Technological Studies

Systems and Control

Students' Notes

Higher

4558

Spring 1999

HIGHER STILL

DET: Technological Studies

Systems and Control
Students' Notes
Higher

Support Materials

TECHNOLOGICAL STUDIES

HIGHER

SYSTEMS AND CONTROL

STUDENTS' NOTES

OUTCOME 1

Outcome 1 - Control Systems

The purpose of this unit is to introduce the operation of control systems.

When you have completed this unit you should be able to:

- recognise and identify common control systems
- describe the operation of open and closed loop control systems
- make use of systems diagrams and systems technology

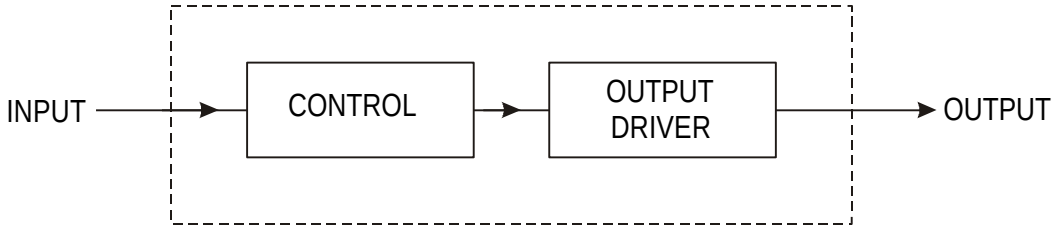
Before you start this unit you should have a basic understanding of:

- Block diagrams

Systems Diagrams

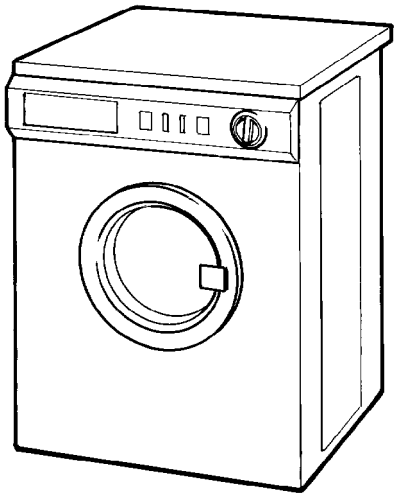
Most industrial product design is solved by the systems approach. This approach involves studying the desired function of the product, and then breaking this function down into a series of subsystems.

When applied to control systems, a **Systems Diagram** is a useful way of visually representing the desired function of the system. The systems diagram is a form of block diagram than contains all the subsystems within a dashed box, called the **systems boundary**. The systems boundary indicates the extent of the control system.

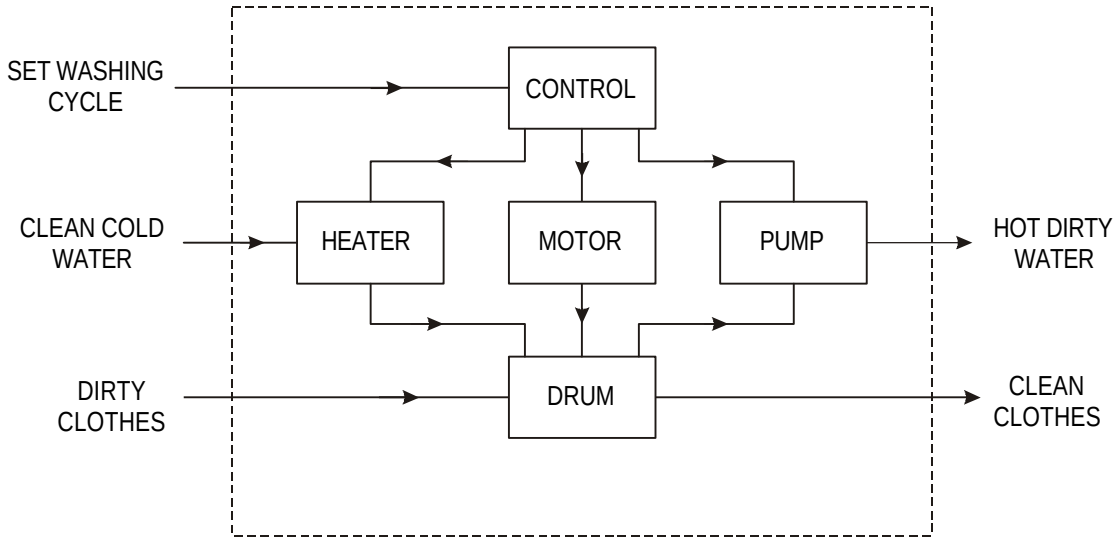


The "real world" input and output conditions of the system are shown as arrows entering, and leaving, the systems diagram.

A product which has a fairly complicated control system is an automated washing machine.



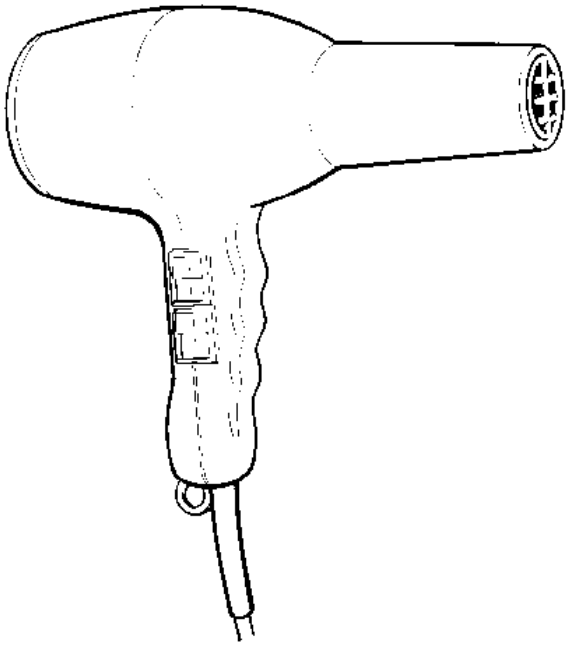
A simplified systems diagram of a washing machine is shown below.



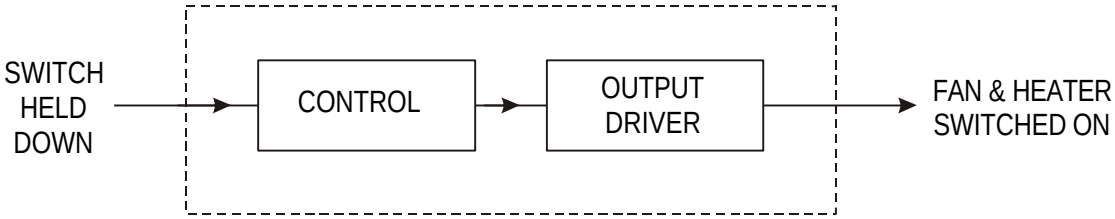
The function of the washing machine is to process dirty clothes to produce clean clothes. This is clearly represented within the systems diagram.

Open Loop Control

At the simplest level a control system can process an input condition to produce a specified output.

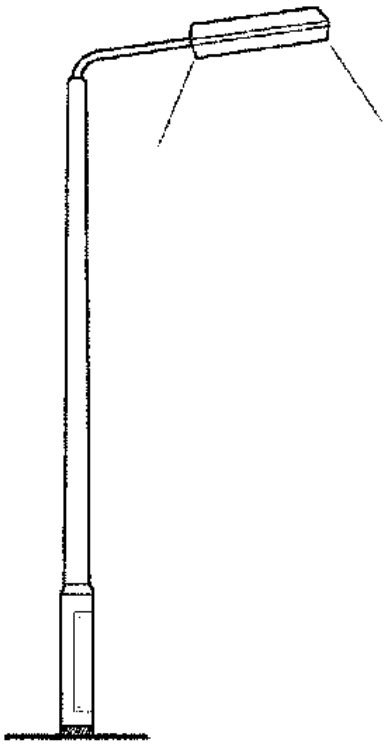


A good example of this type of system is a hand-held electric hairdryer. The heating element and fan motor are switched on when the appropriate switches are held down.

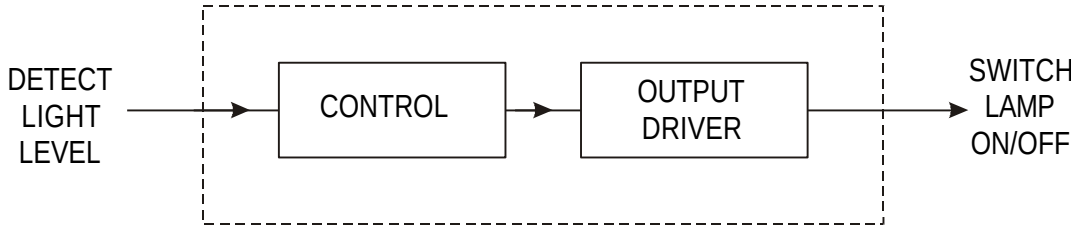


This is an example of **Open Loop Control**, where an input is processed to produce an output. With the hairdryer example the heater and fan motor are held on until the switches are released. The air being blown out of the hairdryer is not temperature monitored or adjusted - the air is simply just blown out at whatever temperature the heater is capable of achieving.

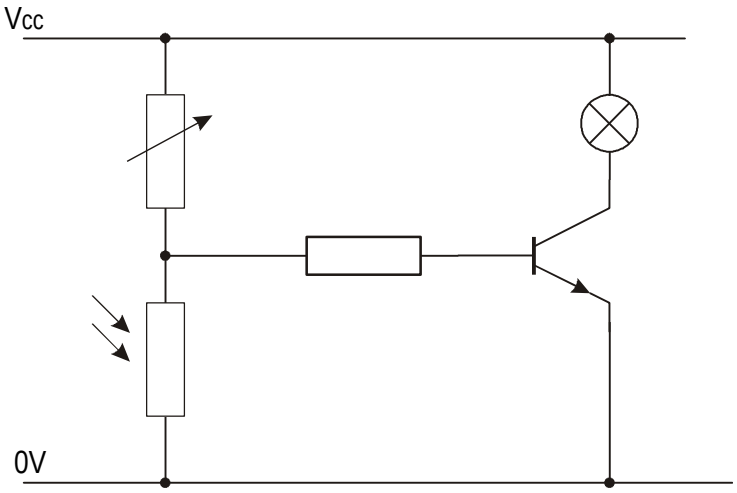
Another example of an open loop system is an automatic street lamp.



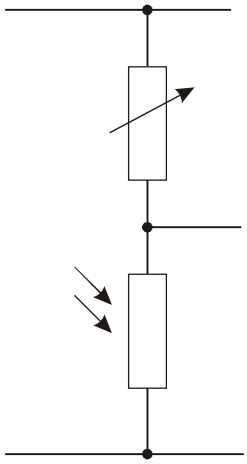
The systems diagram for the street lamp is shown below.



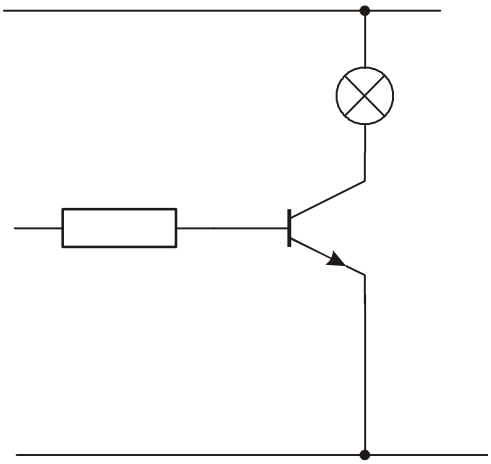
A model of the street lamp can be represented by the following circuit diagram:



The input to the circuit, light, is processed by the control subsystem to produce an electrical signal. In this example the potential divider arrangement provides a varying voltage signal.



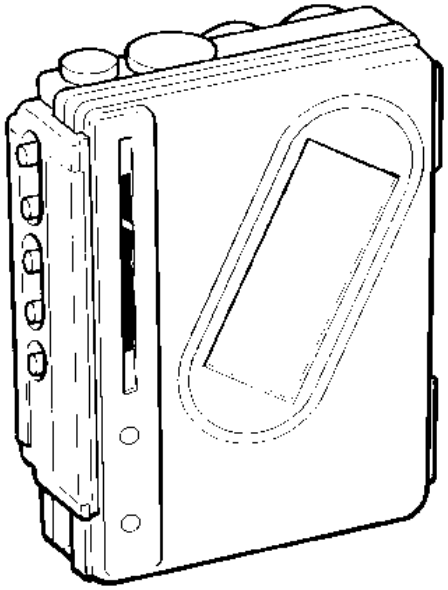
The electrical signal is processed by the output driver subsystem to produce the output. In this example the output is light from the signal lamp



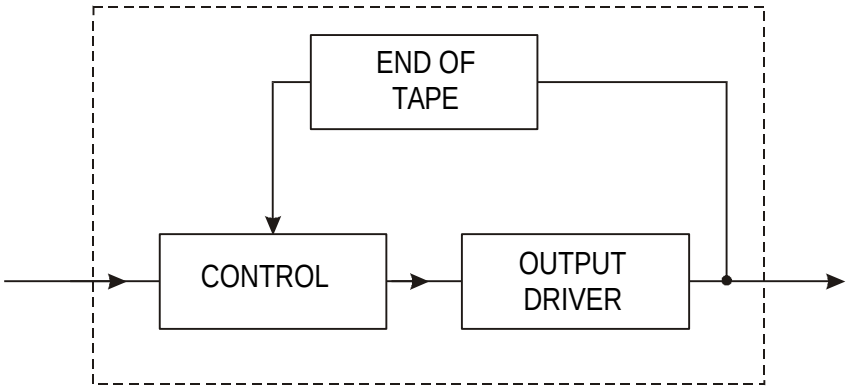
An open loop control system represents the simplest and cheapest form of control. However, although open loop control has many application, the basic weakness in this type of control lies in the lack of capability to adjust to suit the changing output requirements.

Closed Loop Control

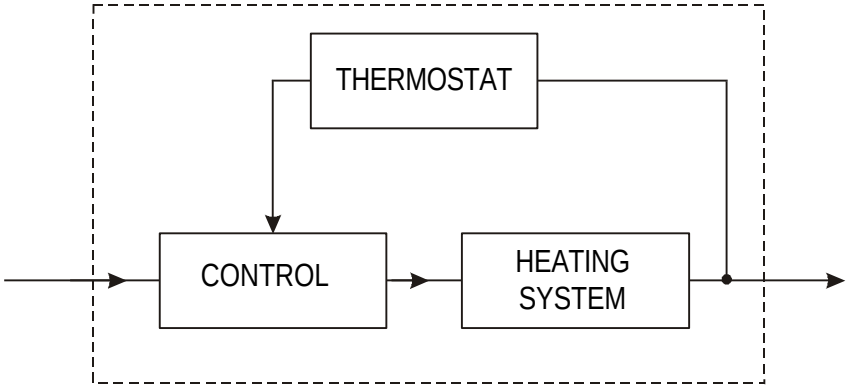
Closed loop control systems are capable of making decisions and adjusting their performance to suit changing output conditions.



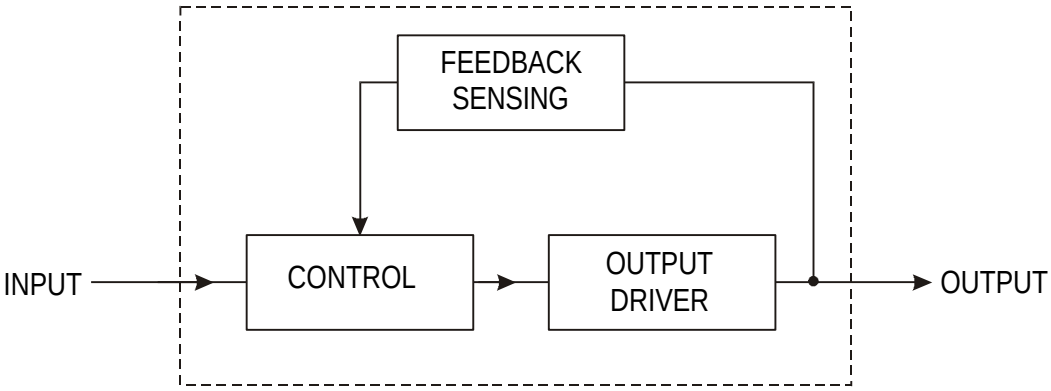
A personal cassette player is capable of detecting the end of the tape and switching the motor off, hence protecting the tape from snapping (or the motor burning out).



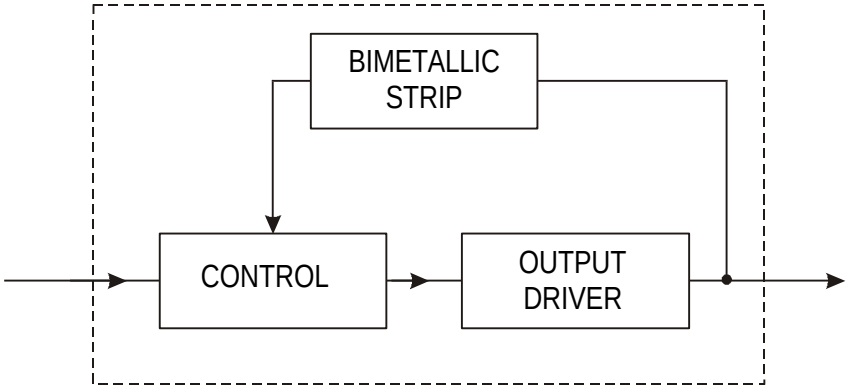
The central heating in most modern houses is controlled by a thermostat. It senses changes in temperature and turns off the heating system automatically when the required temperature is reached. Likewise when the temperature level drops below the minimum acceptable level the heating system is automatically switched back on.



All closed loop control systems include a **feedback sensing** subsystem within the systems diagram. The control subsystem will process the feedback signal by making a 'decision' on whether the state of the output should change.



Another example of a simple closed loop control system is an electric toaster. In this case the feedback sensing system is a mechanical bimetallic strip. This strip bends when heated - the higher the temperature the more the strip bends.

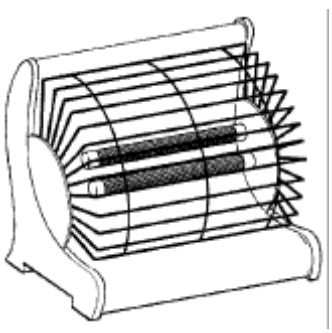


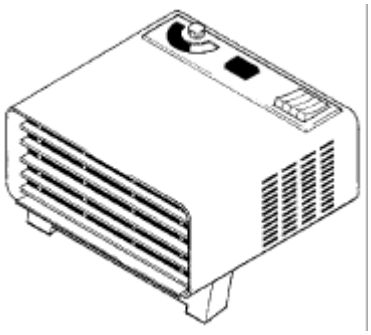
When the toaster cage is lowered into the toaster it compresses a spring, and the cage is 'hooked' into position by a catch attached to a bimetallic strip. A switch also connects the heating element, which starts to heat up. The heating element starts to cook the bread, and also starts to heat the bimetallic strip. When the bimetallic strip has been heated to the required temperature it bends enough to mechanically release the spring holding the cage down. This automatically ejects the toast and switches the heating element off.

Assignments:

1) Explain what is meant by the term 'systems diagram'.
2) Describe the differences between an open loop and closed loop control system.
3) Describe the purpose of the feedback sensing subsystem in a closed loop control system.

Assignment 4





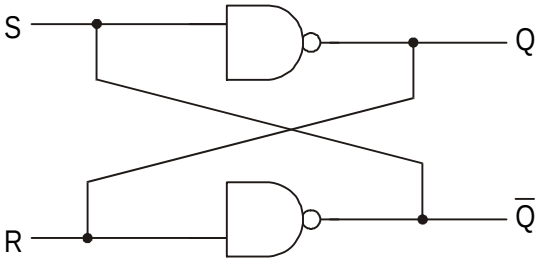
The figures above show two types of electric fire. The second electric fire is a more modern device fitted with a thermostat.

a) Name the type of control system used in each type of electric fire
b) Draw a system diagram for each type of electric fire.
c) Name a type of electronic sensor that may used for measuring temperature.

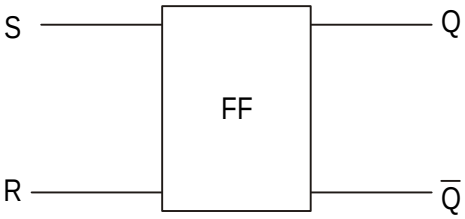
Bistable Two State Closed Loop Control

The most basic 'decision' which a closed loop control system can be required to make involves switching the system off (or resetting the system) when the output reaches a maximum or minimum output state level. Many safety systems incorporate this form of control as a means of switching off a system when a dangerous condition is reached.

This form of control can be based around a sequential logic device called the S-R bistable or latch. An S-R bistable can be represented by one of two symbols.

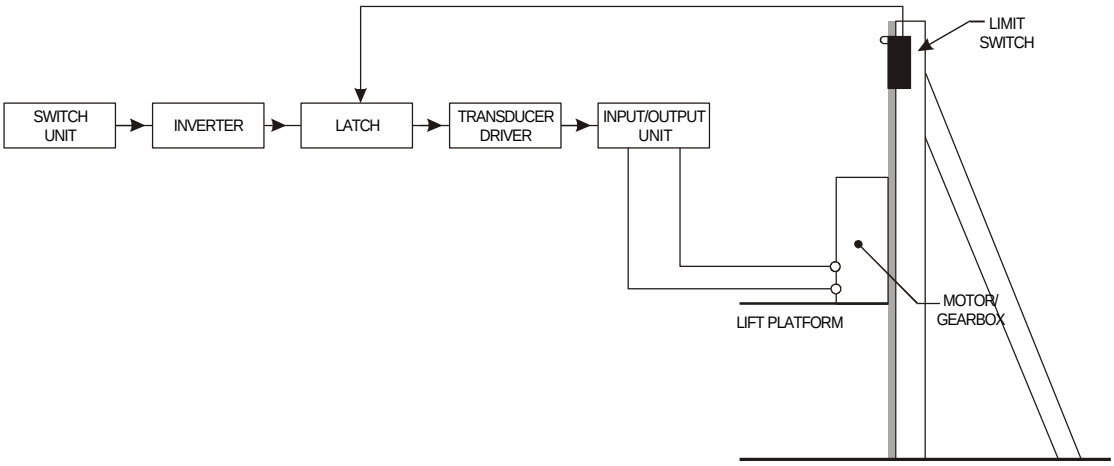


CROSS-COUPLED
NAND GATE
S-R BISTABLE

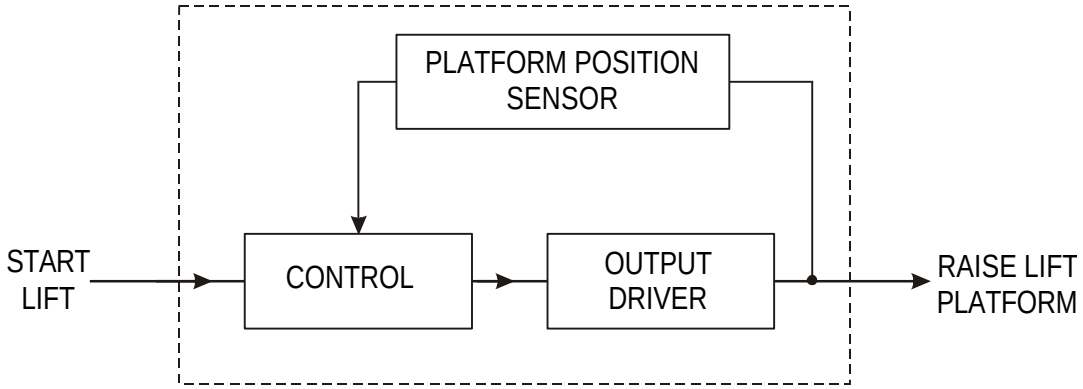


S-R BISTABLE
SYMBOL

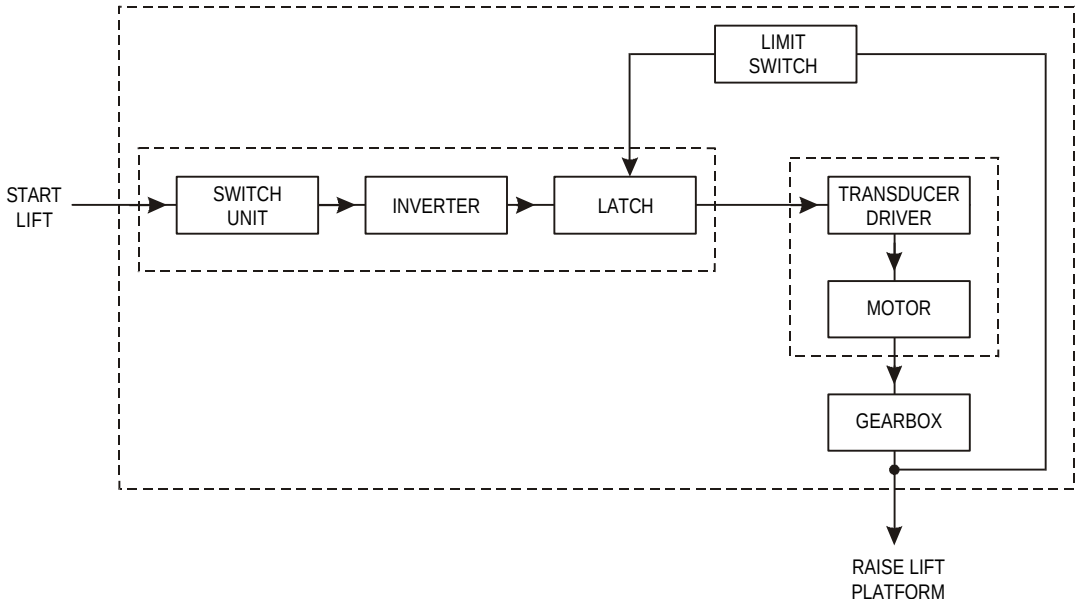
Assignment 5



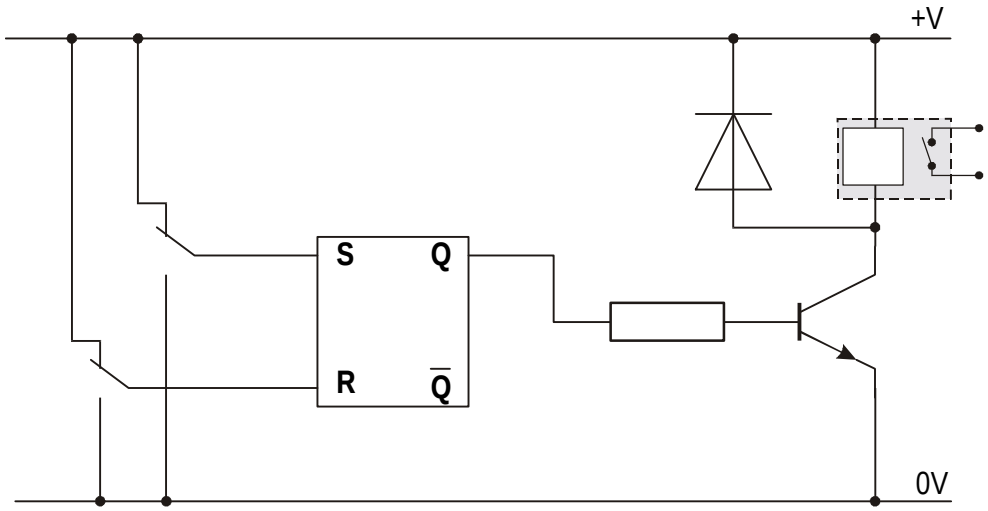
The systems diagram for a simple lift control system is shown below.



