Popular Electronics®
AUGUST 1976

# BUILD THE COSMAC "ELF"

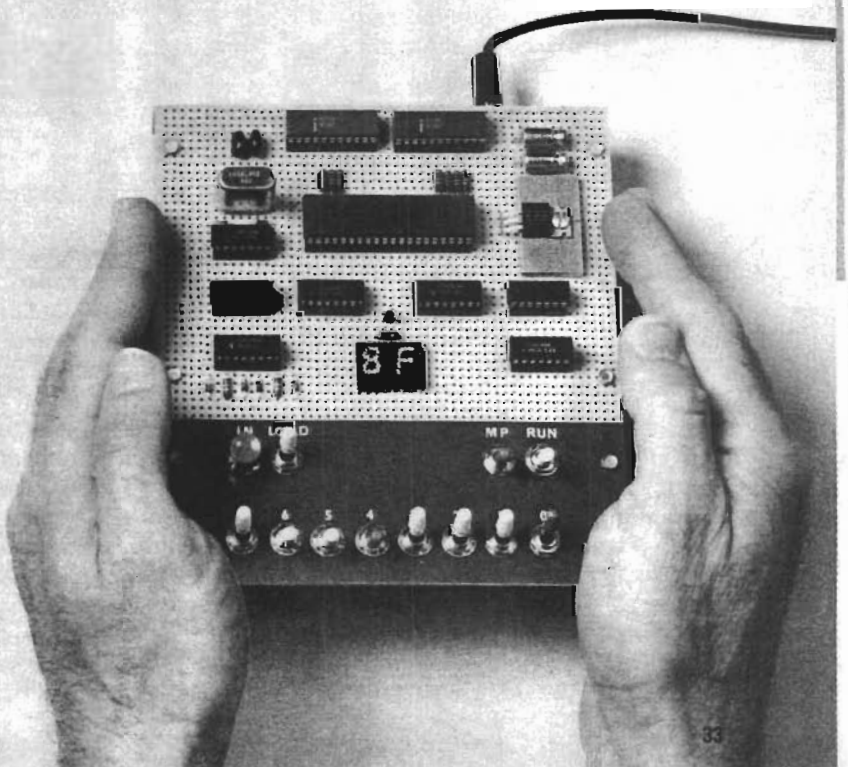
## A LOW-COST EXPERIMENTER'S MICROCOMPUTER

## PART 1

PE TESTED

*Simple-to-build computer trainer can be expanded for advanced applications.*

BY JOSEPH WEISBECKER




THERE are basically two ways in which you can get involved with microcomputers on the nonprofessional level. You can buy one of several reasonably priced hobby computer kits, add a TV or typewriter terminal, and learn to use high-level language. On the other hand, you can build your own inexpensive system from scratch. This permits you to experiment with simple applications that do not require an expensive terminal or a large memory. You can communicate with the computer in a relatively simple language.

The "Elf" microcomputer project gives you the latter category of computer system—for about $80. It is an excellent hardware and software trainer that uses machine language and can be easily expanded to do just about anything a full-blown microcomputer can. Packaging, however, is up to you.

The basic Elf has toggle-switch input, hex LED display, 256 bytes of RAM, four input lines and a latched output line. It can be used to play games, sequence lights, control motors, generate test pulses, count or time events, monitor intruder-alert devices, etc. You can do all these things while learning how to program in order to produce a "real" output to determine whether or not the program you designed works. If you prefer not to control or time things, a simple LED can be used to indicate whether or not your program works.

Our focus here is on the construction of the low-cost computer and some simple examples of programming.

**Design Details.** The heart of the Elf microcomputer is the new RCA CDP1802 COSMAC microprocessor chip that sells for less than $30. The chip can use any combination of standard RAM and ROM devices and can address up to 65,536 (65 k) bytes of memory. It has flexible programmed I/O and program-interrupt modes, an on-chip DMA (direct memory access), four I/O flag inputs directly tested by branch instructions, and a 16 × 16 matrix of registers for use as multiple program counters, data pointers, or data registers.

Other features of the 1802 chip include voltage operation between 3 and 12 volts dc at very low current drain, TTL compatibility, built-in clock, and simplified interfacing. There is also a built-in program load-

