30

COMPUTER CONTROL VIA PC PRINTER PORT

In the "Wizard of Oz," the Scarecrow laments "If I only had a brain." He imagines the wondrous things he could do and how important he'd be if he had more than straw filling his noggin! In a way, your robot is just like the Scarecrow. Without a computer to control it, your robot can only be so smart. Hardwiring functions into the robot is a suitable alternative to computer control, and you should always look to simpler approaches than immediately connecting all the parts of your robot to a super Cray-2 computer.

Yet there are plenty of applications that cry out for computer control; some tasks, like image and voice recognition, require a computer. One of the easiest ways to connect a robot to a computer is to use an IBM PC compatible. You can readily wire up your robot to the PC's parallel port. The parallel port is intended primarily for connecting the computer to printers, plotters, and some other computer peripherals. With a few ICs and some rudimentary programming, it can also be used to directly control your robots. If your computer has several parallel ports, you can use them together to make a very sophisticated control system.

Despite the many advantages of the computer's parallel port, using it involves some disadvantages too. You are limited to controlling only a handful of functions on your robot because the parallel port has only so many input and output lines—although with some creative design work you can effectively increase that number. Also, most parallel ports are designed primarily to get data *out* of the computer, not into it, though many parallel ports are bidirectional under low-level software control. The average parallel port also has a

number of input-specific lines for directly communicating with a printer or other peripheral, though the number of input lines is small.

This chapter deals primarily with how to use the parallel port on an IBM PC compatible. Why the PC? It's a common computer—hundreds of millions of them are in use today. While the PC comes in many styles, shapes, and sizes, they all do basically the same thing and provide the same specifications for both software and parallel port output. If you don't want to use your main PC for your robot work, you can probably find a secondhand machine for under $100.

Note:

The text that follows pertains to the parallel port on an IBM PC compatible used in standard mode, and not in enhanced, bidirectional mode. On some computers, you may need to modify the system BIOS settings to turn off enhanced and/or bidirectional settings. Otherwise, the port may not behave the way you want it to. You can modify the BIOS settings by restarting your computer and following the "Setup" instructions shown on the screen as the PC boots.

The Fundamental Approach

In the original design of the IBM PC, system input and output—such as the parallel port, serial port, and video display—were handled by "daughter boards" that were plugged into the computer's main motherboard. This design practice continues, though today the average PC compatible comes with features such as parallel and serial port, video display, modem, and even a sound card already built into the motherboard. Whether these features are built into the motherboard or added by plugging in a daughter board, all of are input/output (I/O) ports of one type or another.

The PC accesses its various I/O ports by using an *address* code. Each device or board in the computer has an address that is unique to itself, just as you have a home address that no one else in the world shares with you. Very old IBM PCs and compatibles used a monochrome display adapter board, which included its own parallel port. The printer port on this board used a starting address of 956. This address is in *decimal,* or base-10 numbering form. You may also see PC system addresses specified in *hexadecimal,* or base-16, form. In hex, the starting address is 3BCH (the address is really 3BC; the *H* means that the number is in hex). By convention, the parallel port contained on an I/O expansion board, or built into the motherboard, has a decimal address of 888 (or 378H hex) or 632 (278H).

Parallel ports in the PC are given the *logical* names LPT1:, LPT2:, and LPT3:. Every time the system is powered up or reset, the ROM BIOS (Basic Input/Output System) chip on the computer motherboard automatically looks for parallel ports at these I/O addresses, 3BCH, 378H, and 278H, in that order. (It skips 3BCH if you don't have a monochrome card or printer port installed, which you probably don't unless your machine is *ancient!*). The logical names are assigned to these ports as they are found.

Table 30.1 shows the port addresses for the parallel ports in the PC. Applications software often use the logical port names instead of the actual addresses, but in attaching a robot to the computer we'll need to rely on the actual address—hence the need to go into these details.

TABLE 30.1 ADDRESSES OF PARALLEL PORTS.

ADAPTER	DATA	STATUS	CONTROL
Parallel port on monochrome display card	3BCH, 956D	3BDH, 957D	3BEH, 958D
PC/XT/AT printer adapter	378H, 888D	379H, 889D	37AH, 890D
Secondary LPTx card (as LPT2:)	278H, 632D	279H, 633D	27AH, 634D
"H" Suffix < Hex			
"D" Suffix 5 Decimal			

The PC parallel port is a 25-pin connector, which is referred to as a DB-25 connector. Cables and mating connectors are in abundant supply, which makes it easy for you to wire up your own peripherals. You can buy connectors that crimp onto 25-conductor ribbon cable or connectors that are designed for direct soldering. Fig. 30.1 shows the pinout designations for the connector (shown with the end of the connector facing you). Note that only a little more than half of the pins are in use. The others are either not connected inside the computer or are grounded to the chassis. Table 30.2 shows the meaning of the pins.

Notice that not one address is given, but three. The so-called starting address is used for *data output register.* The data output register is comprised of eight binary weighted bits, something on the order of 01101000 (see Fig. 30.2). There are 256 possible combinations of the eight bits. In a printer application, this means that the computer can send specific code for up to 256 different characters. The data output pins are numbered 2 through 9. The bit positions and their weights are shown in Table 30.3.

The other two registers of the parallel port, have different addresses (base address of the port, plus either one or two). These registers are for *status* and *control.* The most commonly used status and control bits (for a printing application, anyway) are shown in Fig. 30.3 on page 464. The function of the status and control bits is shown in Table 30.4 on page 465.

To a printer, one of the most important control pins is pin number 1. This is the STROBE line, which is used to tell the peripheral (printer, robot) that the parallel data on lines 2 through 9 is ready to be read. The STROBE line is used because all the data may not arrive at their outputs at the same time. It is also used to signal a change in state. The output lines are latched, meaning that whatever data you place on them stays there until you change it or turn off the computer. During printing, the STROBE line toggles HIGH to LOW and then HIGH again. You don't have to use the STROBE line when commanding your robot, but it's a good idea if you do.

