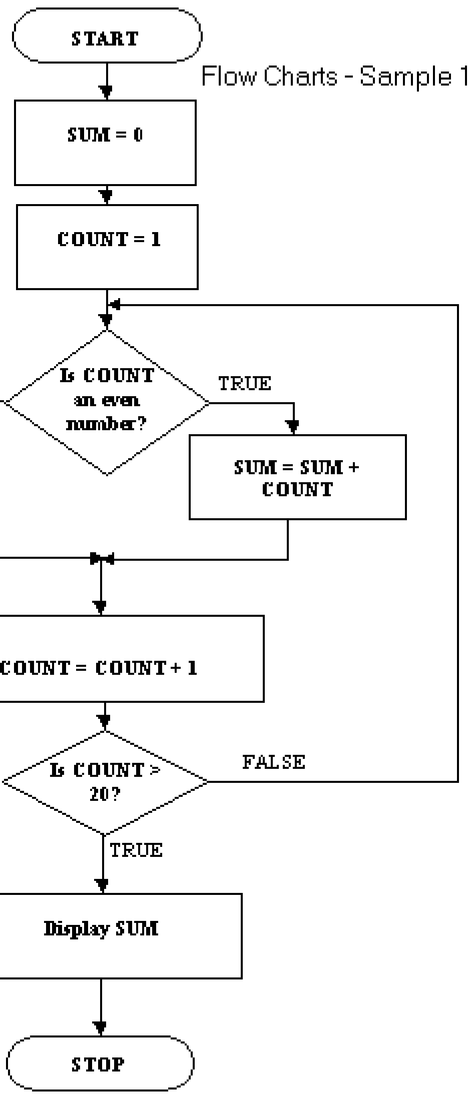
Engineering Science

Basic Flowcharts / Programming

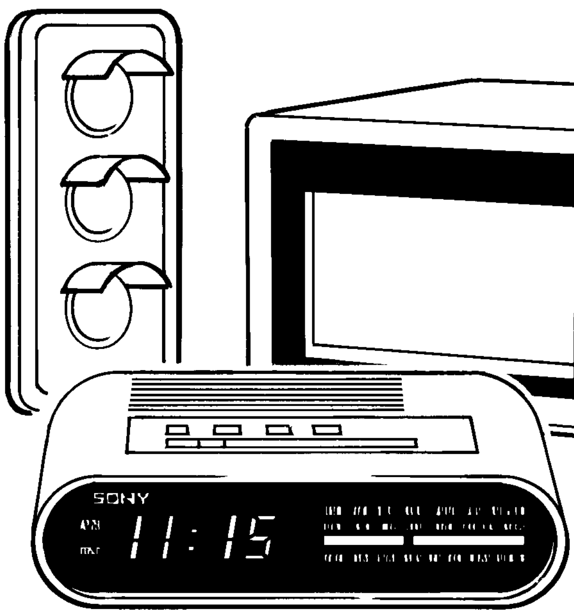


Name .... Class ....

Section 1 - Introduction

What is a microcontroller?

A **microcontroller** is often described as a 'computer-on-a-chip'. Microcontrollers have memory, processing units, and input/output circuitry all built into a single chip. As they are small and inexpensive they can easily be built into other devices to make these products more intelligent and easier to use.



Microcontrollers are usually programmed to perform one specific control task - for instance, a microwave oven may use a single microcontroller to process information from the keypads, display user information on the seven segment display, and control the output devices (turntable motor, light, bell and magnetron).

Microcontrollers are computers designed to control specific processes or products. The microcontroller is programmed with a specific software program to complete the desired task. By altering this software program the same microcontroller can be used to complete different tasks. Therefore the same device can be used in a range of different products by simply programming it with a different software program.

