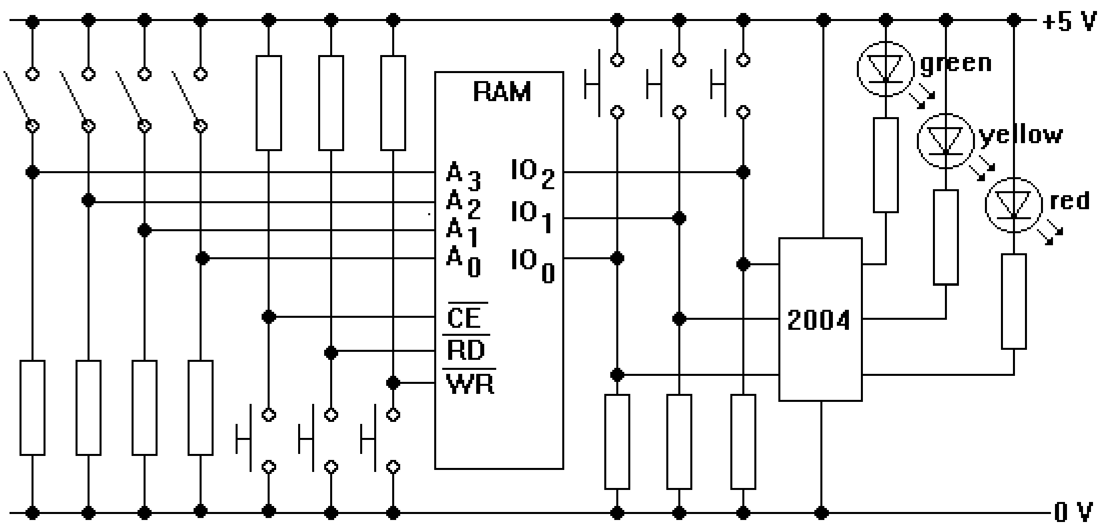
Programming a RAM chip

The circuit you will be assembling is shown below. The RAM can be any static RAM i.c. which holds more than sixteen three-bit words with the standard 3-bit control bus. Most 2048   8 RAMs have the pinout of a 6116 i.c.



1 Start off by setting up the **data switches** as follows. On the far right-hand end of your breadboard, insert the LEDs, driver, three switches and 1 k   pull-down resistors. Check that each LED glows when its switch is pressed.

2 Place your RAM i.c. on the breadboard. Connect it to the supply rails. Connect any of the address inputs which you aren't going to use to either +5 V or 0 V.

3 Set up the four DIL switches and their 2.2. k   pull-down resistors connected to the four address inputs. Use a voltmeter to check that each **address switch** can be used to feed 1's and 0's into the address inputs.

4 Now set up the three push switches and 4.7 k   pull-up resistors which fix the state of the control bus. Use a voltmeter to check that each control line goes low and high when the relevant **control switch** is pressed and released.

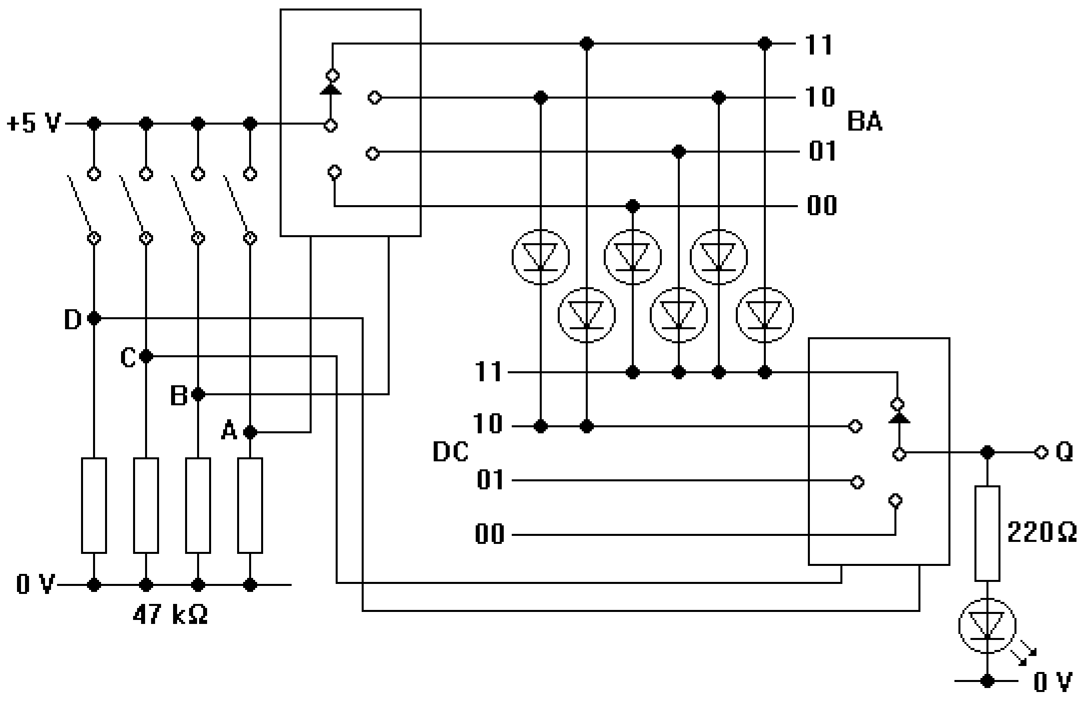
5 Test your circuit by entering 010 at location 0000 as follows:
set the address bus to 0_h
hold $\overline{WR}$ low and keep it there
make the yellow LED glow by pressing its switch
pulse $\overline{CE}$ low
release all of the control and data switches
pull both $\overline{CE}$ and $\overline{RD}$ low; if all is well, only the yellow LED should glow.