Other control lines you may find on parallel printer ports include the following (some of these lines aren't always implemented):

Auto form feed
Select/deselect printer

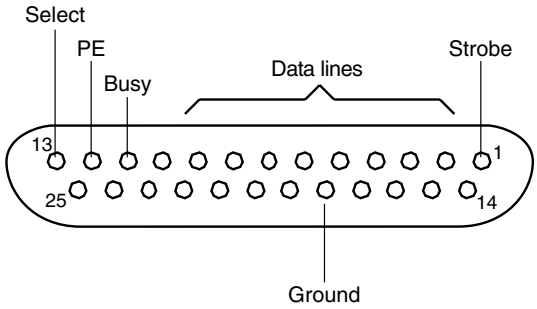


FIGURE 30.1 **Pinout of the DB25 parallel port connector, as used on IBM PC-compatible computers.**

TABLE 30.2 PARALLEL PORT PINOUT FUNCTIONS.

PIN	FUNCTION (PRINTER APPLICATION)
1	Strobe
2	Data bit 0
3	Data bit 1
4	Data bit 2
5	Data bit 3
6	Data bit 4
7	Data bit 5
8	Data bit 6
9	Data bit 7
10	Acknowledge
11	Busy
12	OE (out of paper, or empty)
13	Printer online
14	Auto line feed after carriage return
15	Printer error
16	Initialize printer
17	Select/deselect printer
18–25	Unused or grounded

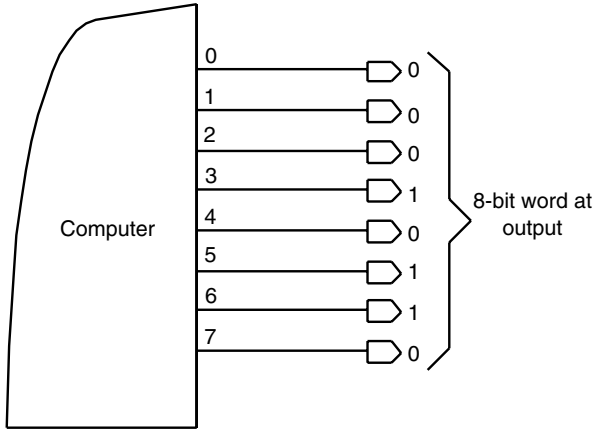


FIGURE 30.2 The parallel port outputs eight bits at a time.

TABLE 30.3 BIT POSITION WEIGHTS.

BIT POSITION		WEIGHT
D7	<	128
D6	<	64
D5	<	32
D4	<	16
D3	<	8
D2	<	4
D1	<	2
D0	<	1

Initialize printer
Printer interrupt

Traditionally, the status lines are the only ones that feed back into the computer (as mentioned earlier, most parallel printer ports are now bidirectional, but this is not a feature we'll get into this time around). There are five status lines, and not all parallel ports support every one. They are as follows:

Printer error
Printer not selected
Paper error

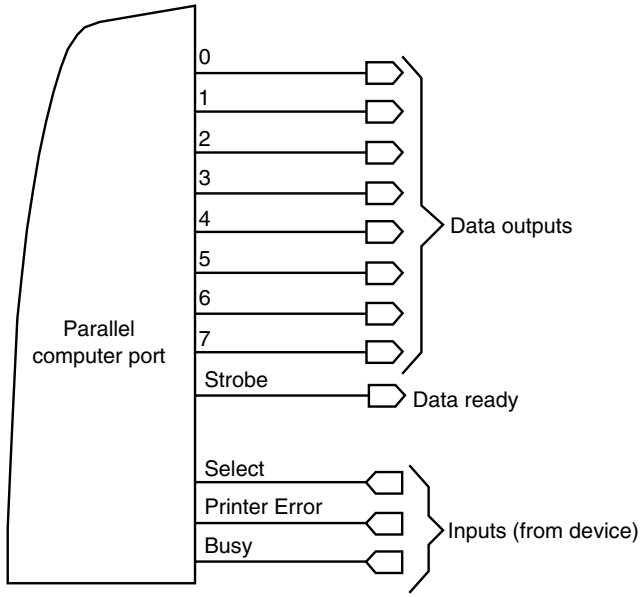


FIGURE 30.3 **The minimum parallel port: eight data outputs, a STROBE (Data Ready) line, and inputs from the printer, including Select, Printer Error, and Busy.**

Acknowledge
Busy

The acknowledge and busy lines are commonly used for the same thing in a printer application. However, depending on the design of the port in your computer, you can use the two separately in your own programs. (One helpful tidbit: for a printing application when the BUSY line is LOW, the ACK line is HIGH.)

Robot Experimenter's Interface

It's not generally a good idea to connect robot parts directly to a parallel port because wiring mistakes in the robot could damage the circuitry in your PC. Moreover, the parallel port in your PC may not have the drive current needed to directly operate relays, solenoids, and power transistors. By using an interface, discussed later in this chapter, you can help protect the circuitry inside your computer and provide more drive current for operating robotic control devices. This interface, called the Robot Experimenter's Interface for lack of a better name, lets your PC control up to 12 robotic functions (such as motors) and read the values of up to four robotic switches or other digital sensing devices.

TABLE 30.4 PARALLEL PORT STATUS AND CONTROL BITS.

CONTROL BITS	
BIT	**FUNCTION**
0	LOW < normal; HIGH < output of byte of data
1	LOW < normal; HIGH < auto linefeed after carriage return
2	LOW < initialize printer; HIGH < normal
3	LOW < deselect printer; HIGH < select printer
4	LOW < printer interrupt disables; HIGH < enabled
5–7	Unused
STATUS BITS	
BIT	**FUNCTION**
0–2	Unused
3	LOW < printer error; HIGH < no error
4	LOW < printer not on line; HIGH < printer on line
5	LOW < printer has paper; HIGH < out of paper
6	LOW < printer acknowledges data sent; HIGH < normal
7	LOW < printer busy: HIGH < out of paperH

Caution:

Any time you mess around with a computer there is a risk of damaging it, and this goes for the circuit presented next. This is not a project you should consider if you're new to electronics and aren't sure what you're doing.

