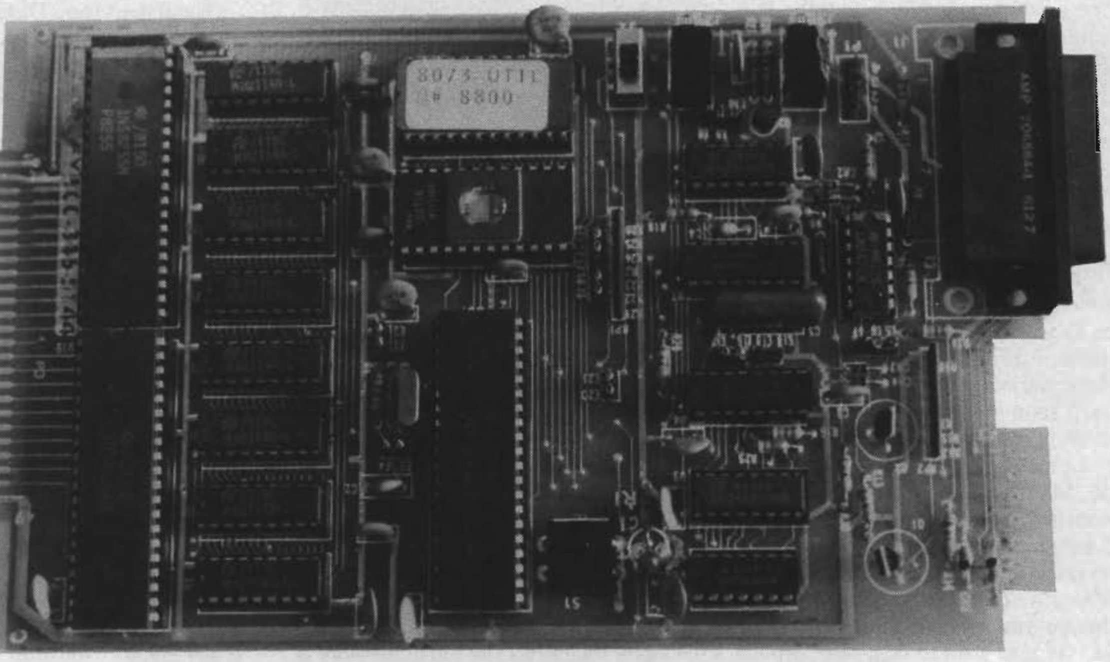

How to Design Microprocessor-based Projects

Make your next project "intelligent." Use a single-chip microcomputer to control it.

TOM FOX

Part 2 LAST MONTH, WE INtroduced National Semiconductor's INS8073 microinterpreter and described, in detail, its pin connections and their functions. In this second part of the series we will look at a handy demo/development board as well as introduce the language of the 8073, NSC Tiny BASIC.

Before we get started with that, we should mention that National Semiconductor has, unfortunately, discontinued production of the INS8073. However, many suppliers still have stocks of that IC, so you should be able to find it rather easily. Even if you can't, don't worry. There are many other single-IC computers available. You may even learn more if you try to use another microinterpreter.

Even if you don't want to pore over data sheets and design interfacing circuits, you can still put the power of a microinterpreter to work by using development and demonstration boards. Several manufacturers offer such boards. For example, we'll look at a demonstration board for the INS8073 that is available from Digi-Key Corp. (Highway 32 South, PO Box 677, Thief River Falls, MN 56701). That board, shown in Fig. 5, sells for $250. A user manual is $10.

The demonstration board makes designing with the 8073 so simple it seems as if you're cheating! After you receive the board, all you have to do to start programming in NSC Tiny BASIC is to connect up power (+5 and −12 volts). If you use an RS-232 terminal, you have to add a jumper to the board, and use a standard male-male cable to connect the D-type connector on the board (J1) to your terminal's RS-232 connector. Of course, you can also use your computer along with communications software to emulate a smart terminal. Baud rates (110, 300, 1200 4800 baud) can be set by adding jumpers on the board.

After you turn on the power, a prompt (>) should appear on your terminal screen. You are then ready to start programming in Tiny BASIC.

Besides being easy to hook up and get running, the development board contains 4K of RAM, a 8255 programmable peripheral interface IC, an 8154 RAM-I/0 that provides 128 bytes of scratch-pad memory for use in assembly-language subroutines, and has 16 I/0 lines that can be individually programmed.