6 Try entering different 3-bit words at a variety of locations. Check that you can read them out again.

7 Program the RAM with a 16 step traffic light sequence. Then use a 1 Hz oscillator and a binary counter to run the LEDs through the sequence.

Diode PROMS

The simple PROM shown below holds sixteen bits. You are going to assemble it, read out its truth table and then reprogram it.



1 Place two 4052 multiplexer i.c.s at opposite ends of your breadboard. Think carefully about the layout of the eight lines connecting the six programming diodes.

2 Insert the DIL switches and their associated pull-down resistors. Connect D, C, B and A to the address inputs of the multiplexers. Use a voltmeter to check that the address lines go high and low as the switches are opened and closed.

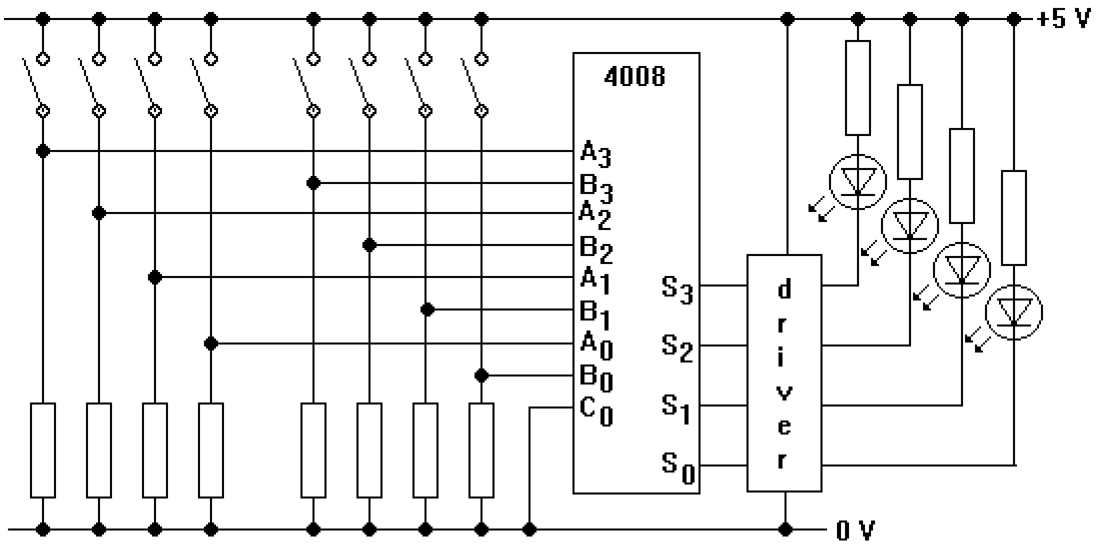
3 Complete the rest of the circuit. Use 1N4148 diodes for the programming.

4 If all is well, the LED should glow when DCBA represents any number higher than 9 i.e. outside the BCD range.

5 Reprogram the system so that the LED only glows when DCBA represents a prime number (2, 3, 5, 7, 11 or 13).

Four-bit arithmetic

This practical is based on the use of a 4008 four-bit full adder i.c.. You have access to the A and B inputs of all four full adders, but only to the C input of the first one.