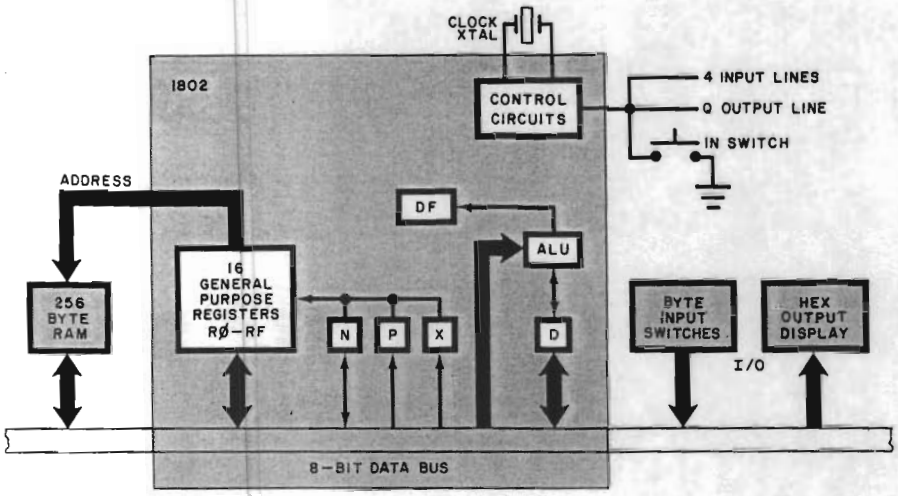



*Fig. 1. Block diagram of basic computer. Up to 65K bytes of memory, 91 instructions, and varied I/O ports can be added as the system grows.*

ing capability that allows you to load a sequence of bytes without having to toggle in a new address for each byte. No ROM is required for the minimum trainer system described here. The multiple program counters permit some interesting programming "tricks," and the many single byte instructions keep programs short.

A block diagram of the Elf system is shown in Fig. 1. The pinout for the 1802 microprocessor chip is shown in Fig. 2.

**Basic Operation.** The key to understanding the computer is the method used for addressing the memory. At first, the procedure may appear to be complicated, but you will soon see that it is not difficult.

The 1802 chip contains 16 general-purpose registers, each holding 16 bits (two bytes) of memory addresses or data. The registers are labelled R0 through RF to conform to the hexadecimal numbering system, as shown in Fig. 3. (In the diagrams, and in computer technology in general, a Danish zero—a zero with a slash through it—is used to distinguish zero from a capital letter O.) Hence, if we refer to the low-order, or least-significant, byte of R1, we can call it R1.0, while the high order byte of RF would be called RF.1.

There is also an 8-bit D register that is used to move bytes around. In the instruction set shown in part in the Instruction Subset Table, note that the 8N (8 with a digit) code will copy a low-order general register byte into register D. Writing this instruction as 81 in a program will cause R1.0 to be copied into D when the instruction is executed. We can then use instruction BF (BN in the table, with B and a digit) to copy the D byte into RF.1. It takes two bytes, 81 BF, to transfer a byte from R1.0 to RF.1 via temporary holding register D. The byte in D can also be used in arithmetic operations performed by the ALU (arithmetic logic unit) circuits.

There are three other important registers that are labelled N, P, and X. Each can hold a 4-bit digit that is used to select one of the 16 general-purpose registers. For example, if you wanted to talk about the general-purpose register selected by the hex digit in X, you would call it RX. If you wanted just the low-order byte of RX, call it RX.0. RN would refer to the general-purpose register designated by the 4-bit digit currently contained in N; if the digit is 4, RN = R4.

The general-purpose registers can contain 16-bit memory addresses. Suppose register R3 contains data 0012. M3 would mean the memory location specified by the contents of R3, and M(0012) means memory location 0012 directly. MX means the memory location addressed by the contents of the general register selected by the current digit in X. If X = 3, MX = M3; if R3 = 0012, MX = M3 = M(0012).

Since the basic computer has only 256 bytes of memory, we use just the low-order bytes of the general registers to address the memory. In expanded-memory systems, you can use the high-order bytes of the general-purpose registers to select individual 256-byte pages of random-access memory (RAM).

The memory contains both instructions and data bytes. Instruction bytes tell the computer what to do with the data bytes. One-byte instructions have two hex digits, where high-order bits 7, 6, 5, and 4 tell the computer what type of operation to perform. Low-order bits, 3, 2, 1, and 0 are usually placed in the N register when a new instruction is fetched from memory.

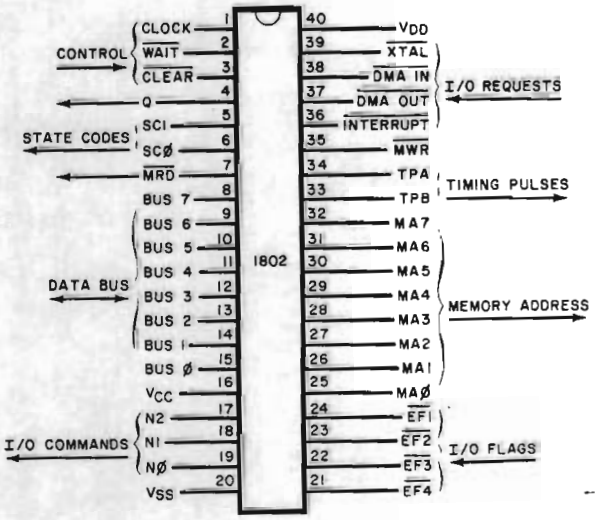



*Fig. 2. Pin out for the CDP1802 COSMAC microprocessor.*

Any one of the general-purpose registers can be used as a program counter. The program counter addresses instruction bytes in memory. Each time an instruction is fetched from memory, the program counter is