Perhaps the nicest features of the board is that it has an EPROM-resident utility program as well as an EPROM programmer (which is controlled by the utility program). The utility program allows you to store and retrieve programs on cassette tapes and download programs from a "host" computer. It also has a variety of other features such as the DUMP command, which allows you to "display" part of memory in both the hexadecimal and converted ASCII format. However, for our purposes (the design of smart machines), its greatest feature is that it makes the programming of EPROM's easy and nearly foolproof. All you need is a singlesupply 2716 EPROM and a +25-volt source. The built-in program does all the rest. One caution: the S (setup) command is not mentioned in the manual, but it should be used immediately before the P (program) command.

While the demonstration board is definitely helpful if you want to design circuits using the 8073, it is not essential if you have access to an EPROM programmer. However, if you don't have an EPROM programmer available, then the board is recommended.

NSC Tiny BASIC

Like most tiny BASIC's, that supplied with the INS8073 allows only signed integers from −32767 to +32767 and 26 single-letter variables (A to Z). However, as you will soon see, this Tiny BASIC is remarkable in most other respects.

The commands RUN, CONT, CLEAR, LIST, and NEW are similar to the commands used by many standard BASIC languages. The command NEW<expr> resets the program pointer to that of the <expr>. That command should be given before you enter a program. For instance, say you want to start your program at 1100H. (The "H" indicates that the number is in hexadecimal). You should type: NEW#1100<return> and then type: NEW<return>. NOTE: In NSC Tiny BASIC "#" is used as a prefix to indicate to the interpreter that the number is in hexadecimal. Numbers that do not have that prefix will be treated by the interpreter as if they are decimal.

The following NSC Tiny BASIC statements and operators are similar to those of other BASIC's and will be listed without comment: REM, LET, PRINT, IF–THEN, FOR–NEXT, GOTO, GOSUB, RETURN, INPUT, STOP, +, −, ×, /, >, <, =, AND, OR, NOT, and MOD.

The DO–UNTIL, statement is used to program loops and has been borrowed from PASCAL. That statement is rare in the BASIC language, although the proposed ANSI standard BASIC contains it.

The LINK statement is used to transfer control to machine language subroutines, which can be used where speed is important.

The DELAY statement causes a delay in the program execution up to a maximum of just over 1 second. The number following the DELAY statement gives the time in milliseconds that the program will be delayed. DELAY 100 causes a delay of about 100 milliseconds (1/10 of a second) and DELAY 0 gives the maximum delay of about 1040 milliseconds. (Note all times are only approximate and depend on the frequency of the clock.) The DELAY command is used extensively in our example project.

The function RND (random) is also used frequently in our example project.

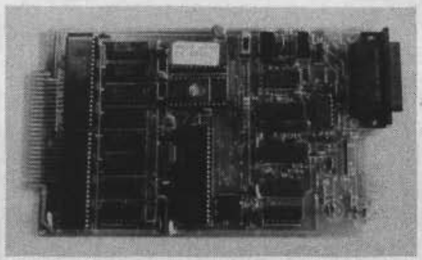
FIG. 5—THIS DEMO/DEVELOPMENT BOARD is not necessary if you want to use the 8073, but its on-board EPROM programmer and utility program makes it easy.

The statement LET A=RND(1,10) assigns a pseudo-random number between 1 and 10 (inclusive) to the variable A.

Let's take a look at a short program segment that will give us a delay in program execution between 1 and 5 seconds:

```
310 LET L=0
320 LET M=RND(1,5)
330 DELAY Ø
340 LET L=L+M
350 IF L<5 GOTO 330
```

The TOP function gives the address of the top of memory. (The top of memory is the first memory location in external RAM that is not used by the NSC Tiny-BASIC program.)

The following is an example of how the TOP function is used. (Note that this example is printed here exactly as it would appear on your terminal.)

```
>NEW#1100
>NEW
>10 PRINT TOP
>RUN
4366
```

The number displayed on the terminal after the program has run (4366) is the first address of unused RAM. Note that like all numbers displayed on the terminal under the control of the INS8073 microinterpreter, 4366 is in decimal. While the interpreter "understands" numbers given to it in hexadecimal (with a "#" prefix), it only displays decimal numbers on the terminal.

The STAT function allows you to monitor and set the 8073 status register. For details on that register, refer to the *National Semiconductor Tiny BASIC User Manual* and/or the INS8070 data sheet.

