An Introduction to Microcontrollers and Software Design

Name ______________________

MRGS Technology Electronics

Available online from www.techideas.co.nz

B. Collis - Mount Roskill Grammar School 2003-2009

Table of Contents

Introduction to Microcontroller Electronics 4
Computers and Microcontrollers 5
What exactly is a Microcontroller 6
What you do when learning to program 7
Achievement Objectives from the NZ Curriculum 8
Hardware - The AVR Microcontroller 9
Power Supplies 9
AVR Programming 10
Breadboard 10
Simple AVR circuit 11
Circuit description 12
AVR programming cable 12
Introduction to writing programs using Bascom-AVR IDE 13
Reading and Writing using flowcharts 14
Input and Output Control 15
Sending Morse code 16
Microcontroller ports: write a Knightrider program using 8 LED’s 17
Multiple LEDs - Traffic lights exercise 18
Multiple LEDs - 7 Segment Displays 19
Different types of switches 20
First input device – a single push button switch 21
BASCOM and AVR Assignment 23
Words you need to be able to use correctly when talking about programming 24
A Bit about Numbers 25
Programming Codes of Practice 26
Programming Template 27
Variables 28
The BASCOM-AVR Simulator 29
Control statements – IF THEN 31
Connecting and programming multiple switches 31
Reading multiple switches in software 32
Using flowcharts to help solve problems 33
Using IF-THEN to control which parts of a program happen 37
Debounce 36
More Interfacing 38
Analogue to Digital Conversion 39
Reading an LDR’s values in Bascom 41
Temperature measurement using the LM35 43
Keypad Interfacing 44
Alternative keypad interface 44
Controlling high power loads (outputs) 45
Parallel Data Communications 46
LCDs (Liquid Crystal Displays) 47
Connecting an LCD to a 40pin AVR 48
Don’t delay 51
Programs as solutions: understanding algorithms and flowcharts 54
Planning Sequence for an AVR project 54
One Page Brief 55
One Page Brief 56
Algorithm Development Worksheet 57
Example Brief 58
Algorithm Planning Example 59
Example Brief 60

Algorithm Development Example 61
Glue Gun Timer Flowchart 62
Multiplication Algorithms 64
Algorithm exercises 68
LCD programming exercises 69
LCD programming exercises 69
Scrolling Message Assignment 70
Strings Assignment 71
ASCII Character Table 73
ASCII Assignment 74
Time 77
Sounding Off 81
System and Software Design 84
Understanding how simple systems work 84
Problem Decomposition Example 85
Statecharts 87
Token Game – Statechart Design Example 90
Serial Communications 95
RS 232 Serial Communications 97
Serial IO using Inkey() 102
Introduction to I2C 103
Real Time Clocks 105
DS1307 RTC 107
Arrays 112
Computer Programming detail 115
AVR Internals – how the microcontroller works 116
Interrupts 120
Polling versus interrupt driven architecture 122
Timer/Counters 123
PWM - Pulse Width Modulation 126
AVR Clock/Oscillator 129
Assignment – Maths In The Real World 130
Math Assignment - Part 1 133
Math Assignment - Part 2 134
Math Assignment - Part 3 135
Math Assignment - Part 4 136
Math Assignment - Part 5 137
Math Assignment - Part 6 138
Bascom Keyword Reference 139
AVR Development Boards we can use 141
AVR Development Board 2 144
ATMEGA Development Board 3 147
ATMEGA16/32 Microcontroller Pin Functions and Connections 149
ATMEGA16/32 40pin DIP package– Pin Connections 150

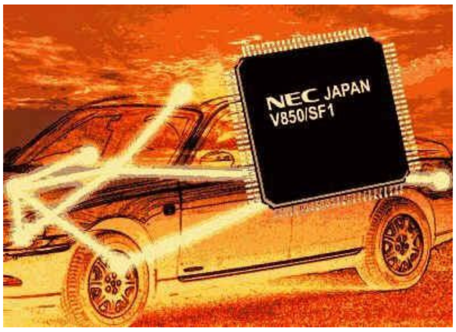
Introduction to Microcontroller Electronics

The course is an introductory course for students in design using microcontrollers; it covers both hardware interfacing and software design.

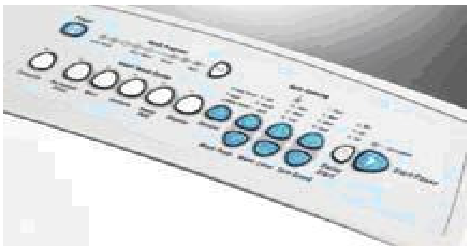
Microcontrollers are a common electronic building block used for many solutions to needs throughout industry, commerce and everyday life.

They are found inside aircraft instruments.

They are used extensively within cellular phones, modern cars,



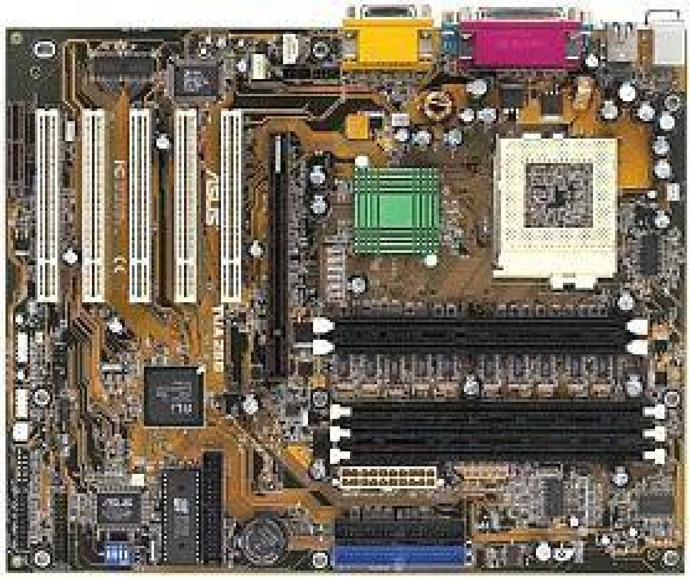
domestic appliances such as stereos and washing machines

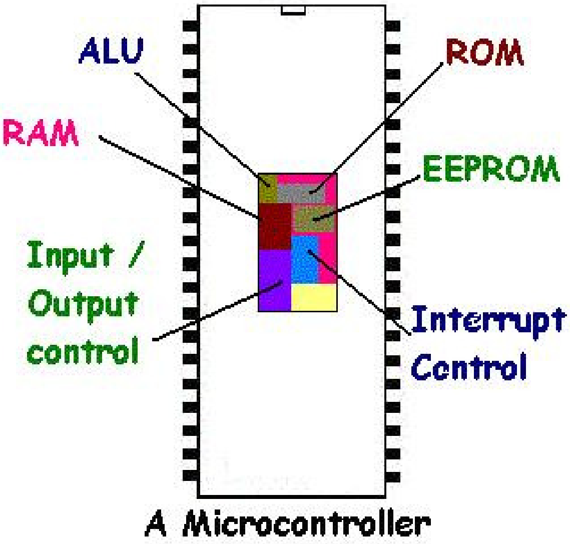


and in automated processes through out industry

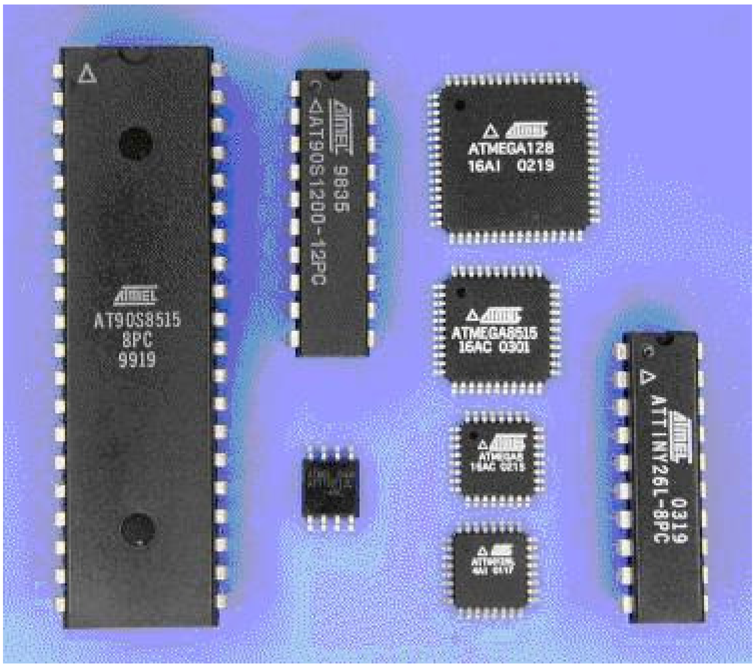
Computers and Microcontrollers

A microcontroller is very much everything that you would find inside a PC's case, but on a smaller scale. There is a processor, temporary memory for data (the RAM) and memory for programs (the ROM).





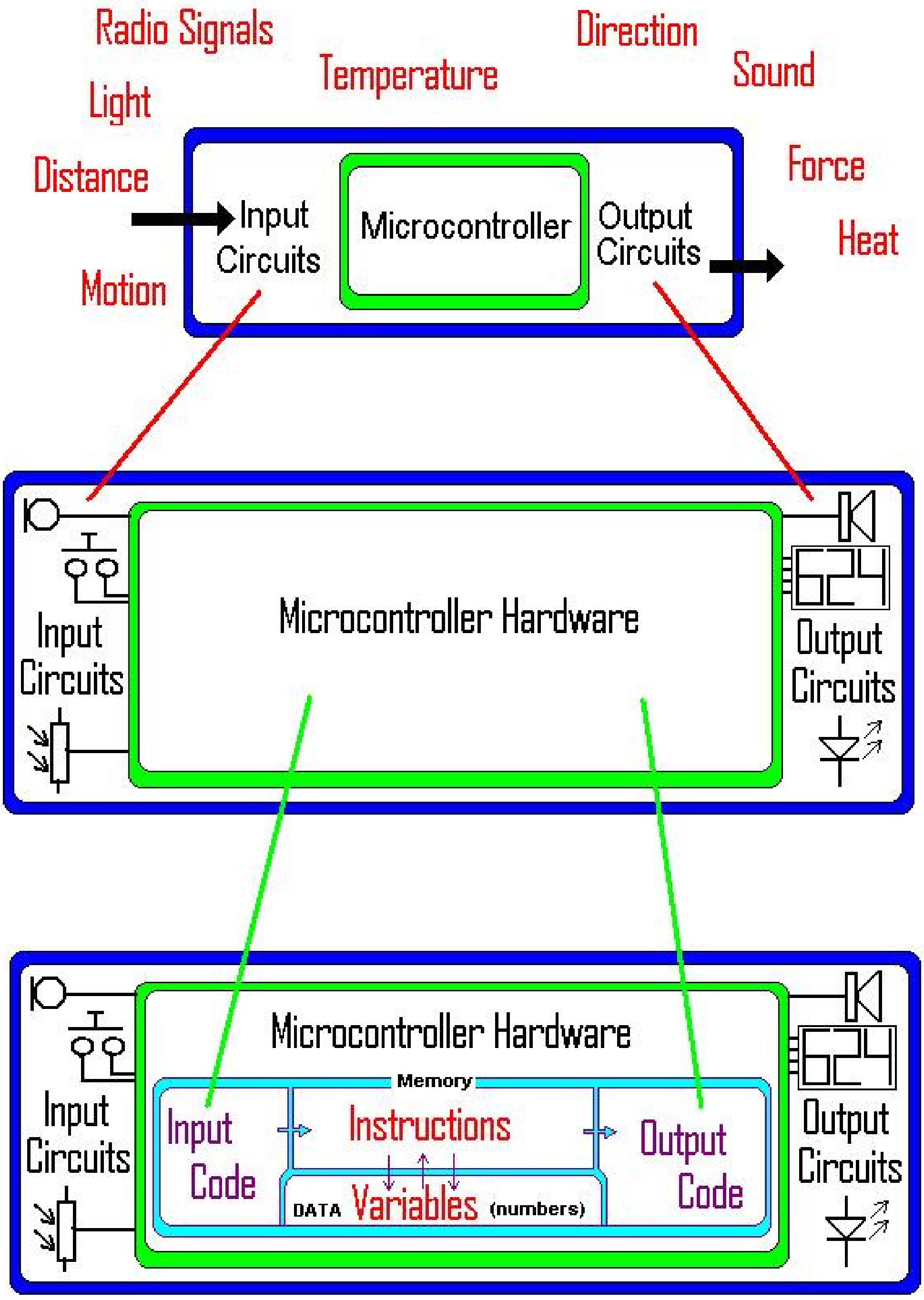
However don't think that because a microcontroller is smaller than a PC that it is the same comparison as between a real car and a toy car. The microcontroller is capable of carrying out millions of instructions every second. And there are billions of these controllers out there in the world doing just that. You will find them inside cars, stereos, calculators, remote controls, airplanes, radios, microwaves, washing machines, industrial equipment and so on.



What exactly is a Microcontroller

As with any electronic circuit the microcontroller circuit has three parts,
the INPUT, PROCESS AND CONTROL.

The input circuitry converts the real world into the electronic; the microcontroller processes the electronic signals; the output circuitry converts the electronic into the real world.



Inside the microcontroller there is however another level of conversion.

The micro has input code, output code and instructions (process code), as well as variables to store data.

The input code converts the electronic signals to data (numbers). The process code manipulates the data. The output code converts the data (numbers) to electronic signals. Variables are locations in memory that data is stored in.

So in a microcontroller circuit that creates light patterns based upon sounds the control process is
SOUND to ELECTRICITY to DATA
Processing of the DATA (numbers)
DATA to ELECTRICITY to LIGHT

What you do when learning to program

1. Get to know the hardware you are using
    a. Get a copy of the datasheet
    b. Learn about the power supply required
    c. Learn how to configure and connect to input and outputs
    d. Find out about the different types of memory and amount of each
    e. Find out about the speed of processing
2. Get to know the language and the IDE you are using
    a. Learn to access the helpfile (e.g. highlight a word and press F1)
    b. The language has syntax, specific grammar/word rules you must use correctly
    c. The IDE (Integrated Development Environment) has special commands and built in functions you must know and use: $crystal, $regfile, config, alias, const, port, pin
    d. Learn common I/O functions: set, reset, debounce, locate, LCD, GetADC
    e. Understand the limitations of and use variables: byte, word, long, single, double)
    f. Use constants instead of numbers in the code (e.g. waitms timedelay)
    g. Get to know the control functions: Do-Loop (Until), For-Next, While-Wend, If-Then (Else)
    h. Get to know about text and math functions (read help file, write a few simple programs using the simulator)
3. Develop Algorithms (written plans for the process the program must carry out)
    a. Have a goal in mind for the program – use specifications from the brief
    b. Plan your i/o by drawing a system block diagram
    c. Determine variables and constants required in the program
    d. Determine the state of all the I/O when the program begins
    e. Write the algorithm - Identify and describe the major processes the micro must do.
4. Draw Flowcharts or Statecharts (visual diagram for the process the program must carry out)
    a. Identify the blocks/states that will be used
    b. Use arrows to link the blocks and visualise control processes and program flow
5. Develop code from the flowcharts
    a. The outer looping line is replaced with a do-loop
    b. Backwards loops are replaced with do-loop do-loop-until, for-next, while-wend
    c. Forward loops are generally replaced with If-Then-EndIf
    d. Replace the blocks with actual commands
    e. Layout the code with correct indentations(tabs)
    f. Develop an understanding of subroutines and when to use them
    g. Experiment by purposely putting in errors and seeing their effects

This is not a step by step process as you get to know about one area you get to know about others at the same time. Depth of knowledge and understanding comes from LOTS OF EXPERIMENTATION!

Achievement Objectives from the NZ Curriculum

Technological Practice

- **Brief –** one page brief, with conceptual statement and specifications
- **Planning** – algorithms, flowcharts, pcb design, case design
- **Outcome Development** – functioning circuit, microcontroller program, PCB, case

Technological Knowledge

- **Technological Modelling** – flowcharts, statecharts,, bread-boards
- **Technological Products**
- **Technological Systems** - i/o/process model, programming

Nature of Technology

- **Characteristics of Technological Outcomes**
- **Characteristics of Technology –** microcontrollers as the basis for modern technologies

Key Competencies

- Thinking –algorithm design, flowchart development, debugging program, fault finding circuits
- Relating to others – work in pairs/groups,
- Using language symbols and texts – programming language syntax, reading schematics
- Managing self –use workshop equipment safely, use time wisely
- Participating and contributing

Technological Skill development

- **Breadboard circuits**
- **Program microcontrollers**
- **Accurately describe problem solving processes (algorithms),**
- **Logically plan software solutions using flowcharts and statechart diagrams**

Become methodical in solving and debugging problems,

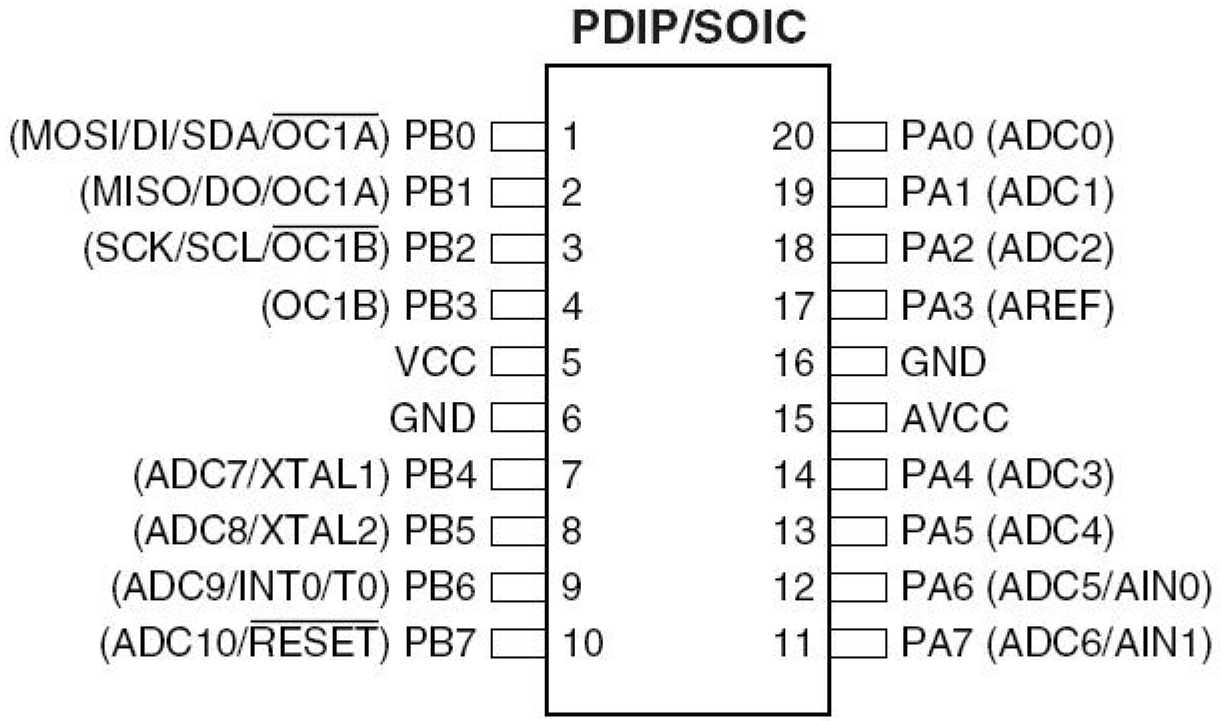
Hardware - The AVR Microcontroller

A microcontroller is a general purpose electronic circuit; it is a full computer inside a single integrated circuit (IC or chip). Normally with an IC like the TDA2822M amplifier or LM386 opamp its function and its pins are fixed, you have no control over what they do, and therefore limited control over how to connect them.
With a microcontroller however you are in control, you decide:

- what the function of the IC is
- what most of the pins are used for (inputs or outputs)
- and what external input/output devices these pins are connected to.

If you want an egg timer, a car alarm, an infrared remote control or whatever, it can all be done with a microcontroller.

A commercial range of microcontrollers called ‘AVR’ is available from ATMEL (www.atmel.com) We will start by using the ATTINY26, it has 2kbytes of Flash for program storage, 128 bytes of Ram and 128 bytes of EEPROM for long term data storage



Of the 20 pins:

- VCC(5) & GND(6,16) are dedicated for power, VCC is positive voltage, e.g 4.5V
- AVCC (15) is a special voltage for measuring analogue voltages (connect to VCC/5V).
- There are two I/O ports accessible portA and portB (larger AVR microcontrollers have more ports) A port is a group of 8 I/O pins which can be controlled together
- PB0(1), PB1(2), PB2(3), PB7(10) are pins used to upload the programs.
  (you cannot use PB7 as an I/O pin, but PB0,PB1,PB2 can be used with care)

Power Supplies

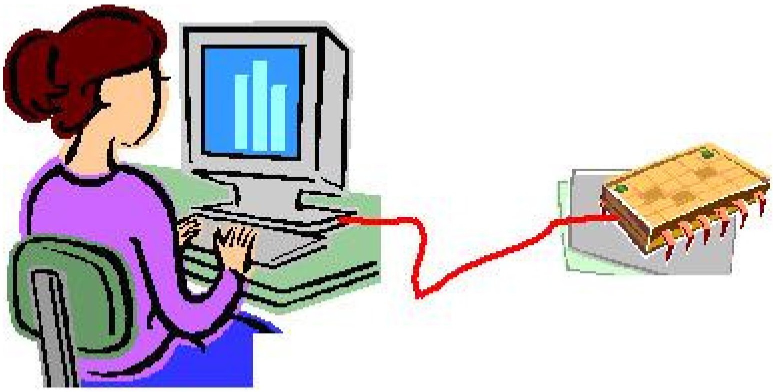
Most microcontrollers work off low voltages from 4.5V to 5.5V, so it can be run off batteries or a dc power pack, voltages in excess of these will destroy the micro. An L in an AVR model number means it can run at an even lower voltage. Some only run at 1.8V, so check the datasheet!

AVR Programming

Microcontrollers, such as the AVR, are controlled by software and they do nothing until they have a program inside them.

The AVR programs are written on a PC using the BASCOM-AVR.
This software is a type of computer program called a compiler, it comes from www.mcselec.com. It is freeware so students may download it and use it freely at home.

The AVR is connected to the PC with a 5 wire cable.

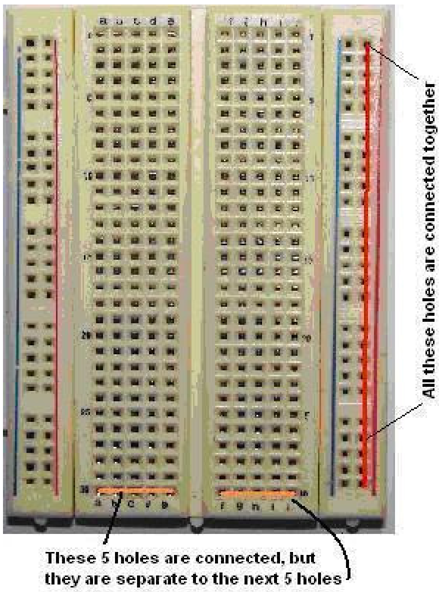


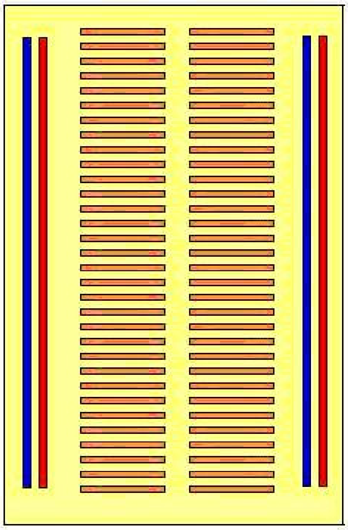
Breadboard

Often in electronics some experimentation is required to prototype (trial) specific circuits. A prototype circuit is needed before a PCB is designed for the final circuit.

A breadboard is used to prototype the circuit. It has holes into which components can be inserted and has electrical connections between the holes as per the diagram below.

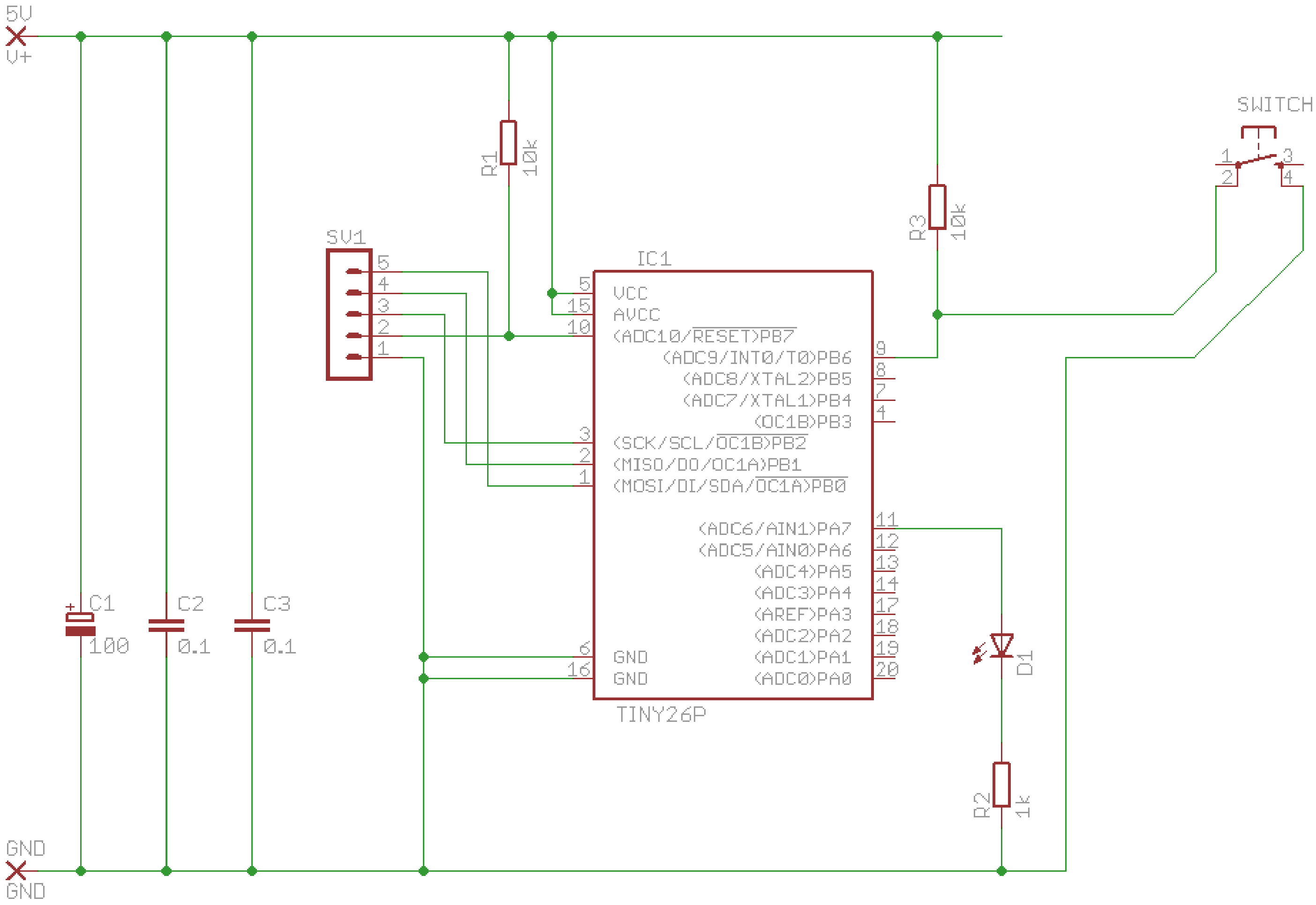
Using a breadboard means no soldering and a circuit can be constructed quickly and modified easily before a final solution is decided upon.