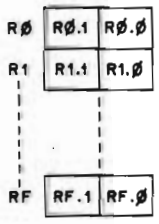



*Fig. 3. The 16 registers in the 1802 are labelled RØ through RF (hex).*

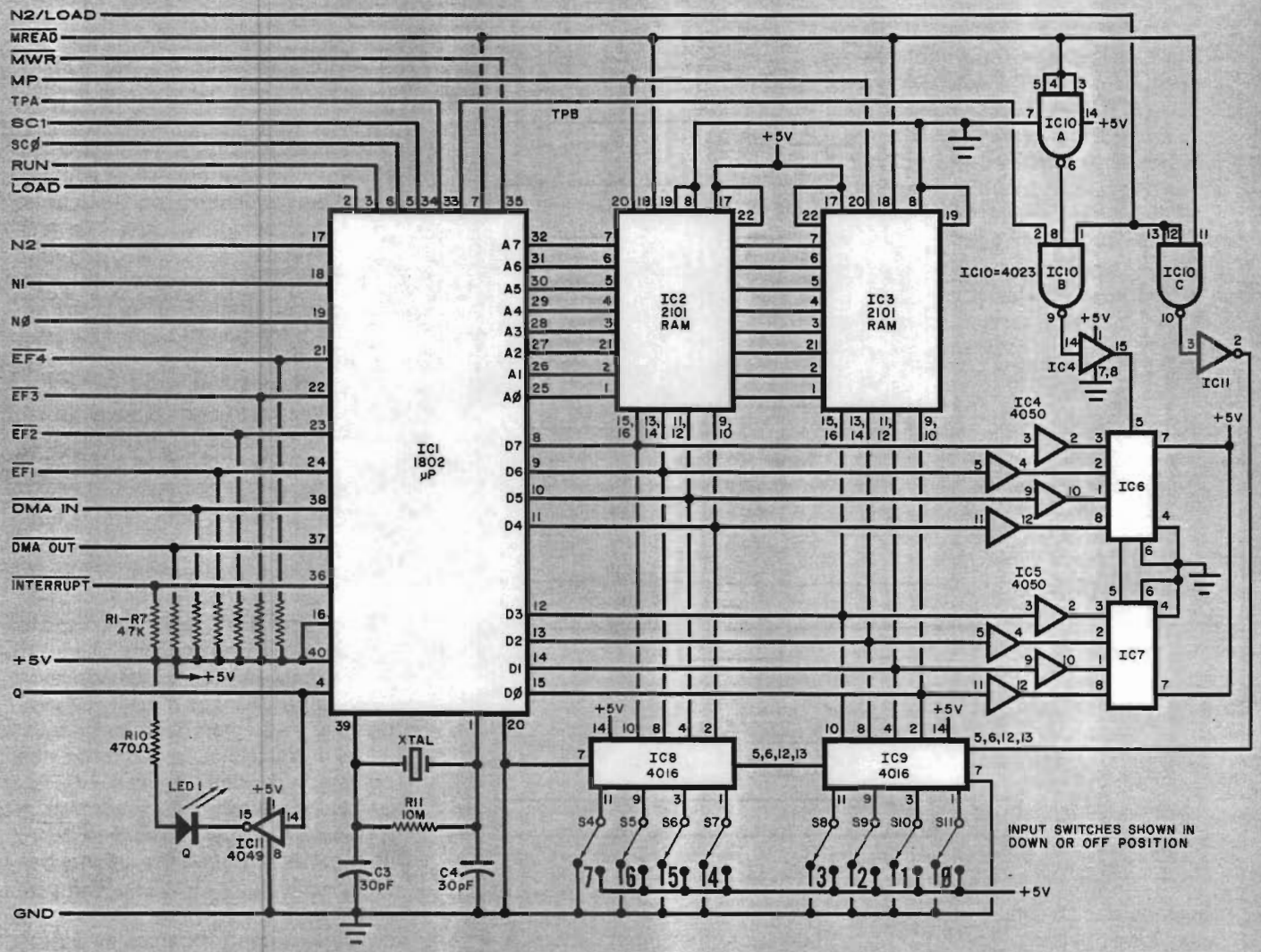



*Fig. 4. Complete circuit for the Elf computer. Identified connections on the left go to the "front panel" with the eight data switches. The remaining can be left "floating" at 1802, or tied to terminal strip.*