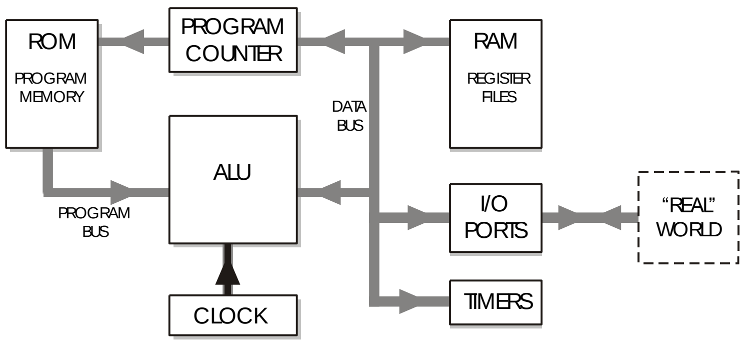
One microcontroller can often replace a number of separate parts, or even a complete electronic circuit. Some of the advantages of using microcontrollers in a product design are:

- increased reliability and reduced stock inventory (as one microcontroller replaces several parts)
- simplified product assembly and smaller end products
- greater product flexibility and adaptability since features are programmed into the microcontroller and not built into the electronic hardware
- rapid product changes or development by changing the program and not the electronic hardware

Applications that use microcontrollers include household appliances, alarm systems, medical equipment, vehicle subsystems, and electronic instrumentation. Although microprocessor systems (such as those based around the Intel Pentium™ processor) tend to be more widely publicised (mainly via personal computer systems), microcontroller manufacturers actually sell hundreds of microcontrollers for every microprocessor sold.

Microcontroller Architecture

The main features of the microcontroller are shown in the block diagram.



SIMPLIFIED PIC MICROCONTROLLER
BLOCK DIAGRAM

Microcontrollers contain all these features within a single package, as opposed to the microprocessor system where each block in the diagram above is normally a separate integrated circuit. In general the only component that needs to be added to a microcontroller is a clock resonator, which sets the operating speed of the microcontroller.

Arithmetic / Logic Unit (ALU) and Clock

The processing unit (full name **arithmetic and logic unit (ALU)**) is the 'brain' of the microcontroller. It operates by reading instructions from the **read only memory ROM** (permanent program memory) and then carrying out the mathematical operations for each instruction. The speed at which these operations occur is controlled by the clock circuit.

The **clock** circuit within the microcontroller 'synchronises' all the internal blocks (ALU, ROM, RAM etc.) so that the system remains stable. The clock circuit is built into the microcontroller, but an external crystal or resonator is required to set the clock frequency. A typical clock frequency for use with a microcontroller is 4MHz, but speeds as high as 20MHz can also be achieved. With a clock frequency of 4MHz the microcontroller completes one million instructions a second!

Memory (ROM and RAM)

Microcontrollers contain both **ROM** (permanent memory) and **RAM** (temporary memory).

The ROM (Read Only Memory) contains the operating instructions (i.e. the 'program') for the microcontroller. The ROM is 'programmed' before the microcontroller is installed in the target system, and the memory retains the information even when the power is removed. Most microcontrollers are one-time-programmable types, which means the ROM can only be programmed once. If you make a mistake, and have to change the program, the chip has to be thrown away and a new chip programmed with the revised program. To overcome this problem some microcontrollers now use FLASH EEPROM memory instead. This type of 'erasable-permanent' memory allows the ROM to be re-programmed if a mistake is made.

The RAM (Random Access Memory) is 'temporary' memory used for storing information whilst the program is running. This memory is 'volatile', which means that as soon as the power is disconnected the contents of the memory is lost.

Buses.

Information is carried between the various blocks of the microcontroller along 'groups' of wires called **buses**. The 'data bus' carries the 8-bit data between the ALU and RAM / Input-Output registers, and the 'program bus' carries the 13-bit program instructions from the ROM.

The size of the data bus provides a description for the microcontroller. Therefore an '8 bit microcontroller' has a data bus '8-bits' wide. Microcontrollers with 16-bit and 32-bit data buses are also available.

