by George Meyerle

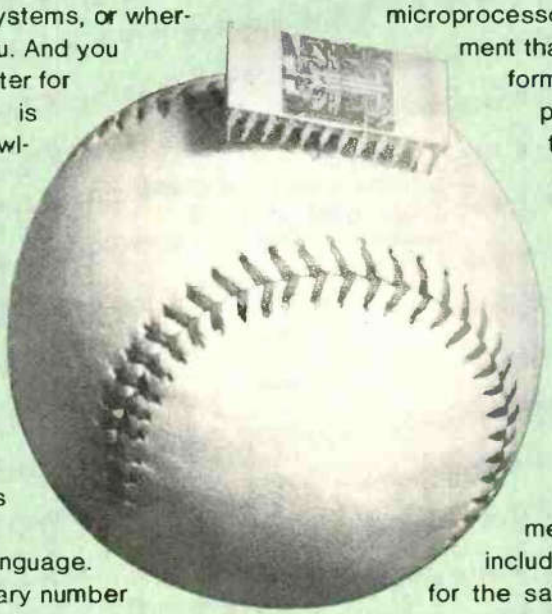
Part 1 of a series devoted to the use of inexpensive microprocessors in custom applications

Microprocessor Applications for the 1980's...

It's a Whole New Ballgame!

THE MICROPROCESSOR (central processing unit or CPU), a powerful and versatile integrated circuit, was born only in the last decade. We have witnessed its startling price decline and reveled in the result—modestly priced computers.

With $10 microprocessors available now, it's clear that the devices can be used economically for *non-computer* purposes—electronic games, telephone dialers, photographic timers, robots, "smart" thermostats, sophisticated security systems, or wherever your imagination leads you. And you need not tie up a $1000 computer for these applications. All that is required is a reasonable knowledge of microprocessors, which will also give you a better understanding of computer hardware and software.



To use a microprocessor for any of a host of applications, you need only become familiar with:

- The way the processor functions and relates to its inputs and outputs.
- The processor's program language.
- The fundamentals of the binary number system.

The foregoing doesn't require much more effort than learning about 100 words or so of a foreign language and some simple grammar. Mastering this, you can use inexpensive microprocessors where discrete parts would be awkward at best and often prohibitively costly and bulky.

Clearly, using microprocessors in the 1980's will be a whole new ballgame for electronics enthusiasts. To be certain that readers of POPULAR ELECTRONICS will be able to keep up with what we plan to present to you as time unfolds, we're launching this first-in-a-series microprocessor training course for the new decade.

What Is a Microprocessor? In its simplest terms, a microprocessor IC can be considered an element that can read data from inputs, perform computations, and control outputs. What makes it special is that it can be *programmed*. That is, it can be made to perform its various functions in any desired sequence. This flexibility is the key characteristic that lets a microprocessor and external circuitry perform such a wide variety of electronic tasks.

A microprocessor-based system is shown in Fig. 1. Note that there are four main elements. These elements can be included in the microprocessor IC, but for the sake of simplicity we will assume that they are all separate components. We will deal with the microprocessor itself separately.

The microprocessor is connected to all of the other components via the data bus by which information is passed back and forth. Control signals from the microprocessor along with the memory address signals determine which elements communicate with

the processor at any one time.

The block labelled Program is the storage area for the sequential instructions to be executed by the microprocessor. On reset or power on, the microprocessor will automatically obtain the first instruction from this memory via the data bus. After executing the first instruction it will signal for the next instruction, etc. To generate this series of instructions, better known as the program, is called "programming."

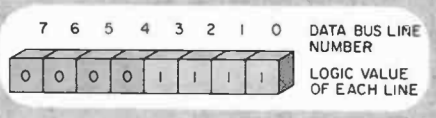
The input port section is the communication link between the microprocessor and data from keyboards, sensor switches, or the like. An input port usually can signal the microprocessor via the control bus when data is available. Output ports are used to transfer data from the microprocessor or memory components to output devices.

As an example, let's consider this system to be the controller for a simple robot and follow some theoretical steps which might occur in its operation. On reset or power on, the processor will signal the program memory to put the first instruction on the data bus. The processor reads and executes that instruction. Let's assume that the instruction tells the microprocessor to read the contents of the input port and store that data in one of its internal registers. The processor now requests the next instruction, which could be to have the microprocessor test the data from the input port. Subsequent instructions, called on the basis of the data analysis, would have the microprocessor issue to the output port data that would cause the robot to take a specific action.

While the robot is performing its mission, the processor will request its next instruction, which might be to reread the input port that monitors the progress of the robot. Each reading of the input port would similarly be tested, with the microprocessor issuing commands to control the course of action. This general system could just as easily operate as a telephone dialer or other product.

Signal Lines. Before we tackle the microprocessor functions, let's review the types of signals you might find on the signal lines. These lines carry voltages that represent binary numbers. The microprocessor, the I/O ports, and memory also respond only to these electrically coded binary numbers. It would be a great asset, therefore, to have an understanding of binary numbers and their decimal and hexadecimal (base 16) equivalents.

In an 8-bit microprocessor, the data bus will consist of 8 lines. The status of each of these lines can either be a logic 1 or a logic 0, each represented by one of a pair of voltage levels (high and low, positive and negative, etc.). If the microprocessor is reading from the data bus, and lines 0, 1, 2, and 3 are logic 1, and lines 4, 5, 6, and 7 are at logic 0, you can write that input as 00001111 as shown.



Writing long lists of these inputs in binary form would be very tedious and cumbersome. It is convenient to convert this binary representation of the 8 bits into a hexidecimal form. Thus, 00001111 can be represented by 0F (hex). Writing a 16-bit address in binary will convince you of the value of binary to hexidecimal conversion. This conversion is listed in Table I.

We cannot, however, completely forget about binary representation of these numbers because in that form they will allow easy identification of which switch is to be turned on for a given input or which light or motor will go on as a result of particular outputs. For example, if we are at an output port capable of driving low-voltage lamps and we want a certain sequence of these lamps to light at any one time, we must remember that the lamp will go on only when there is a logic 1 appearing on the data line at the output port. It will be helpful to remember that binary digit, 0 or 1, is referred to as a bit. This group of eight bits is called a byte. A 16-bit address is thus made up of two bytes. Four bits (half a byte) are sometimes referred to as a nibble.

