byte (a binary number) 00000000, to 377 (equivalent to decimal 255), corresponding to binary 11111111.

Binary	Octal
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

FIG. 6—THREE-BIT BINARY WORD and corresponding octal value.

We can now build a front panel just like we did for the hexadecimal system. But this time instead of using a hexadecimal keypad, we would use an octal keypad. Instead of using a 2-digit LED display for the data, we would use a 3-digit LED display. The memory address requires a 6-digit LED display instead of a 4-digit LED display. Otherwise, the front panel remains basically the same. Even the way we enter the programs remains basically the same. To see what an octal-based machine-language program looks like, the simple addition program already discussed is shown in Fig. 7 in octal format.

One final note on front panels: Front panels from different manufacturers differ in their features and therefore in the way programs are entered. Some front panels have automatic incrementing features while others may not. Some front panels are octal while others are hexadecimal. The Heath H8 and the Equinox from Parasitic Engineering are examples of full-featured octal front panels (with display and keyboard). In any event, since the exact sequence of pushing buttons to enter a machine-language program does vary from computer to computer, do take a close look at the front panel before you buy your computer. Some computers are completely devoid of front-panel controls. This means that they have no switches or displays on the front panel other than a reset and/or a power ON-OFF switch.

Program Step	Memory Location		Instruction/ Data
	Low Order	High Order	
1	000	000	041
2	000	001	014
3	000	002	000
4	000	003	021
5	000	004	015
6	000	005	000
7	000	006	032
8	000	007	206
9	000	010	062
10	000	011	016
11	000	012	000
12	000	013	166
13	000	014	XXX
14	000	015	XXX
15	000	016	XXX

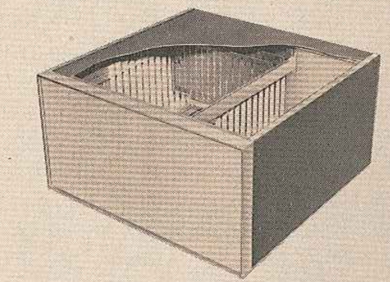
FIG. 7—OCTAL MACHINE-LANGUAGE PROGRAM is identical to the program shown in Fig. 1, except the binary numbers are represented by their octal equivalents.

The Alpha computer from TDL and the 6800 from Southwest Technical Products are examples of this type. To program this type of computer in machine language or any other language requires a keyboard connected to the computer's input port. Even though the nitty-gritty may be different, entering a machine-language program is basically (and I mean basically) the same no matter which computer you use.

Now you know what a machine-language program is and what it looks like. As you probably already guessed, it's no fun programming in machine language. The 8080 microprocessor has no less than 78 instructions, while the Z-80 microprocessor has 158. This means that if you were to program in machine language, you would need a list of the instructions and the corresponding op-code (the binary form of the instruction). Then, depending on the computer, you might have to convert the op-code into octal or hexadecimal. If you were lucky, you might even find an instruction list with the op-code already in hexadecimal or octal. In any case, having to look up the op-code for each instruction is slow and tedious. If you were ambitious, you could get around this by memorizing the entire instruction list and corresponding op-code for each instruction. But I, for one, do not relish the prospect of having to commit such a list to memory. There must be a better way, and there is.

Assembly language

Let's suppose that we were able to program the computer using alphanumerics instead of just plain numbers. We could code the instructions into a 3- or 4-letter group very similar to the original instruction, and the computer would translate this code group into the machine-language instruction. We could use a hexadecimal keyboard to speed the process. Certainly, a group of meaningful characters are much easier to commit to memory than a group of 8 zeroes and ones. Also, if we wrote this "translator" program so that it takes care of all the memory addresses, programming would be much easier.



VECTORS VP2 enclosure permits you to design your own microcomputer.

The translator program is called an assembler program and the 3- or 4-letter groups that stand for the instruction are called a mnemonic (*neemonik*—an easy-to-remember code word). A mnemonic is an abbreviated word that is intended to remind us of the original word. Let's take a closer look at mnemonics.

Program step 7 in Fig. 1 is an instruction that loads the accumulator with the contents of the memory location specified by the DE register pair. The corresponding mnemonic for this instruction is LDA (*LoaD Accumulator*). Program step 9 contains a store accumulator instruction. The mnemonic for this is STA (*STore Accumulator*). Another instruction is the halt instruction, and the corresponding mnemonic is HLT (*HaLT*). As you can see, mnemonics are definitely easier to remember than the binary op-code. In fact, we really don't even have to memorize the mnemonics. It's enough just to become familiar with them.

Let's take a look at what an assembly-language program is. Programming in assembly language is a little more difficult to grasp at first than machine language, but stick with it. We'll try to walk it through one step at a time. For discussion purposes, let's suppose you were writing the program down on a piece of paper. The assembly-language program is divided into four vertical columns. Each column is referred to as a field. The four fields are, from left to right, the LABEL, OPCODE, OPERAND and COMMENT fields.

So far you have a piece of paper divided into four columns. The LABEL field is always the first column. A label is a letter, or a group of letters, that reference a memory location. For example, suppose you assigned the first line of your assembly program with the label START. Then, if at any place in the program you had to refer to the starting address of the program, you could simply use the word START. The assembly program would automatically translate the word START into the starting address of the program. Or, suppose you wanted to store the result of a calculation in a memory location. You could assign a label to that memory location. Then, if you wanted to fetch the result from memory, you could simply use that label. You don't have to label every memory location when you write the assembly program if you don't want to. In fact, you don't have to use any labels at all. To avoid labeling, simply leave the first column (label field) blank.

The second column (the OPCODE field) contains the mnemonic abbreviation of the instruction. Some instructions, however, require more information than just the mnemonic. For example, suppose you wanted to store the contents of the accumulator in a memory location. You would use the STA mnemonic in the opcode field, but you would also have to specify the memory location where you want the accumulator stored. This information would go in the OPERAND field. Depending on the particular assembler you were using, you could specify the memory location as a 16-bit binary number, or an octal number, or hexadecimal, or even good old decimal. If you don't want to, you don't even have to worry about keeping track of memory locations. Remember the label field? If you assigned a label to the memory location, you could specify that memory location by using the label in the operand field. The operand field contains necessary information other than just memory locations. For example, suppose you used a mnemonic (instruction) that involved a register pair. The operand field would specify which register pair the instruction would operate on. In other words, the operand field contains any additional information that the particular instruction requires.

The last column, the COMMENT field, is not used by the computer at all. It contains documentary information that you, as a programmer, can add to help you understand the program. You can add whatever alphanumeric information you desire in this column.

Now that you have at least a basic understanding at what an assembly language is, let's look at an actual program written in assembly language. The simple addition program that was discussed throughout this article appears in Fig. 8 in assembly language.

The first line of the assembly-language program contains the *label* START. The first line also contains the first instruction of the program in the form of the mnemonic LXI. This LXI mnemonic instructs the microprocessor to load the register pair with the immediately following data bytes (X's in the mnemonic stand for register pairs.) The operand column leads off with an H. This defines the register pair as the HL register pair. Next comes the hexadecimal data that is to be loaded in the HL register pair. A comma separates the two pieces of information in the operand field. The second line of the program contains another LXI instruction, but this time, as you can see from the operand field, it operates on the DE register. The comma again separates the hexadecimal data that is to be stored in the DE register pair.

Label	Opcode	Operand	Comment
START	LXI	H, 000C	Load HL register pair with 000C
	LXI	D, 000D	Load DE register pair with 000D
	LDAX	D	Load accumulator with memory specified by DE
	ADD	M	Add accumulator and contents of memory specified by HL
	STA	000E	Store accumulator in 000E
	HLT		Halt
	END	START	Assembler directive, load program counter with starting address

FIG. 8—ASSEMBLY-LANGUAGE PROGRAMMING involves using mnemonics to represent machine-language instructions.

The third line of the program contains the mnemonic LDAX, which instructs the microprocessor to load the accumulator with the contents of the memory location specified by a register pair. The operand field specifies the DE register pair. In the next line of the program, the combination of the mnemonic and operand instructs the microprocessor to add the contents of the accumulator and the contents of the memory location specified by the HL register pair. The next line stores the contents of the accumulator in the memory location specified in the operand field. The next-to-last instruction is the HLT mnemonic, which instructs the microprocessor to stop execution.

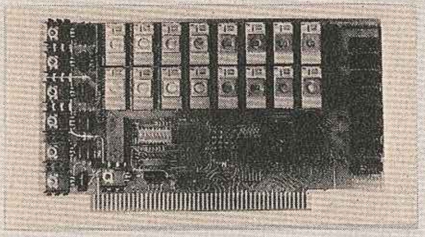
SOUTHWEST TECHNICAL'S 6800 system has optional floppy disc, printer and CRT terminal.

So far, as you can see, the assembly-language program is exactly the same as the machine-language program, but written differently in a much easier format to look at and understand than just binary numbers. The last line in Fig. 8 contains a mnemonic that is not a microprocessor instruction. The assembler program, in addition to understanding a mnemonic for each microprocessor instruction, understands several additional mnemonics called *assembler directives*. These are used to direct the assembler itself to do something. The mnemonic END tells the assembler that the program is complete and to stop assembling the program. The operand field contains the label START that loads the program counter with the starting address of the program. This means that after the program is assembled and loaded into the computer in that machine language, the program counter is set to the memory address that contains the first LXI instruction.

That is basically what assembly language looks like. Incidentally, the program shown in Fig. 8 is called a *source program*. It is in its raw form, it hasn't been translated into machine language. The assembler will translate into machine language, and Fig. 8 is the source for the assembler. However, thus far you only have the source program written down on a piece of paper. Now, you'll have to put it into a form that the assembler can use. This requires some additional equipment for your computer.

Since assembly language is written in alphanumerics, you'll need an alphanumeric keyboard and an alphanumeric display; both of these are contained in a CRT terminal. To connect the terminal to your computer, you'll need a parallel interface board if your computer doesn't already have one. You'll also need an external device to store your source and assembler programs, and then load them back into the computer when needed at a later time. The most common storage device used in personal computing is the audiocassette tape recorder. This device is inexpensive and surprisingly reliable. To interface the cassette recorder to your computer, you need a cassette interface.

