

The following material is excerpted from:

The Microcontroller Idea Book

Circuits, Programs, & Applications

featuring the 8052-BASIC Microcontroller

by Jan Axelson

copyright 1994, 1997 by Jan Axelson

ISBN 0-9650819-0-7

Published by Lakeview Research

Distribution by International Thomson Publishing (ITP) in arrangement with
Peer-to-Peer Communications.

For more information, contact:

Lakeview Research
2209 Winnebago St.
Madison, WI 53704
USA

Phone: 608-241-5824

Fax: 608-241-5848

Email: jaxelson@lvr.com

World Wide Web: <http://www.lvr.com>

You may distribute this material if you agree to distribute it in full and unchanged and agree to charge no fee for such distribution with the exception of reasonable media charges.

The author and publisher have used their best efforts in preparing this work and the materials in it. The author built and tested the electronic circuits described, ran and tested the computer programs presented, and reviewed all materials for completeness and accuracy. The author and publisher make no warranty with regard to the circuit schematics, program listings, and other materials in this work. The author and publisher take no responsibility for any damages resulting from any use of the material in this work.

3

Powering Up

This chapter presents a circuit that enables you to start using the 8052-BASIC chip. You can write and run programs and experiment with the BASIC-52 programming language. Later, you can add non-volatile memory for permanent program storage and interfaces to displays, keypads, and whatever else your projects require.

About the Circuit

Figure 3-1 contains all of the components you need to get a BASIC-52 system up and running, plus a few optional extras for future use. Table 3-1 is a parts list for the circuit.

The circuit has five major components: the 8052-BASIC chip (U2), an address latch (U4), an address decoder (U6), static RAM (U7), and an RS-232 interface (U5). As I'll explain below, a few of the components aren't essential at this point, but I've included them to allow easy expansion later on.

The circuit configuration is a more-or-less standard design, similar to many other microcontroller circuits. When you understand this circuit, you're well on your way to understanding many others.

The following paragraphs explain the circuit operation, component by component. If you're impatient to get started, you can skim or skip over this section for now, and go straight to the construction details.

Figure 3-1. Complete 8052-BASIC system for experimenting.

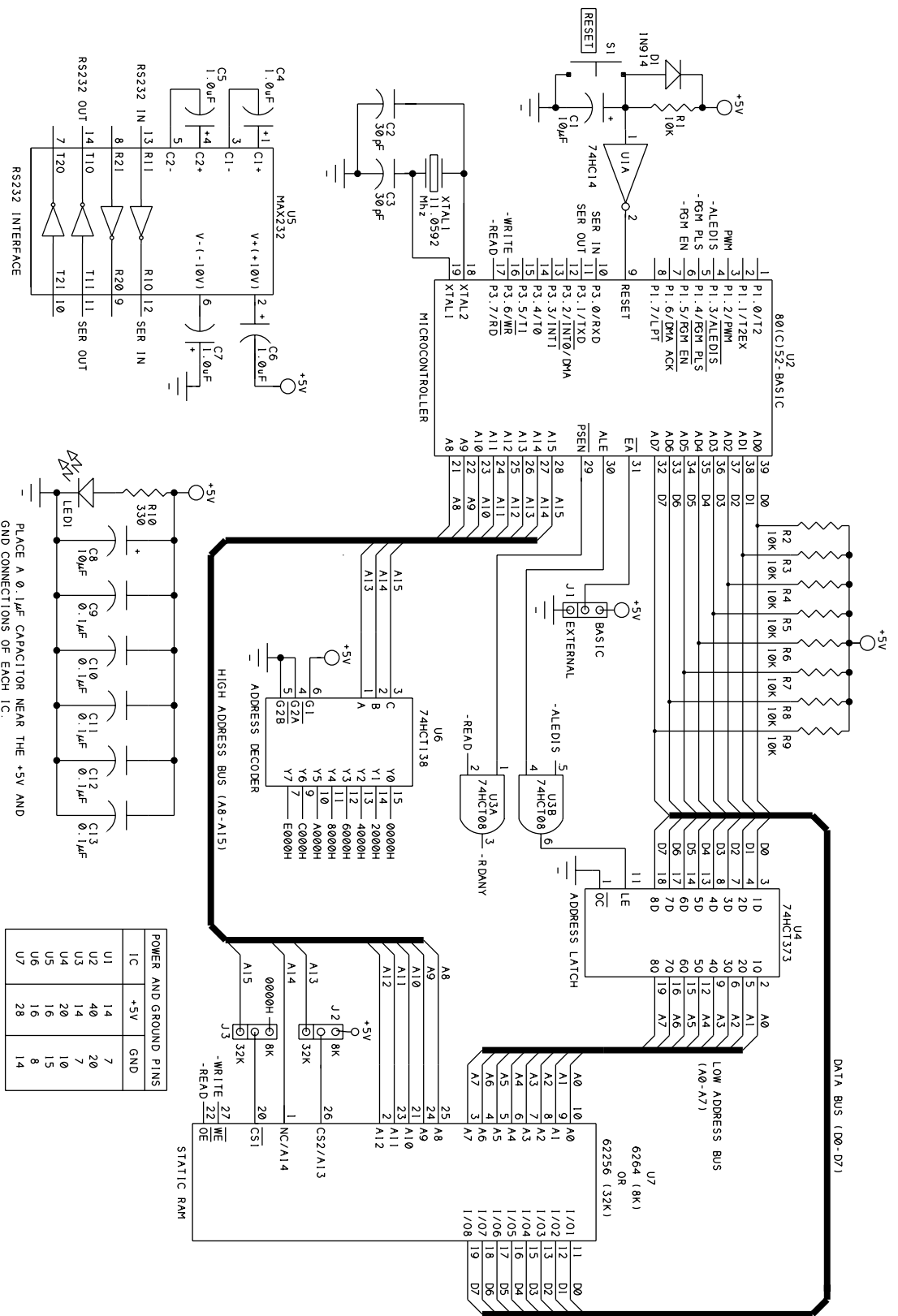


Table 3-1. Parts list for Figure 3-1's circuit.**Semiconductors**

LED1	Light-emitting diode
U1	74HC14 quad inverting Schmitt trigger
U2	8052-BASIC or 80C52-BASIC microcontroller
U3	74HCT08 quad AND gate
U4	74HCT373 octal transparent latch
U5	MAX232, RS-232 driver/receiver
U6	74HCT138 3-to-8-line decoder
U7	6264 (8 kilobyte) or 62256 (32 kilobyte) static RAM, access time 250ns or less

Resistors (1/4-watt, 5% tolerance)

R1-R9	10,000-ohm
R10	330-ohm

Capacitors (16WVDC, 20% tolerance)

C1,C8	10-microfarad, aluminum or tantalum electrolytic
C2,C3	30-picofarad ,ceramic disc
C4-C7	1.0-microfarad,aluminum or tantalum electrolytic
C9-C13	0.1-microfarad, ceramic disc

Miscellaneous

J1-J3	SIP header, 3-terminal, and shorting block
S1	Switch, normally-open momentary pushbutton
XTAL1	11.0592-Mhz crystal

RS232 connector, IC sockets, perforated board, wire, solder, and other circuit-construction materials

The Microcontroller

U2 is the 8052-BASIC chip. The circuit is designed so that you can use either the NMOS version or the CMOS 80C52-BASIC.