A limit switch at the top of the lift detects when the lift platform has reached the top of the run. The block diagram below shows the system diagram sub-systems broken down into smaller blocks.



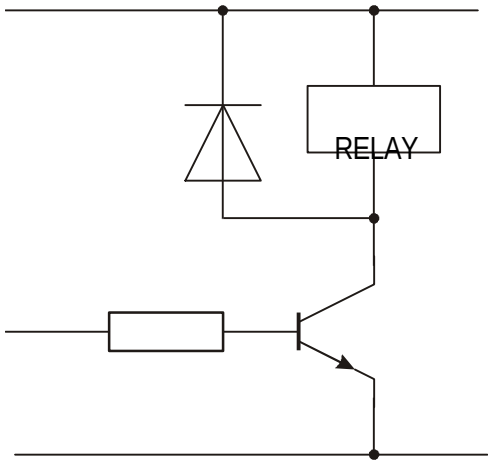
A circuit diagram of the system is shown below.



Build the circuit shown, using discrete components, circuit simulation or a modular electronics system.

a) Clearly explain how the system operates.

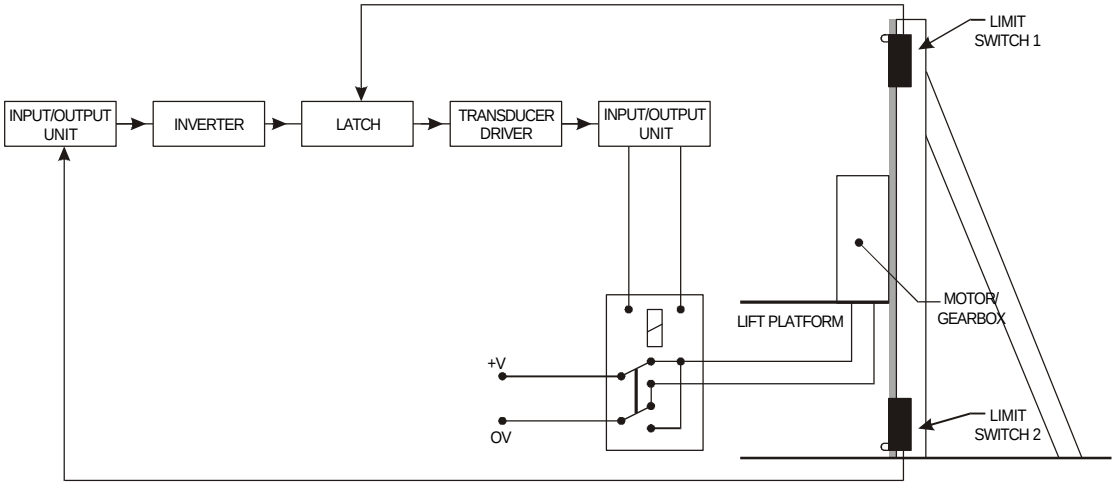
The 'transducer driver' circuit used in this instance is a **relay driver** circuit.



b) Explain the purpose of the relay, transistor and diode within the relay driver circuit.

Assignment 6

The output condition using this type of control does not have to be limited to on or off but can be applied to any two desired output states.



Build the circuit shown, using discrete components or a modular electronics system.

a) Draw a system diagram of the control system.

b) Draw a circuit diagram of the control system.

c) Explain the operation (function and effect) of the DPDT relay.

TECHNOLOGICAL STUDIES

HIGHER

SYSTEMS AND CONTROL

STUDENTS' NOTES

OUTCOME 2

Outcome 2 - Analogue Control Systems

The purpose of this unit is to introduce the operation of analogue control systems.

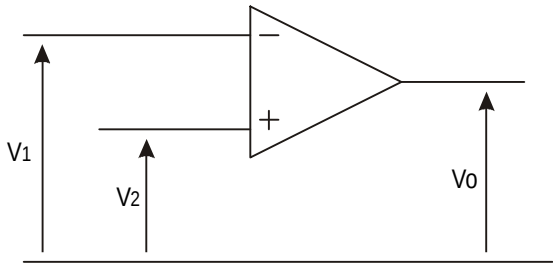
When you have completed this unit you should be able to:

- correctly identify common applications of analogue control
- represent analogue control systems using control diagrams and circuit diagrams
- select the appropriate configuration of op-amp in the design of analogue closed loop control systems
- select appropriate output drive subsystems for control applications

Before you start this unit you should have a basic understanding of:

- System diagrams
- Open and closed loop control
- Operational amplifiers

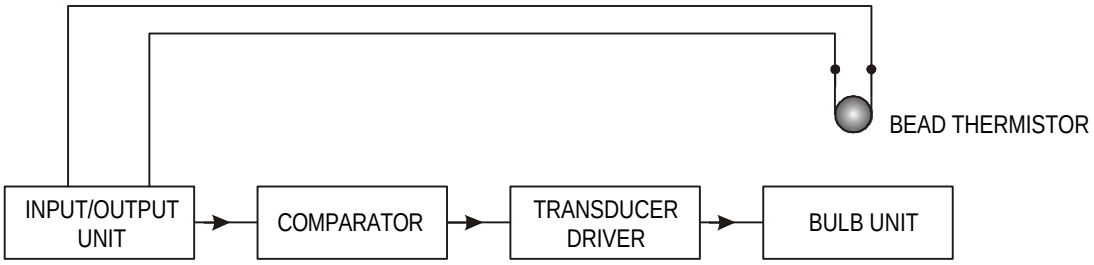
Analogue Closed Loop Control Systems



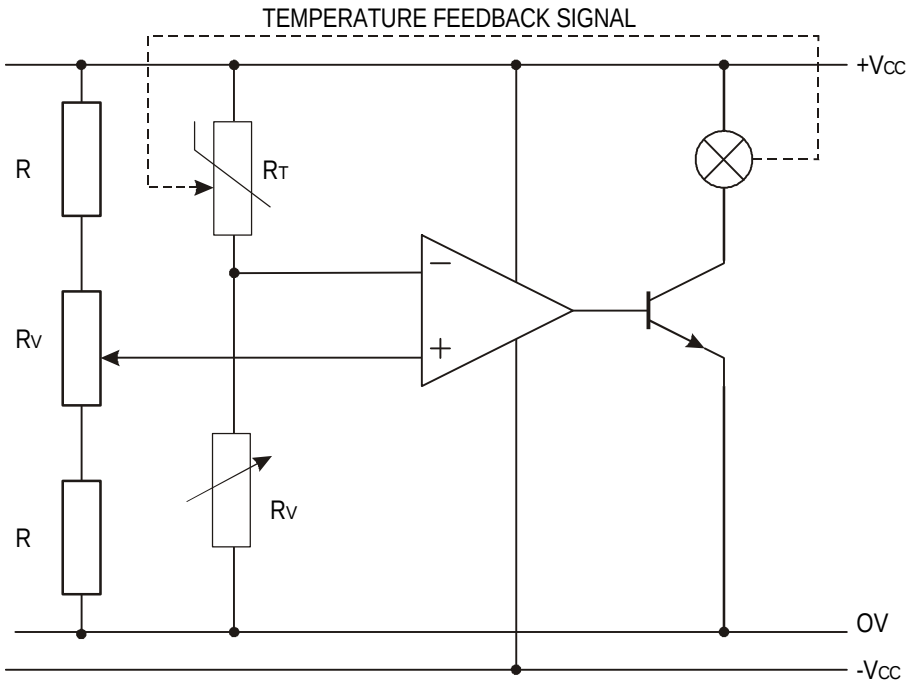
Many control systems involve processing analogue signals such as heat, light and movement. Therefore analogue closed loop control systems require analogue processing devices such as the **operational amplifier (op-amp)**.

One of the most common control applications involves using the op-amp as a **comparator**. In it's simplest form a comparator just compares two voltage signals, V1 and V2. If V1 is higher than V2 the output, Vo, is 'low', if V1 is lower than V2 the output is 'high'

Assignment 1
The block diagram below simulates an automatic heating system. When the bulb heats up the temperature rise is detected by the bead thermistor, which sends a feedback signal back to the comparator.



A circuit diagram of the system is shown below.



The variable resistor, R_{V1} (connecting to the non-inverting input of the op-amp) is used to set the **reference level** (or threshold). This sets the desired temperature of the bulb.

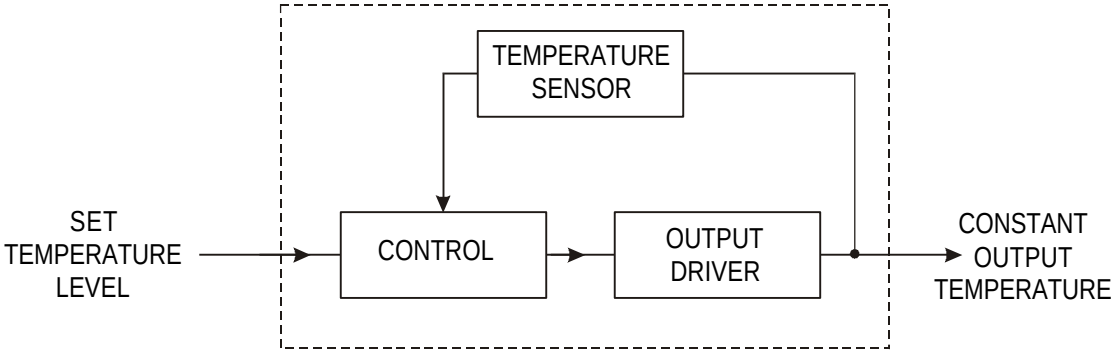
Build the circuit shown, using discrete components or a modular electronics system. Calibrate the system to the following performance criteria:

- When the bulb heats above the reference level the thermistor should sense the temperature and send a signal to the comparator which will switch the bulb off.
- When the bulb cools below the reference level the bulb should switch on again.
- The system should operate continuously.

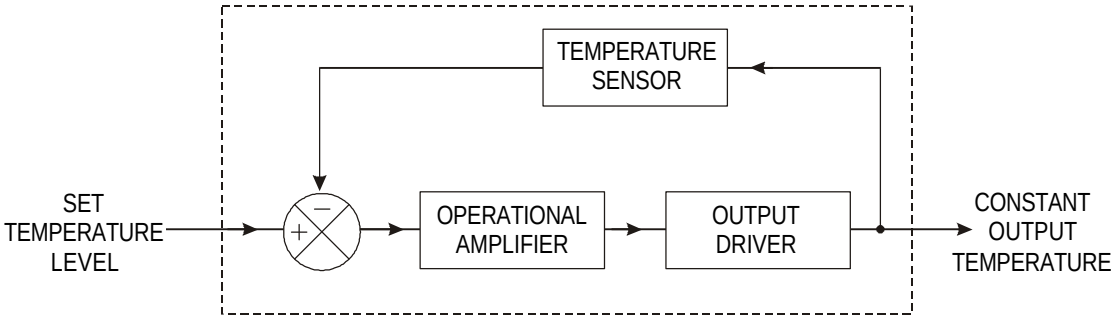
a) Describe how the circuit operates.
b) Explain clearly how you calibrated the system.

Control Diagrams

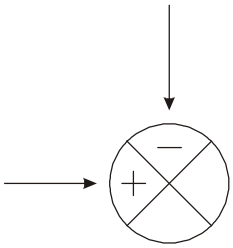
The heating control system described in assignment 1 can be represented by the following **systems diagram**.



However when an op-amp is used in a control system it is usual to draw a **control diagram** instead.



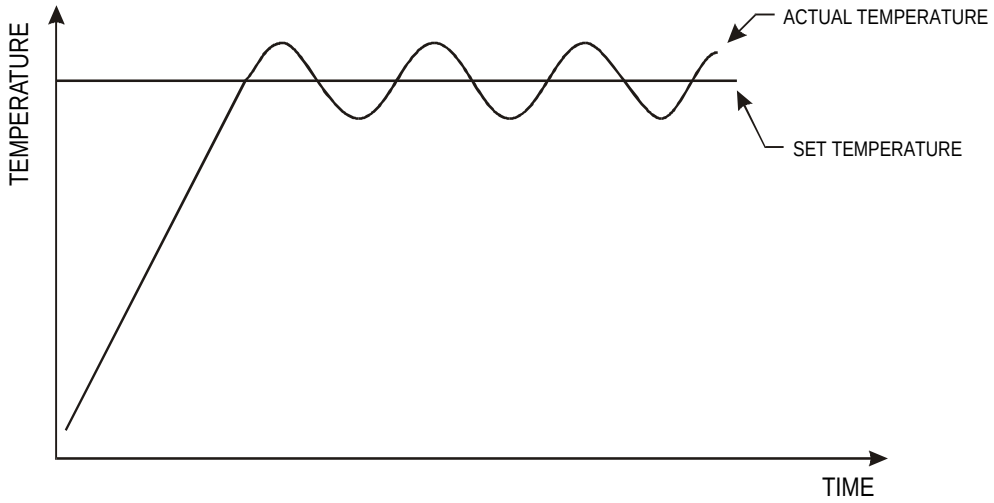
The control diagram breaks the generalised 'control' subsystem of the systems diagram into more specific blocks.



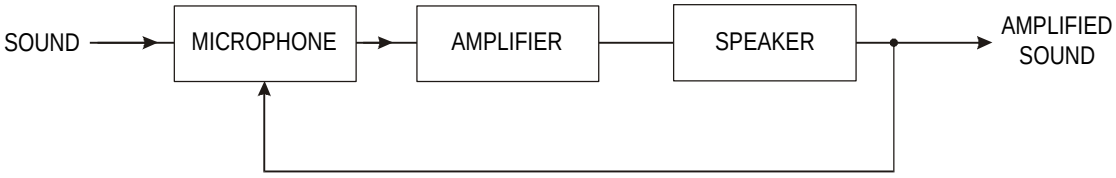
The **error-detection symbol** is also used to indicate that the control involves two signals. The feedback signal is connected to the negative symbol to indicate the use of **negative feedback**.

Negative and Positive Feedback

The purpose of closed loop control is to ensure that the output is maintained, as closely as possible, to the desired output level. In the case of a central heating system, a graph of the temperature in a room might appear as in the graph below.

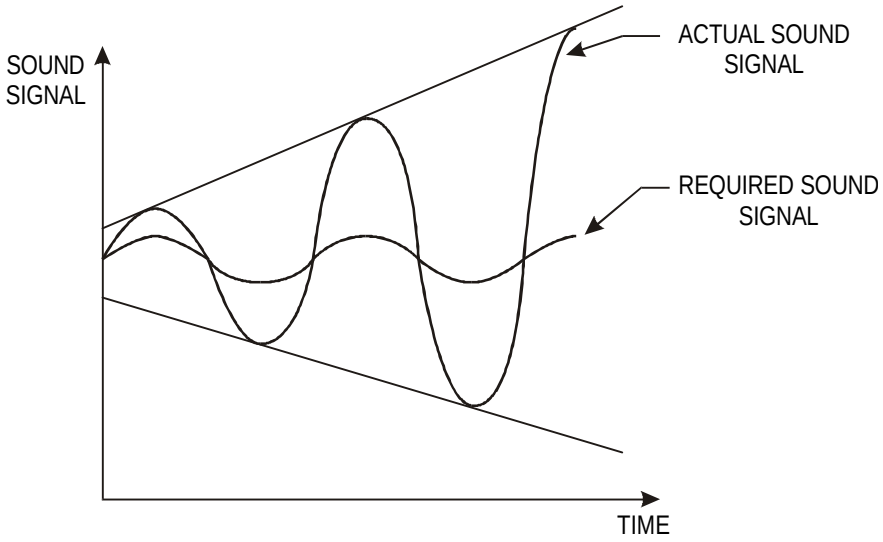


As can be seen from the graph, the control system is constantly trying to pull the temperature of the room back towards the set temperature level by reducing the error. This type of control uses **negative feedback** to reduce the error.



AMPLIFIED SOUND FEEDBACK TO SPEAKER

The opposite effect can be created by reinforcing the error, as can sometimes happen with public address systems when the microphone is held too close to the speakers. A sound is picked up by the microphone, amplified, and then output through the speaker. The amplified sound is then picked up, re-amplified and so on. The net result is a high pitch sound, which can be represented by the graph below.



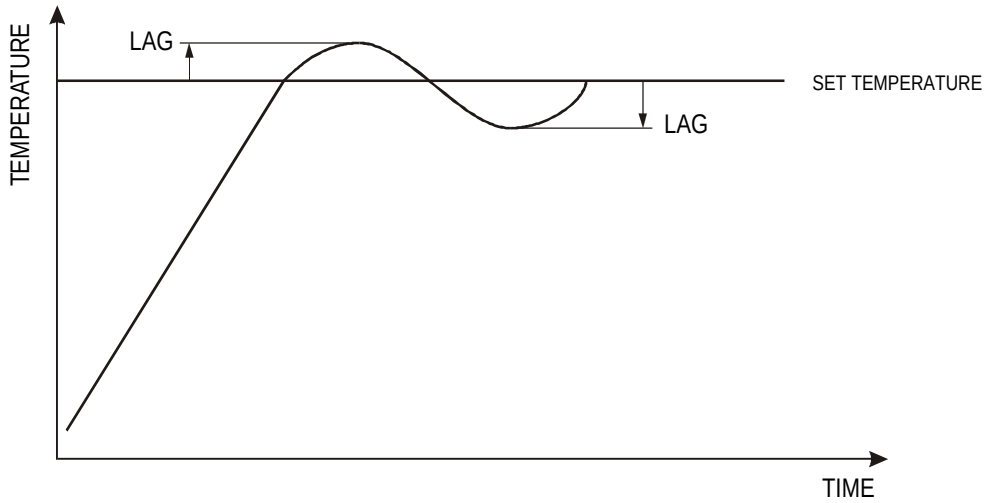
This is an example of **positive feedback**. Although positive feedback does have some useful applications, negative feedback is far more widely used in control systems.

Assignment 2

Explain the following terms when applied to control systems:
open loop, closed loop, negative feedback, positive feedback, error detector

Proportional Closed Loop Control Systems

The main problem with comparator based control systems is that there is a time 'lag' built into their operation. This means that the desired output state is never actually reached.

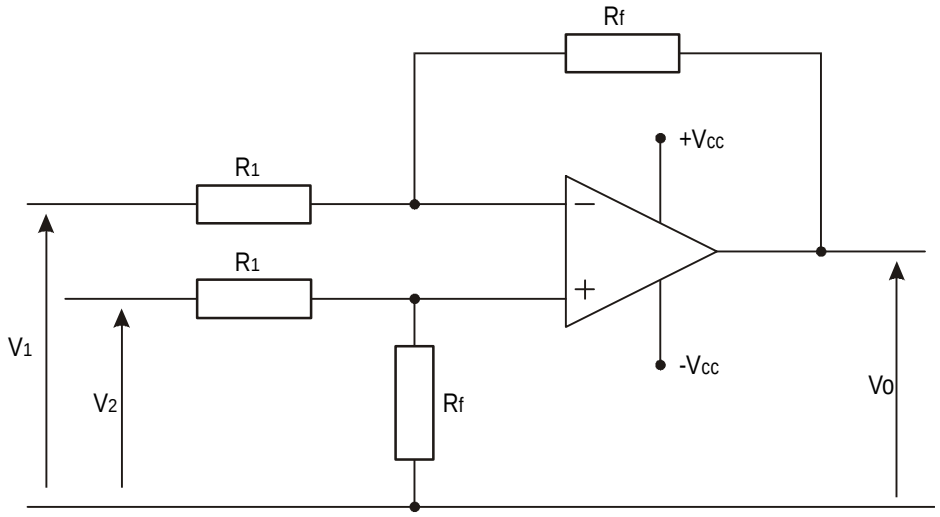


The graph of the temperature control system built in assignment 1 clearly shows the effect of lag.

When the reference level ("set temperature") is reached the heating system will continue to heat the room for a short period until the heating element cools down. This raises the temperature in the room above the set or desired temperature level. When the temperature in the room cools down to the reference level the heating system will be switched on. Whilst the heating element is reaching its operating temperature the room continues to cool below the reference level.

Whilst in heating systems the presence of lag does not present a problem, and in fact presents some advantages, in many other applications the inaccuracy of this form of control is not acceptable.

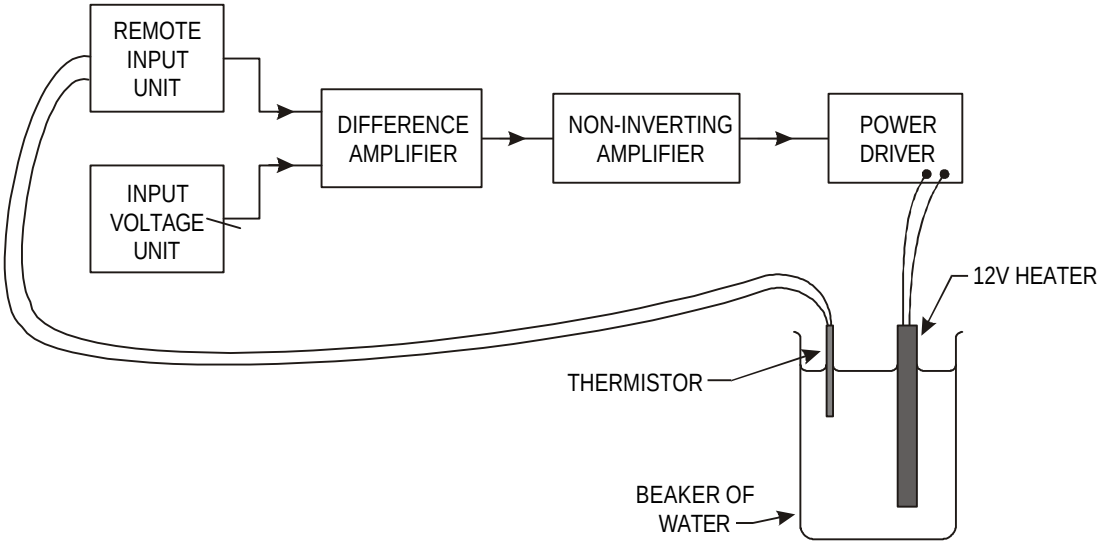
Proportional closed loop control systems are applied as a means of proving a more accurate form of analogue closed loop control. The operation of proportional control is based around the op-amp configured as a **difference amplifier**.



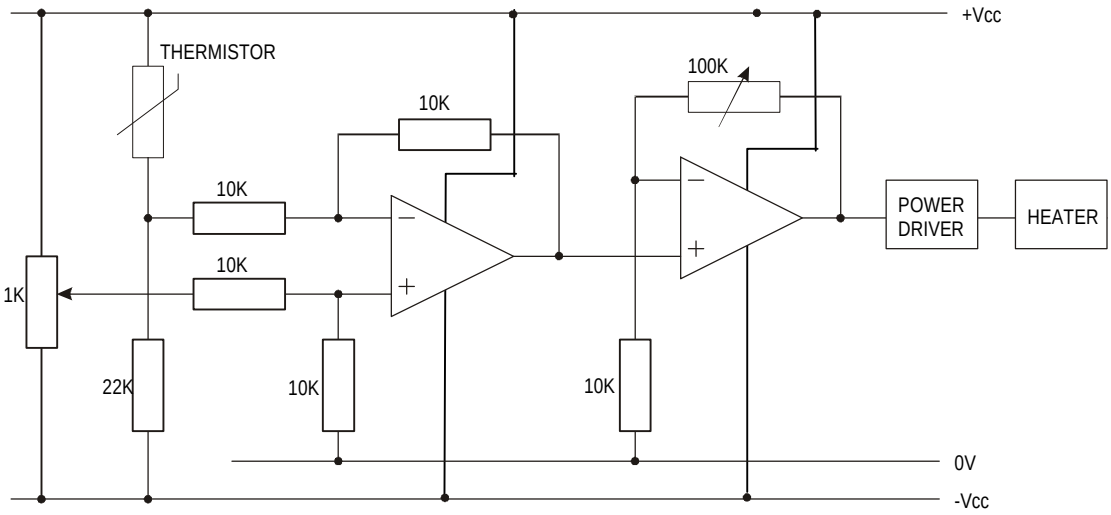
The **difference amplifier** amplifies the difference between the **reference level** and the **feedback signal**. Therefore if there is a large error there will be a larger output signal than when there is a small error. This help prevents the 'overshoot' and 'time lags' seen in the comparator based systems.