TABLE I—DECIMAL—BINARY—HEXADECIMAL EQUIVALENTS

Decimal	Binary	Hexadecimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

The RCA 1802. In this article we will discuss the RCA 1802. In subsequent articles, other processors will be described. Although microprocessors vary greatly in terms of their capabilities and specific language or instruction set, they are similar in many functions. Some are faster, more input/output compatible, easier to program, or more suitable for data processing, etc. But if you can understand the basic workings of the 1802 and its instruction set, evaluating any microprocessor specification sheet will be easier. (The complete 1802 specification sheet and instruction summary can be had at no charge by sending a self-addressed, stamped (30c) envelope to Netronics R & D Limited, 333 Litchfield Road, New Milford, CT 06776.)

As our first example does not require all the elements of this processor, we will omit those that will not be used. The processor elements we will need are:

SCRATCH-PAD REGISTERS: There are 16 scratch-pad registers, each of which holds 16 bits. They are used to hold intermediate results. As the data bus can handle only 8 bits at a time, these registers are loaded and unloaded 8 bits at a time—the high-order byte in one operation, the low-order byte in another.

D-REGISTER: The D or Data register is used as an input and output to the ALU register (Arithmetic Logic Unit). Data is transferred to the D register, tested or modified by the ALU register and then returned to the data bus. The DF register is a flag used in arithmetic operations to determine if a carry or borrow occurred.

Q: This is a single-bit output line which can be set or reset by a program instruction. It is often used as an output. For example, it can turn a light on or off.

INPUT FLAGS: The 1802 has four input flags that are tested for their logic level by instructions.

Figure 2 shows a practical hook-up of an 1802 microprocessor and a simple array of peripheral equipment. Control inputs EF1,2,3, and 4 sense when any of the four pushbuttons is closed. The Q output is coupled to transistor *Q1* that, in turn, drives a solenoid. Instructions, called for from ROM via the 8-line address bus, are delivered via the 8-line data bus. This arrangement is sufficient to solve our example problem.

The Problem. For our example application, we have selected a solenoid-operated lock that will open only when four pushbutton switches are operated in the proper sequence. If any button is pressed out of sequence, the controller ignores all inputs for a period of one minute. After that time, it will respond only to the entire combination in correct sequence. To further forestall attempts to solve the combination, each button must be released before the system will register that the next one has been pushed. Finally, when the lock does open, it will remain in that condition for only five seconds.

The Instruction Set. The 91 instructions recognized by the 1802 fall into nine categories. In this introductory article, however, we will need only four. These are the Control, Short Branch, Memory Reference and Register Operations subsets. Each instruction has two identifications—one called a *mnemonic* (memory aid for humans), the other called an *op code* (the digital representation required by the processor). The mnemonic is closely allied to the specified instruction, and in many cases is an abbreviation. This is exemplified by BR for BRanch and REQ for RESET Q.

The mnemonic REQ corresponds to the binary op code 0111 1010, which can, for convenience, be written 7A (hex). Eight-bit binary numbers are often written as two groups of four (nibbles). Each nibble can be converted into a single hex digit. Binary numbers are the *only* ones the processor "understands." Instructions and binary data written in binary form or the hex equivalent are said to be in "machine language." Data is sometimes entered into a processor via a hex keypad that automatically produces a binary output.

Control Instructions. As the category name implies, these instructions are used where some general control over processor operations is required. We will use three of the 10 instructions in this subset. For each, we give the mnemonic, op code, name, and description.

NOP-C4-No Operation. This instruction performs no processor operations. It causes the processor to remain idle for three machine cycles, then fetch the next instruction. It is used mainly in timing applications to generate a delay.

REQ-7A-Reset Q. This instruction causes the processor's Q line to assume a low state (0 volt).

SEQ-7B-Set Q. This instruction causes the Q line to assume the high state (+5 volts).

Branch Instructions. Normally, a processor executes instructions in the order in which they are stored in memory—the one at the lowest-order memory location first, then the next, etc. A branch causes the processor to depart from the sequence, jump to another part of the memory, and execute one or more of the instructions stored there.

Branches are often (but not necessarily) conditional, taking place only when a defined condition arises. They require two bytes of data. The first is the op code for the branch instruction and the second is the address to which the processor will branch. For reasons of its own internal organization, the 1802 has "long" and "short" branch instructions. We will use 10 of the short ones.

BR-30__-Short Branch. Branch to the memory location specified by the byte (__) following the 30 op code.

B1-34__-Short Branch if EF1=1. If the EF1 line has a logic value of 1, the program will branch to the memory location that follows the op code. If EF1 is not 1, the processor goes to the next instruction in sequence.

B2-35__, B3-36__ and B4-37__ perform the same action on the EF2, EF3 and EF4 lines respectively.

BNZ-3A__-Short Branch if D not 0. If the D register contains other than zero, the program will branch to the specified memory location. If it contains zero, the program advances to the next instruction in sequence.

BN1-3C__-Short Branch if EF1=0. If the EF1 input line is a logic zero, the program will branch to the specified memory location. If the EF1 line is 1, the program proceeds to the next instruction in sequence.

BN2-3D__, BN3-3E__, and BN4-3F__ are used similarly to test the EF2, EF3 and EF4 lines, respectively.

Memory Reference. These instructions allow the user to load data into the temporary storage registers in the processor. We use only one of the seven provided.

LDI-F8__-Load Immediate. This instruction places the "__" data, following the F8 op code, into the D register of the processor. If the instruction were F8 55, the processor would place 55 (01010101) in the D register.

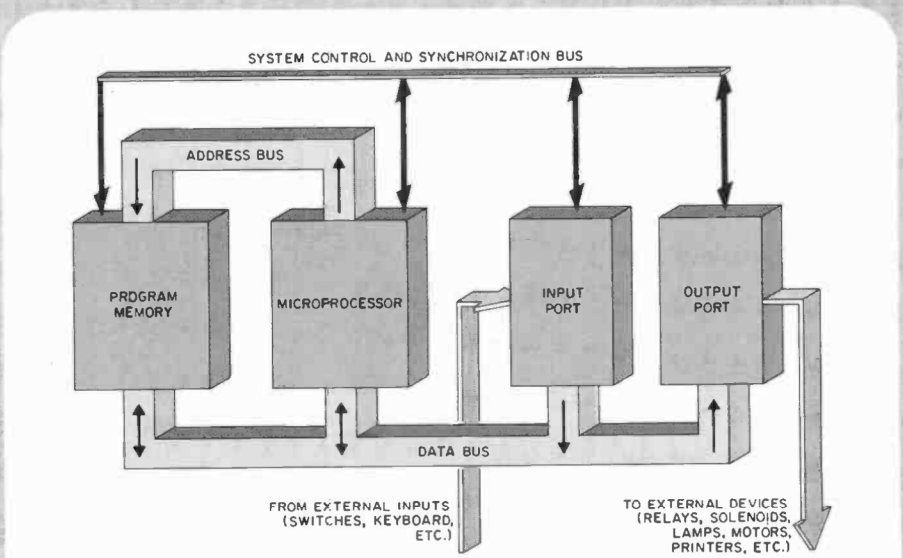


Fig. 1. Block diagram of a microprocessor-based system.