\overline{EA} , the External Access Enable input (pin 31 of U2), connects to +5V. This causes the 8052 to run the BASIC-52 interpreter in ROM on boot-up. If \overline{EA} is low, the 8052 ignores its internal ROM and instead accesses external program memory on boot-up. You can wire \overline{EA} directly

Chapter 3

to +5V, or use a jumper as shown in the schematic, to allow you to bypass BASIC-52 and boot to an assembly-language program in external memory, as described in Chapter 13.

The crystal. XTAL1 is an 11.0592-Mhz crystal that connects to pins 18 and 19 of U2. This crystal frequency has two advantages. It gives accurate baud rates for serial communications, due to the way that the 8052's timer divides the system clock to generate the baud rates. Plus, BASIC-52 assumes this frequency when it times the real-time clock, EPROM programming pulses, and serial printer port.

However, you should be able to use any crystal value from 3.5 to 12 Megahertz. If you use a different value, you can use BASIC-52's XTAL operator to adjust the timing to match the frequency of the crystal you are using. The serial communications are reliable if the baud rate is accurate to within a few percent. The higher the crystal frequency, the faster your programs will execute, so most designs use either 11.0592 Mhz or 12 Mhz, which is the maximum clock frequency that the standard 8052 chip can use.

Capacitors C2 and C3 are 30 picofarads each, as specified in the 8052's data sheet. Their precise value isn't critical. Smaller values decrease the oscillator's start-up time, while larger values increase stability.

Reset circuit. A logic high on pin 9 of U2 resets the chip. On power up, pin 1 of U1 rises slowly from 0V to +5V as capacitor C1 charges through resistor R1. Inverter U1 has a Schmitt-trigger input, which has upper and lower switching thresholds that help to ensure a clean reset pulse at pin 9 of U2. On a logic gate that doesn't have a Schmitt-trigger input, the output may oscillate if a slowly changing input remains near the switching threshold. In contrast, at U1, when pin 1 reaches the upper switching threshold (about 2.8V), pin 2 switches from high to low, but won't go high again until pin 1 drops to the lower threshold of about 1.8V.

Pressing and releasing S1 resets the 8052-BASIC chip by discharging C1 and then allowing it to recharge, which brings RESET high, then low again

External Memory

The remaining connections to U2 have to do with reading and writing to external memory.

Read and write signals. To enable reading combined program and data memory, AND gate U3A's output is \overline{RDANY} . This signal is low when either \overline{READ} or \overline{PSEN} is low. Figure 3-1's circuit doesn't use \overline{RDANY} , but I've included U3A for future use. Writing to data memory is controlled by \overline{WRITE} . Code memory can't be written to.

AD0-AD7 connect to U4, a 74HCT373 octal transparent latch that stores the lower address byte during memory accesses. The chip contains a set of D-type latches that store logic states.

74HCT138
3-TO-8-LINE DECODER

INPUTS						OUTPUTS							
ENABLE			SELECT										
G1	$\overline{G2A}$	$\overline{G2B}$	C	B	A	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
L	X	X	X	X	X	H	H	H	H	H	H	H	H
X	H	X	X	X	X	H	H	H	H	H	H	H	H
X	X	H	X	X	X	H	H	H	H	H	H	H	H
H	L	L	L	L	L	L	H	H	H	H	H	H	H
H	L	L	L	L	H	H	L	H	H	H	H	H	H
H	L	L	L	H	L	H	H	L	H	H	H	H	H
H	L	L	L	H	H	H	H	H	L	H	H	H	H
H	L	L	H	L	L	H	H	H	H	H	L	H	H
H	L	L	H	L	H	H	H	H	H	H	H	L	H
H	L	L	H	H	L	H	H	H	H	H	H	H	L
H	L	L	H	H	H	H	H	H	H	H	H	H	L

74HCT373
OCTAL TRANSPARENT LATCH

OUTPUT CONTROL	LATCH ENABLE	DATA	OUTPUT
OC	LE	1D-8D	1Q-8Q
L	H	H	H
L	H	L	L
L	L	X	NO CHANGE
H	X	X	Z

L=LOGIC LOW
 H=LOGIC HIGH
 X=DON'T CARE
 Z=HIGH IMPEDANCE

Figure 3-2. Truth tables for the 74HCT138 decoder and 74HCT373 octal transparent latch.

A latch-enable input (LE) controls whether the outputs are latched (stored), or not latched (immediately follow the inputs). Figure 3-2 shows the truth table for the chip. When pin 11 is high, 1Q-8Q follow 1D-8D. When pin 11 goes low, outputs 1Q-8Q will not change until pin 11 goes high again.

During each external memory access, 1Q-8Q store the low address byte, so the eight lines that connect to these outputs carry the label **LOW ADDRESS BUS**. AND gate U3B latches, or stores, U4's outputs only when both ALE and \overline{ALEDIS} are high. During normal memory accesses, \overline{ALEDIS} remains high, and ALE controls U4. \overline{ALEDIS} disables the latches when BASIC-52 executes its programming commands. Figure 3-1's circuit doesn't use the programming commands, so ALE could control U4 directly, but again, I've included U3B for future use.

Because AD0-AD7 hold the data to be read or written during a memory access, the signals as a group carry the label **DATA BUS**. Each line of AD0-AD7 has a 10K pullup resistor. These are

required for the programming functions, and are included for future use. You can use eight individual resistors, or a resistor network that contains eight resistors in a SIP or DIP package. In a bussed resistor network, one pin connects to one side of all of the resistors, so you have fewer connections to wire.

The remaining bus is the HIGH ADDRESS BUS (A8-A15), which consists of the upper eight address lines, and is not multiplexed.

