Sending Data between the Computer and Control Panels

The purpose of this document is to familiarize you with the strategies that we use to integrate our control panels, doors, and other electronic equipment with the computers and software on which we run our simulations. It is not designed to be a primer on basic electronics. These already exist. The best ones that I have found to date are:

http://www.kpsec.freeuk.com/index.htm
http://ourworld.compuserve.com/homepages/Bill_Bowden/page6.htm
http://www.play-hookey.com/digital/basic_gates.html

Our goals are to understand:

1) how to send signals to the computer and how receive signals from the computer
2) how the computer interprets the signals we send it
3) how to use different circuit components to manage these signals

Question 2: How Computers Manage Numbers

To understand how the computer interprets the signals going to and from our control panels, we need to know computers handle numbers. All of the signals that are control panels send to the computer or that the computer sends to our control panels are interpreted as numbers by the computer. Computers and people use numbers in much the same way, the only difference is the how many different numerals are used. We use 10 different numerals (0,1,2,3,4,5,6,7,8,9), computers use only 2 (0,1). Our numbers are ***decimal*** (10 numerals), the computer's are ***binary*** (2 numerals) The reason for using two numerals is that each corresponds to an electrical state in a circuit: ***on*** or ***off***, or if you want to think in terms of voltage, ***5 volts*** or ***0 volts***.

All of the signals (messages) that get passed around inside a computer are in the form of voltages on a wire. Cmponents in the computer (the central processing unit and the video card, for example) are connected to each other by wires (usually in the form of little lines of metal on a circuit board rather than loose plastic covered wires). The component sending a message does so by setting the voltage of the wire to either 5 volts (the computer equivalent of the number "1") or 0 volts (the computer equivalent of "0"). The component receiving the message reads it by registering the wire voltage. The information in the message is encoded in the pattern of voltage pulses: 5V, 0V, 5V, 5V, 0V, 5V, 0V, 0V,... rather like morse code. In terms of math, the computer's circuits interpret these pulses as 1,0,1,1,0,1,0,0,...

There is no special reason for using 5 as opposed to any other voltage; 5 volts is the voltage that most of the circuit components that we use were designed to work with. Lower voltages do not provide enough energy to power the component and higher voltages damage them. Thus, only two voltages are available, and so there are only two numerals for counting. In electronics, the negative voltage of a source (battery, generator etc.) is the zero voltage or ground voltage, and the positive voltage is the high voltage.

The circuit in figure 1 is a circuit in which the lamp represents the value of a one-digit binary number. The switch changes the value of the digit from 1 (high voltage) to 0 (low or ground voltage). The lamp only lights up when the switch is on the high voltage. The switch could just as easily switch from on (closed on +5V) to off (open). However, most of the integrated circuit chips that we use (more on these later) do

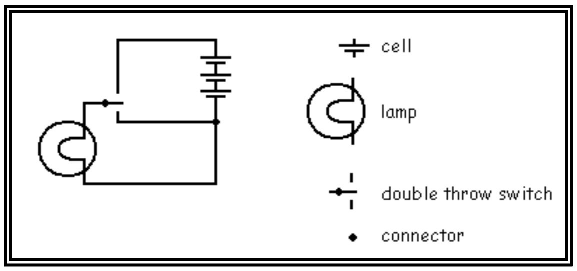


Figure 1
A circuit with a lamp indicating the state of a single digit binary number. The switch either contacts the + voltage wire or the - voltage wire. When lamp is on, this represents the numeral 1, when the lamp is off, this represents the numeral 0.

not respond well to the open state. If they are getting neither a high or low voltage, their voltage tends to drift unpredictably. So, we give them an unambiguous voltage: 0V or +5V.

What is Moving in a Circuit

Electronics people think of positive charges moving around a circuit from the positive terminal of a battery (the longer of the two lines) to the negative terminal. Thus, the negative terminal is the zero-voltage point in the circuit (ground). The reason for this is that the first electrical circuits predate the discovery of the electron. These early electrical scientists had a 50% chance of picking the correct charge to move and they picked incorrectly (they did correctly conclude that only one charge moved). As far as the math is concerned, it makes no difference which you choose. In fact, in the semiconductor circuits, it does make sense to think of moving positive charges (more on this in another section).

Binary Numbers

In a number, each digit occupies a place in the number. Each place in a number corresponds to a multiplier which is a base raised to an exponent. The place multiplier is multiplied by the value of the numeral in that place and the product is added to the total value of the number.

In decimal numbers, the base used is ten. In binary numbers, the base is 2.

236 is a decimal number.

place	10 000s place	1000s place	100s place	10s place	1s place	total
multiplier	10^4	10^3	10^2	10^1	10^0	
numeral	0	0	2	3	6	
place value	0	+ 0	+ 200	+ 30	+ 6	= 236

With only two numerals, binary numbers require more digits to represent a given value than a decimal number.

11101100 is a binary number that represents the same value as 236 in decimal.

place	128	64	32	16	8	4	2	1	total
multiplier	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
numeral	1	1	1	0	1	1	0	0	
place value (decimal)	128	+ 64	+ 32	+ 0	+ 8	+ 4	+ 0	+ 0	= 236

Digits and Wires

In an electrical circuit, each digit of a number corresponds to the voltage of a wire. If you have a bundle of 8 wires, the biggest number that you can represent is an 8 digit binary number: any value from 0 to 255 (decimal), 0 to 11111111 (binary). Eight binary digits used to be the maximum size of number that computers could pass between the different components of the computer. In computer talk, the word '***bit***' is used to describe an individual digit. A group of bits that make up a number is referred to as a '***byte***.' Again, for historical reasons, the term byte almost always refers to 8 bits. The largest package of bits that the computer can handle as a single package is referred to as a '***word***.'

The reason that all this is important to us at space sim is because of the method that we use to communicate between computer and control panels.
The input/output (I/O) device that we use is the ***printer port***. This also is called the ***parallel port*** because it has 8 data wires so that it can pass numbers of 8 digits all at once along its 8 separate wires. The parallel port is what is known as a legacy port: it is installed on all computers but is rarely used these days except by older hardware (like us).

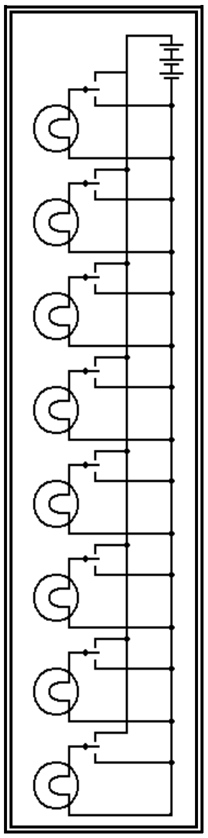


Figure 2
A circuit to represent a 8-bit binary number

Serial ports and USB ports are more modern. They have fewer wires, so that they must send numbers as a sequence of digits one at a time. This makes life more complicated because the computer and hardware have to coordinate the passing of data very precisely. The parallel port sets voltages on the 8 wires and waits until the computer or hardware are ready to read the data.

The parallel port has 3 sets of 8 wires: 3 I/O bytes. One set, the data byte, is used to send character codes (each number between 0 and 255 specifies a different character to print). Another output byte, the control byte, is used to pass control codes to the printer (start a new page, for example) and also to equalize the two voltages (5V and 0V) between the computer and printer. The input byte is used to read messages from the printer (out of paper for example). In fact, the input and control bytes do not use all 8 wires. The unused ones are connected directly to the ground (0V) wire so that their voltage always is 0V. Why these wires are there at all is another strange bit of history, I am sure. The control byte uses only 3 wires and the input byte uses only 5. This limits the size of the numbers that we can pass on these bytes.

We need to learn binary about numbers because the computer deals with the input and output bytes as if they are numbers. It thinks that it is sends numerical data to the 'printer' and receives numerical data from the printer.