Register Operations. These instructions allow operations to be performed on the data in any of the 16 temporary storage registers of the 1802. These instructions are formed from a hex digit followed by a number to identify the register. We will use four of the available seven instructions.

PLO-AN-Put Low Reg N. Places the data byte currently in the D register into the low-order register specified by N.

PHI-BN-Put High Reg N. Places the data byte currently in the D register into the high-order register specified by N.

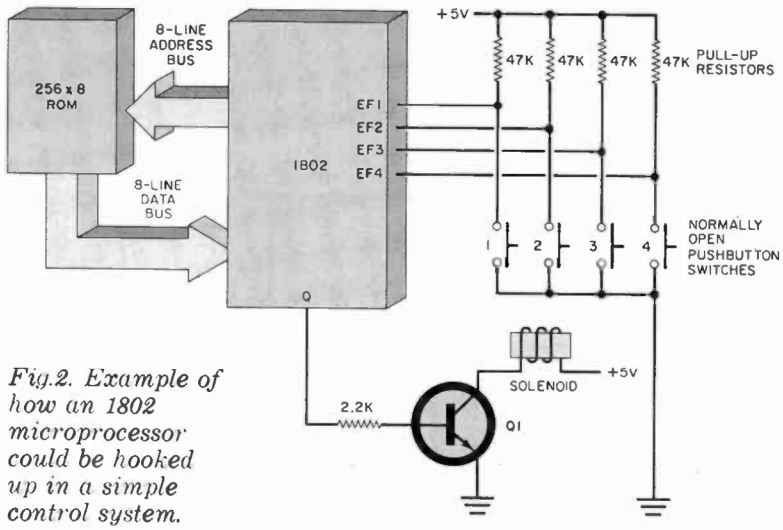


Fig.2. Example of how an 1802 microprocessor could be hooked up in a simple control system.

DEC-2N-Decrement Reg N. Decrement (reduce by 1) the digital value stored in the register specified by N.

GHI-9N-Get High Reg N. Places the data currently in the high-order register designated by N into the D register.

Programming. The example we will use was previously described. However, that description is in human terms that make no sense to the processor. Therefore the problem has to be restated in language that the processor can decode. The restatement will constitute a program.

To create a program, it is convenient to start with a diagram or flowchart that covers all the steps that need be taken by the processor to fulfill the task. The analysis of the task is necessarily very detailed, because any step omitted or misstated, no matter how minor, can cause the program to malfunction.

Creating the Flowchart. As shown in Fig. 3, begin by writing the word START in the center at the top of the page. Since this will also be a label and referred to in the program, write this word on the same line in the LABEL column. Referring to the hardware diagram in Fig. 2, we can see that when the flow starts, we want the solenoid to be de-activated to keep the door locked. Since this occurs when the Q line is low, write RESET Q=0 in a small box directly under START. A line, signifying flow direction, joins the two boxes. The four pushbutton switches must be operated in a 3-4-2-1 sequence, and if any switch is depressed out of sequence, a one-minute time delay will be invoked, after which the flow will proceed directly back to the START. We have also decided that after the correct pushbutton sequence has been entered, the solenoid will be activated for only five seconds.

At this point, some decisions regarding switch condition must be made. A decision box is diamond-shaped with the flow entering the upper corner and either of the three remaining corners used for the YES or NO answers. The first decision, labelled "1st Test," determines if any of the wrong switches (1, 2 or 4) has been closed. If the answer is YES, the flow then proceeds to the one-minute delay. At the conclusion of the delay period, the flow returns back to the START, keeping the solenoid de-activated. If the answer is NO (neither switch 1, 2 or 4 has been operated), the next decision determines if switch 3 (the correct one) has been depressed. If the answer is NO, this means that *no* switches were operated, so the flow goes back to the start of the 1st Test.

As long as *no* switches are touched, the flow "loops" around the 1st Test element, waiting for some switch activity. If switch 3 was depressed, the answer to the second decision box is YES, forcing the flow to proceed to the 1st Release that checks whether switch 3 has been released (opened). If switch 3 has not been released, the flow "loops" around this decision box until switch 3 is released.

The next nine decision boxes, down to 4th Release, ask similar questions of switches 4, 2 and 1 (the correct sequence). If during these queries, a wrong switch is depressed, the YES answer to the decision sends the flow to the 1-minute delay and back to the START. Note that in the flowchart, each pair of switch operation decisions form a labelled Test, and each Release is suitably identified.

Once the four pushbuttons have been properly depressed and released, we want the solenoid activated so that the door will be unlocked. Since the solenoid is activated when the Q line is high, the next block (labelled UNLOCK) is entitled SET Q=1. When the Q line goes high in response to this block, two simultaneous events should occur: the solenoid is activated, and a five-second delay is invoked. At the conclusion of the five-second period, the flow returns to START and de-activates the solenoid. It now scans the 1st Test, awaiting further switch action.

Creating the Program. The program that enables the 1802 to implement the flowchart is shown in Table II. The extreme left column, marked LABEL, defines the various parts of the program corresponding to the flowchart. They are very useful because the various branch addresses are usually filled in after the program has been structured. In the second column, identified as PROGRAM ADDRESS, are the memory addresses where each element of the program will reside. These too can be filled in after the program has been written. The reason for this is that a particular line in the program may consist of two or more bytes, and each byte must have an address. Programming convention is that the program address shown in this column is

Start Computing For Just $129.95 With An 8085-Based Professional Computer Kit—

Explorer/85

100% compatible with all 8080A and 8085 software & development tools!

No matter what your future computing plans may be, Level "A"—at $129.95—is your starting point.

Starting at just $129.95 for a Level "A" operating system, you can now build the exact computer you want. Explorer/85 can be your beginner's system, OEM controller, or IBM-formatted 8" disk small business system...yet you're never forced to spend a penny for a component or feature you don't want and you can expand in small, affordable steps!

Now, for just $129.95, you can own the first level of a fully expandable computer with professional capabilities—a computer which features the advanced Intel 8085 cpu, *thereby giving you immediate access to all software and development tools that exist for both the 8085 and its 8080A predecessor (they are 100% software compatible)*—a computer which features onboard S-100 bus expansion—plus instant conversion to mass storage disk memory with either 5-1/4" diskettes or standard IBM-formatted 8" disks.

For just $129.95 (plus the cost of a power supply, keyboard/terminal and RF modulator, if you don't have them already), Explorer/85 lets you begin *computing* on a significant level... *applying* the principles discussed in leading computer magazines...*developing* "state of the art" computer solutions for both the industrial and leisure environment.