Input/Output Circuitry

Microcontrollers communicate with the outside world via pins which are grouped together in 'ports', with up to eight pins in each port. Smaller microcontrollers may only have one port, whilst larger devices may have five or more. Generally each pin within the port can be configured as an output or as an input, or can even be multiplexed to change functions as the program is run!

The CMOS fabrication techniques used to build modern microcontrollers provides a relatively high current capability (approx. 20mA) for each pin. However further 'interfacing' circuits are required for most output devices.

Timers

Most microcontrollers have one or more 'timers' built into the system. The 'watchdog timer' is the most common type of timer. This is a special timer that 'resets' the microcontroller if it stops processing for any reason (e.g. a 'bug' in the program). This ensures that the microcontroller continues working at all times - which is essential in some applications, for instance medical monitoring equipment.

Assignments:

1.1) List the advantages of using a microcontroller within a product design.

1.2) Describe the input sensors and output transducers that may be linked to a microcontroller in the following common household appliances:

- microwave oven

- washing machine

- electronic bicycle speedometer

1.2) Explain the following microcontroller terms: ALU, bus, clock

1.3) Explain the differences between the following types of memory:
- RAM, ROM, EEPROM

Port Addressing and the Data Direction Register

Microcontrollers communicate with the outside world by input/output pins which are grouped together in **'ports'**.

Each pin can be addressed individually, or all eight pins in the port can be addressed simultaneously. In the PBASIC language the pins are labelled 0 to 7, and the whole port address is allocated the label **'pins'**.

For example, to switch pin b.3 'high' individually the command would be:
```
high b.3
```

To switch pin b.3 'low' individually the command would be:
```
low b.3
```

Each pin is referenced to by a single bit in the port address called **'pins'**.

input/output pin	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
bit of address 'pins'	7	6	5	4	3	2	1	0
decimal value	128	64	32	16	8	4	2	1

To switch all pins 'high' the command would be
```
let pinsb = 255
```

To switch all pins 'low' the command would be
```
let pinsb = 0
```

To switch pins b.0-b.2 'high' and pins b.3-b.7 'low' the command would be
```
let pinsb = %00000111
```

Note how the use of the binary number system can be used on this occasion to clearly illustrate which pins are switched high (=1) or low (=0).

Each pin can be configured to be an output (to send digital signals) or an input (to receive digital signals). The **Data Direction Register (DDR)** is used to configure the port, and in the PBASIC language the DDR is allocated the label **'dirs'**.

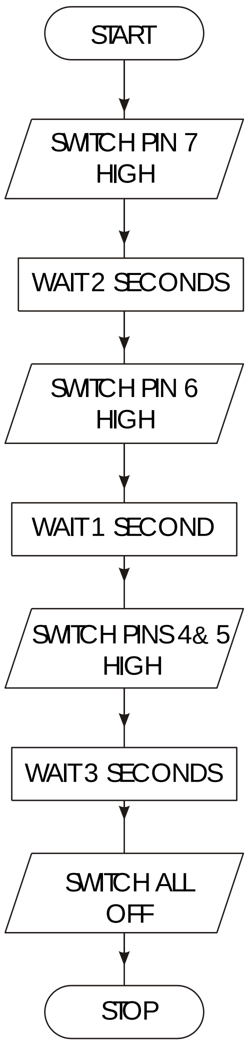
If all the bits in the DDR are set high then all the pins will be set as outputs. If all the bits are set low each pin will be set as an input.

For example,

```
let dirsb = 255              ' set all pins as outputs
let dirsb = 0                ' set all pins as inputs
let dirsb = %11110000        ' set 0-3 inputs, 4-7 outputs
```

Beginning Programming

A simple example of a control operation is represented by the flowchart shown below.