<u>Sending Data to the Computer</u>
On the input byte the computer hardware turns the voltages (+5V or 0V) on the 5 wires into a number between 0 and 63. In Figure 3, there are 5 switches on a control panel, each of which are connected to one of the wires of the parallel port's input byte wires. If the switch is flipped on, the wire is set to +5V, which the computer reads as "1." If the switch is set to off, the wire is set to 0V, which the computer reads as "0."

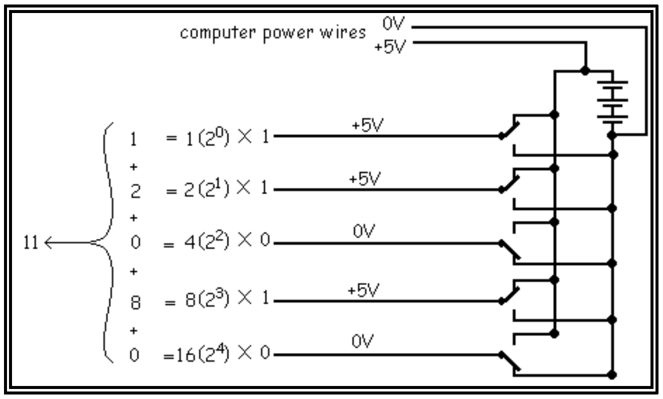


Figure 3
5 switches that communicate with a computer through the parallel port input byte.

In Figure 3, switches 1, 2, and 4 are on. Thus, the wires for 1, 2, and 8 are "1" and the wires for 4 and 16 are "0." The number generated is 1+2+8 = 11. Any other combination of switches will yield a different number in the input byte. Thus, the computer can know exactly which switches are on and which are off.

Just to keep things clear, the value of the multiplier is $2^{(n-1)}$ where "n" is the wire number.

Sending Data from the Computer
To send data on the control or data bytes, we send a number to the byte and it gets turned into 8 (or 3) different wire voltages.

In Figure 4, we have 8 LED indicator lights on a control panel. The power wire for each LED is one of the wires from the data byte of the parallel port. If the wire from the parallel port is at +5V, the LED lights up, otherwise it is dark.

To get the LEDs to light up, we send a number between 0 and 255 to the printer port (the computer thinks that it is printing a character on the printer). In the example in Figure 4, the number is 157. In binary, 157 is 10011101. This means that wires 1, 3, 4, 5, and 8 are at +5V and wires 2, 6, and 7 are at 0V. The 1st, 3rd, 4th, 5th, and 8th LED light up. Each number between 0 (all dark) and 255 (all lighted) will cause a different combination of LEDs to light up.

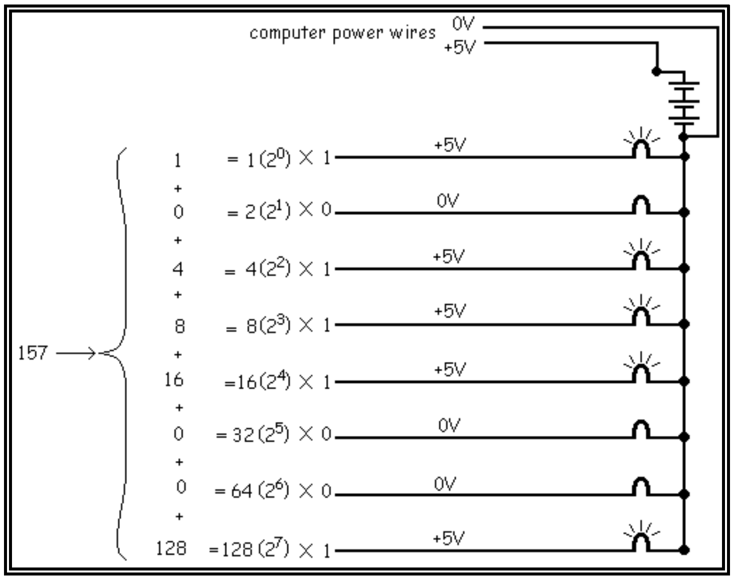


Figure 4
8 LED lights controlled by the data byte of the parallel port.

Reading signals from and Writing Signals to the Printer Port Using Software
The computer has an identifying number for each of its input and output ports (devices that take in signals or send out signals). In PC compatible computers the port numbers for the primary printer port (LPT1:) are 888 (data byte), 889 (input byte), and 890 (control byte).

Each computer language has its own commands for sending data to and from the port. The easiest way to send signals to the data port is to print a character. For example, the command in BASIC:

```
PRINT "A"
```

would send the following signal to the data byte: 01000001, turning on the 1st and 7th LEDs in figure 4. This is because the character "A" is character number 65 in the ASCII (American Standard Code for Information Interchange) character list. The number 65 decimal is equivalent to 01000001 in decimal.

Sending Data from the Computer
However, it is inconvenient to remember all the ascii codes and this method does nothing to help us use the control and input bytes. Fortunately, computer languages have more generic commands for reading data from and writing data to ports. In BASIC, the command:

```
OUT 888, 65
```

would send the same signal to data byte as PRINT "A".

All that we need to do is generate a number that represents the state of our 8 LEDs. If the state of each LED is given by a variable state(i) where 'i' is the LED number, then the following would code in BASIC would set each of the 8 LEDs in figure 4 on or off as specified by the value of the variable state(i) for each LED.

```
LEDstatus=0
FOR i = 1 to 8
  LEDstatus = LEDstatus + (state(i) * (2^(i-1)))
NEXT i
OUT 888, LEDstatus
```

In the case of Figure 4, LEDs 1, 3, 4, 5, and 8 are supposed to be on. So, state(1), state(3), state(4), state(5), and state(8) will be set to "1" by the software before we get to this part of the program. The variables state(2), state(6), and state(7) would be set to "0."

When i=1, (state(i) * (2^(i-1))) works out to 1 X 2^0 which equals 1. So we add "1" to LEDstatus

When i=2, (state(i) * (2^(i-1))) works out to 0 X 2^1 which equals 0. So we add "0" to LEDstatus

When the For-Next loop is done, LEDstatus will equal 1+4+8+16+128 = 157 and we send the number 157 to the data byte to turn on LEDs 1, 3, 4, 5, and 8.

Sending Data from the Control Panel

To read data from the input byte (port number 889) , BASIC also has a command.

```
INP(889)
```

This expression is treated as a variable by the BASIC language. If the number that we read is "10," for example then the 2nd and 4th switches are on and the 1st and 3rd are off. This is because the number "10" in binary is 1010.

You might think that it will be a pain to convert every number we read from the input port into binary then figure out which places are "1" and which are "0." However, BASIC includes a handy little command that takes care of that.

```
IF (2 AND 10) = 2 THEN...
```

This command compares the place values of the numbers 2 and 10. It looks at all the places that are "1" in both the numbers 2 and 10. If these places add up to 2, then it executes the code after "THEN". In this case, the answer would be 2 and the code would be executed.

```
                 2:    0010
                10:    1010
places in common:      0010  =  2      therefore, (2 AND 10) = 2
```

If we tried IF (3 AND 5) = 3 THEN... the expression would not be true and the code after "THEN" would not be executed.

```
                 3:    0011
                 5:    0101
places in common:      0001 = 1        therefore, (3 AND 5) = 1
```

So, to read the signals from 5 switches through the input byte (as in Figure 3) and figure out which switches are "on" or "off," we would use the following code. The status of a switch is stored in a variable called switch(i) where 'i' is the switch number. If the switch is on, the variable is set to "1." If the switch is off, the variable is set to "0."

```
SWITCHstatus = INP(889)
FOR i = 1 to 5
  IF (SWITCHstatus and (2^(i-1))) = 2^(i-1) THEN switch(i)=1 ELSE switch(i)=0
NEXT i
```

Since, in Figure 3, switches 1, 2, and 4 are on while switches 3 and 5 are off, we get the number 01011 binary or 11 decimal sent in on the input byte. In our software, the first line would result in SWITCHstatus variable being set to "11." The FOR-NEXT loop would make the following 5 comparisons:

$2^0 = 1$; If (11 and 1) = 1; the answer is yes, therefore switch(1) =1
$2^1 = 2$; If (11 and 2) = 2; the answer is yes, therefore switch(2) =1
$2^2 = 4$; If (11 and 4) = 4; the answer is no (it equals 0), therefore switch(3) =0
$2^3 = 8$; If (11 and 8) = 8; the answer is yes, therefore switch(4) =1
$2^4 = 16$; If (11 and 16) = 16; the answer is no (it equals 0), therefore switch(5) =0

So our software is able to tell that switches 1, 2, and 4 are on and switches 3 and 5 are off.