## PARTS LIST

C1, C2—10-μF, 16-volt electrolytic capacitor
C3, C4—30-pF disc capacitor
D1 through D6—IN914 switching diode
IC1—CDP1802 COSMAC microprocessor chip (RCA)
IC2, IC3—2101 (256 × 4) static RAM IC
IC4, IC5—4050 noninverting hex buffer IC
IC6, IC7—Hex LED display (H-P No. 5082-7340)
IC8, IC9—4016 quad bilateral switch IC
IC10—4023 triple 3-input NAND gate IC
IC11—4049 inverting hex buffer IC
IC12—4013 dual D flip-flop IC
IC13—LM309K 5-volt regulator IC
LED1—Red light-emitting diode
R1 through R9—47,000-ohm, ¼-watt resistor
R10—470-ohm, ¼-watt resistor
R11—10-megohm, ¼-watt resistor
S1 through S11—Spdt toggle switch
S12—Pushbutton switch with one set each normally open and normally closed contacts
XTAL—1-to-2-MHz crystal (see text)
Misc.—5½" × 4" (14 × 10.1 cm) perforated board with 0.1" (2.54 cm) hole spacing; 5½" × 2" (14 × 5.1cm) piece of thin aluminum; ¾" × ⅜" (19.1 × 9.5 cm) pine for chassis rails; 14-pin IC sockets (4); 16-pin IC sockets (3); 22-pin IC sockets (2); 40-pin IC socket; connector for power supply; 9-volt, 350-mA dc power source; 1¼" × ¾" × ⅛" (31.8 × 19.1 × 3.2 mm) piece of aluminum; dry-transfer lettering kit; machine and wood hardware; hookup wire; solder; etc.

Note: the CDP1802 COSMAC microprocessor chip is available from any RCA parts distributor as is the COSMAC user manual.

automatically incremented so that it points to the next instruction to be fetched. Branch instructions can be used to change the address in the program counter to permit jumping (branching) to a different part of the program when desired. The digit in the 4-bit P register specifies which 16-bit general-purpose register is being used as the program counter.

**Timing Sequence.** Since many of the 1802 microprocessor's instructions are only one-byte long and require two machine cycles, the first cycle is always an instruction fetch, or memory read. The fetched instruction is executed during the next machine cycle, which could be a memory-read, memory-write, or register-transfer type of cycle.

Program execution always consists of a sequence of fetch-execute cycles, and the two SC0 and SC1 lines (see Fig. 4 and Fig. 5) indicate what type of cycle is being performed according to the following criteria:

| SC1 | SC0 | Type of Machine Cycle |
|---|---|---|
| 0 | 0 | instruction fetch |
| 0 | 1 | instruction execute |
| 1 | 0 | DMA in/out |
| 1 | 1 | interrupt |

Direct memory access (DMA) and interrupt are special types of cycles, which we will discuss later.

Circuit timing is shown in Fig. 6.

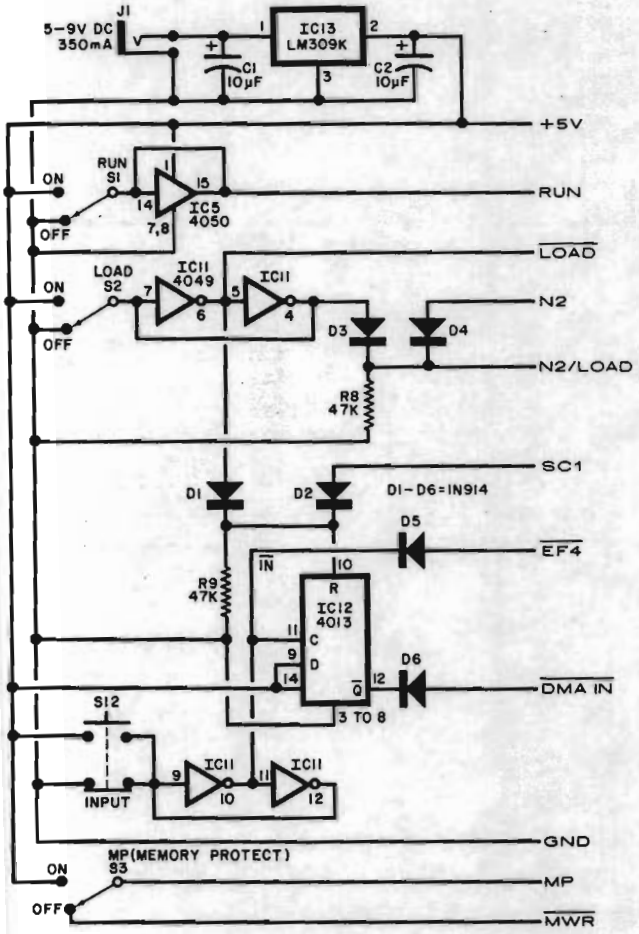



*Fig. 5. Control circuits for the computer. Connections at right go to similarly marked connections on main circuit.*

Note that each machine cycle requires eight clock pulses.