1 Assemble the 2004 driver, LEDs and 220 pull-up resistors on the far right-hand of your breadboard. Hold each driver input high in turn to test the LEDs.

2 Now put together the DIL switches and 100 k pull-down resistors at the far left-hand end of your breadboard. Use the driver and LEDs to check that each switch pulls its line high when it is closed.

3 Finally, insert the 4008 full adder i.c. Set A to 0111 (+7) and B to 0001 (+1). If all is well, the LEDs should indicate 1000 (+8).

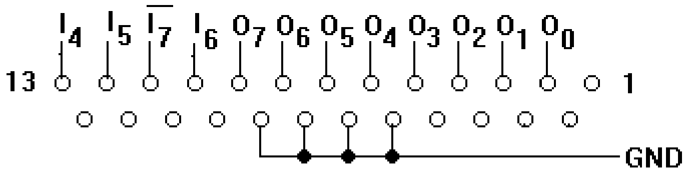
4 Calculate the outcomes (in four-bit binary code) of the following sums. Remember that the outcome can only be a four-bit word. Then use the circuit to see if you are right.

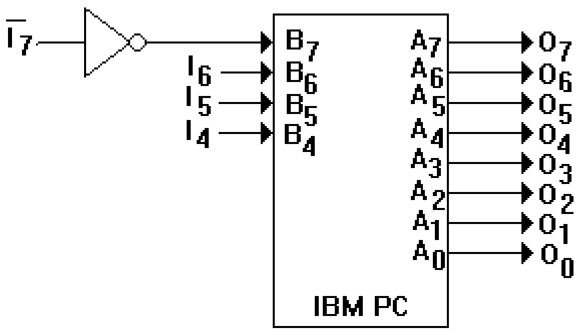
5 + 3 = ?
11 + 4 = ?
10 + 7 = ?
14 + 13 = ?
9 + (-3) = ?
6 + (-8) = ?
(-7) + (-4) = ?

5 With the addition of some NOT gates and suitable connection of C_0, convert your system into a parallel subtractor.

PC parallel port with DEBUG

This experiment will allow you control the pins of the printer port of a PC through DEBUG. The diagram shows the pinout of the PC parallel port, looking into the socket.





1 Arrange for $\overline{I_7}I_6I_5I_4$ and GND to be fed to a breadboard. Jamming 0.6 mm wire into the socket can be effective. Each input floats high. You can pull them low by connecting them to GND, the 0 V line from the PC.

2 The first step is to find the address of the PC's input port (port B). Load up DEBUG from the DOS directory. The prompt is -. Type D 4 : . Press ENTER.

3 If all is well, you should be looking at half a page of memory, starting at 4 $_h$. The least significant byte (lsb) of the address of the A port is stored at 4 8$_h$. The msb is stored at 4 9$_h$.

4 The address of the B port is the address of the A port plus one. It is quite common for the A and B port addresses to be 378$_h$ and 379$_h$. Whenever either of these addresses appear below, you will need to substitute your own.

5 Test the input port with DEBUG as follows:

type I 379 and press ENTER

the screen should display a two-bit word in hex, with 7 as the msb. This is because $I_7I_6I_5I_4$ = 0111.

set $\overline{I_7}I_6I_5I_4$ to 0111, 1011, 1101 and 1110 in turn. Use DEBUG to read the byte at the input port each time. Verify that the msb is as it ought to be.

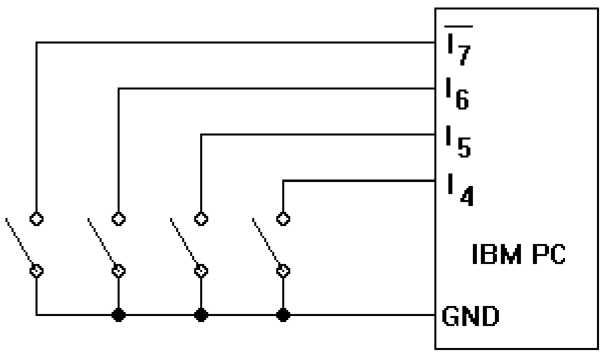
6 Now connect a LED and 220 series resistor between O_7 and GND, so that it glows when O_7 is high.