Level "A" Specifications

Explorer/85's Level "A" system features the advanced Intel 8085 cpu, an 8355 ROM with 2k deluxe monitor/operating system, and an 8155 ROM-I/O—all on a single motherboard with room for RAM/ROM/PROM/EPROM and S-100 expansion, *plus* generous prototyping space.

(Level "A" makes a perfect OEM controller for industrial applications and is available in a special Hex Version which can be programmed using the Netronics Hex Keypad/Display.)

Level "A" at $129.95 is a complete operating system, perfect for beginners, hobbiests, or industrial controller use.

PC Board: glass epoxy, plated through holes with solder mask • **I/O:** provisions for 25-pin (DB25) connector for terminal serial I/O, which can also support a paper tape reader ...provision for 24-pin DIP socket for hex keyboard/display...cassette tape recorder input...cassette tape recorder output...cassette tape control output...speaker output... LED output indicator on SOD (serial output) line...printer interface (less drivers)...total of four 8-bit plus one 6-bit I/O ports • **Crystal Frequency:** 6.144 MHz • **Control Switches:** reset and user (RST 7.5) interrupt...additional provisions for RST 5.5, 6.5 and TRAP interrupts onboard • **Counter/Timer:** programmable, 14-bit binary • **System RAM:** 256 bytes located at F8*00*, ideal for smaller systems and for use as an isolated stack area in expanded systems...RAM expandable to 64k via S-100 bus or 4K on motherboard.

System Monitor (Terminal Version): 2k bytes of deluxe system monitor ROM located at F*000* leaving *0000* free for user RAM/ROM. Features include tape load with labeling ...tape dump with labeling...examine/change contents of memory ...insert data...warm start...examine and change all registers...single step with register display at each break point, a debugging/training feature...go to execution address... move blocks of memory from one location to another...fill blocks of memory with a constant...display blocks of memory ...automatic baud rate selection...variable display line length control (1-255 characters/line)...channelized I/O monitor routine with 8-bit parallel output for high speed printer... serial console in and console out channel so that monitor can communicate with I/O ports.

System Monitor (Hex Version): Tape load with labeling... tape dump with labeling...examine/change contents of memory...insert data...warm start...examine and change all registers...single step with register display at each break point ...go to execution address. Level "A" in the *Hex Version* makes a perfect controller for industrial applications and can be programmed using the Netronics Hex Keypad/Display.

Hex Keypad/Display.

Hex Keypad/Display Specifications

Calculator type keypad with 24 system defined and 16 user defined keys. 6 digit calculator type display which displays full address plus data as well as register and status information.

Level "B" Specifications

Level"B" provides the S-100 signals plus buffers/drivers to support up to six S-100 bus boards and includes: address decoding for onboard 4k RAM expansion select-able in 4k blocks...address decoding for onboard 8k EPROM expansion selectable in 8k blocks...address and data bus drivers for onboard expansion...wait state generator (jumper selectable), to allow the use of slower memories...two separate 5 volt regulators.

Explorer/85 with Level "C" card cage.

Level "C" Specifications

Level "C" expands Explorer's motherboard with a card cage, allowing you to plug up to six S-100 cards directly into the motherboard. Both cage and cards are neatly contained inside Explorer's deluxe steel cabinet. Level "C" includes a sheet metal superstructure, a 5-card gold plated S-100 extension PC board which plugs into the motherboard. Just add required number of S-100 connectors

Level "D" Specifications

Level "D" provides 4k or RAM, power supply regulation, filtering decoupling components and sockets to expand your Explorer/85 memory to 4k (plus the original 256 bytes located in the 8155A). The static RAM can be located anywhere from *0000* to EFFF in 4k blocks.

Level "E" Specifications

Level "E" adds sockets for 8k of EPROM to use the popular Intel 2716 or the TI 2516. It includes all sockets, power supply regulator, heat sink, filtering and decoupling components. Sockets may also be used for soon to be available RAM IC's (allowing for up to 12k of onboard RAM).

Order A Coordinated Explorer/85 Applications Pak!

Experimenter's Pak (SAVE $12.50)—Buy Level "A" and Hex Keypad/Display for **$199.90** and get FREE Intel 8085 user's manual plus FREE postage & handling!

Student Pak (SAVE $24.45)—Buy Level "A," ASCII Keyboard/Computer Terminal, and Power Supply for **$319.85** and get FREE RF Modulator plus FREE Intel 8085 user's manual plus FREE postage & handling!

Engineering Pak (SAVE $41.00)—Buy Levels "A," "B," "C," "D," and "E" with Power Supply, ASCII Keyboard/Computer Terminal, and six S-100 Bus Connectors for **$514.75** and get 10 FREE computer grade cassette tapes plus FREE 8085 user's manual plus FREE postage & handling!

Business Pak (SAVE $89.95)—Buy Explorer/85 Levels "A," "B," and "C" (with cabinet), Power Supply, ASCII Keyboard/Computer Terminal (with cabinet), 16k RAM, 12" Video Monitor, North Star 5-1/4" Disk Drive (includes North Star BASIC) with power supply and cabinet, all for just **$1599.40** and get 10 FREE 5-1/4" minidiskettes ($49.95 value) plus FREE 8085 user's manual plus FREE postage & handling!