A PBASIC program which would achieve this control operation is:

```
let dirsb = %11110000          ' set pins 0-3 inputs, 4-7 outputs

high b.7                       ' set pin 7 high
pause 2000                     ' wait for 2 seconds (= 2000 ms)
high b.6                       ' set pin 6 high
pause 1000                     ' wait for 1 second
let pinsb = %11110000          ' set pins 4-7 high
pause 3000                     ' wait for 3 seconds
let pinsb = 0                  ' switch all pins low
end                            ' end the program
```

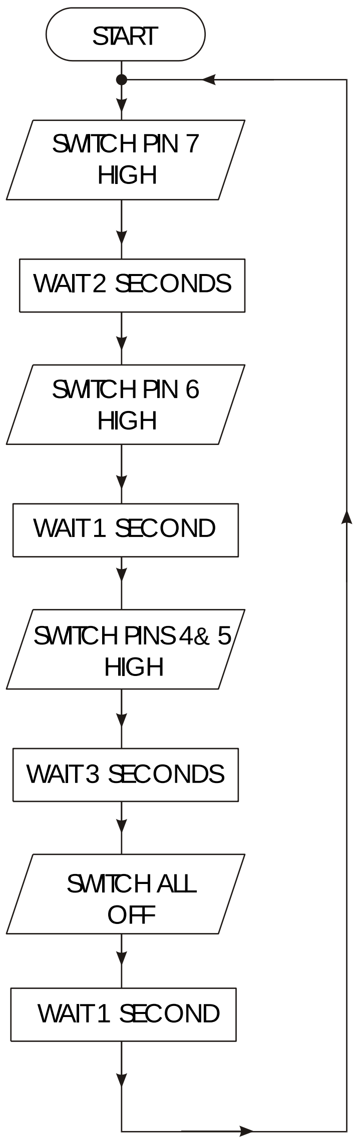
Activity 3.1
Key in the program listed above and then simulate it in 18M2 mode.

Pin b.7 should light first, followed by the indicators on pin b.6, and then indicators b.4 and b.5.

Read through the program carefully, and make sure you understand exactly what each program line achieves.

Note the 'comments' at the end of each line. A comment starts after an apostrophe (') and continues to the end of the line. Although the comments are not needed to make the program work, they are an essential part of the program as they explain in 'plain language' what the program is doing. You should **always add a comment** to every line of your program, particularly if the program is to be studied by someone else at a later date.

Labels and Addressing



Sometimes it is necessary to create programs that loop 'forever', as shown by the flowchart. In this case it is necessary to add labels to the program, and to use the **'goto'** command to jump to the line marked by the label.

A PBASIC program which would achieve this control operation is listed below.

```
init: let dirsb = %11110000   ' set pins 4-7 as outputs

main: high b.7                ' set pin 7 high
      pause 2000              ' wait for 2 seconds
      high b.6                ' set pin 6 high
      pause 1000              ' wait for 1 second
      let pinsb = %11110000   ' set pins 4-7 high
      pause 3000              ' wait for 3 seconds
      let pinsb = 0           ' switch all pins low
      pause 1000              ' wait for 1 second
      goto main               ' loop forever
```

Activity 3.2
Key in and simulate the program listed above in 18M2 mode.

After the first line (which simply sets up the DDR), a label called 'main' has been added to the listing. Note that all address labels must end with a colon (:) when they are first defined. It is also a good programming technique to use tabs (or spaces) at the start of lines without labels so that all the commands are neatly aligned. The term **'white-space'** is used by programmers to define tabs, spaces and blank lines, and the correct use of white-space can make the program listing much easier to read and understand.

The last line 'goto main' causes program flow to 'jump back' to the line labelled 'main'. This means that this program will loop 'forever'.

Note:
Some early BASIC languages used **'line numbers'** rather than **labels** for 'goto' commands. Unfortunately this line number system can be inconvenient to use, because if you modify your program by later adding, or removing, lines of code you then have to modify all the line numbers within the 'goto' commands accordingly. The label system, as used in most modern BASIC languages, overcomes this problem automatically.

Assignments
3.3) What is the function of the Data Direction Register (DDR)?

3.4) What is meant by the term "white-space"? Why is it important to use white-space and comments when writing programs?

Assignment 3.3

A set of temporary traffic lights are required for a system of road-works.