CONSTRUCTING THE INTERFACE

The schematic diagram for the Robot Experimenter's Interface is shown in Fig. 30.4. You can build it in under an hour, and it requires very few components. The interface uses a solderless experimenter's breadboard so you can create circuits right on the interface. The input and output buffering is provided by the 74367 hex buffer driver. Three such chips are used to provide 18 buffered lines, which is more than enough.

You may wish to build the interface in an enclosure that is large enough to hold the breadboard and the wire-wrapping socket. Make or buy a cable using a male DB-25 connector and a four- or five-foot length of 25-connector ribbon cable. Solder the data output, status, and control line conductors to the proper pins of the 74367 ICs. Route the outputs to the bottom of the wire-wrap socket. A finished interface should look something like the one in Fig. 30.5. Using the interface requires you to provide a 5 vdc source. *Do not* try to power the interface from the parallel port! Use a length of 22 AWG solid

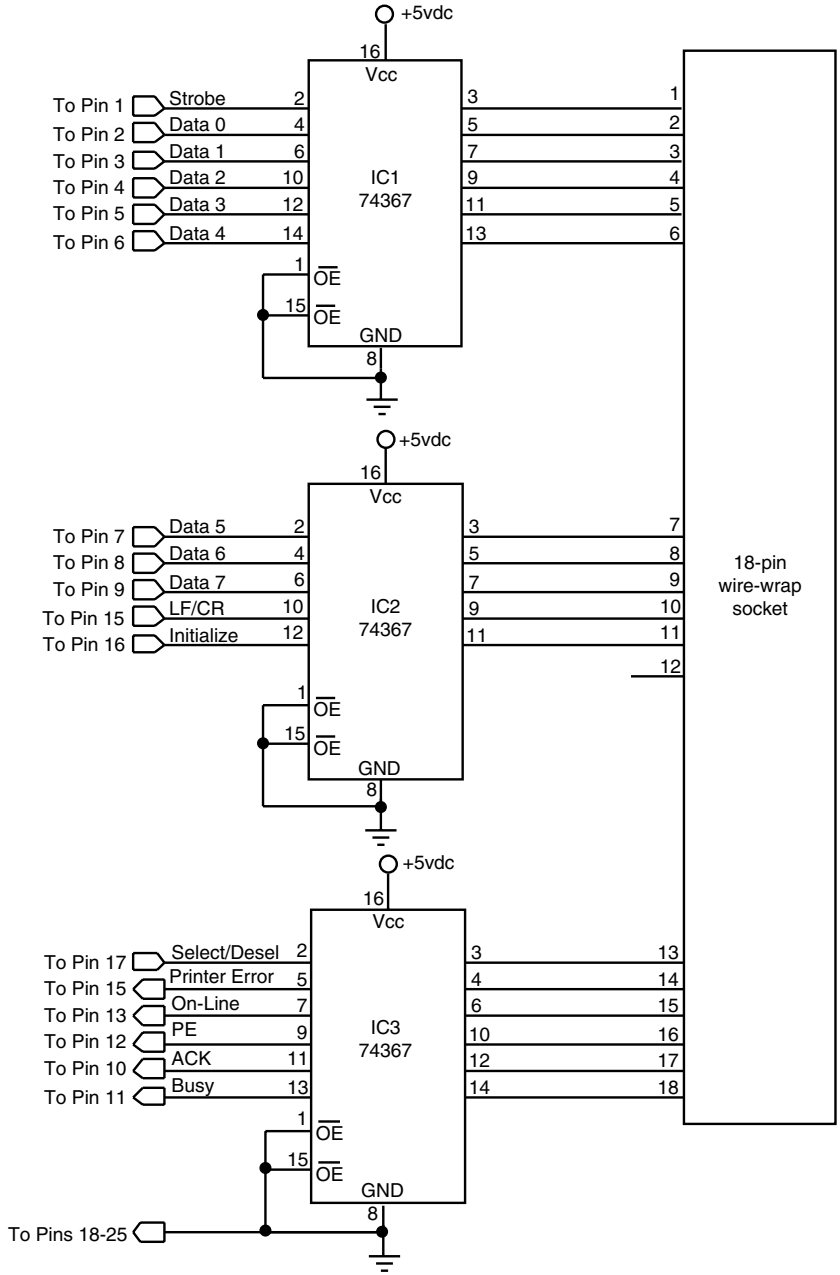


FIGURE 30.4 Schematic for the Robot Experimenter's Interface.

TABLE 30.5 PARTS LIST FOR THE ROBOT EXPERIMENTER'S INTERFACE.	
IC1–IC3	74367 TTL Hex Inverter/Buffer IC
Misc	18-pin wire-wrap socket, solderless experimenter's board, binding posts (for power connection), enclosure