Address decoding. U6 is a 74HCT138 3-to-8-line decoder. It functions as an address decoder for the 64K external memory space. Address decoding allows multiple chips to connect to the address and data buses, with each chip enabled only when it is selected.

Figure 3-2 shows a truth table for the decoder. The 8052-BASIC chip uses the three highest address lines (A13-A15) to generate a chip-select signal for each of eight 8K blocks in memory. This is by no means the only way to decode memory, but it's a common and flexible one. In the schematic, each output is labeled with the base, or bottom, address in the block it controls.

For example, when U2 reads or writes to an address between 0 and 1FFFh in external memory, A13, A14, and A15 are low, so pin 15 of U6 is low. For all other addresses, pin 15 is high. If we connect pin 15 to the chip-select input of an 8K RAM, the RAM will be enabled only when addresses from 0 to 1FFFh are accessed. (Remember that 8K, or 8 kilobytes, is 2000h, or 0 through 1FFFh, in hexadecimal.)

If you use a 32K RAM, you don't need U6 to decode its addressing. For all of the 32K RAM's addresses (0 to 7FFFh), A15 is low, and for all other addresses (7FFFh to FFFFh), A15 is high. This means that you can use A15 directly as a chip select, without additional decoding. U6 will come in handy later, however, even if you use a 32K RAM.

RAM choices. The minimal circuit includes just one memory chip, U7, which can be an 8K or 32K static RAM, or SRAM. BASIC-52 requires at least 1K of RAM, but I've used the larger capacities, since the extra room is useful and doesn't cost much more. The pinouts of the two chips are similar, with jumpers J2 and J3 routing the signals that vary.

The 8K chip has 13 address inputs (A0-A12), while the 32K chip has 15 (A0-A14). Eight data I/O pins (I/O1-I/O8) connect to the data bus and hold the bytes to be read or written.

The RAM has three control inputs whose functions match those of the 8052's control outputs. Pin 20 ($\overline{CS1}$, or Chip Select 1) enables U7 whenever the 8052 reads or writes to the chip, with the address decoding determining the address range of the chip.

Jumper J3 chooses the chip select for an 8K or 32K device. Some 8K RAMs have a second chip select (CS2), which is tied high (always selected) by J2. If you limit yourself to either 8K or 32K RAMs, you can eliminate J2 and J3 and wire the appropriate connections directly.

Pin 27 (\overline{WE} , or Write Enable) is driven by \overline{WRITE} , and is strobed low during each write to external data memory. Pin 22 (\overline{OE} , or Output Enable) is driven by \overline{READ} , and strobes low when either external data or code memory is read.

With an 8K RAM, each write cycle follows this sequence: The 8052 brings ALE high and places the address to be written to on AD0-AD7 and A8-A15. For addresses from 0 to 1FFFH, A13-A15 are low, so U7 is selected at its pin 20. After a short delay, the 8052 brings ALE low, which causes U7 to store the lower address byte. After another short delay, the 8052 replaces the address on AD0-AD7 with the data to be written. A low pulse at pin 27 (\overline{WE}) causes the RAM to write the data into the address specified by A0-A12.

Read cycles are similar, except that a pulse at pin 22 (\overline{OE}) causes the requested data to appear on AD0-AD7, where the 8052 reads it.

With a 32K RAM, the process is the same, except that A15 is the chip select and there are two more address lines on the chip.

Static RAM chips are rated by their read-access time, which is the maximum time the chip will require to place a byte on the data bus after a read is requested. With a crystal frequency of 12 Mhz or lower, an access time of 250 nanoseconds or less is fine for accessing external data or code memory. Access times and other timing characteristics are described in the timing diagrams in the data sheets for the 8052 and RAM.

When you use the 8052-BASIC, you don't have to worry about any of these specifics about the read and write cycles. If the circuit is wired correctly, and if all of the components are functioning as they should, reading and writing occur automatically in the course of executing BASIC-52 statements and commands. A single program line in BASIC-52 can cause dozens or more read and write operations to occur.

Logic families. Logic chips U3, U4, and U6 are HCT-family components, which have TTL-compatible inputs and CMOS-compatible outputs. This means that they can interface directly to either TTL or CMOS logic.

If HCT-family parts aren't available, there are alternatives. You may use an LSTTL chip (74LS08, 74LS138, 74LS373) for U3, U4, or U6. Or, if you use a CMOS 80C52-BASIC for U2, you may use an HCMOS 74HC08 or 74HC138 for U3 or U6. If U3 is a 74HC08 or 74HCT08, you may use a 75HC373 for U4. For U1, you may use a 74HC14 or 74LS14.

Table 3-2. Voltage specifications for different types of logic, powered at 5V.

Logic Type	Output		Input	
	0 (maximum)	1 (minimum)	0 (maximum)	1 (minimum)
TTL, including LSTTL most NMOS	0.4V	2.4V	0.8V	2.0V
HCTMOS	0.1V	4.9V	0.8V	2.0V
HCMOS	0.1V	4.9V	1.0V	3.5V
4000-series CMOS	0.1V	4.9V	1.5V	3.5V

Table 3-2 summarizes the input and output voltage specifications for different logic-device families. The main point to remember is that a TTL logic-high output voltage (and most NMOS high outputs) may be as low as 2.4V, which does not meet the minimum input-voltage requirement for HCMOS or 4000-series CMOS devices. To interface a TTL output to CMOS, use an HCTMOS device, which accepts TTL-logic inputs. Or, you may add a pull-up resistor to a TTL output to pull it near +5V.

Serial Interface

The final chip in the schematic is U5, a MAX232 driver/receiver, which is the popular single-chip solution for RS-232 interfaces. One side connects to the 8052's serial input and output on pins 10 and 11 of U1, and the other side sends and receives signals at standard RS-232 levels to a personal computer. Larger capacitor values for C4-C7 are fine, and the MAX232A version can use values as small as 0.1 microfarad. If you splurge on a MAX233, which has internal capacitors, you don't need C4-C7 at all.

Power Supply

A final essential component is the power supply. For the basic system, all you need is a regulated +5-volt supply. These are widely available from mail-order suppliers. An output capability of at least 500 milliamperes is recommended.

Capacitors C8-C13 provide power-supply decoupling. Digital devices draw current as they switch. Capacitors C9-C13 store energy that the components can draw quickly, without causing spikes in the supply or ground lines. C8 stores energy for quick recharging of C9-C13. The exact values aren't critical, but C9-C13 should be a type with good high-frequency response, such as ceramic, mica, or polystyrene.

LED1 and current-limiting resistor R10 are an optional power-on indicator.