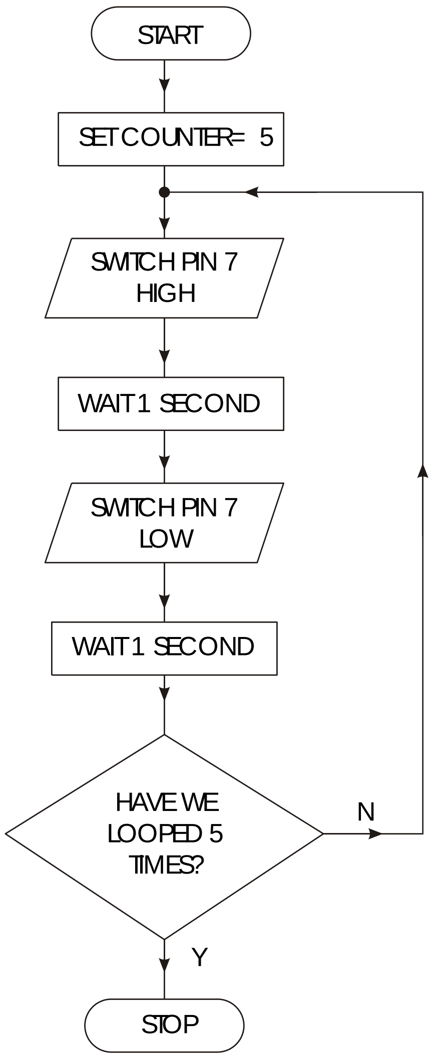
red	10 sec
red and amber	2 sec
green	10 sec
amber	2 sec

Draw a flowchart for the lights sequence shown by one set of the traffic lights. Use the times shown in the table for each stage.

Write a PBASIC program to achieve this operation. Use the following pin configuration - red (7), amber (6) and green (5).

Flowchart	Program

For...Next Loops



A **for...next** loop is used when you wish to repeat a section of code a number of times. The number of times the program runs for is set by a variable. A variable is a number that is stored in the RAM memory of the PicAXE Controller. There are 14 memory locations that byte variables can be stored in. These locations are labelled b0 to b13, but can also be 'renamed' to more appropriate names by use of the **'symbol'** command.

Activity 3.4
Key in, download and run the following program.

```
symbol counter = b0     ' define the variable "counter"
symbol red = 7          ' define pin 7 with the name "red"

init: let dirsb = %10000000   ' set up pin 7 as an output

main: for counter = 1 to 5    ' start a for...next loop
         high red             ' switch pin 7 high
         pause 1000           ' wait for 1 second
         low red              ' switch pin 7 low
         pause 1000           ' wait for 1 second
      next counter            ' end of for...next loop

      end                     ' end program
```

Note again how white-space (extra spaces) has been used to clearly illustrate all the commands that are contained between the **for** and **next** commands. The 'symbol' command is also used to label pins and variables to make them easier to use.

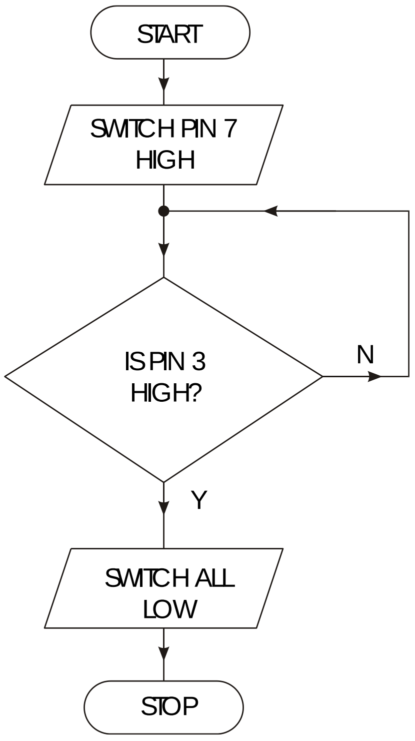
Assignment 3.9
As part of a Christmas decoration, a lighting sequence is to be controlled by a microcontroller. The output connections are shown below.