You also need a program to drive the cassette interface. This software is a program that directs the microprocessor to transfer the contents of memory, one location at a time, to the cassette interface when you want to store a program on a cassette tape. It also contains a routine that takes the digital data stored on the cassette tape and transfers it to sequential memory locations. This software is called a *bootstrap program*. You'll

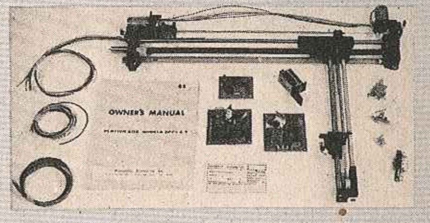

COMBINATION RAM AND ROM memory board mates directly with the S100 bus.

also need software to drive the CRT terminal. If you're lucky, your computer contains both these programs, as well as several additional systems programs, stored in a permanent memory called a ROM. The system software is called a monitor program. If your computer doesn't contain a monitor ROM, then you'll first have to load the bootstrap program into the computer through the front-panel switches. Once the bootstrap program is loaded, you can then load the rest of the programs using the cassette tape recorder.

Now that you have the necessary equipment, you can write your source program in assembly language.

Assembly-language programming

To write your source program on the CRT terminal requires an additional program called a *text editor*. The text editor makes it much easier to write text on the terminal. It provides you with various edit functions in case you make a mistake, it has a tab function for setting up columns on the terminal, and it even has a cassette file routine that enables you to transfer several source programs to the same cassette tape.


X-Y PLOTTER connected to computer permits hard copy drawings.

The text editor is loaded into the computer via the cassette tape recorder. Once this is loaded, you can write your source program on the terminal. The keyboard on the terminal is very similar to a typewriter keyboard. It has all the letters and numbers, and it even has a shift key, space bar and carriage return. So to enter the source program, you merely type it out as if you were entering it on a typewriter.

The source program is entered by typing it one line at a time. The four fields are separated by spaces. For example, to designate the end of the opcode field and the beginning of the operand field, you enter a space by hitting the space bar on the keyboard. There will be times when you run across a column with nothing in it. To get around this, you again hit the space bar. For example, if the first column, the label field, is to be left blank, simply hit the space bar and you'll automatically start the next field, which is the opcode field.

The technique of using just a single space to separate the various fields results in a CRT display with the four fields appearing slightly shifted on each line. This may be confusing when you want to analyze the source program on the CRT display. I get around this by using the tab function associated with the text editor to automatically insert the correct number of spaces so that the four fields appear in four vertically aligned columns. Figure 9 shows what the source program in Fig. 8 would look like on the CRT display after you enter it. You'll notice that Fig. 9 looks exactly like Fig. 8 but without the names of the four fields. I also left out the comment field in Fig. 9.

```
START    LXI      H, 000C
         LXI      D, 000D
         LDAX        D
         ADD         M
         STA       000E
         HLT
         END       START
```

FIG. 9—ASSEMBLY-LANGUAGE source program as it would appear on a CRT terminal.

Once the source program is entered on the terminal, you then transfer the source program to a cassette tape. This is done by using the various commands provided us by the text editor.

You're now ready to use the assembly-language program to translate the source program into machine language. For the sake of discussion, let's assume that you're using a two-pass assembler. It gets its name because the source program is loaded into the computer twice. The first thing you do is load the assembler program into the computer using the cassette tape recorder. Then, the source program is loaded into the computer, again using the cassette recorder. Next, rewind the cassette tape containing the source program and load it into the computer a *second* time. The CRT display will then tell you if there are any mistakes in the source program as the computer compares the two programs. If there are no mistakes, you can then store the machine-language program on another cassette tape.

If there was a mistake, the source program will have to be corrected. To make a correction, first load the text editor back into the computer. Then load the source program. The corrections are easily made using the various edit commands available with the text editor. The corrected source program is then stored on the cassette tape and the whole assembly procedure is repeated.

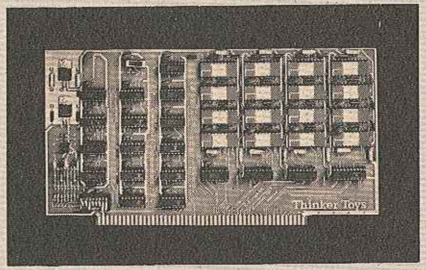
Assembler programs are different

Not all assembler programs are the same. Before you buy an assembler program, make sure that it is intended to be used with the micro-

UC-2000 from Infinite Systems contains computer, CRT terminal, and floppy disc drives.

processor you have in your computer. Since assembler programs generate machine language, the machine language must be compatible with the particular microprocessor in your computer. For example, if you have an 8080 microprocessor in your computer, you must use an 8080 assembler program.

There are two types of assembler programs available; the most common type used in personal computers is the *two-pass assembler.* The other type of assembler program is called a *one-pass assembler.* Without going into the details of how an assembler program works, let's take a look at what the differences between these two types mean to you as a programmer. First, the two-pass assembler requires less memory to assemble the exact same source program than a one-pass assembler requires. So, with the same amount of memory in your computer, you could assemble larger programs using a two-pass assembler than with a one-pass assembler. However, using a two-pass assembler requires more time since the source program must be loaded into the computer twice. As fast as cassette tape recorders seem for loading programs, they're still relatively slow. So, the trade-offs for the two types of assemblers are speed and memory requirements. Since memory is still quite expensive, the two-pass assembler is most commonly used in personal computers. The total memory requirements for using either type of assembler depends on the length of your source programs.



8K RAM memory board from Thinker Toys mates directly with the S100 bus.

Higher-level languages

So far, this article has been discussing machine language and assembly language. When you program in machine language, you communicate with the computer on the same level as the microprocessor. When you program in assembly language, you communicate with the computer by using mnemonics, and the assembler program translates the mnemonics into machine language. Is assembly language considered a higher-level language? The answer is no. The mnemonics are just another form of microprocessor instructions. One mnemonic is translated into one microprocessor instruction by the assembler program. A higher-level language translates one instruction into many microprocessor instructions.

The most common higher-level language for personal computers is BASIC. This language uses English-like statements and commands. For example, let's look at your addition program again, but this time written in BASIC. The program is written so it will add the numbers 2 and 3 together:

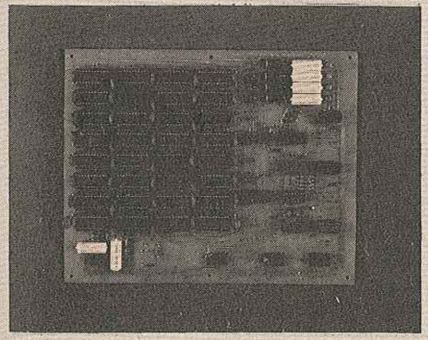
```
10 A = 2 + 3
20 PRINT A
30 END
```

The numbers preceding the statements are called *line numbers.* These tell the computer which statements to execute first. The computer merely executes the statements in numerically ascending order. The line numbers are also used for reference by a few other commands in the BASIC repertoire. The first statement sets the value of the variable A equal to the sum of 2 and 3. The PRINT A statement outputs the result to either the CRT display or a hard-copy printer if you have one. The last statement tells the computer that the program is finished and to stop execution. As you can see, writing programs on this level frees you from worrying about such things as memory locations, microprocessor registers, etc. You simply tell the computer what to do and the BASIC takes care of the rest.

MICROCOMPUTER CONTROL of AC appliances is possible with AC controller boards.

To program in BASIC, you again need a CRT terminal and a cassette tape recorder. First, you load the BASIC software into the computer using the cassette tape recorder. Then, you write the BASIC program on the alphanumeric keyboard. The program is written one line at a time. When you're finished, the CRT display will look exactly like the simple addition example. To execute the program, simply type RUN and hit the return key on the keyboard.

There are many, many variations of BASIC available. First, there's BASIC, and then there's EXTENDED BASIC. EXTENDED BASIC adds more features and commands to the BASIC repertoire. Also, there are variations between



WWW ENTERPRISES memory board uses Texas Instruments TMS-4060 memory IC's.

BASIC's written by different software suppliers. A program written using one kind of BASIC may not run under another BASIC without some minor modifications. The use of semicolons, colons, commas, quotation marks, etc., in the program is called *syntax.* Some BASIC's are very flexible in the use of syntax while others are quite rigid . . . a point you should investigate before purchasing a BASIC.

Other higher-level languages are also available, including APL, EMPL, FORTRAN and others. However, these languages are not in common use in personal computers for various reasons. For example, APL requires the use of additional symbols not found on a standard alphanumeric keyboard. So, to use APL requires a special APL keyboard.

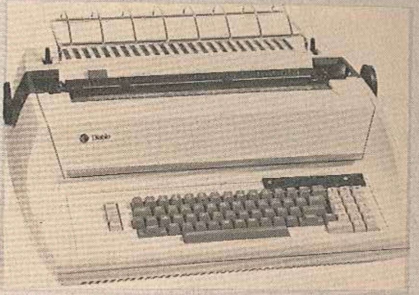
Higher-level languages are also classified as either *compilers* or *interpreters.* A compiler is very similar to an assembler program. That is, a compiler translates a source program into machine language, except that the source program is written using the statements associated with the higher-level language instead of mnemonics. The higher-level compiler resides in a high-speed external memory, such as a floppy disc. The source program is written and the compiler

HEATH H8 computer has an octal display and keyboard on the front panel.

translates it into machine language. Then the machine-language program is loaded into the computer and executed.

An interpreter, on the other hand, resides within the computer's internal memory. The source program is written using the higher-level statements, just as you did with the simple addition program. The interpreter translates the source progam *one statement at a time* into machine language, the computer executes the machine-language statements and then the interpreter translates the next statement in the source program.This process continues until the entire source program is translated *and* executed.