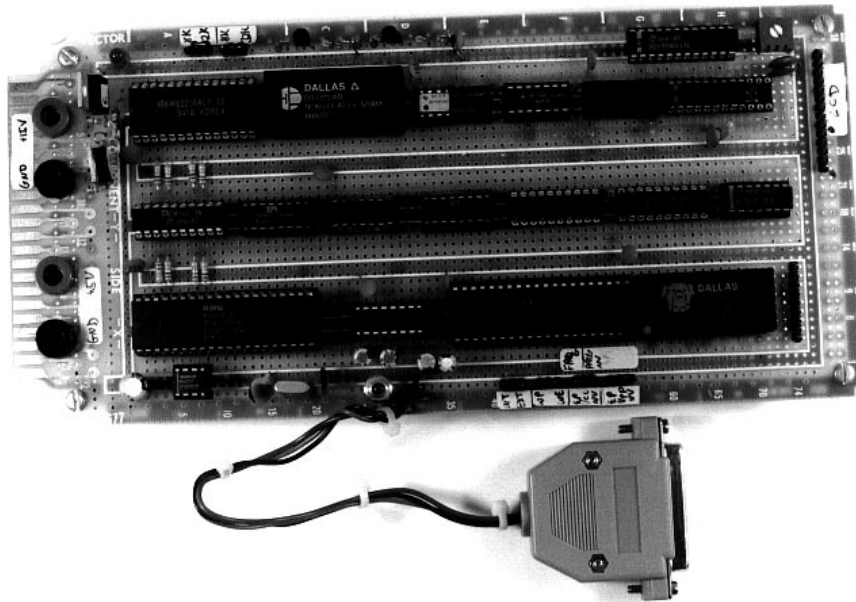


Figure 3-3. This is the circuit board on which I wire-wrapped and tested many of the circuits in this book.

Circuit Construction

This circuit is intended for use as a flexible system for testing and experimenting, rather than a fixed, unchanging design for a single application. For this reason, I recommend building it with wire-wrapping or another construction method that allows easy changes and additions. Figure 3-3 shows an 8052-BASIC circuit wire-wrapped onto perfboard.

Reading the Schematic

In the schematic, I used a couple of different techniques to represent connections between pins and components. In the reset circuit, connections are drawn as direct point-to-point lines. For the address and data lines, I used buses for a neater, more compact schematic. When you wire these connections, use the signal labels as a guide. For example, the label D0 tells you to interconnect these points: pin 39 of U2, pin 3 of U4, pin 11 of U7, and one end of R2. Other connections are indicated by labels. For example, the $\overline{\text{WRITE}}$ label tells you to connect pin 16 of U2 and pin 27 of U7.

Another point to be aware of is the conventions used in the schematics and text of this book for indicating an active-low signal, or a signal that is valid, or enabled, when low. In this book, the schematics use a leading hyphen (-WRITE), while the text uses an overscore ($\overline{\text{WRITE}}$). Their meanings are the same.

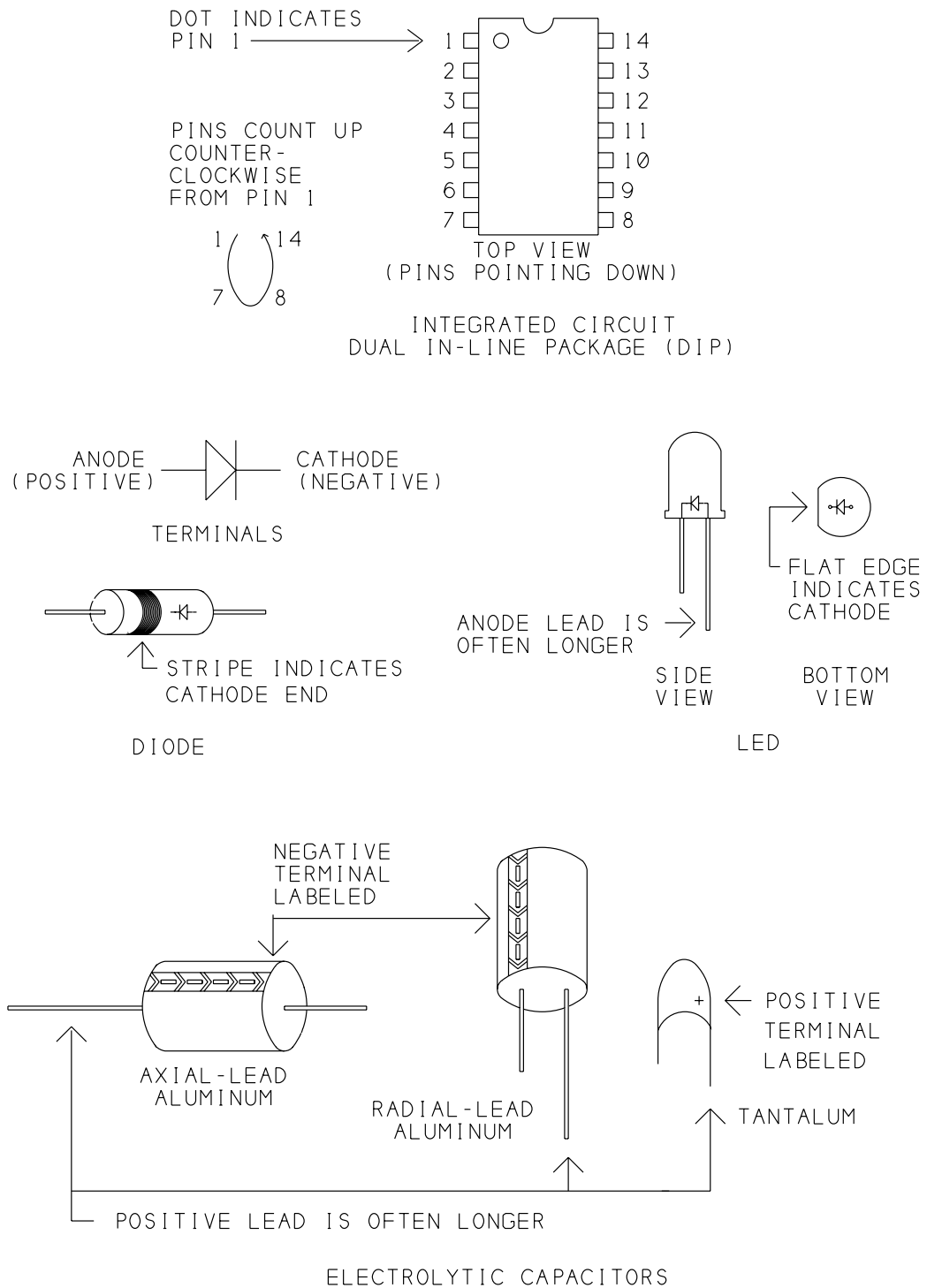


Figure 3-4. How to determine the correct orientation for ICs, diodes, LEDs, and electrolytic capacitors.