Irritating Issues

There are two issues that complicate what we do with inputs and outputs.
1) If you look at Figure 5, you will see that the input byte has pins for bits 3,4,5,6, and 7. Bits 0, 1, and 2 are the ones that are left out rather than bits 5, 6, and 7. This just changes our code a bit. Instead of comparing our SWITCHstatus variable to 2^{i-1}, we need to compare it to 2^{i+2} instead
2) For some strange reason, the computer hardware inverts the signal on pins 11, 14, and 17. That is, if 5 volts comes in on that pin, the parallel port hardware assigns a value of "0" rather than 1 and if 0 volts comes in, it assigns a value of "1." Mystifying, irritating, but easily dealt with in the software. No changes to the hardware that we build will be made to deal with this issue.

Connecting Control Panel Switches to the Computer

As you already know, we have 5 input pins on the LPT 1 (printer) port with which to send signals from switches to the computer. Five switches is not a very complicated control panel; we need dozens. One possibility is to install more parallel port cards into our computers, but this is expensive and difficult, since printer ports are not commonly sold anymore and our older computers make it hard to install these cards because there are only so many interrupts (interrupts are the coded signals that the peripheral cards use to tell the main processor of the computer that they have something to send it). Not only that, if we installed two more parallel ports (the maximum that can be installed) that would only give us 15 inputs in total; still not a very complex panel.

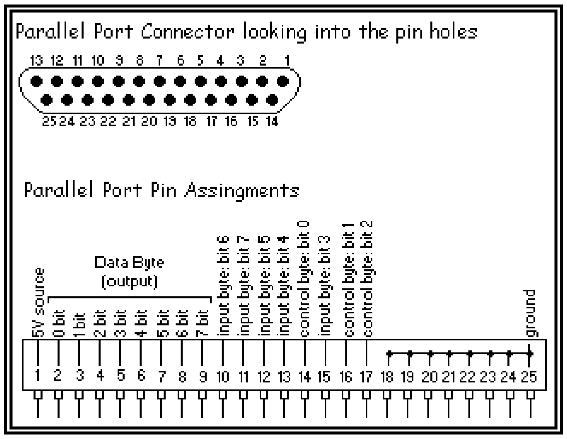


Figure 5
Parallel port pin assignments

A better method is to send lots of signals to the parallel port a few at a time over a smaller number of input bits. There is a very handy chip that allows us to do this: the ***multiplexer***. The multiplexer is an integrated chip with 16 pins in two rows: a dual inline pin, or DIP chip (Figure 6). Integrated chips are named such because they have many different components integrated into one unit. We use a number of different types of chip in our control board circuits.

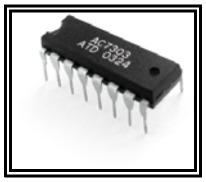


Figure 6
Typical DIP integrated chip

Multiplexers

A multiplexer takes in two groups of 4 inputs: group A and group B (there are other varieties with different numbers) and sends them to 4 outputs one group at a time. So, we can send 8 separate signals from 8 separate switches: A1, A2, A3, and A4 in one group and B1, B2, B3, and B4 in the other group (Figure 7). Which of the two groups gets sent on to the parallel port depends on the value of a selector input. If the selector pin is set to 0V, then it is the signals from group A that get sent on. If the selector pin is set to 5V, then it is the signals from group B that gets sent on. The selector pin gets its signal from one of the output bytes of the parallel port (Figure 8).

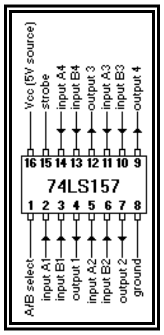


Figure 7
Multiplexer Integrated Chip type 74LS1157 pin assignments

To operate the multiplexer (Figure 8), the 4 output pins are connected to 4 of the 5 input pins in the parallel port. The selector pin is connected to one of the 3 control pins (output). In the example in Figure 8, the selector pin of the multiplexer is connected to the pin for bit 0 of the control byte. If we write the number "0" to the control byte (000 binary), bit 0 of the control byte will have the value "0" and the selector pin of the multiplexer will be set to 0 volts. This will cause input group A to be sent to the output pins of the multiplexer and, thus, on to the input byte of the

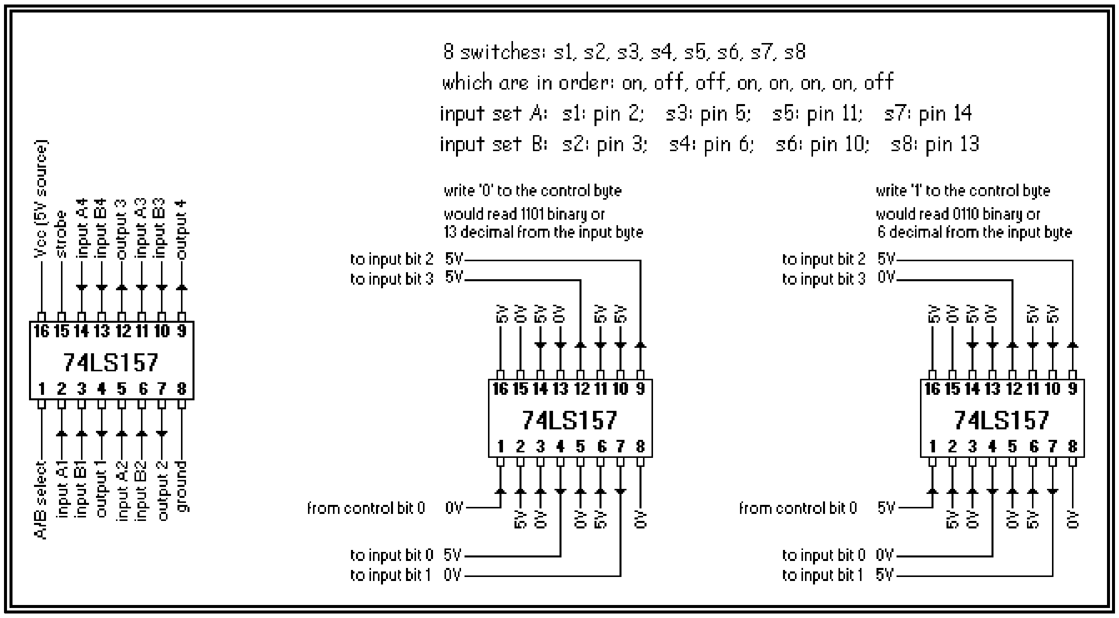

8 switches: s1, s2, s3, s4, s5, s6, s7, s8
which are in order: on, off, off, on, on, on, on, off
input set A: s1: pin 2; s3: pin 5; s5: pin 11; s7: pin 14
input set B: s2: pin 3; s4: pin 6; s6: pin 10; s8: pin 13
Vcc (5V source)
strobe
input A4
input B4
output 3
input A3
input B3
output 4
16 15 14 13 12 11 10 9
74LS157
1 2 3 4 5 6 7 8
A/B select
input A1
input B1
output 1
input A2
input B2
output 2
ground
write '0' to the control byte
would read 1101 binary or
13 decimal from the input byte
to input bit 2 5V
to input bit 3 5V
from control bit 0 0V
to input bit 0 5V
to input bit 1 0V
write '1' to the control byte
would read 0110 binary or
6 decimal from the input byte
to input bit 2 5V
to input bit 3 0V
from control bit 0 5V
to input bit 0 0V
to input bit 1 5V

Figure 8
8 switches routed through 1 multiplexer in 2 groups of 4

parallel port.