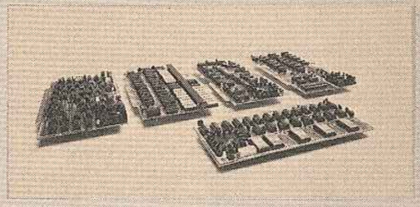
The disadvantage of using any higher-level language is the additional memory requirements. For example, even a BASIC interpreter is also classified by the additional internal memory it occupies. There are 4K, 8K and 12K BASIC's available. The



DIABLO PRINTER is offered by International Peripheral Systems for personal computer systems.

4K and 8K usually signify a standard BASIC, while 12K usually signifies an extended BASIC. If you have a computer with 16K of internal memory and you use a 12K BASIC, you only have 4K of memory left to write your programs in.

The disadvantage of using an interpreter is lack of speed, since each



COMPUTER BOARDS from Processor Technology mates with the S100 bus.

time a source program is executed, it must be translated. A compiler, on the other hand, translates the source program only once. Each time the program is executed, it is the machine-language program that is executed. A compiler, therefore, is much faster than an interpreter.

The disadvantage of using a compiler is the high-speed external memory requirements. This high-speed external memory is expensive. It is for this reason that interpreters are in far more common use than compilers. **R-E**

computer corner

How to prepare programs for microcomputers.

PETER RONY, CHRIS TITUS, DAVID LARSEN AND JONATHAN TITUS*

DEFINITIONS

Editor—Allows edit functions such as addition of a line or character to a program, insertion, deletion, etc. It permits you to alter your program. The input could be anything from programs or reports to raw instrument data.

Assembler—Converts the assembly language code into machine code, accepting mnemonics and symbolic addresses instead of actual binary values for addresses, instructions and data.

Monitor—Controls the operation of the various programs available. The monitor can access the editor, assembler or other programs.

Debugger—Allows a step-by-step observation of the program flow and the results of the program's operation. A debugger can be used to change data or instructions, alter registers, etc.

Breakpoint—A special instruction that can be inserted in a program to break off the normal program control and return control to a debug-type program. When a breakpoint is executed, the debug program indicates what the computer was doing at that point.

Cross Assembler—An assembler program that generates the program binary code for a computer other than the type it is being used with. For example, an 8080 cross-assember could operate on a PDP-8 minicomputer.

ONE OF THE PROBLEMS MANY MICROCOMPUTER users face is preparing software for their particular applications. The software examples that we have provided in past columns are short enough to be put together or *assembled* by hand; that is, we translated each mnemonic into its octal, hexadecimal or binary equivalent. Addresses for jumps, calls and input/output devices are easily added or changed since the computer programs are short and the addresses are probably listed in sequential order on the rough draft. Unfortunately, not all software preparation is this easy. Many application programs can be many thousands of steps long. This column will discuss the aids that are available for microcomputer program development.

One difficulty in software development is a clear, concise statement of the problem and how it is to be solved. All the desired results, inputs, outputs and the complete program flow, including all decision-making, must be considered before starting to program. An outline or block diagram can be used, but a flow chart is much easier to follow. Figure 1 shows a typical flow chart.

After the problem has been well thought out and a solution put in flow-chart form, a decision must be made. Is the program short enough to be easily translated by hand? In many cases, particularly with simple programs, hand assembly makes sense. In other cases, software development aids called *editors* and *assemblers* are faster and more efficient. To understand how editors and assemblers work, consider the process we used to put together this column.

We first outline the subject so that it can be covered well in short-column format. A handwritten copy is then typed, corrected, retyped and perhaps corrected and typed a final time. The illustrations and examples are formulated and drawn separately. This is the *editing* process. When writing a column, avoid references such as, ". . . the example below" or "the table on the following page." When the column is composed or *assembled,* references to tables, figures, etc., are much easier to follow.

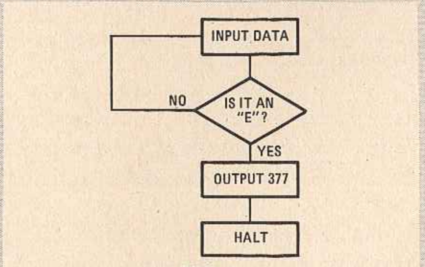


FIG. 1

Computer software is similarly developed. An editor program is used, either on a microcomputer or a timesharing system, to edit the individual program steps. The editor can correct, change, insert and delete steps in the program, just as with a manuscript. The editor program is generally unaware that you are writing a computer program, since you can use most editors to write a letter, prepare mailing lists, etc. When an editor is used to prepare a mnemonic program, *symbolic addresses* are often assigned to software tasks within the program. In this way, the actual value of the addresses for subprograms or subroutines is not needed. Just as in a column we refer to Fig. 4, the program refers to the letters, LOOP, as the starting address of a time-delay loop. Allowing the use of symbolic addresses for program steps means that the program can be changed without regard to the actual numeric values of addresses.

The assembler program must accept information from the editor and generate an output in a form that is compatible with your computer. Just as you assemble short programs one step at a time, so does the assembler. The assembler contains a table of mnemonics and their equivalent values. For example, an 8080 assembler would translate an MVIA instruction into 076 octal. The assembler also assigns real, 16-bit addresses to your symbolic addresses, such as LOOP. When using symbolic addresses, be sure to have a program step for each symbolic address and assign an address if you use a symbol. You cannot assign the same "name" to more than one address. Most assemblers will recognize a *redefined symbol* or an *undefined symbol,* and will produce an error message to let you know what needs to be corrected.

The final assembler output will be in punched paper tape, cassette or disc form ready to run on your system. Most assemblers will also produce a program listing showing the address of each step, the date in each successive location, a symbolic address name and the mnemonic plus any comments.

After a program has been assembled it will probably have to be debugged. The program checkout and debugging can be difficult without additional software "tools." Computer control panels are useful, but reading binary codes can become tedious, and there are many computers without external controls and readouts. There are *debugging programs* for most microcomputers that allow you to change instructions, list blocks of data or instructions and single-step through a program.

Many debug programs can establish a *breakpoint* in the software being tested. When the computer reaches a breakpoint, the instruction at that address is executed and an output device, such as a teletypewriter, lists the contents of important internal CPU registers. Breakpoints indicate not only that the computer reached a certain point in the software, but also what the computer did when it got there. If a breakpoint is set in the normal program flow and it is not reached, there is something wrong with the program. In this case, the breakpoint is moved closer and closer to the start of the program until the error is found. When the error is found, it can be corrected by using the debug program to change an instruction, data, etc.

Once the program operates correctly, the debug program should be able to save it on paper tape, a cassette, or some other medium. It should also be able to read such programs back into memory. In any case, when errors are found you should re-edit and reassemble the software to produce a complete, error-free documented listing. **R-E**

* This article is reprinted courtesy of American Laboratories. Dr. Rony, Department of Chemical Engineering, and Mr. Larsen, Department of Chemistry, are with the Virginia Polytechnic Institute & State University. Mr. Jon Titus is president of Tychon, Inc., a microcomputer consulting firm in Blacksburg, VA.

Integer multiplication and division routines for the 8080 microprocessor.

SINCE MICROPROCESSORS LIKE THE 8080 and 6800 do not have multiplication and division instructions, subroutines (containing addition and subtraction instructions) must be written to perform these operations. A typical paper-and-pencil decimal and binary multiplication for two different number sets is shown in Fig. 1.

Multiplication

Figure 1 shows that the mechanics of multiplication in the two number sets are

```
    1024            00100110
  × 596           × 00011011
  ------          ----------
    6144            00100110
   92160           001001100
+ 512000          0000000000
  ------          00100110000
  610304         001001100000
                0000000000000
               00000000000000
            + 000000000000000
            -----------------
              000010000000010
```

FIG. 1—MULTIPLICATION EXAMPLES shown in decimal and binary.

very similar. As the multiplicand is multiplied by larger and larger powers of 10 or powers of 2, the result of the multiplication must be shifted to the left by one, to increase the significance of the result. For instance, when 1024 is multiplied by the 9 in 596, the result (9216) is shifted to the left by one place, because the multiplication is really 90 × 1024 and the result is really 92,160.

To multiply two binary numbers, the 8080 must examine the multiplier one bit at a time. If the bit is a logic 1, the multiplicand is added to the *partial sum* (initially 0). If the bit examined is a logic 0, then the multiplicand is not added to the partial sum. Regardless of whether or not that addition takes place, the partial sum *must* be shifted one bit position, after each bit in the multiplier is examined.

To keep the 8080 multiplication software example (see Table 1) as simple as possible, we write a subroutine that multiplies two 8-bit numbers. These two numbers must be stored in registers D and E of the 8080, and the 16-bit result is stored in registers B and C (register pair B). When the subroutine is called, register pair B is cleared because it will be

TABLE 1—EIGHT-BIT MULTIPLICATION SUBROUTINE

```
MP88,      LXIB      /SET THE REGISTER PAIR THAT WILL HOLD THE
           000       /RESULT OF THE MULTIPLICATION, TO
           000       /000 000 (HEXADECIMAL 0000)
           MVIL      /LOAD L WITH 8 (DECIMAL), THE BIT COUNT
           010       /OR THE NUMBER OF SHIFTS TO TAKE PLACE
NXTBIT,    MOVAD     /MOVE THE MULTIPLIER INTO A
           RAR       /SHIFT IT ONE BIT TO THE RIGHT
           MOVDA     /THE CARRY IS EITHER 1 OR 0, SAVE THE
                     MULTIPLIER
           JNC       /IF THE CARRY IS 0, JUST SHIFT THE
           NOADD     /RESULT. IF THE CARRY IS A 1, ADD THE
           0         /MULTIPLICAND TO THE RESULT, THEN SHIFT IT
           MOVAB     /GET THE MSBY OF THE RESULT
           ADDE      /ADD THE MULTIPLICAND
           MOVBA     /AND SAVE THE MSBY OF THE RESULT
NOADD,     MOVAB     /NOW SHIFT THE 16—BIT RESULT ONE
           RAR       /PLACE TO THE RIGHT.
           MOVBA     /SAVE THE NEW MSBY
           MOVAC     /NOW SHIFT THE LSBY TO THE RIGHT.
           RAR
           MOVCA
           DCRL      /HAVE ALL 8 BITS OF THE MULTIPLIER
           JNZ       /BEEN TESTED YET? NO, TEST ANOTHER BIT
           NXTBIT
           0
           RET       /YES, THE ANSWER IS IN REGISTER PAIR B
```

used to store the *partial sum* and finally the 16-bit result of the multiplication. Register L is loaded with the number of bits in the multiplier, octal 010, hexadecimal 08 or decimal 8. At NXTBIT, the multiplier that is contained in register D is moved to register A, shifted once to the right and saved back in register D. These instructions shift a single bit of the multiplier into the carry so that the state of the bit (logic 1 or logic 0) can be tested with software instructions.