The ON statement is used for processing interrupts. It has two formats; ON 1 <expr> and ON 2 <expr>. Notice there are two interrupts, 1 and 2. Number 1 interrupt occurs when there is a high-to-low transition at pin 38. Interrupt number 2 occurs when there is a high-to-low transition at pin 39. After the 8073 senses the interrupt, the interpreter will execute a GOSUB beginning at the line number given by <expr>. It is important to note that the statement, STAT="odd number" (where the odd number is often 1), should be used before an ON statement in order to enable the interrupts.

The INC(X) and DEC(X) functions increment and decrement, respectively, the memory at the locations given by X. Those functions are useful in multiprocessing. For our purposes, however, we will ignore them.

The indirect operator

One of the most important features of NSC Tiny BASIC, especially for those who look to the 8073 to provide "brains" for "brainless" machines, is the indirect operator "@." That operator is simpler to use than the PEEK and POKE commands found in most BASIC languages. For instance, to store a 4 (decimal) at location C000H, simply make the following statement; @#C000=4. The statement "LET B=@#6000" sets the value of the variable B equal to the value stored at the hexadecimal address 6000.

Be aware that the INS8073 is an 8-bit device and it can only store an 8-bit binary number (0 to 255 in the decimal system) at any memory location. Thus, if you use the @ operator to attempt to store a two-byte (16 bit) number or larger at any memory location, you might experience sleepless nights trying to debug your program. The interpreter will only "see" the least-significant byte. For instance, the statement "@#C000=260" (260=100000100) would actually store "4" (00000100) at location C000.

Putting the microinterpreter to work

Now that we've introduced you to the INS8073 microinterpreter and taken a quick look at a demo/development board and the NSC Tiny BASIC language, it's time to design a project that is based on the 8073. In particular, we will describe, in detail, how to build a "burglar outwitter."

The first step in the design of any project should be to decide exactly what you want the project to do. As the name of this project suggests, we would like to fool or outwit a burglar into thinking that someone is in the house, even when it is empty. The circuit we'll describe is fairly simple—it's meant simply to be a demonstration of what you can do with the INS8073. If you decide to build such a device, you'll learn a lot more if you customize the circuit and/or program to better meet your needs. We'll give you some hints on how to do that.

The burglar outwitter

We want our demonstration circuit to control three lights: The living-room light (called light A), the hallway light (light B) and the bathroom light (light C). When it gets dark, we want light A to go on for random length of time between 10 and 60 minutes. We want light B to go on about a second after light A goes out. Light B will stay on for 1 to 5 seconds. About a second after light B goes out, we would like light C to light for a random length of time that is no greater than 20 minutes but no less than 5 minutes. About a second after this light goes out light B should go on and stay lit for 1 to 5 seconds. After a second pause let's have light A go on again, but only if it is still dark.

That completes the first cycle. After 5 such cycles, we would like the controller to go into a "sleep" phase for the rest of the night. During that "sleep" phase, light C (the bathroom light) will go on every 1½ to 5 hours for between 1 and 10 minutes. That should continue until sunrise, when the controller will be reset and prepared for the next night.

If possible, we would like to use a 2716 EPROM to store the software. And we would like to design the system so that it is easy to expand. After all, we'll probably want to add another light—or maybe a dozen appliances.

Now that we know what we want our controller to do, we have to design an actual circuit to do the job. First, however, let's look at some input and output details. Let's pick address C000H for both our output and input port address. Note that C000H is equal to 1100 0000 0000 0000 in binary notation. To decode that address, address lines A15 and A14 (the two most significant bits) must be high while A13 and A12 are low. (We don't care about the other address lines here since there is no memory at those locations.)

The NSC Tiny BASIC statement LET B=@#C000, assigns the value at address C000 to the variable B. (Remember the prefix "#" stands for "hexadecimal" in NSC Tiny BASIC.) We have to design a circuit that places information on dark/light at input-port location C000. When its dark, let's make DØ "one" (high) and when its light, let's have DØ "zero" (low).

For the output port, we will use the same address, C000. We can do that because the 8073 has READ and WRITE control lines that tell external circuits (such as address decoders) whether the 8073 is inputting information (reading) or outputting information (writing). TTL latches will be used to temporarily store output information. We'll use one latch for each light controlled.

The output circuit will be designed so that the statement @#C000=1 will set latch 1. Also the statement @#C000=2 will set latch 2 and @#C000=4 will set latch 3. All latches will be reset by the statement @#C000=0. (Note that those statements work because a decimal 1 = 00000001 (binary), 2=00000010 (binary) and 4=00000100 (binary).

A buffer/driver/relay circuit will be connected to the latches' output to actually control the lights. But we're getting a little ahead of ourselves. We'll go into specific detail on how to wire real-life parts. But first, let's look at a program that will accomplish our objectives.