Input Connection	Pin	Output Connection
	b.7	
	b.6	Yellow Light
	b.5	Purple Light
	b.4	
	b.3	Blue Light
	b.2	Green Light
	b.1	
	b.0	Red Light

The red and green lights should come on together and stay on for 5 seconds. Then they both go off and the yellow and blue lights should come on together for 8 seconds. They then go off and the purple light flashes on and off 6 times (the 'on' and 'off' times being 0.5 seconds each). The sequence then repeats itself.
Draw a flowchart and write a PBASIC program for this sequence.

Flowchart	Program

If...Then...



The **if...then** programming structure allows the computer to make a decision based on information received from an input pin.

The following program switches pin 7 high, and then waits for an input connected to pin 3 to go high. When the input switch is pushed the output pin is switched low

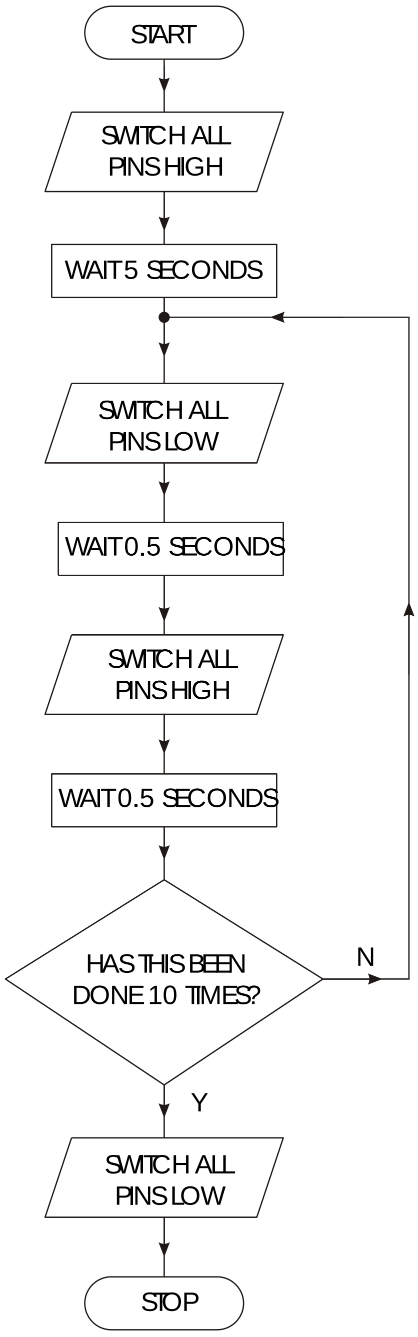
Activity 3.10
Key in and simulate the program below:

```
init: let dirsb = %10000000   ' setup the DDR

main: let pinsb = %10000000   ' switch pin 7 high
      if pinb.3 = 1 then skip ' jump to 'skip' if input 3 is high
      goto main               ' loop

skip: let pinsb = 0           ' switch all pins off
      end                     ' end the program
```

Note:
Unlike some other BASIC languages, the **then** command can only be followed by a label to jump to. You cannot add extra commands on the same line within the PBASIC language.

Assignment 3.5

Develop a PBASIC program that will carry out the instructions shown in the flowchart below.