Construction Tips

These are some things to be aware of as you build the circuit:

- Choose a circuit board that has room for additions, at least 4 by 6 inches.
- A board with interleaved buses, such as Vector's 3677 series, allows easy, low-impedance connections to +5V and ground. Designate one bus as ground, and the other as +5V. For power and ground connections, wrap one end of the wire to the appropriate pin on the chip, and trim and solder the other end directly to the bus.
- To connect the power and ground buses to the +5V supply, use thick (AWG #22 or lower) wires, not #30 wire-wrap wires. You can solder the other ends of the wires to banana plugs or screw terminals, or clip your power-supply leads directly to the wires.
- The schematic doesn't show an ON/OFF switch for the circuit, but you can add a SPST toggle or slide switch in series with the connection to the +5V supply if you wish.
- Place C8 near where the +5V supply connects to the board. Mount decoupling capacitors C9-C13 so that each chip's +5V and GND pins are near a capacitor. In other words, space the capacitors evenly around the board; don't group them all in one area. Keep the wires or traces between the capacitor's leads and the IC's +5V and ground pins as short as possible.
- To minimize noise in the oscillator circuits, place XTAL1, C2, and C3 close to pins 18 and 19 of U2 and connect them with short wires. Wire the ground terminals of C2 and C3 directly to pin 20 of U2.
- When you wire the following components, correct orientation is required: C1, C4-C8, D1, LED1, and U1-U7. Figure 3-4 shows common polarity indicators for these components. Notice that C7's positive terminal connects to ground, and C6's negative terminal connects to +5V, since these capacitors connect to the MAX232's -10V and +10V outputs.
- As you wire the circuits, remember that everything on the wire-wrap or solder side of the board is a mirror image of the way it looks on the component side of the board. If pin 1 is in the upper left corner on the component side, it's in the upper right corner on the wire-wrap side (assuming that you flip the board over from side to side, not top to bottom).
- Labels on the wire-wrap side are helpful. You can place a dot of indelible ink near pin 1, or adhesive labels between the pins, or use pre-labeled and punched plastic labels that slide onto the wire-wrap pins.

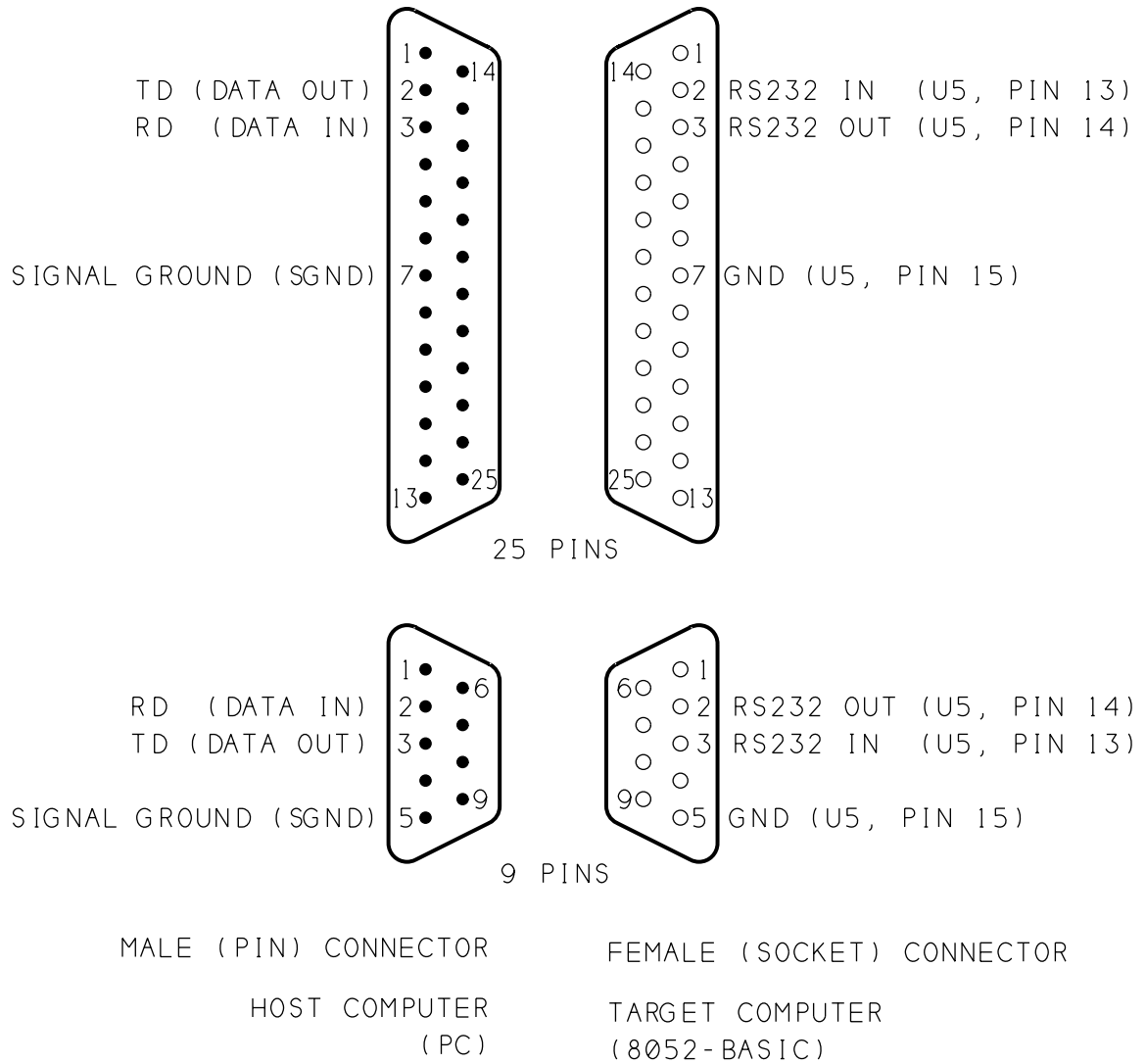


Figure 3-5. Pin connections for 25-pin and 9-pin RS-232 connectors.