The microprocessor has an internal single-phase clock circuit. Connecting a crystal between pins 1 and 39 of the 1802 chip causes the clock to run continuously. If desired, *XTAL, C3, C4,* and *R11* can be omitted and an external clock with a 5-volt swing can be substituted between pin 1 and ground.

During each machine cycle, timing pulses TPA and TPB are available at pins 33 and 34 of the 1802. TPA occurs at the beginning of each machine cycle and can be used to clock the high-order byte of a 16-bit memory address into a memory page-selection register. Note that the 1802 sends out memory addresses as two 8-bit bytes. The high-order byte appears on address lines A0 through A7 first. Then the low-order byte is held on the A0 through A7 lines for the remainder of the machine cycle. This low-order address byte can, by itself, specify one of 256 locations in the minimum 256-byte memory.

TPB occurs toward the end of the machine cycle and is used to clock a byte from the RAM into an output device (such as the hex display used here). An input byte, to be stored in the RAM, is gated to the bus for the duration of the input (memory-write) machine cycle so that no time pulse is needed for input bytes.

The MREAD line is low during any memory-read machine cycle. When low, it opens the pin-18 RAM data output gates of *IC2* and *IC3*, permitting the byte stored in the RAM location addressed by A0 through A7 to appear on the data bus. The RAM's access time is such that the output byte appears on the bus prior to TPB. The bus byte from the RAM can then be clocked into an internal register of the 1802 or clocked to an external register (such as the hex display) with TPB, depending on the type of instruction being executed.

When the 1802 is performing an instruction cycle that requires a byte to be stored in the RAM, the MREAD line is held high to disable the RAM output bus gates. The microprocessor then causes the byte stored in the RAM to be gated onto the bus during the memory-write cycle. This byte can come from an internal register of the 1802 or from an input device (such as switches), depending on the type of instruction being executed. The 1802 then generates a low memory-write pulse (MWR) that causes the bus byte to be stored in the RAM location addressed by the A0 through A7 lines.

**Circuit Operation.** Using Fig. 4, Fig. 5, and the Instruction Subset Table, we can now discuss the logic of the Elf microcomputer. The RAM address is sent out on lines A0 through A7. Eight tri-state bidirectional bus lines are used to transfer the data bytes back and forth between the 1802's registers and the *IC2-IC3* RAM. A RAM byte can be transferred to hex displays *IC6* and *IC7* via the data bus, using *IC4* and *IC5* to supply the current drive for the displays. Displays *IC6* and *IC7* contain latches to store the display byte.

The basic clock frequency of the processor is determined by *XTAL*, which should not go above 2 MHz in this circuit. The MREAD and MWR lines control the read and write cycles of the RAM, while TPA and TPB provide the timing pulses. TPA can be used for memory expansion address latching, TPB to clock bytes into output circuits. SC0 and SC1 indicate the type of cycle being performed by the 1802.

The N0, N1, and N2 lines are used to select input or output devices. In the Elf, selection can be made among four input and four output devices. The table details the values of the N0, N1, and N2 lines during the machine cycle in which an input or output instruction is executed. Instructions 69, 6A, 6B, 61, 62, and 63 are spares that can be used to add I/O devices or ports to the computer. When 6C is executed, the N2 line goes to a logic-1 state and the bus byte is written into the RAM. Since this is a write cycle, MREAD will be high. With both N2 and MREAD high, the output of gate *IC10C* will be low, putting the input toggle switch byte on

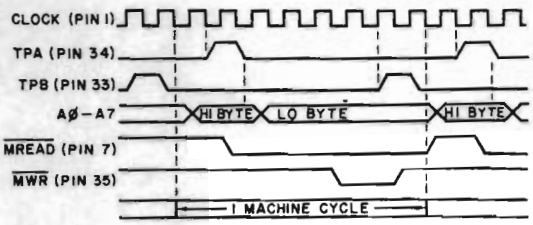



*Fig. 6. Microprocessor timing. One machine cycle requires eight clock pulses. TPA and TPB control various functions, both on and off the computer.*

| ONE BYTE INSTRUCTIONS | |
|---|---|
| 1N | RN+1 |
| 2N | RN-1 |
| 8N | RN.0→D |
| 9N | RN.1→D |
| AN | D→RN.0 |
| BN | D→RN.1 |
| 4N | MN→D,RN+1 |
| 5N | D→MN |
| DN | N→P |
| EN | N→X |
| 7A | 0→Q (LIGHT OFF) |
| 7B | 1→Q (LIGHT ON) |
| F0 | MX→D |
| F1 | MX or D→D |
| F2 | MX and D→D |
| F3 | MX xor D→D |
| F6 | SHIFT D RIGHT,BIT 0→DF |
| 76 | ROTATE D RIGHT,DF→B7,B0→DF |
| FE | SHIFT D LEFT, BIT 7→DF |
| 7E | ROTATE D LEFT,DF→B0,B7→DF |
| F5 | MX-D→D,CARRY→DF |
| F7 | D-MX→D,CARRY→DF |
| F4 | MX+D→D,CARRY→DF |