Assignment 3

The block diagram below simulates the automatic water heating system within a washing machine.



A circuit diagram of the system is shown below.

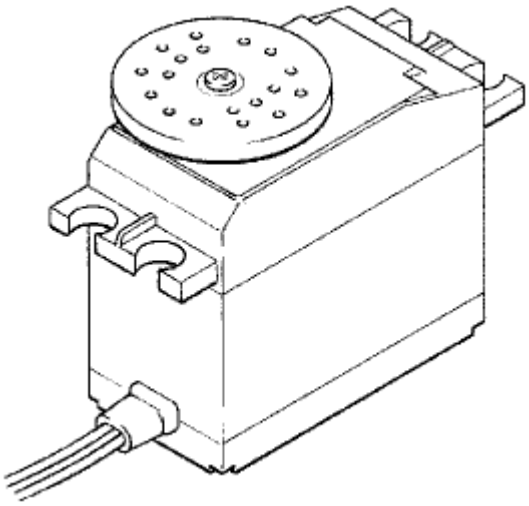


Build the circuit shown, using discrete components or a modular electronics system.

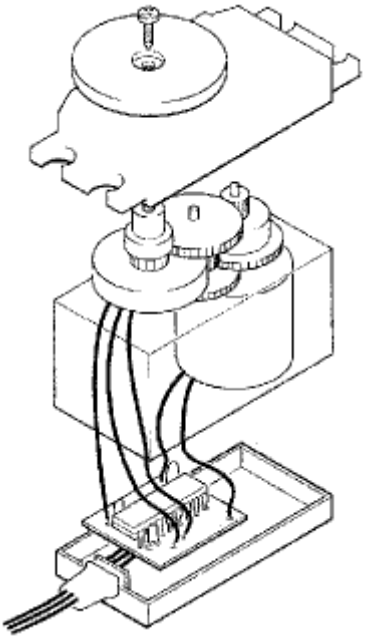
Observe the response of the system when the reference level (desired temperature) is altered (by adjusting the variable resistor). Use a digital thermometer to measure the temperature of the fluid, and a voltmeter to measure the voltage across the heater.

a) Describe how the circuit operates. What is the purpose of the non-inverting amplifier after the difference amplifier?
b) Explain clearly how the system responds when the desired temperature level is altered.

Positional Control



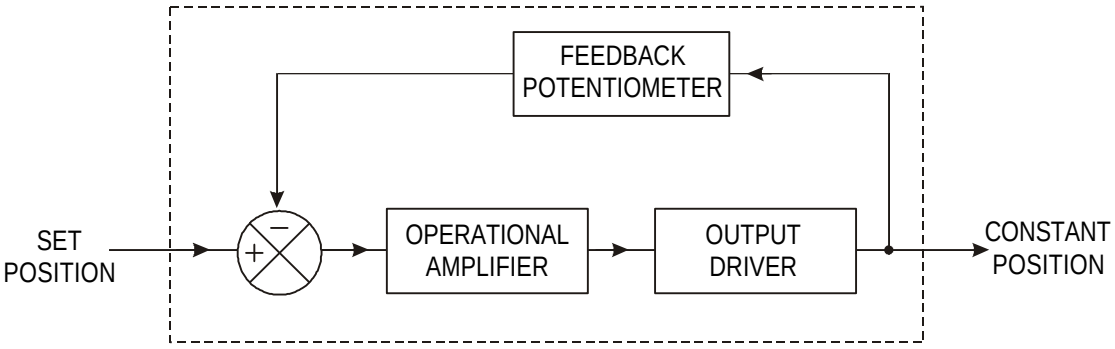
A good example of positional control is the model servo motor commonly found in radio-control cars and aeroplanes.



The servo contains a motor, gearbox, feedback potentiometer and electronic control circuit on a small printed circuit board.

The feedback potentiometer is directly connected to the output shaft. Therefore, when the motor spins, the output shaft is turned slowly by the gearbox, which in turn moves the potentiometer. Naturally movement of the output shaft is restricted to 180º as the potentiometer cannot spin continuously.

The control diagram for the servo is shown below.

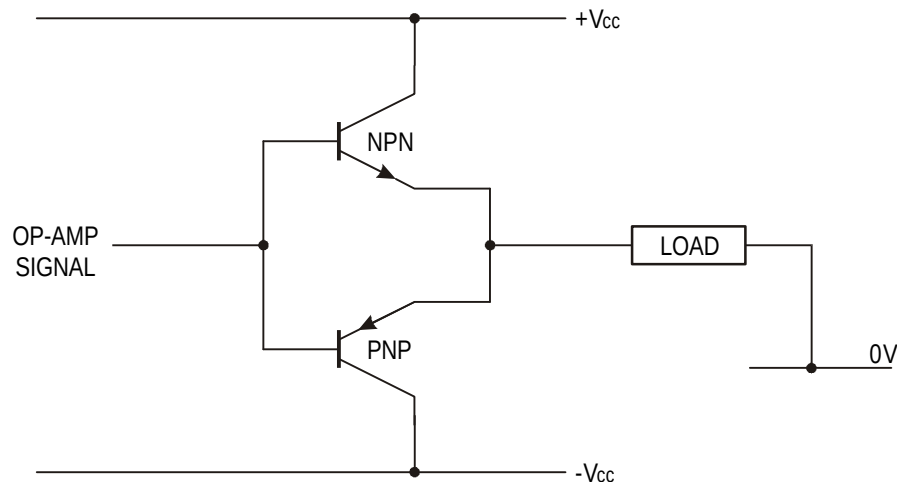


Note how the feedback signal from the feedback potentiometer is fed into the negative symbol of the control diagram, indicating negative feedback.

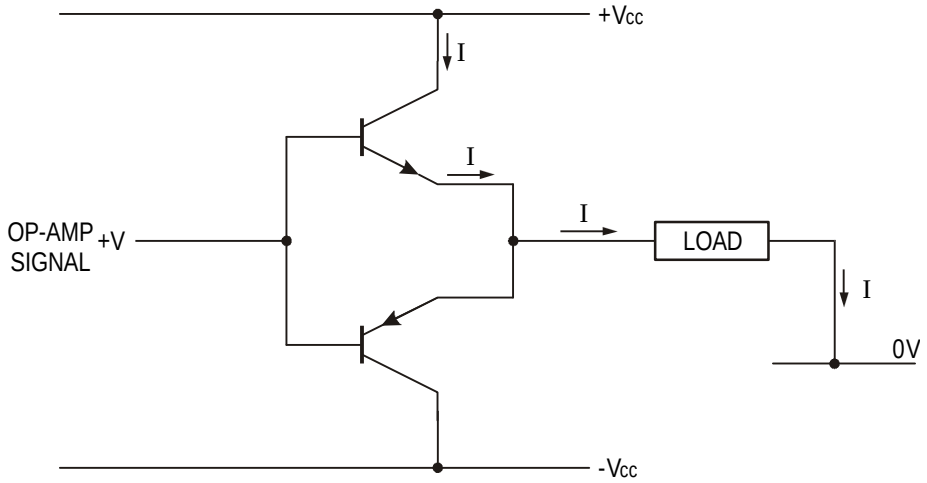
Push-Pull Follower Analogue Driver

An important element in producing positional control, such as that required in the servo, is a dual rail **push-pull follower** analogue driver circuit, that allows the motor to spin in both directions. The op-amp cannot source sufficient current to drive most output components, and so the dual rail push-pull follower is required to drive the output components.

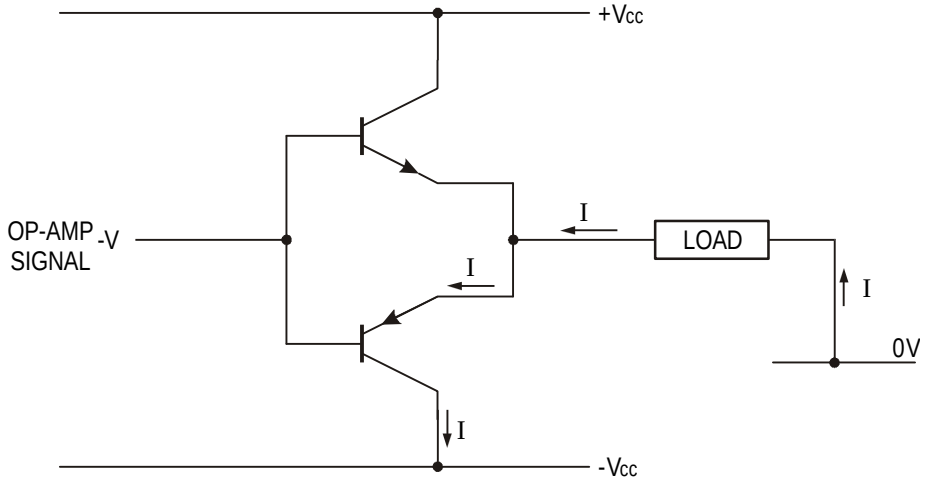
The dual rail push-pull follower is based around a two transistor circuit as shown below.



When the signal from the op-amp is positive the NPN transistor will switch on, and current will flow through the load from the positive supply rail, +Vcc, to the ground rail, 0V.

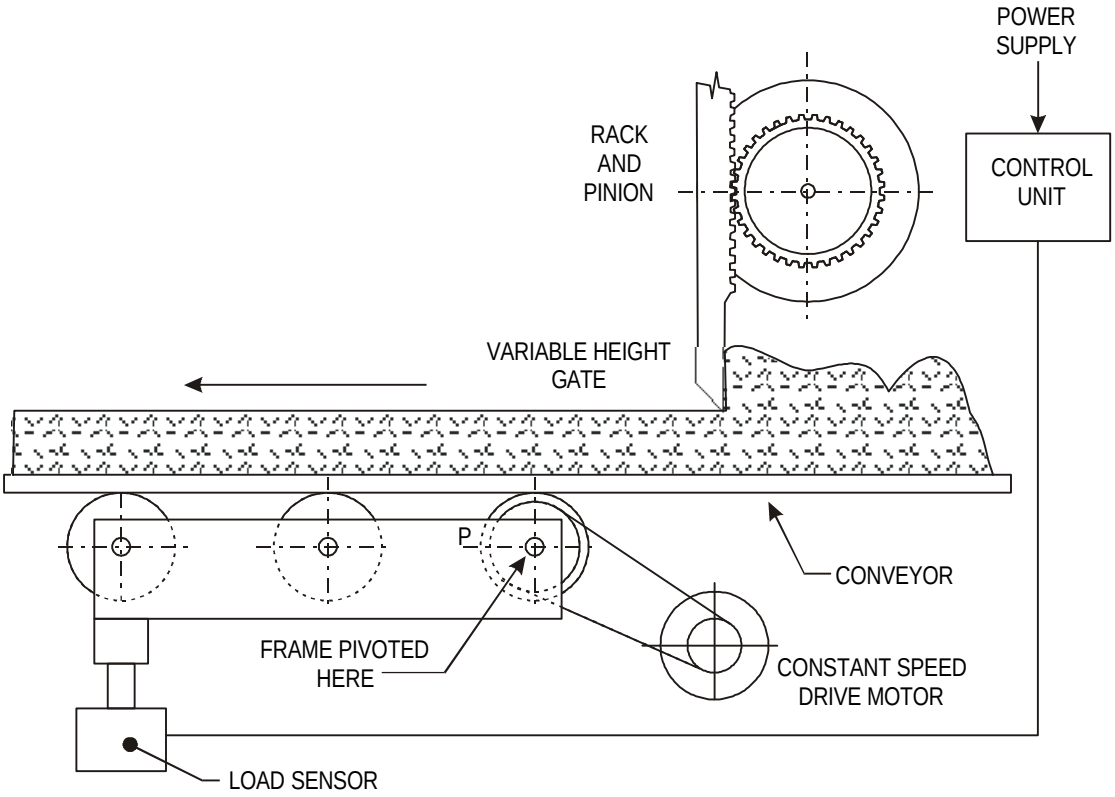


When the signal from the op-amp is negative the PNP transistor will switch on and current will flow through the load from the ground rail, 0V, to the negative supply rail -Vcc.



Assignment 4

A proportional control system is used to regulate the flow rate of coal onto a conveyer belt. The system should sense the weight of coal on the conveyer belt and automatically adjust the gate height to ensure that a constant flow of coal is supplied.



a) Draw a systems diagram of the flow rate control system.

b) Explain the term 'proportional control'.

c i) Name the configuration of op-amp used in proportional control systems.

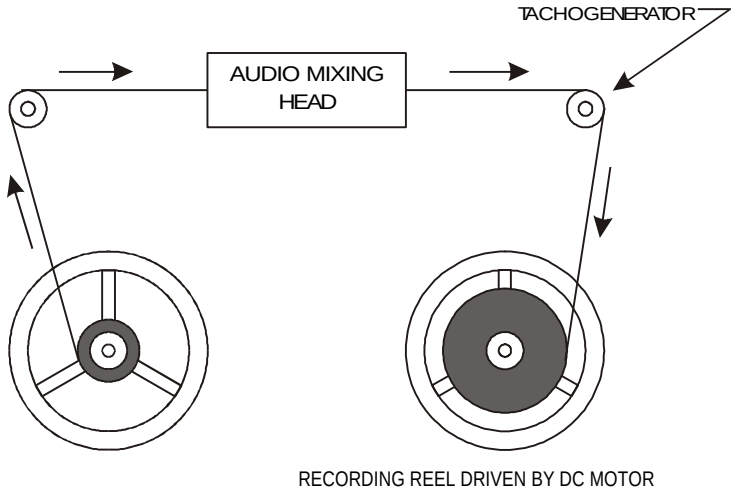
c ii) Draw a circuit diagram of the op-amp.

d i) Name a suitable output driver circuit which could be used with the control system.

d ii) Draw a circuit diagram of the driver circuit.

Assignment 5

The figure shows the layout of an audio mixing desk.



The tape is fed through the audio mixing head by being pulled on to a recording reel driven by a dc motor. As the tape builds up on the recording reel, the tape speed through the audio mixing head will increase.

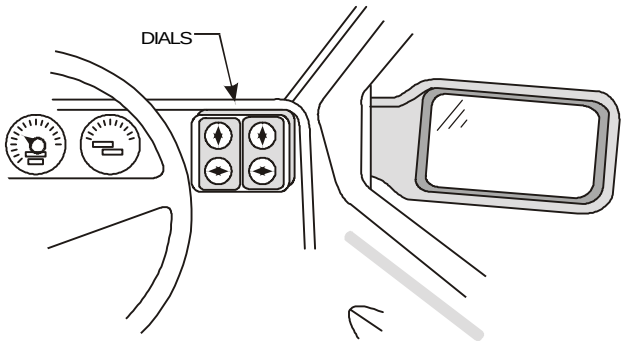
To prevent this from happening, the dc motor is fitted to a closed loop control system.

A tachogenerator connected to a pulley senses the tape feed rate and sends an error signal to the control system.

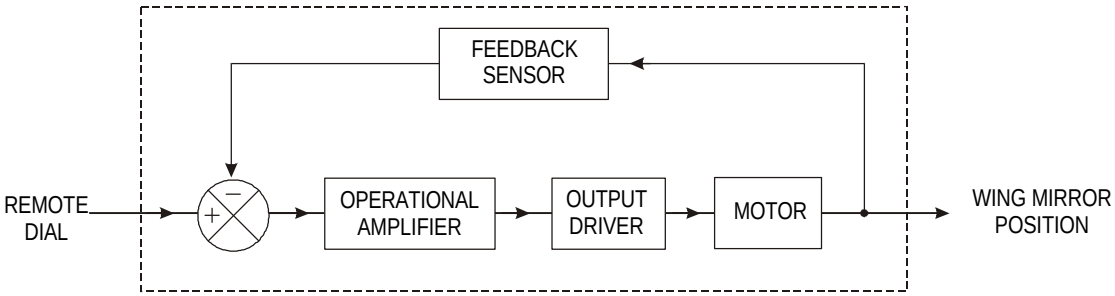
a) Name the type of closed loop control used in this application.
b) Name the amplifier used in this type of closed loop control.
c) Draw a control diagram of the system.
d) Draw a circuit diagram of the control system and explain the function of each part of the circuit.

Assignment 6

The figure illustrates a system for controlling the wing mirrors on a car by adjusting remote dials on the dash.



A control diagram of the system for rotational movement in the X-axis (one mirror) is shown in the figure below. Similar systems are used for the Y-axis and for the other mirror.



a) With reference to the control diagram, explain clearly how the system operates.

b) Name the type of control used in this system.

c) Name the configuration of op-amp required.

d) State two reasons why the op-amp cannot be used to drive the motor directly.

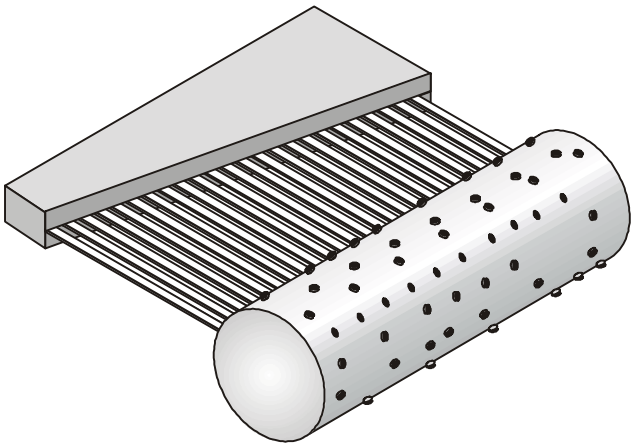
e) Name a suitable output driver which could be used with this system.

f) Draw a circuit diagram of the output driver.

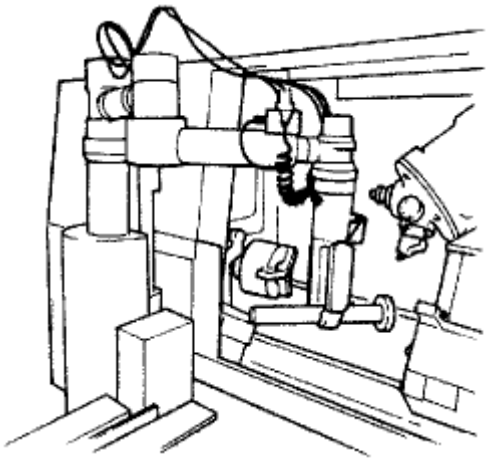
Sequential Control Systems

Sequential control is used to control systems whose outputs are required to follow a fixed cycle of events (i.e. a sequence).

An interesting example of sequential control was used in music boxes. Actuators on the drum are arranged to produce notes, as the rum rotates the notes are played in the correct sequence to produce a tune.



A more modern example of sequential control is a robot arm used to weld cars together on a large production line.



Most modern sequential control systems use discrete logic circuits, or a programmable controller, such as the Basic Stamp system (covered within Outcome 3).

TECHNOLOGICAL STUDIES

HIGHER

SYSTEMS AND CONTROL

STUDENTS' NOTES

OUTCOME 3

Outcome 3 - Microcontroller Controlled Mechatronic Systems

Section 1 - Introduction

The purpose of this section is to introduce the microcontroller and it's architecture.

When you have completed this section you should be able to:

- describe the operation and architecture of microcontrollers
- understand the terms ALU, RAM, ROM, EEPROM, bus
- understand how the Basic Stamp system operates

Before you start this section you should have a basic understanding of:

- No previous knowledge required.

To complete the exercises in this section you require:

- No equipment required.

Section 2 - Number Systems

The purpose of this section is to introduce the main number systems used within programmable systems for the processing of information.

When you have completed this section you should be able to:

- Use the following terms correctly: decimal, binary, hexadecimal.
- Describe contexts when it is appropriate to use the three different number systems.
- Convert between decimal, binary and hexadecimal number systems.

Before you start this section you should have a basic understanding of:

- No previous knowledge required.

To complete the exercises in this section you require:

- No equipment required.

Section 3 - Simple Control Routines

The purpose of this section is to introduce the PBASIC commands used in developing control program listings.

When you have completed this section you should be able to:
- understand the use of flowcharts
- develop flowcharts from a brief
- understand the most common PBASIC commands
- write PBASIC programs that involve loops, if statements and sub-procedures

Before you start this section you should have a basic understanding of:
- The architecture of a microcontroller.
- The decimal, binary and hexadecimal number systems.
- Flowcharts

To complete the exercises in this section you require:
- Stamp Controller
- Output Drivers Module (or MFA Movement Module)
- Buggy (with microswitch sensors)
- MFA Sensor Module and Sensors
- Washing Machine Model

Section 4 - Mechatronic System Interfacing Circuits

The purpose of this section is to understand the need for interfacing input and output devices when building mechatronic systems that are controlled by a microcontroller.

When you have completed this section you should be able to:
- explain why interfacing circuits are required within mechatronic systems
- select appropriate interfacing techniques for common output devices
- understand the different operation of common switch types
- understand how unipolar stepper motors are controlled
- understand the need for D-A conversion
- understand how push-pull drivers are used to control dc motors
- understand pulse-width modulated control of dc motors
- understand the term 'soft-start' when applied to dc motors
- write PBASIC programs to control stepper motors
- write PBASIC programs to control the speed and direction of dc motors

Before you start this section you should have a basic understanding of:
- The decimal, binary and hexadecimal number systems.

To complete the exercises in this section you require:
- Stamp Controller
- Output Driver Module
- MFA Sensor Module and Sensors
- DC Motor
- Stepper Motor
- MFA Stepper Motor Driver
- MFA Power D to A Converter
- Buggy (with microswitch sensors)
- Lock Model
- Light Seeker Model
- Conveyer Belt / Lift Model

Section 1 - Introduction

The purpose of this section is to introduce the microcontroller and it's architecture.

When you have completed this section you should be able to:

- describe the operation and architecture of microcontrollers
- understand the terms ALU, RAM, ROM, EEPROM, bus
- understand how the Basic Stamp system operates

Before you start this section you should have a basic understanding of:

- No previous knowledge required.

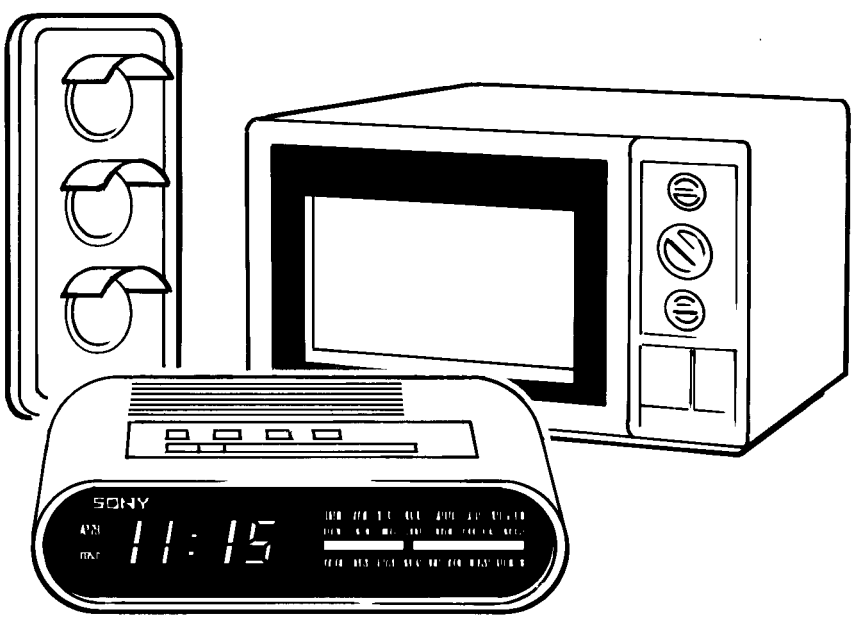
To complete the exercises in this section you require:

- No equipment required.

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.

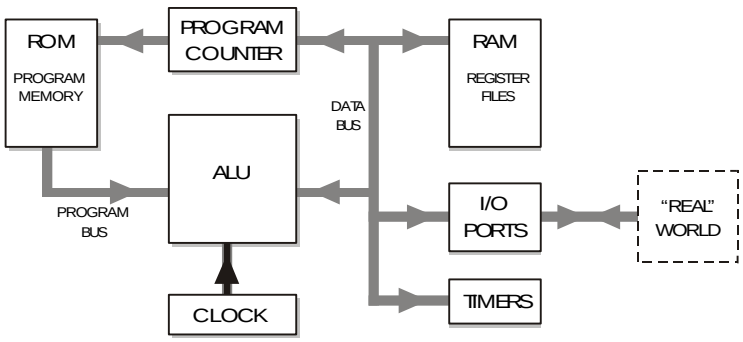
One microcontroller can often replace a number of separate parts, or even a complete electronic circuit. Some of the advantages of using microcontrollers in a product design are:

- 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.