- Don't plug the ICs into their sockets until you've completed wiring all of the circuits.

Unused Gates

Two gates on U3 and five gates on U1 are unused. To prevent the unused CMOS inputs from floating and possibly drawing excessive currents, wire pins 9, 10, 12, and 13 of U3 to ground or +5V. Do the same for pins 3, 5, 9, 11, and 13 of U1. Don't forget to remove these connections if you later use the pins. If you are using LSTTL chips (74LS08, 74LS14), leave the unused inputs open.

Serial Connectors

Connections to RS-232 OUT and RS-232 IN depend on the type of serial connector you have on your personal computer or its serial cable.

Connectors vary, but two common ones are a male 25-pin or 9-pin D-connector. (The outer shell of a D-connector is roughly in the shape of a *D*.) For the 8052-BASIC system, you'll need a mating female 25-pin or 9-pin D-connector. The connection has just three wires. A solder-cup-type connector allows easy soldering of the wires.

Figure 3-5 shows the wiring for 9- and 25-pin connectors. A few computers require additional handshaking signals. BASIC-52 doesn't support these, but you can simulate them by connecting together pins 5, 6, 8, and 20 at the personal-computer end of the link. (Pin numbers are for a 25-pin connector.)

Powering Up

The first time you power up an untested circuit, it pays to be cautious. I recommend the following steps:

First Steps

Visually inspect the circuit. You don't have to spend a lot of time on this, but sometimes a missing or miswired wire or component or another problem will become obvious.

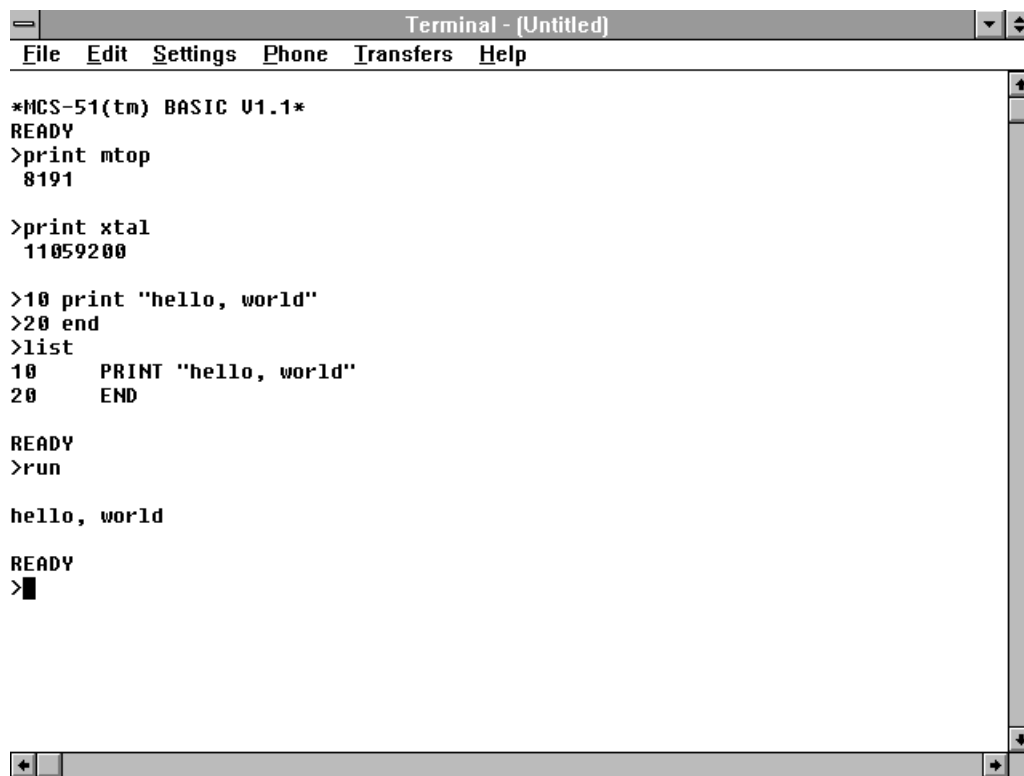
Install U1-U7 on the board, making sure that pin 1 on each is oriented correctly. Set J1 to BASIC, and set J2 and J3 to match the size of your RAM at U7.

With an ohmmeter, measure the resistance from +5V to ground, to be sure these aren't shorted together by mistake. The exact value you measure isn't critical, but if you read less than 100 ohms, something is miswired and you need to find and fix the problem before you continue.

If you suspect a problem, check the wiring of the power and ground connections, comparing the connections to those on the schematic. Be sure all components are oriented correctly. When all checks out, you're ready to boot up BASIC-52.

Booting BASIC-52

For the initial check, begin with everything powered down. I'll use the term *host computer*, or *host system*, to refer to the personal computer, and *target computer*, or *target system*, to refer to the 8052-BASIC circuits. Included are some specific tips for users of Datastorm's

The image shows a screenshot of a Windows Terminal window titled "Terminal - (Untitled)". The window has a menu bar with "File", "Edit", "Settings", "Phone", "Transfers", and "Help". The terminal content shows the following text:

```
*MCS-51(tm) BASIC V1.1*
READY
>print mtop
8191

>print xtal
11059200

>10 print "hello, world"
>20 end
>list
10 PRINT "hello, world"
20 END

READY
>run

hello, world

READY
>
```

Figure 3-6. BASIC-52's sign-on message and a simple program, using the Windows Terminal accessory for communications.

Procomm Plus for DOS and Microsoft Windows 3.1's Terminal Accessory, but other communications software should have similar features and abilities.

Turn on the host computer and run your communications software. Configure the software for 8 data bits, no parity, and 1 stop bit. The baud rate you select isn't critical, since BASIC-52 automatically adjusts to what you are using. To start, use a rate of 9600 or less. Don't enable any handshaking or flow-control options such as XON/XOFF or RTS/CTS.

Select the appropriate serial, or COM, port, if necessary. If you're using an MS-DOS (IBM-compatible) computer, you must find a COM port and interrupt-request (IRQ) level that aren't being used by your modem, mouse, or another device. Because COM1 and COM3 often share an IRQ level, as do COM2 and COM4, you generally can't use COM1 and COM3 at the same time, or COM2 and COM4. If you have an external modem, you can unplug it and use its serial port.

In *Procomm Plus*, use the line/port setup menu (ALT+P) to configure. In the Windows Terminal, use the Settings menu. Cable together the serial ports of the host and target systems.