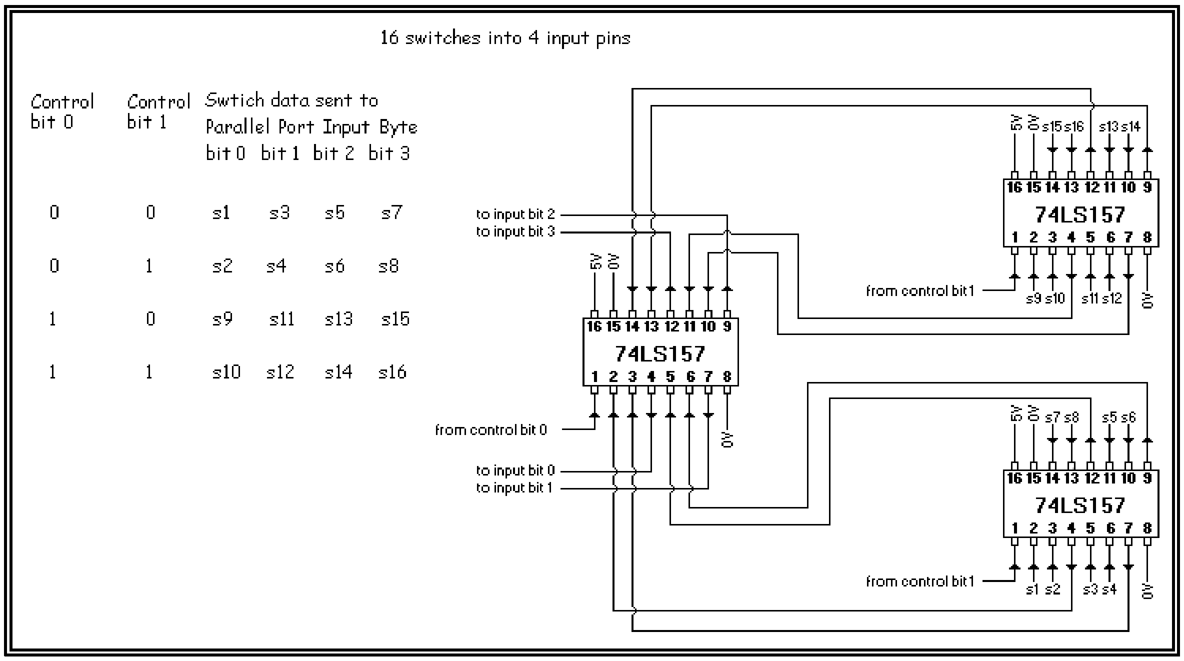
Input Byte:	bit 3	bit 2	bit 1	bit0
Signal at Bit	5V	5V	0V	5V
Bit Value	1	1	0	1
Binary Value at Input Byte		1101		
Decimal Value at Input Byte		13		

As shown in Figure 8, if switches in group A, (switch numbers 1, 3, 5, and 7) are ***on***, ***off***, ***on***, and ***on*** respectively, then the signals to the 4 bits of the input byte will be 5V, 0V, 5V, and 5V (starting with bit 0). This means that when we set the control byte to "0" and read the value of the input byte, we will see the number 13. The computer sees the binary number 1101 and knows that switches 7, 5, and 1 are on.

If we write the number “1” to the control byte (001 binary), bit 0 of the control byte will have the value “1” and the selector pin of the multiplexer will be set to 5 volts, instead of 0 volts. With this setting, the multiplexer will send data set B (switches 2, 4, 6, and 8) on to the parallel port, rather than set A. Since these four switches are, in order from 8 to 2, ***off***, ***on***, ***on***, and ***off***, the value of the input byte will be 0110 binary, or 6 decimal.

Multiple Multiplexers

What if we have more than just 8 switches? We can use groups of multiplexers to handle very large groups of switches (Figure 9). For 16 switches, we need 3 multiplexers. The 16 switches send their signals to one of two multiplexers. Each of these sends either one group of 4 or the other to the third multiplexer. The first two multiplexers send either data group A or B depending on the signal they get from bit 1 of the control byte. The third multiplexer sends along the data from either multiplexer 1 or 2 depending on the signal it gets from bit 0 of the control byte. Thus, the computer must read the input byte from the parallel port 4 times to get the information from all 16 switches. Before each reading, we must send a value to the control byte to set the three multiplexers: 0, 1, 2, and 3 decimal (00, 01, 10, and 11 binary). Each of these 4 control byte values causes a different set of switches to be read by the computer.


16 switches into 4 input pins

Control bit 0	Control bit 1	Swtich data sent to Parallel Port Input Byte bit 0	bit 1	bit 2	bit 3
0	0	s1	s3	s5	s7
0	1	s2	s4	s6	s8
1	0	s9	s11	s13	s15
1	1	s10	s12	s14	s16

Figure 9
Signals from 16 switches sent to 4 parallel input pins in groups of 4 by three multiplexers.

If we have more than 16 switches, we just use more multiplexers.

Connecting Control Panel Indicator Lights to the Computer

Many of the indicator lights on a control panel can be set by the switches themselves (most of our switches are double pole, which means that one pole can send a signal to the computer and the other can send a signal to an indicator light). However, there are many functions that the computer software manages for which we want to have indicator lights. To do this, the data byte on the parallel port can be used to turn light emitting diodes (LEDs) on and off. The low voltage end of the diode can be connected to ground (0V) and the high voltage end of the diode is connected to the output pin of the parallel port. When the pin is at 0 volts, the LED is off. When the pin is at 5V, the LED is on (see Figure 4).

LEDs are designed to work with about 2 volts, so a resistor is wired in series with the LED to drop the voltage and avoid damaging the LED. We also want to limit the current that the parallel port must supply. If the port draws too much current, the motherboard of the computer can be destroyed.

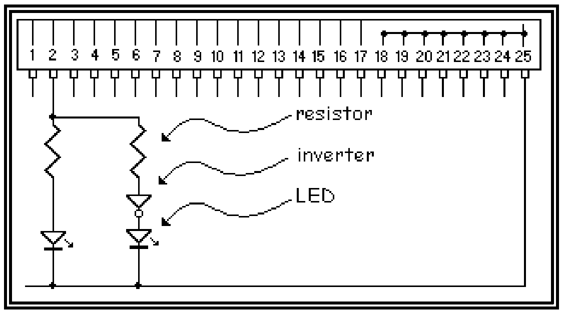


Figure 10
LEDs wired into a parallel port circuit. Using the inverter, one and only one of the LEDs is on at a time. If the left-hand LED is on, the inverter is getting a 5 volt signal and puts out a 0 volt signal so the right-hand LED is off. If the left-hand is off, then the inverter is getting 0 volts and puts out 5 volts, turning the right-hand LED on.

The maximum current that the LED needs is 0.02 amps and we need a resistor that will use up 3 volts (5 V_{supply} - 2 V_{LED} = 3 V). We can use Ohm's law to calculate the necessary resistance.

$$\begin{aligned} R &= V/I \\ &= 3V/0.02A \\ &= 150\ \Omega \end{aligned}$$

Alternating LEDs
Often, it is desirable to have an LED to indicate when some function is on and a separate LED to indicate that it is off (you would not want to have a damaged LED fool you into thinking that a function was off when it is not). A good example of this is our door circuit where we have a red LED to tell us when it is not safe to open a door and a green LED to tell us when it is. However, we do not want to tie up two parallel ports for this function nor increase the complexity of the software by programming two LEDs.