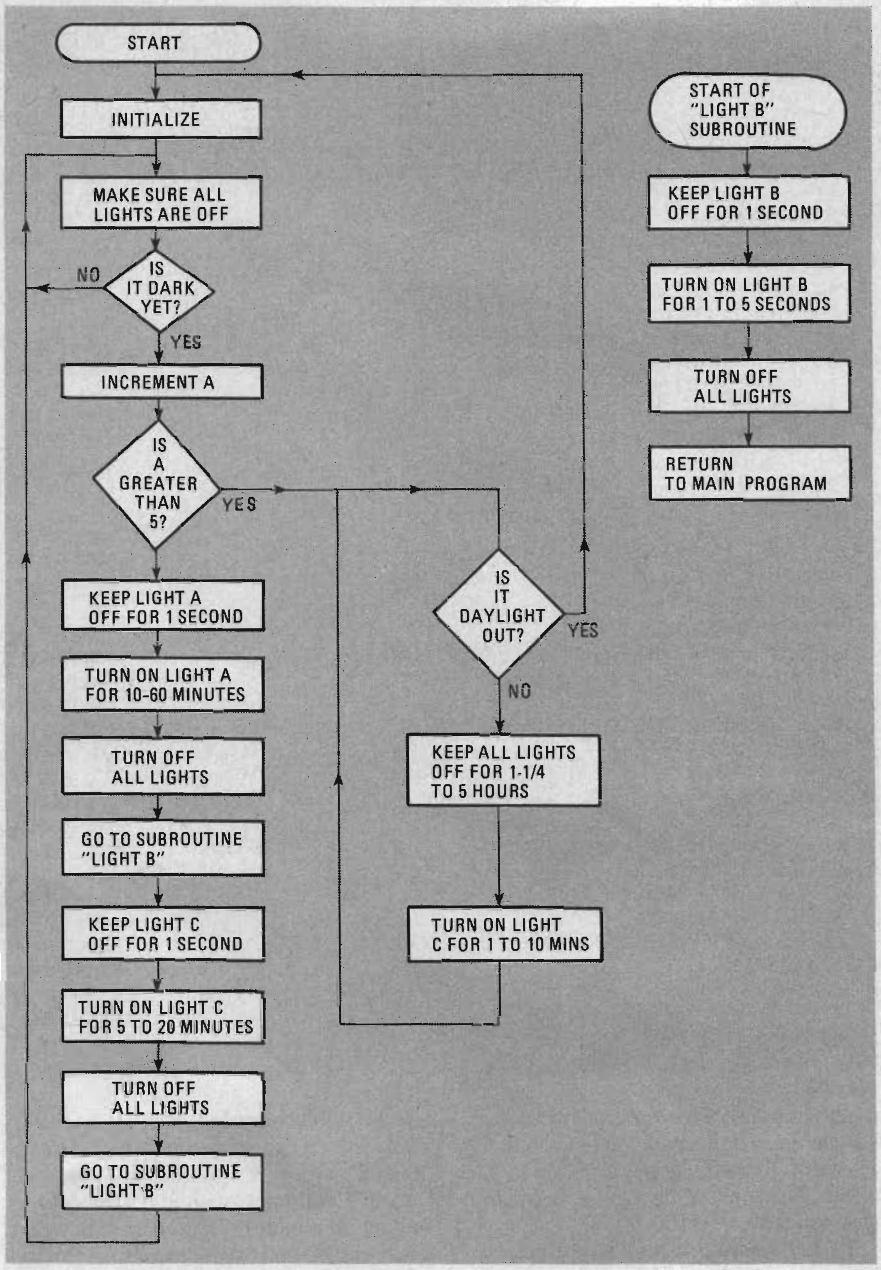
START
INITIALIZE
MAKE SURE ALL LIGHTS ARE OFF
IS IT DARK YET?
NO
YES
INCREMENT A
IS A GREATER THAN 5?
YES
KEEP LIGHT A OFF FOR 1 SECOND
TURN ON LIGHT A FOR 10-60 MINUTES
TURN OFF ALL LIGHTS
GO TO SUBROUTINE "LIGHT B"
KEEP LIGHT C OFF FOR 1 SECOND
TURN ON LIGHT C FOR 5 TO 20 MINUTES
TURN OFF ALL LIGHTS
GO TO SUBROUTINE "LIGHT B"
IS IT DAYLIGHT OUT?
YES
NO
KEEP ALL LIGHTS OFF FOR 1-1/4 TO 5 HOURS
TURN ON LIGHT C FOR 1 TO 10 MINS
START OF "LIGHT B" SUBROUTINE
KEEP LIGHT B OFF FOR 1 SECOND
TURN ON LIGHT B FOR 1 TO 5 SECONDS
TURN OFF ALL LIGHTS
RETURN TO MAIN PROGRAM

FIG. 6—THE FLOW CHART FOR the burglar outwitter. The actual program could have been written more compactly, but then it wouldn't have been as valuable a learning tool.