All these holes are connected together
These 5 holes are connected, but they are separate to the next 5 holes

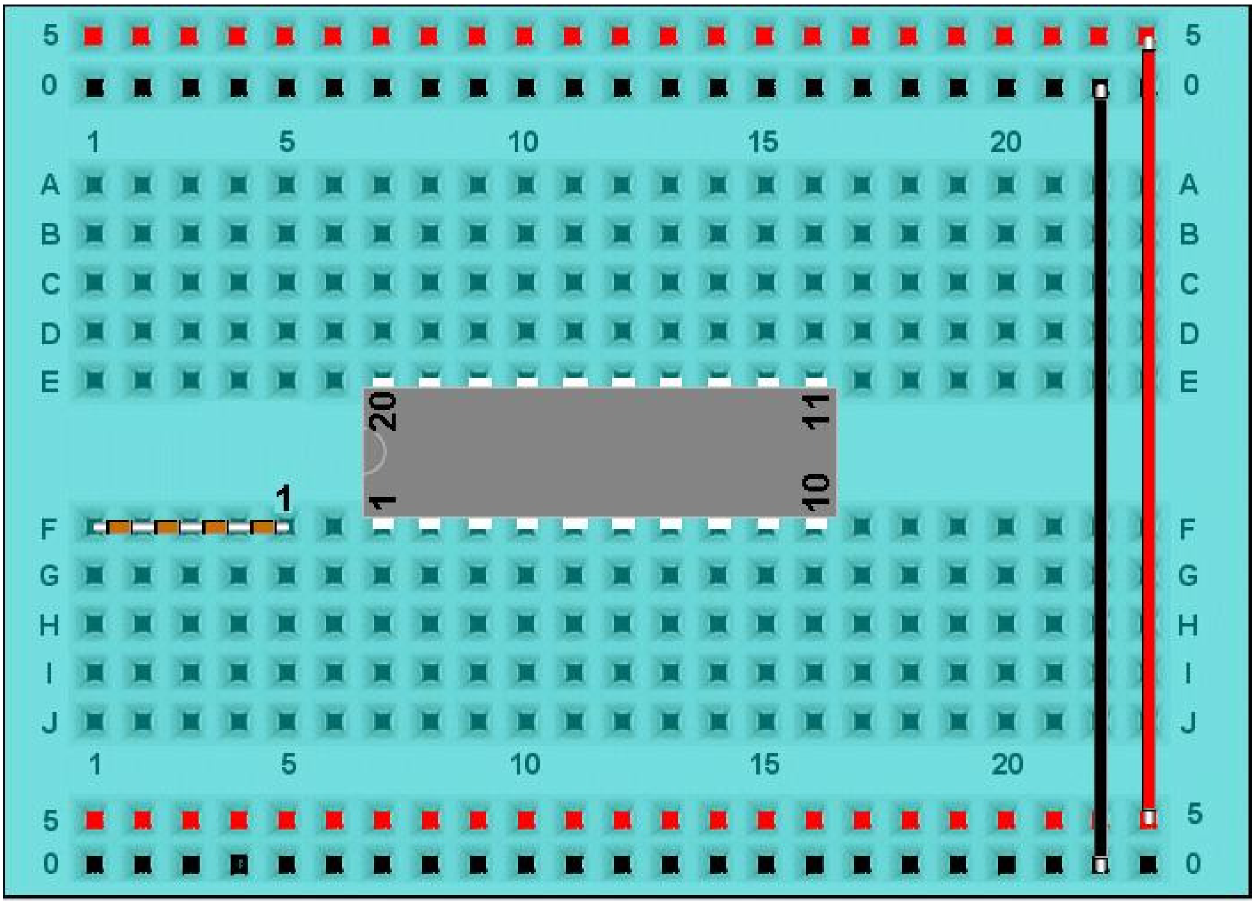


Here are all of the connections on the board

Simple AVR circuit



Design the above circuit onto the breadboard diagram below

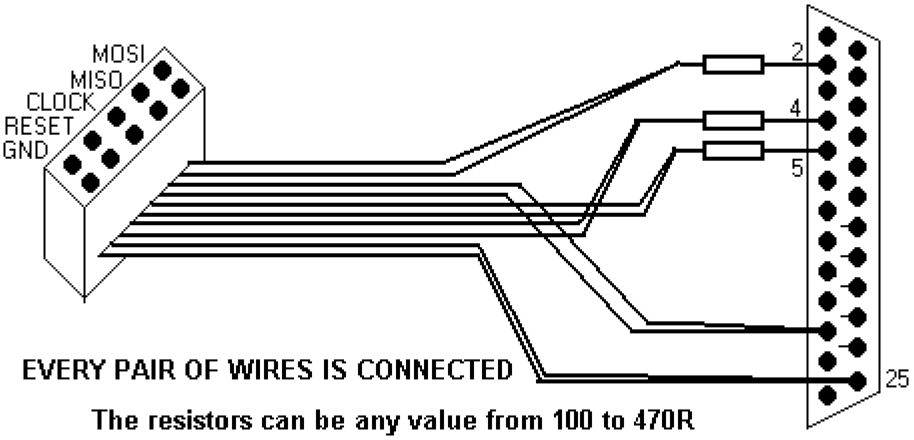


Circuit description

- The 5 pin connector is for programming.
- The 100uF electrolytic capacitor is to reduce any variations in power supply voltage.
- The 10k is a pull-up resistor for the reset pin, a low on this pin will halt the microcontroller and when it is held high the program will run from the beginning.
- The 1N4148 is a protection diode that will stop high voltages possibly damaging the microcontroller (it is only required on the reset pin because all the other microcontroller pins have built in diodes).
- There is an LED with a 1k ‘current limit’ resistor. An LED needs only 2V to operate so if connected without a resistor in series too much current would flow and destroy the LED. With 2V across the LED, there will be 3V across the resistor, and the current will be limited to (V/R) 3/1000 = 3mA.
- The 0.1uF capacitors are to stop electrical noise possibly interfering with the microcontrollers operation.

AVR programming cable

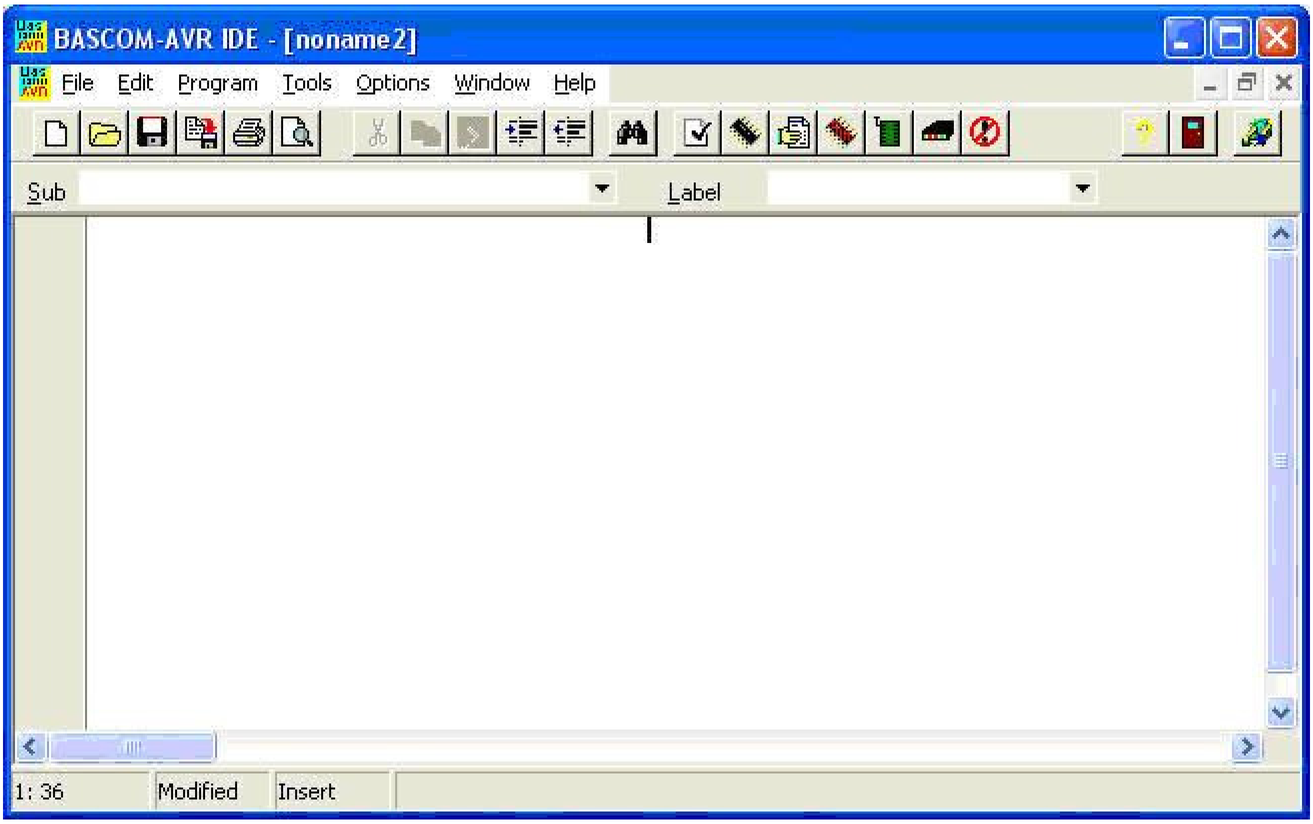
A five wire cable is needed to connect the AVR circuit to a PC.
It connects the PC’s parallel port to the AVR circuit. One end has a DB25M connector on it (as in this picture)


MOSI
MISO
CLOCK
RESET
GND
2
4
5
25
EVERY PAIR OF WIRES IS CONNECTED
The resistors can be any value from 100 to 470R

The other end has a 10 pin connector attached to it (as in this picture)
The 10 wires are arranged in 5 pairs
Put heatshrink over the connections to protect them.

Writing programs using Bascom-AVR IDE

BASCOM-AVR is **four programs in one package**, it is known as an IDE (integrated development environment); it includes the Program Editor, the Compiler, the Programmer and the Simulator all together. A free version is available online.


BASCOM-AVR IDE - [noname2]
File Edit Program Tools Options Window Help
Sub
Label
1: 36
Modified
Insert

After installing the program there are some set-up options that need changing.

From the menu select.
OPTIONS – **PROGRAMMER** and select **Sample Electronics programmer**. Choose the parallel tab and select LPT-address of 378 for LPT1 (if you only have 1 parallel port on the computer choose this), also select **autoflash**.
The following are not absolutely necessary but will help you get better printouts.
OPTIONS – PRINTER change the margins to 15.00 10.00 10.00 10.00
OPTIONS – ENVIRONMENT – EDITOR change the Comment Position to 040.

The Compiler

The command to start the compiler is F7 or the black IC picture in the toolbar.
This will change your high-level BASIC program into low-level machine code.
If your program is in error then a compilation will not complete and an error box will appear. Double click on the error to get to the line which has the problem.

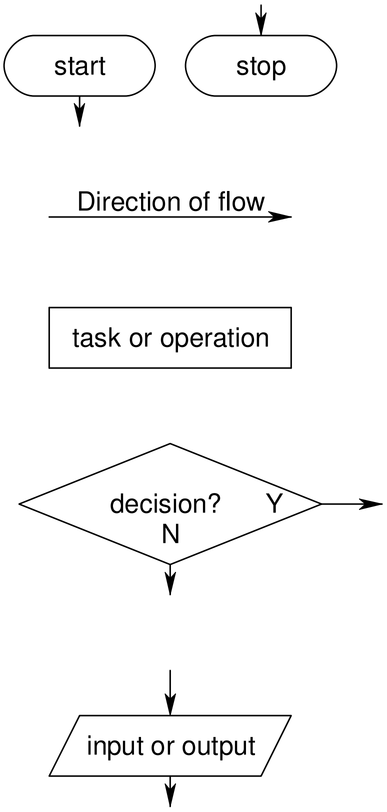
The Programmer

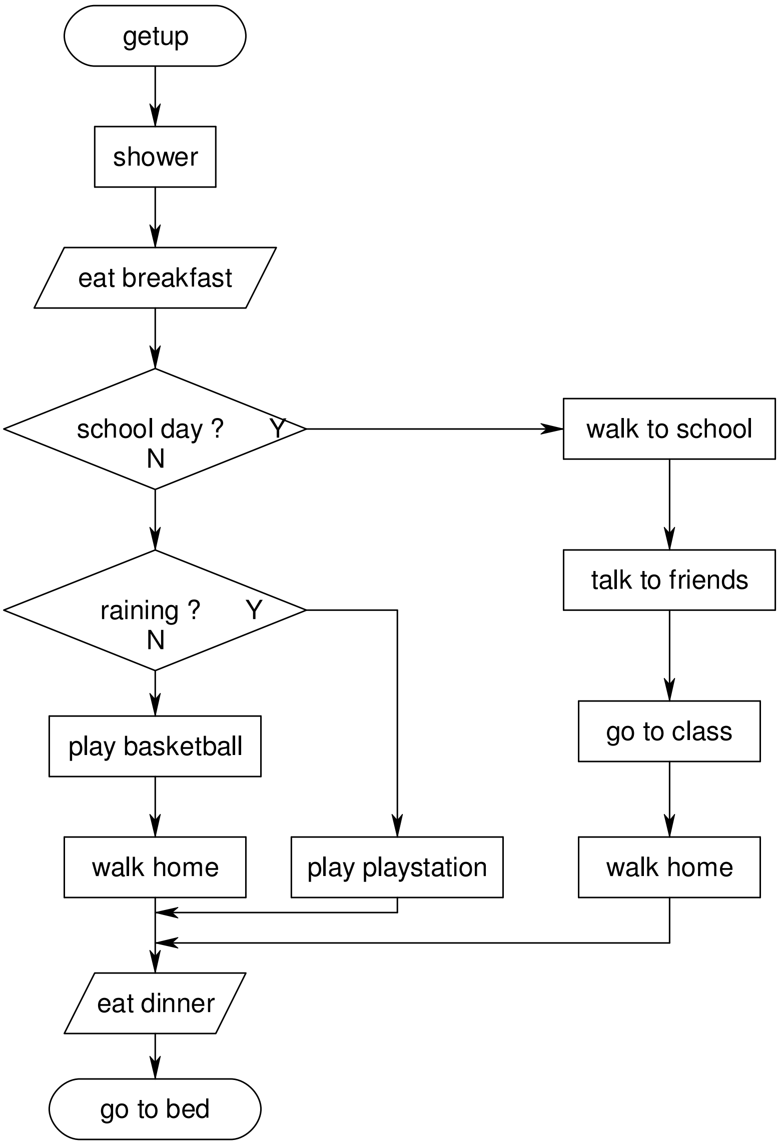
When you have successfully compiled a program pressing F4 or the green IC picture in the toolbar starts the programmer. If no microcontroller is connected an error will pop up. If the IC s connected then the BASCOM completes the programming process and automatically resets your microcontroller to start execution of your program.

Reading and Writing using flowcharts

Daily routine flowchart

Flowchart symbols

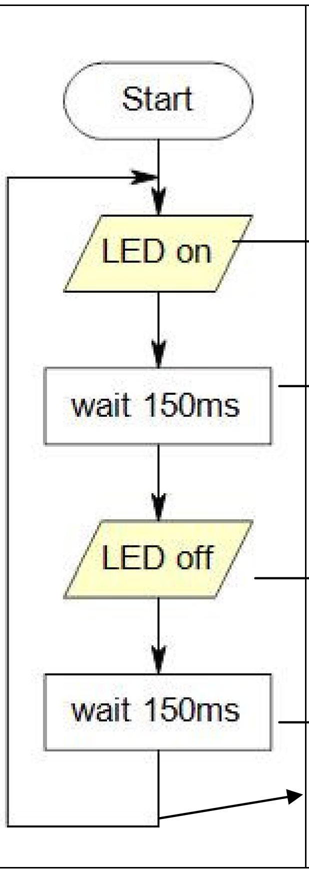




Input and Output Control

Learning Intentions: Learning to link parts of a flowchart with actual program code

Flash1LEDv1.bas
“Flash an LED rapidly on and off”



```
‘ Flash1LEDv1.bas
$regfile = "attiny26.dat”      ‘ bascom needs to know our micro
$crystal = 1000000            ‘ bascom needs to know how fast it is going
Config Porta = Output          ‘make these micro pins outputs
Const Flashdelay = 150         ‘ preset how long a wait will be
Do                            ‘start of a loop
   Porta = &B10000000          ‘ LED 7 on

   Waitms Flashdelay          ‘wait a preset time

   Porta = 0                   ‘all LEDs off

   Waitms Flashdelay          ‘wait a preset time

Loop                          ‘return to do and start again
End
```

This is a typical first program to test your hardware
Every line of the code is important.
$regfile=”attiny26.dat”, Bascom needs to know which micro is being used as each micro has different features
$crystal=1000000, Bascom needs to know the speed at which our microcontroller is setup internally so that it can calculate delays such as waitms properly (1 million operations per second)
Config porta=output, each I/O must be configured to be either an input or output; it cannot be both at once.
Const Flashdelay=150, ‘constants’ are used in a program, it is easier to remember names and it is useful to keep them all together in one place in the program (this is a code of practice).
DO and **LOOP** statements enclose code which is to repeat; when programming it is important to indent (tab) code within loops; this makes your code easier to follow (this is a code of practice).
Porta = &B10000000 make porta.7 high (which will turn on the LED connected to that port) and make all the other 7 output pins on that port low
Porta = 0 make all 8 pins on porta low (which will turn off any LEDs connected to that port)
Waitms flashdelay wait a bit, a microcontroller carries out operations sequentially, so if there is no pause between turning an LED on and turning it off the led will not be seen flashing

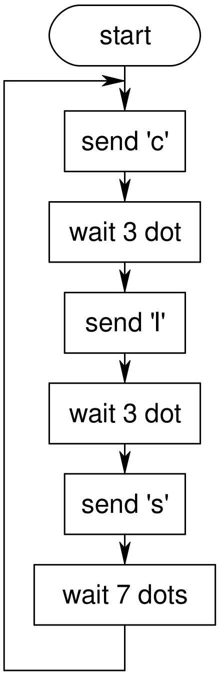
Playing around develop your understanding, carry out AT LEAST these to see what happens
What happens if Const Flashdelay is changed to 1500, 15, 15000
What happens if $crystal = 10,000,000 or 100,000 instead of 1,000,000
What happens if your change the $regfile to "attiny13.dat”
What happens if one of the waitms flashdelay statements is deleted
What happens when the two waitms flashdelay statements are deleted
Change porta=&B10000000 to set porta.7

Sending Morse code

Write a program to send your name in Morse code

- A dash is equal to three dots
- The space between the parts of the same letter is equal to one dot
- He space between letters is equal to three dots
- The space between two words is equal to seven dots

A	● ▬	U	● ● ▬
B	▬ ● ● ●	V	● ● ● ▬
C	▬ ● ▬ ●	W	● ▬ ▬
D	▬ ● ●	X	▬ ● ● ▬
E	●	Y	▬ ● ▬ ▬
F	● ● ▬ ●	Z	▬ ▬ ● ●
G	▬ ▬ ●		
H	● ● ● ●		
I	● ●		
J	● ▬ ▬ ▬		
K	▬ ● ▬	1	● ▬ ▬ ▬ ▬
L	● ▬ ● ●	2	● ● ▬ ▬ ▬
M	▬ ▬	3	● ● ● ▬ ▬
N	▬ ●	4	● ● ● ● ▬
O	▬ ▬ ▬	5	● ● ● ● ●
P	● ▬ ▬ ●	6	▬ ● ● ● ●
Q	▬ ▬ ● ▬	7	▬ ▬ ● ● ●
R	● ▬ ●	8	▬ ▬ ▬ ● ●
S	● ● ●	9	▬ ▬ ▬ ▬ ●
T	▬	0	▬ ▬ ▬ ▬ ▬



```
‘ MorseMeV1.bas
$regfile = "attiny26.dat”      ‘ bascom needs to know our micro
$crystal = 1000000             ‘ bascom needs to know how fast it is going
Config Porta = Output          ‘ make these micro pins outputs
Morseled alias porta.7         ‘ the name morseled is easy to remember
Const dotdelay = 150            ‘ preset how long a dot is
Do                              ‘start of a loop
        ‘letter c
        Set morseled           ‘ on
        Waitms dotdelay        ‘ wait 1 dot time
        Waitms dotdelay        ‘ wait 1 dot time
        Waitms dotdelay        ‘ wait 1 dot time
        Reset morseled         ‘ off
        Waitms dotdelay        ‘ wait 1 dot time
        …
        ‘letter l
         …
         …
Loop                            ‘return to do and start again
```

Microcontroller ports: write a Knightrider program using 8 LED’s

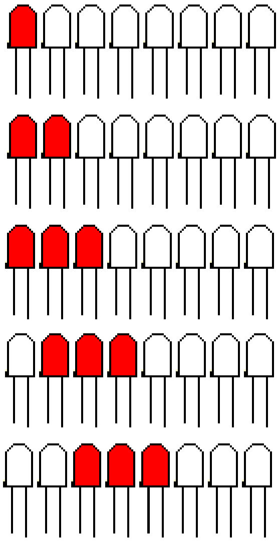
Ports are groups of 8 I/O pins.

Connect another 7 LEDs (**each needs an individual 1k current limit resistor**) to your microcontroller and write a program to flash all 8 LEDs in a repeating sequence e.g. 'led1, 2, 3, 4, 5, 6, 7, 8. 7, 6, 5, 4, 3, 2, 1, 2, 3...

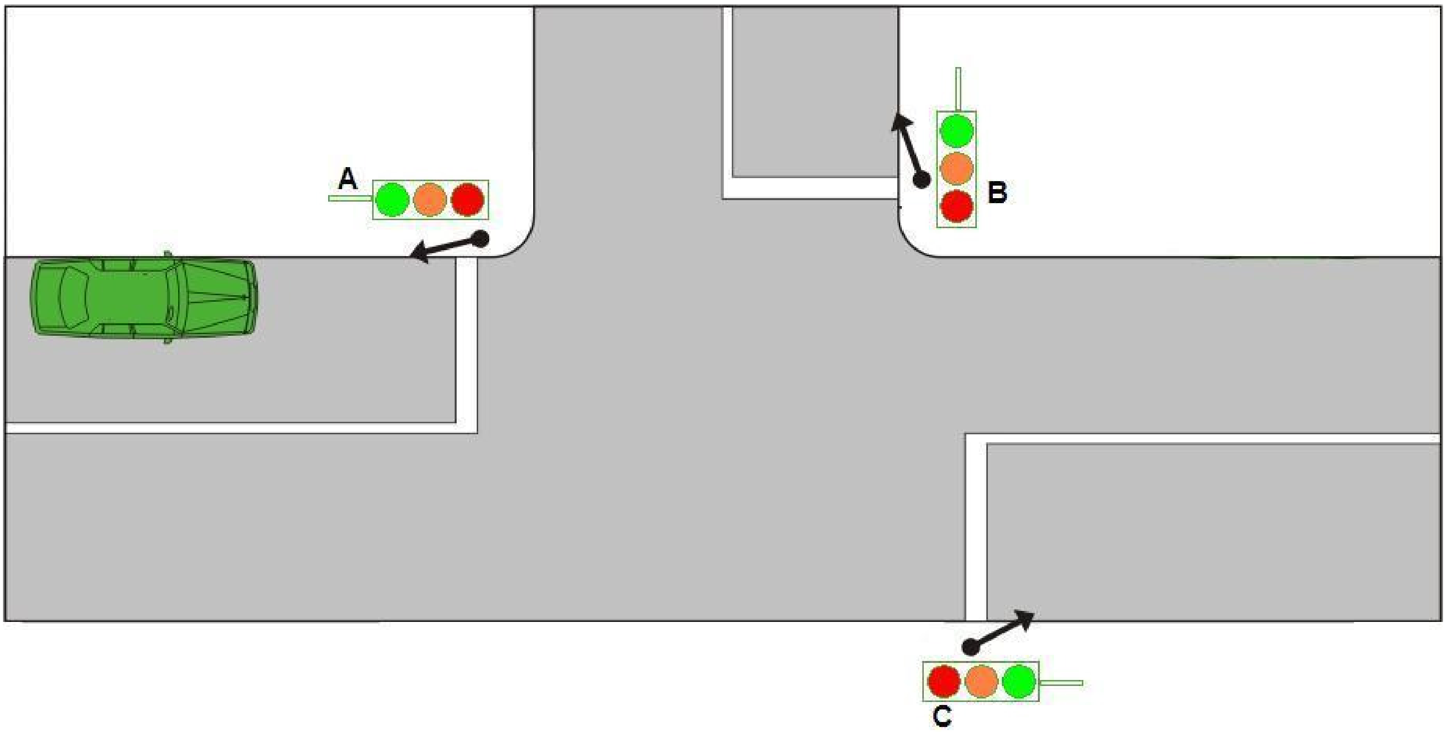
Use the following code to get started

```
Porta=&B10000000
Pause flashdelay
Porta=&B01000000
Pause flashdelay
Porta=&B00100000
```

As a second exercise rewrite the program so that three Leds turn on at once
Sequence = LED1, LED12, LED123, LED234, LED345, LED456, LED567, LED678, LED78, LED8, LED78, LED678…



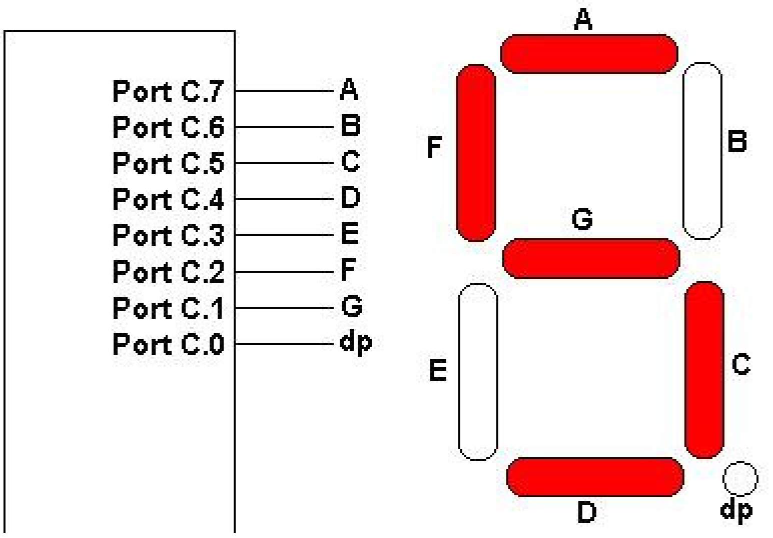
Multiple LEDs - Traffic lights exercise



Connect 3 sets of LEDs to the microcontroller; each set has 1red, 1 orange and 1 green LED.
Write a program that sequences the LEDs in the order A,B,C,A…
Fill in the sequence table below to start with (make it easier by only showing changes)

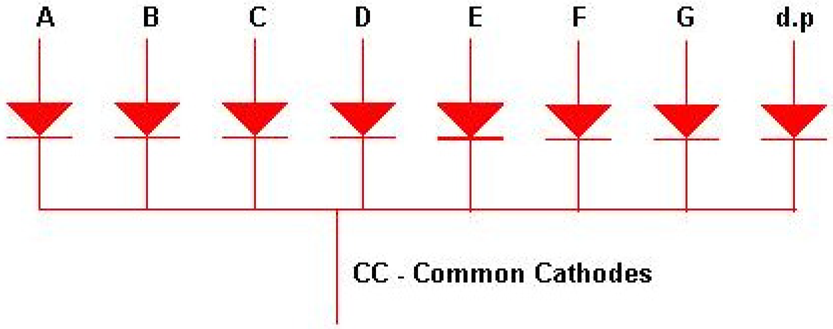
	1	2	3	4	5	6	7	8	9							
A - Red	Off		On	Off												
A - Or	Off	On	Off													
A - Grn	On	Off														
B - Red	On															
B - Or	Off															
B - Grn	Off			On												
C - Red	On															
C - Or	Off															
C - Grn	Off															
Delay to next step	1m	30s	2s	1m												

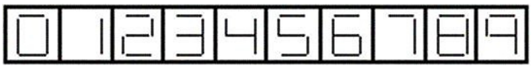
Multiple LEDs - 7 Segment Displays



Each bar in the seven segment display is a single LED.
This schematic for a single display shows how all the cathodes are connected together (some displays are CC common cathode and some are CA common anode

In this diagram a seven segment LED display is shown connected to 8 pins of a port. To display the number five, segments a,b,d,f &g must be on. and the code &B01001001 must be written out the port. Calculate the other values required to show all the digits on the display and determine their corresponding values in Hex and Decimal.





Display	Segments ON	Segments OFF	PORT Binary	Port Hex	Port Decimal
0					
1					
2					
3					
4					
5	a,c,d,f,g	b,e	&B01001001	&H49	73
6					
7					
8					
9					
A					
b					
c					
d					
E					
F					

Different types of switches

Various types of switches can be connected to microcontrollers for various purposes:

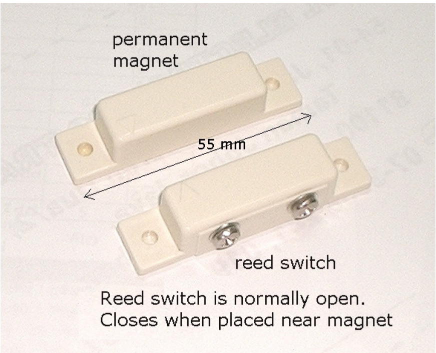
Key switches

So that only authorised people can operate a device

Micro switches

Used inside moving machinery

Magnetic or Reed switch



Reed switch is normally open.
Closes when placed near magnet

Useful for parts that open and close

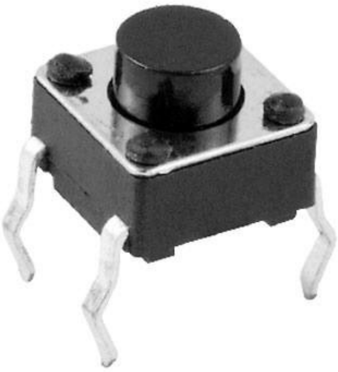
Tilt or Mercury Switch

Useful to sense movement or something falling over

Rotary Switch

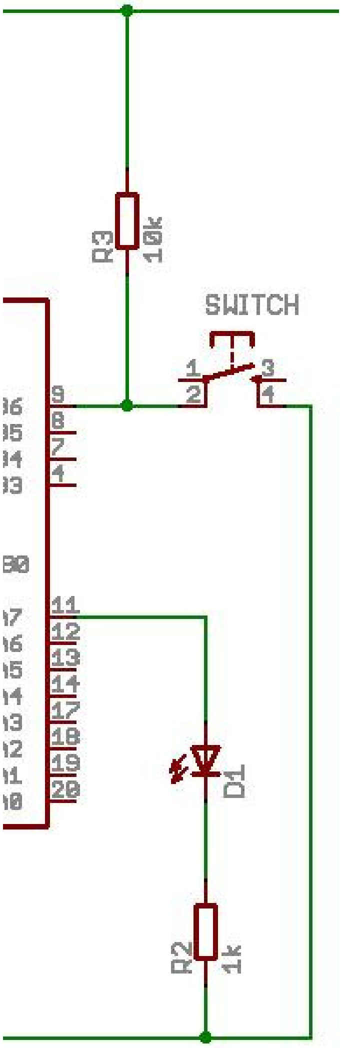
Can be used to set various positions

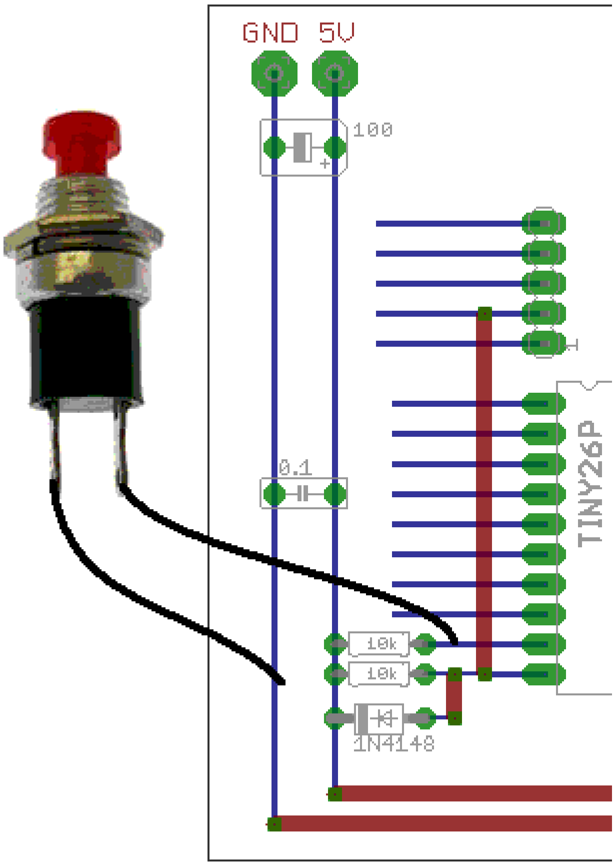
Tact switch



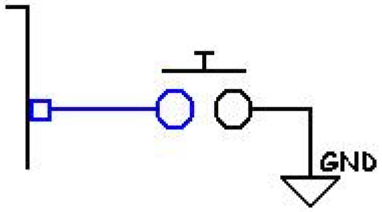
First input device – a single push button switch

Learning Intentions: Continue to learn to link plans for a program (e.g. flowchart) with actual program, develop skills in programming simple switch input connected to a microcontroller.