Program

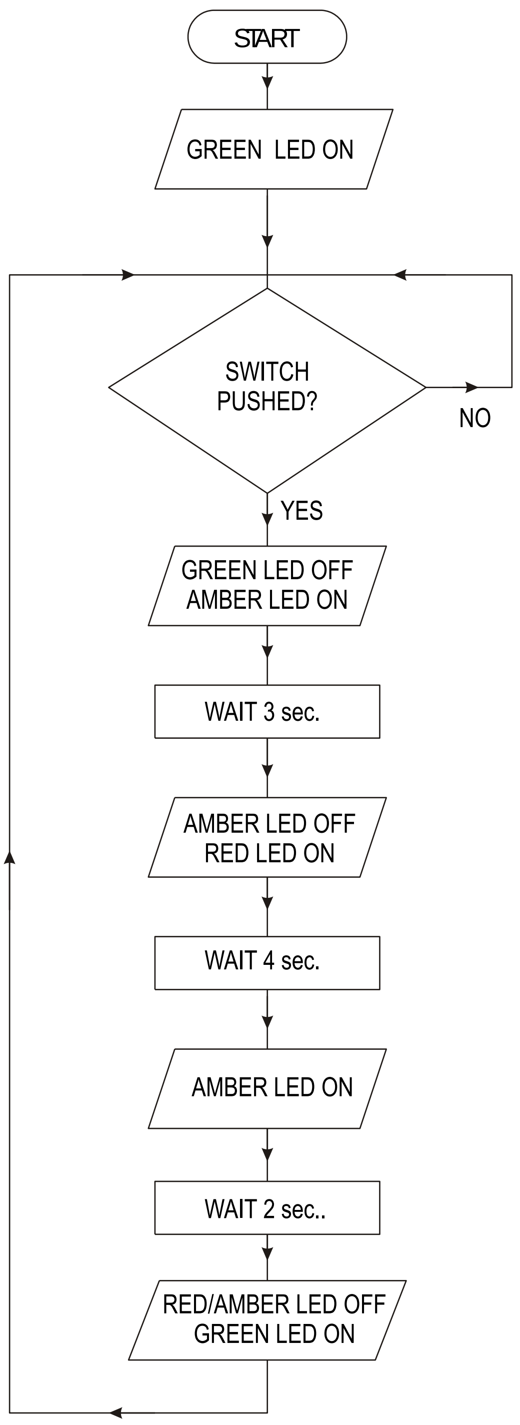
Assignment 3.6
A motor, connected to pin b.7, is to run when a 'start' switch (connected to pin c.7) is momentarily pressed. The motor continues to run until another 'stop' switch (connected to pin c.6) is pressed - at which point the motor switches off. The system should then reset to wait for another 'start' signal.

Draw a flowchart and write a PBASIC program for this sequence. Test this on the PicAXE board.

Flowchart	Program

Assignment 3.7

Develop a PBASIC program that will carry out the instructions shown in the flowchart above. Use the pin configuration on the next page in 8M2 mode