The control program

The first step in writing any non-trivial program should be the sketching out of a flow chart. Figure 6 shows a simplified flow chart for our BASIC program. Although this project has a definite practical application, it also provides us with the opportunity to explain the basics of designing with the INS8073. You may find that the program (and circuit itself) was not designed to be especially compact. The primary objective is to have something that's easy to understand. You might like to write a shorter and more elegant program that accomplishes the same task.

The wordy program shown in Table 1 is allowable here since there is plenty of memory space in the 2716 EPROM. Longer, more complicated programs may not fit in a single 2716.

While we could go through the program line-by-line to explain its operation, it is probably less confusing to study the flowchart. There are some lines that we should describe, however.

The program statement in line 20 turns off all lights, while the statement in line 30 places information on dark/light into variable B. In line 40, all data bits are masked except the least-significant bit of data (DØ). The variable "C" has the value of 1 when it's dark and 0 when light.

When it's dark, the statement on line 60

TABLE 1
BURGLAR OUTWITTER PROGRAM

```
  1 REM GHOSTLY BURGLAR OUTWITTER PROGRAM
  5 REM SET ALL VARIABLES TO ZERO AND INITIALIZE
 10 CLEAR
 20 @#C000=0
 25 REM TEST FOR DARKNESS
 30 LET B=@#C000
 40 LET C=1 AND B
 50 IF C=0 GOTO 20
 55 REM ITS DARK SO INCREMENT A
 60 LET A=A+1
 70 IF A>5 GOTO 500
 80 GOSUB 200
 90 GOSUB 300
100 GOSUB 400
110 GOSUB 300
120 GOTO 20

200 LET J=0: DELAY 0
205 REM TURN ON LIGHT A
210 @#C000=1
220 LET K=RND(1,6)
225 REM KEEP LIGHT A ON FOR 10 TO 60 MINUTES
230 DELAY 0
240 LET J=J+K
250 IF J<3600 GOTO 230
255 REM TURN OFF LIGHT A
260 @#C000=0
270 RETURN

300 LET L=0: DELAY 0
305 REM TURN ON LIGHT B
310 @#C000=2
320 LET M=RND(1,5)
325 REM KEEP LIGHT B ON FOR 1 TO 5 SECONDS
330 DELAY 0
340 LET L=L+M

350 IF L<5 GOTO 330
355 REM TURN OFF LIGHT B
360 @#C000=0
370 RETURN

400 LET N=0: DELAY 0
405 REM TURN ON LIGHT C
410 @#C000=4
420 LET P=RND(1,4)
425 REM KEEP LIGHT C ON FOR 5 TO 20 MINUTES
430 DELAY 0
440 LET N=N+P
450 IF N<1200 GOTO 430
455 REM TURN OFF LIGHT C
460 @#C000=0
470 RETURN

500 LET A=0: LET D=0: LET F=0
510 LET E=RND(1,4)
515 REM TEST FOR DAYLIGHT
520 LET H=@#C000
530 LET I=1 AND H
535 REM IF ITS NOW LIGHT OUT BRANCH TO START
540 IF I=0 GOTO 10
545 REM IF STILL DARK WAIT FOR 1¼ TO 5 HOURS
550 DELAY 0
560 LET D=D+E
570 IF D<18000 GOTO 520
575 REM TURN ON LIGHT C
580 @#C000=4
590 LET G=RND(1,10)
595 REM KEEP LIGHT C ON FOR 1 TO 10 MINUTES
600 DELAY 0
610 LET F=F+G
620 IF F<600 GOTO 600
625 REM TURN OFF LIGHT C
630 @#C000=0
640 GOTO 500
```

increments the "sleep-phase variable," A. After that has been incremented past 5, statement 70 instructs the program to go to the "sleep phase" program segment, which starts at line 500.