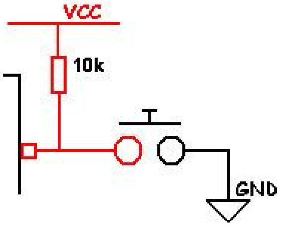




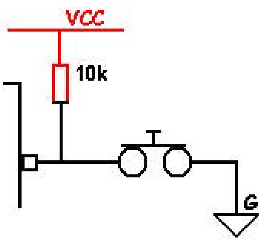
A 'pullup' resistor is essential in this circuit, as when the switch is not pressed it connects the input pin to a known voltage, if the resistor was not there then the input pin would be 'floating' and give unreliable readings



In this circuit the switch is connected without a pull-up resistor. The input pin of the microcontroller has no voltage source applied to it and is said to be 'floating'; the microcontroller input voltage will drift, sometimes be high (5V), sometimes low (0V) and is sensitive to touch and static leading to very unreliable results.



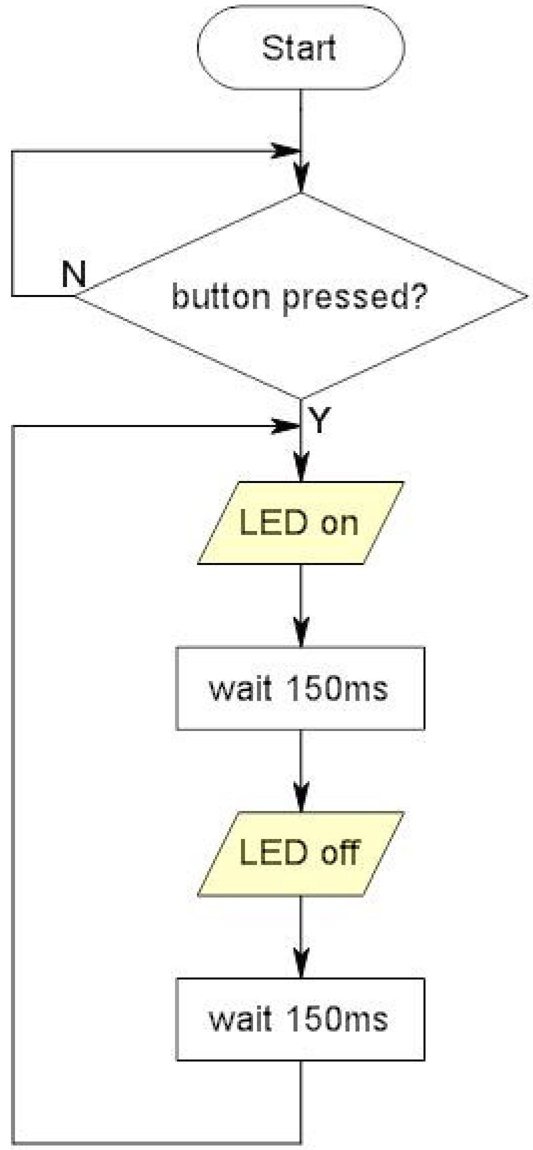
In this circuit the 10k resistor pulls the microcontroller input pin high (to 5V) making the input reliable when the switch is not pressed.



When the switch is pressed the voltage goes low (0V).

Flash1LEDv2.bas

“When the switch is pressed flash an LED rapidly on and off”



```
' Flash1ledv2.bas
$crystal = 1000000
$regfile = "attiny26.dat"

Config Porta = Output
Config Pinb.6 = Input      ‘input uses pin not port

RedSw Alias Pinb.6          ‘ hardware alias
Const Flashdelay = 150



Do
Loop Until Redsw = 0   ‘ wait until switch presses



Do


    Porta = &B10000000   ‘ LED 7 on



    Waitms Flashdelay



    Porta = 0                ‘all LEDs off



    Waitms Flashdelay


Loop
End
```

The input pin pinb.6 is normally pulled up by the resistor to 5V we call this a ‘one’ or ‘High’, when the switch is pressed it connects the pin to Ground or 0V, this is called a ‘zero’ or ‘low’

BASCOM and AVR Assignment

Learning goal:
Students should become independent learners able to find support to help their own learning

The AVR is a microcontroller from which manufacturer________________

The URL for their website is: ________________________

Download the specific datasheet for our microcontroller (the summary version not the full version) and print the first 2 pages and put them in your journal.

The Program Memory size is _______ The RAM size is _________The EEPROM size is _________

The number of I/O lines is __________ and they are arranged in _______ports

BASCOM-AVR is a compiler from _____________________

The URL for their website is: ________________________

Download the latest version of the BASCOM AVR demo and install it on your PC.

There are a number of application notes on the website for the AVR
Describe what AN128 is about

____

____

____

There are a number of other great resource websites for the AVR and BASCOM
Find 3 websites on the internet that have useful resource information on BASCOM
List the websites URL and what you found there

____

____

____

____

____

____

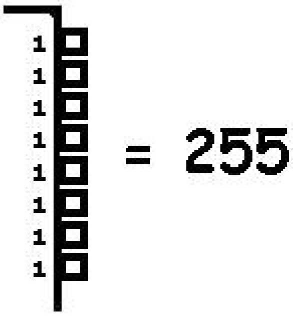
Words you need to be able to use correctly when talking about programming

computer	
microcontroller	
hardware	
software	
memory	
RAM	
variable	
data	
byte	
word	
program	
algorithm	
flowchart	
BASIC	
port	
code	
upload	
sequence	
command	
repetition	
do-loop	
for-next	
subroutine	
gosub	
return	

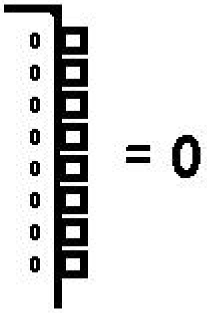
A Bit about Numbers

When we want to turn all the pins of a port on or off at one time there is are easy ways to do it.

- If all port pins are at high then the LED's will be on
  - e.g. portc=&B11111111
  - or portc = &HFF
  - or portc=255
  - or set portc.7, set portc.6, set portc.5…. to set portc.0



- If all port pins are at low then the LED's will be off
  - e.g. portc=0



Binary and Decimal Numbers

Sometimes it is easier to directly use decimal numbers to control the LED's on a port. Note that we represent a binary number using the prefix &B(there isn't prefix for decimal)

Convert &B01010101 to decimal ____

Convert &B10101010 to decimal ____

Hexadecimal Numbers

Hexadecimal is really just an abbreviated way of representing binary numbers.

Note the first 16 hex numbers 0 to F
&B00000000 = &H0 = 0
&B00000001 = &H1 = 1
&B00000010 = &H2 = 2
&B00000011 = &H3 = 3
&B00000100 = &H4 = 4
&B00000101 = &H5 = 5
&B00000110 = &H6 = 6
&B00000111 = &H7 = 7
&B00001000 = &H8 = 8
&B00001001 = &H9 = 9
&B00001010 = &HA = 10
&B00001011 = &HB = 11
&B00001100 = &HC = 12
&B00001101 = &HD = 13
&B00001110 = &HE = 14
&B00001111 = &HF = 15

Programming Codes of Practice

Three steps to help you write good programs

1. Name each program with a meaningful name and save it into its own directory
2. Use a template to setup your program from the start
3. Add lots and lots and lots of comments **as you go**

<u>You must layout programs properly and comment them well to gain achievement</u>

Saving Programs

When saving programs you need a good quality directory / folder structure, so use a different folder for each program:

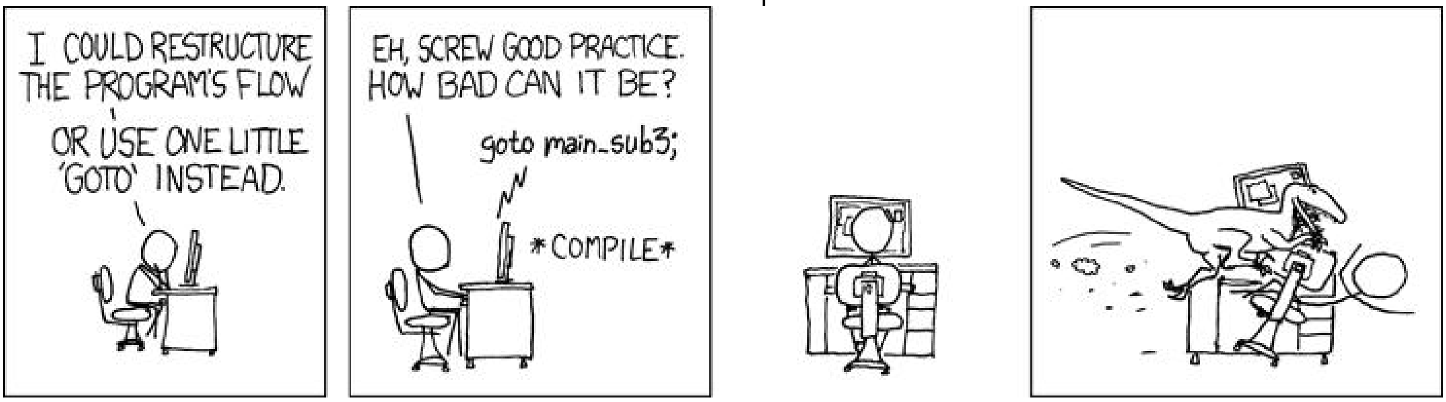
- it keeps the files that BASCOM generates for your program in one place
- this helps you find programs quickly when you want to
- it is less confusing
- it is good practice
- save your program at the beginning when you start it, this helps guard against teachers that like to turn the computers off unexpectedly.

Organisation is everything

As with structuring and organising your folders you also need to structure and organise your program code. Messy code is hard to understand and it is surprising how fast you forget what you did; and then when you want to refer to it in the future you find that you cannot understand what you have written! The use of a template or pattern to follow will help discipline your code writing. Break the code up into the following sections,

1. title block
2. program description
3. compiler directives
4. hardware setups
5. hardware aliases
6. initialise hardware
7. declare variables
8. initialise variables
9. initialise constants
10. main program code
11. subroutines.

A useful hint about codes of practice from xkcd.com


I COULD RESTRUCTURE THE PROGRAM'S FLOW
OR USE ONE LITTLE 'GOTO' INSTEAD.
EH, SCREW GOOD PRACTICE. HOW BAD CAN IT BE?
goto main_sub3;
COMPILE

Programming Template

```
'------------------------------------------------------------------
' 1. Title Block
' Author:
' Date:
' File Name:
'------------------------------------------------------------------
' 2. Program Description:




'------------------------------------------------------------------
' 3. Compiler Directives (these tell Bascom things about our hardware)
$regfile = "attiny26.dat"           'the micro we are using
$crystal = 1000000                  'the speed of the micro

'------------------------------------------------------------------
' 4. Hardware Setups
' setup direction of all ports
Config Porta = Output               'LEDs on portA
Config Portb =Input                 'switches on portB

' 5. Hardware Aliases
Led0 alias portb.0
' 6. initialise ports so hardware starts correctly
Porta = &B11111111                  'turns off LEDs

'------------------------------------------------------------------
' 7. Declare Variables

' 8. Initialise Variables

' 9. Declare Constants

'------------------------------------------------------------------
' 10. Program starts here
Do

Loop
End                           'end program

'------------------------------------------------------------------
' 11. Subroutines
```

Variables

Learning Intention:
1. Be able to use the simulator to quickly test an idea to see if it works,
2. Develop an understanding of how a computer stores data in memory and calls them variables

What is a Variable?

A variable is the **name** we give to a place set aside in the microcontroller’s memory to store a particular piece of data.
When data is stored in memory we say we are storing it in a variable.
Variables can be data read from inputs, places where you can save results of calculations for other parts of your program to use or values to control outputs.

The microcontroller has two places to store variables **RAM** and **EEPROM**. RAM is temporary storage, when the power is lost so is the data stored in RAM, this is called volatile memory. EEPROM is permanent storage (non-volatile) it remains when the power is removed from the microcontroller.

If you wanted to measure the difference between two temperatures you would store them in RAM and use a simple formula to subtract one from the other. If you wanted to record temperature measurements over a long period of time and use that data then you would collect it and store it in the EEPROM so that it would not be lost if the power was removed.

Using Variables

In a calculator with several memory locations each is given a name such as A,B,C,D,E,F,X,Y.M. etc. The name of the memory location has nothing to do with what you are using it for and it is up to you to remember what was stored in each location. In a microcontroller each memory location is given a name by the programmer. This means it is much easier for you to remember what is in the memory location and easier to use within your program.

This program generates a random number from 0 to 5 and stores it

```
' DiceV1.bas
$sim
$crystal = 1000000
$regfile = "attiny26.dat"
Config Porta = Output
Config Portb = Input

Dim Throw As Byte

Do
    'generate a random number from 0 to 5
    Throw = Rnd(6)
Loop
End
```

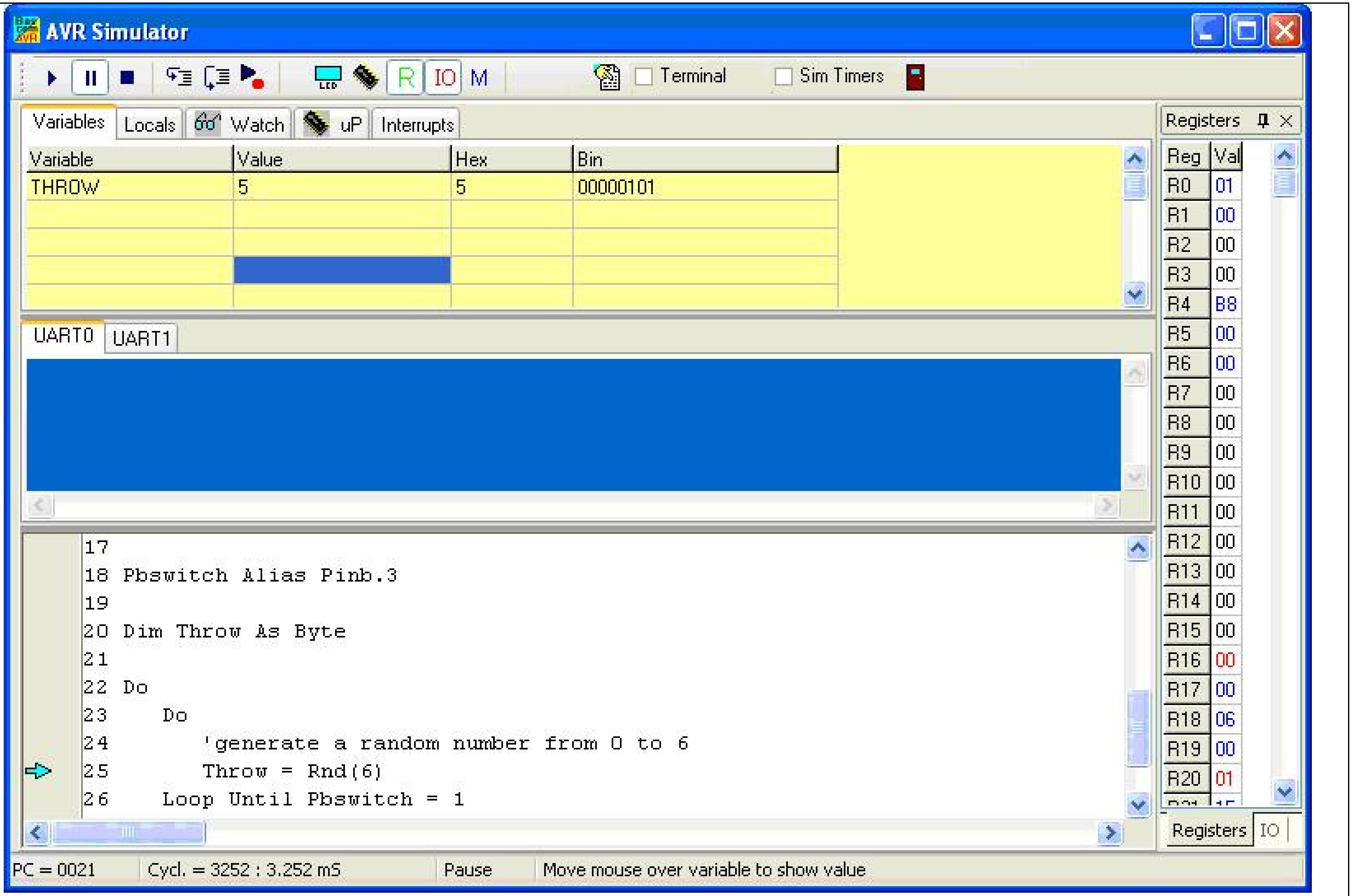
The line Dim Throw As Byte refers to our variable called Throw. Throw is the name of a location in memory (RAM) that will be used to store the random number.
Every variable must be dimensioned before it can be used.

Compile the program and then open the simulator (F2), select the variable THROW from the variables list and use F8 to step through the program to see the numbers generated by the program.

The BASCOM-AVR Simulator

Double click in the yellow area under the word VARIABLE to select the variables you want to watch.

Press F8 to step through the program and see what happens at each step.



Variable	Value	Hex	Bin
THROW	5	5	00000101

Reg	Val
R0	01
R1	00
R2	00
R3	00
R4	B8
R5	00
R6	00
R7	00
R8	00
R9	00
R10	00
R11	00
R12	00
R13	00
R14	00
R15	00
R16	00
R17	00
R18	06
R19	00
R20	01

```

Pbswitch Alias Pinb.3

Dim Throw As Byte

Do
   Do
      'generate a random number from 0 to 6
      Throw = Rnd(6)
   Loop Until Pbswitch = 1
```

PC = 0021 Cycl. = 3252 : 3.252 mS Pause Move mouse over variable to show value

The simulator is an ideal tool for testing small parts of a program to see if you achieved what you wanted to. We will use to explore different types of variables
Here is some code to show off a few BASIC commands. Copy this program into BASCOM and compile it and see if you can understand what is happening and why.

' ShowComandsV1.bas $crystal = 1000000 $regfile = "attiny26.dat" Config Porta = Output Config Portb = Output Config Pinb.6 = Input	.
'dimension variables Dim Byte1 As Byte Dim Byte2 As Byte Dim Word1 As Word Dim Int1 As Integer Dim Single1 As Single Dim Single2 As Single	

Byte1 = 12
Byte1 = Byte1 + 3
Incr Byte2
	Addition

Byte2 = Byte1 / 10	Division - a byte can only represent numbers form 0 to 255 so division truncates (IT DOESN'T ROUND)
Byte2 = Byte1 Mod 10	MOD gives you the remainder of a division
Byte2 = Byte1 * 150	This gives the wrong answer
Word1 = Byte1 * 150	This gives the right answer
For Byte2 = 1 To 8 Rotate Byte1 , Left Next	Rotate is like multiplying and dividing by ____?

Int1 = 500
For Byte2 = 1 To 8
  Int1 = Int1 - 100
Next
	Want negative numbers then use Integer or Long

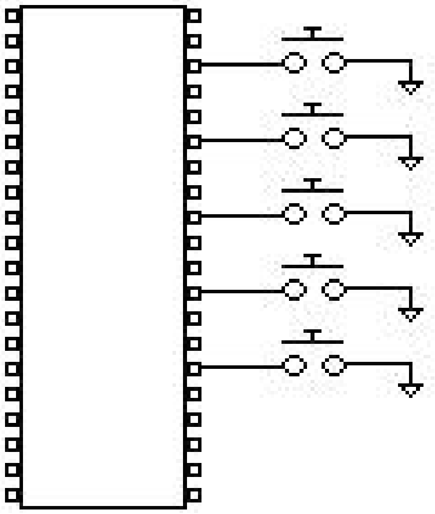
For Single1 = 0 To 90 Step 5
  Single2 = Deg2rad(single1)
  Single2 = Cos(single2)
Next
	WANT DECIMALS USE Single or Double

End	Make sure you put an END to your program or it will continue on and potentially cause problems with your projects

Control statements – IF THEN

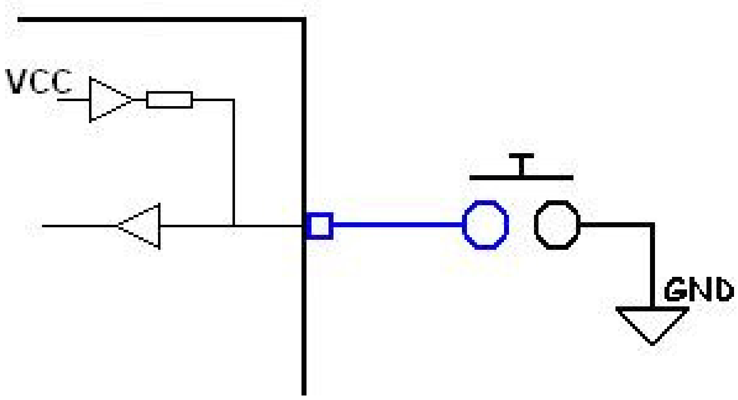
Already the first control statement DO-LOOP has been introduced the next is the IF_THEN

Connecting and programming multiple switches



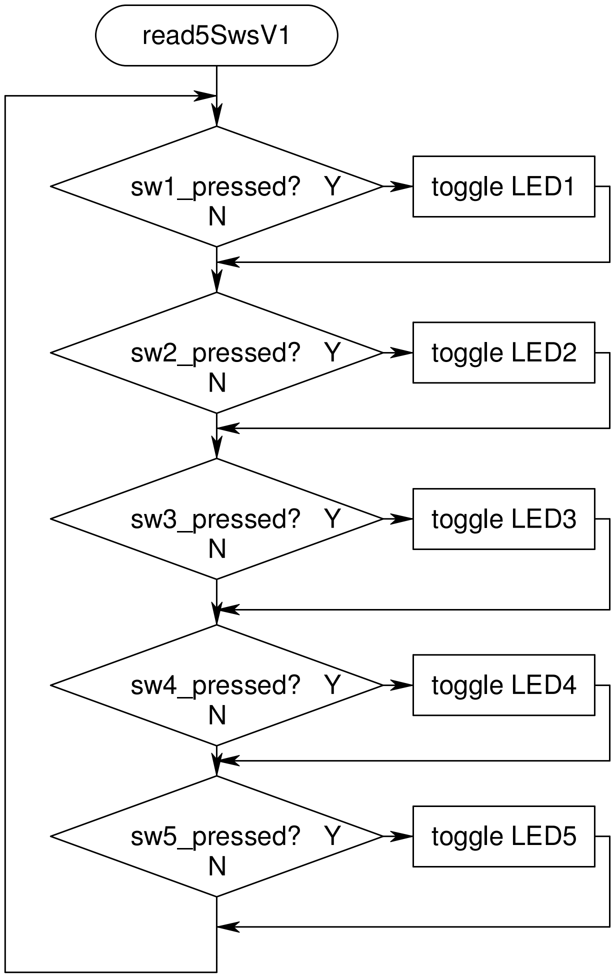
Often the microcontroller is required to read multiple input switches and then control something based upon the switch inputs. These switches might be connected to an assembly line to indicate the presence of an item, to indicate if a window is open or to the landing gear of a jet aircraft to indicate its position.
When connecting a switch a pull-up resistor is required however…

The AVR has switchable internal pull-up resistors
These can be activated from within software; this means you don't have to connect a separate resistor; however you still have to activate it. **Note that by default it is not activated.**



Config Pind.2 = Input
Set portd.2   'activate internal pull-up

Reading multiple switches in software



```
'Config 5 input switches
Config Portb = Input

'5. Alias names for the hardware
Sw_A Alias Pinb.1
Sw_B Alias Pinb.2
Sw_C Alias Pinb.3
Sw_D Alias Pinb.4
Sw_E Alias Pinb.5'when reading inputs
use pins

' 10. Program starts here
Do

    If Sw_A = 0 Then Toggle Portb.1

    If Sw_B = 0 Then Toggle Portb.2

    If Sw_C = 0 Then Toggle Portb.3

    If Sw_D = 0 Then Toggle Portb.4

    If Sw_D = 0 Then Toggle Portb.5

Loop     ' loop back to beginning
End     'end program
```

A common method of using switches within a program is to **poll** the switch (check it regularly to see if it has been pressed). When you run this program you will notice that sometimes the LED changes and sometimes it doesn't and while the switch is held down the led brightness is dim.

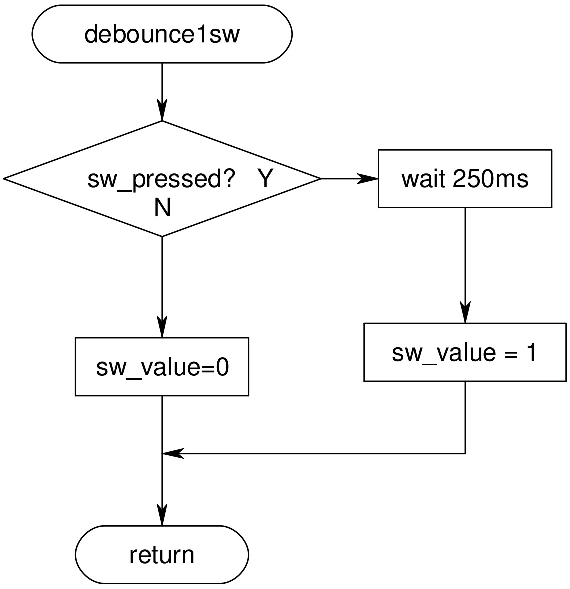
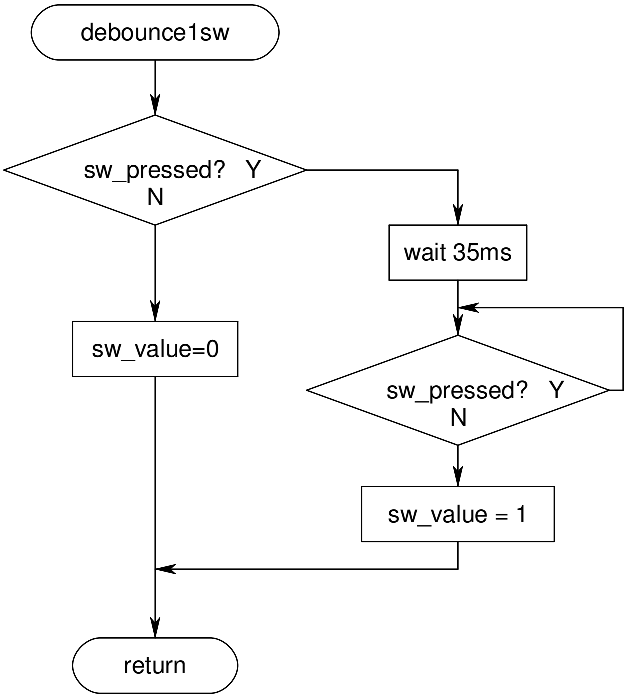
The complication causing this is the microcontrollers speed; it is running at 1MHz (million clocks per second) and so after the code is compiled the micro can test all 5 switches approximately 50,000 times or more every second. If you press and release a switch as fast as you can the micro will still test it a thousand times while you have it pressed down. This means that the LED is actually flashing on and off while the button is being held down, you can see this by observing the LED carefully as when the switch is held down the LED will dim.

There are important problems with this program:

1. If the do loop takes a short amount of time then the microcontroller will return to checking the switch before it has been released and it could be counted detected hundreds or thousands of times.
2. If the do loop takes a long time then the switch could be pressed and returned to normal and the program would never know.
3. The electrical contacts within the switch generally do not make perfect contact as they close. They actually bounce a few times. Contact bounce is a real problem when the microcontroller is running at 1,000,000 operations per second, as it can sense each bounce and interpret that as more than one press of the switch.