To have an ***on*** and an ***off*** LED, we use a circuit component called an inverter (also called a NOT gate). The inverter puts out a signal that is the opposite of the signal that it takes in. If its input pin is a 5 volts, it output pin is at 0 volts. If its input pin is at 0 volts, its output pin is at 5 volts (see Figure 10). Inverters come in integrated chips with 6 separate inverters: a hex inverter (see Figure 11).

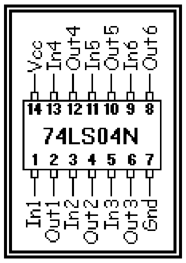


Figure 11
74LS04 hex inverter

More Than 8 LEDs
As with switches, usually we have more LEDs on our control panel than we have output ports on the parallel port. We need to do what we did with the switches, but this time, we need to send our 8 bits of data to 8 LEDs and have them remember whether they are supposed to be on or not while we use the same 8 output bits to send signals to 8 more LEDs. We need a memory chip that can remember a set of signals. There are a multitude of different types of memory, all packaged into DIP integrated chips.

Memory Latches
I think that the easiest type of memory chip to use is the latch. Some of the others probably are easier once you figure them out, but they don't look that easy to me! You need to keep in mind that I am in the process of learning this stuff as well. An individual memory latch has an input and an output. The memory latch chip also has a latch enable pin. When the latch enable pin is at 5 volts, the output will be the same as the input. As soon as the latch enable pin goes to 0 volts, the output pin holds what ever voltage was at the input pin when the latch enable pin was still 5 volts. Actually, there is a small time lag of a few nanoseconds after the latch enable pin goes to 0 volts during which we need to keep the input pin at our desired voltage.

Steps in the Memory Latch Process

Step 1) Set the input pin(s) to 5 volts or 0 volts as desired (Figure 12).

Step 2) Set the latch enable pin to 5 volts.
This can be done using one of the wires from the control byte.
The corresponding output pin(s) will match the input pin(s).

Step 3) Set the latch enable pin to 0 volts.
Wait 20 nanoseconds (take care of this in the software).

Step 4) The input pin(s) now can be set to either 5 volts or 0 volts, but the output pins will remain at the voltages that they had in step 2.

To reset the output pin(s), repeat the process from step 1.

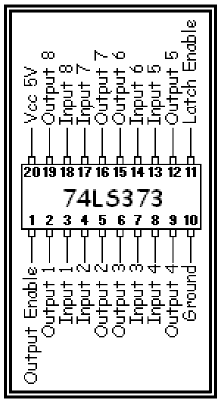


Figure 12
74LS373 octal memory latch

The output pins on the memory latch chip can sustain enough current to operate an LED or to hold a parallel port input pin at a given voltage. This means that we can use the 8 pins of the parallel port's data byte to set output pins on one memory latch, then use the same 8 parallel port pins to set the output pins on a different memory latch (Figure 13).

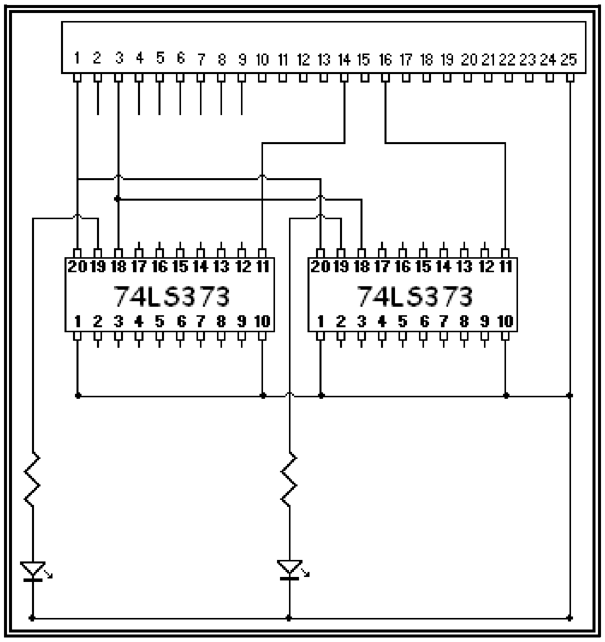


Figure 13
2 octal memory latches allowing 1 output pin to control two LEDs independently. The left-hand memory latch is controlled by the output from pin 14 on the parallel port; the right-hand latch from parallel port pin 16.

In Figure 13, we only are showing the output from parallel pin 3 only to keep the picture clear. This parallel port output sends a signal to both memory latches simultaneously. However, they ignore it, unless their latch enable pin (pin 11 on the chip) is set to 5 volts.

To set the output pin (pin 19 in this case) on the left-hand memory latch, parallel port pin 14 is set to 5 volts and parallel port 16 is set to 0 volts (see Figure 5 for parallel port pin assignments). We do this by writing the number 1 decimal (001 binary) to the control byte of the parallel port. Now, pin 19 on the left-hand memory latch will adopt what ever voltage is coming from parallel port pin 3. Then, we write 0 decimal (000 binary) is written to the control byte. Pin 19 on the left-hand memory latch will maintain it current value and the left-hand LED will stay on or off as desired. The voltage on parallel port pin 3 is changed to the value we want for the output of the right-hand memory latch. Then, we write the number 10 decimal (010 binary) to the control byte of the parallel port and output pin 19 on the right-hand memory latch will adopt the voltage of parallel port pin 3, turning the LED on or off. We then write the number 0 decimal (000 binary) to the control byte and the two memory latches will keep their respective voltages and the LEDs will keep their current status until we want to change them.

Controlling Higher Voltage and Current Devices

The 5 volt logic components (parallel port pins, memory latches, etc.) cannot power higher voltage components like our 28 V warning lamps, nor can they power high current (low resistance) components. To switch these on and off using the computer we have two choices: transistors and relays.

Transistors

Transistors are like an electronic switch. They have three wires: the collector (C), base (B), and emitter (E). There are two types of transistor: NPN and PNP. NPN transistors are more common, cheaper, and do exactly what we need to get done.

It is important to keep straight which pin is which because transistors are destroyed easily if they are not connected correctly.

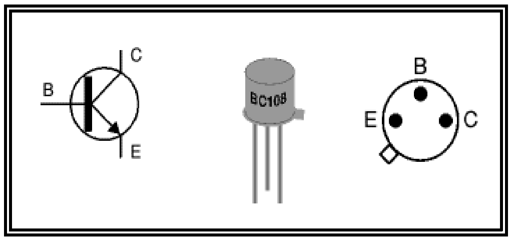


Figure 14
Left: circuit symbol for an NPN transistor.
Center: a typical metal capped transistor.
Right: use of metal tab to identify the thee leads when viewed from the bottom.

The main current that the transistor is controlling flows from the collector to the emitter. No current will flow, however, unless a small voltage is applied to the base. When this happens, current can flow from collector to emitter and from base to emitter. The collector, emitter, and base do not have to be at the same voltage, but there are limits to the voltage difference, especially between the base and emitter. For this reason, these transistors usually are placed after the main load. Careful reading of transistor specifications and planning of the circuit is necessary to avoid transistor damage.

A transistor allows a low voltage circuit, such as a 5 volt parallel port, to turn a higher voltage circuit on and off (Figure 15). Thus, we have a controlling circuit (the parallel port circuit) and a controlled circuit (the lamp and battery).

The current from the base and the current from the collector both pass through the emitter. For this reason, the two circuits must share a common ground so that the two currents can separate and complete their individual circuits.

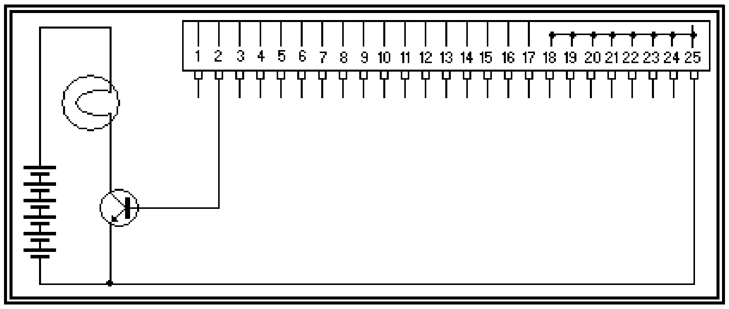


Figure 15
Using a parallel port output pin to turn a transistor on or off, which, in turn, turns a battery powered lamp on or off. Writing 1 decimal (00000001 binary) will turn the lamp on (the transistor base is at 5 volts, which lets current flow from collector to base). Writing 0 decimal (00000000 binary) turn the lamp off (the transistor base is a 0 volts).