| TWO BYTE INSTRUCTIONS | |
|---|---|
| 30MM | GO TO MM |
| 31MM | GO TO MM IF Q=1 |
| 39MM | GO TO MM IF Q=0 |
| 32MM | GO TO MM IF D=00 |
| 3AMM | GO TO MM IF D≠00 |
| 33MM | GO TO MM IF DF=1 |
| 3BMM | GO TO MM IF DF=0 |
| 34MM | GO TO MM IF EF1=1 |
| 3CMM | GO TO MM IF EF1=0 |
| 35MM | GO TO MM IF EF2=1 |
| 3DMM | GO TO MM IF EF2=0 |
| 36MM | GO TO MM IF EF3=1 |
| 3EMM | GO TO MM IF EF3=0 |
| 37MM | GO TO MM IF EF4=1 (IN SWITCH) |
| 3FMM | GO TO MM IF EF4=0 (IN SWITCH) |
| F8KK | KK→D |
| F9KK | KK or D→D |
| FAKK | KK and D→D |
| FBKK | KK xor D→D |
| FDKK | KK-D→D,CARRY→DF |
| FFKK | D-KK→D,CARRY→DF |
| FCKK | KK+D→D,CARRY→DF |
| 7CKK | KK+D+DF→D,CARRY→DF |

| ONE BYTE INPUT INSTRUCTIONS | | N2 | N1 | N0 |
|---|---|---|---|---|
| 69 | BUS→MX,D | 0 | 0 | 1 |
| 6A | BUS→MX,D | 0 | 1 | 0 |
| 6B | BUS→MX,D | 0 | 1 | 1 |
| 6C | INPUT SWITCH BYTE→MX,D | 1 | 0 | 0 |

| 1-BYTE OUTPUT INSTRUCTIONS | | N2 | N1 | N0 |
|---|---|---|---|---|
| 61 | MX→BUS,RX+1 | 0 | 0 | 1 |
| 62 | MX→BUS,RX+1 | 0 | 1 | 0 |
| 63 | MX→BUS,RX+1 | 0 | 1 | 1 |
| 64 | MX→HEX DISPLAY,RX+1 | 1 | 0 | 0 |

*Instruction Subset Table shows required sequence of steps.*

the bus so that it can be stored at the memory location addressed by RX. This input byte will also be placed in the 1802's D register.

When a 64 instruction is executed, N2 is high and $\overline{\text{MREAD}}$ is low, making the output of *IC10C* high and preventing the input switch byte from getting onto the bus. Instead, gate *IC10B* generates an output clock pulse with TPB that clocks the RAM output byte into the hex display.

The four external flag input lines—EF1, EF2, EF3, and EF4—can be pulled low by external switches. These four lines can be tested by instructions 34, 3C, 35, 3D, 36, 3E, 37 and 3F. Note in Fig. 5 that the INPUT pushbutton switch, debounced by portions of *IC11*, is connected to the $\overline{\text{EF4}}$ line. This means that $\overline{\text{EF4}}$ = 1 when *S12* is depressed and $\overline{\text{EF4}}$ = 0 when *S12* is in its normal position.

Latched output line Q can be set high by a 7B instruction or reset to low by a 7A instruction. The Q LED comes on when Q is high. The $\overline{\text{DMA IN}}$, $\overline{\text{DMA OUT}}$, and $\overline{\text{INTERRUPT}}$ lines can be pulled low to cause these operations to occur.

The $\overline{\text{LOAD}}$ and RUN lines control the operation of the microprocessor according to the following conditions:

| $\overline{\text{LOAD}}$ | RUN | Mode |
|---|---|---|
| gnd | gnd | load |
| +5V | gnd | reset |
| gnd | +5V | — |
| +5V | +5V | run |

RUN and LOAD switches *S1* and *S2* in Fig. 5 control the operation of the computer. With both switches set to OFF, $\overline{\text{LOAD}}$ is +5V and RUN is at ground potential. This resets the 1802. Neither TPA nor TPB are generated in the reset state and R0 = 0000, P = 0, X = 0 and Q = 0 after the 1802 is reset. When the LOAD switch is set to ON, $\overline{\text{LOAD}}$ goes low and RUN stays low, forcing the system into the load mode. Now you can load a sequence of bytes into the RAM, starting at address 0000, by setting the bytes into the input toggle switches, one at a time, and operating the INPUT switch.