Continental U.S.A. Credit Card Buyers Outside Connecticut

CALL TOLL FREE 800-243-7428

To Order From Connecticut Or For Technical Assistance, Etc. Call (203) 354-9375

Netronics R&D Ltd., Dept. PE-5
333 Litchfield Road, New Milford, 06776

Please send the items checked below—

- ☐ **Explorer/85 Level "A" Kit** (ASCII Version), **$129.95** plus $3 p&h.
- ☐ **Explorer/85 Level "A" Kit** (Hex Version), **$129.95** plus $3 p&h.
- ☐ **8k Microsoft BASIC** on cassette tape, **$64.95** postpaid.
- ☐ **8k Microsoft BASIC in ROM Kit** (requires Levels "B," "D," and "E"), **$99.95** plus $2 p&h.
- ☐ **Level "B" (S-100) Kit, $49.95** plus $2 p&h.
- ☐ **Level "C" (S-100 6-card expander) Kit, $39.95** plus $2 p&h.
- ☐ **Level "D" (4k RAM) Kit, $69.95** plus $2 p&h.
- ☐ **Level "E" (EPROM/ROM) Kit, $5.95** plus 50¢ p&h.
- ☐ **Deluxe Steel Cabinet** for Explorer/85, **$49.95** plus $3 p&h.
- ☐ **ASCII Keyboard/Computer Terminal Kit** (features a full 128 character set, upper & lower case, full cursor control, 75 ohm video output convertible to baudot output, selectable baud rate, RS232-C or 20 ma. I/O, 32 or 64 character by 16 line formats, and can be used with either a CRT monitor or a TV set (if you have an RF modulator), **$149.95** plus $2.50 p&h.
- ☐ **Hex Keypad/Display Kit, $69.95** plus $2 p&h.
- ☐ **Deluxe Steel Cabinet** for ASCII Keyboard/Terminal, **$19.95** plus $2.50 p&h.
- ☐ **Power Supply Kit** (±8V @ 5 amps) in deluxe steel cabinet, **$39.95** plus $2 p&h.
- ☐ **Gold Plated S-100 Bus Connectors, $4.85** each, postpaid.
- ☐ **RF Modulator Kit** (allows you to use your TV set as a monitor), **$8.95** postpaid.
- ☐ **16k RAM Kit** (S-100 Board expands to 64k), **$199.95** plus $2 p&h.
- ☐ **32k RAM Kit, $329.95** plus $2 p&h.
- ☐ **48K RAM Kit, $459.95** plus $2 p&h.
- ☐ **64k RAM Kit, $589.95** plus $2 p&h.
- ☐ **16k RAM Expansion Kit** (to expand any of the above up to 64k), **$139.95** plus $2 p&h each.
- ☐ **Intel 8085 cpu User's Manual, $7.50** postpaid.
- ☐ **Special Computer Grade Cassette Tapes, $1.90** each or **3 for $5**, postpaid.
- ☐ **12" Video Monitor** (10 MHz bandwidth), **$139.95** plus $5 p&h.
- ☐ **North Star Double Density Floppy Disk Kit** (One Drive) for Explorer/85 (includes 3 drive S-100 controller, DOS, and extended BASIC with personalized disk operating system—just plug it in and you're up and running!), **$699.95** plus $5 p&h.
- ☐ **Power Supply Kit** for North Star Disk Drive, **$39.95** plus $2 p&h.
- ☐ **Deluxe Case** for North Star Disk Drive, **$39.95** plus $2 p&h.
- ☐ **Experimenter's Pak** (see above), **$199.90** postpaid.
- ☐ **Student Pak** (see above), **$319.85** postpaid.
- ☐ **Engineering Pak** (see above), **$514.75** postpaid.
- ☐ **Business Pak** (see above), **$1599.40** postpaid.

Total Enclosed $ _____
(Conn. res. add sales tax) By—
☐ Personal Check ☐ M.O./Cashier's Check ☐ Visa ☐ Master Charge
(Bank # _ _ _ _)

Acct. # _____

Signature _____ Exp. Date ____

Print Name _____

Address _____

City _____

State _____ Zip _____

☐ Send Me Information

By Netronics

ASCII/BAUDOT, STAND ALONE

Computer Terminal

COMPLETE FOR ONLY . . . $149.95

The Netronics ASCII/BAUDOT Computer Terminal Kit is a microprocessor-controlled, stand alone keyboard/terminal requring no computer memory or software. It allows the use of either a 64 or 32 character by 16 line professional display format with selectable baud rate, RS232-C or 20 ma. output, full cursor control and 75 ohm composite video output.

The keyboard follows the standard typewriter configuration and generates the entire 128 character ASCII upper/lower case set with 96 printable characters. Features include onboard regulators, selectable parity, shift lock key, alpha lock jumper, a drive capability of one TTY load, and the ability to mate directly with almost any computer, including the new Explorer/85 and ELF products by Netronics.

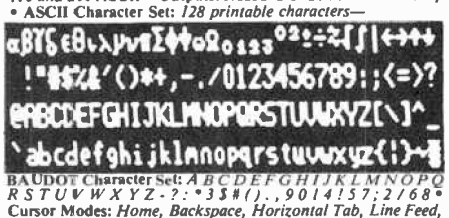
The Computer Terminal requires no I/O mapping and includes 1k of memory, character generator, 2 key rollover, processor controlled cursor control, parallel ASCII/BAUDOT to serial conversion and serial to video processing—fully crystal controlled for superb accuracy. PC boards are the highest quality glass epoxy for the ultimate in reliability and long life.

VIDEO DISPLAY SPECIFICATIONS

The heart of the Netronics Computer Terminal is the microprocessor-controlled Netronics Video Display Board (VID) which allows the terminal to utilize either a parallel ASCII or BAUDOT signal source. The VID converts the parallel data to serial data which is then formatted to either RS232-C or 20 ma. current loop output, which can be connected to the serial I/O on your computer or other interface, i.e., Modem.

When connected to a computer, the computer must echo the character received. This data is received by the VID which processes the information, converting to data to video suitable to be displayed on a TV set (using an RF modulator) or on a video monitor. The VID generates the cursor, horizontal and vertical sync pulses and performs the housekeeping relative to which character and where it is to be displayed on the screen.

Video Output: *1.5 P/P into 75 ohm (EIA RS-170)* • **Baud Rate:** *110 and 300 ASCII* • **Outputs:** *RS232-C or 20 ma. current loop* • **ASCII Character Set:** *128 printable characters—*



BAUDOT Character Set: *A B C D E F G H I J K L M N O P Q R S T U V W X Y Z - ? : • 3 $ # ( ) . , 9 0 1 4 ! 5 7 ; 2 / 6 8 •*

Cursor Modes: *Home, Backspace, Horizontal Tab, Line Feed, Vertical Tab, Carriage Return. Two special cursor sequences are provided for absolute and relative X-Y cursor addressing* • **Cursor Control:** *Erase, End of Line, Erase of Screen, Form Feed, Delete* • **Monitor Operation:** *50 or 60Hz (jumper selectable.*

Continental U.S.A. Credit Card Buyers Outside Connecticut

CALL TOLL FREE 800-243-7428

To Order From Connecticut Or For Technical Assistance, Etc. Call (203) 354-9375

Netronics R&D Ltd., Dept. PE-5
333 Litchfield Road, New Milford, CT 06776

Please send the items checked below—

- ☐ **Netronics Stand Alone ASCII Keyboard/Computer Terminal Kit, $149.95** plus $3.00 postage & handling.
- ☐ **Deluxe Steel Cabinet** for Netronics Keyboard/Terminal In Blue/Black Finish, **$19.95** plus $2.50 postage and handling.
- ☐ **Video Display Board Kit** alone (less keyboard), **$89.95** plus $3 postage & handling.
- ☐ **12" Video Monitor** (10 MHz bandwidth) fully assembled and tested, **$139.95** plus $5 postage and handling.
- ☐ **RF Modulator Kit** (to use your TV set for a monitor), **$8.95** postpaid.
- ☐ **5 amp Power Supply Kit** In Deluxe Steel Cabinet (±8VDC @ 5 amps, plus 6-8 VAC), **$39.95** plus $2 postage & handling.