If the state of the carry after the shift is a logic 0, this means that the multiplicand is not added to the partial sum, so the JMP to NOADD (NO ADDition) is executed. If the carry is a logic 1, the JMP to NOADD is not executed. Instead, the multiplicand, contained in register E, is added to the partial sum, which is contained in register pair B.

At NOADD, the 16-bit number contained in register pair B is shifted to the right by one bit position. The multiplier's bit count, which is contained in register L, is then decremented by one. When this bit count is decremented to 0, the 8080 will return from the subroutine, with the 16-bit result of the multiplication in register pair B. If the bit count is nonzero, the JMP to NXTBIT is executed, so that another bit in the multiplier can be tested and any additions performed.

Subtraction

The multiplication of the two 8-bit binary numbers was performed by an *add and shift algorithm.* Binary division can be performed by a *subtract and shift algorithm.* An example of binary division is shown in Fig. 2. Binary division is more

```
            00001110
  1011   |10011011
        -  1011
           10000
        -   1011
             1011
        -    1011
            00001
```

FIG. 2—BINARY DIVISION example.

complex than binary multiplication. To divide two binary numbers, the divisor is subtracted from a larger and larger portion of the dividend that has less and less significance. If the divisor is larger than the part of the dividend from which it is being subtracted, a borrow occurs. In this case, the divisor is added to the *result of*

TABLE 2—EIGHT-BIT DIVISION SUBROUTINE

```
DIV88,      LXIH      /LOAD THE L REGISTER WITH 010 (DECIMAL 8)
            010       /OR HEXADECIMAL 08 AND LOAD THE H
            000       /REGISTER WITH 000 (THE RESULT WILL BE IN H)
            MVIC      /LOAD THE C REGISTER WITH 000
            000       /THIS REGISTER WILL BE USED FOR STORAGE
NXTBIT,     MOVAE     /MOVE THE DIVIDEND TO A
            RAL       /SHIFT THE MSB OF A INTO THE CARRY
            MOVEA     /SAVE THE SHIFTED DIVIDEND BACK IN E
            MOVAC     /GET THE PARTIAL DIVIDEND STORED IN C
            RAL       /SHIFT THE CARRY INTO THE LSB OF A.
            SUBD      /SUBTRACT THE DIVISOR FROM THIS NUMBER
            JNC       /IF THE CARRY=0, THE SUBTRACTION DID NOT
            NOADD     /PRODUCE A BORROW. THEREFORE, SHIFT THE
            0         /QUOTIENT. OTHERWISE ADD THE DIVISOR BACK TO
                      A
            ADDD      /ADD THE DIVISOR BACK TO THE CONTENT OF A.
NOADD,      MOVCA     /SAVE THE PARTIAL DIVIDEND BACK IN C.
            CMC       /COMPLEMENT THE CARRY.
            MOVAH     /AND SHIFT THE CARRY INTO THE LSB
            RAL       /OF THE H REGISTER. IF A BORROW, C=0
            MOVHA     /IF NOT, C=1
            DCRL      /HAVE ALL EIGHT BITS BEEN SHIFTED YET?
            JNZ       /NO, SHIFT ANOTHER BIT OF THE
            NXTBIT    /DIVIDEND AND TRY ANOTHER SUBTRACTION
            0
            RET       /THE ANSWER IS IN H WHEN THE 8080 RETURNS
```

the subtraction to re-generate the original part of the dividend being tested. A 0 is then entered in the quotient for the bit position being tested. If no borrow occurs when the subtraction is performed, the result of the subtraction is used as the new partial dividend, and a 1 is entered into the quotient since the divisor was successfully subtracted from the dividend. The subroutine listed in Table 2 divides the content of register E (the dividend) by the 8-bit content of the register D (the divisor) and the 8-bit result (the quotient) is saved in register H and the remainder is saved in register C.

The LXIH instruction in the subroutine (Table 2) loads the number of bits in the divisor (octal 010, decimal 8) into register L and register H is loaded with 0. This is done because register H will be used to store the quotient. The MVIC instruction loads register C with 0. Register C will be used to store the *partial dividend*. At NXTBIT, the dividend is shifted one bit to the right. The most-significant-bit (MSB) is shifted into the carry, and the remaining bits of the dividend are saved back in register E. The partial dividend in register C is then moved to register A, and the bit from the dividend is shifted from the carry into the least-significant-bit (LSB) of register A. The SUBD instruction subtracts the divisor from the partial dividend, which was in register A. If the divisor is subtracted from a larger or equal number, the JMP to NOADD is executed. If the divisor is greater than the partial dividend, a borrow occurs, therefore the divisor is added to the result of the subtraction by the ADDB instruction. Register A now contains the original partial dividend.

When the 8080 executes the instructions at NOADD, it must enter a logic 0 or logic 1 into the quotient. Therefore, the state of the carry is complemented by the CMC instruction and then saved in register H. If the subtraction did not generate a borrow, then the carry is a logic 0, but a logic 1 must be entered in the quotient. If a borrow was generated, the carry is a logic 1. This means that a logic 0 must be entered into the quotient. The CMC instruction simply complements the state of the carry to the state needed in the program. Finally, the content of register L is decremented by the DCRL instruction. If more bits within the dividend must be tested, the 8080 jumps back to NXTBIT, otherwise it returns from the subroutine with the quotient in register H and the remainder in register C.

There are a number of *software tricks* that can be used to simplify these two mathematical subroutines. However, unless your microcomputer can execute multiply-and-divide instructions or has special *multiply/divide hardware*, these operations will have to be performed using these or similar algorithms. **R-E**

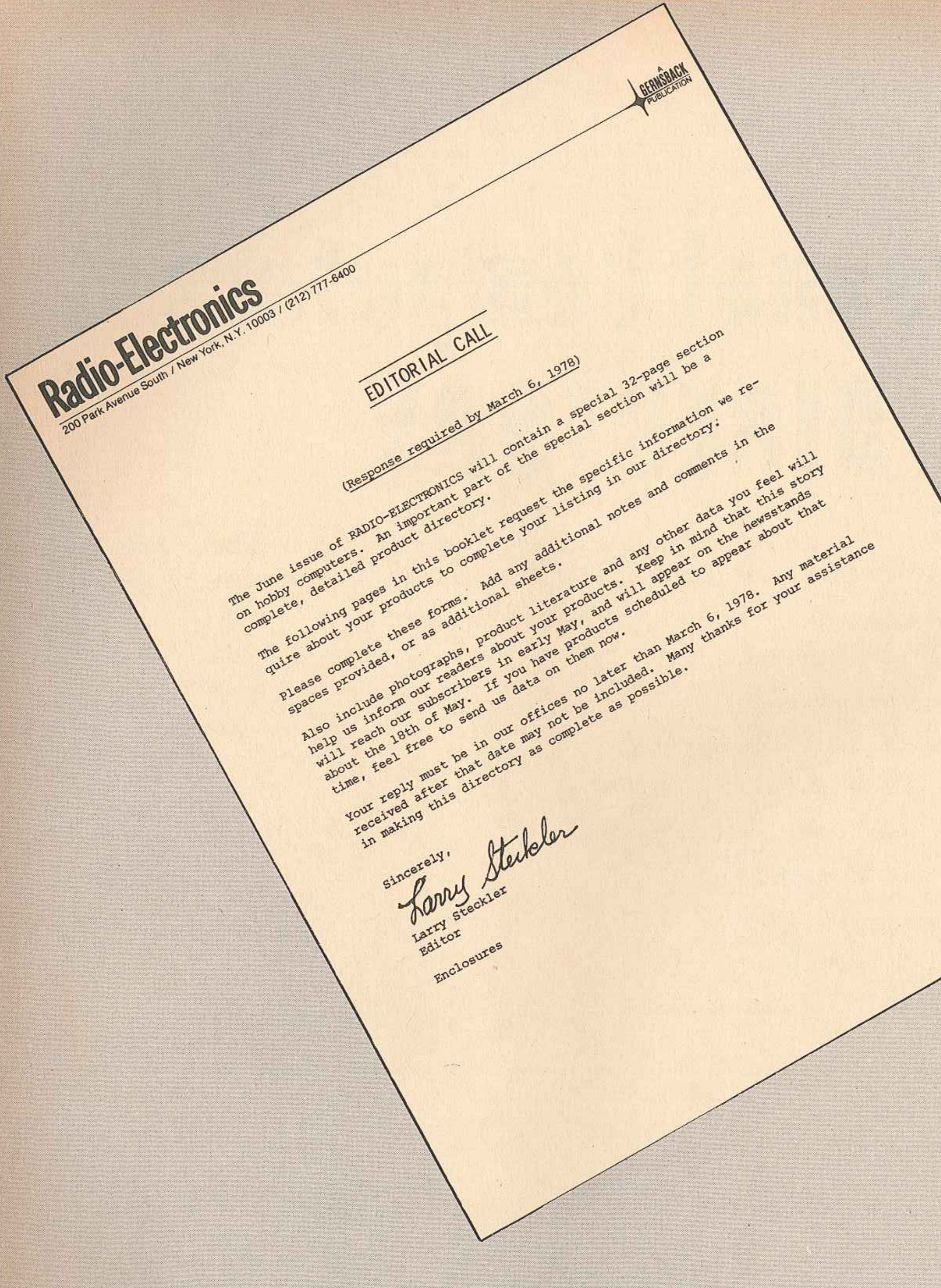
NOT IN THE CHARTS

The following companies provide products of interest to the computer hobbyist, but are not listed in the charts on the following pages.