In the load mode, the 1802 does not execute instructions but waits for a low to appear on the $\overline{\text{DMA IN}}$ line. When this happens, the 1802 performs one memory write cycle during which the switch input byte is stored in memory. R0 is used to address memory during the DMA IN cycle. After the input byte is stored at the address specified by R0, this register is in-

## INTRODUCTION TO PROGRAMMING.

Once you have built your Elf, you must learn how to load a sequence of bytes into memory and then go back and display the sequence. Let us write a simple program that can be loaded into the memory and run.

Suppose you want to program the computer to turn on the Q LED whenever the INPUT switch is set to ON. First, you must draw a flow chart that shows the required sequence of steps (Fig. 7). Locate the correct instructions in the Instruction Subset Table. A 7A instruction will perform Step 1. Load this instruction into M(0000). Note that when the INPUT switch is not depressed, EF4 = 0. A two-byte 3F 00 instruction will jump (branch) back to the 7A instruction at M(0000) as long as the INPUT switch is not operated (EF4 = 0). This condition is known as a "loop," and the program will stay in this loop while it is waiting for the INPUT switch to be depressed. Load 3F 00 into memory locations M(0001) and M(0002) to perform the second step in the flow chart. All GO TO MM instructions shown in the Table put MM into the low-order byte of the program counter if a GO TO condition exists. Otherwise, the next instruction in sequence is fetched by the 1802.

Loading a one-byte 7B instruction into M(0003) takes care of Step 3, while a 30 01 instruction will jump back to the 3F 00 instruction at M(0001). Load the 30 01 instruction into M(0004) and M(0005) to complete the program.

You load this 6-byte program by placing the LOAD switch on the ON position, with RUN and MP set to OFF, setting up the toggle switches for the hex number 7A, and depressing the INPUT switch. Release the INPUT switch, insert 3F and operate the INPUT switch again. Then load 00 and so on until the last byte, 01, has been stored at M(0005). If you "blow" the program, set MP to ON and LOAD to OFF. Then set LOAD to ON and operate the INPUT switch until you get to the byte immediately preceding the wrong entry. Set MP to OFF, set up the correct byte, and operate INPUT. Flip MP back to ON to protect what you have stored in memory.

To start the program running, set LOAD to the down position to reset the 1802 and set the RUN switch to ON. Nothing should happen until you depress the INPUT switch, at which time the Q LED should come on. Releasing the INPUT switch should cause the LED to extinguish. If you like, you can now observe the timing signals of the 1802 on an oscilloscope while the program is running.

Another simple program involves counting the number of times the INPUT switch is operated and then turning on the Q LED at the end of the count. The flow chart for this program is shown in Fig. 8. When you load and run this program, nothing will happen until you operate the INPUT switch five times, at which point the LED will come on and remain on. Note in Step 1 that you can change the number of times the INPUT switch is operated. Step 6 just loops on itself to terminate the program after the INPUT switch has been operated the specified number of times.

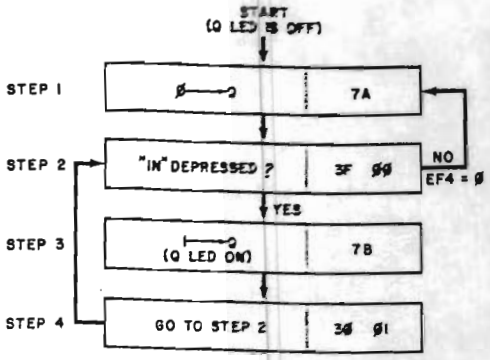



*Fig. 7. Program turns on Q-LED when INPUT switch is operated.*

cremented by one so that input bytes will be sequentially loaded into RAM locations. Line SC1 goes high during the DMA IN cycle so that the control circuits know when the input byte has been stored in the RAM.

Depressing and releasing INPUT switch *S12* sets flip-flop *IC12* (Fig. 5). The $\overline{Q}$ output of this stage goes low, causing the required low on the $\overline{\text{DMA IN}}$ line. The 1802 responds to this request with a memory-write cycle during which SC1 is high. During this cycle, $\overline{\text{MREAD}}$ is high and, since LOAD switch *S2* is also ON, the N2/LOAD signal causes gate *IC10C* to go high, gating the switch input byte to the data bus and storing it in memory. When SC1 goes high, it also resets *IC12*, which causes $\overline{\text{DMA IN}}$ to return to its high state. The computer then waits for the next switch input byte and LOAD switch operation.

Following each DMA IN cycle, the 1802 holds the A0 through A7 lines at the address of the byte just stored in the RAM. $\overline{\text{MREAD}}$ is also held low while waiting for the next input byte. This means that the previously loaded byte is being gated to the bus (from the RAM) while waiting for a new byte. This bus byte is continuously clocked into the hex display, since the LOAD switch is holding *IC10B* open.