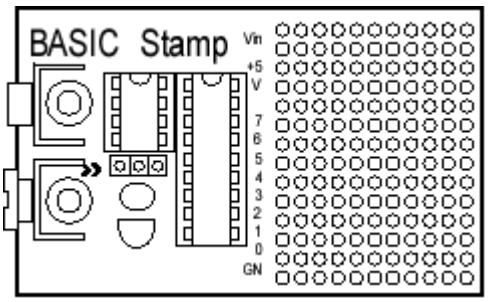
Timers

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.

Understanding the Basic Stamp System

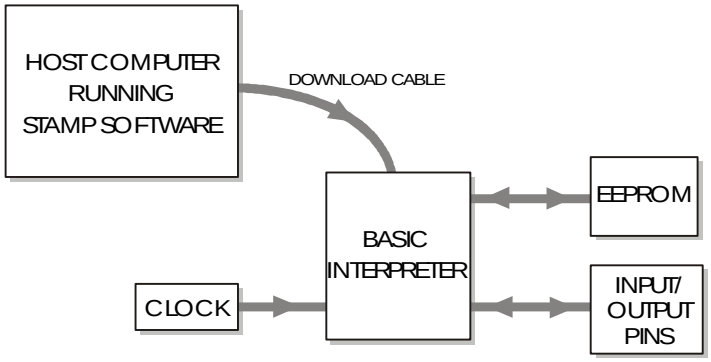
In industry microcontroller programs are normally developed using the 'assembler' or 'C' programming languages. Unfortunately these languages are not particularly easy for the beginner to understand, and it can take a great deal of time and study before a programmer is skilled enough to construct a complex program.

For this reason it is easier for the beginner to program with 'user friendly' languages such as **BASIC** (Beginners' All Purpose Symbolic Instruction Code). This language is specifically designed to be 'easily understood' and so primarily uses standard 'English language' words as instructions. However, before the microcontroller can understand the BASIC instructions, these instructions must be processed by an 'interpreter' into machine code assembler. The extra processing time involved in this conversion results in a BASIC program running slower than an equivalent machine code assembler program. However, as microcontrollers can process over one million assembler code instructions a second, the extra processing time required by the BASIC program is negligible in most cases!



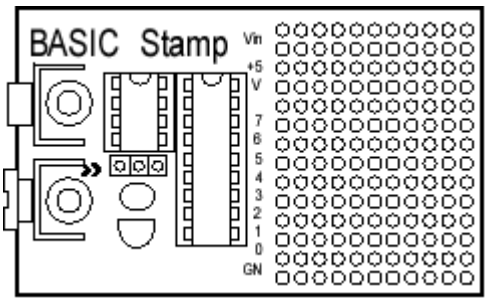
The 'Basic Stamp' system was developed in the early 1990s (by Parallax, Inc., USA) to enable design engineers to quickly prototype systems using microcontrollers by programming in a modified BASIC language (called PBASIC - short for Parallax BASIC) rather than assembler or C. The Basic Stamp system is an ideal compromise for rapid prototyping - all the power and versatility of a microcontroller combined with a simple programming language.

The 'Basic Stamp' system consists of three main components - the 'Basic Stamp' software, a download cable and the Stamp module itself. The Stamp module contains two main integrated circuits - a PIC microcontroller pre-programmed with the PBASIC interpreter, and an EEPROM memory chip.

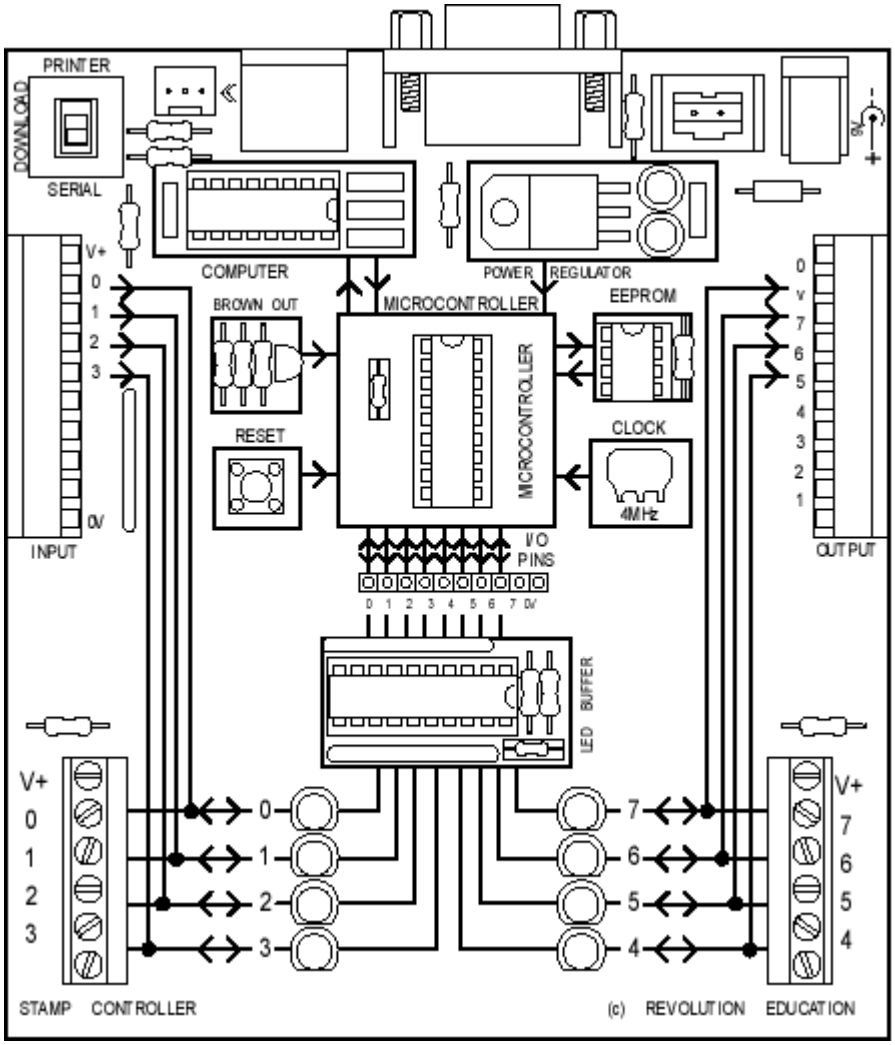


The program instructions are written on a host computer in the simple PBASIC language. The download cable is then connected from the computer to the module, and the program is downloaded into the EEPROM memory chip. The download cable is then removed, and the program (now stored in the memory chip) is carried out sequentially by the microcontroller 'interpreter' chip.

Therefore the primary difference between programming standard microcontrollers and programming the Basic Stamp is that, when you 'download' a program to the Basic Stamp, you are actually programming the external EEPROM, rather than the microcontroller ROM. However when you build electronic systems you are still connecting directly to the 'microcontroller' input/output pins, and so the electronic connections are identical to those made to standard microcontrollers.



The original Parallax Basic Stamp module consisted of a small printed circuit board (PCB) with battery clip and 'prototyping areas'. Although suitable for prototyping work, this module is not so appropriate for classroom exercises, and so the Stamp Controller has been developed for educational use. This uses identical 'chips' to the original module, but is configured on a larger PCB with all the necessary connectors etc.



The memory chip used on the Stamp Controller to store the program is the EEPROM type. This type of memory can be reprogrammed when desired, but also retains the program when the power supply is removed. This means the Stamp Controller will start to run the program currently in memory whenever the power supply is connected.

Summary - Programming Procedure.

1. Write the program on a host computer using the Stamp software.
2. Connect the download cable from the computer to the Stamp Controller.
3. Connect the power supply to the Stamp Controller.
4. Use the Stamp software to download the program. The download cable can be removed (if desired) after the download.

The program will start running on the Stamp Controller automatically. However the program can also be restarted at any time by pressing the reset switch.

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

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

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

1.5) Describe the similarities and differences between programming, and constructing interfacing circuits for, a 'true' microcontroller and the 'Basic Stamp' system.

Section 2 - Number Systems

The purpose of this section is to introduce the main number systems used within programmable systems for the processing of information.

When you have completed this section you should be able to:

- Use the following terms correctly: decimal, binary, hexadecimal.
- Describe contexts when it is appropriate to use the three different number systems.
- Convert between decimal, binary and hexadecimal number systems.

Before you start this section you should have a basic understanding of:

- No previous knowledge required.

To complete the exercises in this section you require:

- No equipment required.

Number Systems

A microcontroller operates by performing a large number of mathematical calculations in a very short space of time. This is possible because each piece of information can be expressed as a series of electronic signals. These signals are recognised as being in one of two states, which are described as **high** and **low** (or "on" and "off").

The counting system used in everyday activities is the **decimal system**. This number system uses the ten familiar digits 0 to 9 to express the magnitude of the number.

However, as the microcontroller only recognises the two electronic states high and low, it uses the **binary number system**. This number system uses just two digits 0 and 1. An electrical signal which is low is represented by 'logic 0', and a signal which is high is represented by a 'logic 1'.

The first sixteen numbers in the decimal and binary systems are shown in the table below:

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

A single binary digit is referred to a **bit** (binary digit). Different systems carry out calculations using different quantities of bits, and so systems are often referred to as 8-bit, 16-bit or 32-bit systems. The most common microcontrollers use the 8-bit system, although 32-bit microcontrollers are also now becoming more readily available.

Bits and Bytes

Eight bits grouped together are described as a **byte**. The decimal value of a byte is calculated by adding together the corresponding decimal value of each of the individual bits. The eight bits in a byte are labelled bits 0 to 7, from right to left. The right most bit is called the **Least Significant Bit (LSB)** and the left most bit is called the **Most Significant Bit (MSB)**. The decimal value of each bit is given in the table below:

bit number	7	6	5	4	3	2	1	0
decimal value	128	64	32	16	8	4	2	1

The binary number 10010111 when converted into decimal would be:

1	x	128	=	128
0	x	64	=	0
0	x	32	=	0
1	x	16	=	16
0	x	8	=	0
1	x	4	=	4
1	x	2	=	2
1	x	1	=	1
			Total:	**151**

Note that when writing binary numbers it is quite common to write **all** 8 bits, even if the first bits are equal to zero (unlike the decimal system, where leading zeros are not normally written).

Assignments:
Convert each of the following binary numbers into decimal:
2.1) 11110000
2.2) 11000011
2.3) 01010101
2.4) 10101010

Converting Decimal to Binary

To convert any decimal number into binary repeatedly divide the decimal number by two and record the remainder after each division. The decimal number 29 is used as an example.

29	÷	2	=	14	rem 1
14	÷	2	=	7	rem 0
7	÷	2	=	3	rem 1
3	÷	2	=	1	rem 1
1	÷	2	=	0	rem 1

Therefore the decimal number 29 equals the binary number 00011101

Assignments:
Convert each of these decimal numbers into binary:
2.5) 17
2.6) 23
2.7) 11
2.8) 38
2.9) 33

Hexadecimal Number System

Writing binary numbers in groups of eight bits is quite time consuming, although on some occasions binary numbers are very useful to clearly illustrate the condition of each bit in a byte.

However a more user-friendly system of writing numbers is the **hexadecimal system**. The hexadecimal system uses 16 different digits - 0 to 9 and A to F. The 'digits' A to F correspond to the decimal numbers 10 to 15.

Decimal	Hexadecimal
10	A
11	B
12	C
13	D
14	E
15	F

This system allows 4-bit binary numbers to be converted into a single hexadecimal digit (a group of four bits is called a nibble, so there are two nibbles in a byte).

Decimal	**Binary**	**Hexadecimal**
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Notation

When using a number of different counting systems it is important to distinguish which counting system you are using. For instance the number '10' means three different values in the three different counting systems!

Therefore the following notations are used within PBASIC programs:

Decimal values are written as usual:	10	(= 10 in decimal)
Binary values are preceded by a % symbol:	%10	(= 2 in decimal)
Hexadecimal values are preceded by an & symbol:	&10	(= 16 in decimal)

Converting Binary to Hexadecimal

1. Divide the binary number into groups of nibbles (four bits)
2. Convert the nibbles into decimal
3. Convert the decimal nibbles into hexadecimal (i.e. convert any decimal values greater than 9 into the hexadecimal letter 'digits')

Example:

Convert %01101010 into hexadecimal

1. Divide into nibbles 0110-1010
2. Convert into decimal 6-10
3. Convert into hexadecimal 6-A

Therefore %01101010 = &6A

Assignments:
Convert the following binary numbers to hexadecimal:
2.10) %11011101
2.11) %11111001
2.12) %01010101
2.13) %10101010
2.14) %11001010

Converting Hexadecimal to Binary

Changing a hexadecimal number to its binary equivalent is the reverse of the Binary to Hexadecimal procedure.

Example:

Convert &AF into binary.

1. Divide into separate digits	A-F
2. Convert to decimal	10-15
3. Convert to binary	1010-1111
4. Join the two four bit numbers	10101111

Therefore &AF = %10101111

Assignments:
Convert the following hexadecimal numbers to binary:
2.15) &4E
2.16) &0D
2.17) &FD
2.18) &5A
2.19) &B2

Section 3 - Simple Control Routines

The purpose of this section is to introduce the PBASIC commands used in developing control program listings.

When you have completed this section you should be able to:
- understand the use of flowcharts
- develop flowcharts from a brief
- understand the most common PBASIC commands
- write PBASIC programs that involve loops, if statements and sub-procedures

Before you start this section you should have a basic understanding of:
- The architecture of a microcontroller.
- The decimal, binary and hexadecimal number systems.
- Flowcharts

To complete the exercises in this section you require:
- Stamp Controller
- Output Drivers Module (or MFA Movement Module)
- Buggy (with microswitch sensors)
- MFA Sensor Module and Sensors
- Serial LCD Module
- Washing Machine Model

Port Addressing and the Data Direction Register

Microcontrollers communicate with the outside world by input/output pins which are grouped together in **'ports'**. The Stamp Controller has one input/output port, which contains eight input/output pins.

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 3 'high' individually the command would be:

```
high 3
```

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

```
low 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 pins = 255
```

To switch all pins 'low' the command would be

```
let pins = 0
```

To switch pins 0-2 'high' and pins 3-7 'low' the command would be

```
let pins = %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.

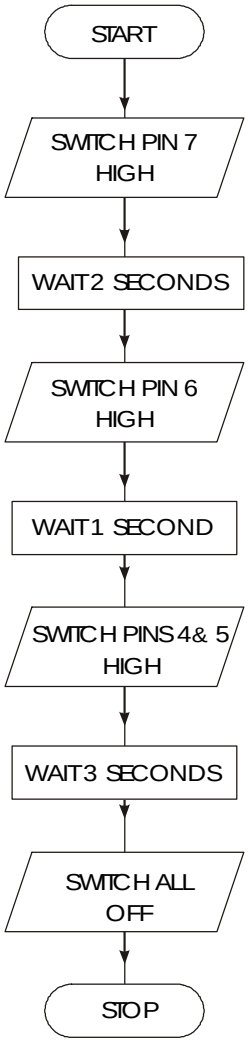
For example,

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

Every PBASIC program listing should always begin with a 'let dirs =' statement to correctly setup the DDR. By default all pins are set to inputs when the Stamp Controller is reset.

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 dirs = %11110000          ' set pins 0-3 inputs, 4-7 outputs

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

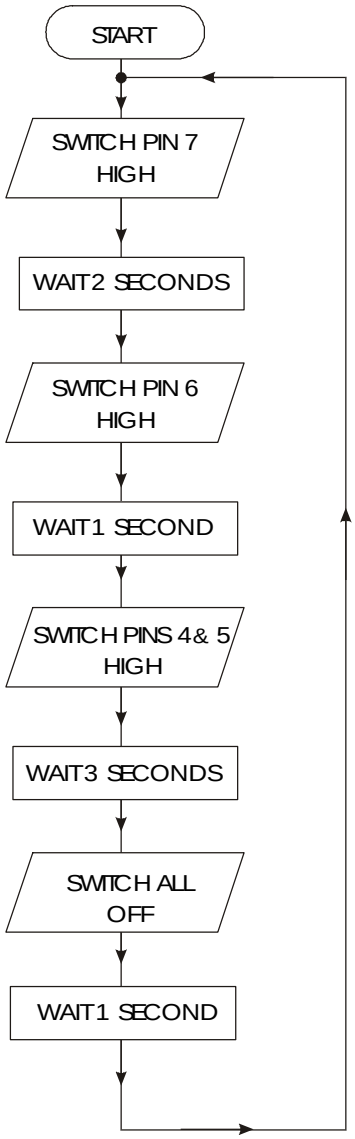
Activity 3.1
Key in the program listed above, and then download it to the Stamp Controller.

The LED indicator on pin 7 should light first, followed by the indicators on pin 6, and then indicators 4 and 5. To re-run the program press the reset switch on the Stamp Controller.

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 dirs = %11110000    ' set pins 4-7 as outputs

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

Activity 3.2
Key in, download and run the program listed above.

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 3.5

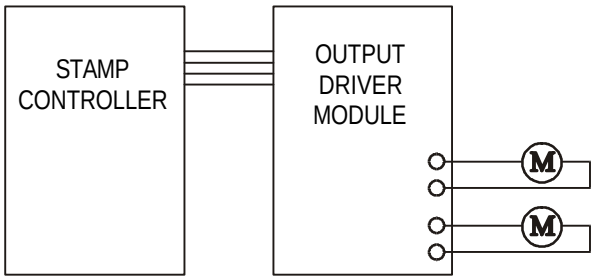
A set of temporary traffic lights are required for a system of road-works.

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

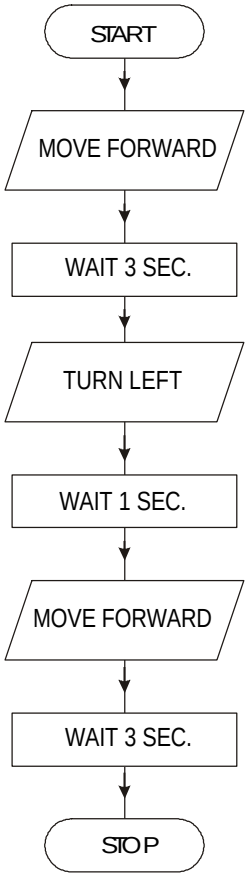
Draw a flowchart for the lights sequence shown by one set of the traffic lights. Use the times shown in the table for each stage.

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

Assignment 3.6

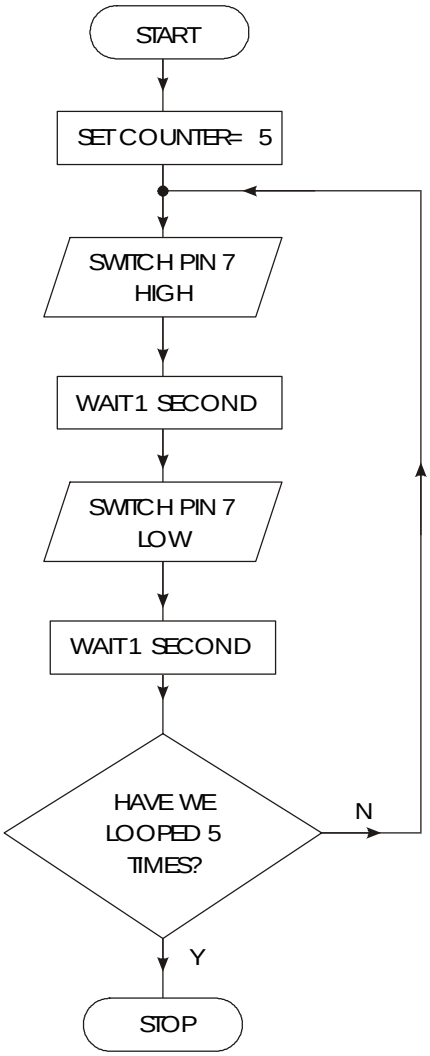


Connect a buggy to the output driver module (or MFA movement module) as shown in the diagram.



Write a high level program in PBASIC to control the movement of the buggy as shown by the flowchart above.

For...Next Loops



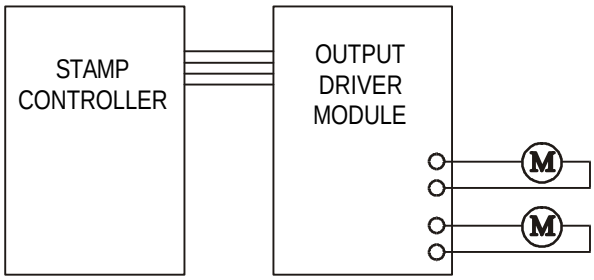
A **for...next** loop is used when you wish to repeat a section of code a number of times. The number of times the program runs for is set by a variable. A variable is a number that is stored in the RAM memory of the Stamp Controller. There are 14 memory locations that byte variables can be stored in.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

Activity 3.7
Key in, download and run the following program.

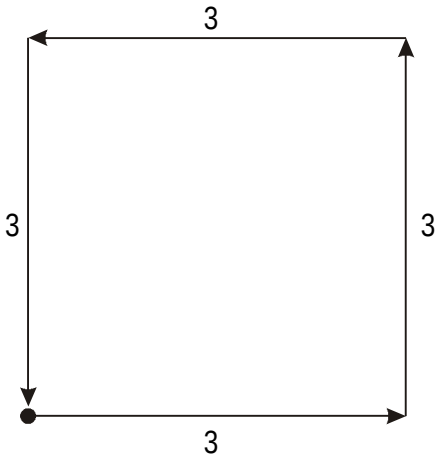
```
symbol counter = b0     ' define the variable "counter"
symbol red = 7          ' define pin 7 with the name "red"

init: let dirs = %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 3.8



Connect a buggy to the output driver module (or MFA movement module) as shown in the diagram.



The buggy should follow the movement path as shown in the diagram above.

Draw a flowchart for the movement of the buggy, making use of a for...next command structure.

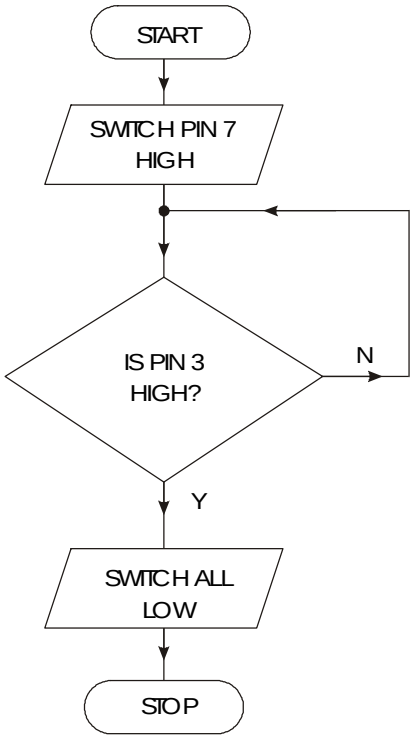
Write a high level program in PBASIC to control the movement of the buggy as shown by your flowchart. (It will be necessary to experiment with time delays to establish how quickly your buggy moves and turns).

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
	7	
	6	Yellow Light
	5	Purple Light
	4	
	3	Blue Light
	2	Green Light
	1	
	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.

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

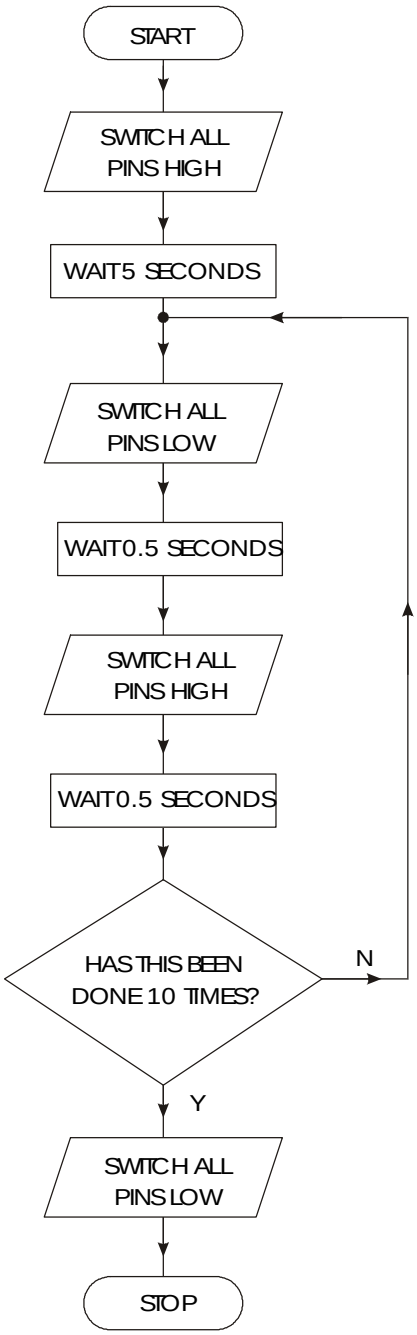
Activity 3.10
Key in, download and run the program listed below.

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

main: let pins = %10000000    ' switch pin 7 high
      if pin3 = 1 then skip   ' jump to 'skip' if input 3 is high
      goto main               ' loop

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

To test the program you will need to connect the sensors module to the Stamp Controller, and connect a micro-switch to sensor 3 on the sensor module.

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 3.11

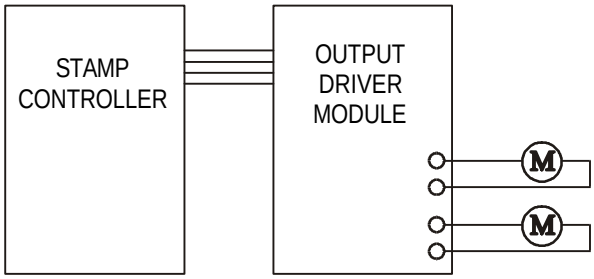


Develop a PBASIC program that will carry out the instructions shown in the flowchart above.