Using flowcharts to help solve problems

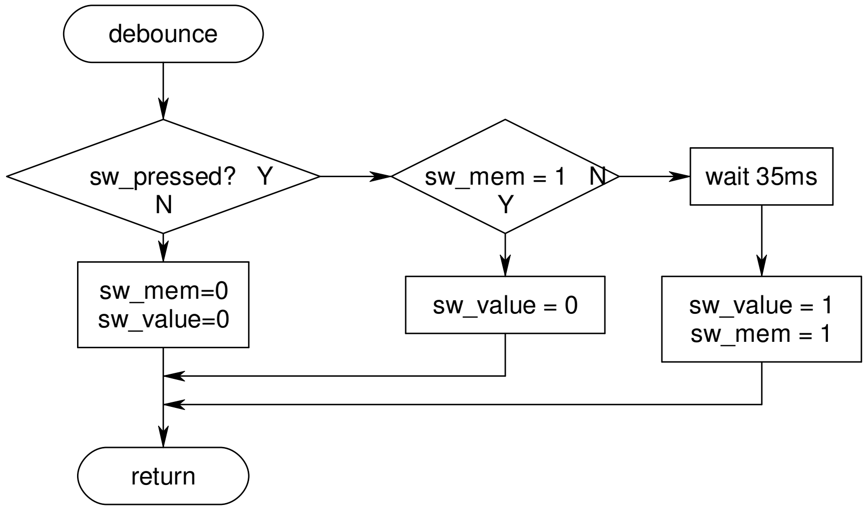
Debouncing switches is all about making sure that the program does not recognise more switch activations than it should either due to contact bounce or due to the user taking time to release the switch.

Solution stage 1: Wait for 250mS when switch pressed ALGORITHM: 1. Check if a switch is pressed 2. if not exit 3. if it is then wait for 250mS and exit VARIABLES REQUIRED: sw_value holds the number of the sw pressed The problems with this solution include: the program waits for 250mS even if the user releases it before the 250mS is up	 debounce1sw sw_pressed? Y N wait 250ms sw_value=0 sw_value = 1 return
Solution stage 2: Wait for 35mS if switch pressed (to allow for contact bounce) then check the switch until the user releases it ALGORITHM: 1. Check if a switch is pressed 2. if not exit 3. if it is then wait for 35mS 4. Check the sw again and if released exit The problem is that it still waits for the user to release the switch, and then the micro can do nothing else during that time.	 debounce1sw sw_pressed? Y N wait 35ms sw_value=0 sw_pressed? Y N sw_value = 1 return

Solution stage 3:
We only want to detect the switch when it closes and not again until it is released and pressed again and wait for 35mS to avoid detecting any contact bounces

ALGORITHM:
1. Check if a switch is pressed, if it is not pressed then exit
2. check if it is still pressed from last time, if so then do not indicate it and exit
3. wait for 35mS and
4. see if it is still pressed, if so output that this switch was pressed and remember it

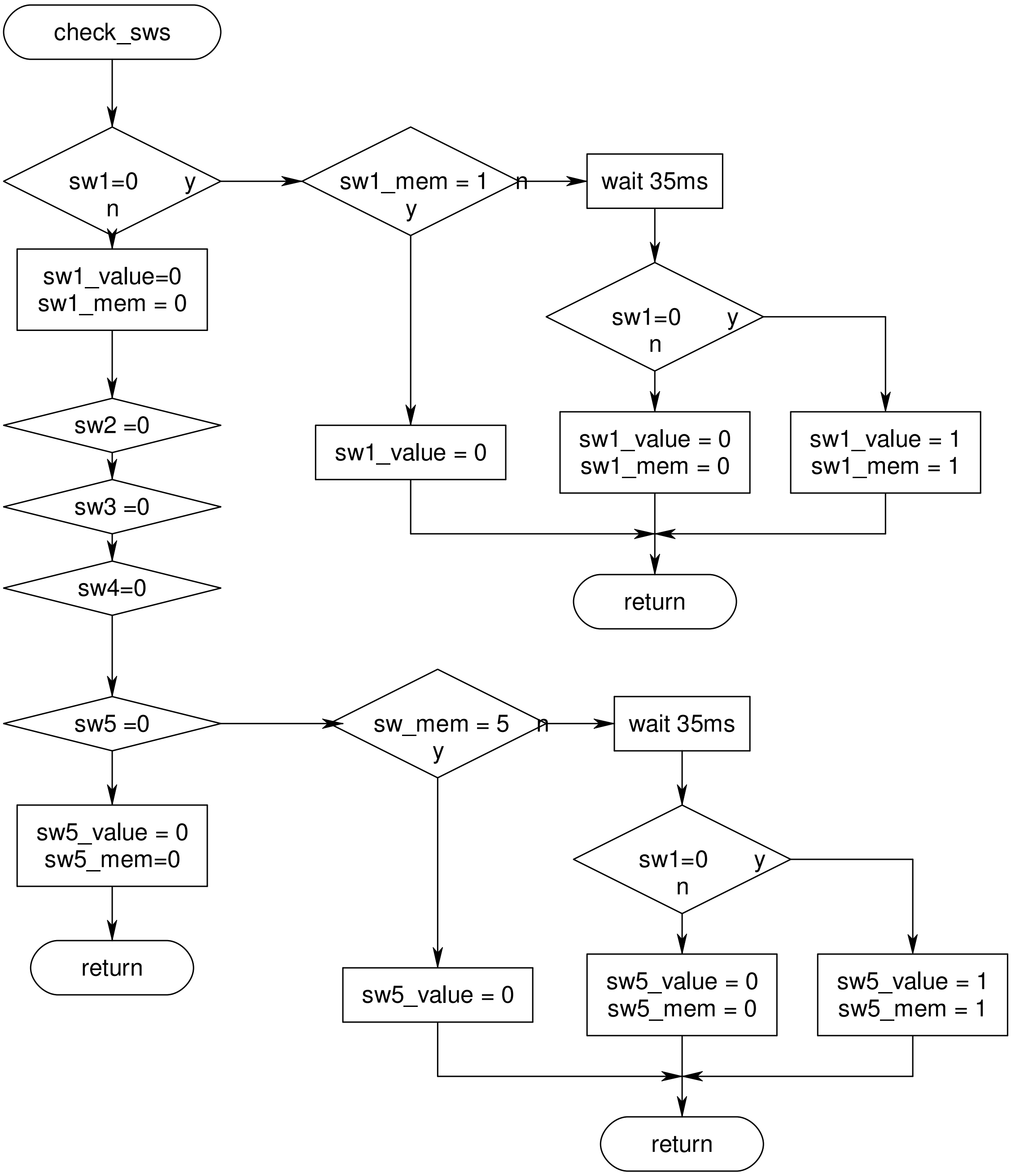
VARIABLES REQUIRED:
sw_value holds the number of the sw pressed,
sw_mem is used to remember which switch was pressed



```
Checkswitches:
  If Sw = 0 Then                 ‘switch pressed?
      If Sw_mem = 1 Then         ‘still pressed?
          Sw_value = 0          ‘ if still pressed no new value
          Exit Sub               ‘exit
      Else                        if new press
          Waitms 35              'wait for any contact bounce
          Sw_value = 1           ‘sw 1 is pressed
          Sw_mem = 1             ‘remember isw=1 t for next time
      End If
  Else                           ‘no switch pressed
      Sw_mem = 0                 ‘clear any switch
      Sw_value = 0
  End If
Return
```

Solution stage 4:

When you think through the logic of the previous solution you think it might work, however there are still problems to solve when there are multiple switches. These problems occur when multiple switches are pressed, especially if one is pressed and while it is held down another is pressed and released. It seems that it is necessary to have a separate variable for each switch and each switch memory. This may seem like a lot of memory however it can be achieved using only 1 bit for each. With 4 switches 1 byte would be need, for 8 switches 2 bytes.



\

This may or may not be the best solution, however it is important to understand that before jumping on the keyboard to write code problems should be explored fully using some sort of paper method.

This process is a god example of what counts towards excellence credits for Planning (and also the new Modelling) Achievement Standards.

Bascom Debounce

Bascom has a built in debounce command which automatically checks the state of the switch to see that it has been released before it will be counted again.

```
Do
   Debounce Sw_A , 0 , A_sub , Sub
   Debounce Sw_B , 0 , B_sub , Sub
   Debounce Sw_C , 0 , C_sub , Sub
   Debounce Sw_D , 0 , D_sub , Sub
   Debounce Sw_E , 0 , E_sub , Sub
Loop ' keep going forever
End
'-----------------------------------------------------------------
' 13. Subroutines
A_sub:
   Toggle Portb.1
Return
…
E_Sub:
   Toggle Portb.5
Return
```

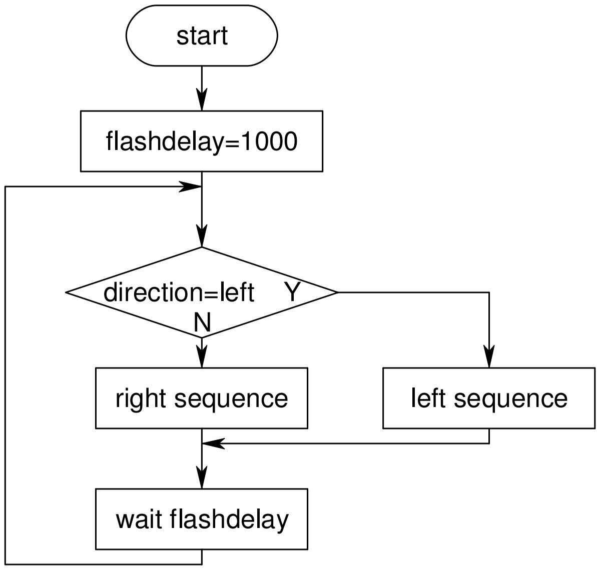
Debounce has two major ideas attached to it:
1. it checks the input pin to see if it has changed then waits for 35mSec and checks it again to see if it is still changed, otherwise it ignores it.

2. if the switch has been pressed the subroutine is carried out after the 35mS debounce time; however if the user holds the switch down when the program loops around it will not go back to the subroutine again until the switch has been released and then pressed again.

Using IF-THEN to control which parts of a program happen

An important code of practice when programming is to maintain a logical structure to your code. This makes your programs easier to read, understand and to debug. Code is broken up into large chunks and each chunk put into its own subroutine.

With the Knightrider program we can reduce the complexity by changing it to use two subroutines, one to go left and one to go right.



```
Dim Flashdelay As Word
Dim Led As Byte
Dim Direction As Bit

Direction = 0
Flashdelay = 1000

Do
   If Direction = 0 Then
      Gosub Nextright
   Else
      Gosub Nextleft
   End If
   Waitms Flashdelay
Loop
End

'Subroutines
Nextright:
Return

Nextleft:
Return
```

```
 ‘Subroutine to handle the next right led to be lit.
Nextright:
          Incr led                              ‘next right led
          Portb=255                             ‘leds off
          if  led= 0 then reset portb.0
          if  led=1 then reset portb.1
          …
          If led=7 then
                     reset portb.7
                    Direction=0                 ‘change direction
          End if
Return

Nextleft:
          decr led                              ‘next right led
          Portb=255                             ‘leds off
          if  led= 0 then
                     reset portb.0
                    direction=1
          end if
          if  led=1 then reset portb.1
          …
          If led=7 then reset portb.7
Return
```

More Interfacing

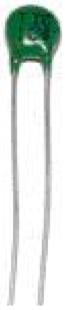
Having completed some introductory learning about interfacing and programming microcontrollers it is time to learn more detail about interfacing.

Switches

Analogue to digital conversion using

LDRS

and Thermistors



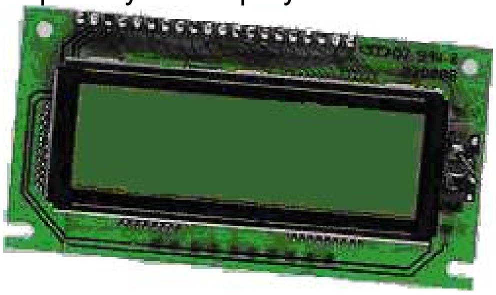
Boosting the power output

to make sound

and drive small inductive loads

Parallel interfaces to

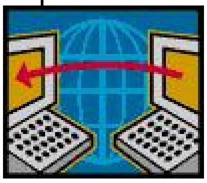
Liquid crystal displays



and seven segment displays

Serial interfaces to
Real Time Clocks

and computer RS232 ports



Analogue to Digital Conversion

In the real world we measure things in continuously varying amounts.

The golf ball is some distance from the hole. It might be 11 metres from the hole, it might be 213.46236464865465437326542 metres from the hole.

The airplane might have an altitude of 11,983 metres or perhaps 1,380.38765983 metres.

A computer works in binary (or digital) which means it has the ability to sense only two states. For example the golf ball is either in the hole or not. The plane is either in the air or not.

When we want to measure the actual distance in binary we must use a number made up of many digits e.g. 101011010 (=346 decimal) metres.

When we need to convert an analogue measurement of distance to digital we convert the decimal number to a binary number.

A to D Conversion

We need to be able to determine measurements of more than on and off, 1 and 0, or in and out. To do this we convert a continuously varying analogue input such as distance, height, weight, lighltlevel etc to a voltage.

Using the AVR this analogue value can then be converted to a binary number within the range 0 to 1111111111 (decimal 1023) within the microcontroller. We can then make decisions within our program based upon this information to control some output.

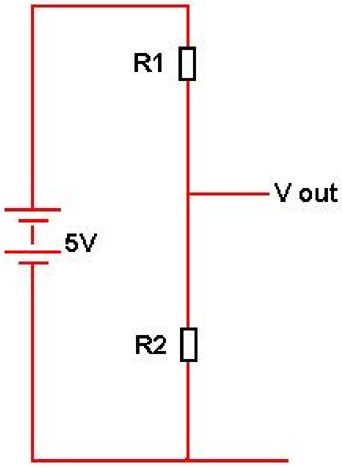
Light level sensing

We will measure the amount of light falling on a sensor and use the LED's on the microcontroller board to display its level.

The LDR
The LDR (Light Dependant Resistor) is a semiconductor device that can be used in circuits to sense the amount of light. Get an LDR and measure the resistance when it is in the dark and measure the resistance when it is in bright sunlight. Record the two values.

Voltage dividers review

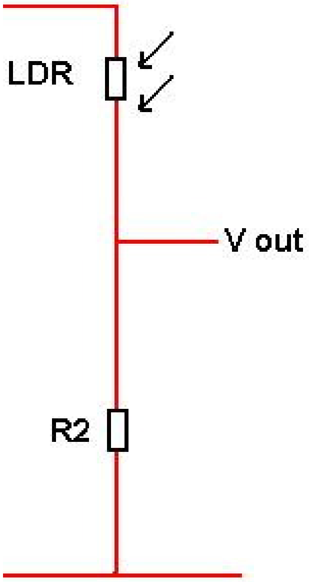
When you studied ohms law you also studied the use of voltage dividers. A voltage divider is typically two resistors across a battery or power supply.



A voltage divider is shown here. With the 5volts applied to the circuit the output voltage will be some proportion of the input voltage.

If the two resistors are the same value then the output voltage will be one_____ (quarter/half/third) of the input voltage; i.e. it has been divided by ______ (2/3/4). If we change the ratio of the two values then the output voltage will vary.

With R1 larger than R2 the output voltage will be low and with R2 larger than R1 the output voltage will be high.



Replace one of the resistors with an LDR, we know that the resistance of an LDR changes with the amount of light falling on it.

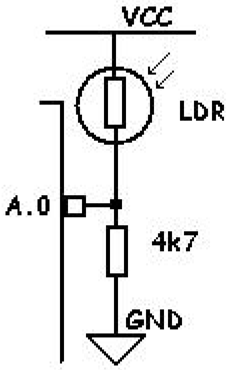
If the light level is low, and then the resistance is _____ (high/low), therefore the output voltage is _____ (low/high).

If the light level is high then the resistance is _____(high/low), therefore the output voltage is _____ (low/high).

Now this is what we call an analogue voltage. Analogue means that the voltage varies continuously between 0 and 5 volts.

But computers only know about digital voltages 0 volts or 5 Volts. We need to convert then the analogue voltage to a digital number that the computer can work with. We do that with the built in ADC (Analogue to Digital Converter) inside the Microcontroller.

AVR ADC Connections



On a micro such as the ATMEga8525, Port A has dual functions inside the microcontroller. Its second function is that of input to the internal ADC. In fact there are 8 separate inputs to the ADC one for each pin.

Reading an LDR's values in Bascom

Now we will write some code to make use of the LDR.
Note that the variable used in this program is of size WORD i.e. 2bytes (16 bits)
This is because the values given from the analogue to digital converter are bigger than 255.
Note also a new programming structure **select-case-end select** has been used.

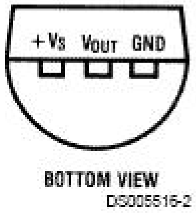
```
'--------------------------------------------------------------
' 1. Title Block
' Author: B.Collis
' Date: 7 Aug 2003
' Version: 1.0
' File Name: LDR_Ver1.bas
'--------------------------------------------------------------
' 2. Program Description:
' This program displays light level on the LEDs of portc
' 3. Hardware Features:
' LEDs as outputs
' An LDR is connected in a voltage divider circuit to portA.0
' in the dark the voltage is close to 0 volts, the ADC will read a low number
' in bright sunlight the voltage is close to 5V, the ADC will be a high value

' 4. Software Features:
' ADC converts input voltage level to a number in range from 0 to 1023
' Select Case to choose one of 8 values to turn on the corresponding LED
' 1023, 895, 767, 639, 511, 383, 255, 127,

' --------------------------------------------------------------
' 5. Compiler Directives (these tell Bascom things about our hardware)
$crystal = 8000000              'the speed of operations inside the micro
$regfile = "m8535.dat"          ' the micro we are using

'--------------------------------------------------------------
' 6. Hardware Setups
' setup direction of all ports
Config Porta = Output 'LEDs on portA
Config Portb = Output 'LEDs on portB
Config Portc = Output 'LEDs on portC
Config Pina.0 = input ' LDR
Config Portd = Output 'LEDs on portD
Config Adc = Single , Prescaler = Auto
Start Adc
' 7. Hardware Aliases
' 8. initialise ports so hardware starts correctly
'    must not put a high on the 2 adc lines as this will turn on the micros
'    internal pull up resistor and the results will be erratic
Porta = &B11111111 'turns off LEDs
Portb = &B11111111 'turns off LEDs
Portc = &B11111100 'turns off LEDs
Portd = &B11111111 'turns off LEDs


'--------------------------------------------------------------
```

```
' 9. Declare Constants
'------------------------------------------------------------------
' 10. Declare Variables
Dim Lightlevel As Word
' 11. Initialise Variables
'------------------------------------------------------------------
' 12. Program starts here
Do
   Lightlevel = Getadc(0) ' number from 0 to 1023 represents the light level
   Select Case Lightlevel
      Case Is > 895 : Portc = &B01111111 'turn on top LED in bright light
      Case Is > 767 : Portc = &B10111111
      Case Is > 639 : Portc = &B11011111
      Case Is > 511 : Portc = &B11101111
      Case Is > 383 : Portc = &B11110111
      Case Is > 255 : Portc = &B11111011
      Case Is > 127 : Portc = &B11111101
      Case Is < 128 : Portc = &B11111110 'turn on bottom LED in dark
   End Select
Loop ' go back to "do"

End 'end program
'------------------------------------------------------------------
' 13. Subroutines
'------------------------------------------------------------------
' 14. Interrupts
```

Temperature measurement using the LM35

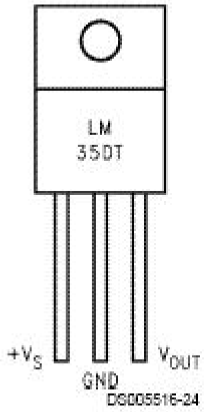
The LM35 series are precision integrated-circuit temperature sensors, whose output voltage is linearly proportional to degrees Celsius temperature.

TO-92
Plastic Package



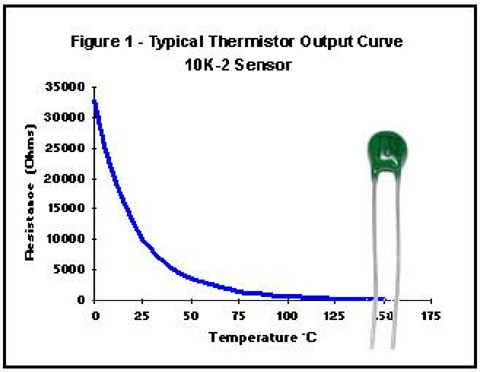
Order Number LM35CZ,
LM35CAZ or LM35DZ
See NS Package Number Z03A

TO-220
Plastic Package*



*Tab is connected to the negative pin (GND).

Order Number LM35DT



The usual temperature sensor that comes to mind is the Thermistor however thermistors are not linear but logarithmic devices as shown in this graph. If you do want to use a thermistor then try putting a resistor in parallel with it to make it more linear, however it will not be linear over its whole range.

The LM35 varies linearly over its range with typically less than a ¼ degree of error. The voltage output varies at 10mV per degree. Connect the LM35 to 5V, ground and one analogue input pin. The code is very straight forward

```
Dim Lm35 as word
Read_LM35:
          Lm35 = getadc(2)
          Locate 2,1
          Lm35 = lm35 / 2
          Lcd “temperature= ” ; LM35
return
```

The value increases by 2 for every increase of 1 degree. When connected to 5V a temperature of 25 degrees will give an output of 250mV and an ADC reading of approximately 50 (the ADC range is 0 to 1024 for 0 to 5v).

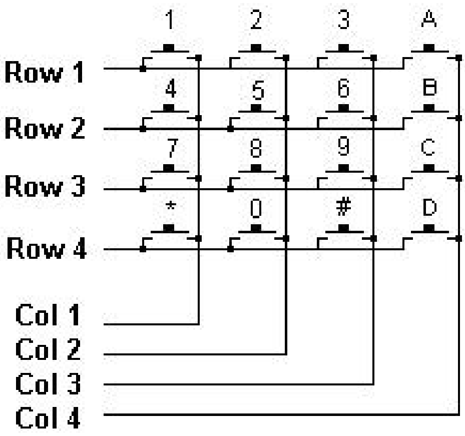
Keypad Interfacing

It is quite straightforward in Bascom to read a keypad, it handles all the hard work for us with the built in function **Getkbd**().

```
Config Kbd = Portb
Dim kbd_data As Byte
Kbd_data = Getkbd()  'keybdb returns a digit from 0 to 15
LCD kybd_data
```

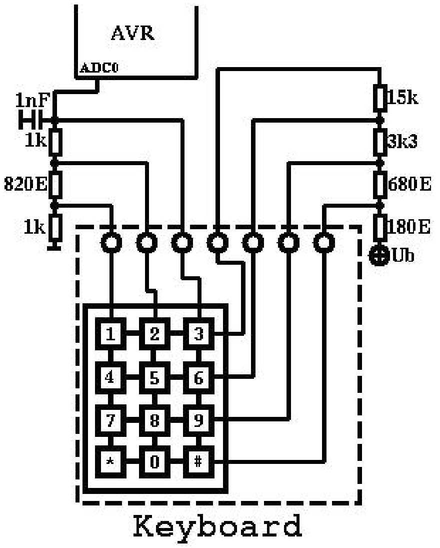
The connection to the microcontroller is straightforward as well, just 8 pins.
Solder headers into the 8 pins of the keypad and 8 pins as shown on the PCB

How do the 16 key keypad and the software work together?



The Keypad is arranged in a matrix of 4x4 and each row and column are connected to the microcontroller.
Software:
The micro sets the rows as outputs and puts a low on those ports. The columns are set as inputs, it reads the columns and if any key is pressed there will be a 0 on one of the columns. If there is a 0 then it reverses the situation with the rows as inputs and columns as outputs and if there is a low on one of the rows it has a valid keypress. The combination of 0's is used to determine exactly which key is pressed.

Alternative keypad interface



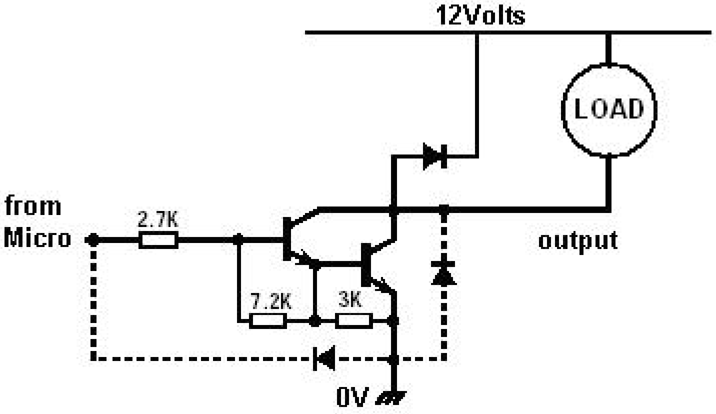
Knowing what you know about keypads and the ADC, how would this keypad circuit work and how you would program it?

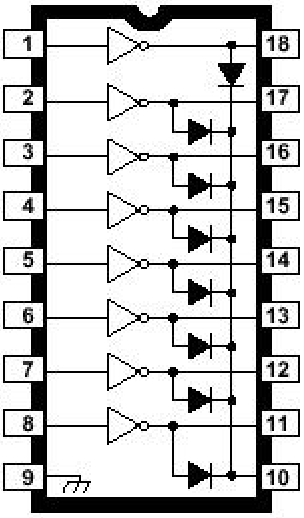
Bascom also has the ability to read a computer keyboard connected directly to an AVR micro, check it out in the samples directory installed with Bascom and the help file.

Controlling high power loads (outputs)

ULN2803 Octal Darlington Driver

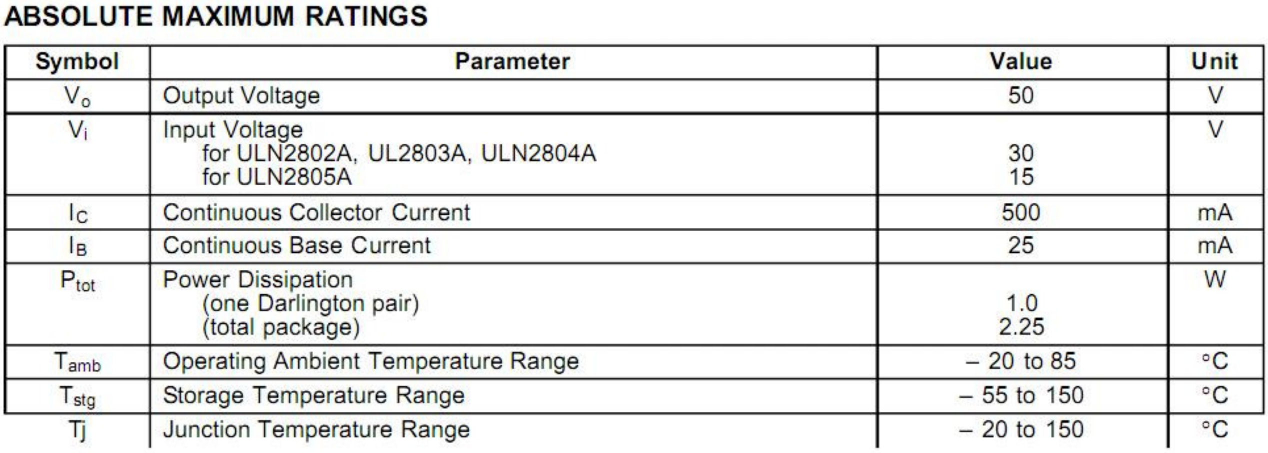
Typically a microcontroller I/O port can only drive 20mA into a load, so when more power is called for a high power transistor or IC is required to drive multiple relays, solenoids, or high power lamps.





This IC has 8 sets of Darlington-pair transistors inside it. The Darlington pair configuration is when two transistors have their collectors connected and the emitter of the first drives the base of the other. This is a very high gain (amplification) device.

Connecting high power loads such as relays, solenoids, light bulbs



ABSOLUTE MAXIMUM RATINGS

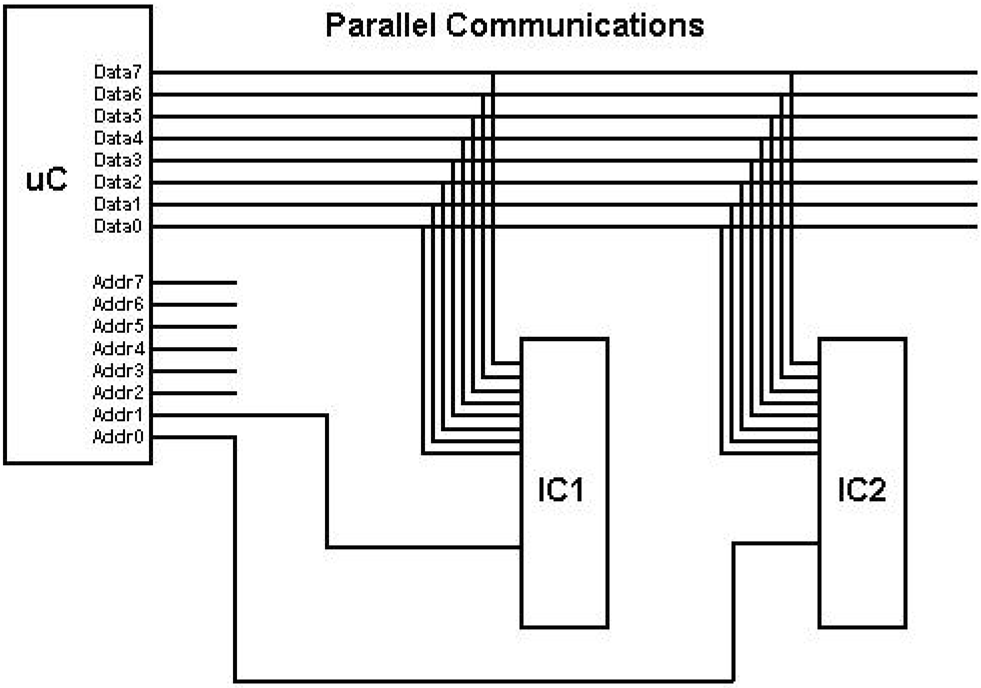
Symbol	Parameter	Value	Unit
V_o	Output Voltage	50	V

V_i	Input Voltage
	for ULN2802A, UL2803A, ULN2804A
	for ULN2805A
		
		30
		15
			V

I_C	Continuous Collector Current	500	mA
I_B	Continuous Base Current	25	mA
P_{tot}	Power Dissipation (one Darlington pair) (total package)	 1.0 2.25	W
T_{amb}	Operating Ambient Temperature Range	– 20 to 85	°C
T_{stg}	Storage Temperature Range	– 55 to 150	°C
Tj	Junction Temperature Range	– 20 to 150	°C

Parallel Data Communications

Both internal and external communications with microcontrollers are carried out via **buses**, these are groups of wires. A bus is often 8 bits/wires (byte sized) however in systems with larger and more complex microcontrollers and microprocessors these buses are often 16, 32 or 64 bits wide.