NAME OF COMPANY	PRODUCT	ADDRESS
COMPUTER FAIRE	Proceedings & Silicon Gulch Gazette—	Box 1579, Palo Alto, CA 94302
COMPUTER PROFESSIONALS	Book Club	P.O.B. 582, Princeton Rd., Hightstown, NJ
CREATIVE COMPUTING	Books	P.O. Box 789-M. Morristown, NJ 07960
HAYDEN PUBLISHING	Books	50 Essex St., Rochelle Park, NJ 07662
HOWARD H. SAMS & CO., INC.	Books	4300 W. 62nd St., Indianapolis, IN 46206
LEXINGTON BOOKS	Books	125 Spring St., Lexington, MA 02173
LOGICAL SERVICES INC.	Microcomputer Programming Course	1080-H E. Duane Ave., Sunnyvale, CA
MCGRAW-HILL PUBLISHING	Books	1221 Ave. of the Americas, New York, NY
O A E	PROM Programmer	676 W. Wilson Ave., Glendale, CA 91203
OSBORNE & ASSOC.	Books	P.O. Box 2036, Berkeley, CA 94702
PRENTICE-HALL, INC.	Books	Route 9W, Englewood Cliffs, NJ 07662
POWER-ONE, INC.	Open frame DC power supplies single dual & triple output models	Power-One Drive, Amarillo, CA 93010
RONDURE COMPANY	IBM Selectric Terminals	2522 Butler St., Dallas, TX 75]35
SHUGART ASSOC.	Floppy & Minifloppy™ Disc Drives	435 Oakmead Pkwy., Sunnyvale, CA 94086
SYLVANHILLS LAB INC.	X-Y Plotter	P.O. Box, Pittsburg, KS 66762
SYSTEX ENTERPRISES INC.	Computer Portrait System	P.O. Box 402, King of Prussia, PA 19406
TAB BOOKS	Books	Monterey & Pinola Aves., Blue Ridge Summit, PA 17214

Personal Computers DIRECTORY

The following pages contain detailed information on the personal computer products made by a large number of manufacturers. We think you'll find it informative



Radio-Electronics

200 Park Avenue South / New York, N.Y. 10003 / (212) 777-6400

A GERNSBACK PUBLICATION

EDITORIAL CALL

(Response required by March 6, 1978)

The June issue of RADIO-ELECTRONICS will contain a special 32-page section on hobby computers. An important part of the special section will be a complete, detailed product directory.

The following pages in this booklet request the specific information we require about your products to complete your listing in our directory.

Please complete these forms. Add any additional notes and comments in the spaces provided, or as additional sheets.

Also include photographs, product literature and any other data you feel will help us inform our readers about your products. Keep in mind that this story will reach our subscribers in early May, and will appear on the newsstands about the 18th of May. If you have products scheduled to appear about that time, feel free to send us data on them now.

Your reply must be in our offices no later than March 6, 1978. Any material received after that date may not be included. Many thanks for your assistance in making this directory as complete as possible.

Sincerely,

Larry Steckler

Larry Steckler
Editor

Enclosures

The letter reproduced at the top of this page was sent as a part of an 8-page questionnaire to all of the personal computer product manufacturers we could locate. We followed up by handing out additional forms at personal computer shows. The resulting tables, on the pages that follow, were compiled from the data we received. For further information on any of the items described, please contact the individual manufacturer. You'll find the name, address and phone number of everyone who participated, on the last page of this special section. When you write or phone for more information please inform the company that you read about their products in this special section. By doing so, you'll help us guarantee a more complete response from all personal computer manufacturers the next time we do this kind of report.

MAIN FRAMES

MANUFACTURER	MODEL	BUS SYSTEM	FRONT PANEL			SLOTS FOR PLUGINS	COMMENTS
			SWITCHES	KEYBOARD	DISPLAY		
APPLE	A250016X	APPLE 50		TYPEWRITER		8	FAN
COMPAL	80	S-100		ARS-33, 37		5	
COMPUCOLOR	II	40-PIN		ASC11	CRT	0	CONVECTION COOLED
COMPUTER SHOP	CS-100	S-100		ASC11 HEX	HEX		FAN BATTERY BACK UP
E & L	MMD-1	E & L		OCTAL HEX	LED		EXPANSION THROUGH SELF-POWERED ADD ONS
ELECTRONIC CONTROL TECHNOLOGY	TT-10	S-100	YES			10	WHISPER FAN
HEATH COMPANY	H-8 H-11	BH-50 38-PIN	YES YES	OCTAL		10 6	CONDUCTION COOLED FAN
INFINITE INC.	UC2000	S-100		ASC11	CRT	8	AXIAL FAN
ITHICA AUDIO		S-100	YES	BINARY			SMART FRONT-PANEL BOARD
JADE COMPUTER PRODUCTS	S-100, SS-50 KIM-1						
MIDWEST SCIENTIFIC INSTRUMENTS	MSI-6800	SS-50	YES	YES	CRT	16	FAN
NBL		SPECIAL		HEX	BINARY		
NETRONICS R & D	ELF II	EEF II	YES	HEX	7-SEG HEX	5	ON BOARD FILTER & REGULATOR
NORTH STAR	HORIZON	S-100	YES			12	FAN
NOVAL INC.	760	44-PIN	YES	ASC11	VIDEO	7	
OHIO SCIENTIFIC	C2-4P C2-S1S C3-S1	48-PIN 48-PIN 48-PIN	YES YES YES	ASC11	VIDEO	4 8 8	INCLUDES AUDIO CASSETTE INTERFACE, 4K RAM, 8K BASIC IN ROM, FAN
PARASITIC ENGINEERING	EQUINOX-100	S-100	YES	12-KEY OCTAL	OCTAL LED	20	CONSTANT VOLTAGE FERRO-RESONANT TRANSFORMER INTELLIGENT FRONT PANEL BOX FAN
QUAY	8000	S-100	YES			12	MUFFIN FAN
REALISTIC CONTROLS	Z1110	S-100		FULL A/N		6	FAN
SEALS ELECTRONICS	PUP-1	S-100	YES		INDICATORS	11	FAN
SOUTHWEST TECHNICAL PRODUCTS	68/2	SS-50	YES			15	7 MAIN PLUG-IN SLOTS 8 I/O PLUG-IN SLOTS
TEI INC.	MCS-112, 122, RM12, RM22	S-100	YES			12 OR 22	FAN
VECTOR GRAPHIC	VECTOR 1, 1+, 1++	S-100	YES			18	FULLY SHIELDED AND TERMINATED MOTHERBOARD WHISTLER FAN
WAVE MATE	JUPITER II, III	72-PIN	YES		4 DIGIT HEX	11	CONVECTION COOLED

	CPU								MEMORY								
			FEATURES														
MANUFACTURER	BUS SYSTEM	MICRO-PROCESSOR	MON. PROM	PARALLEL I/O	SERIAL I/O	OTHER	RAM (K)	ROM (K)	BUS SYSTEM	RAM (K)	ROM (K)	EPROM (K)	SPEED μs	DIRECT MEMORY ACCESS	ON BOARD REGULATORS	ADDRESS SELECTION	COMMENTS
APPLE	APPLE 50	6502	●	●	●		48	12	APPLE 50	48	12	12	500	YES	NO	PLUG IN JUMPERS	
CENTRAL DATA									S-100	16 OR 32			450	NO	YES	16K BLOCKS WITH 2K DG SELECT	
COMPAL	S-100	8080	●		●	VECTORED INTERRUPT	512 BYTES		S-100	UP TO 64	UP TO 3	UP TO 3	250 400	YES	YES		
COMPUCOLOR	40-PIN	8080	●	●	●	TIMERS	4, 32	16, 24	40-PIN	16	16	16	450	NO	YES	NO	MEMORY IS ON CPU CARD
COMPUTER SHOP	S-100	Z80	●						S-100	16 TO 64			200	YES	YES	4K	
THE COMPUTERIST									KIM-1	8		8	450	YES	YES		INDEPENDENTLY JUMPERABLE AT ANY 2K BOUNDRY
E & L INSTRUMENTS	E & L	8080	●	●			512 BYTES	512 BYTES	E & L	2		1	450	NO	NO		MOUNTS ON CPU POWERED FROM CPU
ELECTRONIC CONTROL TECHNOLOGY	S-100	8080, Z-80							S-100	2, 16	2	2	450 200	YES	YES	4K BLOCKS	
ELECTRONIC SYSTEMS									S-100	8			450	YES	YES		
FRANKLIN ELECTRIC									S-100	8			450, 250	YES	YES		
GODBOUT									S-100 BH-50	8-16, 12							4K BLOCKS
HEATH COMPANY	BH-50 LSI-11	8080 LSI-11	●				4	1 X 8	BH-50 LSI-11	8 X 8 4 X 16			450 500	YES	YES		
IMSAI MFG.	S-100	8080, 8085	●	●	●	PROGRAMMABLE RAM MODULE	1 TO 4	2	S-100	4, 16, 32, 64	4 16	4 16	475	YES	YES		
INFINITE INCORP.	S-100, CS-100 INFINITE 72, COSMAC	8080, 8085, Z-80, 1802	●	●	●		.5	3									
ITHICA AUDIO		Z-80	●						S-100	8 STATIC 16, 32, 64 DYNAMIC	16/32	16/32	250 OR 450	YES	YES	DIP SWITCH EPROM 1K INCREMENTS	
JADE COMPUTER PRODUCTS	S-100 KIM-1	8080, 6502, Z-80							S-100, SS-50 KIM-1								
MARINCHIP SYSTEMS	S-100	9900															
MIDWEST SCIENTIFIC INSTRUMENTS	SS-50	6800	●				128	4	SS-50	56	56	56	250	YES	YES	SWITCH	
MOUNTAIN HARDWARE									S-100	.5		7.5	500	NO	YES	EVERY 8K	
NBL		SC/MP II		●	●		1			1			500	YES	YES		
NETRONICS	ELF-II	1802					2.56		ELF-II	4			100	YES		4K BLOCKS TO 64K	FULLY BUFFERED & TRISTATE
NORTH STAR	S-100	8080, Z-80	OPT						S-100	16			200	YES	YES	2 8K REGIONS	
NOVAL INC.	44-PIN	Z-80	●	●		32K PROGRAM MEMORY			S-100	32				NO	YES	EACH 8K OF MEMORY CAN BE JUMPERED ON 8K BOUNDRIES	
OHIO SCIENTIFIC	48-PIN	8080, 6502 Z-80	●	●	●		4 16	8	48-PIN	4 16	8 ON CPU		350 250		NO		PIA BASED PARALLED I/O OPTIONAL
PARASITIC ENGINEERING	S-100	8080A	●				512 BYTES	512 BYTES	S-100	4 STATIC			450		YES	DIP SELECTABLE TO ANY 4K BLOCK	