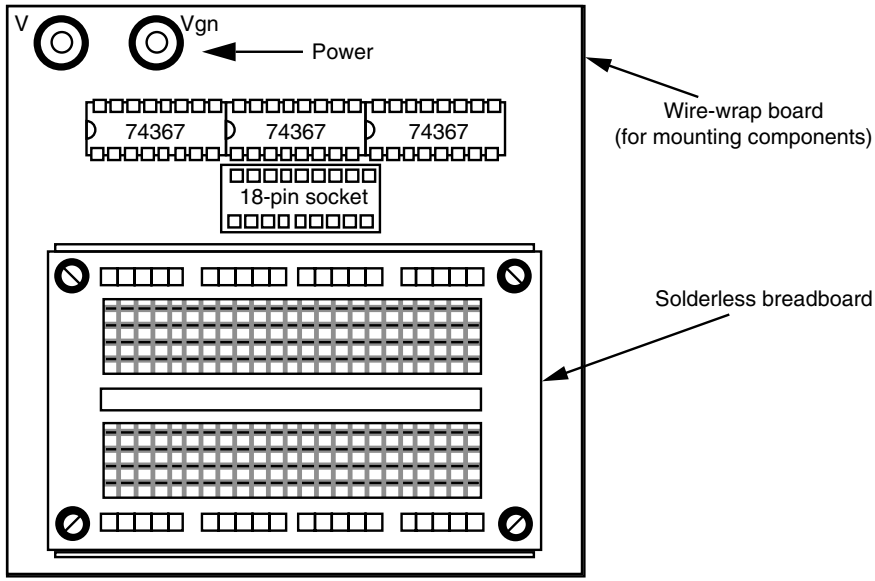


FIGURE 30.5 **The completed Robot Experimenter's Interface. Mount the breadboard, wire-wrap socket, ICs, and power terminals on a perf board, and secure the board into a project case.**

conductor wire to connect the signals at the wire-wrap socket to whatever points on the breadboard you desire.

TESTING THE INTERFACE

The first order of business is to connect the Robot Experimenter's Interface as shown in Fig. 30.6. Connect the cable to the parallel port of your computer (some of the LEDs will light). Use a DOS-based Basic interpreter program to manipulate the three registers (data, control, and status) of the parallel port. Most older PCs will have a Basic interpreter either built into the BIOS (as was the case with the original IBM PC) or provided as a separate .com or .exe executable file. If your PC has MS-DOS 5.0 or later, look for QBasic, an updated version of the venerable Microsoft Basic from the late 1970s. All of the program examples in this chapter assume you're using QBasic, or a similar updated Basic variant.

Note that if you're using Microsoft Windows 95 or later, QBasic probably isn't installed on your computer, but it is provided on the Windows CD-ROM. Look for the *qbasic.exe*

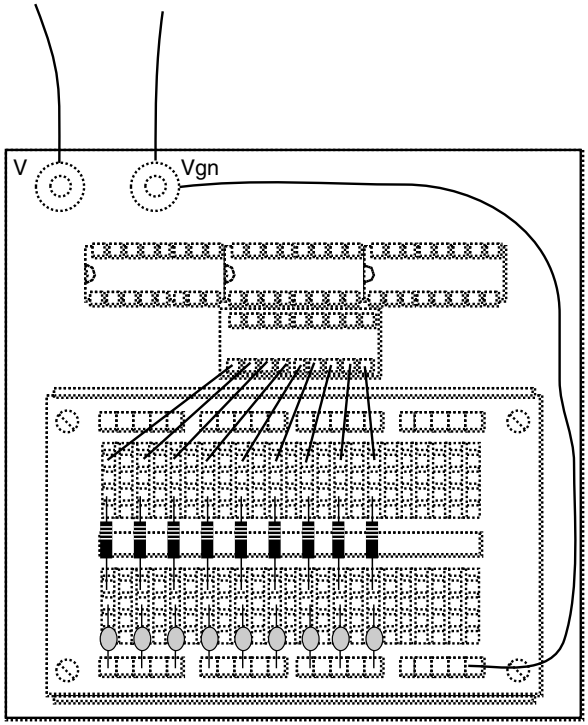


FIGURE 30.6 **Component arrangement for testing the Robot Experimenter's Interface.**

program file in the *OtherOldmsdos* directory, or visit Microsoft's Web page at www.microsoft.com for additional information.

Type the program shown in Listing 30.1. The program assumes you're using the standard LPT1: port, which has an address of 888 decimal (378 hex). If you're using a different parallel port, change the *BaseAddress* as required. Refer to Table 30.1 earlier in this chapter. You're now ready to run the program (in QBasic, press Shift F5).

LISTING 30.1.

```
BaseAddress = 888          ' Base address of parallel port
DataPort = BaseAddress     ' Address of data register

FOR Count = 0 TO 255
    OUT DataPort, Count
FOR x = 1 TO 500: NEXT x
NEXT Count
```

The LEDs connected to each of the data lines should flash on and off very rapidly. Some of the LEDs will flash more than the others; this is normal. When the program finishes all of the LEDs should stay lit. If the LEDs do not flash, recheck your wiring and make sure

the program has been typed correctly. The LEDs that are on represent a logic 1 state; those that are off represent a logic 0 state.

Fig. 30.7 is a blank dotted-line version of the Robot Experimenter's Interface. Feel free to use it to sketch out your own designs.

Using the Port to Operate a Robot: The Basics

The 74367 used in the Robot Experimenter's Interface cannot sink or source more than about 20 mA of current per output, and as you would expect you can't operate a motor directly from it. However, it can drive a low-power relay, transistor, or H-bridge. See Chapter 18, "Working with DC Motors," for some popular ways to bridge the low-level output of the interface to control a real-world robot.

The simplest way to operate your robot via computer is to connect each of the data output lines to a suitable transistor, small relay, or H-bridge input. You can control the on/off state of up to eight motors or other devices using just the eight data lines of the parallel port (you can actually control even more devices; more on this in a bit).