Communication is carried out using 8 or more bits at a time. This is efficient as an 8 bit bus can carry numbers/codes form 0 to 255, a 16 bit bus can carry numbers/codes from 0 to 65,535 and 32 bits can carry numbers/codes from 0 to 4,294,967,295. So data can move fairly fast on a parallel bus.

Parallel communication is often used by computers to communicate with printers, because of this speed. Only one printer can be connected to the parallel port on a computer, however within the computer itself all the devices on the bus are connected all the time to the data bus. They all share access to the data, however only the device that is activated by the address bus wakes up to receive/send data.

LCDs (Liquid Crystal Displays)

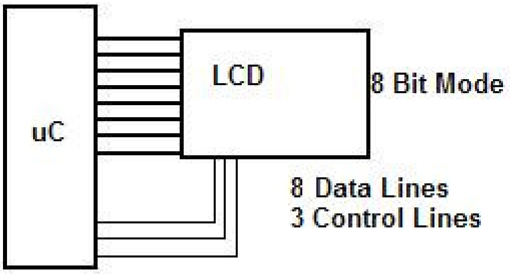
One of the best things about electronic equipment nowadays is the alphanumeric LCD displays, these are not the displays that you would find on a laptop they are simpler single, double or 4 line displays for text. These displays are becoming cheaper and cheaper in cost check out www.pmb.co.nz for great prices on them. The LCD is a great output device and with Bascom so very easy to use.

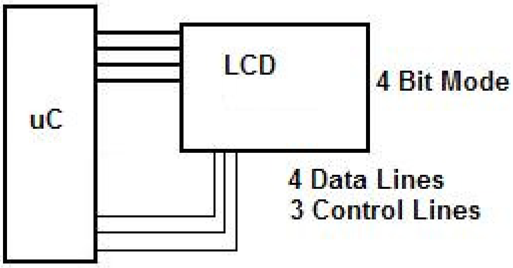
Some common commands are

- cls - clear the screen
- LCD "Hello" - will display hello on the display
- lowerline - go to the lower line
- locate y,x - line and position on the line to start text

Connecting an LCD to the microcontroller is not difficult either.
There are 14/16 pins on the LCD

1. 0V
2. +5V
3. Contrast
4. RS - register select
5. R/W - read/not write
6. E - Enable
7. D0
8. D1
9. D2
10. D3
11. D4
12. D5
13. D6
14. D7
15. Backlight +
16. Backlight 0V



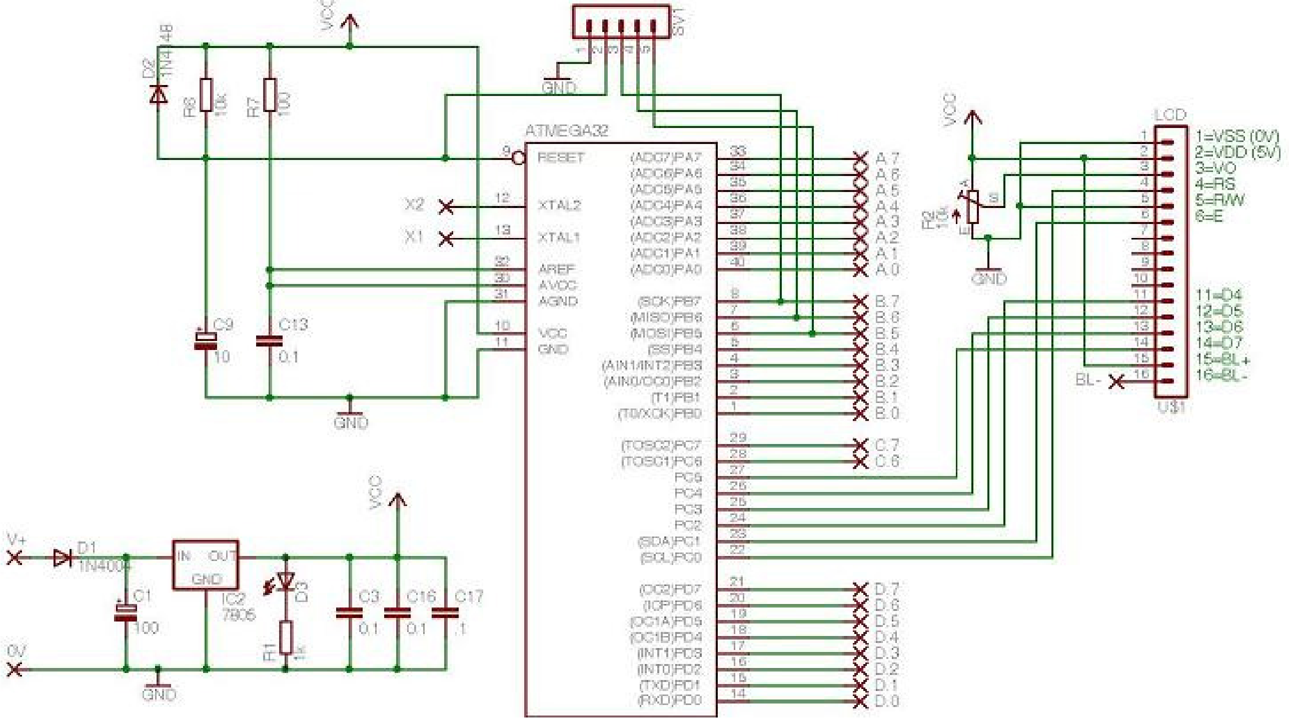


Most LCDs are set up so that they can communicate in parallel with either 4 bits or 8 bits at a time. The faster system is 8 bits as all the data or commands sent to the LCD happen at the same time, with 4 bit operation the data/command is split into 2 parts and each is sent separately. Hence it takes twice as long. The advantage of 4 bit operation is that the LCD uses 4 less lines on the micro.

Another couple of lines are necessary, these are control lines, RS , R/W, E. When using Bascom the R/W line is tied permanently to 0V, and the other two lines need to be connected to the micro.

Connecting an LCD to a 40pin AVR

This requires six I/O lines to be used on the micro.



Software to show off the display

```
'-----------------------------------------------------------------
' 1. Title Block
' Author: B.Collis
' Date: 14 Aug 2003
' Version: 1.0
' File Name: LCD_Ver1.bas
'-----------------------------------------------------------------
' 2. Program Description:
' use an LCD to display
' 3. Hardware Features:
' LCD on portc - note the use of 4 bit mode and only 2 control lines
' 4. Program Features:
' outer do-loop
' for-next control
'-----------------------------------------------------------------
' 5. Compiler Directives (these tell Bascom things about our hardware)
$crystal = 8000000              'the speed of operations inside the micro
$regfile = "m8535.dat"          ' the micro we are using
'-----------------------------------------------------------------
' 6. Hardware Setups
' setup direction of all ports
Config Porta = Output 'LEDs on portA
Config Portb = Output 'LEDs on portB
Config Portc = Output 'LEDs on portC
Config Portd = Output 'LEDs on portD
Config Lcdpin = Pin , Db4 = Portc.2 , Db5 = Portc.3 , Db6 = Portc.4 , Db7 = Portc.5 , E  =
Portc.1 , Rs = Portc.0
Config Lcd = 20 * 4        'configure lcd screen
' 7. Hardware Aliases
' 8. initialise ports so hardware starts correctly
Porta = 0
Portb = 0
Portd = 0
Cls 'clears LCD display
Cursor On  ' cursor displayed
'------------------------------------------------------------------
' 9. Declare Constants
Const Timedelay = 150
'------------------------------------------------------------------
' 10. Declare Variables
Dim Position As Byte
' 11. Initialise Variables
Count = 0
'------------------------------------------------------------------
' 12. Program starts here
Locate 1,5
Lcd “watch this”
Locate 2,6
Lcd “hello”
Waitms timedelay
Locate 2,1
Lcd “      “
Waitms timedelay
Locate 3,5
Lcd "hows that!!"
End
'------------------------------------------------------------------
' 13. Subroutines
'------------------------------------------------------------------
' 14. Interrupts
```

FOR NEXT - Controlling the number of times something happens

If you want some text to move across the LCD.

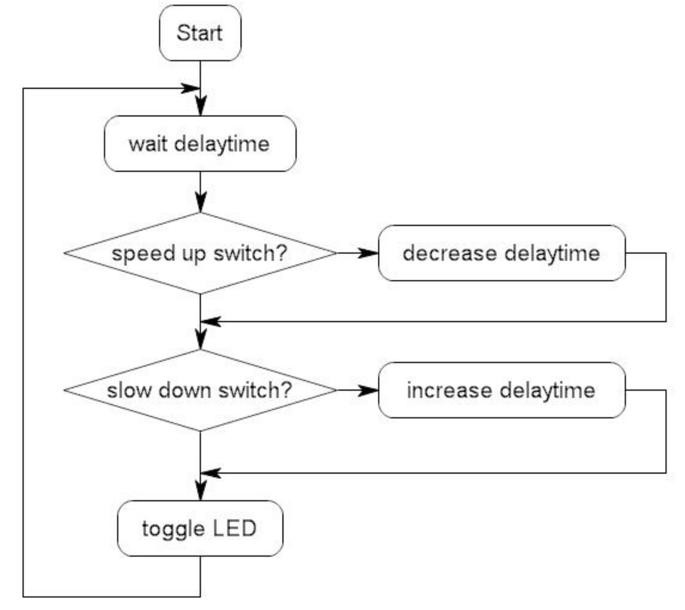
You could do it the long way

```
Do
    Locate 2,1
    Lcd “Hello”
    Waitms timedelay
    Locate 2,1
    Lcd “     “

    Locate 2,2
    Lcd “Hello”
    Waitms timedelay
    Locate 2,2
    Lcd “     “

    Locate 2,3
    Lcd “Hello”
    Waitms timedelay
    Locate 23
    Lcd “     “
Loop
```

OR the smart way

```
Do
    For Position = 1 To 16      ‘for 20 character display
        Locate 2, position      ‘move the cursor to second row
        Lcd “Hello”             ‘display the text starting at this position
        Waitms Timedelay        'wait a bit
        Locate 2, position      ‘move cursor back to
        Lcd “     “             ‘blank over the text to delete it
    Next
    For Position = 16 To 1, step -1       ‘for 20 character display
        Locate 2, position      ‘move the cursor to second row
        Lcd “world”             ‘display the text starting at this position
        Waitms Timedelay        'wait a bit
        Locate 2, position      ‘move cursor back to
        Lcd “ ”                 ‘blank over the text to delete it
    Next
Loop
End          'end program
```

Don’t delay



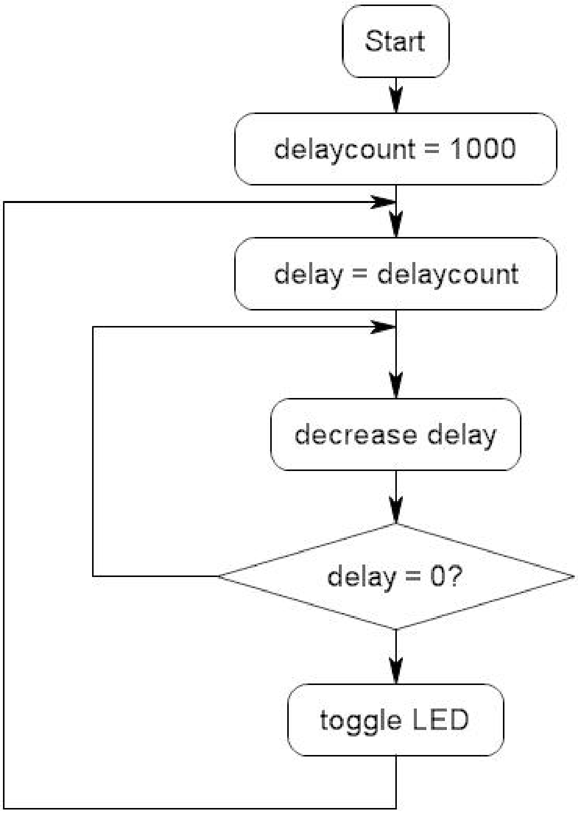
```
Do


    Waitms delaytime


    If swa=0 then decr delaytime


    If swa=0 then decr delaytime


    Toggle led
Loop
```

In this program two switches are used to change the rate at which an LED flashes.

There is a significant problem with this program however and it is that when the microcontroller is waiting (wait delaytime) it cannot read a switch press.

As the delay increases this only becomes a bigger problem.

For this reason we do not use lengthy waitms statements in programs we find alternative solutions to our problems



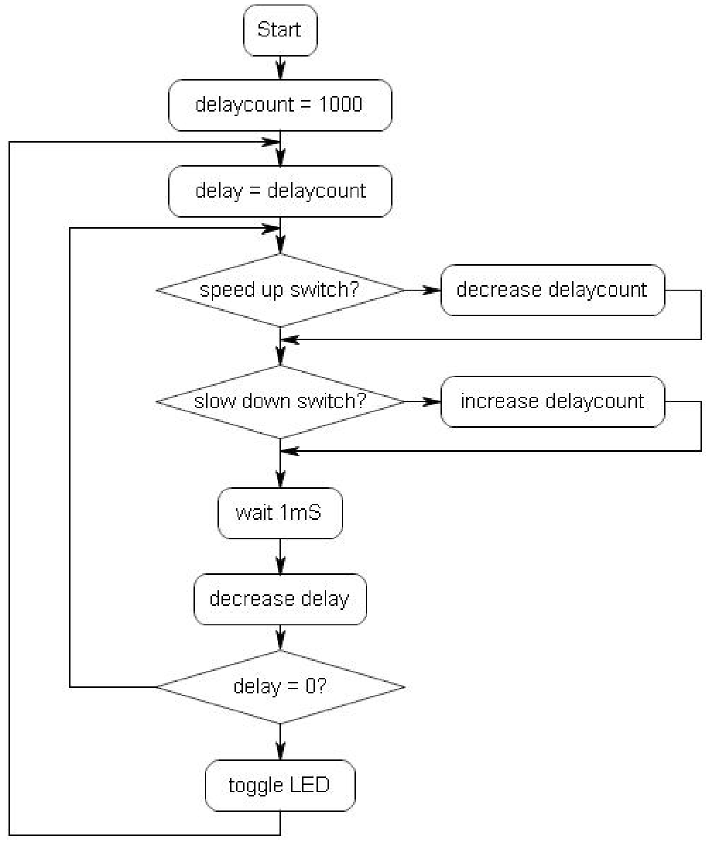
To begin to solve the issue you should understand that a delay routine in a program is simply a loop that repeats a large number of times e.g.

If this loop takes approximately 2 uSec (microseconds) to complete and does it 1000 times then it will give a delay of 2 mSec

How many times would the loop have to repeat to delay:

1mS ?
10mS ?
1 Second ?
1 Minute ?

In a program like this it is acceptable to put in a very small delay. For example a press button switch would typically be held down for much longer than 1mS so in this program there is a 1mSec delay used and we put the switch checking and 1mSec delay within our own longer delay.
Note that we need to keep 2 variables, one is DelayCount which we increase and decrease using the switches. The other is a temporary copy of it tDelay which is decremented within the loops.



```
Delaycount=0


do

    tDelay=delaycount
    do

        if swa=0 then decr delaycount


        if swb=0 then incr delaycount


        waitms 1

        decr tdelay

    loop until tdelay = 0

   toggle led

loop
```

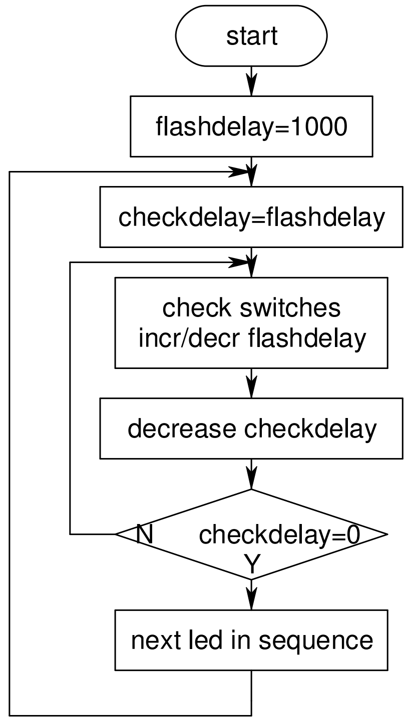
Although the main problem is fixed there are some other problems to fix:

1. When you keep incrementing delaycount eventually it will get to 65535, and another increment will cause it to roll over or **overflow** back to 0.
2. Also when delaycount gets down to 0, another decrement will cause it to **underflow** to 65535!
3. The resolution (degree of change) of our delay program is not very good if we increase or decrease each time by one. Perhaps a bigger increment/decrement value might be more useful.

A neat feature for the Knightrider program would be if the speed of the sequence could be varied.

So for the same reasons as before the switches need checking often; so after each led in the sequence of LEDs, read the switches, wait a preset amount of time, if one button is pressed increase the delay time, if the other button is pressed decrease the delay time.
The switches should be checked at least every 1mS so that they can detect user input.

To do this we implement a loop within the program that initially begins at the value of flashdelay and counts down to 0, a second variable checkdelay is needed as a copy of flashdelay



```
Dim Flashdelay As Word
Dim Led As Byte
Dim Checkdelay As word
dim direction as bit
Flashdelay = 1000
Do

   Checkdelay = Flashdelay
    Do

        Gosub Checkswitches


        Decr Checkdelay


    Loop Until Checkdelay = 0

    If Direction = 0 Then
        Gosub Nextright
    Else
        Gosub Nextleft
    End If
Loop
End
'Subroutines
…
```

The check switches subroutine using debounce commands

```
Checkswitches:
  Debounce Sw1 , 0  , Decrflashdelay, Sub
  Debounce Sw2 , 0  , Incrflashdelay, Sub
Return

Decrflashdelay:
  Decr Flashdelay
Return

Incrflashdelay:
  Incr Flashdelay
Return
```

Programs as solutions: understanding algorithms and flowcharts

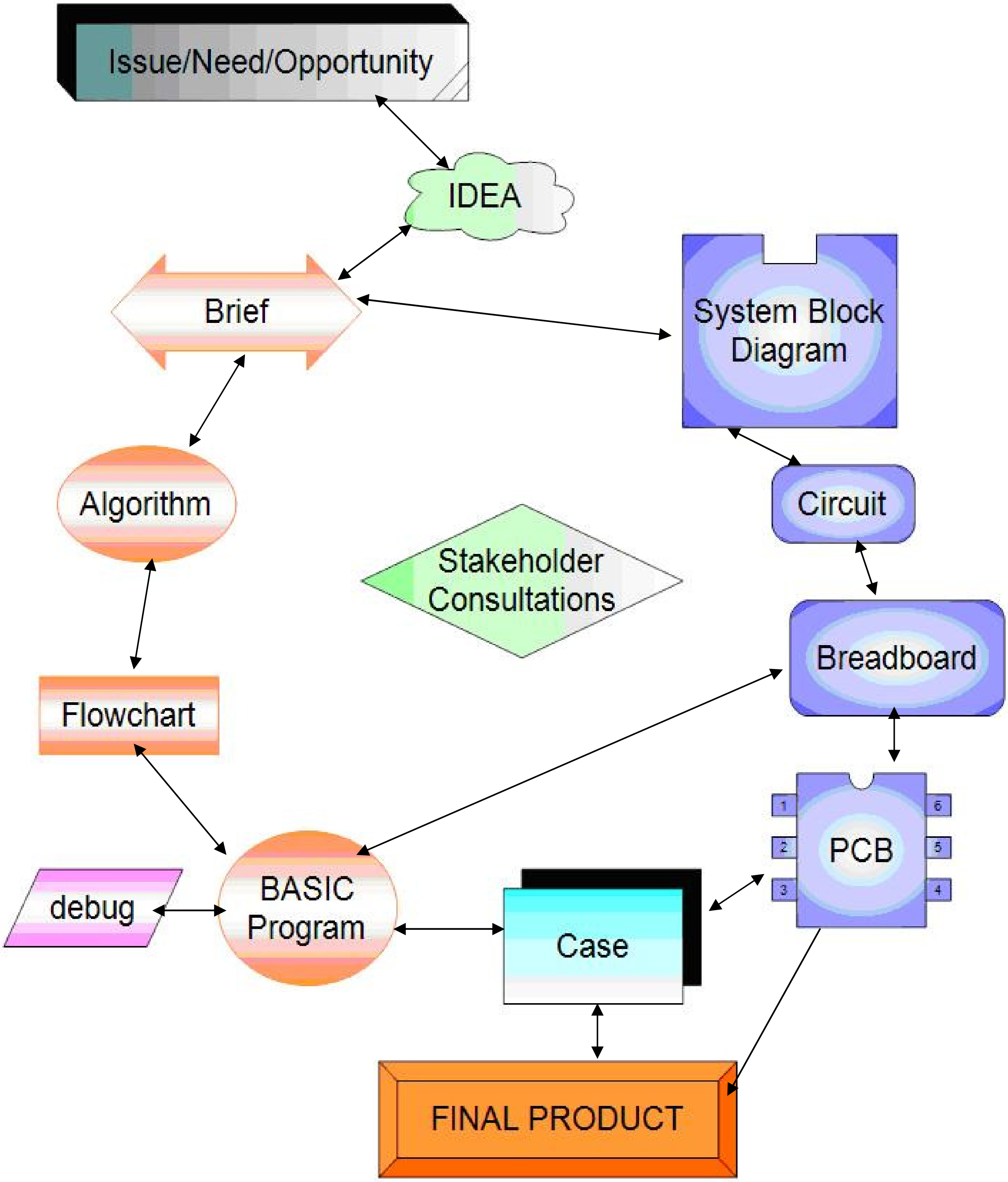
When learning to program students find it straight forward to write programs which contain one simple process and result in a few lines of code; however they often struggle to reach the next stage which requires them to write programs that require a more complex process or multiple processes together. Because of their ease with uncomplicated code they begin programming at the keyboard rather than with pen and paper and their programs can become confused very quickly.

Technological practice (at all levels) requires students to undertake planning and to conform to good codes of practice; so when writing software students must not write software without planning it first.

You will learn how to follow through a process of developing a program from initial idea through to code.

Planning Sequence for an AVR project

1. Research on, then write an explanation of, the problem, issue, need or opportunity
2. Draw a System Block Diagram and write any comments to clarify what you need to do (this is called a brief)
3. Sketch the physical device
   (e.g. case drawings)
4. Write down the algorithm (process) to be carried out by the program (this can be quite difficult, however if you can't do it now then there is no way you can write code to do it later!)
5. Determine the variables to be used by the program
6. Design a flowchart for the process
7. Test it using a range of inputs
8. Identify the control statements that need to be used
9. Develop the circuit
   - Decide which input and output devices to develop first
   - Start with simple circuits and build up to the final circuit in stages, planning each stage as you go with:
     i. schematic and layout diagrams
     ii. a flowchart
        - A visual diagram of the way the software operates
     iii. a program
10. Write and test your program
11. Design or find a suitable case
12. Design a PCB to suit the case
13. Make and test the PCB
14. Put into the case


Issue/Need/Opportunity
IDEA
Brief
System Block Diagram
Algorithm
Circuit
Stakeholder Consultations
Breadboard
Flowchart
PCB
1
2
3
6
5
4
debug
BASIC Program
Case
FINAL PRODUCT

One Page Brief

Name: __________________ Project: _______________ Date: _____ Version: ___

Client, customer or end-user:
Description of the problem, issue, need or opportunity(diagrams may be required):
Conceptual Statement: Design and construct a …
System Block Diagram: (include all input and output devices)
Further written specifications:

Algorithm Development Worksheet

Name: __________________ Project: _______________ Date: _____ Version: ___

Define all the input and output devices				
Inputs		**Outputs**		
Device Description	**Name**	**Device Description**	**Name**	**Starting State**

The algorithm (description of the operation of the system)

Initially the

For each input describe what happens to any output devices
Use "**if** __________ **then** _________" or " ____________ **until** ___________" statements

Example Brief

Name: ___________________ Project: ________________ Date: _____ Version: ___

Client, customer or end-user: …

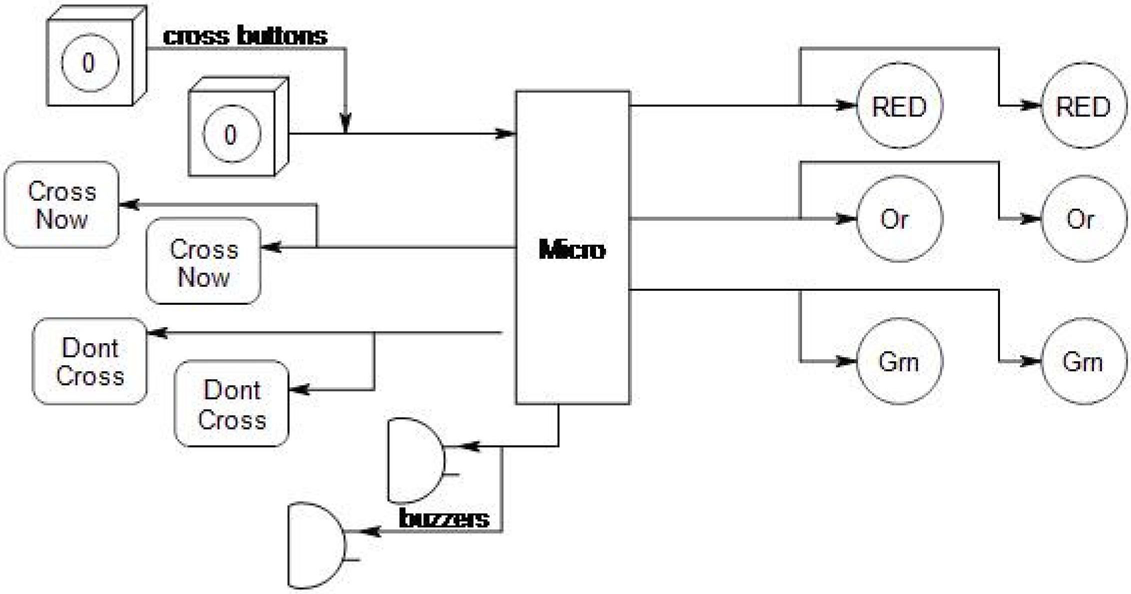
Description of the problem, issue, need or opportunity (diagrams may be required):

Vehicles travel at high speeds on this road and although there is a pedestrian crossing, pedestrians are at risk

Conceptual Statement:

Design and construct a set of traffic lights for a pedestrian crossing

System Block Diagram: (include all input and output devices)



Further written specifications:

Lights go from red to orange when the button is pressed, waits for 25 seconds then goes red for 1.5 minutes then back to green, Cross and DontCross lights work as per expected.

Algorithm Planning Example

Name: __________________ Project: _______________ Date: _____ Version: ___

Define all the input and output devices				
Inputs		**Outputs**		
Device Description	**Name**	**Device Description**	**Name**	**Starting State**
Large buttons on each pole for pedestrians to press to cross	CROSSBUTTON	RED traffic lights for cars on pole	REDLIGHT	OFF
		Orange traffic lights for cars	ORANGELIGHT	OFF
		Green traffic lights for cars	GREELIGHT	ON
		Buzzer to indicate to pedestrians to cross now	BUZZER	OFF
		CROSS NOW light on each pole	CROSSNOW	OFF
		DON'T CROSS light on each pole	DONTCROSS	On

The algorithm (description of the operation of the system)

Initially the
Redlight , orangelight, buzzer and cross are off,
Greenlight, dontcross are on

For each input describe what happens to any output devices
Use "**if** __________ **then** _________" or " ____________ **until** ___________" statements

If the pedestrian presses the crossbutton then
The greenlight goes off, the orange light goes on

After 25 seconds the orangelight goes off the redlight goes on

Draw a flowchart, write test and debug your program
Changes to the brief to consider:
- After the redlight comes on should there be any delay before the crossnow?
- How long should the buzzer stay on after crossnow comes on?
- What signals are given to pedestrians before the lights go green again?