	CPU								MEMORY								
			FEATURES														
MANUFACTURER	BUS SYSTEM	MICRO-PROCESSOR			SERIAL I/O	OTHER	RAM (K)	ROM (K)	BUS SYSTEM	RAM (K)	ROM (K)	EPROM (K)	SPEED μs	DIRECT MEMORY ACCESS	ON BOARD REGULATOR	ADDRESS SELECTION	COMMENTS
QUAY	S-100	Z-80	●	●	●		1		S-100	8 STAT IC		ON CPU	450	YES	YES	4K BLOCKS	
REALISTIC CONTROLS	S-100		●		●		64	2	S-100	64	2	2	250	YES	YES	0000_H-$FFFF_H$	
SD COMPUTER PROD	S-100	Z-80	●	●	●		1	4 8	S-100	4, 8 STATIC 8 TO 64		16 & 32	250/500 STATICRAM 375 EXPANDORAM 450 EPROM	YES	YES	AT ANY LOCATION	
SEALS									S-100 SS-50	8, 16 32, 8	4		250 & 500				
SMOKE SIGNAL BROADCASTING									SS-50	16		8	250 & 400	YES	YES	SWITCH SELECT	
SOLID STATE SALES									F8, KIM, 6800	4			450	YES	NO		
SOUTHWEST TECHNICAL PRODUCTS	SS-50	6800	●				128 BYTES	8 BYTES	SS-50	4, 8 16, 32		UP TO 8 BYTES		YES	YES	ANY 8K BLOCK	
SZERLIP ENTERPRISES									S-100	1		UP TO 32	450	YES	YES		
TEI INC	S-100	8080, Z-80							S-100	16			250		YES		
THINKER TOYS	S-100	8080	●	●	●		256 BYTES	1 BYTES	S-100	8 65			250 250	NO		4K BLOCKS 8K WINDOWS	
VECTOR GRAPHICS	S-100	8080, Z-80							S-100	8, 16		2, 12		YES	YES	SWITCH 8K BLOCKS	
WWW ENTERPRISES									NONE	16			270 DYNAMIC 250 STATIC	YES	NO		
WAVE MATE	72-PIN	Z-80, 6800							72-PIN	UP TO 32		350	350	YES	YES		

	I/O BOARDS																				
	I/O PARALLEL							I/O SERIAL								I/O CASSETTE					
				BAUD						BAUD											
MANUFACTURER	BUS SYSTEM	NO PORTS	HANDSHAKING	RATE	SOFTWARE SELECT	JUMPER SELECT	COMMENTS	BUS SYSTEM	NO PORTS	RATE	SOFTWARE SELECT	JUMPER SELECT	EIA COMPATIBLE	TTY COMPATIBLE	COMMENTS	BUS SYSTEM	BAUD RATE	RECORDING FORMAT	START/STOP CONTROL	COMMENTS
APPLE	APPLE 50	1	●					APPLE 50	1	9600 110	●		●				1500	APPLE ENCODED		
COMPAL		1							2	5600 50	●		●	●			2400 300	PHASE ENCODED	●	
COMPUCOLOR							ON CPU CARD								ON CPU CARD					
COMPUTER SHOP	S-100		●			●		S-100	2	9600 55		●	●	●						
E & L INSTRUMENTS	E & L	3		1750 0			PART OF CPU CARD	E & L	1	110 110					PART OF CPU CARD	E & L	300 300	2-TONE AUDIO VART FORMATTED		

I/OBOARDS

	I/O PARALLEL							I/O SERIAL								I/O CASSETTE				
				BAUD						BAUD										
MANUFACTURER	BUS SYSTEM	NO PORTS	HANDSHAKING	RATE	SOFTWARE SELECT	JUMPER SELECT	COMMENTS	BUS SYSTEM	NO PORTS	RATE	SOFTWARE SELECT	JUMPER SELECT	EIA COMPATIBLE	TTY COMPATIBLE	COMMENTS	BUS SYSTEM	BAUD RATE	RECORDING FORMAT	START/STOP CONTROL	COMMENTS
ELECTRONIC CONTROL TECHNOLOGY								S-100	2	19200 50		●	●	●						
ELECTRONIC SYSTEMS								APPLE II		30000 0						S-100	1200 0	VART (FSK)	●	2 DIFFERENT UNITS
FRANKLIN ELECTRONICS	S-100	1				●		S-100	3	9600		●	●	●						
GENERAL MICRO SYSTEMS												●	●			S-100		BIPHASE MANCHESTER	●	1600 BITS/INCH HI-DENSITY DIGITAL 2000 BYTES/SEC
HEATH COMPANY	BH-50 LSI11	3	●					BH-50 LSI-11		9600 110 9600 50		●								
IMSAI MFG	S-100	2 OR 6	●					S-100	2	56000 560		●				S-100	4800 800	TARBELL	●	
IOR	S-100 CUSTOM	6	●																	
JADE COMPUTER PRODUCTS	S-100							S-100								S-100				
MIDWEST SCIENTIFIC PRODUCTS	SS-50	8	●	19200 110		●		SS-50	3	19200 110		●	●	●		SS-50	330 110	KC	●	
NBL		1			●													KC		
NETRONICS	ELF-II	2	●		●			ELF-II				●	●	●		ELF-II	1000	SIMILAR TO KC	●	BOARD ALSO CONTAINS MONITOR PROM
NORTH STAR		1					ON MOTHER BOARD		1	9600 75		●	●	●	ON MOTHER BOARD					
NOVAL	44-PIN						8 8-BIT PORTS ON CPU CARD	44-PIN	1	9600 150		●	●	●			2500	DIGITAL PHASE ENCODED	●	
OHIO SCIENTIFIC	48-PIN	2		1 MHZ SYSTEM																
PARASITIC ENGINEERING	S-100	1	●					S-100	1	4800 110	●		●	●		S-100	300	KC	●	CONTROLS 3 RECORDERS
PERCOM								S-100	1	9600 110		●	●	●		S-100 SS-50	2400 300 1200 300	EXTENDED KC EXTENDED KC	●	
PERSONAL COMPUTING																SS-50	2400 110	KC @ 300 BAUD MANCHESTER @ 2400 BAUD	●	
QUAY	S-100	8	●																	
REALISTIC CONTROLS	S-100	2	●	200000			BUILT INTO FLOPPY CONTR	S-100		56000 110		●	●	●						
RONDURE COMPANY															FOR IBM SELECTRIC					
SOUTHWEST TECHNICAL	SS-50	1	●	9600 110		●		SS-50	1	9600 110		●	●	●			300	KC	●	CONTROLS 2 RECORDERS
TEI	S-100	3		19200		●		S-100	3	19200		●	●	●						
THINKER TOYS																S-100	300	KC	●	CONTROLS 3 CASSETTES INCLUDES 3 SERIAL & PARALLEL PORTS
WAVE MATE	72-PIN	2 OR 4	●					72-PIN	1 OR 2	19200 50	●	●	●	●		72-PIN	1200 300	KC	●	

	FLOPPY INTERFACE							OTHER BOARDS				
MANUFACTURER	BUS SYSTEM	MINI 5"	FLOPPY 8"	NO. DRIVES	HARD SECTORED		OPERATING SYSTEM	COMMENTS	BUS SYSTEM	SPEECH	MUSIC	CONTROLLERS	OTHERS
APPLE		●		2		●	YES			●	●	●	
CALIFORNIA INDUSTRIAL													MOTHER BOARD
CANDA SYSTEMS						●	YES		S-100			●	REAL TIME CLOCK AC POWER SWITCH
COMPAL	S-100	●		4	●		MICROPOLIS						4-SLOT EXPANDER
COMPUTALKER									S-100	●			AUDIO AMPLIFIER AND SPEAKER REQUIRED
COMPUCOLOR		●				●	IN ROM						
COMPUTER SHOP	S-100	●		1 TO 3	●		NORTH STAR						
E & L									E & L				EPROM & 8K RAM
ELECTRONIC CONTROL TECHNOLOGY													CARD CAGE, MOTHER BOARD, EXTENDER
ELECTRONIC SYSTEMS													VART & BAUD-RATE GENERATOR TV TYPERWRITER MODEM
HEATH COMPANY	BH-50	●		2	●								
IMSAI	S-100	●	●	7		●	ENHANSED CP/M						
INFINITE	S-100	●		3			NORTH STAR						
ITHICA AUDIO	S-100	●	●	8	●							●	A/D-D/A-8 BIT A/D-D/A-12 BIT
JADE COMPUTER PRODUCTS									S-100	●	●		
MICRONICS									S-100				BUG TRAP
MIDWEST SCIENTIFIC INSTRUMENTS	S-100 SS-50		●	4	●		YES		SS-50				CALENDAR CLOCK OPTO ISOLATOR
MOUNTAIN HARDWARE									S-100 APPLE-II			●	REMOTE CONTROL CLOCK CALENDAR
NETRONICS													PROTOTYPE
NORTH STAR	S-100	●		3			2.5K DOS						HARDWARE, FLOATING POINT
NOVAL	44-PIN	●		2		●	YES		44-PIN				PROM BURNER 2ND DISPLAY DRIVE COLOR SELECTOR
OBJECTIVE DESIGN									S-100				PROGRAMABLE CHARACTER GEN 512 X 256
OHIO SCIENTIFIC	48-PIN	●		1 2		●	YES		48-PIN				GRAPHICS 128 X 128 4 COLOR ALPHABETICS
PAIA											●		
PARASITIC ENGINEERING	S-100	●	●	4		●	CP/M						