If there have been less than 5 passes, the program jumps to the subroutine at line 200, which controls light A. First, the subroutine clears the variable J and delays the program for about a second (DELAY 0). Next, the subroutine turns on light A (@#C000 = 1). The statement on line 220 sets K equal to a pseudo-random number between 1 and 6. The statement "DELAY 0" delays further program execution for about a second and is the first part of the loop, which consists of the statements in lines 230, 240, and 250. The statement in line 240 increases the value of the variable J by the amount of K. (Remember "K" is between 1 and 6.) Line 250 causes a branch to line 230 if J is less than 3600. Note that if K = 1, this program loop will cause a delay of about 3600 seconds (60 min.). When K = 6 the delay is about $3600/6$ = 600 seconds (10 min). Notice that with this program shorter delay times are more likely than longer ones. That fact has nothing to do with an inherent defect in the RND command. Rather, it is built-into the design of the program. For instance, when K = 2, the delay time is about 30 minutes, with K = 3 the delay is about 20 minutes. There is no delay time between 60 and 30 minutes. Can you figure out the approximate delay times for K = 4 and K = 5?

The other light-control subroutines work the same way, but take the time to read through them to make sure you understand.

The statement at line 120 causes a branch back to line 20. After making sure all lights are off, the program checks to see if it's still dark out and if it is, variable "A" is incremented. When A reaches 6, the program branches to line 500. The program segment from lines 500 to 640 is the sleep phase. Lines 520, 530 and 540 check to make sure it's still dark. If it is light, the program branches back to the beginning. If it's still dark the program proceeds. Lines 550, 560 and 570 causes a delay of between 4,500 seconds (1¼ hours) and 18,000 seconds (5 hours).

When that "rest" time is over, the next statement, @#C000 = 4, turns on light C. The delay loop formed by the statements in lines 600, 610 and 620 keeps light C on for a period of time between about 1 and 10 minutes. The statement "@#C000 = 0" turns off all lights and the program branches back to line 500.

The controller circuit

The schematic of the complete burglar outwitter is shown in Fig. 7. The first thing we'll look at is photodarlington transistor Q1, which monitors the light level. When hit by light, the photodarlington conducts current. That produces a voltage drop across R4. That voltage also appears at pin 3 of IC5-a, the op-amp's non-inverting input. When the voltage at that input exceeds that at the inverting input (pin 2), the op-amp output of jumps to almost 5 volts. That triggers IC6-a (a Schmitt trigger NAND gate), whose output jumps low.

The output of IC6-a is fed to the input of the three-state buffer, IC7-a. This buffer is enabled only when a READ operation is taking place on location C000. (When a three-state buffer is not enabled, it is basically disconnected from the circuit.) The decoding for that buffer is provided by IC6-b, IC9-a and IC10-a.