7 Use DEBUG to test the output port by typing O 378 8 and pressing ENTER. If all is well, the LED should now be glowing.

Stepping through a program

This practical will give you some practice at using the various features of DEBUG, as well as confirming the behaviour of the PC parallel port.

1 Connect a set of four DIL switches to the printer port of your PC as shown below. You don't need an external power supply as the inputs float high when disconnected from 0 V.



2 Open all the switches. Load up DEBUG from the DOS directory.

3 Set the instruction pointer to address E h as follows:
type R IP
type E
type R to verify that IP = E

The following program reads in the state of the input port and stores it at address 7 h. You may need to replace the second and third bytes with the address of your input port.

BA 79 3 BB 7 EC 88 7 EB FE

4 Enter the program as follows:
type E E
type each byte of the program, separated by a space
type

5 Use the -D E command to display the program. Check that it has been entered correctly.

6 Use the -U E E A command to display the program in assembler.

7 Save the program with the name PROG01.MPC on a floppy disc as follows:
name the file by typing N A:PROG01.MPC
enter the size of the file to be saved in CX by typing R CX 8
reset the BX register by typing R BX
write the file, starting at address 0800_h, to disc by typing W 8
This saves the entire contents of memory between 0800_h and 1000_h.

8 Use the -R command to look at all the registers. Check that IP = E .

9 Step through the program with the -T command until it enters the loop at the end. Pay particular attention to the IP and DX registers after each step.

10 If all is well, location 7 now holds the byte 7X. Verify this by typing D 7 .

11 Run through the program four times, with a different DIL switch closed each time. Don't forget to set IP to E before you use -T to step through the program. Predict what you will find at location 7 each time.

12 Adapt the program so that it saves the state of the input port at location 9AD. Verify that it works as required. Save your adapted program as A:PROG02.MCP.

13 Load the original program as follows:

    name the file by typing N A:PROG01.MPC
    enter the size of the file to be saved in CX by typing R CX 8
    reset the BX register by typing R BX
    write the file to disc by typing L 8

This replaces the contents of memory between 0800_h and 1000_h with eight pages of data stored on the disc.

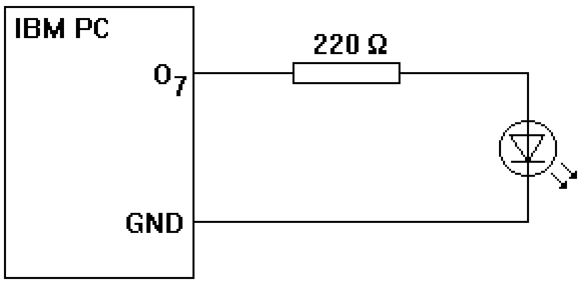
14 Use the -D command to verify that the original program is in place.

15 Connect a printer to the PC. Obtain a printout of the program as follows:

    press CTRL and P to turn the printer on
    type U E E A
    press CTRL and P to turn the printer off again.

How fast is your PC?

This exercise will allow you to measure the time taken for your PC to go through one fetch-execute cycle. You will need to connect a LED to the output port, as shown below. No external power supply is necessary.



1 Here is a program, in assembly code, which should turn the LED on and off rapidly. It starts at location E . You may need to replace the output port address in the first line to that of your own PC.

```
MOV DX,   378
MOV AL,8
OUT DX,AL
XOR AL,8
JMP   E  5
```

2 Use DEBUG to assemble the program as follows:
type A E
type each line of the program, ending each line with
press to stop assembling

3 Use the -U command to verify that the program has been entered correctly. Then save it on a floppy disc.

4 Use the -R IP command to set the instruction pointer to E . Then step through the program with the -T command. Pay particular attention to the state of AX and the LED. How many fetch-execute cycles are needed for O_7 to go from 0 to 1 and back to 0?