Example Brief

Name: Mr C **Project: Glue Gun Timer** **Date: someday!** **Version: 1**

Client, customer or end-user: ME

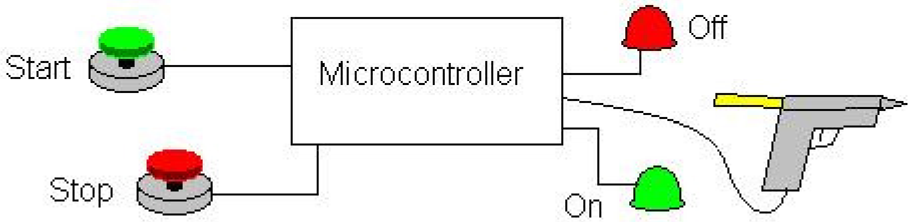
Description of the problem, issue, need or opportunity (diagrams may be required):

The hot glue gun is left on and forgotten about making a mess on the bench and creating a potential hazard

Conceptual Statement:

Design and construct a timer for the hot glue gun to turn it off automatically after 60 minutes

System Block Diagram: (include all input and output devices)



Further written specifications:

The glue gun turns on only when the start button is pressed

It automatically goes off after 60 minutes

It goes off if the stop button is pressed

If the start button is pressed at anytime the timer starts from 60 again

The green and red LEDs indicate if the glue gun is on or off

Algorithm Development Example

Name: Mr C **Project: Glue Gun Timer** **Date: someday! Version: 1**

Define all the input and output devices

Inputs		**Outputs**		
Device Description	**Name** (use single words)	**Device Description**	**Name** (use single words)	**Starting State**
Green pushbutton switch	**startbtn**	Red LED	**offled**	On
Red push button switch	**stopbtn**	Green LED	**onled**	Off
		Hot Glue Gun	**gluegun**	Off

The algorithm (description of the operation of the system)

Initially the

Offled is ON, onled and ggun are off

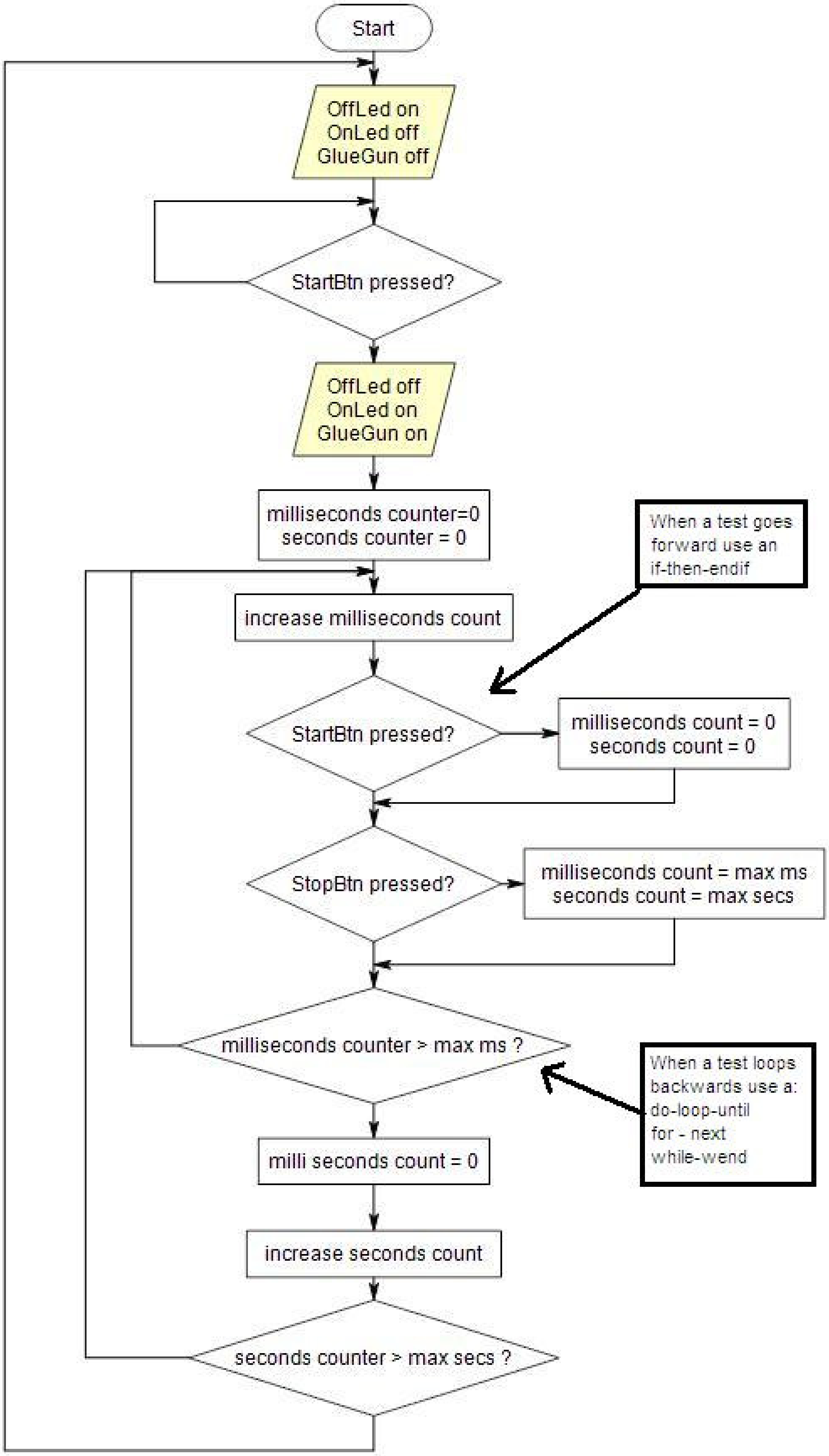
For each input describe what happens to any output devices
Use “**if** ___________ **then** __________” or “ _____________ **until** ____________” statements

If the user presses **startbtn** then the **Offled** goes off, **onled** and **ggun** go on

These stay in this state until 60 minutes has passed then the **Offled** goes ON, **onled** and **gluegun** go off

If the user presses **stopbtn** then the **Offled** goes ON, **onled** and **gluegun** go off

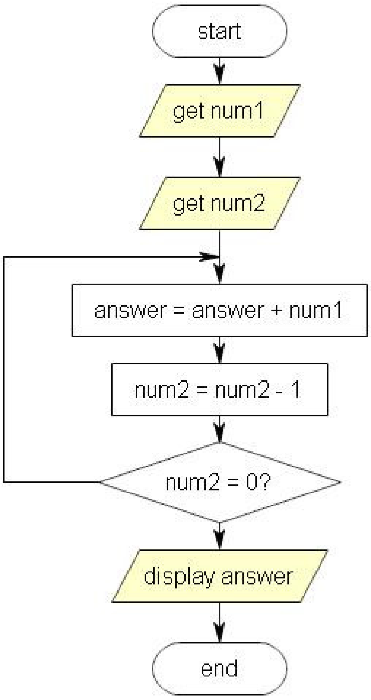
Glue Gun Timer Flowchart



```
' OneHourTimerResetStop.bas
' B.Collis 1 Aug 2008
' 1 hour glue gun timer program
' the timer restarts if the start button is pressed again
' the timer can be stopped before timing out with the stop button
$crystal = 1000000
$regfile = "attiny26.dat"
Config Porta = Output
Config Portb = Output
Config Pina.2 = Input
Config Pina.3 = Input
Gluegun Alias Porta.5
Offled Alias Porta.6
Onled Alias Porta.7
Startbutton Alias Pina.2
Stopbutton Alias Pina.3
Dim Mscount As Word
Dim Seccount As Word
Const Max_mscount = 999
Const max_secCount = 3599
Do
     Set Offled
     Reset Onled
     Reset Gluegun Initially Off
     Do
     Loop Until Startbutton = 0

     Reset Offled
     Set Onled
     Set Gluegun
     Mscount = 0
     Seccount = 0
     Do
         Do
              Incr Mscount                      'add 1 to milliseconds
              Waitms 1
              If Startbutton = 0 Then           'Check Switch
                    Mscount = 0                 ‘set time back to start
                    Seccount = 0
              End If
              If Stopbutton = 0 Then            'Check Switch
                      Mscount = Max_mscount     ‘set time to max
                      Seccount = Max_seccount
              End If
          Loop Until Mscount > Max_mscount      'loop 3600 times
          Mscount = 0
          Incr Seccount
     Loop Until Seccount > Max_seccount    'loop 1000 times
loop
```

Multiplication Algorithms

<table>
<tr><th>Process</th><th><em>Notes</em></th></tr>
<tr><td>Issue: Multiply two numbers together using only addition e.g. AxB=Answer</td><td><em>Not all microcontrollers can do multiplication within their internal hardware</em></td></tr>
<tr><td>Algorithm:
Add A to the answer B times</td><td><em>Finding the right words to describe the algorithm can be difficult at times, you need to concise, accurate and clear. This can be a step students struggle with.</em></td></tr>
<tr><td>Variables:
(memory locations to store data in)
numA – byte size
numB – byte size
Answer – word size</td><td><em>Choose useful names and think about the size of the variable (a byte stores 0-255, a word 0-65535, an integer stores -32768 to 32767, a long stores -2147483648 to 2147483647)</em></td></tr>
<tr><td>Flowchart:



</td><td><em>Note the shapes of the elements:</em>

<em>Start and end</em>
<em>Inputs and outputs</em>
<em>Processes</em>
<em>Decisions</em>

<em>Learn the process of keeping track of how many times something is done. A variable is used to count the number of times a loop is carried out. In this case the variable is decreased each time through the loop until it is 0. An alternative is to increase a variable until it reaches a specific value.</em>

<em>Within a microcontroller though it is often faster to test a variable against 0 than some other number.</em></td></tr>
<tr><td>Test the flowchart with an example
<table>
<tr><th>Answer</th><th>Num2</th></tr>
<tr><td>6</td><td>8</td></tr>
<tr><td>12</td><td>7</td></tr>
<tr><td>18</td><td>6</td></tr>
<tr><td>24</td><td>5</td></tr>
<tr><td>30</td><td>4</td></tr>
<tr><td>36</td><td>3</td></tr>
<tr><td>42</td><td>2</td></tr>
<tr><td>48</td><td>1</td></tr>
<tr><td>54</td><td>0</td></tr>
</table>
</td><td><em>Does it work?</em>
<em>Note how the columns in the test follow the same order as the processes in the loop.</em>

<em>This stage can be a little confusing and often we can be out by 1 either way (if it is then our answer might not be 54 but 48 or 60)</em>

<em>If you get wrong answers after a loop check that you are decreasing or increasing them the right number of times.</em></td></tr>
</table>

Identify the control statements to be used.

```
' SimpleMultiplicationV1.bas
$crystal = 1000000
$regfile = "attiny26.dat"
Config Porta = Output
Config Portb = Output
Config Pina.3 = Input

Dim I As Byte
Dim Num1 As Byte
Dim Num2 As Byte
Dim Answer As Word

'*****************************************
Num1 = 6
Num2 = 9
Answer = 0
Do
  Answer = Answer + Num1
  Decr Num2
Loop Until Num2 = 0

'*****************************************
Num1 = 6
Num2 = 9
Answer = 0
For I = 0 To Num2
  Answer = Answer + Num1
Next

'*****************************************
Num1 = 6
Num2 = 9
Answer = 0
For I = Num2 To 0 Step -1
  Answer = Answer + Num1
Next

'*****************************************
Num1 = 6
Num2 = 9
Answer = 0
While Num2 > 0
  Answer = Answer + Num1
  Decr Num2
Wend
End
```

In BASCOM there are several control mechanisms to manage loops.

If you copy this code into BASCOM-AVR, then save it and compile it you can try it out using the simulator (F2).

Do-Loop Until…

For-Next…
this requires another variable to act as the loop counter, and can either count up or count down.

While – Wend

When you run this program you will find that two of them work correctly and two do not! You need to understand which and fix them; so watch carefully the values of the variables in the simulator and fix the two that need fixing.

Multiplication of very large numbers

The previous code is OK for small to medium size problems however there are much more efficient algorithms; here are 2 alternatives.

'Peasant' Multiplication 75 x 41

75 41
37 ~~82~~
18 ~~164~~
9 ~~328~~
4 ~~656~~
2 ~~1312~~
1 2625
3075

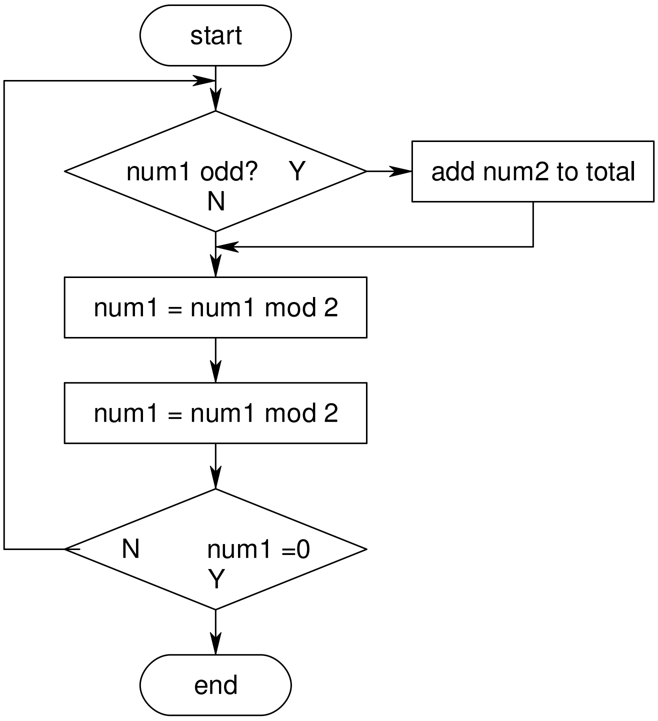
Write down the Algorithm:

Divide the first number by 2 (ignore remainder) and multiply the second number by 2. If the second number is odd add it to the total. Keep doing this process until after the first number is 1.

What variables will be needed:

Num1
Num2
Total

Flowchart:



Program:

```
' PeasantMultiplicationV1.bas

$crystal = 1000000
$regfile = "attiny26.dat"

Config Porta = Output
Config Portb = Output

Dim Temp As Word
Dim Num1 As Word
Dim Num2 As Word
Dim Answer As Word

Num1 = 16
Num2 = 39
Answer = 0

Do

   Temp = Num1 Mod 2
   If Temp = 1 Then Answer = Answer + Num2

   Num1 = Num1 / 2

   Num2 = Num2 * 2

Loop Until Num1 = 0

End
```

Long Multiplication 41,231 x 3,1231

```
       41,321
      x 3,131
       41,321
    1,239,630
    4,132,100
  123,963,000
  129,376,051
```

Write down the Algorithm:

What variables will be needed:

Flowchart:

Algorithm exercises

1. A factory fills drink bottles; it has a machine that puts the drink bottles in to cartons and full cartons onto pallets.
1A. Design an algorithm and flowchart that counts 12 bottles into each carton and keeps track of the number of cartons.
1B. Extend this in a second algorithm and flowchart that tracks the number of bottles and the number of cartons, when number of cartons is over 64 then increase the number of pallets.

2.
A program marks test scores and gives grades of N, A, M, or E based upon the following scores 0% to 33% = N, 34% to 55% = A, 56% to 83% = M 83% to 100% = E
Write the algorithm and draw the flowchart for this process.

3.
Design an algorithm and flowchart for a program that gets a player to guess a random number from 1 to 1000.
If correct, then display the number of guesses and start again
If incorrect then give as too high' or 'too low'
When the number of guesses goes over 8 the player loses

4.
4A. a golf course watering system monitors the time and moisture level of the ground and waters the grass in the early evening if it is needed.
4B. the watering system comes on for 30 minutes then waits 60 minutes to measure the moisture level and comes on for a second watering if it is below a fixed level.

5.
Design an algorithm and flowchart for a program that calculates powers eg. 25 = 32 (use only addition and loops)

LCD programming exercises.

These exercises will require you to manipulate the display, manipulate text, manipulate numbers. And become familiar with the use of loops to get things done.
You need to save each version of the program separately e.g wassup_b.bas, wassup_p.bas, wassup_a.bas.

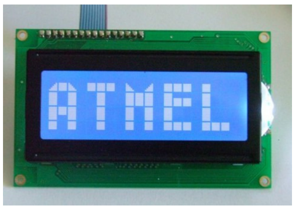
Basic: put 'wassup' on the display
Proficient: Have 'wassup' scroll around the screen continuously
Advanced: Have the 6 letters of 'wassup' appear spread out over the display and then after a brief delay move in towards the centre and in order.

Basic: calculate 2^8 and display it
Proficient: for n from 1 to 25, display 2^n on the screen, wait for 1 sec and then do the next number
Advanced: Write you own code to calculate the square root of the answer for each of the above answers

Basic: Display a static weather report for Auckland on the LCD screen
Proficient: Do graphics for sunny, cloudy, wet, and snowy for your weather report, that flash on the screen, these graphics should be larger than a single lcd square, perhaps 2/3 lines x 4squares
Advanced: Scroll the message on and off the display and have the graphics flash for a while, then the weather report scrolls back on again.

Basic: Display 2 random numbers between 2,000 and 99,000
Proficient: repeat this process continuously, and also subtract the smaller from the larger number and display the answer, have a 3 second delay between each new calculation
Advanced: Scroll the results off the display 0.5 seconds after the calculation

Basic: Create 4 different pacman graphics: one pacman mouth open, one pacman mouth closed, one a target and the last the target exploding
Proficient: Have the pacman move around the screen these, staying on each square for only 0.5 seconds.
Advanced: Generate a random location on the LCD and place the target there, have the pacman move around the screen and when it lands on the target the target explodes and the pacman moves on around the rest of the screen

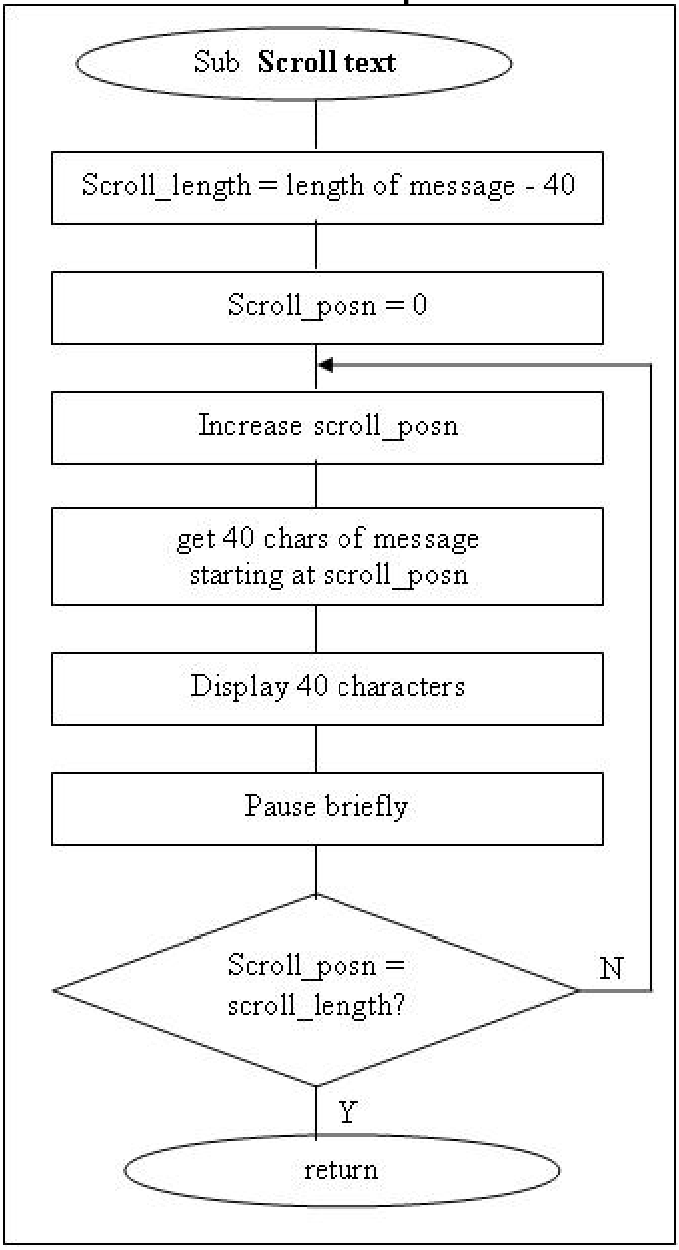


Basic: create 'TCE' in one large font that covers all four lines of the lcd
Proficient: flash the message on the screen three times, 1 second on then 1 second off after that have it stay on for 12 seconds then repeat the 3 flashes.
Advanced: Have this text scroll in from the right and out through the left

Scrolling Message Assignment

AIM: students will be able to manipulate text in Bascom.

An alphanumeric (text) LCD is a very common output device used with microcontrollers however they have limited screen size so a longer message must be either split up and shown page by page or scrolled across the screen. In this assignment you will scroll a message across the screen. The message will be an information message regarding a news item or weather forecast up to 200 characters in length.


Sub Scroll text
Scroll_length = length of message - 40
Scroll_posn = 0
Increase scroll_posn
get 40 chars of message starting at scroll_posn
Display 40 characters
Pause briefly
Scroll_posn = scroll_length?
N
Y
return

```
‘Declare Variables
Dim message as string * 200
Dim scroll_length as byte
Dim scroll_posn as byte
Dim forty_chars as string * 40

‘Initialise Variables
Message = “ the weather today will be …..”


Scroll_text:
    Scroll_length = len(message)
    If Scroll_length > 40 then
        Scroll_length = scroll_length – 40
    End if
    Scroll_posn = 0
    While scroll_posn < scroll_length
        Incr scroll_posn
        Forty_chars =mid(message,scroll_posn,40)
        Locate 1,1
        Lcd forty_chars
        Waitms 150
    Wend
Return
```

1. Change the While-Wend to a Do-Loop-Until structure

2. Change the While-Wend to a For-Next

This routine scrolls the complete message once and then returns to the main loop, it is a very long routine taking 150mS x the length of the message to complete. This makes it almost useless as part of a larger program.

This routine needs to be replaced so that it returns to the main loop after each shift of the message. This would make the routine a general purpose routine that could be used as part of a larger program.
Of course a delay will be necessary but a loop counter rather than waitms will be needed.

There are many useful commands in Bascom for manipulating text. Text in microcontrollers is stored as codes using ASCII, each character taking up 1 byte of RAM. One subroutine to scroll text might look like this.

Strings Assignment

```
'------------------------------------------------------------------
' 6. Hardware Setups
' setup direction of all ports
Config Porta = Output                 'LEDs on portA
Config Portb = Output                 'LEDs on portB
Config Portc = Output                 'LEDs on portC
Config Portd = Output                 'LEDs on portD
'config inputs
Config Pina.0 = Input               ' ldr
Config Pind.2 = Input               'switch A
Config Pind.3 = Input               'switch B
Config Pind.6 = Input               'switch C
Config Pinb.1 = Input               'switch D
Config Pinb.0 = Input               'switch E
'LCD
Config Lcdpin = Pin , Db4 = Portc.4 , Db5 = Portc.5 , Db6 = Portc.6 , Db7 =
Portc.7 , E = Portc.3 , Rs = Portc.1
Config Lcd = 40 * 2                 'configure lcd screen
'ADC
'Config Adc = Single , Prescaler = Auto , Reference = Internal
'Start Adc

' 7. Hardware Aliases
Sw_a Alias Pinb.0
Sw_b Alias Pinb.1
Sw_c Alias Pind.2
Sw_d Alias Pind.3
Sw_e Alias Pind.6

' 8. initialise ports so hardware starts correctly
Porta = &B11111100                  'turns off LEDs ignores ADC inputs
Portb = &B11111100                  'turns off LEDs ignores switches
Portc = &B11111111                  'turns off LEDs
Portd = &B10110011                  'turns off LEDs ignores switches
Cls                           'clear lcd screen
Cursor On Noblink
'------------------------------------------------------------------
' 9. Declare Constants
'------------------------------------------------------------------
' 10. Declare Variables
Dim Mix As Byte
Dim Firstname As String * 12
Dim Middlename As String * 12
Dim Lastname As String * 12
Dim Fullname As String * 40
' 11. Initialise Variables
Mix = 0
Firstname = "Edgar"
Middlename = "Alan"
Lastname = "Poe"
Fullname = ""
```

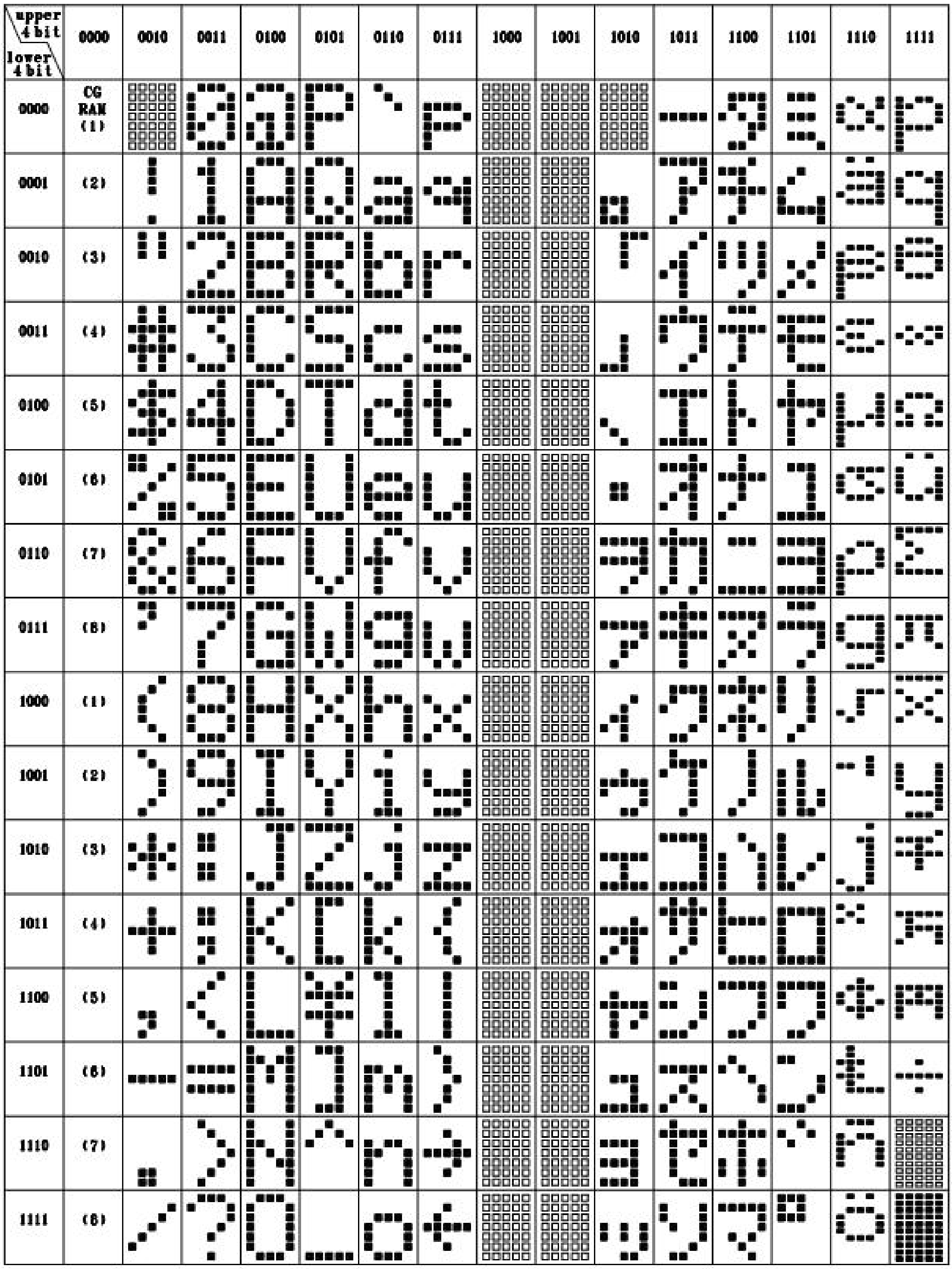
```
'-----------------------------------------------------------------
' 12. Program starts here
Cls
Gosub Welcome
Do
  Debounce Sw_a , 0 , Welcome , Sub
  Debounce Sw_b , 0 , Mixup , Sub
Loop
End                              'end program

'-----------------------------------------------------------------
' 13. Subroutines
Welcome:
  Cls
  Lcd "Welcome"
  Lowerline
  Lcd Chr(126) : Lcd "to strings" : Lcd Chr(127)
Return

Mixup:
  Incr Mix
  Select Case Mix:
    Case 1 : Fullname = Firstname + " " + Middlename + " " + Lastname
    Case 2 : Fullname = Middlename + " " + Lastname + " " + Firstname
    Case 3 : Fullname = Lastname + " " + Firstname + " " + Middlename
    Case 4 : Fullname = Mid(fullname , 10 , 5)
    Case 5 : Fullname = Lastname + "," + Left(firstname , 2)
    Case 6 : Fullname = Version(1)
    Case 10 : Mix = 0
  End Select
  Cls
  Lcd Fullname
Return
```

Insert case statements 7,8 and 9 above. From the help file find out how to use and then add them to this program 3 of the following

INSTR LCASE LEN LOOKUPSTR LTRIM RIGHT RTRIM SPACE SPC STR STRING TRIM UCASE

ASCII Character Table

What is available inside the LCD using the command **LCD CHR(__)**



upper 4 bit / lower 4 bit	0000	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0000	CG RAM (1)		0	@	P	`	p				ｰ	ﾀ	ﾐ	α	p
0001	(2)	!	1	A	Q	a	q			｡	ｱ	ﾁ	ﾑ	ä	q
0010	(3)	"	2	B	R	b	r			｢	ｲ	ﾂ	ﾒ	β	θ
0011	(4)	#	3	C	S	c	s			｣	ｳ	ﾃ	ﾓ	ε	∞
0100	(5)	$	4	D	T	d	t			､	ｴ	ﾄ	ﾔ	μ	Ω
0101	(6)	%	5	E	U	e	u			･	ｵ	ﾅ	ﾕ	σ	ü
0110	(7)	&	6	F	V	f	v			ｦ	ｶ	ﾆ	ﾖ	ρ	Σ
0111	(8)	'	7	G	W	g	w			ｧ	ｷ	ﾇ	ﾗ	g	π
1000	(1)	(	8	H	X	h	x			ｨ	ｸ	ﾈ	ﾘ	√	x̄
1001	(2)	)	9	I	Y	i	y			ｩ	ｹ	ﾉ	ﾙ	⁻¹	y
1010	(3)	*	:	J	Z	j	z			ｪ	ｺ	ﾊ	ﾚ	j	千
1011	(4)	+	;	K	[	k	{			ｫ	ｻ	ﾋ	ﾛ	ˣ	万
1100	(5)	,	<	L	¥	l	|			ｬ	ｼ	ﾌ	ﾜ	¢	円
1101	(6)	-	=	M	]	m	}			ｭ	ｽ	ﾍ	ﾝ	Ⱡ	÷
1110	(7)	.	>	N	^	n	→			ｮ	ｾ	ﾎ	ﾞ	ñ	
1111	(8)	/	?	O	_	o	←			ｯ	ｿ	ﾏ	ﾟ	ö	█