Let's say that you only have three motors connected to the interface and that you are using lines 0, 1, and 2 (pins 2, 3, and 4, respectively). To turn on motor 1, you must

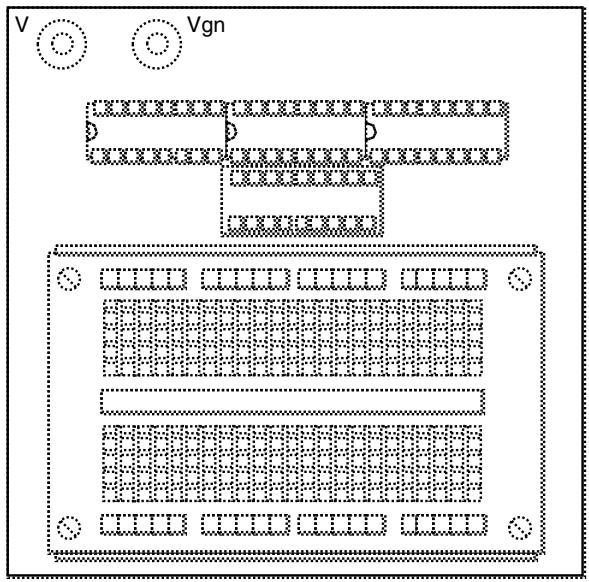


FIGURE 30.7 **A blank Robot Experimenter's Interface layout. Feel free to copy it and use it to make your own designs.**

activate the bit for line 0, that is, make it HIGH. To do this, output a *bit pattern* number to the port using the BASIC OUT command. The OUT command is used to send data to an I/O port. The command is used with two parameters: *port address* and *value.* The two are separated by a comma. For port address, use the base address of the parallel port; for data, use the value you want to send to the port. Here's an example:

```
OUT 888, 10
```

(Note: In the test code you used variables, *BaseAddress* and *DataPort,* instead of "hard-wired" literal values for the port address. It's a better practice to use variables because that makes it easier to change your program. For right now, however, I'll use literal values such as *888* for short examples, but revert back to using variables in the larger ready-to-go program code.)

The base address is 888, and the value is decimal 10. Table 30.6 shows the first 16 binary numbers and the bit pattern that constitutes them.

For most robotic applications where you use the parallel port to control motors, you'll need two data lines for each motor: one to turn the motor on and off and another to con-

TABLE 30.6 DECIMAL AND BINARY EQUIVALENTS (0–15 ONLY).

DECIMAL	BINARY
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

trol its direction. You can use the four bits in Table 30.6 to control the on/off state of the motors as well as their direction. For this you might use data lines 0, 1, 2, and 3 (pins 2, 3, 4, and 5, respectively, of the interface).

Table 30.7 lists all the possible bit patterns for data lines 0–3. You can connect the motor relays to the pins in any order, but the table assumes the following:

Data line 0 (bit 1) controls the On/Off relay for motor 1
Data line 1 (bit 2) controls the On/Off relay for motor 2
Data line 2 (bit 3) controls the Direction relay for motor 1
Data line 3 (bit 4) controls the Direction relay for motor 2

You should get into the habit of initializing the port at the beginning of the program by outputting decimal 0. That prevents the motors from energizing at random. The line of code for this is as follows:

TABLE 30.7 DATA BITS FOR CONTROLLING TWO MOTORS.

		MOTOR1		MOTOR 2	
BINARY	DECIMAL VALUE	CONTROL (BIT 1)	DIRECTION (BIT 3)	CONTROL (BIT 2)	DIRECTION (BIT 4)
0000	0	Off	Forward	Off	Forward
0001	1	On	Forward	Off	Forward
0010	2	Off	Forward	On	Forward
0011	3	On	Forward	On	Forward
0100	4	Off	Reverse	Off	Forward
0101	5	On	Reverse	Off	Forward
0110	6	Off	Reverse	On	Forward
0111	7	On	Reverse	On	Forward
1000	8	Off	Forward	Off	Reverse
1001	9	On	Forward	Off	Reverse
1010	10	Off	Forward	On	Reverse
1011	11	On	Forward	On	Reverse
1100	12	Off	Reverse	Off	Reverse
1101	13	On	Reverse	Off	Reverse
1110	14	Off	Reverse	On	Reverse
1111	15	On	Reverse	On	Reverse

```
OUT 888, 0
```

To activate just motor 1, choose a decimal number where only the first bit changes. There is only one number that meets that criterion: it is decimal 1, or 0001 (we will ignore bits 5 through 8 for this discussion since they are not in use). So type:

```
OUT 888, 1
```

Run this program; motor 1 turns on. To turn it off, send a decimal 0 to the port, as described earlier. You use the same technique to turn on motor 2 or both motors 1 and 2 at the same time. To turn on both motors at the same time, for example, look for the binary bit pattern where the first and second bits are 1 (it's decimal 3), and output this value to the port.

Controlling a Two-wheel Robot

Controlling the common two-wheeled robot is a simple matter of sending the right bit patterns to the parallel port. Note that binary 0000 (decimal 0) turns off both motors, so the robot stops. Changing the binary bit pattern activates the right or left motor and controls its direction. Table 30.8 lists the most common bit patterns you will use.

When writing the control program for your robot you may find it necessary to insert short pauses between each state change (motor 1 forward and reverse, for example). You can create simple pauses in Basic with "do nothing" *FOR...NEXT* loops as shown in the testing program in Listing 30.2. The program first resets all bits to 0, then sleeps (waits) one second. The program then goes through a timed routine turning on different motors and reversing their direction: forward, reverse.

Note that do-nothing *FOR...NEXT* loops are processor-speed dependent. Adjust the value of one or both loops to control the actual delay for your computer. You may also wish to use the *SLEEP* statement, which inserts a delay for the number of seconds you specify. Other versions of Basic provide for additional time-delay commands. Most Basic programming environments, such as Microsoft QBasic (QuickBasic), allow you to terminate

TABLE 30.8 COMMON BIT PATTERNS FOR CONTROLLING TWO MOTORS.

BINARY	DECIMAL	FUNCTION
0000	0	All stop
0011	3	Forward
1111	15	Reverse
0010	2	Right turn
0001	1	Left turn
0111	7	Hard right turn (clockwise spin)
1011	11	Hard left turn (counterclockwise spin)

the program at any time by pressing Ctrl Break (break is the Pause/Break key, usually located near the numeric keypad).