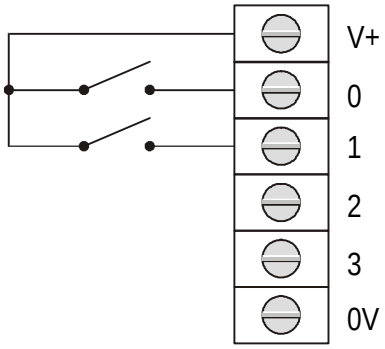
Assignment 3.12
A motor, connected to pin 7, is to run when a 'start' switch (connected to pin 2) is momentarily pressed. The motor continues to run until another 'stop' switch (connected to pin 3) is pressed - at which point the motor switches off. The system should then reset to wait for another 'start' signal.

Draw a flowchart and write a PBASIC program for this sequence.

Assignment 3.13



Connect a buggy to the output driver module (or MFA movement module) as shown in the diagram. Connect the micro-switch 'bumpers' to pins 0 and 1 on the Stamp Controller via the screw terminals as shown below.

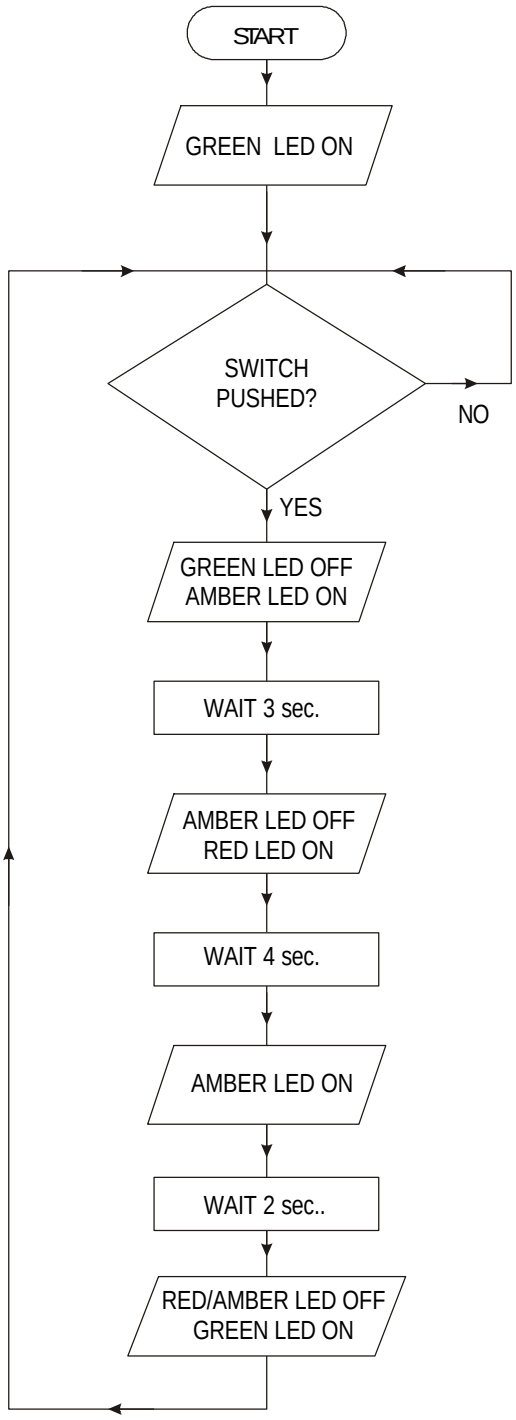


SHOWS HOW TO WIRE 2 SWITCHES TO
THE SCREW TERMINALS ON THE
STAMP CONTROLLER

The buggy should continue in a forward direction until either of the two micro-switch bumpers is activated. At this point the buggy should reverse for 3 seconds, rotate 90 degrees clockwise, and then continue in a forwards direction.

Draw a flowchart and write a PBASIC program to control the movement of the buggy as described above.

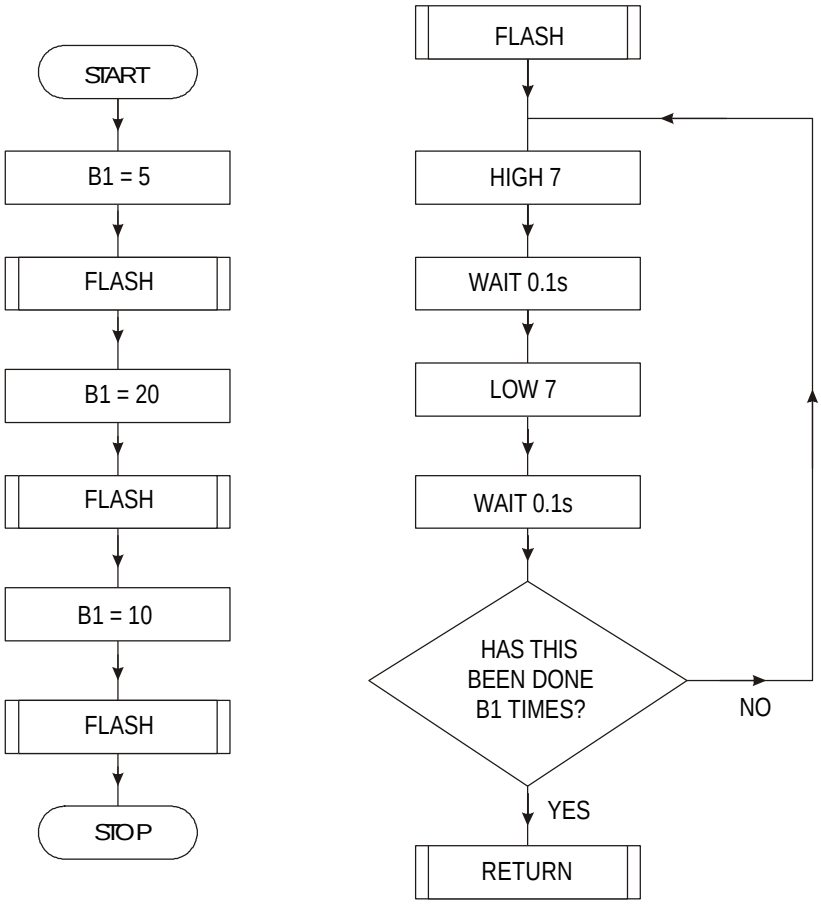
Assignment 3.14



Develop a PBASIC program that will carry out the instructions shown in the flowchart above. Use the following pin configuration.

Input Connection	Pin	Output Connection
	7	Red Light
	6	Amber Light
	5	Green Light
	4	
Start Switch	3	
	2	
	1	
	0	

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 3.15
Key in, download and run the following program.

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

main: let pins = %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 7               ' output pin on
         pause 100            ' wait 100ms
         low 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 Stamp Controller 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.

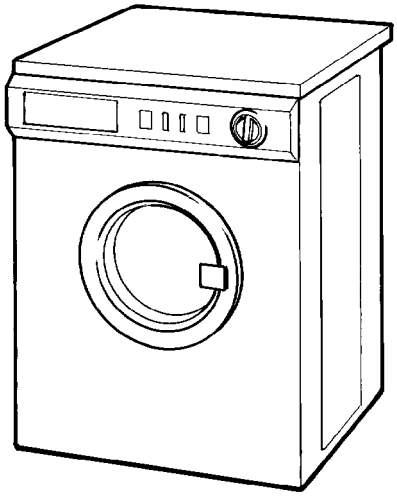
Assignment 3.16
As part of a shop display, a lighting sequence is to be controlled by a microcontroller. The output connections are shown below.

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

Assignment 3.17



Connect the washing machine model to the Stamp Controller.

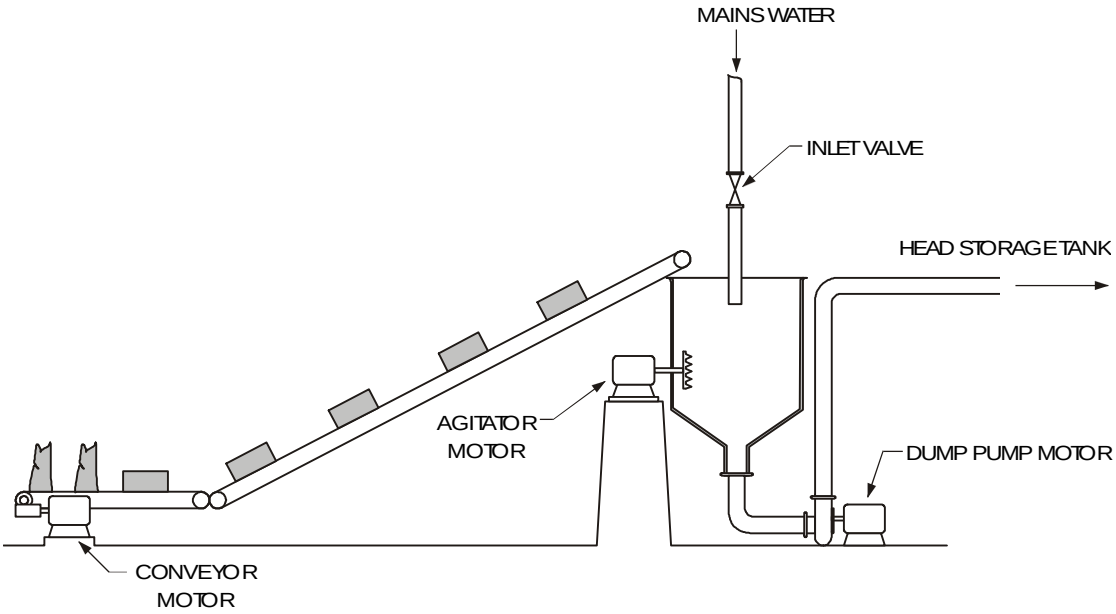
A sequence of events for the operation of the washing machine is given in the table below:

Step 1	Wait until the door switch is closed.
Step 2	Wait until the start switch is pushed.
Step 3	Rotate the drum clockwise for 10 seconds.
Step 4	Rotate the drum anti-clockwise for 10 seconds
	Repeat steps 3 and 4 10 times
Step 5	Rotate the drum clockwise for 5 seconds.
Step 6	Rotate the drum anti-clockwise for 5 seconds
	Repeat steps 5 and 6 5 times

Draw a flowchart for the control sequence described above.

Write, and test, a PBASIC program for the control sequence described above.

Assignment 3.18



The diagram shows the layout of a pulper machine used in a paper mill.
A batch of pulp and fillers, i.e. 5 x 20 kg bales of pulp and 2 x 50 kg bags of fillers, is mixed during the pulping process with 1600 litres of water.

The operation of the pulping process is controlled by a microcontroller. The microcontroller connections are as follows:

Input Connection	Pin	Output Connection
	7	Agitator motor
	6	Dump Pump
	5	Conveyer
	4	Inlet valve
	3	
	2	
	1	
Start switch	0	

Event	Operator Action	Control Activity
1	Load conveyer with batch	
2	Press start button	Sequence commences.
3		Open inlet valve. Close valve after 10 minutes.
4		1 minute after inlet valve opens, start agitator motor. run agitator motor for 11 minutes.
5		Start conveyer 2 minutes after inlet valve opens. Run conveyer for 3 minutes.
6		1 minute after agitator motor stops, start dump pump. run dump pump for 3 minutes.
7		Reset and wait for next start command.

(a) Based on the instructions given in the table, draw up a flow chart which shows the control sequence for the pulping sequence.

(b) With reference to your flow chart, write a high level program in PBASIC to control the operation of the pulper.

Assume use of a pre-written time delay sub-procedure called 'delay' which will produce a time delay in multiples of one minute, according to the value set in the variable 'mins' before the sub-procedure is called.

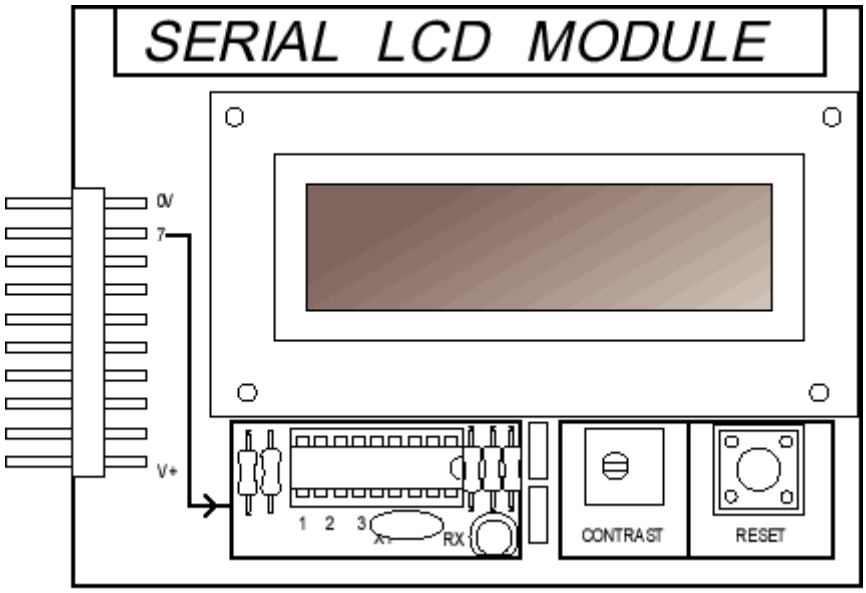
Example - To wait five minutes:

```
let mins = 5
call delay
```

Specialised Commands

The PBASIC language also contains some specialised commands dedicated for use with microcontrollers. One of the most useful commands is **serout**.

The serout command can be used to send serial ASCII text strings out of a single pin. This can be used to send information to a host computer via a serial link, or to another device, such as the serial LCD module, which can interpret ASCII text strings.



For example, to send the word "Hello" out of pin 7, the PBASIC command would be
```
serout 7,T2400,("Hello")
```

Connect the serial LCD module to the Stamp Controller. Read through the serial LCD datasheet so that you understand the operation of the LCD module.

Activity 3.19
Key in, download and run the program listed below. The program prints two different messages onto the two lines of the LCD.

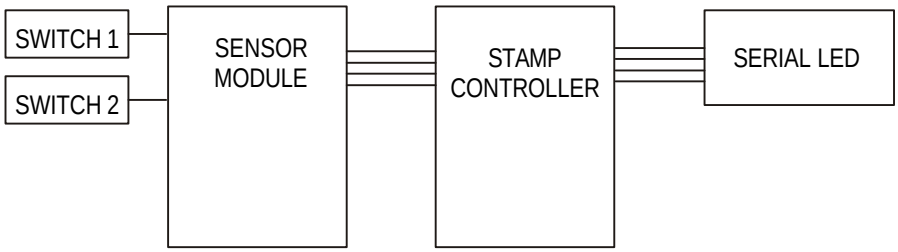
```
init: let dirs = %10000000          ' make pin 7 an output
      let pins = %10000000          ' switch pin 7 high

main: pause 5                       ' short pause
      serout 7,T2400,(254,1)        ' send 'clear LCD' command
      pause 500                     ' wait 0.5 second

      serout 7,T2400,(254,128)      ' send 'line 1' command
      serout 7,T2400,("Hello!")     ' send message

      pause 1000                    ' wait 1 second

      serout 7,T2400,(254,192)      ' send 'line 2' command
      serout 7,T2400,("Goodbye!")   ' send message

      pause 1000                    ' wait 1 second
      goto main                     ' loop
```

Activity 3.20
Key in, download and run the program listed below. The program demonstrates how the value of a variable can be displayed.

```
symbol counter = b0                       ' define the variable

init: let dirs = %10000000                ' make pin 7 an output
      let pins = %10000000                ' switch pin 7 high

      pause 5                             ' short pause
      serout 7,T2400,(254,1)              ' send 'clear LCD' command
      pause 30                            ' wait 30 ms

main: for counter = 0 to 20               ' start a for...next loop
         serout 7,T2400,(254,128)         ' send 'line 1' command
         serout 7,T2400,("Count = ")      ' send message
         serout 7,T2400,(#counter,"  ")   ' send counter value
         pause 500                        ' wait 0.5 second
      next counter                        ' next loop
      end
```

Note how the # prefix to the variable name causes the 'value' of the variable to be transmitted, rather than the direct ASCII code. It is always a good idea to transmit two spaces after a variable value. This ensures that a single digit number ("5") correctly 'overwrites' an existing three digit number ("123"). (If the spaces were not added after the "5" the display would show "523" instead).

Assignment 3.21



Connect two switches to the sensors module, and the serial LCD module, to the Stamp Controller as shown.

When the switch connected to pin '0' is pressed the Stamp Controller should **add 1** to the current total, and then display the new value on the LCD. When the switch connected to pin '1' is pressed the Stamp Controller should **subtract 1** from the current total, and then display the new value on the LCD.

Draw a flowchart and write a PBASIC program to control the operation described above.

Section 4 - Mechatronic System Interfacing Circuits

The purpose of this section is to understand the need for interfacing input and output devices when building mechatronic systems that are controlled by a microcontroller.

When you have completed this section you should be able to:
- explain why interfacing circuits are required within mechatronic systems
- select appropriate interfacing techniques for common output devices
- understand the different operation of common switch types
- understand how unipolar stepper motors are controlled
- understand the need for D-A conversion
- understand how push-pull drivers are used to control dc motors
- understand pulse-width modulated control of dc motors
- understand the term 'soft-start' when applied to dc motors
- write PBASIC programs to control stepper motors
- write PBASIC programs to control the speed and direction of dc motors

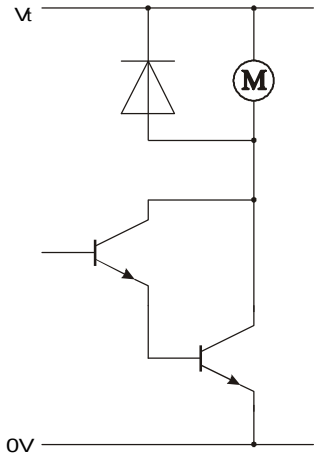
Before you start this section you should have a basic understanding of:
- The decimal, binary and hexadecimal number systems.

To complete the exercises in this section you require:
- Stamp Controller
- Output Driver Module
- MFA Sensor Module and Sensors
- DC Motor
- Stepper Motor
- MFA Stepper Motor Driver
- MFA Power D to A Converter
- Buggy (with microswitch sensors)
- Lock Model
- Light Seeker Model
- Conveyer Belt / Lift Model

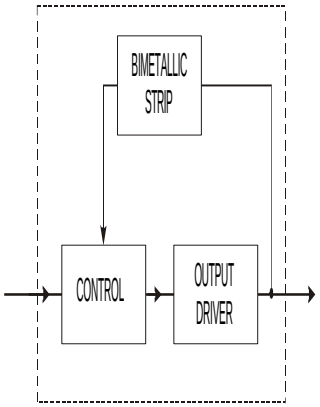
Section 4 - Mechatronic System Interfacing Circuits

The microcontroller input/output pin can only provide a small drive current when configured as an output. Therefore most output devices require an interfacing circuit.

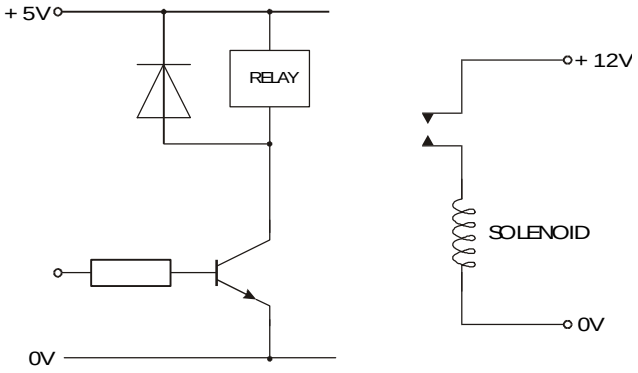
The simplest interfacing circuit uses a bipolar transistor, and the Darlington pair configuration is often used to increase the drive current capabilities.



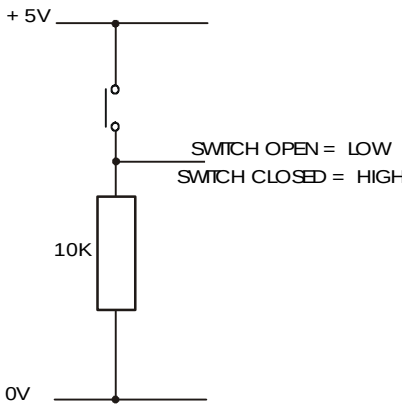
The ULN2803A integrated circuit can be very useful because it contains eight 'Darlington transistor pairs' in a convenient 18 pin DIL package. For added convenience the back emf protection diodes are also included on the integrated circuit.



For devices which use larger currents (e.g. a solenoid), or a separate power supply (e.g. mains electricity) , a relay can be used. The relay is usually driven by a transistor as shown in the diagram.

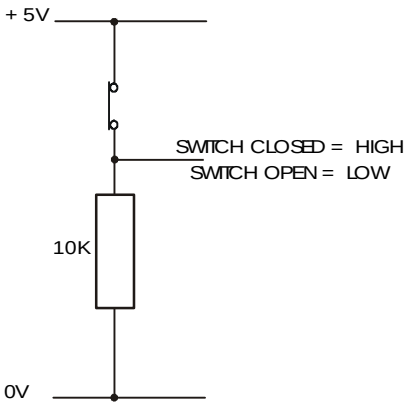


Connecting Digital Switches



NORMALLY OPEN SWITCH

Digital switches are available in two main 'families'. **Normally-open** switches are the most common. As the name suggests, the contacts on these switches are normally open, so the switch is electrically 'off'. When the switch is activated the contacts close and so the switch is electrically 'on'.



NORMALLY CLOSED SWITCH

Normally-closed switches operate in reverse. The contacts are normally closed, and so the switch is normally electrically 'on'. When the switch is activated the contacts open and so the switch is electrically 'off'. This type of switch are commonly used in alarm systems, when a 'break' in the circuit needs to be detected.

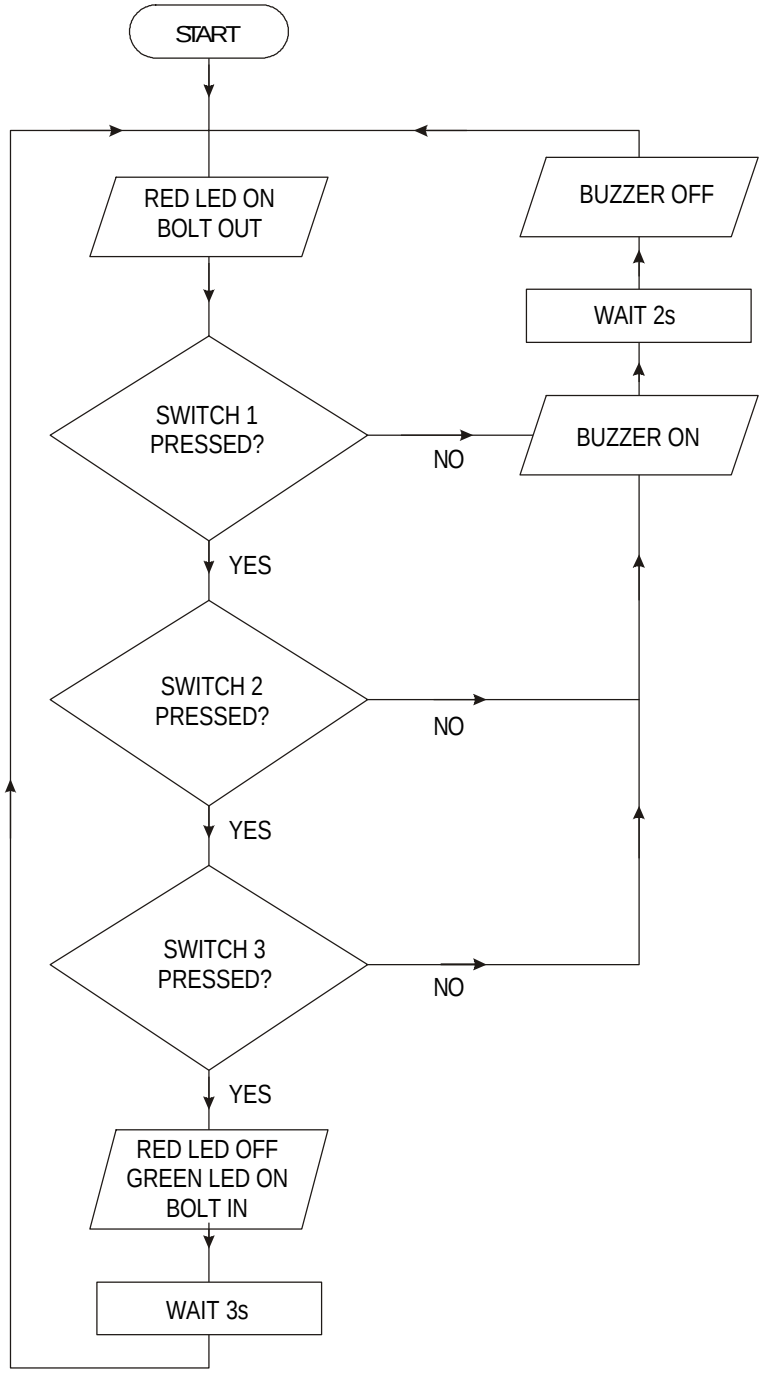
Assignment 4.1

Draw appropriate interfacing circuit diagrams for the following output devices. Clearly explain your choice of circuit in each case.

a) LED
b) Buzzer
c) 6V DC motor
d) 12V DC solenoid
e) 24V pneumatic solenoid valve

Assignment 4.2

Connect the electronic lock model to the Stamp Controller.