ASCII stands for ____

ASCII Assignment

1. Copy the following code into BASCOM
2. Compare the datasheet for the LCD with the characters that actually appear on your LCD.
3. Write the code for the **decrementcode** subroutine

```
'------------------------------------------------------------------
' 1. Title Block
' Author: B.Collis
' Date: 1 June 2005
' File Name: LCDcharactersV1.bas
'------------------------------------------------------------------
' 2. Program Description:
' everytime btn is pressed the character on the lcd changes
' highlights the use of the ASCII code
' 3. Hardware Features:
' LEDS
' LDR, Thermistor on ADC
' 5 switches
' LCD
' 4. Program Features
' do-loop to keep program going forever
' debounce to test switches
' if-then-endif to test variables
'------------------------------------------------------------------
' 5. Compiler Directives (these tell Bascom things about our hardware)
$crystal = 8000000 'the speed of the micro
$regfile = "m8535.dat" 'our micro, the ATMEGA8535-16PI
'------------------------------------------------------------------
' 6. Hardware Setups
' setup direction of all ports
Config Porta = Output 'LEDs on portA
Config Portb = Output 'LEDs on portB
Config Portc = Output 'LEDs on portC
Config Portd = Output 'LEDs on portD
'config inputs
Config Pind.2 = Input 'switch A
Config Pind.3 = Input 'switch B
Config Pind.6 = Input 'switch C
Config Pinb.1 = Input 'switch D
Config Pinb.0 = Input 'switch E
'LCD
Config Lcdpin = Pin , Db4 = Portc.4 , Db5 = Portc.5 , Db6 = Portc.6 , Db7 = Portc.7 , E =
Portc.3 , Rs = Portc.2
Config Lcd = 40 * 2 'configure lcd screen
' 7. Hardware Aliases
Sw_a Alias Pinb.0
Sw_b Alias Pinb.1
Sw_c Alias Pind.2
Sw_d Alias Pind.3
```

```
Sw_e Alias Pind.6

' 8. initialise ports so hardware starts correctly
Porta = &B11111100 'turns off LEDs ignores ADC inputs
Portb = &B11111100 'turns off LEDs ignores switches
Portc = &B11111111 'turns off LEDs
Portd = &B10110011 'turns off LEDs ignores switches
Cls 'clear lcd screen
'---------------------------------------------------------------
' 9. Declare Constants
'---------------------------------------------------------------
' 10. Declare Variables
Dim Code As Byte
Dim State As Byte
' 11. Initialise Variables
Code = 0
State = 0
'---------------------------------------------------------------
' 12. Program starts here
Do
 Sw_a , 0 , Swa_press , Sub
Debounce Sw_b , 0 , Swb_press , Sub
If State = 0 Then Gosub Intro
If State = 1 Then Gosub Increasecode
If State = 2 Then Gosub Decreasecode
If State = 4 Then Gosub Waiting
Loop
End 'end program

'---------------------------------------------------------------
' 13. Subroutines
Intro:
Lcd "ASCII codes"
Lowerline
Lcd "btn A incrs code"
Return

Waiting:
'do nothing
Return

Increasecode:
If Code < 255 Then 'max value is 255
Incr Code
Else
Code = 0 'if > 255 reset to 0
End If
Cls
Lcd Code : Lcd " " : Lcd Chr(code)
State = 4
Return
```

```
Decreasecode:
'write your code here
```

```
Return

Swa_press:
State = 1
Return

Swb_press:
State = 2
Return
```

Time

Bascom has built in functions for managing the time and date. These require a 32.768Khz crystal to be connected to the micro.

```
‘SoftClockDemoProgam1.bas
‘32.768kHz crystal is soldered onto C.6 and C.7

$crystal = 8000000
$regfile = "m8535.dat"

Config Porta = Output
Config Portb = Output
Config Portc = Output
Config Portd = Output
Config Lcdpin = Pin , Db4 = Portc.2 , Db5 = Portc.3 , Db6 = Portc.4 , Db7 = Portc.5 , E
= Portc.1 , Rs = Portc.0
Config Lcd = 20 * 4

Enable Interrupts                       '1activate internal timer

Config Date = Mdy , Separator = /       '2
Config Clock = Soft                     '3

Date$ = "06/24/09"                      '4 string to hold the date
Time$ = "23:59:56"                      '5 string to hold the time

Cls
Cursor Off

Do
   Locate 1 , 1
   Lcd Time$ ; " " ; Date$              '6 display the two strings
Loop
End
```

```
‘SoftClockTrialDemoProgam2.bas
$crystal = 8000000
$regfile = "m8535.dat"

Config Porta = Output
Config Portb = Output
Config Portc = Output
Config Portd = Output
Config Lcdpin = Pin , Db4 = Portc.2 , Db5 = Portc.3 , Db6 = Portc.4 , Db7 = Portc.5 , E
= Portc.1 , Rs = Portc.0
Config Lcd = 20 * 4
Grnled Alias Portd.7

Enable Interrupts
Config Date = Mdy , Separator = /
Config Clock = Soft , Gosub = Sectic        '1 every second do sectic Bweekday As Byte
Dim Strweekday As String * 10               '2 days of week
Date$ = "06/24/09"
Time$ = "23:59:56"

Cls
Cursor Off
Do
   Locate 1 , 1
   Lcd Time$ ; " " ; Date$
   Locate 2 , 1
   Lcd _sec ; _min ; _hour ; _day ; _month ; _year        '3

   Bweekday = Dayofweek()                   '4
   Strweekday = Lookupstr(bweekday , Weekdays)        '5
   Locate 3 , 1
   Lcd Bweekday ; " = " ; Strweekday        '6
   ' DayOfWeek, DayOfYear, SecOfDay, SecElapsed, SysDay, SysSec ,SysSecElapsed     7
Loop
End

Sectic:                                     '8
   Toggle Grnled                            '9
Return

Weekdays:                                   '10
Data "Monday" , "Tuesday" , "Wednesday" , "Thursday" , "Friday" , "Saturday" , "Sunday"


‘Extend the code to display the month
```

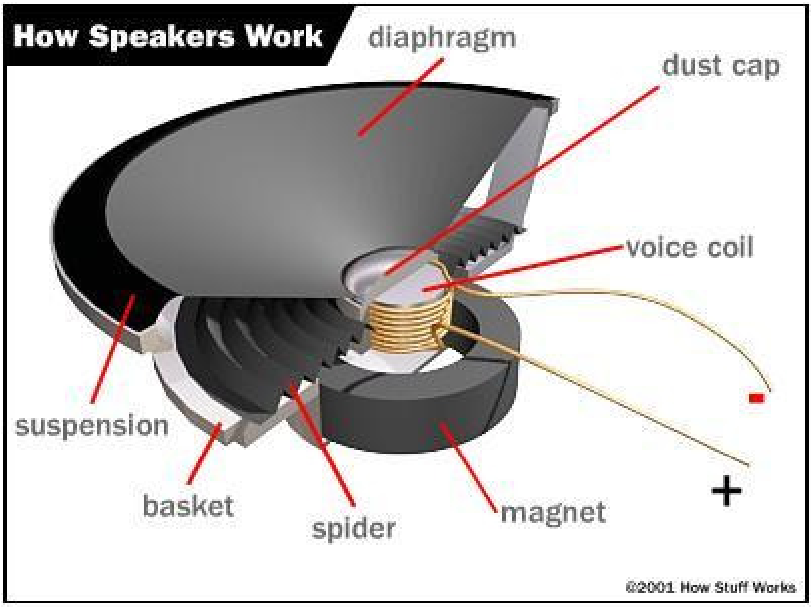
```
‘SoftClockTrialDemoProgam3.bas
$crystal = 8000000
$regfile = "m8535.dat"

Config Porta = Output
Config Portb = Output
Config Portc = Output
Config Portd = Input                    '1

Redsw Alias Pind.2                      '2

Config Lcdpin = Pin , Db4 = Portc.2 , Db5 = Portc.3 , Db6 = Portc.4 , Db7 = Portc.5 , E
= Portc.1 , Rs = Portc.0
Config Lcd = 20 * 4

Enable Interrupts

Config Date = Mdy , Separator = /
Config Clock = Soft

Date$ = "06/24/09"
Time$ = "23:59:56"

Cls
Cursor Off

Do
   Debounce Redsw , 0 , Red , Sub       '3
   Locate 1 , 1
   Lcd Time$ ; " " ; Date$
Loop
End

Red:                                    '4
   Incr _min
   If _min > 59 then _min = 0           ‘5 stop overflow
Return
```

```
'SoftClockTrialDemoProgam4.bas
$crystal = 8000000
$regfile = "m8535.dat"

Config Porta = Output
Config Portb = Output
Config Portc = Output
Config Portd = Input

Redsw Alias Pind.2

Config Lcdpin = Pin , Db4 = Portc.2 , Db5 = Portc.3 , Db6 = Portc.4 , Db7 = Portc.5 , E
= Portc.1 , Rs = Portc.0
Config Lcd = 20 * 4

Enable Interrupts

Config Date = Mdy , Separator = /
Config Clock = Soft

Date$ = "06/24/09"
Time$ = "23:59:56"

Cls
Cursor Off

Do
   If Redsw = 0 Then Gosub Red          '1 your own simple debounce
   Locate 1 , 1
   Lcd Time$ ; " " ; Date$
Loop
End

Red:
   Waitms 25                            '2 wait for contact bounce
   Do                                   '3 wait for switch release
   Loop Until Redsw = 1
   Incr _min
   If _ min > 59 then _min=0
Return
```

Sounding Off

How is sound made?

A speaker makes sound by moving a paper diaphragm (the speaker cone) back and forth rapidly. This vibrates the air which vibrates our ear drum causing us to hear the sound.

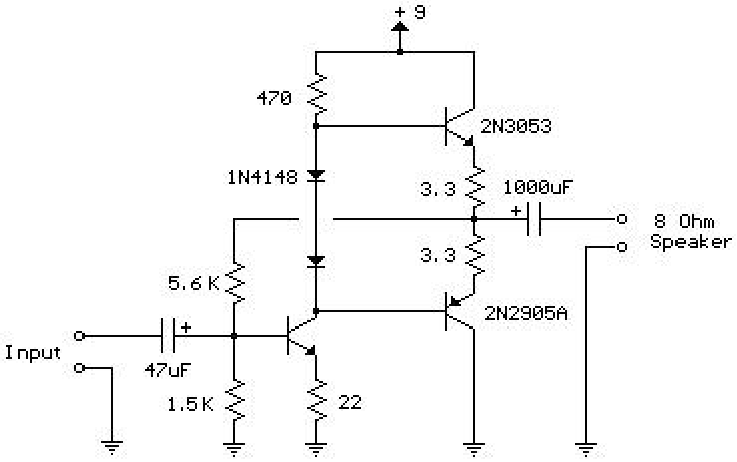


When the voltage to a speaker is switched on and off or reversed the speaker diaphragm will move in and out. The greater the voltage the greater the vibrations will be and the louder the sound will seem.

Attaching the speaker to a microcontroller

This uses one of the outputs of the microcontroller to drive a speaker. The speaker however is typically only 8 ohms and if we connect it directly between a port pin and 5V it will draw too much current and could damage the microcontroller's internal circuits or burn out the speaker (or both).

We can use a transistor circuit as a driver/amplifier circuit.



http://ourworld.compuserve.com/homepages/Bill_Bowden/page8.htm#amp.gif

(or use a dedicated amplifier chip like the LM386)

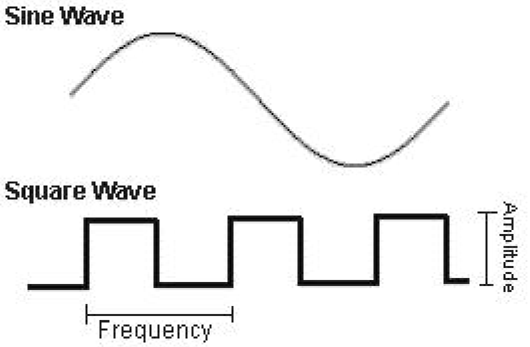
Code to make a siren

```
'----------------------------------------------------------------
' 6. Hardware Setups
Config Timer1 = Timer , Prescale = 1
On Ovf1 Timer1_isr                                  ‘at end of count do this subroutine
Enable Interrupts                              ‘global interrupt enable
' 7. Hardware Aliases
Spkr Alias Portb.2                                  ‘speaker is on this port
'----------------------------------------------------------------
' 9. Declare Constants
Const Countfrom = 55000                  ‘use constants to aid program understanding
Const Countto = 64500
Const Countupstep = 100
Const Countdnstep = -100
Const Countdelay = 3
Const Delaybetween = 20
Const numbrSirens = 10
'----------------------------------------------------------------
' 10. Declare Variables
Dim Count As Word                        ‘use useful names to help program understanding
Dim Sirencount As Byte
Dim Timer1_preload As Word
Timer1 = Timer1_preload
'----------------------------------------------------------------
' 12. Program starts here
Do
   Gosub Makesiren
   Wait 5
Loop
End
'----------------------------------------------------------------
' 13. Subroutines
Makesiren:
  Enable Timer1                                           'sound on
  For Sirencount = 1 To numbrSirens                       ‘how many siren cycles to do
   For Count = Countfrom To Countto Step Countupstep      ‘rising pitch
     Timer1_preload = Count                               ‘pitch value
     Waitms Countdelay                                    ‘length of each tone
   Next
   For Count = Countto To Countfrom Step Countdnstep      ‘falling pitch
     Timer1_preload = Count                               ‘pitch value
     Waitms Countdelay                                    ‘length of each tone
   Next
   Waitms Delaybetween                                    ‘delay between each cycle
  Next
  Disable Timer1                                          'sound off
Return
'----------------------------------------------------------------
' 14. Interrupt service routines (isr)
Timer1_isr:
  'if the timer isnt preloaded it will start from 0 after an interrupt
  Timer1 = Timer1_preload
  Toggle Spkr
Return
```

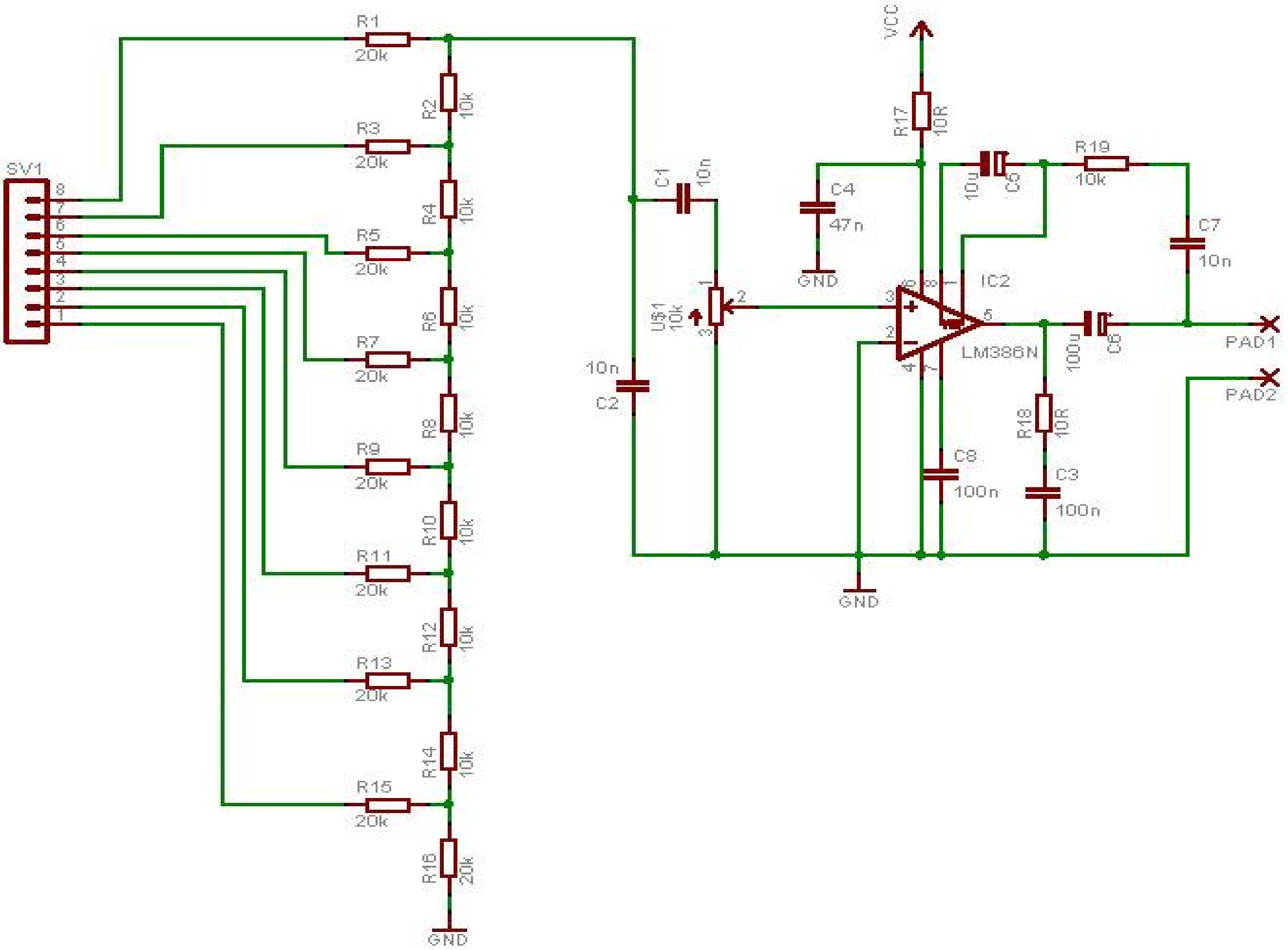
High tech (better) ways of generating sound

The method used above is not a nice way to generate sound, i.e. making a square wave that switches a speaker rapidly from on to off. Signals that produce good quality sound are sine waves not square waves.

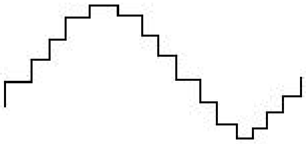
The difference between the two is that a sine wave varies smoothly in voltage over time.



To generate a reasonable sine wave from a computer we use a step process, the signal is increased in voltage steps using a DAC (Digital to Analogue) Converter.



The **R2R ladder network** is used as a digital to analogue converter, turning on combinations of resistors causes the voltage to step up and down, the output voltage will look a little like the waveform below (however it will have 256 different steps).



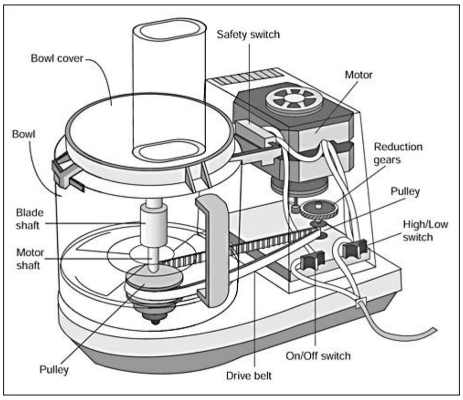
This stepped wave is a much better approximation to a sine wave than the square wave. The smaller the steps and the more there are of them the better the sound.

System and Software Design

Understanding how simple systems work

A product or device is not just a collection of components, it is much more, the inventor of the device didn’t just combine some bits together they created something when they thought of it. They envisaged it as a system where all the parts have a unique purpose and function to make the product complete.

A first example is a food processor.



To analyse the system

1. Draw a system diagram
2. Identify and describe all the inputs and outputs of the system
    a. Motor – half/full speed
    b. power switch - on/off
    c. speed switch – high/low
    d. bowl safety switch – on/off
3. Describe in words and drawings how these interact with each other, use logic descriptors such as AND,OR and NOT.

Here are some possible descriptions. Are they all correct? Which one is best? Why?

1. The motor goes when the safety switch is closed AND the power switch is on AND the speed switch is either position.
2. The motor runs at half speed if the speed switch is in low AND safety switch is on AND the main switch is on.
3. The motor runs at full speed if the safety switch is on AND the main switch is on.

A toaster is another good example of a system.

1. Draw a system diagram

2. Identify all the parts of the toaster
    a.
    b.
    c.
    d.
    e.
    f.

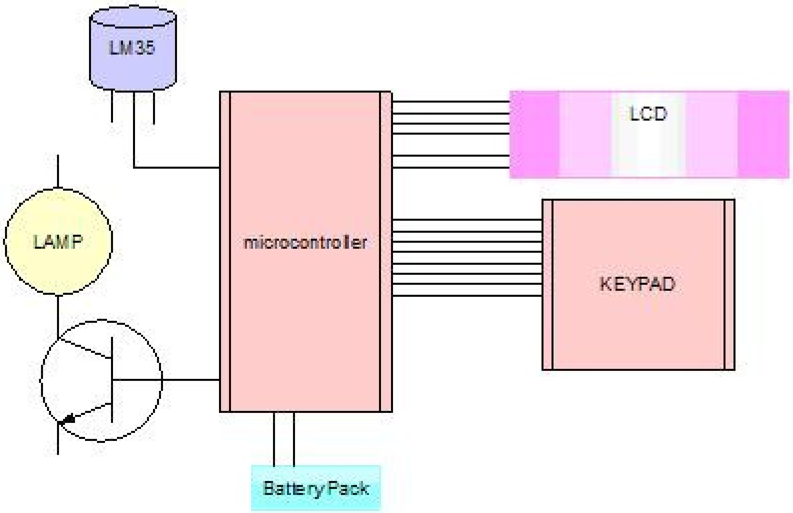
2. Describe how the parts of the system interact with each other

Problem Decomposition Example

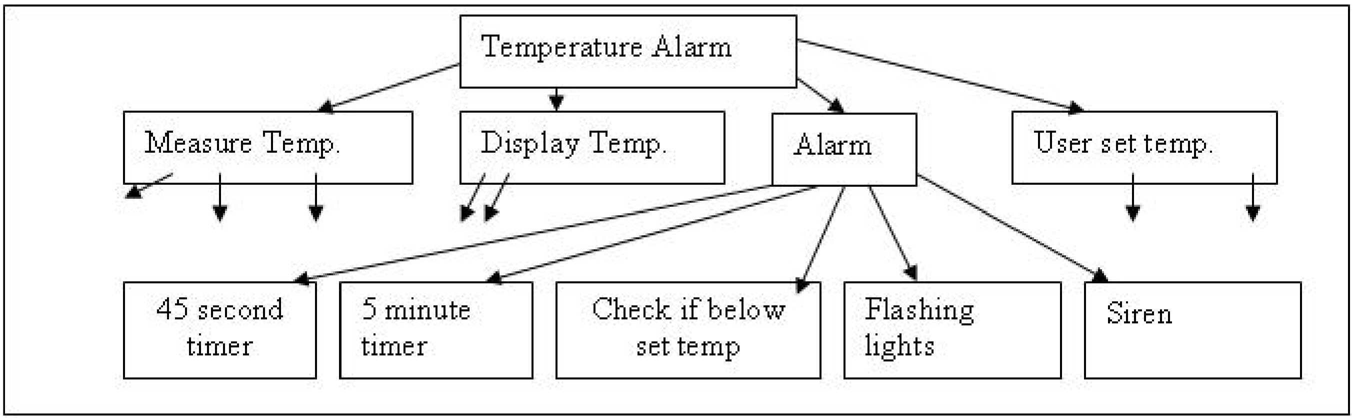
Here is a more complex system that we will develop the software for

1. Define the problem in writing (a brief), e.g.

   The system will monitor temperature inside a room and display it on an LCD, an alarm will sound for 45 seconds if it goes below a user preset value. A light will stay flashing until reset. If not reset within 5 minutes the alarm will retrigger again. If the temperature rises at any time then the alarm will automatically reset.
2. Draw a system block diagram of the hardware



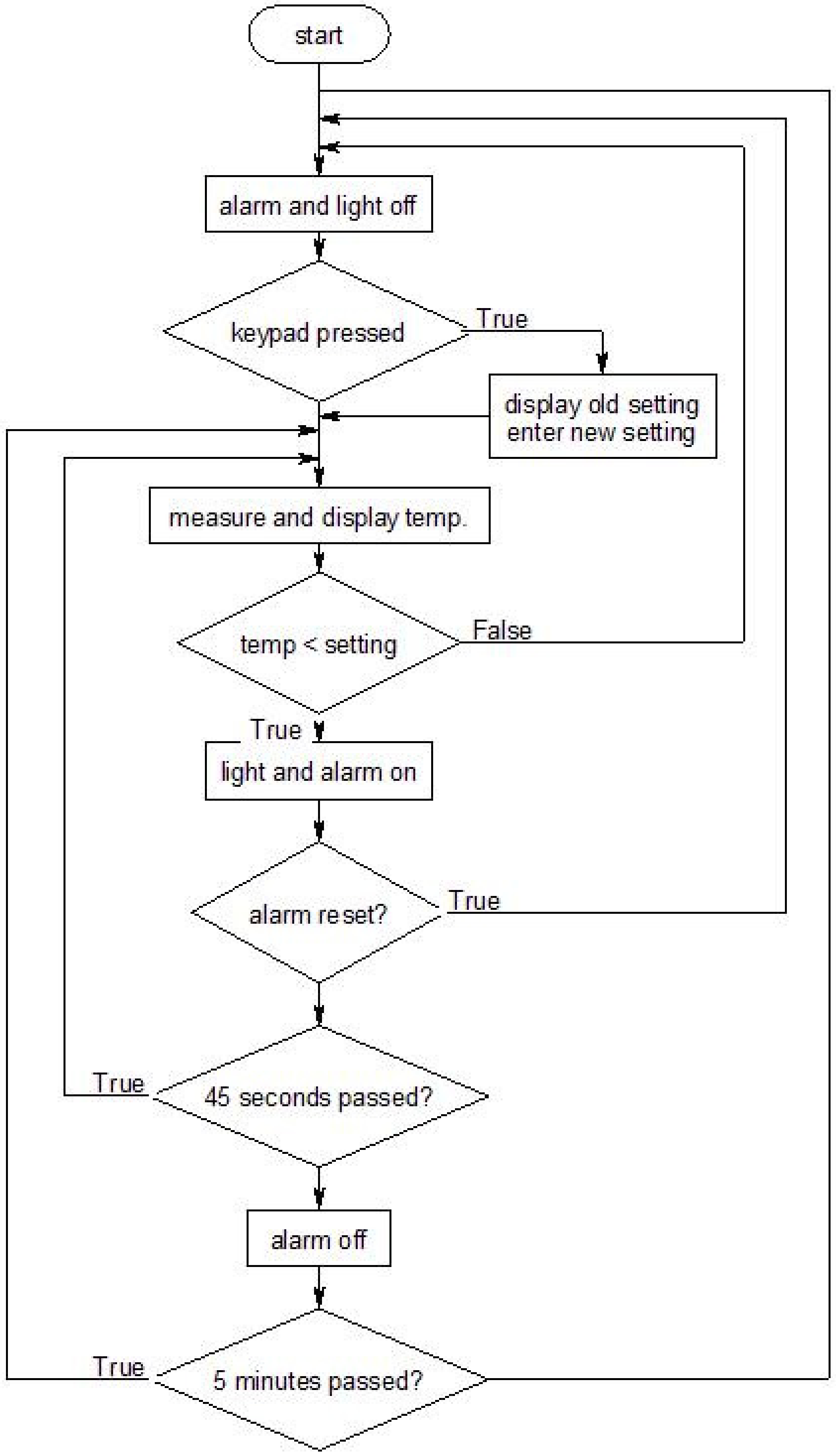
3. Research and identify the interfaces to the system e.g.
   a. An LM35 temperature sensor
   b. A 2 line x 16 character LCD
   c. A flashing light that can be seen from 6 meters away
   d. A speaker with sufficient volume to be heard in the next room
   e. A keypad for entering values
4. Draw interface circuits for each of the interfaces
5. build the interfaces one at a time, design test subroutines for them and test them thoroughly
6. Problem decomposition: break the system down into successive sub-systems, until the sub-systems are trivial in nature. In this diagram the Alarm function has been broken down into 4 sub parts of which one has been broken down further.



7. Design the logic flow for the solution using flow or state diagrams

   Test your logic thoroughly! If you miss an error now you will take 19.2 times longer to finish!

Here is one flow chart for the temperature system.



This is a small but very complex flowchart and it is not a good solution for a number of reasons:

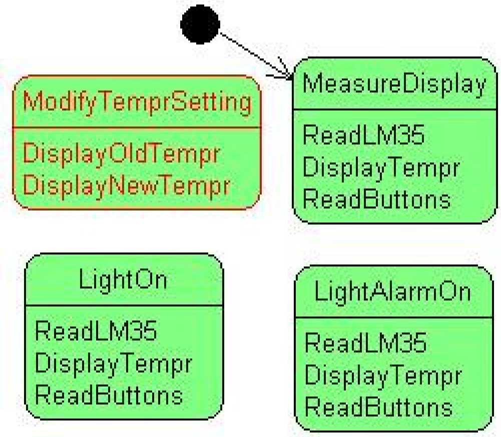
A. It is difficult to manage all the relationships to get the logic absolutely correct, it took a while to think it through and it may not be exactly right yet!
B. It is very difficult to write a program to match this flowchart without the use of goto statements which are poor programming practice and not a feature of the higher level languages you will meet in the future.
C. Once the code is written it is difficult to maintain this code as it lacks identifiable structure

It is OK to use flowcharts for small problems but if a flowchart has more than 3 loops or the loops cross over each other use an alternative method!

Statecharts

Statecharts are a better solution. First think about the finished device and identify the different states of operation it will be in and secondly identify the conditions or events that will cause one state to transition (change) to another.

Here are the 4 states for the temperature controller and a diagram representation of it (using Umlpad) The black circle indicates the stating state.