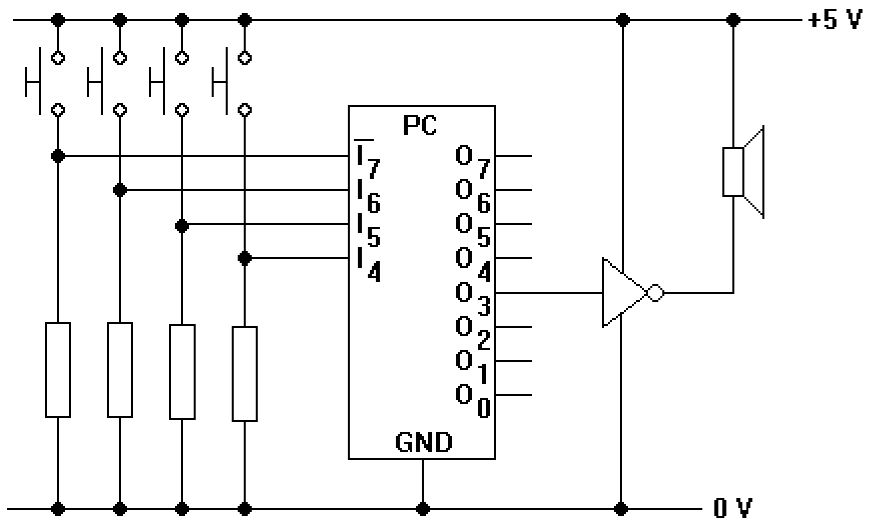
5 Once you are sure what the program makes the PC do, use the -G E command to run the program at full speed.

6 Use an oscilloscope to look at the square wave at O_7. Measure the period of the signal. Hence calculate an estimate for the average time it takes your PC to execute a single fetch-execute cycle.

7 You can only stop the program by pressing CTRL, ALT and DEL. Rebooting DEBUG might be quicker if the A drive contains a system disc with DEBUG on it.

Making noises with a PC

This exercise will show you how a couple of useful subroutines can be used to convert your PC into a simple musical instrument. You will need to connect some hardware to the printer port, as shown below. Use 220 pull-down resistors.



1 Load up DEBUG and use the -A command to enter this time delay subroutine in assembly code at location D .

```
PUSH AX
MOV AL,
DEC AL
JNZ  D 3
DEC CX
JNZ  D 1
POP AX
RET
```

2 Use the -E command to enter the start of this other program at address E . It sets the stack pointer to 7FF, loads 1 into CX and calls the subroutine at D .

```
BC FF  7 B9  2    BB     D FF D3 EB FC
```

3 Check the code with the -U command. Once you are happy that the program is correct, save the 8 bytes from address 8 on disc.

4 Use the -R IP command to set the instruction pointer to E . Use the -T command to step through the program. Pay particular attention to the IP and SP registers. Stop when you get stuck in the loop.

5 Use the -R IP command to set the instruction pointer to D A. Use the -T command to watch what happens to SP and IP as the program jumps out of the subroutine.

6 Here is another subroutine. It processes the data from the input port at 379 so that it can be used sensibly by other parts of the program.

```
BA 79   3 EC 34 8   D   E8 D   E8 D   E8 D   E8 C3
```

Use the -E command to enter the subroutine at address D1 . Use the -U command to verify that the code has been entered correctly. Then save the 8 pages below 8 onto disc.

7 Adapt the main program so that it calls the input port subroutine time after time. Step through the program so that you can find out how the information about the input port ends up in AL.

8 Now write a program in assembly code which does the following:

sets up the stack pointer
places 8 into AX
pushes AX onto the stack
calls the input port subroutine
transfers AX into CX
calls the time delay subroutine
pops AX off the stack
feeds AL to the output port
inverts the fourth bit of AL with AL AL 8
jumps to the third step

9 Start the program at address E . Use the -U command to check that you have calculated the jump address correctly.

10 Save the program and subroutines before going any further. Then step through the program, skipping the delay loop, until you are sure that it operates correctly. You should be able to hear a click from the speaker each time that the output port is accessed by the program.

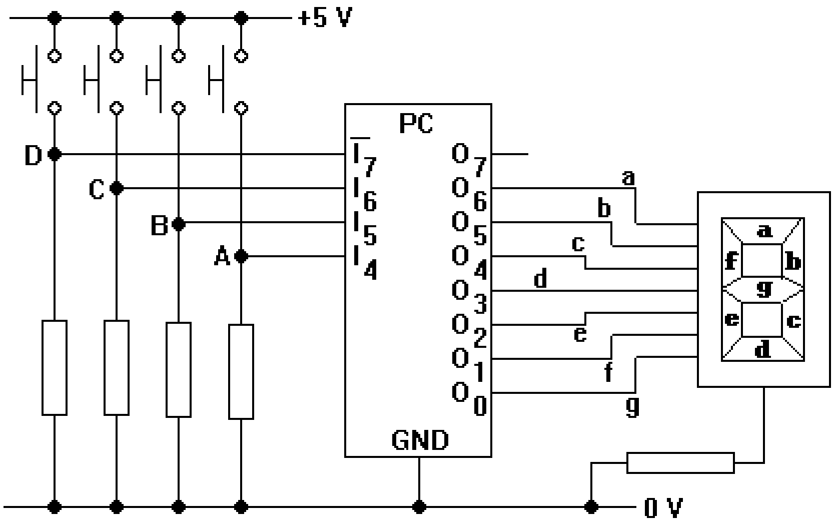
11 Set IP to E . Use the -G command to run the program at full speed. The noise from the speaker should change as you press the switches. You will have to use CTRL, ALT and DEL to stop the program running.