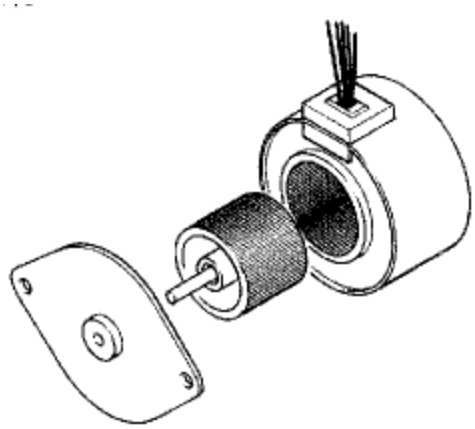


Write, and test, a PBASIC program for the control sequence shown in the diagram above.

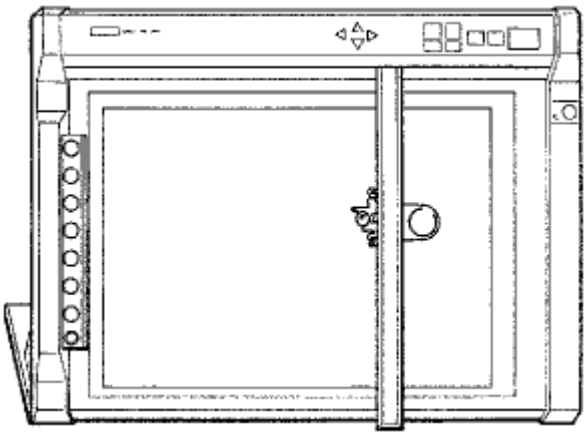
Input Connection	Pin	Output Connection
	7	Bolt
	6	Buzzer
	5	Green LED
	4	Red LED
Switch 3	3	
Switch 2	2	
Switch 1	1	
	0	

Develop the lock sequence further so that the switches must be pressed consecutively, rather than simultaneously, for the lock to operate.

Stepper Motors

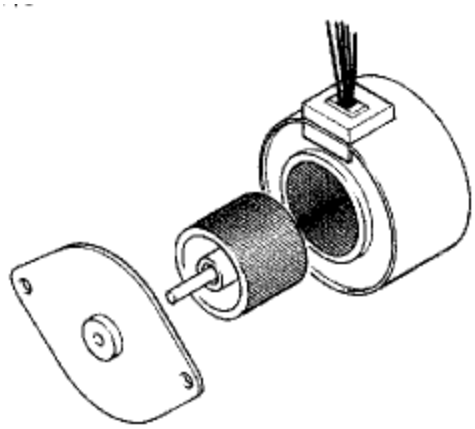


Stepper motors are very accurate motors that are commonly used in computer disk drives, printers, XY plotters and clocks. Unlike dc motors, which spin round freely when power is applied, stepper motors require that their power supply be continuously pulsed in specific patterns. For each pulse, the stepper motor moves around one 'step', typically 7.5 degrees (giving 48 steps in a full revolution).

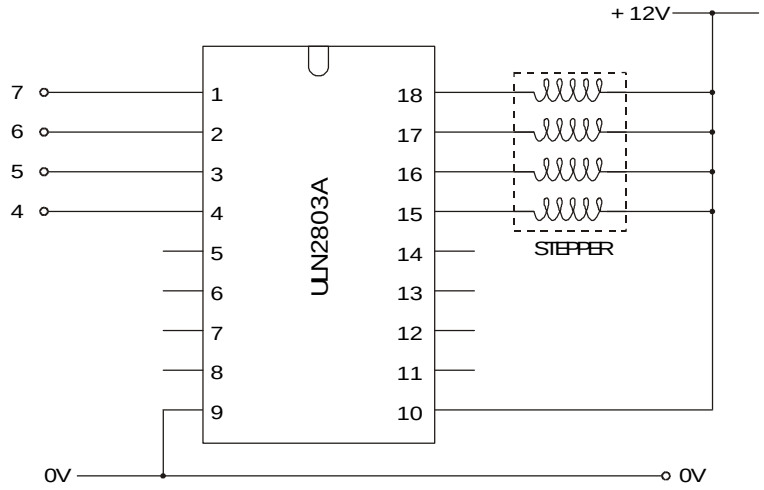


Stepper motors do have some limitations. First, the power consumption is greatest when the stepper motor is stopped (as all coils are still energised). Secondly the output torque is not very high, and so some type of gearbox is normally required. Unless this gearbox is very carefully designed the 'backlash' of the gears can disrupt the accuracy of the final system. Finally the speed of revolution is limited to around 100 steps per second, which provides a rotational speed of 2 rev / s or 120 rev / min.

There are two main types of stepper motors - unipolar and bipolar. Unipolar motors usually have four coils which are switched on and off in a particular sequence. Bipolar motors have two coils in which the current flow is reversed in a similar sequence. It is the unipolar type which is described in this section.



The stepper motor contains magnets which are fixed to the central armature. Four electronic coils are located around the casing. When a current is passed through these coils they generate a magnetic field, which attract/repels the permanent magnets on the armature, and so the armature spins one 'step' until the magnetic fields align. The coils are then energised in a different pattern to create a different magnetic field, and the armature spins another step.



To make the armature rotate continuously the four coils must be switched on and off in a certain order. Many microcontroller systems use four output lines to control the stepper motor, each output line controlling the power to one of the coils.

As the stepper motor operates at 12V, a transistor switching circuit is used to switch each coil. As the coils create a back EMF when switched off, a suppression diode on each coil is also required. The ULN2003 Darlington driver integrated circuit provides a convenient device housing these transistors and coils.

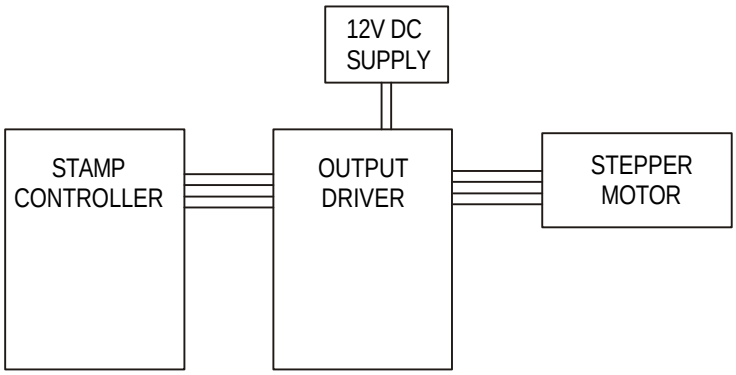
The table below show the four different steps required to make the motor turn

Step	Coil 4	Coil 3	Coil 2	Coil 1
1	1	0	1	0
2	1	0	0	1
3	0	1	0	1
4	0	1	1	0
1	1	0	1	0

Note:
The wiring configuration of stepper motors varies from different manufacturers. Therefore, it may be necessary to rearrange the coil connections for the above sequence to operate correctly. An incorrect coil arrangement will result in the stepper motor oscillating back and forth rather than rotating.

The sequence of wires for the Middlesex Stepper Motor is, from the top:
white white blue yellow brown red and you may have to change the wires on these stepper motors to this sequence

Activity 4.3



Build the circuit as shown.

Key in, download and run the program listed below. The program demonstrates how to continuously spin the stepper motor. Try changing the speed by altering the value of delay.

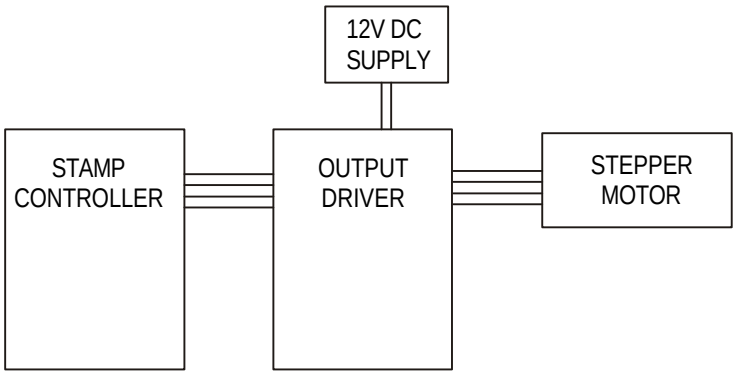
```
symbol delay = b0                       ' define the variable

init: let dirs = %11110000              ' make pins 4-7 outputs
      let delay = 100                   ' set delay to 100ms

main: let pins = %10100000              ' first step
      pause delay                       ' pause for delay
      let pins = %10010000              ' next step
      pause delay                       ' pause for delay
      let pins = %01010000              ' next step
      pause delay                       ' pause for delay
      let pins = %01100000              ' next step
      pause delay                       ' pause for delay
      goto main                         ' loop forever
```

Assignments

4.4) Describe three products which may contain stepper motors. Describe how the motor is used in each case.

4.5) A toy manufacturer is designing a new programmable robot toy. Describe the advantages and disadvantages of using stepper motors (rather than dc motors) to manoeuvre the robot.

Assignment 4.6



Connect a stepper motor to the output driver module as shown.

Develop a PBASIC program that will rotate the stepper motor 48 steps in one direction, and then 48 steps in the other direction.

Assignment 4.7

Connect the stepper motor light seeker apparatus to the Stamp Controller.

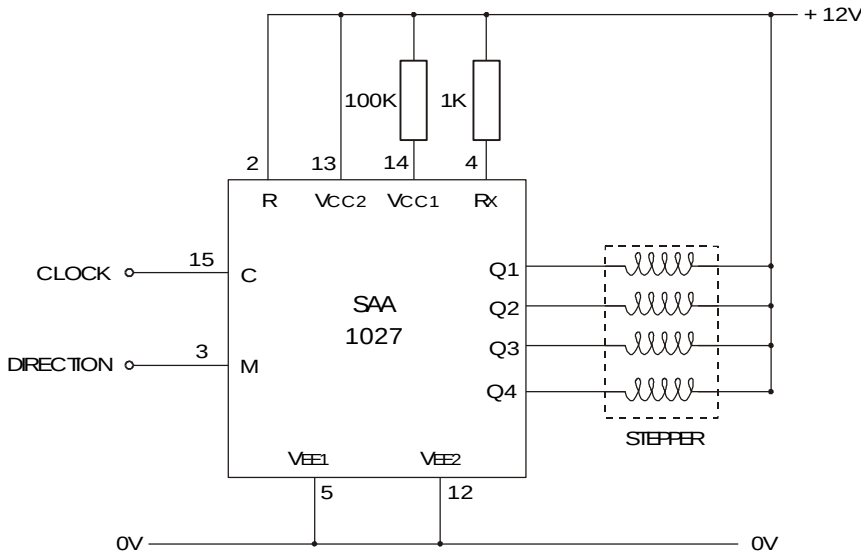
Input Connection	Pin	Output Connection
	7	Coil 4
	6	Coil 3
	5	Coil 2
	4	Coil 1
	3	
	2	
LDR1	1	
LDR0	0	

The two LDRs are connected to input pins 1 and 2, and the two potentiometers may require adjustment for sensitivity (depending on light conditions).

Draw a flowchart and write a PBASIC program that will cause the apparatus to 'follow' a light source that is moved around the apparatus.

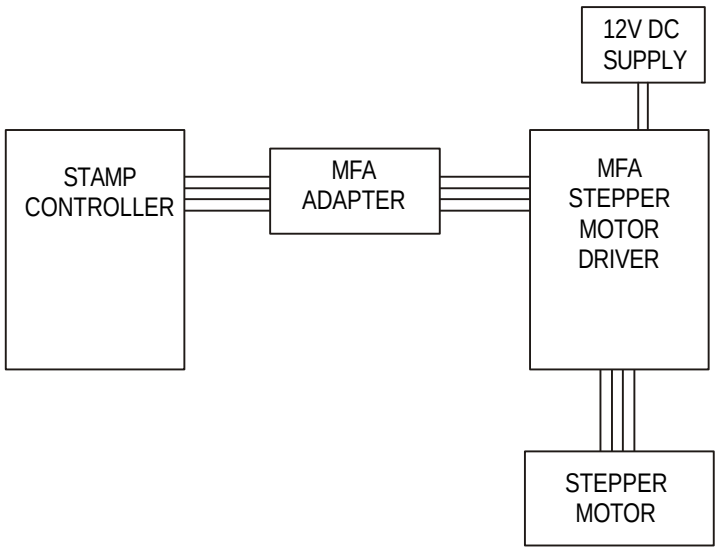
Stepper Motor Driver IC

The use of four output pins to drive a stepper motor can be an inefficient way to use the microcontroller input/output pins. A dedicated integrated circuit, called the SAA1027 stepper motor driver, has been designed to overcome this problem by building the logic step generator, and the transistor switches, into one package.



The stepper motor driver IC just requires two signals - direction and step. The 'direction' signal sets the direction of rotation, when set high the motor turns one way, when set low the motor turns the other way. The 'step' signal is pulsed (switched high then low again) to move the stepper motor one step. Therefore to move the stepper motor ten steps the 'step' pin would be pulsed ten times.

Activity 4.8



Build the circuit as shown.

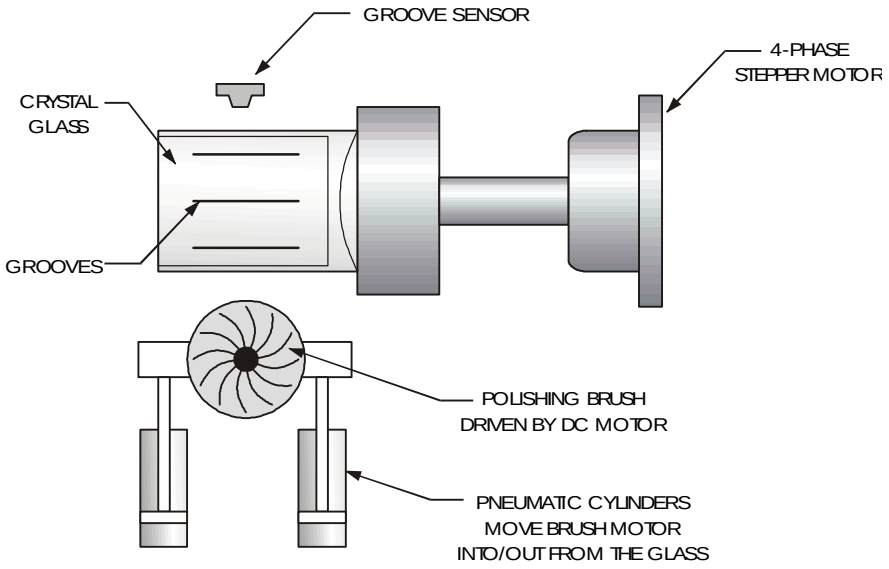
Key in, download and run the program listed below. This program moves a stepper motor a full rotation (48 steps) in each direction.

```
symbol direct = 5                  ' pin 5 is direction pin
symbol clock = 4                   ' pin 4 is clock pin
symbol counter = b0                ' variable b0 is loop counter

init: let dirs = %00110000         ' pin 4 & 5 outputs

main: high direct                  ' set dir pin high

      for counter = 1 to 48        ' setup a for...next loop
         high clock                ' clock pin high
         pause 10                  ' wait 10ms
         low clock                 ' clock pin low
         pause 10                  ' wait 10ms
      next counter                 ' next loop

      low direct                   ' set dir pin high

      for counter = 1 to 48        ' setup a for...next loop
         high clock                ' clock pin high
         pause 10                  ' wait 10ms
         low clock                 ' clock pin low
         pause 10                  ' wait 10ms
      next counter                 ' next loop

      goto main                    ' loop forever
```

Assignment 4. 9

Crystal glassware is decorated by cutting grooves on to the outside surface. The process leaves the untouched parts of the glass clear and reflective, whereas the cut grooves are dull and rough and require to be polished.

The diagram shows the layout of a prototype system, developed by a student, to polish crystal glasses with eight parallel grooves cut along the length of the glass.



The system is to be controlled by a microcontroller. With a glass in place, the start button is pressed to cause the cutting disk to switch on and move into the cutting position. The first groove is cut for five seconds. The cutter is then withdrawn. The stepper motor rotates the glass through 90 degrees into position for the next groove. The process continues until all four grooves have been cut.

The stepper motor used by the student has a step angle of 7.5 degrees. It is controlled by means of an SAA1027 driver IC signalled from the microcontroller.

Input Connection	Pin	Output Connection
	7	Pneumatic Cylinder (1 = outstroke, 0 = instroke)
	6	DC Motor (1 = on, 0 = off)
	5	Stepper Direction (0 to 1 pulse = 1 step)
	4	Stepper Pulse (1 = clockwise, 0 = anticlockwise)
	3	
	2	
	1	
Start button (logic 1 when pressed)	0	

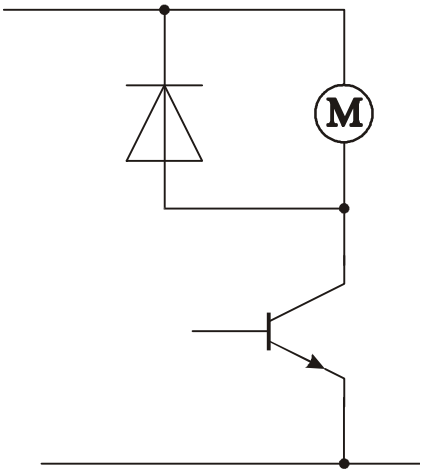
(a) Calculate the number of steps the stepper motor must rotate to rotate the glass through 90 degrees.

(b) Draw up a flow chart which shows the control sequence for the polishing process.

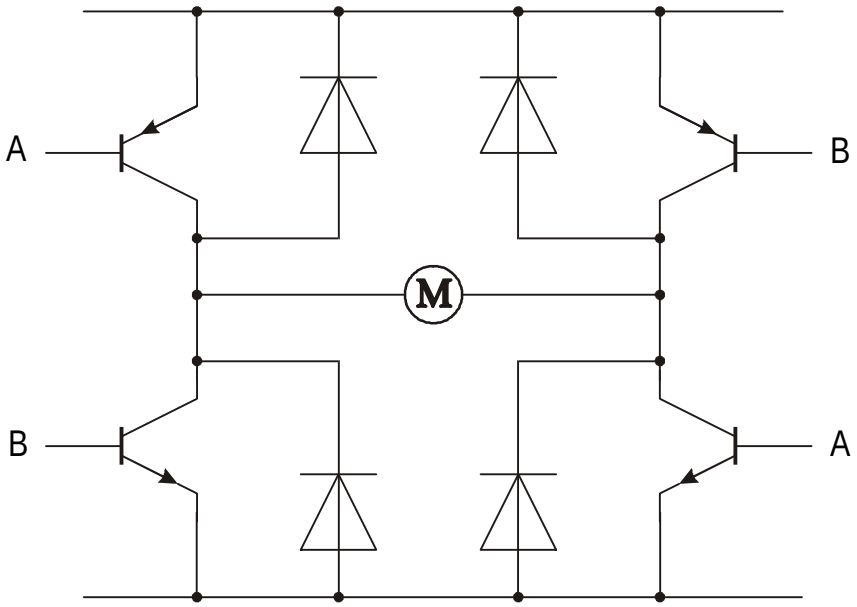
(c) With reference to your flow chart, write a high level program in PBASIC to control the polishing process.

Push-Pull Motor Drivers

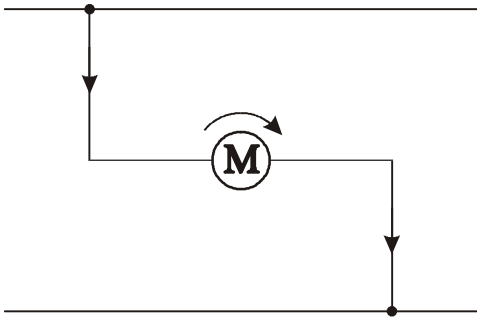
A dc motor could be switched on and off via a single transistor as shown below.



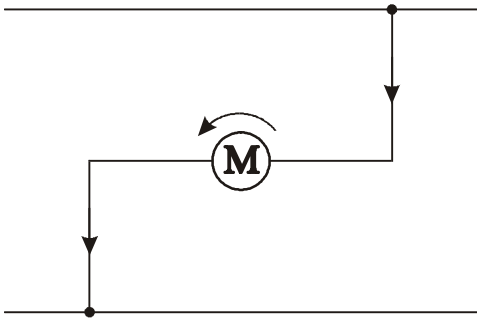
However this only allows the motor to spin in one direction. To allow the motor to spin in either direction a **push-pull** arrangement of transistors can be used, as shown in the diagram below.



Note the use of the protection diodes across each transistor to protect the transistors from the back EMF generated by the motors.



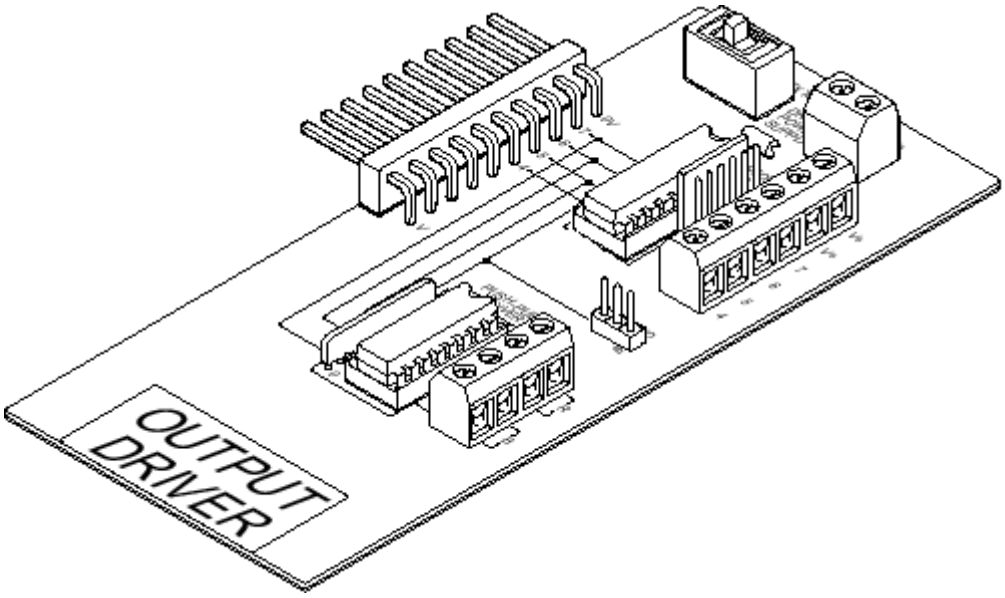
When the two transistors labelled A are both switched on the motor spins in one direction. If the other two transistors are switched on the motor spins in the other direction.



Naturally it is important that both sets of transistors are not switched on simultaneously! This would result in a short circuit between the power rails and the transistors would overheat and hence be destroyed.

It is possible to buy a dedicated integrated circuit, called the L293D, which contains all the transistors and protection diodes required. The inputs are also specifically protected so that the short-circuit condition cannot be achieved.

The L293D IC is included on the Output Driver module.

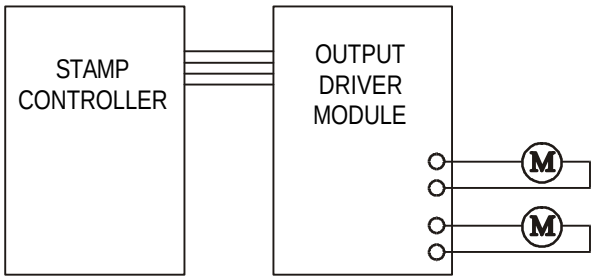


The following tables show how the two motor outputs 'A' and 'B' are controlled by the four input pins.

pin 4	pin 5	motor A
0	0	halt
0	1	forward
1	0	reverse
1	1	halt

pin 6	pin 7	motor B
0	0	halt
0	1	forward
1	0	reverse
1	1	halt

Activity 4.11



Build the circuit as shown.

Key in, download and run the program listed below. The program demonstrates how to spin both dc motors in both directions.

```
init: let dirs = %11110000                ' make pins 4-7 outputs

main: let pins = %01010000                ' motors forward
      pause 3000                          ' pause for 3 sec
      let pins = %00000000                ' motors halt
      pause 3000                          ' pause for 3 sec
      let pins = %10100000                ' motors reverse
      pause 3000                          ' pause for 3 sec
      let pins = %11110000                ' motors halt
      pause 3000                          ' pause for 3 sec
      goto main                           ' loop forever
```

Assignment 4.12

Connect the conveyer belt model to the Stamp Controller.

A single block is to be moved continuously back and forth along the conveyer belt without falling off either end.

Draw a flowchart that will control the movement as described. Write, and test, a PBASIC program for the control sequence as drawn in your flowchart.

DC Motor Speed Control

The speed of a dc motor varies directly with the voltage applied across it. Microcontroller do not normally have an analogue output, and so they cannot be used to vary the voltage to a motor in this manner.

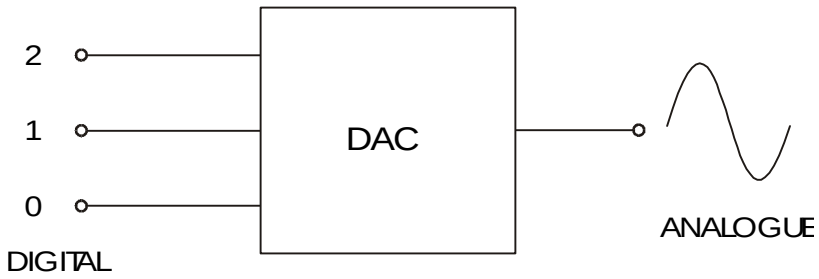
However there are two main methods by which a microcontroller can control the speed of a DC motor:

- Digital to Analogue Conversion (DAC)
- Pulse Width Modulation

Digital to Analogue Conversion

The simplest way to control the speed of a DC motor is to vary the voltage applied to the motor coils - the higher the voltage the faster the motor will spin (within the motor operating limits).