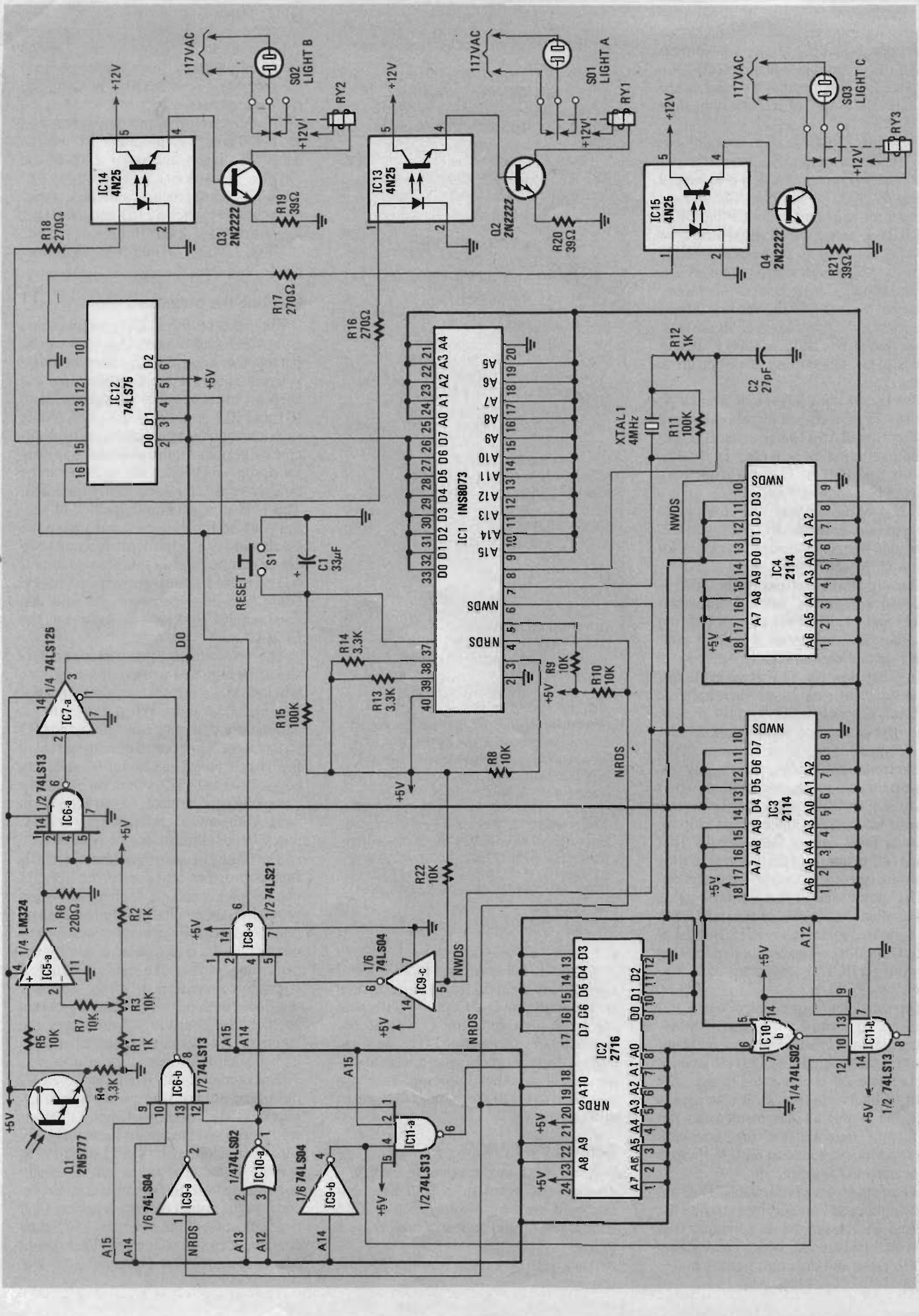

FIG. 7—THE BURGLAR OUTWITTER. Note that, although not shown in the schematic, bypass capacitors (about .1 μF) should be installed at each IC between the power lead and ground.

The output of IC7-a is fed to data-line DØ. Pulling the previous discussion together, we can see that when a READ operation is taking place on address C000, IC7-a's output will be low (DØ will be Ø) when it is light out. Of course, when it's dark, the output of IC7-a will be high (DØ will be 1).

The 2716 EPROM, IC2, contains our Tiny BASIC program. As far as the microinterpreter is concerned, it is located at addresses 8000 to 87FFH. The address decoder for IC2 consists of IC9-b, IC10-a and IC11-a. Since the EPROM's OUTPUT ENABLE pin (20) is connected to IC1's NRDS (READ DATA STROBE) output, IC2 is ENABLED only during a READ operation. Note that with the 8073, the first ASCII character of the program in EPROM must be located at hex address 8000 if the circuit is to be self-starting at power up or reset.

Two 1K × 4 static RAM's, IC3 and IC4, provide the scratch-pad memory required by the Tiny BASIC interpreter. The decoding required is provided by IC9-b, IC10-b, and IC11-b. That RAM memory starts at hex address 1000.

The 4-MHZ crystal and associated lowpass filter network (R11, R12, and C2) form the frequency-control network for the on-chip oscillator.

Start-up (initialization) of the 8073 is achieved with the R-C network consisting of R15 and C1. The RESET pin of IC1 (pin 37) goes low for about a second after power-up and then gradually rises to +5 volts. That slow rise in voltage is allowable as input since the RESET pin is buffered with a TTL-compatible Schmitt trigger. (The first instruction to be fetched, after pin 37 goes high, is at location 0001. Unlike microprocessors, which only understand machine language, the designer who uses an 8073 need not know this specific information since the on-chip interpreter takes care of these details. That is just one example of the savings in time microinterpreters provide to the designer.)

The output circuit that controls lights and appliances consists of the quad-latch IC12, optocouplers IC13–IC15, and associated circuitry. The address-decoding network for IC12 is made up of IC8-a, IC9-c, and IC10-a.

The quad latch is enabled when IC1 attempts to write to location C000H. Once a latch is enabled it can store data. Thus the statement @#C000 = 1 enables IC12. Since DØ is connected to the "D" input of latch one (pin 2), it also causes latch one to store a 1. The result is that pin 16, the "Q" output of latch one, goes high and stays that way at least until IC1 writes to location C000 again.