	FLOPPY INTERFACE								OTHER BOARDS				
MANUFACTURER	BUS SYSTEM	MINI 5"	FLOPPY 8"	NO. DRIVES	HARD SECTORED	SOFT SECTORED	OPERATING SYSTEM	COMMENTS	BUS SYSTEM	SPEECH	MUSIC	CONTROLLERS	OTHERS
PERCOM	SS-50	●					IN IK ROM						
PERSONAL COMPUTING	SS-50	●	●	4		●	YES		SS-50				PROTOTYPE
PROCESSOR CONTROL SYSTEM													PROGRAM ORGAN
QUAY	S-100	●		4			QUAY						
REALISTIC CONTROLS	S-100	●	●	8	●		CP/M		S-100	●	●	●	
SMOKE SIGNAL BROADCASTING	SS-50	●				●	YES						
SOUTHWEST TECHNICAL	SS-50	●	●	4		●	YES		SS-50			●	INTERRUPT TIMER CALCULATOR INTERFACE
TEI	S-100	●	●	3 TO 8		●	CP/M						
THINKER TOYS	SS-50	●	●	3 TO 8			CP/M						WUNDERBUS MOTHER BOARD
VECTOR													CARDCAGE S-100 HARDWARE PROTOTYPE
VECTOR GRAPHICS	S-100		●	2		●	CP/M	DUAL STORE					
WAVE MATE	72-PIN		●	2 OR 4		●	YES						

VIDEO DISPLAY MODULES

					GRAPHICS RESOLUTION		TEXT CHARACTERISTICS															
MANUFACTURER	BUS SYSTEM	RF OUTPUT	VIDEO OUTPUT	GRAPHICS	VERT ELEM	HORIZ ENTS	DOT MATRIX	CHARACT. PER LINE	NO. LINES	PROGRAMMABLE CURSOR CONTROL	BLACK ONLY	WHITE ON BL	SCROLLING	HIGHLIGHT	UPPER CASE	LOWER CASE	COLOR	ON BOARD MEMORY (K)	SYSTEM MEMORY REQUIRED	SERIAL I/O KEYBOARD INPUT	PARALL I/O	COMMENTS
APPLE	APPLE		●		280	192	5 X 7	40	24	●	●	●	●	●	●			UP TO 46 RAM12ROM	8K		●	15 COLORS STANDARD 4-COLOR HI RESOLUTION
CENTRAL DATA	S-100		●		640	192	6 X 8	80	16	●		●	●		●	●		2	0			INCLUDES 2650, CASSETTE INTERFACE
COMPAL	S-100		●	●	128	48	7 X 9	64	16	●		●			●	●		1	0		●	
COMPUCOLOR				●	384	256	5 X 7	64	32	●			●	●	●	●	●	4	0		●	COLOR (8 BACKGROUND) (8 FOREGROUND)
COMPUTER SHOP	S-100		●		64	16	YES											16 TO 64	16		●	
ELECTRONIC SYSTEMS	STAND ALONE		●				5 X 7	32	16	●		●	●		●			1	0		●	
IMSAI MANUFACT.	S-100		●	●			5 X 7	80 40	24 12	●	●	●	●		●	●		2 RAM 1 PROM				
INFINITE INC.	S-100		●	●	128	48	7 X 9	64	16			●			●	●		1	2		●	

VIDEO DISPLAY MODULES

					GRAPHICS RESOLUTIONS		TEXT CHARACTERISTICS															
MANUFACTURER	BUS SYSTEM	RF OUTPUT	VIDEO OUTPUT	GRAPHICS	VERT ELEM	HORIZ ENTS	DOT MATRIX	CHARACT. PER LINE	NO. LINES	PROGRAMMABLE CURSOR CONTROL	BLACK ONLY	WHOTE ON BL	SCROLLING	HIGHLIGHT	UPPER CASE	LOWER CASE	COLOR	ON BOARD MEMORY (K)	SYSTEM MEMORY REQUIRED	SERIAL I/O KEYBOARD INPUT	PARALL I/O	COMMENTS
IOR	S-100 CUSTOM		●				7 X 9	64	30	●			●	●	●	●		2	0			
ITHICA AUDIO	S-100		●	●	512	512	5 X 7	64 80	16 24	●	●		●			●		1 2				2 BOARDS
JADE COMPUTER PRODUCTS	S-100		●		128	48	7 X 9	32 64	16	●	●	●	●		●	●		1				
NETRONICS	ELF-II		●	●	128	64																
NOVAL INC.	PART OF CPU		●		256	224	8 X 8	32	28	●	●	●	●	●	●	OPT- IONAL	●					VIDEO IS PART OF CPU
OHIO SCIENTIFIC	48-PIN			●			5 X 7	32 64	32			●	●		●	●		2	0			PROCESSOR CONTROLLED
PERCOM	SS-50		●	●	400	360	7 X 9	80	16 OR 24		●		●	●	●	●		2	0			MEMORY MAPPED
QUAY	S-100		●				5 X 7	64	16	●		●	●		●	●		11				
REALISTIC CONTROLS	S-100						7 X 9	80	24	●	●	●	●	●	●	●	●		2		●	
TEI	S-100		●				7 X 9	80	24	●	●	●	●	●	●	●		2	2	●		
VECTOR GRAPHICS	S-100		●		128	48	7 X 9	64	16	●	●	●	SOFT- WARE	●	●	●		1 X 12 BITS	2		●	
WAVE MATE	72-PIN		●	●	128	96	7 X 13	64 96	32 16	●	●	●	●		●	●		1 2			●	

VIDEO DISPLAY TERMINALS

	PARALLEL I/O				KEYBOARD					VIDEO DISPLAY			TEXT									
MANUFACTURER	NO. LINES	HANDSHAKING	COMPATIBILITY	SERIAL I/O COMPATIBILITY	NO. KEYS	NO. USER DEFINABLE KEYS	CURSOR	UPPER CASE	LOWER CASE	GRAPHICS	COLOR	RESOLUTION	CHAR. PER LINE	NO. LINES	DOT MATRIX	SCROLLING	BLACK ON WHITE	WHITE ON BLACK	HIGHLIGHTING	UPPER CASE	LOWER CASE	COMMENTS
COMPUCOLOR				RS232 EIA	117	16	●	●	●	●	●	384 256	64	32	5 X 7	●			●	●	●	
COMPUTER STORE				RS232M	54	3	●	●	●				64	16	●	●		●		●	●	
E & L INSTRUMENTS				RS232 & 20 MA LOOP	64	0	●	●					64	16	7 X 9	●	●	●		●		REPROGRAMMABLE CHARACTER GENERATOR PARALLEL I/O FOR ASC11 KEYBOARD
HEATH COMPANY			RS-232C 20 MA LOOP	YES	67	67	●	●			●		80 20	12 48	5 X 7	●		●		●		
OHIO SCIENTIFIC				YES	53		●	●	●				64	16		●		●		●		ONLY APPLY TO MICROTERM ACT-1
REALISTIC CONTROLS					70	70	●	●	●	●	●	600 LINES	80	24	●	●	●	●	●	●		
SOUTHWEST TECHNICAL				YES	56	2	●	●	●			600 LINES	64	16	●	●	●	●	●	●		FULL CURSOR DECODING
WAVE MATE	20			SYNC & VIDEO COAX	73	73	●	●	●	●		128 X 96	64 96	32 16	7 X 4	●	●	●		●	●	

LIST OF PERSONAL COMPUTER MANUFACTURERS

Here's a list of names addresses and phone numbers for every company that filled out one of our questionnaires. If a name you are looking for is missing, they did not respond to our request for information.

APPLE COMPUTER INC.
10260 Bandley Drive
Cupertino, CA 95014
(408) 996-1010

CALIFORNIA INDUSTRIAL
15214 Grevill Ave.
Lawndale, CA 90260
(213) 772-0800
T. RYDER

CANADA SYSTEMS, INC.
1353 Foothill Blvd.
La Cañada, CA 91011
(213) 790-7957
KEN FINSTER

CENTRAL DATA CO.
P.O. Box 2484, Station A
Champaign, IL 61820
(217) 359-8010
JEFF ROLOFF

COMPAL
(Computer Power & Light)
12321 Ventura Blvd.
Studio City, CA 91604
(213) 760-3345
IVA KALB

COMPUCOLOR CORP.
5965 Peachtree Corners East
Norcross, GA 30071
(404) 449-5961
RODNEY HUNT

COMPUTALKER CONSULTANTS
P.O. Box 1951
Santa Monica, CA 90406
(213) 392-5230

THE COMPUTER FAIRE
Box 1579
Palo Alto, CA 94302
(415) 851-7075
JIM WARREN

COMPUTER SHOP
288 Norfolk St.
Cambridge, MA
(617) 473-2323
R. RIVERA

THE COMPUTERIST
P.O. Box 3
S. Chelmsford, MA 01824
(617) 256-3649
ROBERT M. TRIPP

E & L INSTRUMENTS
61 First Street
Derby, CT 06418
(203) 735-8774
RICHARD J. VUILLEQUEZ

ELECTRONIC CONTROL TECHNOLOGY
763 Ramsey Avenue
Hillside, NJ 07205
(P.O.B. 6, Union, NJ 07083)
(201) 686-8080
DENNIS P. DUPRÉ

ELECTRONIC SYSTEMS
P.O. Box 212
Burlingame, CA 94010
(408) 374-5984

FRANKLIN ELECTRIC
733 Lakefield Rd.
Westlake Village, CA 91361
(805) 497-7755
FRANK PETERS

GENERAL MICRO-SYSTEMS
12369 W. Alabama Place
Lakewood, CO 80228
(303) 985-3423
BOB SMITH

BILL GODBOUT ELECTRONICS
Box 2355
Oakland Airport, CA 94614

THE HEATH COMPANY
Benton Harbor, MI 49022
(616) 982-3417
V. VIRGIL BENNETT

IMSAI MFG. CORP.
14860 Wicks Blvd.
San Leandro, CA
(415) 483-2093

INFINITE INC.
1924 Waverly Place
Melbourne, FL 32901
(305) 724-1588
BILL HABERHERN