LISTING 30.2.

```
DECLARE SUB DelaySub ()
BaseAddress = 888                  ' Base address of parallel port
DataPort = BaseAddress             ' Address of data register

OUT DataPort, 0
SLEEP 1
OUT DataPort, 3
DelaySub
OUT DataPort, 15
DelaySub
OUT DataPort, 2
DelaySub
OUT DataPort, 1
SLEEP         2
OUT DataPort, 0

SUB DelaySub
' adjust delay as necessary
FOR DELAY = 1 TO 100000: NEXT DELAY
END SUB
```

Controlling More Than Eight Data Lines

As shown in the previous examples each motor requires two bits. Therefore, one parallel port can control the action and direction of four motors. However, you can actually control more motors (or other devices) by using a number of simple schemes and without resorting to using additional parallel ports.

The most straightforward method for expanding a single parallel port is to make use of some or all of the data lines of the control register. You send bits to these control lines in exactly the same way as you send bits to the data output lines, except that you use a different address. For the standard LPT1: port at decimal 888, the decimal address for the control lines is 890. Only the first five bits of the address are used in the port, which means the decimal numbers you use will be between 0 and 31.

Let's say you are using bit 2 of the control address (in a printer application, bit 2 is used to initialize the printer). You turn that bit on—and no others—by entering the following program line:

```
OUT 890, 4
```

Note that you can output a binary pattern to address 890, and it will not affect the data output lines.

USING EXPANDED IO

Another way of increasing the number of controlled devices is to use a data demultiplexer. There are several types in both the TTL and CMOS IC families. A popular data

demultiplexer (or "demux") is the 74154. This chip takes four binary weighted input lines (1, 2, 4, 8) and provides 16 outputs. Only one output can be on at a time. See the schematic in Fig. 30.8 to see how to hook it up. The IC is shown connected to the first four data output lines of the parallel port. You can actually connect it to any four, and you don't even have to use all four lines. With just three lines, the demux allows you to control up to eight devices.

To select the device connected to the number 3 output of the demux, for example, you apply a binary 3 (0011) to its input lines. Write the line as follows:

```
OUT 888, 3
```

A limitation of the demux is that you can't control more than one device connected to it at any one time. You can't, for example, attach both drive motors to the demux outputs and have

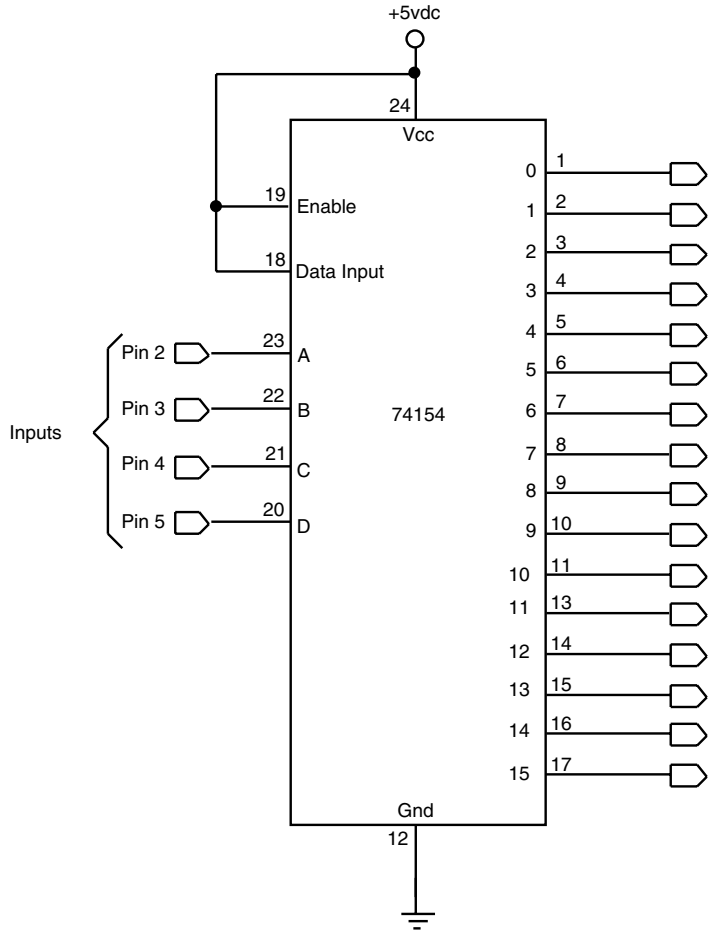


FIGURE 30.8 Basic wiring diagram for the 74154 demultiplexer chip.

them on at the same time. There will be many times, however, when your robot will only be doing one thing (such as triggering an ultrasonic ranger). In these cases, the demux is perfect.

EXTERNAL ADDRESSING

As mentioned earlier in this chapter, all sorts of data and control lines are inside the computer, on the microprocessor bus. There is also a set of special-purpose lines, the address lines, that are used to pass data to specific devices and expansion boards. For example, you address the data output lines of the parallel port by sending out the address 888.

The address for the parallel port triggers just the parallel port, but with some ingenuity (and no extra components) you can wire up a "subaddress" scheme so the one parallel port can fully control a very large number of devices. This is the third and most sophisticated way to sap all the power out of the parallel port.

You can disable the 74367 hex buffer IC, which is used to link the port to the outside world. In the Robot Experimenter's Interface, the ENABLE lines of the chip, pins 1 and 15, are held LOW by tying them to the ground, so data is passed from the input to the output. When the ENABLE pins are brought HIGH, the outputs are driven to a high-impedance state and no longer pass digital data. In this way, the 74367 acts as a kind of valve. The two ENABLE lines control different input/output pairs, as shown in Fig. 30.9. The high-impedance disabled state is engineered so that many 74367 chips can be paralleled on the same data lines, without loading the rest of the circuit.

You can use the ENABLE pins of the 74367 and a few of the unused control lines in the parallel port to make yourself an electronic data selector switch. In operation, you output a binary word onto the data output lines. You then send the word to the desired device by addressing it with the control lines.