Total Enclosed (Conn. res. add sales tax) $_____
By—
☐ Personal Check ☐ Cashiers Check/Money Order
☐ Visa ☐ Master Charge (Bank # _ _ _ _)

Acct. # _____

Signature _____ Exp. Date _____

Print Name _____

Address _____

City _____

State _____ Zip _____

☐ Send Me More Information

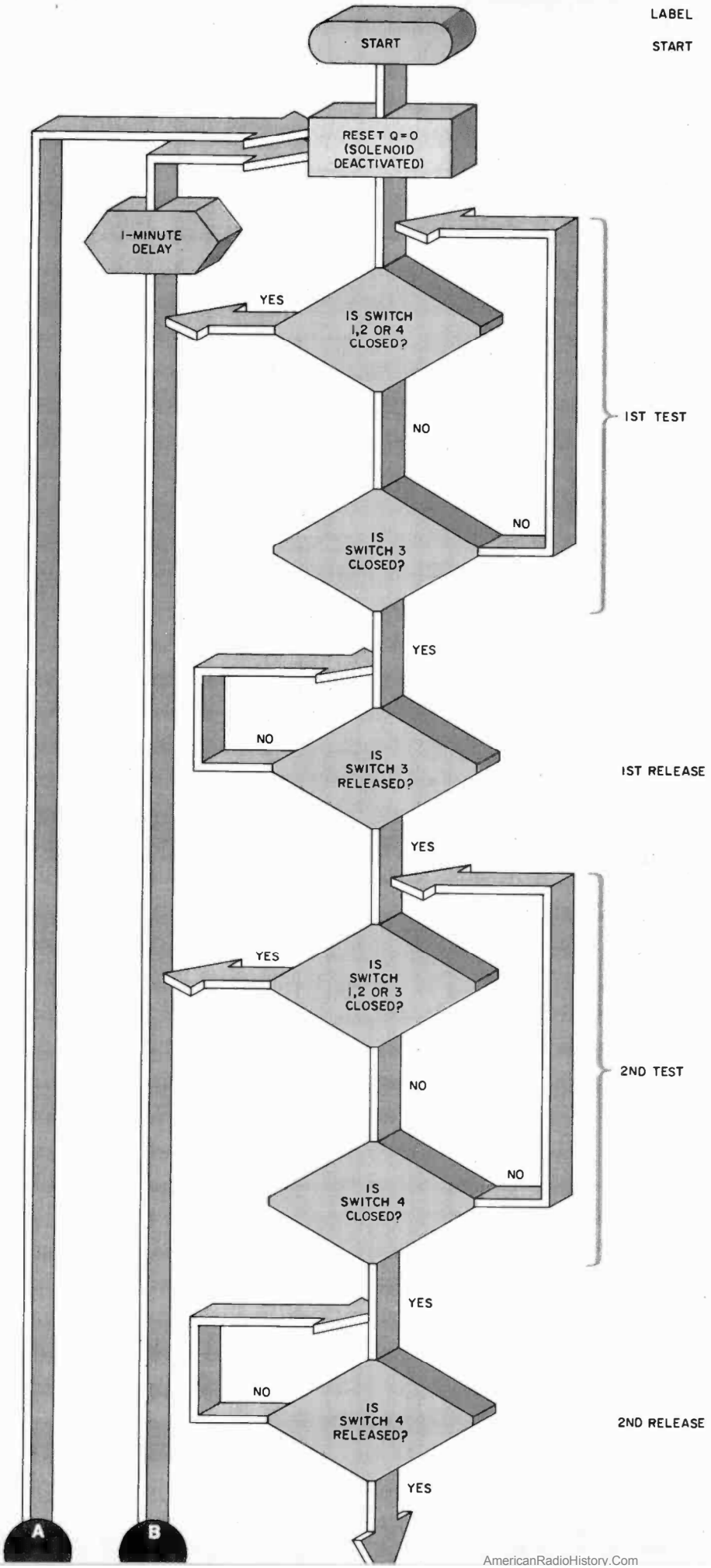


Fig. 3. Flowchart, at left and on opposite page, gives steps the microprocessor must perform.

the beginning address of that set of instructions.

The third column, OP CODE, represents the hexadecimal machine code for the instruction plus any extra byte required to modfy the instruction. In most cases, the extra byte is a Branch address and is filled in after the program has been completed. The fourth column, MNEMONIC, contains the word-like version of the op code. This column, once you get used to it, makes reading a program considerably easier.

The column marked COMMENTS is strictly for the human being. At some later date, when you return to the program, reading this column will tell you in detail what is supposed to happen at each step. Trying to figure out what the program does without referring to the comments or the flowchart is difficult.

When first powered up, the processor fetches its first instruction at memory address 0000.

The label "Start" should use program address 0000. Since we have decided that, at the beginning, the solenoid should be de-activated, we use the mnemonic instruction REQ (reset Q) having the op code of 7A on the start line. The comments column then explains this action and the result.

The next step is a decision. In the flowchart, the three wrong switches were tested in one box. In the program, we force the processor to test each switch in turn for an open or closed condition. Like the flowchart, we label this 1st Test, and use the next memory address 0001. Switch 1 is tested by the mnemonic B1 having op code 34. This op code checks the status of switch 1 (actually the EF line associated with it). It is a two-part op code that requires a branch address if the switch was depressed. Since we don't know the address of the 1-minute time delay at this time, we temporarily leave the required second byte of the op code. blank. Therefore, this line of the op code column is 34__.

Since two bytes were used at address 0001, the next memory address is 0003. Here the program op code queries the status of switch 2 via op code

35 (B2). Thus program address 0003 contains 35__. Keep in mind that, as each line is created, sufficient detail must be inserted in the COMMENTS column to explain what is going on.

Since two bytes were used as address 0003, the next address is 0005. Here, op code 37 (B4) questions the status of switch 4. Since this is still the wrong switch, we must branch to the 1-minute delay. Thus, address 0005 contains 37__.