Relays

There are situations in which it is not possible or advisable to the controlling circuit and the controlled circuit to share a common ground. This could be because the controlled circuit is alternating current or because it has a higher voltage or current than a transistor can manage. A relay, like a transistor, is a switch. Relays have the advantage that they maintain complete separation between the controlling and controlled circuits. Relays are much larger than transistors, they are slower to switch on and off, and they consume more power (Figure 16).

Figure 16
Typical relay

Relays use an electromagnet to flip their switches between the on and off position. The electronmagnet is a coil which is powered by the controlling current. The magnetic field created by they coil causes the switch attached to the main current (the controlled current) to flip positions. This switch is a double throw switch (like in Figure 1). The control switch can be wired so that the coil can either turn the main switch on or off. One wire for the controlled circuit always is attached to the common lead of the relay. The other wire for the controlled circuit is attached to the Normally Open lead (NO) if the circuit is supposed to off most of the time and only turned on by the controlling circuit. Alternately, the controlled circuit can be attached to the Normally Closed (NC) lead if the controlled circuit is supposed to be on most of the time (Figure 17).

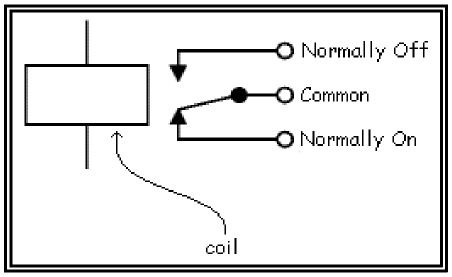


Figure 17
Circuit diagram for a relay

Controlling a Relay
Relays usually need a larger voltage and current to activate the coil than can be provided by a 5 volt logic circuit. So, the parallel port or other 5 volt signal is used to turn on a transistor which, in turn, activates a larger current to activate the relay coil (Figure 18). In addition to the transistor, a diode must be used, wired in reverse, to divert the current spike that accompanies the turning off of the relay coil. This *protection diode* prevents the sensitive logic circuit components from a dangerous spike in voltage that would occur without it each time the relay was turned off.

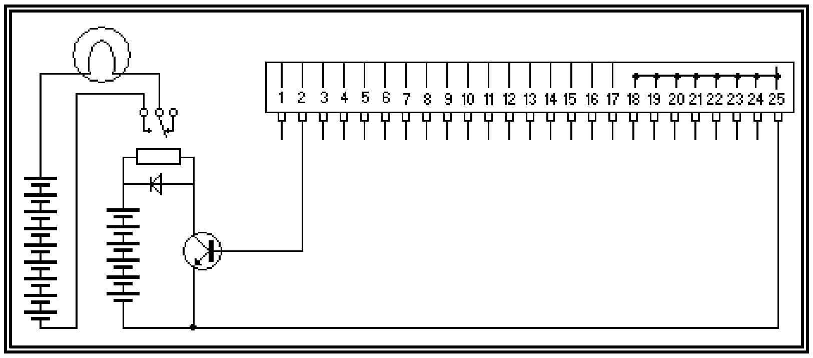


Figure 18
A 13.5 volt circuit powering a lamp which is turned on by the activation of a relay. They 9 volt relay circuit is turned on by a transistor which is activated by an data pin from the parallel port. When a “1” is written to this pin in the software, the pin goes to 5 volts, which activates the transistor, which activates the relay. The lamp circuit is wired to the NO lead of the relay which means that the circuit is open unless the relay is activated.

For even greater separation of the controlled circuit from the controlling computer circuit, one can use an optocoupler to power the transistor. An optocoupler takes the 5 volt signal from the parallel port and uses it to power an LED. The light from the LED powers a photodetector which converts the light energy back into electrical energy. Thus, even if the insulation in the relay shorts out, the computer still is protected from the dangerous voltage in the controlled circuit.

With a relay, we could use the computer’s parallel port to switch power to the habitat’s three electrical circuits on and off. We also could use relays as sensors to detect when those same electrical circuits are switched on or off at the circuit breaker.

Space Sim Circuits
The door circuit card does is not an interface for a control panel, rather it is an interface for the computer and the doors. In essence, each door constitutes a mini control panel as it has one switch which opens and closes with the door and two pairs of LEDs to indicate if it is safe to open the doors or not. All of the components are similar to ones which we have discussed, with the exception of ***pull-up resistors*** which are attached to the 4 input pins.

Pull-up Resistors

The door switches connect the input pins on the multiplexer to the ground wire (0 volts). When the door is closed, its input pin on the multiplexer goes to 0 volts and, as a result, so does the corresponding input pin on the parallel port. But, when the door is open, there is nothing holding the input pin at any particular voltage. As a result, its voltage will tend to drift. We actually want the input pin voltage to go to 5 volts so that it will be different from what it is when the door is shut (otherwise the computer cannot tell the difference between shut and open). To make the input pins go to 5 volts when the doors are open, we attach the input pins to the 5 volt wire using high resistance resistors (about 3000 ohms). These are sufficient to keep the input pins at 5 volts when the doors are open with very little current, but have too much resistance to stop the switch from dragging the input pin to 0 volts when the door is closed.

Supplemental Current

Another addition to these circuits is that we draw electrical power from the computer's power supply directly in order to supplement the current that the parallel port must supply. The connection is in parallel (remember grade 9?) so that the parallel port only has to supply part of the current.

Decoupling Capacitors

All the chips on a circuit board create little voltage spikes when they switch input and output states. Other chips can erroneously interpret these spikes as data and we can end up getting strange results in our circuits. To counteract this problem, most experts advise the use of decoupling capacitors.

A capacitor is like a small rechargeable battery. It consists of 2 metal plates separated by a thin layer of insulating material. One plate is attached to a negative voltage, the other is attached to a positive voltage. The negative voltage of the negative plate drives electrons out of the positive plate (giving it a net positive charge), and the positive voltage of the positive plate attracts electrons into the negative plate (giving it a net negative charge). The motion of electrons into one plate and out of the other causes a current in the circuit. When the repulsion of the charges within each plate equals the attraction to the opposite plate, no further movement of charges takes place and the current caused by the capacitor stops.