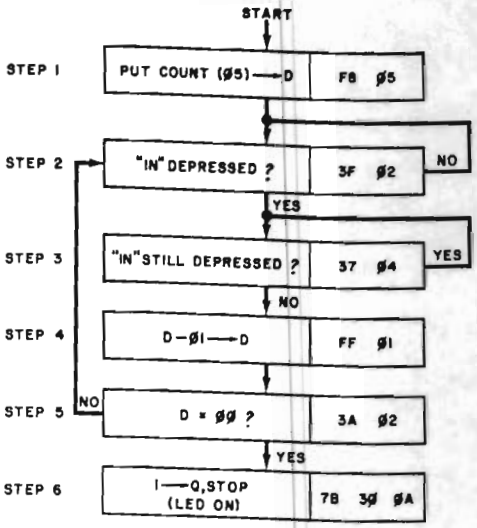



*Fig. 8. Program counts number of times INPUT switch is operated.*

A sequence of program bytes can be loaded into the RAM starting at M0 = M(0000) by setting the LOAD switch to the ON position, with the RUN switch set to OFF. Set the eight input switches, *S4* through *S11*, to the desired byte code (in hexadecimal) and depress the INPUT switch to store the byte in the RAM. The value of this byte will be displayed with the hex displays *IC6* and *IC7*. Repeat this procedure for each byte to be loaded. Setting the LOAD switch to OFF puts the 1802 back in the reset state where R0 = 0000, P = 0, X = 0, and Q = 0. If you wish to see what is stored in memory, set MP (memory-protect) switch *S3* and the LOAD switch to ON. Now, each time you operate the INPUT switch, successive bytes in the RAM, starting with M(0000), will be displayed.

To change a byte, proceed to the byte just before the one to be changed. Flip the MP switch to OFF, set the input toggle switches to the hex value of the new byte, and depress the INPUT switch once. This new byte will be displayed and stored in the RAM at the location following the byte at which you stopped. Place the MP switch in the ON position. You can now continue to operate the INPUT switch to sequence through the RAM without modifying the bytes in memory.

To start the executive cycle of a program, set both the LOAD and RUN switches to OFF (to reset the 1802). Then set the RUN switch to ON. The program counter is always specified by the hex digit in register P, which can be set to zero by reset so that the program counter will always initially be R0. Set R0 to 0000 by resetting so that instruction fetching, or program execution, will always begin at M(0000). Instructions will continue to be fetched from the RAM and executed until the RUN switch is set to OFF, resetting the computer. Make sure that the MP switch is OFF when running programs so that computer operation is not inhibited.

**Construction Notes.** Hardware assembly is relatively simple, permitting the project to be put together with ordinary perforated board with 0.1" (2.54-mm) hole spacing and IC sockets, using either Wire Wrap® or a wiring pencil. (See photo.) The perf board measures 5½"L × 4"W (14 × 10.2 cm) and is supported on a base made up of lengths of ¾" × ⅜" (19.1 × 9.5) pine. A sheet of thin aluminum provides the support for the eight toggle-type data switches. The LM309 voltage regulator IC (*IC13*) is mounted on a 1¼" × ¾" × ⅛" (31.6 × 19.1 × 3.2-mm) piece of aluminum to serve as a heat sink.

Do not mount the IC's (except the display devices) in their sockets until after all wiring is complete. Socket, switch, and component layout should be roughly the same as shown in the photo. Be sure to locate the crystal close to pins 1 and 39 of the microprocessor's socket. Then wire the circuit in accordance with the schematics in Figs. 4 and 5.

Any crystal with a frequency of between 1 and 2 MHz can be used in the Elf, or you can substitute a simple 555 or CMOS oscillator with a 5-volt signal swing between pin 1 of the 1802 and circuit ground, in which case, you will have to omit *XTAL, C3, C4,* and *R11*. There is no lower limit to the clock frequency, but most of the sample programs discussed in this series of articles are based on a clock frequency between 1 and 1.8 MHz.

Displays *IC6* and *IC7* are relatively expensive hex devices. They are the only TTL devices in the computer and, as a result, draw most of the power required by the circuit. If you wish to economize, you can substitute ordinary LED's for the displays. (Next month, we will discuss how to make the substitution.)

An inexpensive 9-volt, 350-mA dc battery eliminator, like those used as battery charger/eliminators for calculators, can be used to power the Elf.

When the computer is completely assembled, use a dry-transfer lettering kit to label all switches and positions, IC socket locations, and pins 1 of all sockets. Then, exercising the usual safety procedures for handling MOS devices, install the integrated circuits in their respective sockets.

**Coming Up.** In future articles, we will provide more programs as well as methods of adding other types of inputs and relay-control output circuits. We will also detail how to save programs in battery-powered COSMOS RAM's and describe a simple operating system that lets you read/write any memory location and inspect general register contents for program debugging purposes. Memory expansion, hex keyboard input, and an inxpensive video graphics display are other subjects we will cover in detail. ◈