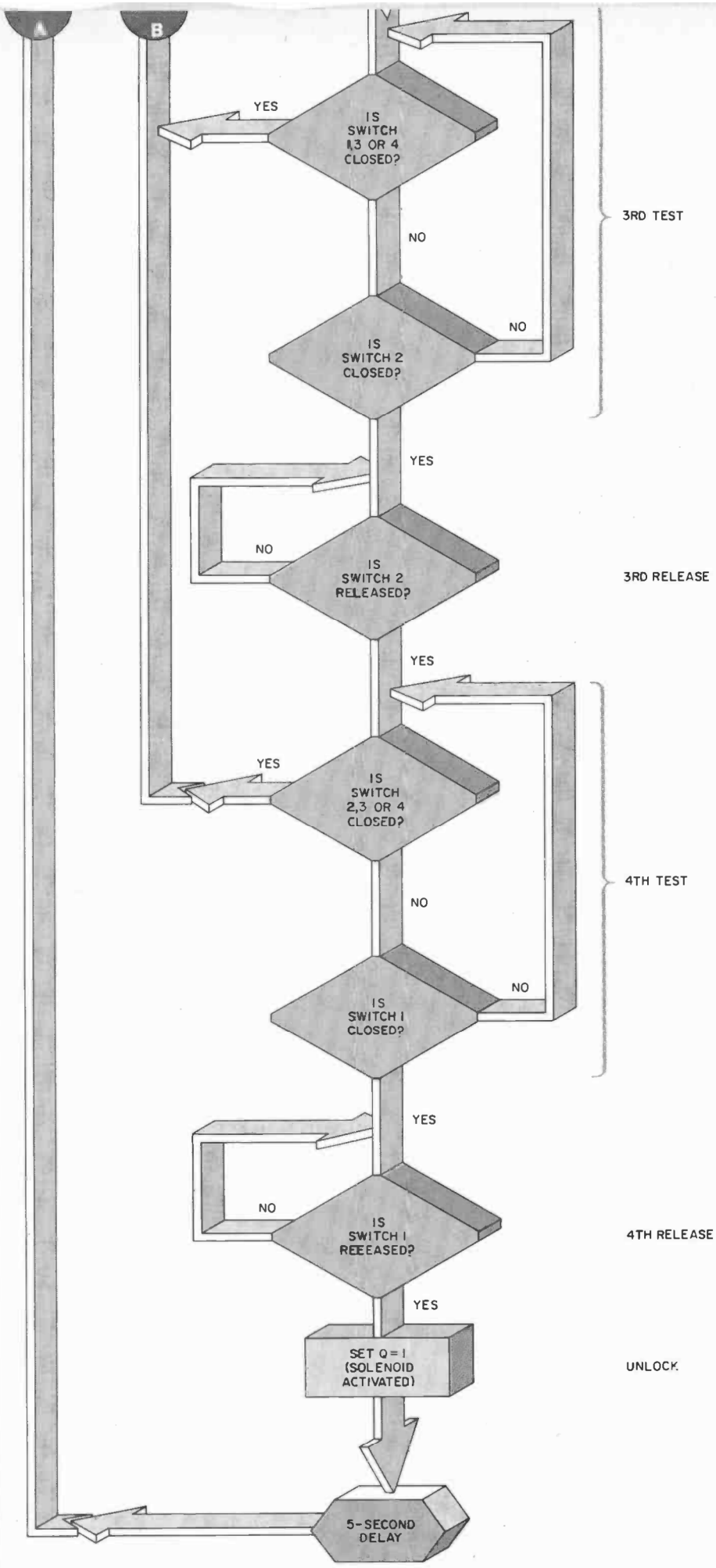
Now we are left with only one switch (switch 3, the correct one). Address 0007 checks this switch via op code 36. If this switch was depressed, the program branches to the 1st Release, whose address is as yet unknown. Thus, address 0007 contains 36__.

Now, what happens if *no* switches were touched? This is the purpose of program address 0009. If the flow gets to this point without branching, address 0009 forces the program to return to the 1st Test, which we now know is located at program address 0001. Now you see the value of the label. Since we are working with only one page of memory (256 bytes), the leading two zeroes of the address are not required, so address 0009 contains BR (branch immediately) to program address 01. Thus, if during the 1st Test no switches were operated, the program "loops" around this section, awaiting switch action.

Since we have determined that switch 3 was depressed at address 0007, we now perform the 1st Release at address 000B. The 3E instruction at address 000B says that, if the switch was released, branch the program to 0F, the start of the 2nd Test. If the switch is still depressed, the instructions at 000D force the program to return (30, Branch Immediate) back to memory address 0B, and await switch release.

The 2nd Test, switch 2 Release, 3rd Test, switch 3 Release, 4th Test and switch 4 Release, operate just as did the 1st Test and switch 1 Release. The reader can follow the program flow to make sure that the four switches must be depressed in proper sequence before the program arrives at the Unlock block at program address 0039.

The Unlock statement is one-byte instruction 7B that causes the 1802 to set its Q-output line to the high state. Since the solenoid is connected to this



output, when this instruction is carried out by the processor, the solenoid becomes activated and allows the door to be opened.

As soon as the op code at program address 0039 is executed, the flow passes to the next block having the label 5-Sec. Delay located at program address 003A.

To create a delay, we can take advantage of the fact that it takes time for the processor to execute an instruction. Therefore, we can give it a series of "busy work" instructions to let it waste the required time. Such maneuvers should of course produce no other external effects. In this case, data is passed back and forth between internal registers of the 1802 in a particular sequence. It is not really important which registers are used, but the instruction set allows us to "play" with the D register more easily than with some others.

The delay starts at program address 003A by loading FF (1111 1111) into the D register. The next step uses instruction A1 (address 003C) to load the FF from the D register into the low-order half of register-1. Then, according to address 003D, the FF is also loaded into high-order half register-1 using the B1 instruction. Next, the program contains four NOPs (No Operations), instructions during which the processor does absolutely nothing but waste clock cycles. The 1802 uses the op code C4 for NOP, each wastes 10-11 microseconds (using a 1.71-MHz timing reference oscillator or "clock"). The four instructions at address 003E "pad" the time delay so that the loop will come to five seconds.

As we now know, register-1 contains 1111 1111 (FF). Address 0042 uses op code 21 to decrement (reduce by one) the contents of register-1. After the first cycle, low-order register-1 contains 1111 1110 (FE). Address 0043 loads the high-order byte of register-1 into the D register using instruction 91. We know that the D register will contain 1111 1111 (FF). The 3A instruction at address 0044 checks the contents of the D register. If the D register is not all zeros (and we know it isn't), the program branches back to address 3E and continues decrementing the contents of register-1, moving the high-order byte into the D register, and checking the contents of the D register for all zeros, etc.