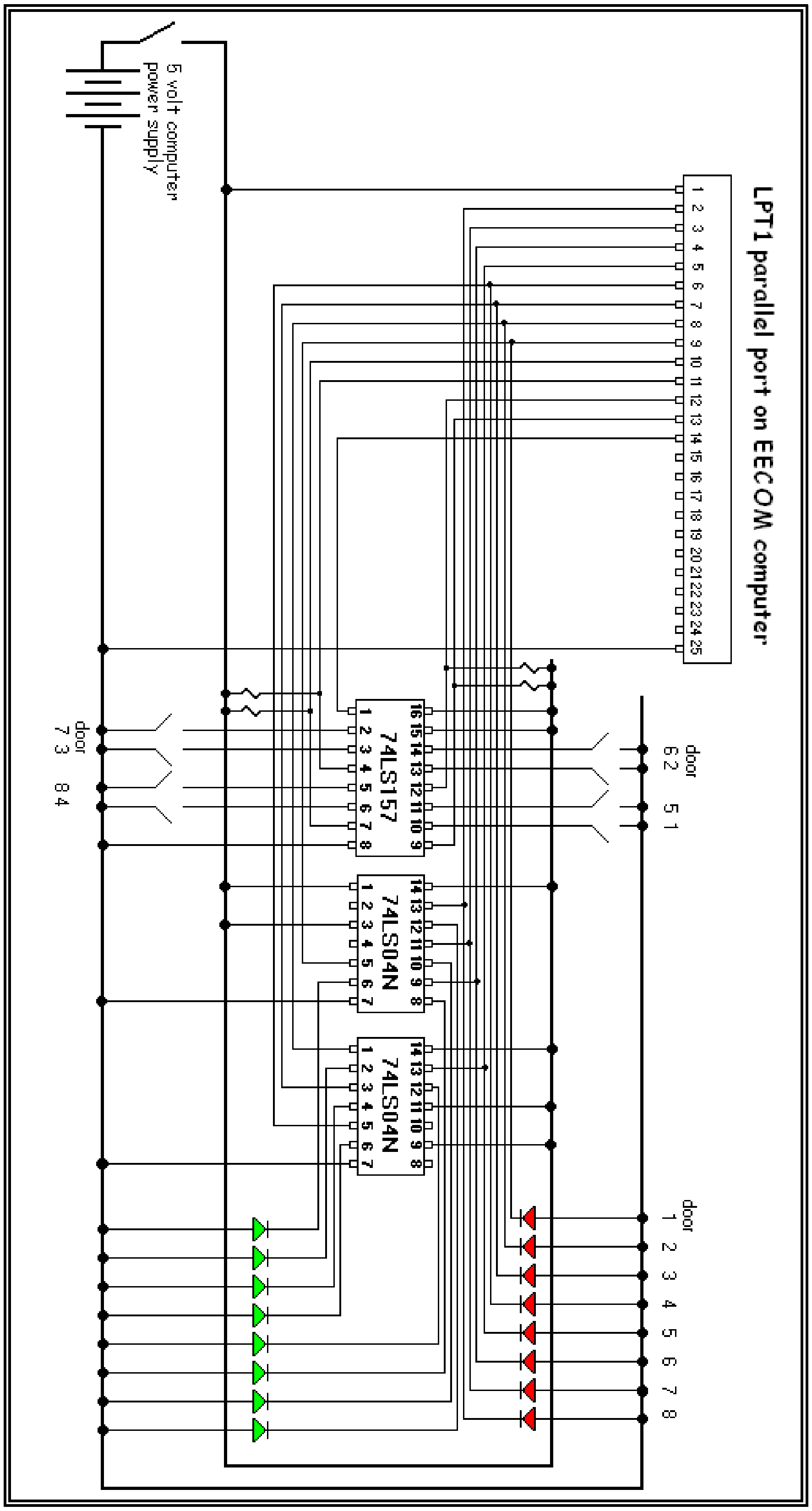
If the battery that supplied the voltage is disconnected, there is no longer any voltage to overcome the mutual repulsion of the charges in each plate. The extra electrons move out of the negative plate through the circuit and back into the positive plate so that both plates are neutral again. While this is going on, a small current will be caused in the circuit by the capacitor.

So, capacitors store charges whenever the voltage of the circuit increases and release it when the voltage decreases. In other words, they are like a small sponge for current. If there is a small spike in voltage in a circuit, the capacitor will soak up the current that this voltage spike creates and release it when the voltage goes back to normal. Thus, the effects of the voltage spike are reduced by being spread out over a much longer time interval. Very clever!

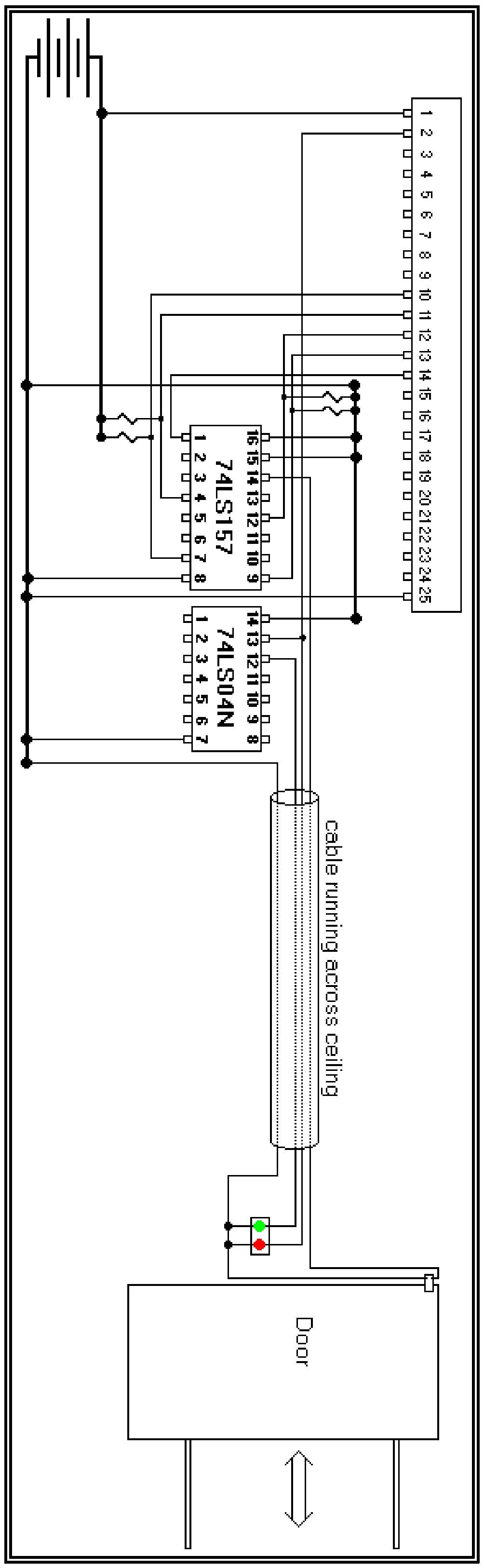
The decoupling capacitors are connected between the supply and ground pins of the integrated chips on a circuit board. Usually only 1 decoupling capacitor is needed for every 4 chips to be effective. We do not have any decoupling capacitors in our circuit boards, but we may find that we have to add them.

Door Circuits

Each of the 8 doors sends a 0 volt signal to the input byte of the parallel port when the door is closed. Otherwise, the input pins are held at 5 volts. The 8 data byte pins are used to send a 5 volt signal each of 8 red door warning lights as well as to 8 inputs on the two hex inverter chips. If the data byte pins are at 5 volts, the red LEDs light up and the inverter puts out 0 volts to the green LEDs. If the data byte pins are at 0 volts, the red LEDs are dark, but the inverter output pins go to 5 volts and the green LEDs light up.