Here is an example: Let's say that you have connected three subaddress ports to the parallel printer port, as shown in Fig. 30.10. Control lines 1, 2, and 3 are connected to the ENABLE pins of the 74367. The inputs of the three 74367s are connected together. The outputs of each feed to the specific device.

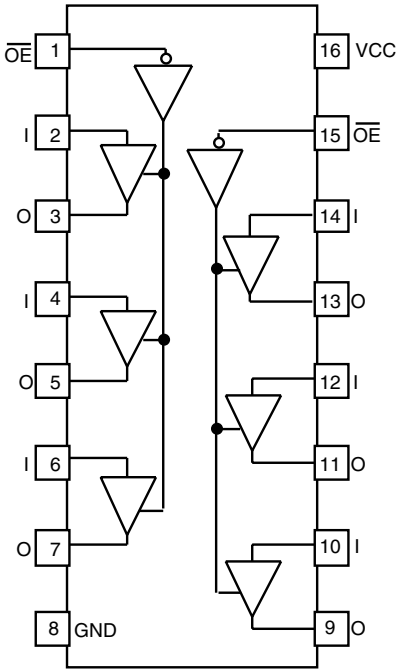
To turn on bits 0 and 1 on device 2, enter the following lines into Basic and run the program:

```
OUT 888, 3
OUT 890, 2
```

The first line of the program outputs a decimal 3 to the data output register. That places the binary bit pattern 00000011 on the parallel port data output lines. The next line enables device 2 because it turns on the second 74367.

Inputting Data

Recall that a parallel port has a third and final input register for providing status. Most parallel ports support four or five status lines, which you can use to input data back into the computer. The Robot Experimenter's Interface uses the four status lines you are likely to find in any parallel port. To read data from the port, you use the Basic *INP* statement (*INP* for input). The input command is used as follows:



Inputs		Outputs
$\overline{OE}$	I	O
L	L	L
L	H	H
H	X	HI-Z

Truth table -- 74367

FIGURE 30.9 **The internal configuration of the 74367 chip. Note the two independent ENABLE lines, on pins 1 and 15.**

```
Y=INP(x)
```

In place of x you put the decimal address of the port you want to read. In the case of the main printer port at starting address 888, the address of the status register is 889. The Y is a variable used to store the return value for future use in the program. For testing, you can PRINT the value of Y, which shows the decimal equivalent of the binary bit pattern on the screen.

Listing 30.3 is a sample program that displays the current values of the four inputs connected to the Robot Experimenter's Interface. The values are shown as 0 ("false") and -1 ("true"). Bear in mind that the Busy and Online lines are *active-low*; therefore their logic is the reverse of the others. The code in the test program "compensates" for the active-low condition by reversing the logic in the *If* expressions.

Also note the less-than-straightforward method for determining if pins 15 and 12 are triggered exclusively. These extra *If* tests are needed because the parallel port (most, anyway) will automatically bring pin 12 HIGH if pin 15 is brought HIGH. Weirdness is also

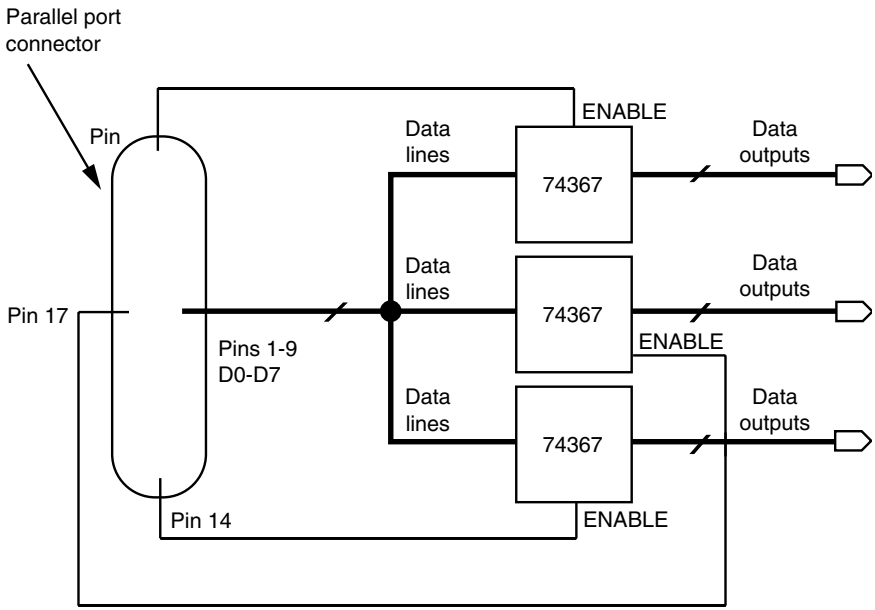


FIGURE 30.10 **Block diagram for a selectable parallel port, using three 74367 ICs to independently control three separate devices.**

encountered if pin 15 is brought HIGH while trying to read the values of pins 10 and 11. The port reads pins 10 and 11 as LOW, even though they may be HIGH on the interface. Again, this is the action of pin 15 (printer error), and for this reason, it's usually a good idea to limit its use or to ensure that the values of other inputs are ignored whenever pin 15 is HIGH.