However the digital output from a microcontroller is at a fixed voltage, and the microcontroller cannot supply enough current to drive the motor. Therefore an interfacing circuit is required to boost the supply current for the motor and to provide different voltage levels.

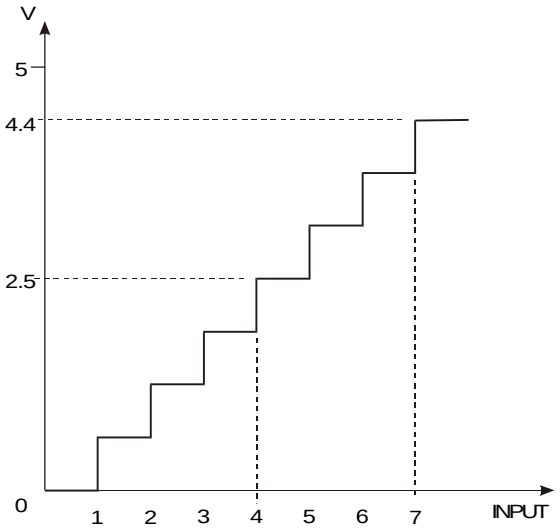


A **Digital to Analogue Converter (DAC)** is an integrated circuit that decodes binary information and generates an analogue voltage proportional to the binary information provided.

A three bit DAC, with an output range of 0-5V, may produce a voltage output according to the table below:

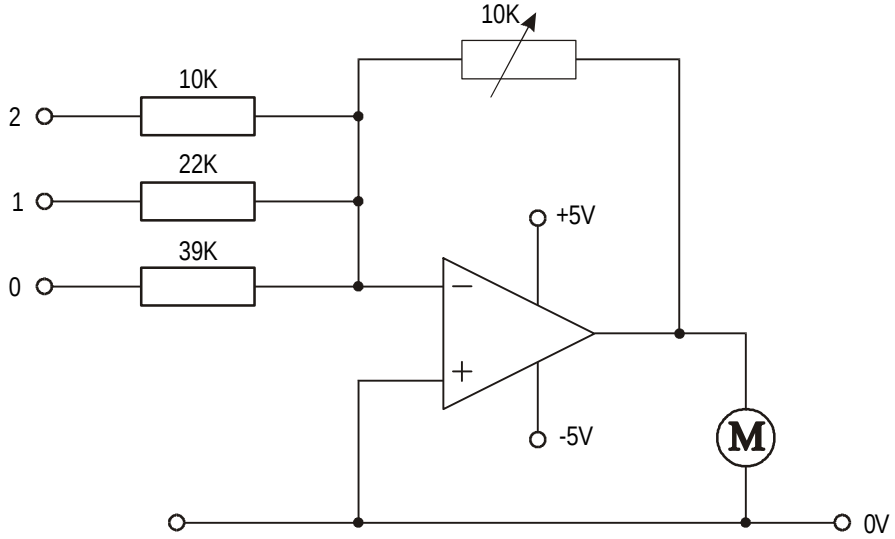
binary input	analogue output (V)
000	0.00
001	0.62
010	1.25
011	1.87
100	2.50
101	3.12
110	3.75
111	4.37

Each increase in the binary input produces a step increase of 0.625V (5V ÷ 8 steps) in the analogue output. In most practical applications the DAC cannot supply the full output voltage (e.g. 5V) due to the technical operating limitations of the device.



When plotted as a graph this transfer function appears as a 'staircase' shape. An increase in the number of input bits will increase the resolution of the DAC, therefore producing 'smaller steps'.

The simplest form of DAC is made from a summing amplifier, as shown in the diagram below.

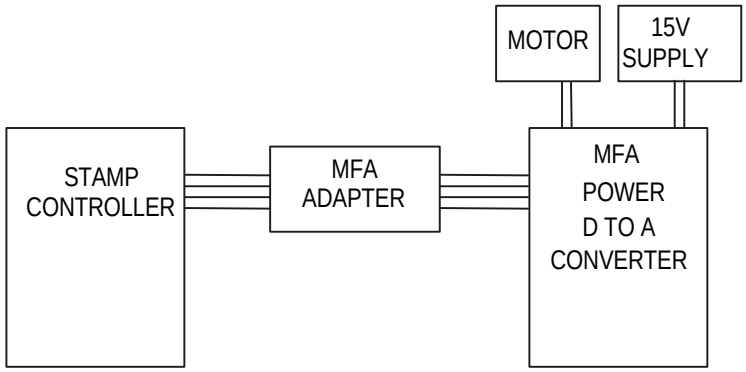


In this example the output voltage is equal to

Vo = -5 (Pin 2 x (10K/10K) + Pin 1 x (10K/22K) + Pin 0 x (10K/39K))
= -5 (Pin 2 + 0.5 (Pin 1) + 0.25 (Pin0))

where the pin value is 0 or 1 (presuming a high logic signal of 5V)

Activity 4.13



Build the circuit as shown.

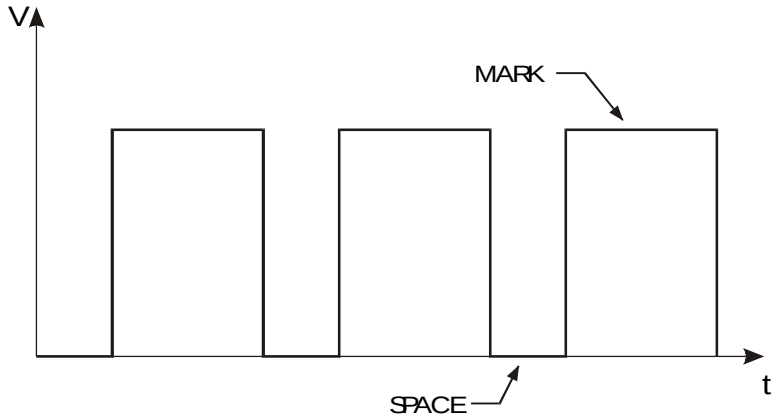
Key in, download and run the program listed below. This program drives the motor at eight different speeds for five seconds each speed.

```
symbol counter = b0                 ' variable b0 is loop counter

init: let dirs = %11110000          ' pin 4-7 outputs
      let pins = %00010000          ' starting speed

main: high direct                   ' set dir pin high
      for counter = 0 to 7
         let pins = pins + 16 ' next speed
         pause 5000                 ' wait 5 seconds
      next counter                  ' next loop

      end                           ' end
```

Pulse Width Modulation

With DAC, the voltage applied to the motor is directly varied. As the voltage is decreased the motor turns more slowly. However the current flowing through the motor coils also decreases, and so the output torque (turning moment) of the motor also falls.

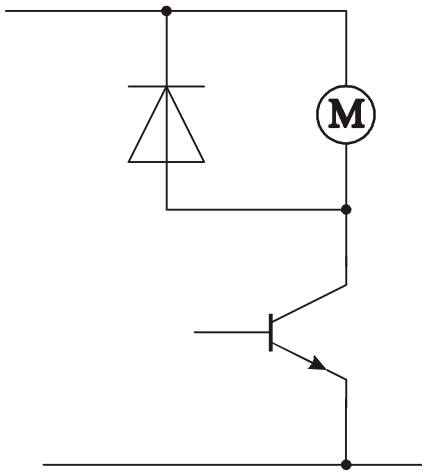
Therefore this solution is often unsatisfactory for controlling DC motors due to the undesirable loss of motor torque. **Pulse Width Modulation (PWM)** is a digital method which can be used to vary the motor speed. In this method the full voltage is applied to the motor, but it is rapidly pulsed on and off. By varying the on and off ratio of the pulses the speed of the motor can be varied. As the full voltage is applied to the motor during the 'on' pulses the torque of the motor remains high.



The graph shows how the technique is applied. The 'on' time for the motor is called the mark, the 'off' time is called the space. When the voltage is applied to the motor it accelerates to top speed. However before the top speed is reached the motor is switched off, thus slowing it down. By increasing the frequency of the pulses this acceleration/deceleration becomes negligible, and the motor rotates constantly at a slower speed.

The PWM technique does have certain limitations. It cannot be used with mechanical relays, as the rapid switching would damage the mechanical contacts. The frequency of the pulses must also be carefully selected - if the frequency is too slow the motor will stall.

Activity 4.14



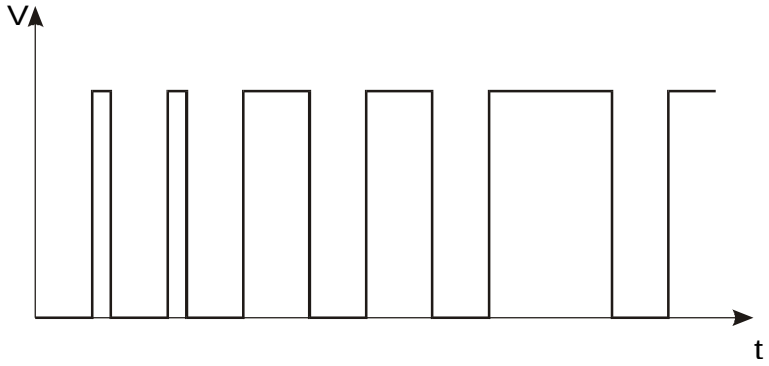
Build the circuit as shown. This is most simply achieved by connecting the dc motor across the 'V+' and '7' Darlington Driver terminals on the Output Driver module.

Key in, download and run the program listed below. This program drives the motor at approximately half speed, as the space is twice the length of the mark.

```
symbol mark = b1
symbol space = b2
symbol motor = 7

init: let dirs = %11110000          ' pin 4-7 outputs
      let mark = 10                 ' set mark to 10ms
      let space = 20                ' set space to 20ms

main: high motor                    ' output high
      pause mark                    ' pause for mark time
      low motor                     ' output low
      pause space                   ' pause for low time
      goto main                     ' loop
```

Try out different speeds (by experiment) by altering the values of 'mark' and 'space'.

Soft Start of DC Motors

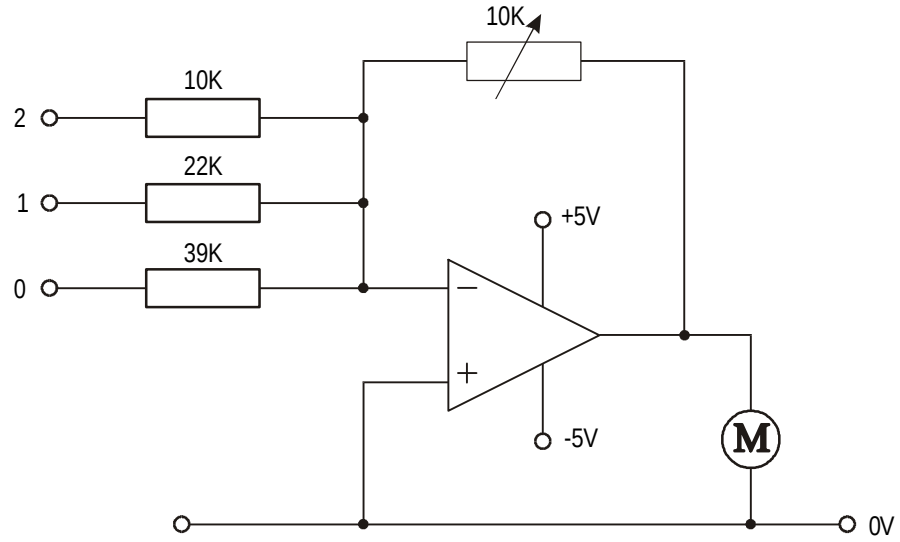


In some devices, such as electric drills, it is desirable for the motor to start rotating slowly and then build up speed, rather than rapidly 'accelerating' up to full speed. This is called '**soft starting**' the motor, and the use of PWM is often appropriate in these situations. The motor is started at a low speed and then gradually accelerated by varying the mark to space ratio over a period of time.

Activity 4.15
Key in, download and run the program listed below. This program gradually increases the speed by increasing the length of the mark time over a period of time.

```
symbol counter = b0                ' variable b0 is loop counter
symbol mark = b1
symbol space = b2
symbol motor = 7

init: let dirs = %11110000         ' pin 4-7 outputs

      let mark = 10                ' set mark to 10ms
      let space = 20               ' set space to 20ms

main: gosub puls                   ' call sub-procedure
      let mark = mark + 2          ' increase mark time
      goto main                    ' loop

' sub-procedure

puls: for counter = 0 to 50        ' start a for...next loop
         high motor                ' output high
         pause mark                ' pause for mark time
         low motor                 ' output low
         pause space               ' pause for low time
      next counter                 ' loop
      return                       ' return from sub-procedure
```

Assignment 4.16
Details of a speed control unit for a dc motor are shown below.



a) Name the configuration of operational amplifier being used.

b) Name the method of dc motor speed control being used.

c) Explain clearly how the system operates.

d) State what the output voltage supplied to the motor will be when the following values are applied to the input pins (presume the feedback resistor is at it's maximum value of 10K).
   i) 1
   ii) 3
   iii) 4

Assignment 4.17
A small dc motor, which is used to drive a model conveyer belt, is to be controlled by a microcontroller. The microcontroller gives out a 6V signal when an output bit is switched to logic 1. The speed of the motor must be varied according to the loads which are being carried. for convenience, the speed is to be reduced using pulse-width modulated control.

Output connections to the microcontroller are as follows.

Input Connection	Pin	Output Connection
	7	
	6	
	5	Motor Anticlockwise
	4	Motor Clockwise
	3	
	2	
	1	
	0	

Develop a short PBASIC procedure which could be used to drive the motor is a clockwise direction with a mark-to-space ratio of 2:1. Explain how you could alter the procedure to make the motor rotate in an anti-clockwise direction, with the same mark-to-space ratio.

TECHNOLOGICAL STUDIES

HIGHER

SYSTEMS AND CONTROL

STUDENTS' NOTES

OUTCOME 4

Outcome 4 - Microcontroller Controlled Monitoring Systems

Section 1 - Analogue to Digital Conversion

The purpose of this section is to introduce the application of A-D converters. The section explains the function and operation of A-D converters in the context of a programmable system.

When you have completed this unit you should be able to:
- understand the need for A-D conversion
- understand the use of an A-D converter with a microcontroller
- understand the term multiplexing and appreciate the reasons for it
- understand the term signal conditioning
- write PBASIC programs that include ADC routines

Before you start this section you should have a basic understanding of:
- The decimal, binary and hexadecimal number systems.
- Operational amplifiers

To complete the exercises in this section you require:
- Stamp Controller
- Serial ADC Module
- Serial LCD Module
- MFA Sensor Probes
- MFA Multiplexer Module

Section 2 - Data-Logging

The purpose of this section is to introduce the concept of data-logging. The section also involves the use of a host computer for analysing data recorded with a spreadsheet application.

When you have completed this section you should be able to:
- understand the term data-logging
- understand the term sampling frequency
- appreciate the use of data-logging in a practical application
- understand how data can be analysed with a spread-sheet application
- write PBASIC programs to control data-logging experiments

Before you start this section you should have a basic understanding of:
- Analogue to Digital Conversion.
- The decimal, binary and hexadecimal number systems.

To complete the exercises in this section you require:
- Stamp Controller
- Datalogging Module
- Serial ADC Module
- Serial LCD Module
- Serial Printer Module
- MFA Sensor Probes
- MFA Multiplexer Module

Section 1 - Analogue to Digital Conversion

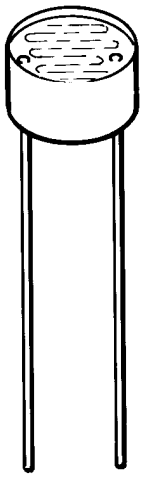
The purpose of this section is to introduce the application of A-D converters. The section explains the function and operation of A-D converters in the context of a programmable system.

When you have completed this unit you should be able to:
- understand the need for A-D conversion
- understand the use of an A-D converter with a microcontroller
- understand the term multiplexing and appreciate the reasons for it
- understand the term signal conditioning
- write PBASIC programs that include ADC routines

Before you start this section you should have a basic understanding of:
- The decimal, binary and hexadecimal number systems.
- Operational amplifiers

To complete the exercises in this section you require:
- Stamp Controller
- Serial ADC Module
- Serial LCD Module
- MFA Sensor Probes
- MFA Multiplexer Module

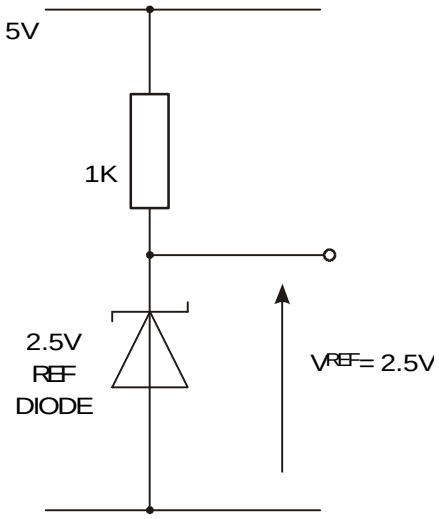
Section 1 - Analogue to Digital Conversion



The microcontroller can only process digital (high/low) signals. However in many practical applications analogue quantities, such as light intensity or temperature, need to be tested.

An **Analogue-to-Digital Converter (ADC)** is a device which can convert analogue quantities into digital signals. An 8-bit ADC can generate a digital signal in the range 0 to 255 (i.e. one byte). ADC's are available as separate integrated circuits, or may even be 'built into' the microcontroller to give it 'on-board' ADC capabilities. With the Stamp Controller system an external 8-bit ADC integrated circuit is used.

Voltage Reference



ADCs are designed to process analogue signals within a certain range. The maximum voltage signal that can be processed by an ADC is called its **reference voltage**. A common reference voltage for an 8-bit ADC is 2.55V, so the ADC can measure signals in the range 0 to 2.55V. Therefore the 8-bit ADC device will generate an 8-bit value directly equivalent to the voltage signal applied.

Signal Conditioning

Analogue sensors, such as the thermistor (temperature sensor), may not directly produce a suitable signal for use with the ADC. Therefore it is often necessary to use a **signal amplifier** circuit so that the input signal can be 'conditioned' to the input range of the ADC (0 to 2.5V). An **operational amplifier**, configured as a voltage amplifier, is commonly used for this task. The following example shows how the gain of the operational circuit is typically calculated.

Example:

A temperature sensor is to be used to monitor the temperature of a factory furnace.

Experimentation has shown that the temperature sensor produces a linear output signal, reading 0V at 0º Celsius and 1.53V at a temperature of 200º Celsius. The maximum operating temperature of the furnace is 1000º Celsius. The maximum voltage that may be fed into the ADC is 2.55V

Draw a circuit diagram of a suitable signal conditioning system based on operational amplifiers.

Solution:

1) Calculate the maximum input signal, which is at 1000º Celsius.

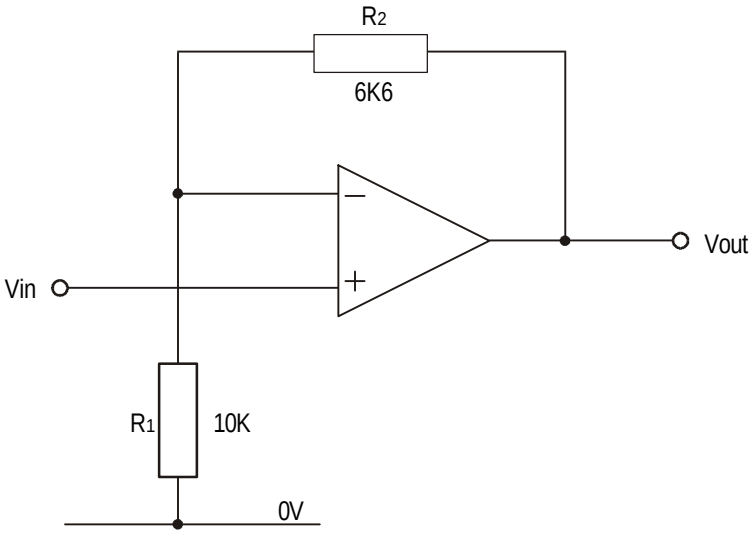
Signal at 200º = 1.53V
1000º is equal to 5 x 200º
Therefore maximum signal is 5 x 1.53V = 7.65V

2) Calculate the gain required from the op-amp circuit.

Input Signal = 7.65V
Required output signal = 2.55V

Therefore Gain = output / input = 2.55 / 7.65 = 0.33

3) Draw circuit diagram of a voltage amplifier circuit with gain set to 0.33



Assignment 1.1
A microcontroller based monitoring system is used to monitor sound signals over a period of time. It is found that the maximum voltage generated from the sound signals is 6V. However the maximum voltage that may be fed into the microcontroller system is 1.8V

a) Draw a circuit diagram of a suitable signal conditioning system, based on operational amplifiers, which will allow the signals to be monitored without damaging the microcontroller system. Indicate the values of any components used in your circuit.

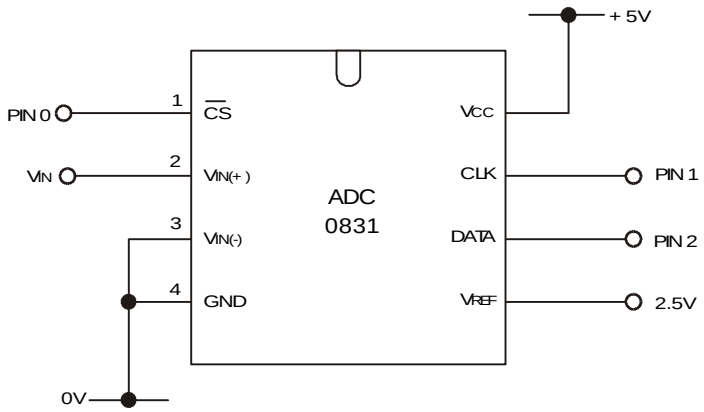
b) If the sound signal is fed into the microcontroller through an ADC which has a voltage reference of 1.8V, write down the 8-bit binary pattern you would expect from the ADC when the sound signal generates a voltage of 4.8V. Clearly identify the least significant bit (LSB).

Assignment 1.2
The signal from a temperature sensing sub-system is processed by an ADC before being read by a microcontroller. The ADC has a voltage reference of 1.8 volts which produces a binary word of %11111111

a) What is an ADC and why is it required before processing by the microcontroller?

b) Calculate the binary word produced by the ADC at 200ºC, if the voltage signal from the temperature sensing sub-system at this temperature is 1.2V. Clearly identify the least significant bit (LSB).

Interfacing to the ADC



The ADC used with the Stamp Controller is a 'serial' type device. This means that the analogue reading is transmitted as a series of eight consecutive bits via a single 'data' pin. However it is obviously necessary that the Stamp Controller and the ADC are synchronised for this communication, and this is achieved by the Stamp Controller sending a number of 'clock' pulses down a second 'clock' pin.

Therefore the sequence for communication between the two devices is:

1. The Stamp Controller selects the ADC by enabling the chip select line.
2. The Stamp Controller sends a number of clock pulses to the ADC.
3. The ADC returns a high/low signal via the data pin on each clock pulse.
4. The Stamp Controller records these signals and stores them in a variable.
5. The Stamp Controller deselects the ADC.

The PBASIC code to achieve this task is fairly complicated. However as this code is always the same, no matter the application, it is usual to save this code segment as a separate 'sub-procedure'. These standard 'sub-procedures' of code are published in datasheets, and so it is not necessary to re-write the sub-procedure for each program.

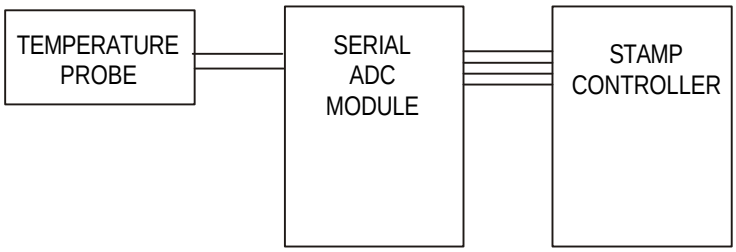
IT IS NOT NECESSARY TO LEARN THESE 'SUB-PROCEDURES'. THESE SUB-PROCEDURES WILL ALWAYS BE PROVIDED, IF NECESSARY, IN EXAMINATIONS.

WHEN CREATING NEW PBASIC PROGRAMS THE TEMPLATE FILE 'TEMPLATE.BAS' SHOULD BE USED. THIS TEMPLATE FILE CONTAINS ALL THE STANDARD SUB-PROCEDURES REQUIRED, SO THAT IT IS ONLY NECESSARY TO KEY IN THE MAIN PROGRAM.

The template.bas file is available on the Higher Still website and on a Higher Still Support Materials CD-ROM

In theory this means that as long as you know ***what*** the sub-procedures do, you do not need to understanding exactly ***how*** they accomplish the task!

Activity 1.3



Build the circuit as shown.

In the case of the serial ADC module the sub-procedure to take a reading is called 'adcread'. This sub-procedure performs a read of the ADC and then stores the reading in a variable named 'data'.