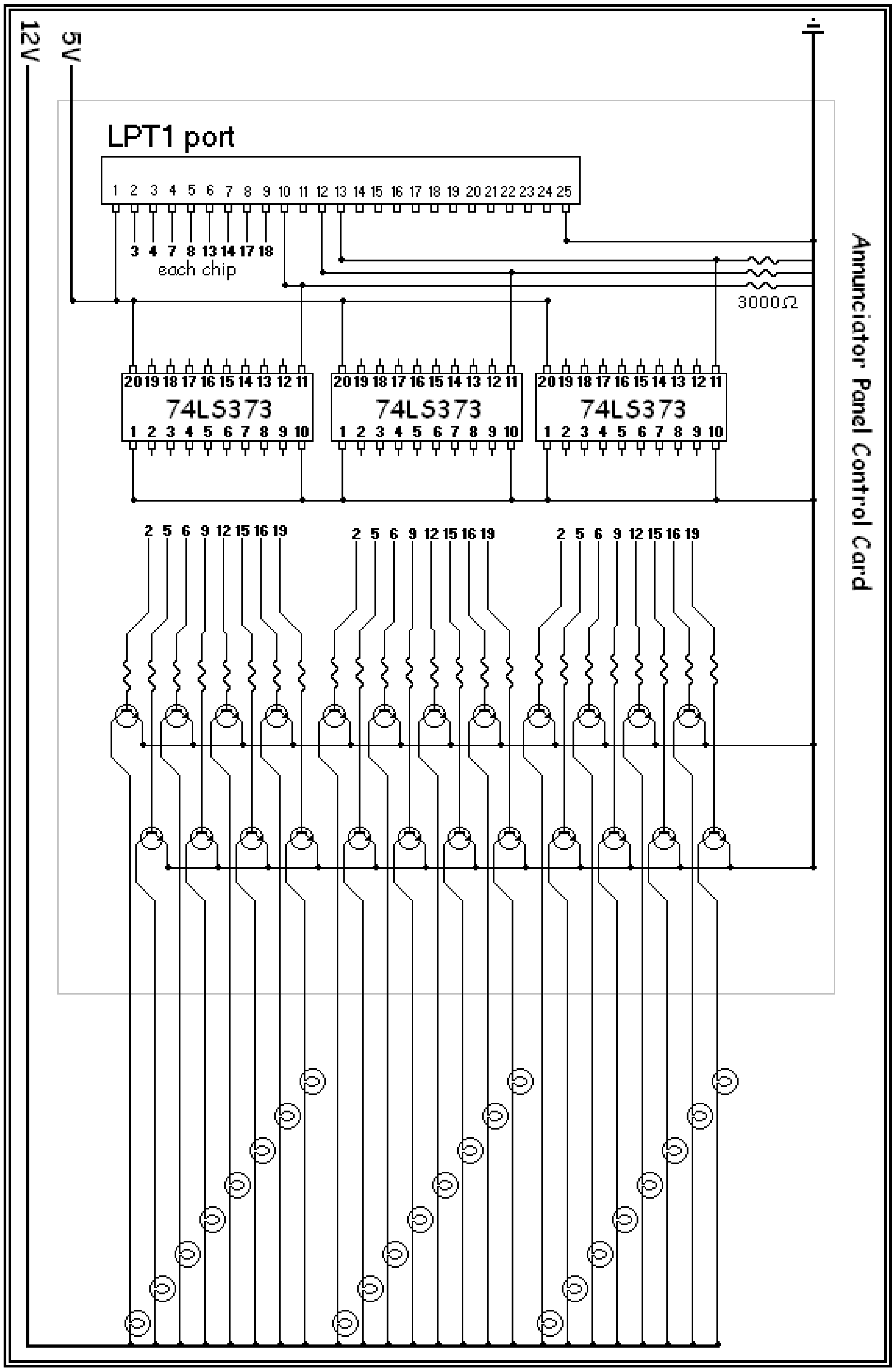


The actual physical arrangement of the components of the door circuit is as follows. The three chips are on the interface card plugged into the computer. The LEDs and the switch for the door are at the door itself (only one hex inverter and 1 door are show for clarity). The door switch and the two LEDs share a common ground wire. When the door is closed, so is the switch and the input pin at the multiplexer is drawn down to 0 volts. The multiplexer output pin is also drawn to 0 volts when the appropriate input group is selected. Since the multiplexer output pins are otherwise held at 5 volts by the resistor attached to the 5 volt source wire, the computer knows whether the door is open or not by the value it reads from the input byte. The EECOM software determines which LED should be on and writes the appropriate number to the data byte of the parallel port.



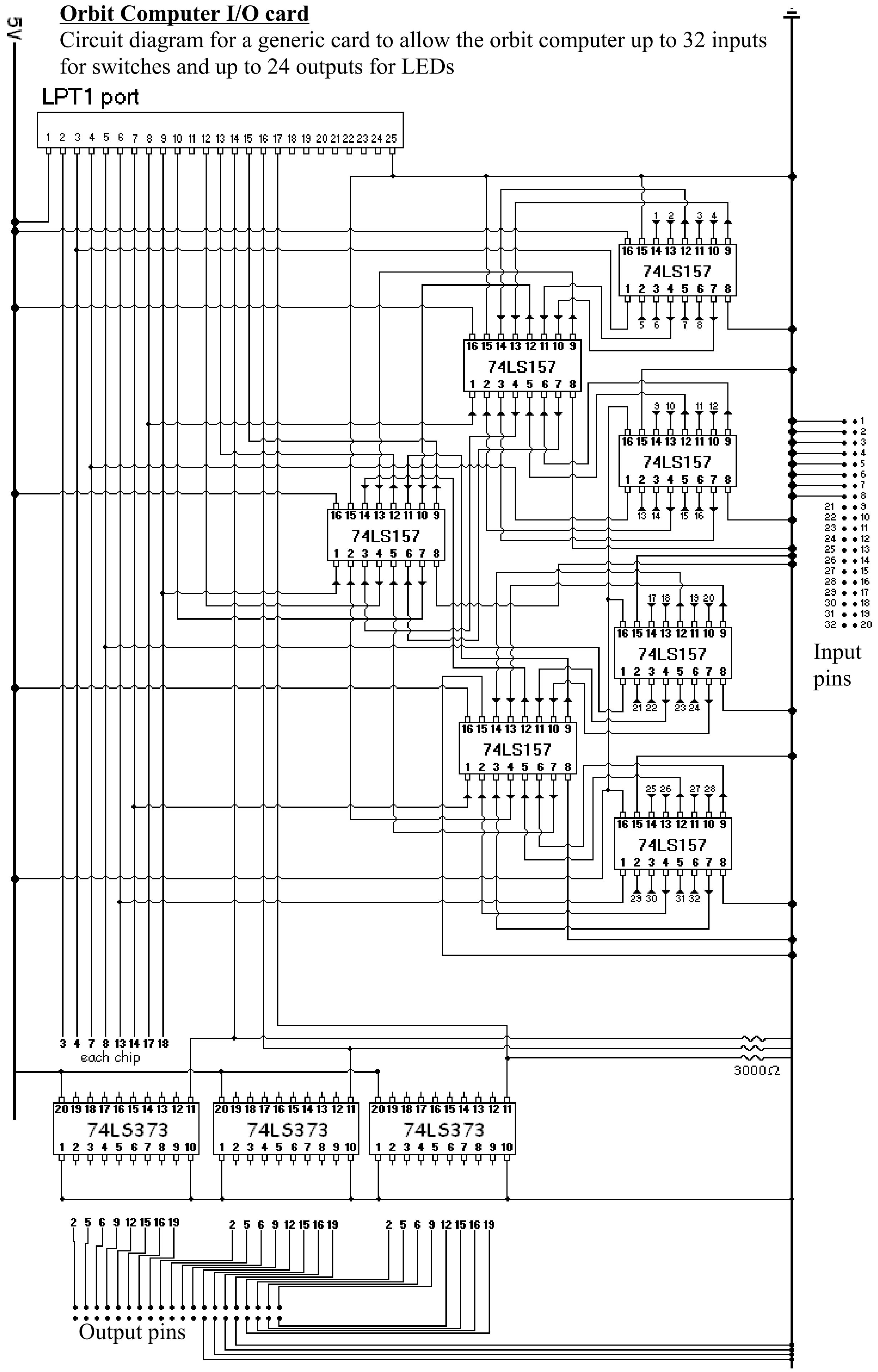
Warning Light Panel

The warning light panel has 24 red lamps that will run on the 12 volt supply voltage from the computer's power supply (they are designed to work on 28 volts, but they are bright enough at 12). They get their signal from the sim computer running the their version of the orbit software. Each data pin sends a signal to the same input pin on three memory latches. The three pins of the control byte are used to enable and disable the latching function of the memory latch chips so that only one latch actually reads the data at a time. For clarity, the data connections to the memory latch chips are not shown, but the pin numbers show where each wire is to be connected.



Orbit Computer I/O card

Circuit diagram for a generic card to allow the orbit computer up to 32 inputs for switches and up to 24 outputs for LEDs



5V
LPT1 port
74LS157
74LS373
3 4 7 8 13 14 17 18 each chip
3000Ω
Input pins
Output pins