12 Use the -U command and CTRL-P to obtain a printed listing of your program.

13 Save the eight pages below 8 on disc as A:SUBS.MPC. This will save you having to write out the input port and time delay subroutines again. It might pay to make it a read-only file!

Nibble to hexadecimal conversion

This exercise will require you to write a look-up table which translates a nibble in binary at four switches into its hexadecimal equivalent on a seven-segment display. Connect some hardware to the printer port of your PC, as shown below. Use 220   resistors throughout.



1 Use the -O command in DEBUG to verify that you have connected the seven-segment display correctly. Then load the program A:SUBS.MPC into the eight pages of memory below 8   .

2 Use the -E command to enter the following program with its look-up table. You may need to alter the output port address at E F.

address	code
C	7E 3 3 6D 3 6D 6D 79 30 6D 6D 79 6D 79 79 33
E	BC FF 07
E 3	B8 C
E 6	BB 1 D
E 9	FF D3
E B	89 C3
E D	8A 7
E F	BA 38 7
E12	EE
E13	EB EE

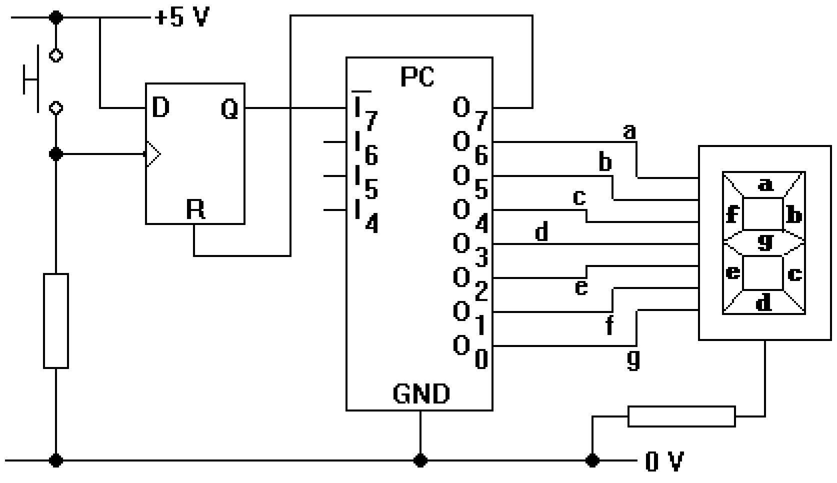
3 Use the -U command to check that the program has been entered correctly. Then save it (with a different name).

4 Use the -T command to step through the program, starting at E   . Then run the program at full speed. If all is well, the display should indicate the number of switches which are being pressed.

5 Rewrite the look-up table so that the LEDs display the hex equivalent of the nibble DCBA placed at the input port by the switches.

Counting down

You are going to write a program to make your PC count from three to zero each time a switch is pressed. The diagram shows the hardware required. Both resistors are 220 .



1 Load the program A:SUBS.MPC into the eight pages of memory below 8 . This will give you access to the input port and time delay subroutines.

2 Write a program, starting at E , which makes the system behave as follows:

reset the flip-flop and blank the display
wait until the switch is pressed
display the numbers 3, 2, 1 and 0 after the other, at one second intervals
return to the first step