You're now ready to power up the target system. Turn on its power supply, and press the SPACE bar at the host's keyboard. You should see this BASIC-52 sign-on message and prompt:

```
*MCS-51(tm) BASIC V1.1*  
READY
```

Figure 3-6 shows the sign-on message and a simple program, using Windows' Terminal accessory for communications.

Troubleshooting

If you don't see the prompt, it's time to troubleshoot. Getting the system to boot the first time can be the most challenging part of a project, especially when serial communications are involved. Here are some things that may help you isolate the cause of the problem:

- Try again by pressing and releasing S1 and pressing the space bar. If you are using a 32K RAM for U7, BASIC-52 requires about 1 second to perform its memory check after a reset, before it will respond to the space bar. With an 8K RAM, the delay is a few tenths of a second (proportionately longer with slower crystals).
- Double-check the easy things. Are the communications parameters correct? Did you select the correct serial port? Are all ICs inserted?
- Verify that pin 9 of U2 goes high, then low, when you press and release S1.
- Check the power and ground pins of all ICs for proper voltages.
- Connect a logic probe to pin 10 of U2. When you press the space bar, you should see the logic level toggle as U2 receives the ASCII code for a space (20h). If not, you probably have a problem in the setup of your communications software or in the serial cabling.
- Verify that pin 30 of U2 is toggling (at 1/6 the crystal frequency, if you have an oscilloscope to measure). This indicates that the oscillator circuit is functioning.
- Verify that pins 21-28 and 32-39 of U2 toggle as BASIC-52 performs its memory check immediately after powering up or rebooting.
- If all else fails, recheck your wiring for missing or misrouted wires. Sometimes there's no alternative but to go through the schematic connection by connection, checking each with an ohmmeter.

Basic tests

When your system boots, you're ready for some basic tests. The BASIC-52 programming manual is a useful reference at this point.

In some ways, BASIC-52 is similar to BASIC compilers like Microsoft's *QuickBASIC*. Many of the keywords and syntax rules are similar. But BASIC-52 is closer to older interpreted BASICs like *GW-BASIC* or *BASICA*. You can type a statement or command and execute it immediately when you press ENTER, or you can type a series of statements and run them later as a program. When a line begins with a line number, BASIC-52 treats it as a program line rather than as a command to execute immediately.

Here are some quick tests and experiments you can do:

Memory Check

Type

```
PRINT MTOP
```

to learn the amount of external data memory that BASIC-52 detected on boot-up. With an 8K RAM, MTOP should be 8191, and with 32K, it should be 32,767. If you prefer hexadecimal notation, type

```
PH0 . MTOP
```

(In PH0 . , be sure to include the period and use a zero, not the letter "O".)

Crystal Frequency

The special operator XTAL represents the value of the timing crystal that clocks the 8052-BASIC. The default value is 11059200, or 11.0592 Mhz. You can verify this by typing

```
PRINT XTAL
```

Most BASIC-52 statements don't use the XTAL operator, so it doesn't matter if the value isn't accurate. Exceptions are the real-time clock, programming commands, PWM output, and LPT output. For these, XTAL should match your crystal's frequency. To set XTAL for a 12Mhz crystal, type

```
XTAL=12000000
```

To verify, type

```
PRINT XTAL
```

Line Editing

After typing a few commands, you may discover some of BASIC-52's line-editing abilities. While typing a line, you can correct mistakes by deleting back to the mistake and retyping. In *Procomm Plus*, if you select VT100 terminal emulation (under Setup menu, Terminal Options), you can use either the DELETE or BACKSPACE key to delete. With the *Windows* terminal, you must use the DELETE key (not BACKSPACE). Many communications programs allow you remap the keyboard, so you can select whatever delete key you wish.

Once you press ENTER, you can't edit a line you've typed, unless you retype it from the beginning.

BASIC-52 treats upper and lower-case characters the same. In most cases, spaces are ignored, so you can include them or not as you wish.

Running a Program

Here is a very simple program to try:

```
10 FOR I=1 to 10
20 PRINT I
30 NEXT I
20 END
```

Enter each of the lines, including the line numbers. BASIC-52 automatically stores the program in RAM. To run the program, type RUN. You should see this:

```
1
2
3
4
5
6
7
8
9
10
```

To view the program lines, type

```
LIST
```

Chapter 3

To erase the current program, type

```
NEW
```

To verify that the program no longer exists, type

```
LIST
```

You can change individual program lines by typing the line number, followed by a new statement:

```
10 FOR I=1 to 20
```

To erase a line, type the line number and press ENTER:

```
20
```

Getting Out of Trouble

Occasionally, a programming error may cause a program to go into an endless loop or crash the system. If it's an endless loop, you can exit it and return to the `READY` prompt by pressing `CONTROL+C`. If that doesn't work, your only choice is to press `S1` to reset the 8052-BASIC system. Resetting will erase the program in RAM, so you'll have to re-enter it.

Simple Programs to Try

The following sections offer some short programs to try, to help you explore your system and become familiar with BASIC-52. Don't worry if you don't understand every line of the programs. Later chapters get into programming in more detail.

Reading Port 1

You can use BASIC-52 to read and write to Port 1 (pins 1-8) on the 8052-BASIC.

The command

```
PH0 .PORT1
```

will display the hex value of the entire port. Listing 3-1 is a program that displays the value of each of the bits in the port.

Enter each line carefully. Be sure to include all of the punctuation shown. When you run the program, you should see a display like this:

PORT 1 Bit Values:

```
Bit 0 = 1
Bit 1 = 1
Bit 2 = 1
Bit 3 = 1
Bit 4 = 1
Bit 5 = 1
Bit 6 = 1
Bit 7 = 1
```

If a port pin is open, or unconnected, its internal pull-up resistor will cause it to read as 1. If you connect a jumper wire from a port pin to ground, or bring the pin low by driving it with a logic low output, it should read 0. Line 10 in Listing 3-1 brings all of Port 1's bits high, which enables them to be used as inputs.

Writing to Port 1

You can control the bits of Port 1 by writing to them. Listing 3-2 allows you to set or clear individual bits. Here's an example of what happens when you run the program:

```
Enter a bit to set or clear (0-2, 4-7) :7
Enter 1 to set, 0 to clear :0
Enter a bit to set or clear (0-2, 4-7) :3
Do not change bit 3!
```

The program doesn't allow you to change bit 3 (P1.3), because the 8052-BASIC circuit requires this bit to be high when accessing external memory (assuming that you've included U3B in your circuit). If you do clear bit 3 accidentally, you'll crash the system and will have to reboot.