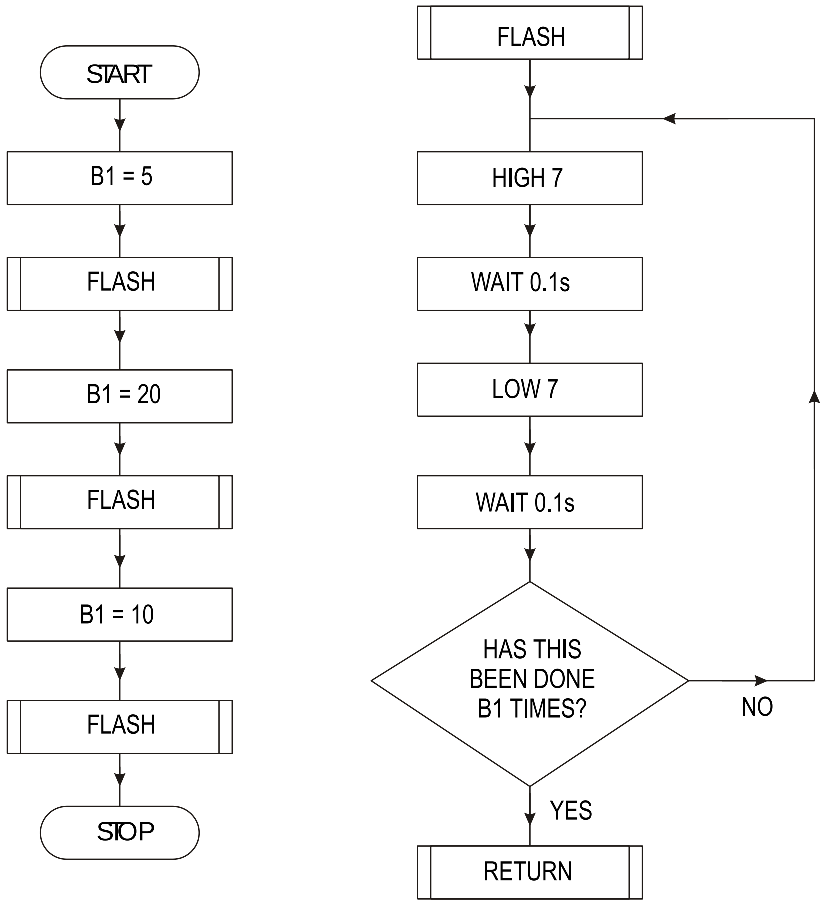


Program

Input Connection	Pin	Output Connection
Start Switch	3	
	2	Green Light
	1	Amber Light
	0	Red Light

Now test your program on the 8M2 board.

Sub-Procedures



It is often useful to be able to re-use sections of code within a program. A **sub-procedure** is a small section of code that can be 'called' from a different part of the program. After the sub-procedure is finished program flow moves back to the original section of the program.

To 'call' a sub-procedure the **gosub** (go-to-sub-procedure) command is used. The last line of the sub-procedure must be **return**, which means 'return to the original position'.

Activity 3.8
Key in, download and run the following program.

```
init: let dirsb = %10000000    ' setup the DDR

main: let pinsb = %10000000    ' switch pin 7 high

      let b1 = 5               ' give variable b1 the value 5
      gosub flash              ' call sub-procedure
      pause 2000               ' wait two seconds

      let b1 = 20              ' give variable b1 the value 20
      gosub flash              ' call sub-procedure
      pause 2000               ' wait two seconds

      let b1 = 10              ' give variable b1 the value 10
      gosub flash              ' call sub-procedure
      pause 2000               ' wait two seconds

      end                      ' end the main program

' Sub-procedures start here.

flash:
      for b2 = 1 to b1         ' setup a for...next loop using b2
         high b.7              ' output pin on
         pause 100             ' wait 100ms
         low b.7               ' output pin off
         pause 100             ' wait 100ms
      next b2                  ' next loop
      return                   ' return form sub-procedure
```

In this example the **flash** sub-procedure is used to actually flash the LED on and off. The number of times the LED flashes is set by variable b1, which is set before the sub-procedure is called. Therefore by changing the value of b1 the number of flashes can be changed.

Note the **end** command between the main program and the sub-procedure. This is essential because it stops the system accidentally 'running into' the sub-procedure after the main program has been completed.

One further advantage of sub-procedures is that they are easily copied from one program to another. Therefore a 'standard' sub-procedure can be created to carry out a specific task. This task can then be carried out in a number of different programs by simply copying the sub-procedure from program to program.

Assignment 3.9
As part of a shop display, a lighting sequence is to be controlled by a microcontroller. The output connections are shown below.

Input Connection	Pin	Output Connection
	b.7	Red Light
	b.6	Yellow Light
	b.5	Green Light
	b.4	Orange Light
	b.3	
switch 2	b.2	
switch 1	b.1	
	b.0	

If switch 1 is pressed, each light should flash on and off 5 times in sequence i.e. orange then green then yellow then red (each 'on' and 'off' time should be 0.5 seconds). the system then resets for the next switch push.
If switch 2 is pressed all lights should flash simultaneously 3 times. There should then be a 2 second pause, before all lights should flash simultaneously 6 times (each 'on' and 'off' time should be 0.5 seconds). The system then resets for the next switch push.

Draw a flowchart and write a PBASIC program for this sequence (use a sub-procedure for the flash routines).

Simulate your program in 18M2 mode.

Flowchart	Program