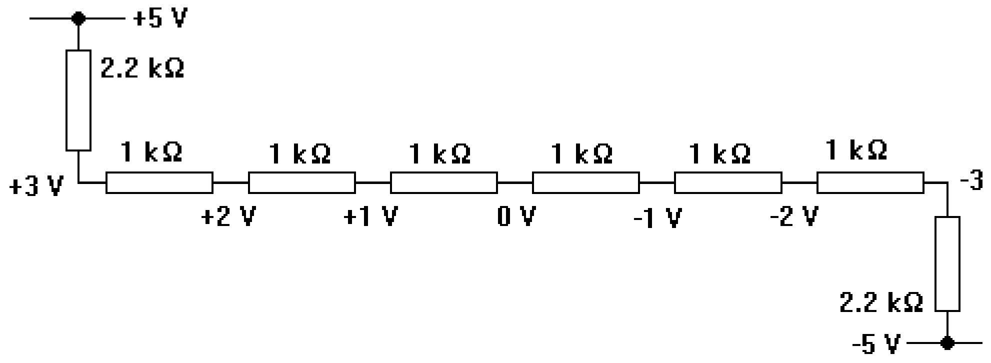
3 Use the -E command to enter the look-up table and program. Save it on disc *before* you run it at full speed as re-booting the PC is the only way of stopping the program!

4 When you have perfected the program, use the -U command and CTRL-P to obtain a printed listing.

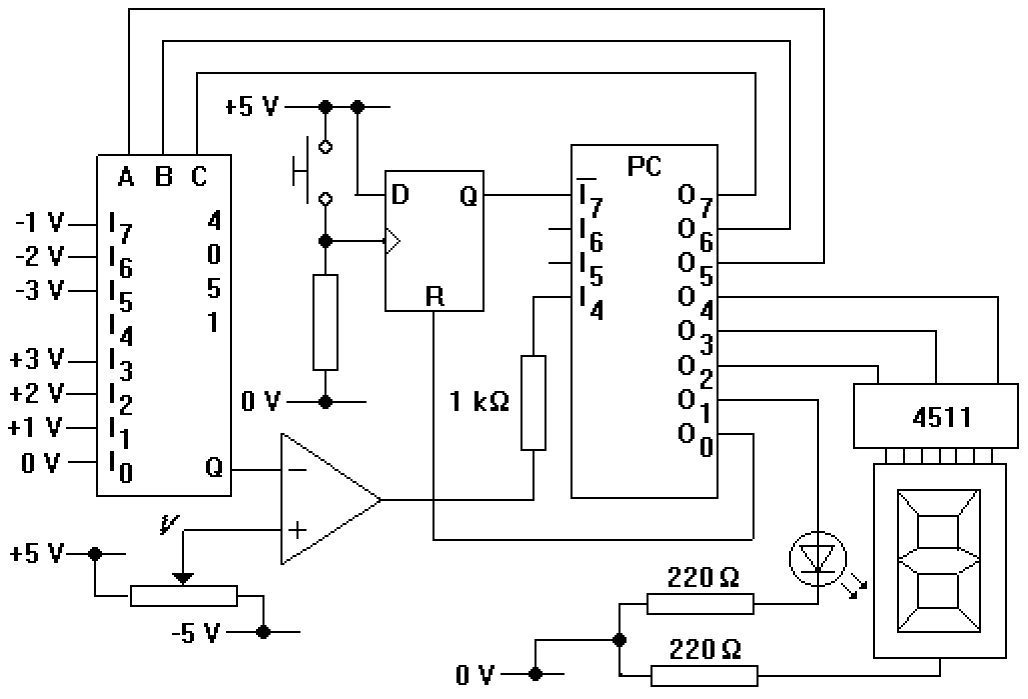
Adding an ADC to a PC

You are going to convert your PC into a simple digital voltmeter.

1 Assemble the resistor ladder shown below. Use a voltmeter to check that the seven reference voltages are approximately as indicated.



2 Now connect the seven reference voltages to the inputs of a 4051 multiplexer, as shown below. Pin 7 of the multiplexer i.c. will need to be connected to -5 V, to allow negative signals through to the output.


A B C
4051
I_7 I_6 I_5 I_4 I_3 I_2 I_1 I_0
Q
-1 V
-2 V
-3 V
+3 V
+2 V
+1 V
0 V
+5 V
D Q
R
0 V
1 kΩ
PC
$\overline{I_7}$ I_6 I_5 I_4
O_7 O_6 O_5 O_4 O_3 O_2 O_1 O_0
4511
V
+5 V
-5 V
220 Ω
220 Ω
0 V

3 You are going to program the PC so that whenever the switch is pressed, the LEDs display the value of *V*. The program will need to go through the following steps:

reset the flip-flop and wait until the switch is pressed
increase the voltage at Q from -3 V to +3 V until the output of the op-amp goes low
feed appropriate signals to the 4511 decoder and the single LED
return to the first step

You will need to use the look-up tables!