Listing 3-1. Displays the value of each bit in Port 1.

```
10  PORT1 = 0FFH
20  PRINT "PORT 1 Bit Values:"
30  PRINT "Bit 0 = ",(PORT1.AND.1)
40  PRINT "Bit 1 = ",(PORT1.AND.2)/2
50  PRINT "Bit 2 = ",(PORT1.AND.4)/4
60  PRINT "Bit 3 = ",(PORT1.AND.8)/8
70  PRINT "Bit 4 = ",(PORT1.AND.10H)/10H
80  PRINT "Bit 5 = ",(PORT1.AND.20H)/20H
90  PRINT "Bit 6 = ",(PORT1.AND.40H)/40H
100 PRINT "Bit 7 = ",(PORT1.AND.80H)/80H
110  END
```

Listing 3-2. Allows you to set or clear individual bits of Port 1.

```
10 INPUT "Enter a bit to set or clear (0-2, 4-7) :",X
20 IF X=3 THEN PRINT "Do not change bit 3!" : GOTO 10
30 INPUT "Enter 1 to set, 0 to clear :",Y
40 IF Y=1 THEN PORT1=PORT1.OR.2**X
50 IF Y=0 THEN PORT1=PORT1.AND.0FFH-2**X
60 END
```

Run the program and follow the on-screen instructions to set or clear a bit. To monitor a port bit as you set and clear it, you can use a logic probe, voltmeter, or oscilloscope. For example, to monitor bit 0, place a logic probe on pin 1 of U1, or connect the + lead of a voltmeter to pin 1 and the - lead to ground.

Accessing Memory

Listing 3-3 allows you to read and write to external RAM. Here is an example of what happens when you run this program:

```
Enter 0 (read), 1 (write), or 2 (quit): 1
Free memory ranges from 397H to 1FFFH
Enter an address to write to : 1000H
Enter data to be written : 55H
55H has been written to address 1000H
Enter 0 (read), 1 (write), or 2 (quit): 0
External RAM ranges from 0 to 1FFFH
Enter an address to read : 1000H
55H is stored in address 1000H
```

If you write to an address outside the range specified as free memory, you will overwrite the RAM currently in use to store your program and run BASIC-52. If you do this accidentally, your system may crash and you'll have to reset the system and re-enter the program.

If you prefer decimal numbers to hex notation, change each PH0 in the program to PRINT. (PH0. includes a period; PRINT does not.)

Real-time Clock

Listing 3-4 demonstrates BASIC-52's real-time clock by displaying an on-screen 60-second timer.

Listing 3-3. Allows user to read and write to external memory.

```
10 DO
20 INPUT "Enter 0 (read), 1 (write), or 2 (quit): ",RW
30 IF RW=0 THEN GOSUB 70
40 IF RW=1 THEN GOSUB 120
50 WHILE RW<>2
60 END
70 PH0."External RAM ranges from 0 to ",MTOF
80 INPUT "Enter an address to read : ",A
90 B=XBY(A)
100 PH0.B," is stored in address ",A
110 RETURN
120 PH0."Free memory ranges from ",LEN+512," to ",MTOF
130 INPUT "Enter an address to write to : ",A
140 INPUT "Enter data to be written : ",B
150 XBY(A)=B
160 PH0.B," has been written to address ",A
170 RETURN
```

For the timer to be accurate, you must set XTAL to match the timing crystal your system uses.

Further Experiments

Feel free to continue experimenting with BASIC-52 programs, using the programming reference as a guide. You can do quite a bit with just these circuits.

Listing 3-4. Real-time clock.

```
10 CLOCK 1:TIME=0:SEC=0
20 DO
30 ONTIME 1,60
40 WHILE SEC<60
50 END
60 TIME=TIME-1
70 SEC=SEC+1
80 PRINT SEC
90 RETI
```

Listing 3-5. This program uses BASIC-52's GET instruction to detect when the user has pressed a key.

```
10  CLOCK1:TIME=0:SEC=0
20  PRINT "Press any key to quit"
30  DO
40  ONTIME 1,100
50  G=GET
60  UNTIL G<>0
70  END
100 TIME=TIME-1
110 PH0. PORT1
120 RETI
```

Exiting Programs

Some programs, such as Listing 3-3's, continue to run until the user requests to end it. In BASIC-52, there are several ways to detect that the user wants to stop a program.

Set a User Variable

In Listing 3-3, the program displays a menu of choices on the host computer's screen. The program continues to run until the user selects *QUIT* by entering 2, which sets the variable RW to 2 and causes the DO . . . WHILE loop and the program to end.

Use GET

Sometimes, selecting a menu option isn't convenient or appropriate. Listing 3-5 reads and displays the value of PORT1 once per second until the user presses any key at the host computer. The program uses BASIC-52's GET operator to detect a keypress. GET stores the ASCII code of a keypress at the host computer. Setting a variable equal to GET (line 50) causes GET to reset to 0. You can detect a keypress by reading GET periodically. If GET

Listing 3-6. This program will end only when the user presses CONTROL+C.

```
10  CLOCK 1:TIME=0:SEC=0
20  DO
30  ONTIME 1,100
40  WHILE 1=1
50  END
100 TIME=TIME-1
110 PH0. PORT1
120 RETI
```

Listing 3-7. This program ends when $\overline{\text{INT1}}$ (pin 13) is brought low and causes an interrupt routine to execute.

```
10  CLOCK 1:TIME=0:SEC=0
20  A=0
30  PRINT "Bring INT1 (pin 13) low to end program."
40  DO
50  ONTIME 1,100
60  ONEX1 200
70  WHILE A=0
80  END
100 TIME=TIME-1
110 PHO. PORT1
120 RETI
200 A=1
210 RETI
```

doesn't equal zero, it means that a key was pressed. In Listing 3-5, when GET no longer equals 0, the program ends.

Wait for CONTROL+C

You can always end a program by pressing CONTROL+C at the host's keyboard. The only exceptions are runaway programs that have crashed the system and force you to reboot. Listing 3-6 is an expanded version of Listing 3-5. It continues to read and display PORT1 in an endless loop (DO . . . WHILE 1=1), until you press CONTROL+C.

Detect a Switch Press

A final method will end a program without any input from the host's keyboard. You can use this in stand-alone projects that don't connect to a host computer at all. Listing 3-7 ends when the 8052-BASIC's pin 13 ($\overline{\text{INT1}}$) goes low, which causes an interrupt routine to execute. Bring the pin low by jumpering it briefly to GND, or connect a pushbutton switch as described in Chapter 7.