When pin 16 goes high, the LED inside optocoupler IC13 is energized causing the internal phototransistor to conduct. That turns on transistor Q2, which causes relay RY1 to close and thus turn light A on.

PARTS LIST

All resistors ¼ watt, 5% unless otherwise specified

R1, R2, R12—1000 ohms
R3—10,000 ohms potentiometer
R4, R13, R14—3300 ohms
R5, R7, R9, R10, R22—10,000 ohms
R6—220 ohms
R11, R15—100,000 ohms
R16, R17, R18—270 ohms
R19, R20, R21—39 ohms

Capacitors

C1—33 µF, 10 volts, electrolytic
C2—27 pF ceramic disc
C3–C13—0.1 µF bypass capacitors (not shown on schematic)

Semiconductors

IC1—INS8073 microinterpreter
IC2—2716 EPROM (properly programmed, see TABLE 1 and the text)
IC3, IC4—2114 1K × 4 RAM
IC5—LM324 quad op-amp
IC6,IC11—74LS13 dual NAND Schmitt trigger
IC7—74LS125 quad buffer
IC8—74LS21 4 input AND gate
IC9—74LS04 hex inverter
IC10—74LS02 quad NOR gate
IC12—74LS75 4-bit bistable latch
IC13, IC14, IC15—4N25 or equivalent optocoupler
Q1—2N5777
Q2,Q3,Q4—2N2222

Other components

RY1,RY2,RY3—6-volt DC relay with 50 ohm coil, (Calectro D1-973 or equivalent)
S1—SPST momentary type switch
XTAL1—4 MHZ crystal
Miscellaneous: PC or prototyping board, two-conductor cable (for Q1), 14-gauge U.L. approved wire (for wiring the light circuits to the relays), hook up wire, solder, etc.

The INS8073 demo/development board (Part No. DB10-ND), is available from Digi-Key Corporation, Highway 32 South, P.O. Box 677, Thief River Falls, MN 56701-9988 (1-800-346-5144) for $250.00.

Similarly, the statement @#C000 = 2 causes D1 to go high, which turns on light B. The statement @#C000 = 4 will cause D2 to go high and light C will go on. Notice that by connecting D3 to pin 7 and adding another optocoupler/transistor/relay circuit, another light/appliance can be controlled by the statement @#C000 = 8.

Burning the EPROM

As we mentioned previously, the demo/development board from Digi-Key contains an on-board burner for 2716 EPROM's. A brief review of how to use it follows.

After you obtain the prompt (>) on your terminal, type: NEW#1100<return>. Then type: NEW<return>. You are now ready to enter the program in Table 1 (or your own program). To save time (and memory space for longer programs), you might want to leave out the REM statements.

After entering your program, connect a +25 volt source to pin 3 of "P1" on the board and insert an erased 2716 in the empty socket. Now type: NEW#8800<return> and then type: S<return>. Switch S2 to P and then type: P<return>. If everything goes well, "DONE,CMD?" should show on the display.

Building the outwitter

Since this project is fairly simple (for a computer-based project, that is) it can be put together on a solderless breadboard. If you use such a system, make sure you keep all the wires that are connected to IC1 and IC2 as short as possible. While such a breadboarding technique isn't suitable for permanent installations, it is ideal for design work since the circuits can be quickly constructed and easily modified. This is also an economic method of testing your design, since the parts from the breadboard can be used in a different project or can be used in the final circuit. If you do make a permanent version on a PC board or wire-wrap board, be sure that you use IC sockets—at least for the EPROM.

The photodarlington transistor, Q1, should be mounted so that it is facing out a window. Make sure that you don't locate it near an indoor light. When it starts to get sufficiently dark out, carefully adjust R3 so that light A goes on. The relays listed in the parts list are Calectro D1-973 and can control 200 watt (120-volt) light bulbs. By connecting the contacts of those relays in series with the coil circuit of a power relay, much larger currents can be controlled.

The beauty of using microinterpreters is that the program contained in the EPROM can be quickly and easily changed—short programs can be modified in less than an hour. As a real-life example, let's look at our circuit. It is possible to modify the program so that the circuit can tell whether it is summer or winter. One way that could be done is by writing a program so that the "outwitter" measures the length of day. The circuit can then modify its behavior according to the season.

The output circuit itself can be easily modified to control more lights/appliances. The addition of light D has already been touched upon previously. By adding an additional 74LS75 quad-latch, 4 more lights/appliances can be easily controlled. With the use of a suitable decoder to the output of these 8 latches (and a modification of the program) 256 lights/appliances can be controlled. That should be nearly enough for everyone! **R-E**