LISTING 30.3.

```
DIM BaseAddress AS INTEGER, StatusPort AS INTEGER
DIM DataPort AS INTEGER, ControlPort AS INTEGER
DIM x AS INTEGER, Count
AS INTEGER
BaseAddress = 888
DataPort = BaseAddress
StatusPort = BaseAddress + 1
ControlPort = BaseAddress + 2

WHILE (1)
     x = INP(StatusPort) + 1
     IF (x AND 64) = 64 THEN
         PRINT "Pin 10: 1"
     ELSE
         PRINT "Pin 10: 0"
     END IF
      IF (x AND 128) <> 128 THEN
         PRINT "Pin 11: 1"
      ELSE
```

```
            PRINT "Pin 11: 0"
        END IF
        IF ((x AND 16) = 0) AND ((x AND 8) = 0) THEN
            PRINT "Pin 12: 1"
        ELSE
            PRINT "Pin 12: 0"
        END IF
        IF ((x AND 32) = 32) AND (x AND 8) = 8 AND (x AND 16) = 16 THEN
            PRINT "Pin 15: 1"
        ELSE
            IF ((x AND 32) = 0) AND (x AND 8) = 0 THEN
                PRINT "Pin 15: 1"
            ELSE
                PRINT "Pin 15: 0"
            END IF
        END IF
        PRINT "": PRINT ""
        FOR Count = 1 TO 10000: NEXT Count
        CLS
WEND
```

Before moving on, notice the use of the *DIM* keyword in the program shown in Listing 30.3. The *DIM* (for "dimension") keyword tells Basic what kind of variables are used in the program. While using *DIM* is not absolutely mandatory (in QBasic and later), you'll find that adopting it in your programs will not only help reduce errors and bugs. Most of all, it will make your programs run *much* faster. Without the *DIM* keyword, the Basic interpreter creates an all-purpose "variant" variable type that can hold numbers of different sizes, as well as strings. Every use of the variable requires Basic to rethink the best way to store the variable contents, and this takes time.

A Practical Application of the Parallel Port Input Lines

You can use the status bits for the robot's various sensors, like whiskers, line-tracing detectors, heat and flame detectors, and so forth. The simple on/off nature of these sensors makes them ideal for use with the parallel port. Listing 30.4 shows a simple demonstrator program that turns two drive motors forward until either switch located on the front of the robot is activated. Upon activation of either switch, the robot will back up for one second, spin on its axis for two seconds, then go forward again.

The demonstrator program is an amalgam of techniques discussed previously in this chapter. The program assumes you have a two-wheel robot of the type described earlier in the chapter, with the motors controlled according to the definitions in Table 30.8. Whisker or bumper switches are attached to pins 10 and 11.

LISTING 30.4.

```
DECLARE SUB GetAway ()
DIM BaseAddress AS INTEGER, StatusPort AS INTEGER
DIM SHARED DataPort AS INTEGER
```

```
DIM ControlPort AS INTEGER
DIM x AS INTEGER, Count AS INTEGER

BaseAddress = 888
DataPort = BaseAddress
StatusPort = BaseAddress + 1
ControlPort = BaseAddress + 2

CLS
PRINT "Press Ctrl+Break to end program..."

WHILE (1)
     OUT DataPort, 3              ' drive forward
     x = INP(StatusPort) + 1      ' read sensors
     IF (x AND 64) = 64 THEN      ' if sensor 1 active
        GetAway
     END IF
     IF (x AND 128) <> 128 THEN   ' if sensor 2 active
        GetAway
     END IF
      FOR Count = 1 TO 500: NEXT Count
WEND
```

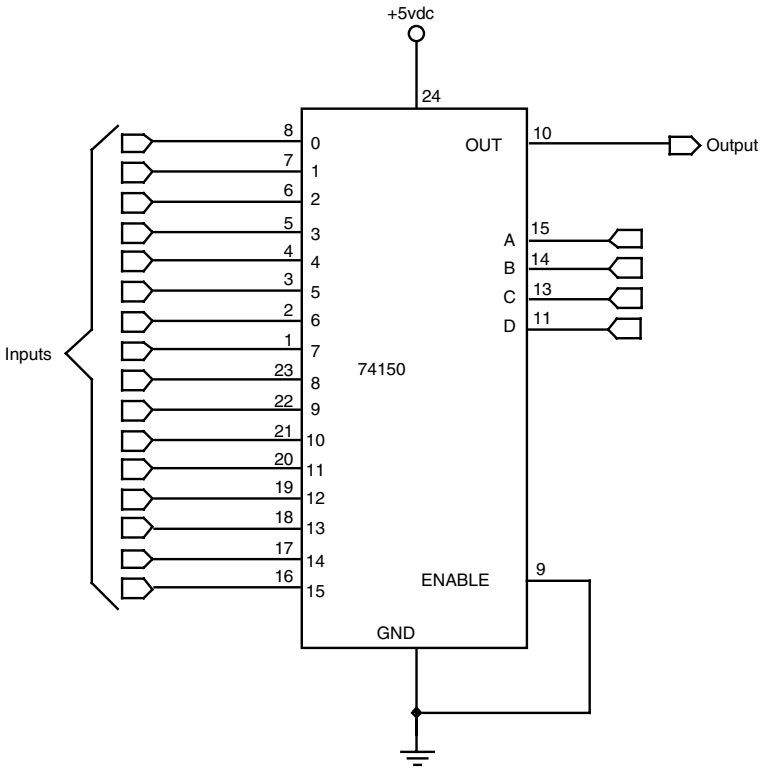


FIGURE 30.11 Basic wiring diagram for the 74150 multiplexer chip.

```
SUB GetAway
OUT DataPort, 15      ' back up
SLEEP 1               ' wait one second
OUT DataPort, 7       ' hard left turn
SLEEP 2               ' wait two seconds
END SUB
```

Expanding the Number of Inputs

Normally, you can have up to five sensors attached to the parallel port (though many ports only support three or four inputs, depending on their specific design). However, by using the ENABLE pins of the buffers in the 74367 chips, it is possible to select the input from a wide number of sensors. For example, using just four control lines with a 74150 data selector means you can route up to 16 sensors to the parallel port. See Fig. 30.11, above, for a pinout diagram of the 74150.

From Here

To learn more about…	***Read***
Computers and microcontrollers for robots	Chapter 28, “An Overview of Robot ‘Brains’”
Connecting computers and microcontrollers to “real-world” devices such as motors and sensors	Chapter 29, “Interfacing with Computers and Microcontrollers”
Using remote control to activate your robot	Chapter 34, “Remote Control Systems”
Using sensors to aid in robot navigation	Part 6, “Sensors and Navigation”