I O R
P.O. Box 28823
Dallas, TX 75228
(214) 358-2671

ITHACA AUDIO
P.O. Box 91
Ithaca, NY 14850
(607) 273-3271

JADE COMPUTER PRODUCTS
5351 West 144th
Lawndale, CA 90260
(213) 679-3313

LEXINGTON BOOKS
D.C. Heath & Co.
125 Spring Street
Lexington, MA 02173
(617) 862-6650

LOGICAL SERVICES INC.
1080-H East Duane Ave.
Sunnyvale, CA 94086
(408) 245-8855

MARINCHIP SYSTEMS
16 St. Jude Road
Mill Valley, CA 94941
(415) 383-1545
JOHN WALKER

MICRONICS, INC.
P.O. Box 12545
Raleigh, NC 27605
LENNY HEATH

MIDWEST SCIENTIFIC INSTRUMENTS, INC.
220 W. Cedar St., Olathe, KS 66061
(913) 764-3273
DR. CHILDRESS

MOUNTAIN HARDWARE, INC.
P.O. Box 1133
Ben Lomond, CA 95005
(408) 336-2495
D. PACE

NBL
Box 1564
Richardson, TX 75080
(214) 231-2703
W.A. KLUCK

NETRONICS R&D LTD.
333 Litchfield Road
New Milford, CT 06776
(203) 354-9375

NORTH STAR COMPUTERS, INC.
2547 Ninth Street
Berkeley, CA 94710
(415) 549-0858
CA GRANT

NOVAL, INC.
8401 Aero Drive
San Diego, CA 92123
(714) 277-8700
AGO KISS

OAE (Oliver Advanced Engineering)
676 West Wilson Avenue
Glendale, CA 91203
(213) 240-0080
DOUGLAS E. OLIVER

OHIO SCIENTIFIC
Box 36
Hiram, OH 44234
(216) 562-3101
DON MUCHOW

OSBORNE & ASSOC.
P.O. Box 2036
Berkeley, CA 94702
(415) 548-2805

PARASITIC ENGINEERING
P.O. Box 6314
Albany, CA 94706
(415) 547-6612

PCS (Processor Control Systems)
Box 544
Celoron, NY 14720
(716) 664-2871
***WALTER E. PELTON,* Gen'l Mgr.**

PERCOM DATA CO., INC.
318 Barnes
Garland, TX 75042
(214) 276-1968
LUCY MAUCH

PERSONAL COMPUTING CO.
3321 Towerwood Drive
Dallas, TX 75234
(214) 620-2776
STEVIE GENTRY

POWER-ONE, INC.
Power-One Drive
Amarillo, CA 93010
(805) 484-2806
STEVE COLE

QUAY CORP.
P.O. Box 386
Freehold, NJ 07728
(201) 681-8700
Mr. ROESSLER

REALISTIC CONTROLS CORP.
404 West 35th Street
Davinport, IA 52806
(319) 386-4400

RONDURE COMPANY
2522 Butler Street
Dallas, TX 75235
(214) 630-4621

SD COMPUTER PRODUCTS
Div. SD Sales
P.O. Box 28810
Dallas, TX 75228
(214) 271-4667
800 527-3460

SEALS ELECTRONICS, INC.
10728 Dutchtown Rd.
Concord, TN 37922
(615) 966-8771
***ROBIN CONTENT,* Mktg. Dir**

SHUGART ASSOC.
435 Oakmead Pkwy.
Sunnyvale, CA 94086
(408) 733-0100
GARY YOST

SMOKE SIGNAL BROADCASTING
6304 Yucca
Hollywood, CA 90028
(213) 462-5652
RIC HAMMOND

SOLID STATE SALES
P.O. Box 74
Somerville, MA 02143
(617) 547-4005
J. HEFFRON

SOUTHWEST TECHNICAL PRODUCTS CORP.
219 W. Rhapsody
San Antonio, TX 78216
(512) 344-9778
DAN MEYER

SYLVANHILLS LAB INC.
P.O. Box 646
Pittsburg, KS 66762
(316) 231-4440
SHARON BELL

SYSTEX ENTERPRISES INC.
P.O. Box 402
King of Prussia, PA 19406
(215) 482-9481
(215) 631-1318

SZERLIP ENTERPRISES
1414 W. 259th Street
Harbor City, CA 90710

TEI Inc.
c/o CMC MARKETING CORP.
5601 Bintliff #515
Houston, TX 77036
(713) 783-8880

THINKER TOYS
1201 10th Street
Berkeley, CA 94710
(415) 527-7548
JEAN MORROW

VECTOR ELECTRONICS CO., INC.
12460 Gladstone Avenue
Sylmar, CA 91342
(213) 365-9661
FLOYD HILL

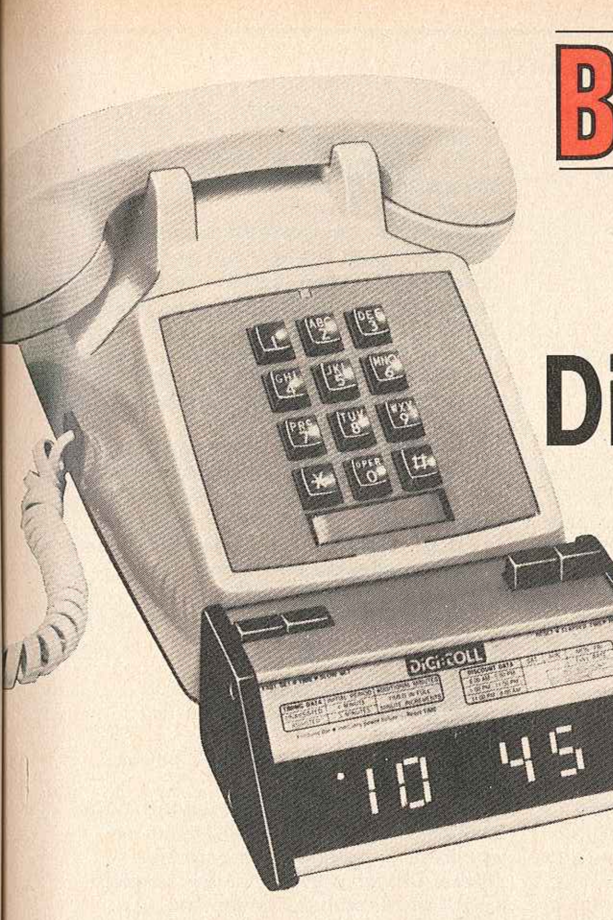
VECTOR GRAPHIC INC.
790 Hampshire Road A&B
Westlake Village, CA 91361
(805) 497-6853

WWW ENTERPRISES
P.O. Box 548
Harbor City, CA 90710
(213) 835-9417
WARREN WEIMER

WAVE MATE
1015 W. 190th Street
Gardena, CA 90248
(213) 329-8941
DENNIS BROWN

BUILD DIGI-TOLL

Save On Long Distance Phone Calls



A digital timekeeping accessory that displays the time of day and elapsed-time data in step with the Telco billing timer. Use it wisely and reduce toll charges 15%.

FRED BLECHMAN, K6UGT

UNTIL RECENTLY, ONLY THE LARGEST corporations could afford the expensive computerized systems required to effectively manage long-distance telephone call expenses. Now, for less than $50, the *Digi:Toll* (Cervco, Inc., 211 Mill Creek Drive, Youngstown, OH 44512) solid-state digital clock/elapsed timer makes it possible to control the cost of long-distance and other toll calls *as they are being made*.

The *Digi:Toll* is a digital timekeeping instrument designed specifically for toll-call management applications. Although it is *not* connected to the telephone line, its special circuits continuously display the same time-of-day and elapsed-time data used by the telephone company to calculate toll charges. With this data and an understanding of phone company timing and discount procedures (covered later on in this article) you can reduce the monthly cost of long-distance and timed local calls by 10% to 20% on each phone equipped with a *Digi:Toll*. Table I shows the projected savings possible if the *Digi:Toll* only reduces your toll charges by 15%.

The standard *Digi:Toll* can be operated from a 117-VAC 60-Hz source and functions as a 12-hour time-of-day clock and a 24-minute call timer. Four versions can be built by cutting foil traces and using wire jumpers. The 24-hour-display "T" version is useful for ham operation or for international calls. The "L" (Legal) version can be used by attorneys, accountants, advertising agencies and others who need a long-duration (24-hour) timer for billing, timing conferences, etc., as well as a 24-minute call timer. The "W" (WATS) version *Digi:Toll* is a *totalizing* timer for WATS-line users. This version *totals* (up to 24 hours) each individual call measured on the 24-minute timer. Also, for use in foreign countries at 50 Hz, modification "P," usable with all other versions, can be constructed with only one foil break and one jumper wire.

All *Digi:Toll* versions are also useful for special applications such as in photo and test labs, where they can even be built into other equipment.

Features of the unit

The use of digital display is a significant improvement over past telephone-timing devices, such as stopwatches and hourglasses, since it combines continuous data presentation with ease of interpretation. The *Digi:Toll*'s unique physical design (Patent No. 242,847) contributes to its overall effectiveness. The anodized aluminum enclosure fits conveniently in front of all commonly used desk phones and is held firmly in place, ready for instant use, by the weight of the telephone itself, as shown in Fig. 1. This arrangement encourages continued use of the *Digi:Toll* for cost-reduction purposes. Since no connection to the phone lines is required, no phone company approval or monthly charges are involved.

The extra-bright ½-inch-high LED display and special glare-reducing filter make for easy readability under a wide range of viewing angles and ambient light conditions. Important timing and discount-data tables are prominently located

TABLE I—TYPICAL SAVINGS USING A DIGI: TOLL

Average Long Distance Cost/Month/Phone ($)	Savings/Month at 15% ($)	1st Year Savings ($)	5-Year Savings ($)
25	3.75	45	225
50	7.50	90	450
75	11.25	135	675
100	15.00	180	900
125	18.75	225	1125
150	22.50	270	1350
175	26.25	315	1575
200	30.00	360	1800