Therefore a sample program to take an analogue reading, and then transmit it to the computer for viewing (using the **debug** command), would be:

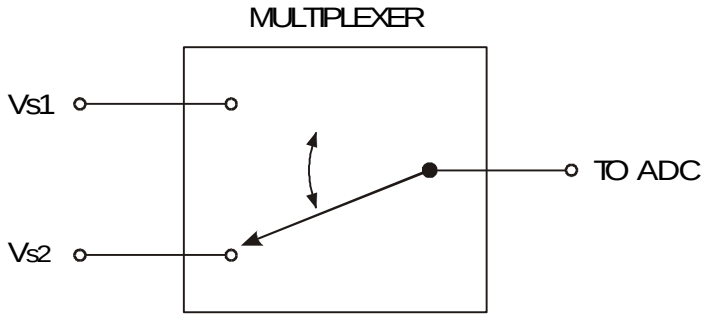
```
main: gosub adcread                 'Get the ADC reading
      debug data                    'Transmit data to computer
      pause 1000                    'Pause for 1 second
      goto main                     'Loop forever
```

Note that this is not the whole program listing, as the actual sub-procedure '**adcread**' and the symbol definitions are not included. However they are not needed to understand how the main program operates, and so can be left out for clarity.

Open the file ‘template.bas’, key in the main program above in the correct section of the template and run the program.

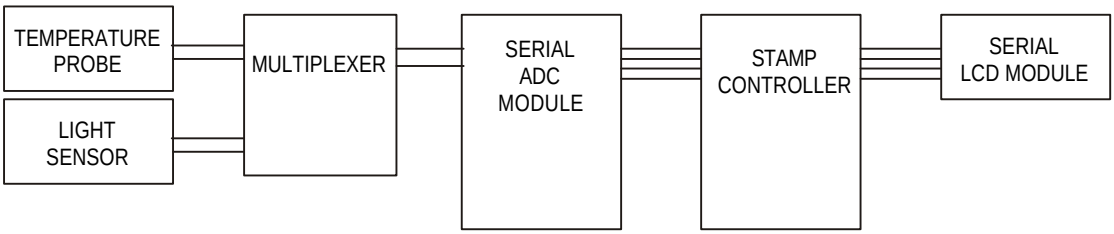
Multiplexers

ADC integrated circuits are relatively expensive to construct. When it is necessary to record more than one analogue signal a 'multiplexer' can be used to prevent the need for multiple ADC circuits. The multiplexer can be physically 'built into' the ADC integrated circuit, or can be provided by a completely separate integrated circuit.



The multiplexer functions in a similar manner to a rotary switch connecting each of the analogue channels in turn to the ADC. Therefore the multiplexer 'selects' which sensor is connected to the ADC at any one time. As only one sensor is connected to the ADC at any one time the whole system is slower than using a separate ADC for each sensor. However modern microcontrollers operate very quickly, and so the cost saving usually outweighs the increased processing time.

Activity 1.4

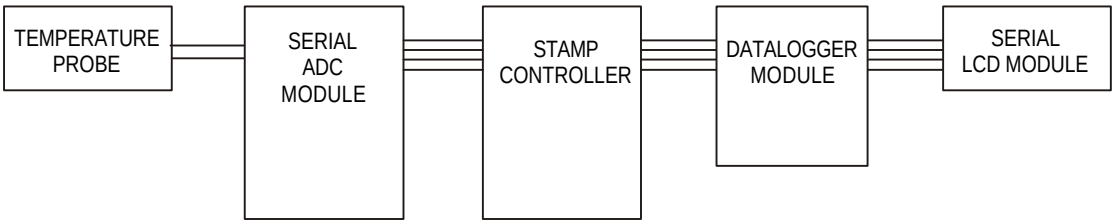


Build the circuit as shown. Note that a 2mm lead is used to connect the MFA Analogue Multiplexer to the Serial ADC Module (pin 3).

Key in, download and run the program listed below. This program reads each of the two sensors every second, and displays the reading on the LCD.

```
      pause 5                          'Short pause
      serout S_OUT,T2400,(254,1)       'Clear LCD command
      pause 30                         'Short pause

main: low ADC_MPX                      'Select sensor 1
      gosub adcread                    'Get the ADC reading

      serout 7,T2400,(254,128)         'Print on LCD line 1
      serout 7,T2400,("Sensor 1= ",#data,"  ")
      high ADC_MPX                     'Select sensor 2
      gosub adcread                    'Get the ADC reading

      serout 7,T2400,(254,192)         'Print on LCD line 2
      serout 7,T2400,("Sensor 2= ",#data,"  ")

      pause 1000                       'Wait 1 second
      goto main                        'Loop
```

Assignment 1.5



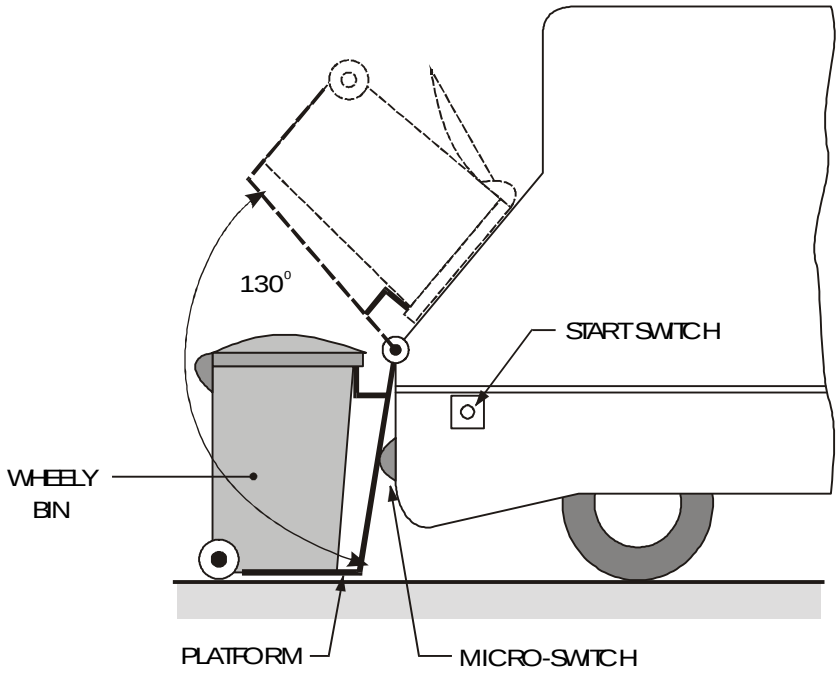
Build the circuit as shown in the diagram above. (The datalogging module is optional at this point)

a) Write a high level program in PBASIC that will continually monitor and display the temperature of the water on the LCD.

b) The temperature of the water should not be allowed to drop below 45 degrees Celsius. Modify your PBASIC program so that the immersion heater (connected to pin3) is automatically switched on when the water cools below the critical temperature.

Assignment 1.6

A local council has issued standard size "wheely bins" to all households so that the dust carts can empty them automatically. The dustman simply clamps the wheely bin onto a platform and then presses a switch. the platform then rotates through 130 degrees as shown in the diagram. The platform stays in the emptying position for 5 seconds and then returns to it's original position, where it triggers a micro switch on the rear of the dustcart. Platform movement is controlled by a dc motor and gearbox.



The process is controlled by a microcontroller. the angle of rotation of the platform is sensed by a rotary potentiometer. The potentiometer is capable of rotating 360 degrees from the starting position, and rotation between 0 and 360 degrees sends a **proportional reading** of between 0 and 255 via an ADC module connected to the microcontroller.

Input Connection	Pin	Output Connection
Start switch	0	
	1	
	2	
Micro-switch	3	
	4	Motor clockwise (Raises bin)
	5	Motor anti-clockwise (Lowers bin)
	6	
	7	

(a) Calculate the analogue reading when the platform is 130 degrees from the original position.

(b) Draw up a flow chart which shows the control sequence for the emptying of the wheely bin.

(c) With reference to your flow chart, write a high level program in PBASIC to control sequence for the emptying of the wheely bin.

Assume use of a pre-written sub-procedure called 'adcread' which will return the analogue reading in a variable called 'data'.

Section 2 - Data-Logging

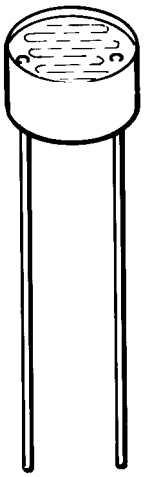
The purpose of this section is to introduce the concept of data-logging. The section also involves the use of a host computer for analysing data recorded with a spreadsheet application.

When you have completed this section you should be able to:
- understand the term data-logging
- understand the term sampling frequency
- appreciate the use of data-logging in a practical application
- understand how data can be analysed with a spread-sheet application
- write PBASIC programs to control data-logging experiments

Before you start this section you should have a basic understanding of:
- Analogue to Digital Conversion.
- The decimal, binary and hexadecimal number systems.

To complete the exercises in this section you require:
- Stamp Controller
- Datalogging Module
- Serial ADC Module
- Serial LCD Module
- Serial Printer Module
- MFA Sensor Probes
- MFA Multiplexer Module

Section 2 - Data Logging



Microcontroller systems are ideal for **data logging** tasks. A data logger is a device which is capable of remotely capturing and recording information, often analogue signals such as temperature and light level (via an Analogue to Digital Converter).

Data loggers are used in a wide range of applications from flight recorders on aircraft to life support systems in incubators for premature babies.

The frequency at which signals are recorded is called the **sample frequency**. A high frequency rate would be used to record a very fast changing signal over a small period of time - for instance when measuring the electrical activity of the human heart. A low frequency rate would be used when a slow changing signal is measured over a long period of time - for instance monitoring the temperature of a green house over a week period.

As data loggers are generally small, portable and battery operated, the data is often stored electronically in EEPROM integrated circuits. This means the data is retained when the batteries are removed, but can also be updated easily when required. Data loggers can be left unattended for long periods of time, and are ideal for hostile or remote environments measuring variations in industrial and weather monitoring situations.

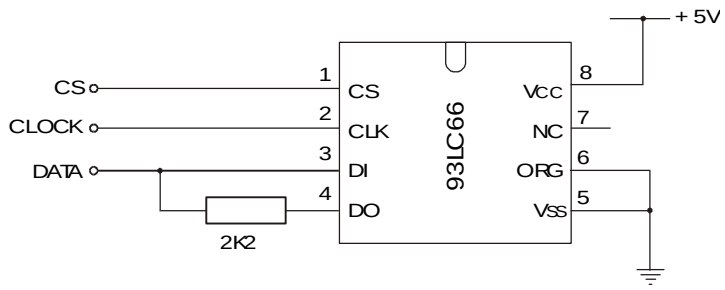
Once a data logging experiment is complete, it is obviously necessary to be able to retrieve the data for analysis. In more basic systems the data may just be printed out directly as a simple table, but in more advanced systems it is common to 'upload' the data to a personal computer for mathematical analysis and/or graph generation in a spreadsheet application.

Assignment 2.1

Suggest a suitable **sampling frequency** for each of the following monitoring systems. In each case clearly explain why your sample frequency is appropriate.

a) Monitoring the temperature in a greenhouse.
b) Monitoring the speed of an aeroplane.
c) Monitoring the heart rate of a premature baby in an incubator.
d) Monitoring the temperature in a restaurant cold store.

The Data Logger Module



The data logger module contains a **512 byte EEPROM** integrated circuit. In practical terms this means that up to 512 8-bit ADC readings can be stored in the EEPROM.

The data logger module also contains the electronic connectors necessary to link a serial computer cable, so that the data stored in the EEPROM can be uploaded to a personal computer for analysis.

The EEPROM integrated circuit is a serial device, and operates in a very similar manner to the serial ADC, using the same three data, clock and chip select signals. However in this case the data signal is bi-directional, as it is necessary to write to, and read from, the EEPROM memory.

The 512 byte memory is divided into two pages, each page containing 256 bytes. Therefore to correctly locate a memory cell you must specify both its **page** number (0 or 1) and its **address** (0 to 255).

Like the ADC, the EEPROM makes use of two 'standard' sub-procedures to carry out the serial communication. These sub-procedures can be copied directly from the datasheets. Sub-procedure '**eewrite**' is used to write data to the EEPROM, and '**eeread**' is used to read data back from the EEPROM.

IT IS NOT NECESSARY TO LEARN THESE 'SUB-PROCEDURES'. THESE SUB-PROCEDURES WILL ALWAYS BE PROVIDED, IF NECESSARY, IN EXAMINATIONS.

WHEN CREATING NEW PBASIC PROGRAMS THE TEMPLATE FILE 'TEMPLATE.BAS' SHOULD BE USED. THIS TEMPLATE FILE CONTAINS ALL THE STANDARD SUB-PROCEDURES REQUIRED, SO THAT IT IS ONLY NECESSARY TO KEY IN THE MAIN PROGRAM.

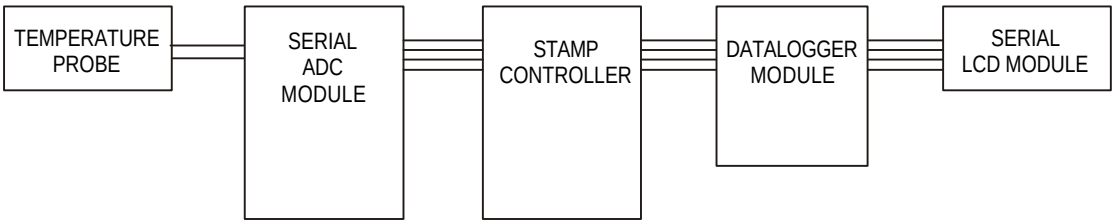
Therefore to write the value '85' to EEPROM address '23' on page '1' the program would be

```
let data = 85           'Set the data.
let address = 23        'Set the address.
let page = 1            'Set the page.
gosub eewrite           'Write the data to the EEPROM.
```

To retrieve the data and show it on the computer screen (via the debug command) the program would be

```
let address = 23        'Set the address.
let page = 1            'Set the page.
gosub eeread            'Read the data from the EEPROM.
debug data              'Send data to the computer screen.
```

Activity 2.2



Build the circuit as shown.

Key in, download and run the program listed below. This program records thirty readings of the analogue sensor. For ease of understanding the readings are also shown simultaneously on the LCD.

```
main: for b10 = 0 to 30             ' Start a for...next loop
         let page = 0               ' set EEPROM page
         let address = b10          ' set EEPROM address
         gosub adcread              ' get analogue reading
         gosub eewrite              ' write to EEPROM

         serout 7,T2400,(254,128)   ' Line 1
         serout 7,T2400,("Address = ",#address,"  ")
         serout 7,T2400,(254,192)   ' Line 2
         serout 7,T2400,("Data    = ",#data,"  ")

         pause 1000                 ' wait 1 second
      next b10                      ' next loop

      end
```

The next section describes a PBASIC program that can be used to retrieve the data, which is now stored in the EEPROM memory chip.

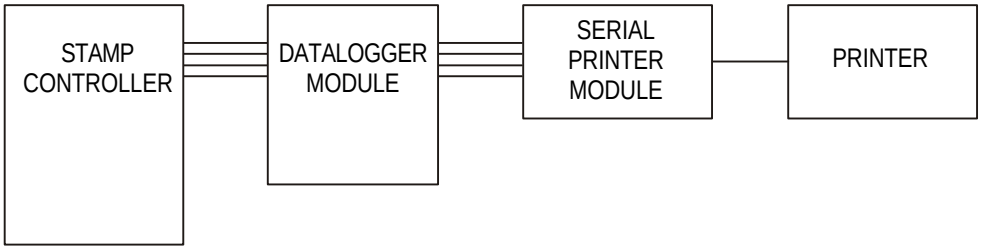
Activity 2.3

Key in, download and run the program listed below. This program retrieves the data from the EEPROM one address at a time. To get the next reading the digital input switch on the Serial ADC module should be briefly pressed.

```
main: for b10 = 0 to 30             ' Start a for...next loop
         let page = 0               ' set EEPROM page
         let address = b10          ' set EEPROM address
         gosub eeread               ' read data from EEPROM

         serout 7,T2400,(254,128)   ' line 1
         serout 7,T2400,("Address = ",#address,"  ")
         serout 7,T2400,(254,192)   ' line 2
         serout 7,T2400,("Data    = ",#data,"  ")

loop: if pin3 = 0 then loop         ' wait until switch is high

      next b10                      ' next loop
      end                           ' end
```

Activity 2.4

In most situations it will be necessary to tabulate the readings for analysis. The serial printer module allows the readings to be printed directly, without the need for a host computer.

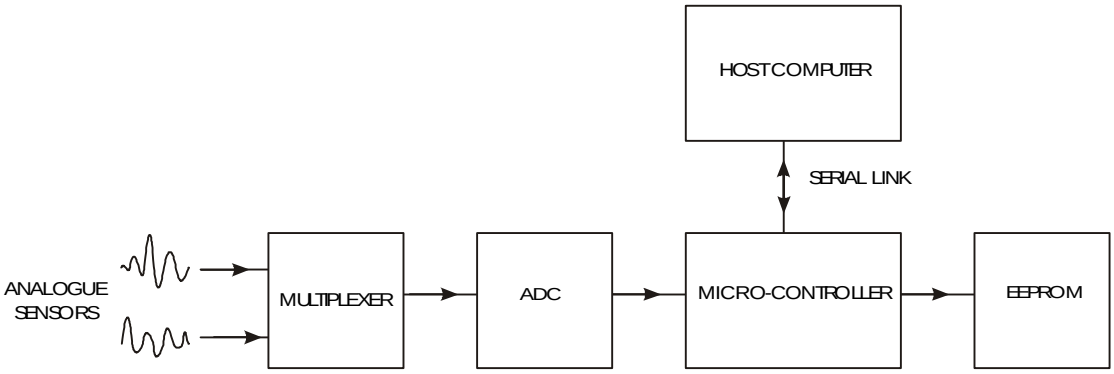


Build the circuit as shown.

Key in, download and run the program listed below. This program prints out the 30 bytes of information from the datalogging experiment.

```
      pause 5                         ' Short pause

start: serout 7,T2400,(STX)   ' Start printing command

      'Print titles
      serout 7,T2400,(TAB,"Address",TAB,TAB,"Sensor")

      serout 7,T2400,(CR,LF,LF)     'Next line.

main: for b10 = 0 to 30             'Start a for..next loop.
         let page = 0               'Set page
         let address = b10          'Set address
         gosub eeread               'Read data from EEPROM.
         serout 7,T2400,(TAB,#address,TAB,TAB,#data)
         serout 7,T2400,(CR,LF)     'Next line
      next b10                      'Next loop.

      serout 7,T2400,(FF)           'Print form-feed
      serout 7,T2400,(ETX)    'Send printer module to sleep
      end                           'end
```

Analysing Data with a Spread-sheet



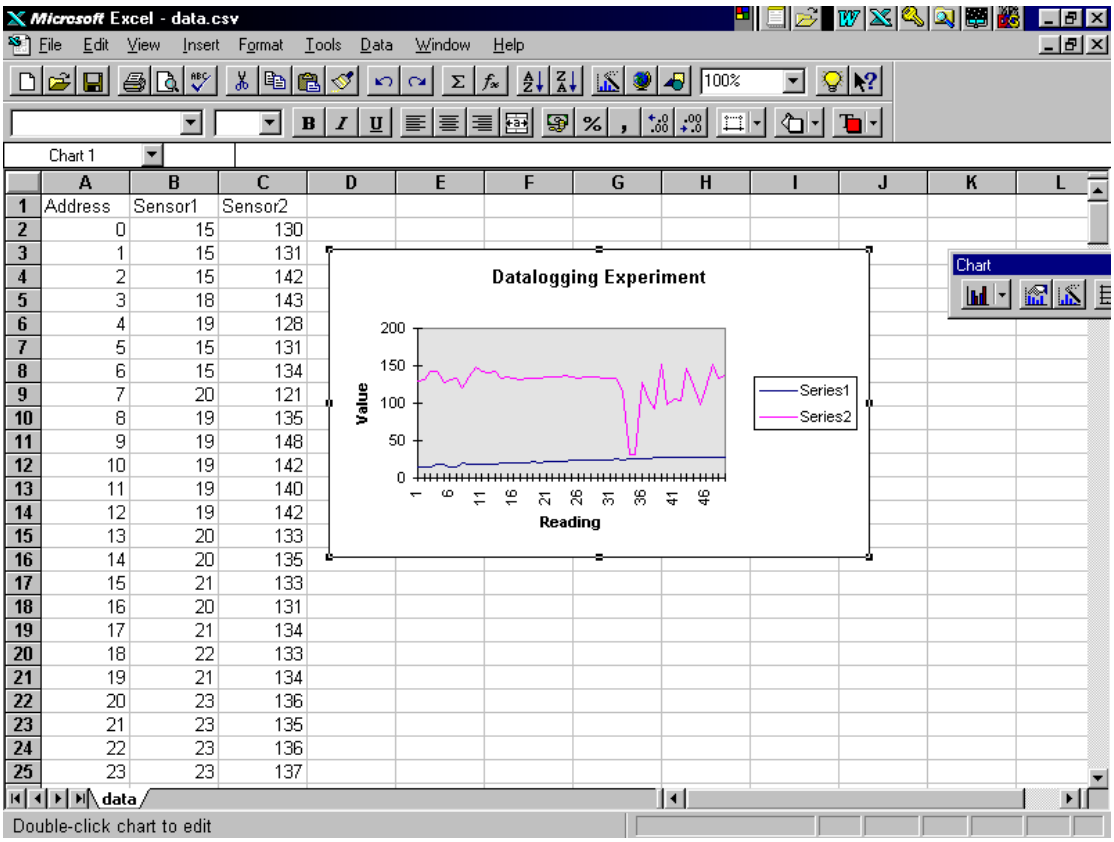
After a data-logging experiment it is usual to 'upload' the data to a host computer so that the readings can be analysed via a spread-sheet application. This enables mathematical calculations to be made, and also allows the generation of graphs.

The 'upload' procedure requires serial communication between the Stamp Controller and the host computer. This is achieved via a serial cable connected to the Datalogger Module.

The procedure is as follows:

1. Download the standard file ‘datalink.bas’ to the Stamp Controller.
2. Move the serial cable from the Stamp Controller to the Datalogger module.
3. Start up the ‘Datalink’ utility software on the computer.
4. Click ‘New’ with the ‘Datalink’ utility to start the upload.
5. Once the upload is complete. save the uploaded data into a CSV format data file.
6. Exit the ‘Datalink’ utility.
7. Start up the spreadsheet application.
8. Open the CSV format data file previously saved.
9. Analyse and draw graphs of the data using the standard spreadhseet functions.

Sample graph (generated by Microsoft Excel (tm)).



	A	B	C
1	Address	Sensor1	Sensor2
2	0	15	130
3	1	15	131
4	2	15	142
5	3	18	143
6	4	19	128
7	5	15	131
8	6	15	134
9	7	20	121
10	8	19	135
11	9	19	148
12	10	19	142
13	11	19	140
14	12	19	142
15	13	20	133
16	14	20	135
17	15	21	133
18	16	20	131
19	17	21	134
20	18	22	133
21	19	21	134
22	20	23	136
23	21	23	135
24	22	23	136
25	23	23	137

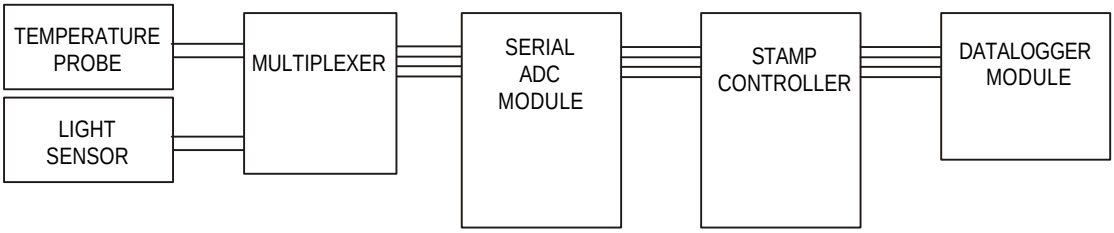
PBASIC 'DATALINK' FILE

Activity 2.5
Open the standard PBASIC file 'datalink.bas'. Follow the procedure detailed on page 24 to upload the data from the datalogger module.

Once the upload is complete, save the CSV file and then exit from the datalink utility. Start up the spread sheet application, import the CSV file, and generate a graph from the data.

Assignment 2.6

A datalogging system is to be used to record the temperature and light level in a greenhouse over a period of 1 day.



Build the apparatus as shown in the diagram above.

a) Explain why it is necessary to use a multiplexer in this situation.

b) Write a 'test' program in PBASIC that will record the temperature and light levels over a 5 minute period.

c) Use the datalink program to retrieve the data from the datalogger module. Using a spreadsheet draw a line graph to show the data recorded.

Assignment 2.7
A pet shop owner requires a data logging system that will allow him to monitor the temperature of one of the large display aquariums in the shop.

The datalogger should record the temperature every ten minutes over a 24 hour period. At the end of the period the owner wants to be able to print out a table of results from the datalogger without having to use a host computer to analyse the data.

(a) Draw a block diagram of the system.

(b) Draw up a flow chart which shows the control sequence for the datalogging experiment.

(c) With reference to your flow chart, write a high level program in PBASIC to perform the experiment.

Assignment 2.8

In a maternity hospital premature babies are cared for in incubators. The temperature inside each incubator is continuously monitored.

A microcontroller based datalogging system is used to monitor the temperature from four separate incubators by means of a 4-bit multiplexer.

a) With the aid of sketches, explain the function of a 4-bit multiplexer and suggest why it is useful in this application.

b) Suggest a suitable way of presenting the output from the datalogging system for use by the nursing staff. Explain your answer.