After 65,280 passes through the "loop," which should total five seconds, the contents of the D register will be 0000 0000. When this occurs, the program moves on to address 0046 and finds a BR (30) that goes back to the Start at address 00. At this point, the Q line is forced low and the solenoid is de-activated. The program then pro-

TABLE II—PROGRAM

LABEL	PROGRAM ADDRESS	OP CODE	MNEMONIC	COMMENTS
Start	0000	7A	REQ	Make Q output low (door locked)
1st Test	0001	34 48	B1	If switch #1 depressed, branch to 1-min. delay, else next instruction
	0003	35 48	B2	If switch #2 depressed, branch to 1-min. delay, else next instruction
	0005	37 48	B4	If switch #4 depressed, branch to 1-min. delay, else next instruction
	0007	36 0B	B3	If switch #3 depressed, branch to #1 Release, else next instruction
	0009	30 01	BR	If no switches depressed, branch to 1st Test.
1st Release	000B	3E 0F	BN3	If switch released, branch to 2nd Test, else next instruction
	000D	30 0B	BR	If switch not released, branch to 1st Release
2nd Test	000F	34 48	B1	If switch #1 depressed, branch to 1-min. delay, else next instruction
	0011	35 48	B2	If switch #2 depressed, branch to 1-min. delay, else next instruction
	0013	36 48	B3	If switch #3 depressed, branch to 1-min. delay, else next instruction
	0015	37 19	B4	switch #4 depressed, branch to #2 Release, else next instruction
	0017	30 0F	BR	No switches depressed, branch to 2nd Test
2nd Release	0019	3F 1D	BN4	If switch released, branch to 3rd Test, else next instruction
	001B	30 19	BR	If switch not released, branch to #2 Release
3rd Test	001D	34 48	B1	If switch #1 depressed, branch to 1-min. delay, else next instruction
	001F	36 48	B3	If switch #3 depressed, branch to 1-min. delay, else next instruction
	0021	37 48	B4	If switch #4 depressed, branch to 1-min. delay, else next instruction
	0023	35 27	B2	If switch #2 depressed, branch to #3 Release, else next instruction
	0025	30 1D	BR	If no switches depressed, branch to 3rd Test
3rd Release	0027	3D 2B	BN2	If switch released, branch to 4th Test, else next instruction
	0029	30 27	BR	If switch not released, branch to #3 Release
4th Test	002B	35 48	B2	If switch #2 depressed, branch to 1-min. delay, else next instruction
	002D	36 48	B3	If switch #3 depressed, branch to 1-min. delay, else next instruction
	002F	37 48	B4	If switch #4 depressed, branch to 1-min. delay, else next instruction
	0031	34 35	B1	If switch #1 depressed, branch to #4 Release, else next instruction
	0033	30 2B	BR	If no switches depressed, branch to 4th Test
4th Release	0035	3C 39	BN1	If switch released, branch to Unlock, else next instruction
	0037	30 35	BR	If switch not released, branch to #4 Release

ceeds to the 1st Test, and loops around it, awaiting further switch action.

The one-minute delay invoked when a wrong switch is depressed starts at program address 0048. It works by running through the 5-Sec. Delay 12 times under control of another outside loop. This action requires 5 × 12 or 60 seconds.

LABEL	PROGRAM ADDRESS	OP CODE	MNEMONIC	COMMENTS
Unlock	0039	7B	SEQ	Set Q output high (unlock door), go to next instruction
5-Sec. Delay	003A	F8 FF	LDI	Load FF (1111 1111) into the D register
	003C	A1	PLO	Loads D register into low-order half of register-1
	003D	B1	PHI	Loads D register into high-order half of register-1.
Timer	003E	C4 C4	NOP	Waste time to increase delay
	0040	C4 C4	NOP	
	0042	21	DEC	Decrement (reduce) the contents of register-1 by 1
	0043	91	GHI	Load the high-order byte of register-1 into the D register
	0044	3A 3E	BNZ	If the content of the D register is not zero, branch to Timer and continue decrementing register-1 and testing the D register. When the D register contains zero, go to next instruction.
	0046	30 00	BR	End of 5-sec. time delay, branch to Start.
1-Min. Delay	0048	F8 0C	LD1	Load the D register with 0C (0000 1100)
	004A	A2	PLO	Load 0C from the D register into the low-order half of register-2
Loop 1	004B	F8 FF	LDI	Load the D register with FF (1111 1111)
	004D	A1	PLO	Load FF from the D register into the low-order half of register-1
	004E	B1	PHI	Load FF from the D register into the high-order half of register-1
Timer Start	004F	C4 C4	NOP	Waste time to pad out the delay
	0051	C4 C4		
	0053	21	DEC	Decrement the contents of register-1 by 1
	0054	91	PHI	Load the high-order byte of register-1 into the D register
	0055	3A 4F	BNZ	If the content of the D register is not zero, branch to Timer Start. Continue decrementing register-1 and testing the D register. When the D register contains zero, next instruction
Loop 2	0057	22	DEC	Decrement the contents of register-2 by 1. (Register-2 contains 1100 from address 004A)
	0058	82	PLO	Load the contents of register-2 into the D register
	0059	3A 4B	BNZ	If the content of the D register is not zero, branch to Loop 1. Continue decrementing, then testing the D register. When the D register contains zero, next instruction
	005B	30 00	BR	End of 1-min. delay, branch to Start. This sets the Q output low to deactivate the solenoid and lock the door. The program then awaits further switch operations.

The 12-times loop starts at address 0048 by loading the D register (F8) with 0C (0000 1100) which is decimal 12. Address 004A loads this byte into register-2 (A2).

The program from address 004B to 0056 is the five-second delay as previously described. In this case, when the D register contains all zeros, the program drops to address 0057 that decrements the contents of register-2 by one (it now contains 1011 or decimal 11).

The next two instructions (address 0058 and 0059) are the remainder of the "outside" loop that decrements register-2 each time the five-second delay is executed. After the twelfth pass, register-2 will contain zero. When this occurs, the program advances to address 005B, where it is told to branch (30) back to the Start (00).

Since we now know the address of the 1-Minute Delay (0048), we can go back into program and substitute 48 for each of the blanks (depicted as __) that were used where the program had to branch to the one-minute delay.

Modifications. The basic program is easily modified even by an inexperienced programmer. For example, at each switch Release segment (identified by its program label), you might as an exercise write a small program that requires, say, that the switch be released within two seconds, otherwise the program branches to the 1-minute time delay. Such a new set of instructions can reside above program address 005C (the end of the present program) and can be invoked (called) from the switch release segment. Some branch statements can be used. If you require a longer combination, then enlarge the program accordingly. Note that you can change the combination easily by modifying the pertinent instruction in each switch Test location.

In Conclusion. We have seen how powerful an element a microprocessor can be. To realize even the simple example presented above in single-function logic gates would be impractical. Implementing the actions we wanted from the microprocessor was a task of but modest difficulty by comparison.

Succeeding articles in this series will extend the uses of microprocessors further. We will also discuss the hardware needed in more detail. ◇