State 1: measure and display temperature
Conditions: temp < setting
keypad to change setting

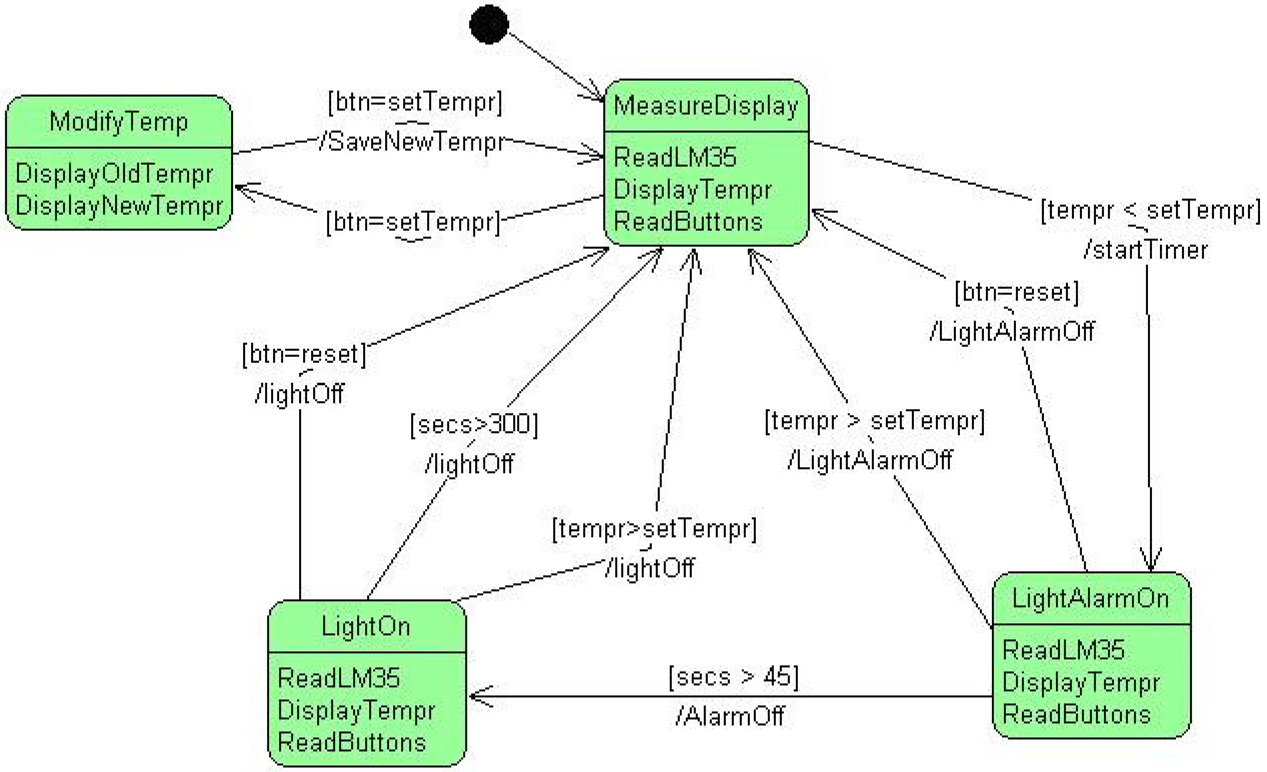
State 2: light and alarm are on
Conditions: reset pressed
temperature >= setting
45 second time out

State 3: light on
Conditions: reset pressed
temperature >= setting
5 minute time out

State 4: modify temp setting
Conditions: finished changing setting

Each state includes the names of subroutines that will be called to do different things. It is a good idea not to put code into the state even if it is trivial, so that structure is easily identifiable. Each subroutine may require a flowchart to plan it or even another statechart.

Here is a statechart diagram of this problem with the transitions and the conditions that cause the transition to occur. A condition is in square brackets [], followed by any actions you want the program to take on the way to the next state. An action is the name of another subroutine.



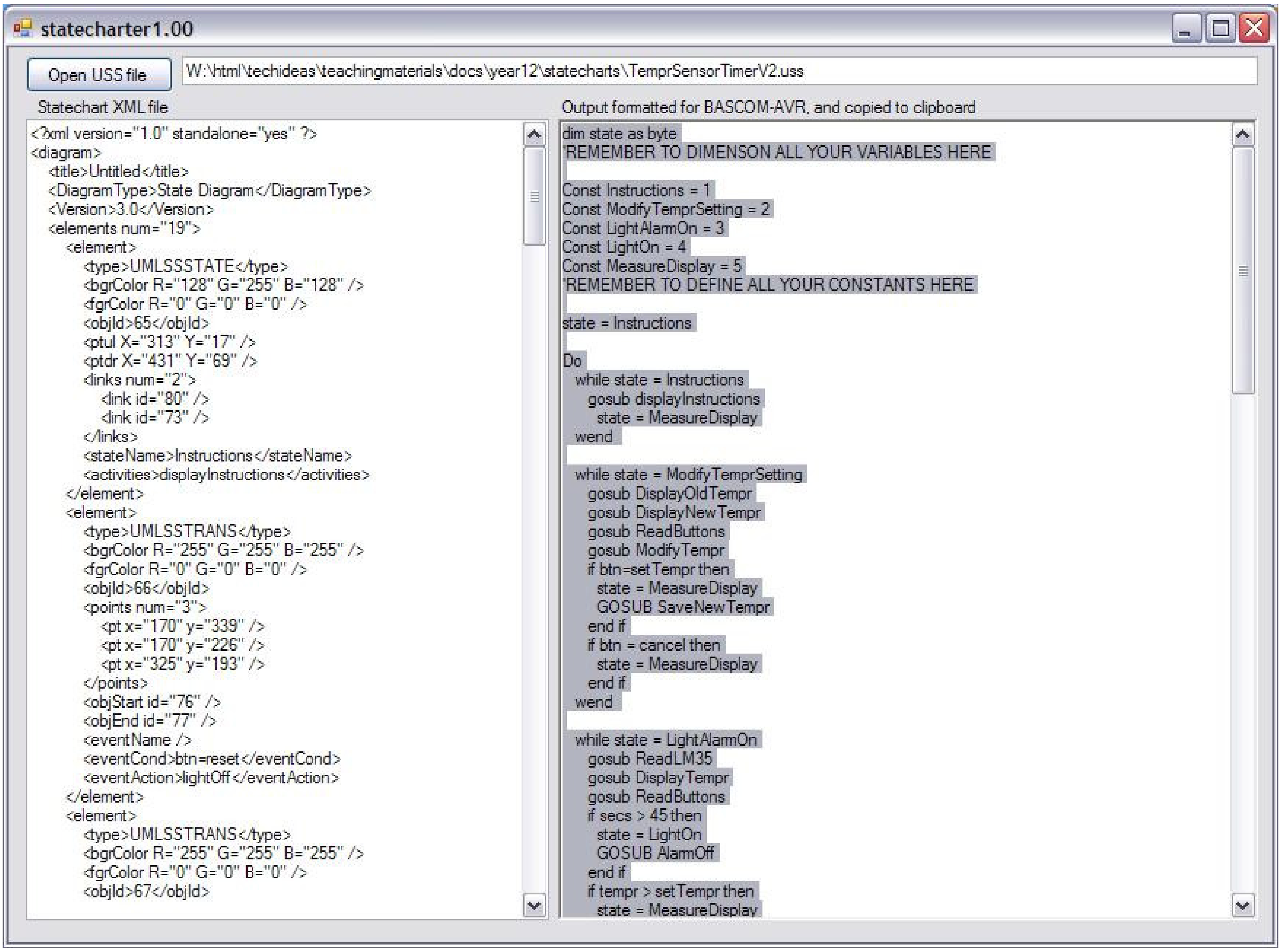
This style of problem solving overcomes the issues identified relating to flowcharts

A. The relationships between states are easily managed and they logically flow so errors are seen quickly.

B. It is easy to write the code to match this diagram using if-then or while wend statements

C. The code is easily maintained and flows logically when it is written making it easier to remember what you did or for others to read and maintain.

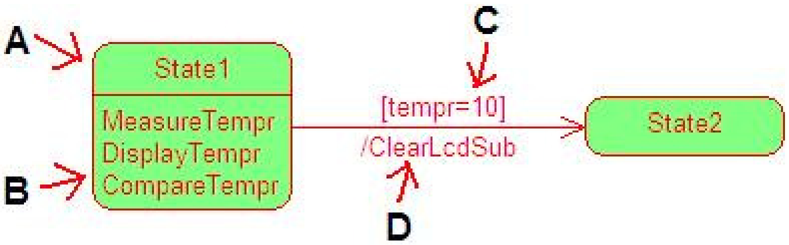
D. If you closely follow the structure using subroutine names then you can use the software I have developed to create the basic structure for your code in BASCOM_AVR.


statecharter1.00
Open USS file
W:\html\techideas\teachingmaterials\docs\year12\statecharts\TemprSensorTimerV2.uss
Statechart XML file
Output formatted for BASCOM-AVR, and copied to clipboard

```
<?xml version="1.0" standalone="yes" ?>
<diagram>
  <title>Untitled</title>
  <DiagramType>State Diagram</DiagramType>
  <Version>3.0</Version>
  <elements num="19">
    <element>
      <type>UMLSSSTATE</type>
      <bgrColor R="128" G="255" B="128" />
      <fgrColor R="0" G="0" B="0" />
      <objId>65</objId>
      <ptul X="313" Y="17" />
      <ptdr X="431" Y="69" />
      <links num="2">
        <link id="80" />
        <link id="73" />
      </links>
      <stateName>Instructions</stateName>
      <activities>displayInstructions</activities>
    </element>
    <element>
      <type>UMLSSTRANS</type>
      <bgrColor R="255" G="255" B="255" />
      <fgrColor R="0" G="0" B="0" />
      <objId>66</objId>
      <points num="3">
        <pt x="170" y="339" />
        <pt x="170" y="226" />
        <pt x="325" y="193" />
      </points>
      <objStart id="76" />
      <objEnd id="77" />
      <eventName />
      <eventCond>btn=reset</eventCond>
      <eventAction>lightOff</eventAction>
    </element>
    <element>
      <type>UMLSSTRANS</type>
      <bgrColor R="255" G="255" B="255" />
      <fgrColor R="0" G="0" B="0" />
      <objId>67</objId>
```

```
dim state as byte
'REMEMBER TO DIMENSON ALL YOUR VARIABLES HERE

Const Instructions = 1
Const ModifyTemprSetting = 2
Const LightAlarmOn = 3
Const LightOn = 4
Const MeasureDisplay = 5
'REMEMBER TO DEFINE ALL YOUR CONSTANTS HERE

state = Instructions

Do
  while state = Instructions
    gosub displayInstructions
     state = MeasureDisplay
  wend

  while state = ModifyTemprSetting
    gosub DisplayOldTempr
    gosub DisplayNewTempr
    gosub ReadButtons
    gosub ModifyTempr
    if btn=setTempr then
     state = MeasureDisplay
     GOSUB SaveNewTempr
    end if
    if btn = cancel then
     state = MeasureDisplay
    end if
  wend

  while state = LightAlarmOn
    gosub ReadLM35
    gosub DisplayTempr
    gosub ReadButtons
    if secs > 45 then
     state = LightOn
     GOSUB AlarmOff
    end if
    if tempr > setTempr then
     state = MeasureDisplay
```

Statecharter is written in C# using SharpDevelop and requires the Microsoft dotnet framework to be installed on the PC; there is no install just run statecharter.exe directly. The statechart file from UMLPAD is an XML file and straight forward to peruse with a text editor. As it follows a very defined format it is not hard to parse to identify the states, transitions etc.



When using UMLPad to create statecharts for conversion using statecharter, you must:
A – name each state without spaces and do not use reserved Bascom words
B – the actions in each state will be calls to subroutines, again no spaces in names and no reserved words
C- When using UMLPad use conditions to trigger transitions, not events, these will appear using if-then statements e.g. if tempr=10
D. If something needs to happen in between states then enter these in the action, these will be calls to subroutines as well, e.g. gosub clearlcdsub

```
Const LightAlarmOn = 1
Const LightOn = 2
Const MeasureDisplay = 3
Const ModifyTemprSetting = 4

Do
  while state = LightAlarmOn
    gosub ReadLM35
    gosub DisplayTempr
    gosub ReadButtons
    if secs > 45 then
      state = LightOn
      GOSUB AlarmOff
    end if
    if tempr > setTempr then
      state = MeasureDisplay
      GOSUB LightAlarmOff
    end if
    if btn=reset then
      state = MeasureDisplay
      GOSUB LightAlarmOff
    end if
  wend

  while state = LightOn
    gosub ReadLM35
    gosub DisplayTempr
    gosub ReadButtons
    if btn=reset then
      state = MeasureDisplay
      GOSUB lightOff
    end if
    if tempr>setTempr then
      state = MeasureDisplay
      GOSUB lightOff
    end if
    if secs>300 then
      state = MeasureDisplay
      GOSUB lightOff
    end if
  wend

  while state = MeasureDisplay
    gosub ReadLM35
    gosub DisplayTempr
    gosub ReadButtons
    if tempr < setTempr then
      state = LightAlarmOn
      GOSUB startTimer
    end if
    if btn=setTempr then
      state = ModifyTemprSetting
    end if
  wend

  while state = ModifyTemprSetting
    gosub DisplayOldTempr
    gosub DisplayNewTempr
    if btn=setTempr then
      state = MeasureDisplay
      GOSUB SaveNewTempr
    end if
  wend
Loop
```

Labels are used for states rather than numbers to facilitate program readability

The state variable is used to manage which subroutines are called

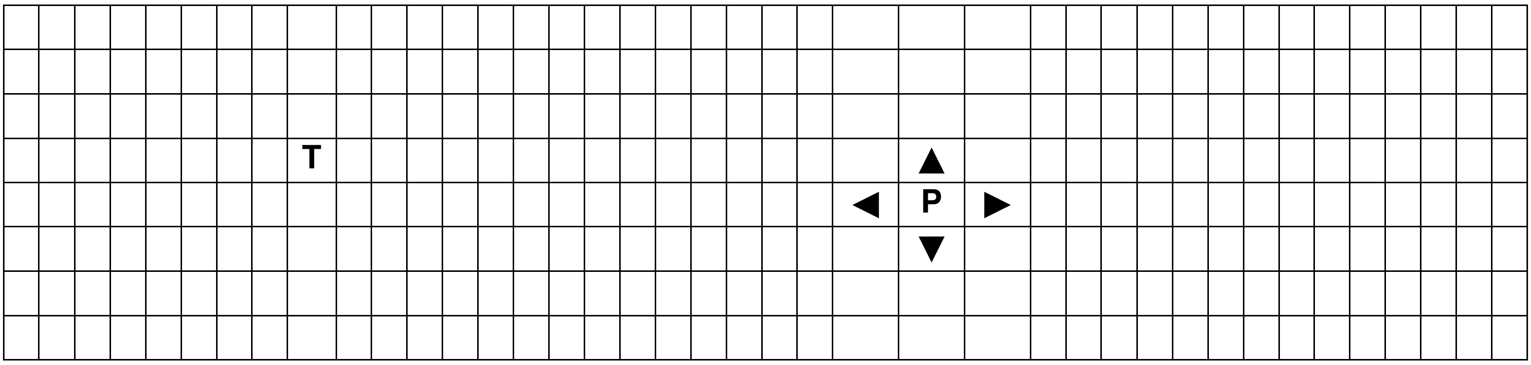
Only change to another state when specific conditions occur.

```
'**********************************
subroutines

ReadLM35:
Return

DisplayTempr:
Return

ReadButtons:
Return

DisplayOldTempr:
Return

DisplayNewTempr:
Return

startTimer:
Return

lightOff:
Return

AlarmOff:
Return

SaveNewTempr:
Return

LightAlarmOff:
Return
```

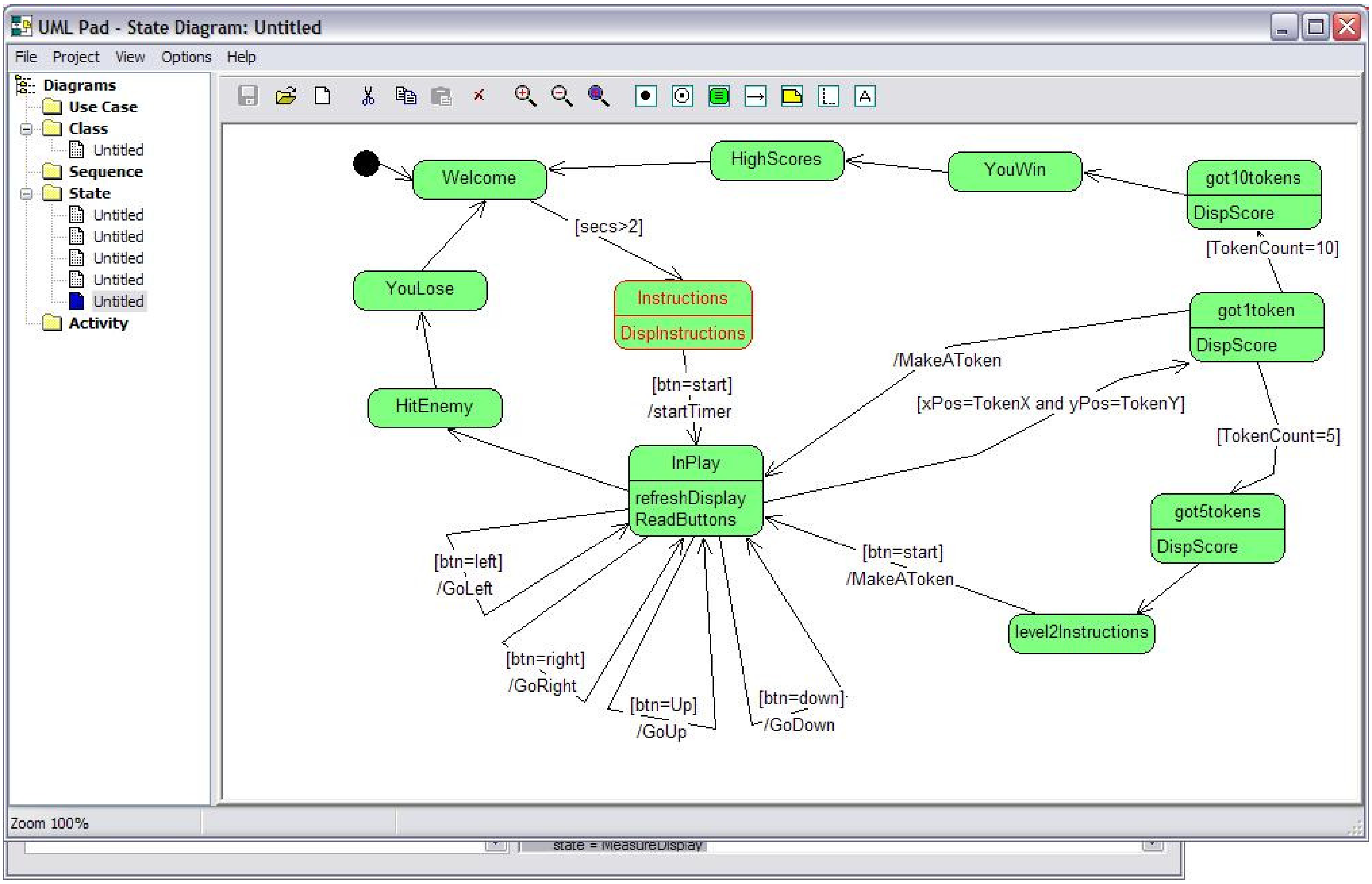
All the rest of the program resides in subroutines which are then easier to write and check individually

Token Game – Statechart Design Example

BRIEF: The game starts with a welcome screen then after 2 seconds the instruction screen appears. The game waits until a button is pressed then a token **T** is randomly placed onto the LCD. 4 buttons are required to move the player **P** around the LCD: 8(up), 4(left), 6(right) and 2(down) to capture the token. Note that the player movements wrap around the screen.
When the player has captured a token, another is randomly generated. After capturing 5 tokens the time taken is displayed, after capturing 10 tokens display the time taken.



Here is the **statechart** for this game (note in this version after collecting 10 tokens nothing happens)**.**



(UMLPAD)

In the program there is a **state** variable that manages the current state and controls what the program is doing at any particular time. This state variable is altered by the program as various events occur (e.g. a token has been captured) or by user input (pressing a button to restart the game).

```
dim state as byte
'REMEMBER TO DIMENSON ALL YOUR VARIABLES HERE

Const got5tokens = 1
Const HitEnemy = 2
Const YouLose = 3
Const InPlay = 4
Const HighScores = 5
Const level2Instructions = 6
Const got10tokens = 7
Const got1token = 8
Const YouWin = 9
Const Welcome = 10
Const Instructions = 11
'REMEMBER TO DEFINE ALL YOUR CONSTANTS HERE

state = Welcome

Do
  while state = got5tokens
    gosub DispScore
     state = level2Instructions
  wend

  while state = HitEnemy
     state = YouLose
  wend

  while state = YouLose
     state = Welcome
  wend

  while state = InPlay
    gosub refreshDisplay
    gosub ReadButtons
    if xPos=TokenX and yPos=TokenY then
     state = got1token
    end if
    if btn=right then
     state = InPlay
     GOSUB GoRight
    end if
    if btn=left then
     state = InPlay
     GOSUB GoLeft
    end if
    if btn=down then
     state = InPlay
     GOSUB GoDown
    end if
     state = HitEnemy
    if btn=Up then
     state = InPlay
     GOSUB GoUp
    end if
  wend

  while state = HighScores
     state = Welcome
  wend

  while state = level2Instructions
    if btn=start then
     state = InPlay
     GOSUB MakeAToken
    end if
  wend

  while state = got10tokens
    gosub DispScore
     state = YouWin
  wend

  while state = got1token
    gosub DispScore
    if TokenCount=10 then
     state = got10tokens
```

In the main do-loop
The subroutines to run
are within the While-Wend
statements

To change what a program is doing you don’t Gosub to a new subroutine. You change the state variable to a new state, the current subroutine is then completed.

The While_Wend statements detect the state change and controls which subroutines are called.

The variable state is a 'flag' or 'signal' or 'semaphore' in computer science. It is a very common technique. We set the flag in one part of the program to tell another part of the program what to do.

```
    end if
      state = InPlay
      GOSUB MakeAToken
    if TokenCount=5 then
      state = got5tokens
    end if
  wend

  while state = YouWin
      state = HighScores
  wend

  while state = Welcome
    if secs>2 then
      state = Instructions
    end if
  wend

  while state = Instructions
    gosub DispInstructions
    if btn=start then
      state = InPlay
      GOSUB startTimer
    end if
  wend

Loop

'**********************************
subroutines
```

```
Disp_welcome:
  Locate 1 , 1
  Lcd "     Welcome to the TOKEN GAME"
  Wait 2
  State = Instructions
  Cls
Return

Disp_instrustions:
      Cls
      State = Instructions
Return

Disp_instructions:
  Locate 1 , 1
  Lcd "capture the tokens  "
  Locate 2 , 1
  Lcd "4=left, 6=right"
  Locate 3 , 1
  Lcd "2=up, 8=down      "
  Locate 4 , 1
  Lcd "D to start"
Return
```

```
Got1:
  Cls
  Incr Tokencount
  Select Case Tokencount
  Case 1 To 4:
    Locate 1 , 10
    Lcd "you got " ; Tokencount      ‘display number of tokens
    Waitms 500                       ‘wait
    Cls
    State = Inplay                   ‘restart play
    Gosub Makeatoken
  Case 5:
    State = Got5tokens
  End Select
Return

Got5:
  Cls
  Locate 1 , 2
  Lcd " YOU GOT 5 TOKENS"
  Locate 2 , 1
  Seconds = Hundredths / 100            'seconds
  Lcd "  in " ; Seconds ; "."
  Seconds = Seconds * 100
  Hundredths = Hundredths - Seconds
  Lcd Hundredths ; "seconds"
  State = Gameover
Return

Got10:
Return

Makeatoken:
  'puts a token on the lcd in a random position
  Tokenx = Rnd(rhs)                  'get a random number from 0 to Xmax-1
  Tokeny = Rnd(bot_row)              'get a random number from 0 to Ymax-1
  Incr Tokenx                        'to fit 1 to Xmax display columns
  If Tokenx > Rhs Then Tokenx = Rhs    'dbl check for errors
  Incr Tokeny                        'to fit 1 to Ymax disp rows
  If Tokeny > Bot_row Then Tokeny = Bot_row     'dbl check for errors
  Locate Tokeny , Tokenx                'y.x
  Lcd "T"                            'Chr(1)
Return
```

```
Go_left:
    Select Case Xpos
    Case Lhs :                         'at left hand side of lcd
      Oldx = Xpos                       'remember old x position
      Xpos = Rhs                        'wrap around display
      Oldy = Ypos                       'remember old y position
    Case Is > Lhs                       'not at left hand side of lcd
      Oldx = Xpos                       'remember old x position
      Xpos = Xpos - 1                    'move left
      Oldy = Ypos                       'remember old y position
    End Select
Return

Go_right:
    Select Case Xpos
    Case Is < Rhs:
      Oldx = Xpos
      Xpos = Xpos + 1
      Oldy = Ypos
    Case Rhs:
      Oldx = Xpos
      Xpos = Lhs
      Oldy = Ypos
    End Select
Return

Go_up:
    Select Case Ypos
    Case Top_row :
      Oldy = Ypos
      Ypos = Bot_row
      Oldx = Xpos
    Case Is > Top_row
      Oldy = Ypos
      Ypos = Ypos - 1
      Oldx = Xpos
    End Select
Return

Go_down:
    Select Case Ypos
    Case Is < Bot_row :
      Oldy = Ypos
      Ypos = Ypos + 1
      Oldx = Xpos
    Case Bot_row :
      Oldy = Ypos
      Ypos = Top_row
      Oldx = Xpos
    End Select
Return
```

These routines keep track of player movements. We always know the current position and the old position for the refresh display routine.

This gets a little complicated when the player moves off the screen, e.g. when going from left to right after the player hits the rhs it wraps around to the lhs.

Serial Communications

Parallel communications is sending data all at once on many wires and serial communications is all about sending data sequentially using a single or a few wires. With serial communications the data is sent from one end of a link to the other end one bit at a time. There are 2 ways of classifying serial data communications.

1. As either Simplex, half duplex or full duplex
2. Or as either synchronous or asynchronous

Simplex and duplex

In serial communications **simplex** is where data is only ever travelling in one direction, there is one transmitter and one receiver.

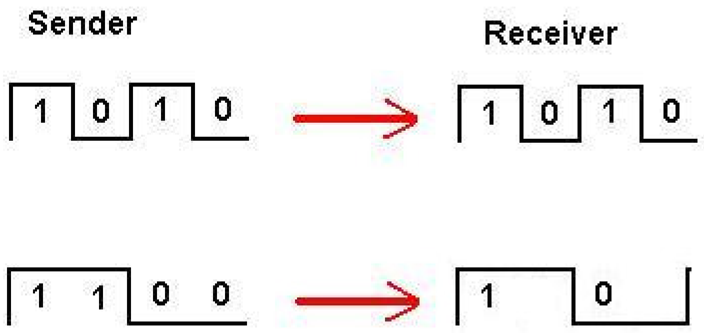
In **half duplex** communications both ends of a link can be transmitter and receiver but they take turns sending and receiving

In **full duplex** both ends can send and receive data at the same time.

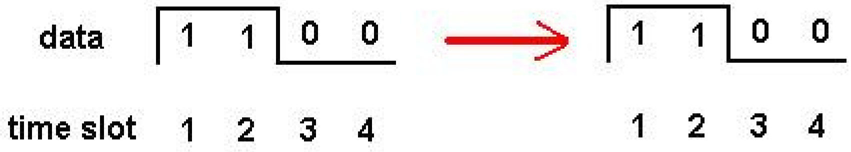
Synchronous and asynchronous

Imagine sending the data 1010 serially, this is quite straight forward, the sender sends a 1 ,then a 0, then a 1, then a 0. The receiver gets a 1, then a 0, then a 1, then a 0; No problems.

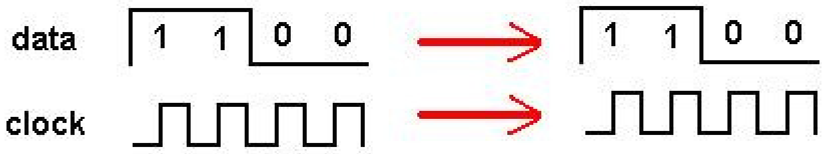
Now send 1100 the sender sends a 1 then 1 then a 0 then a 0, the receiver gets a one then a zero, hey what happened!!



The receiver has no way of knowing how long a 1 or 0 is without some extra information. In an **asynchronous** system the sender and receiver are setup to expect data at a certain number of bits per second e.g. 19200, 2400. Knowing the bit rate means that the spacing is known and the data is allocated a time slot, therefore the receiver will know when to move on to receiving the next bit.



Synchronous communications is where a second wire in the system carries a clock signal, to tell the receiver when the data should be read.



Every time the clock goes from 0 to 1 the data is available at the receiver. Now there is no confusion about when a 1 is present or a zero. The receiver checks the data line only at the right time.

Serial Communications, Bascom and the AVR

The AVR has built in serial communications hardware and Bascom has software commands to use it.

- UART: (universal asynchronous receiver transmitter), which when used with suitable circuitry is used for serial communications via RS232. It has separate txd (transmit data) and rxd (receive data) lines, this is asynchronous (no clock line), and is capable of full duplex, both transmitting and receiving at the same time.
- SPI: (serial peripheral interface) which has 2 data lines and 1 clock line, these are the three lines used for programming the microcontroller in circuit as well as for communications between the AVR and other devices. This is a synchronous communications interface, it has a separate clock line. It is also full duplex. The 2 data lines are MISO (master in slave out) and MOSI (master out slave in) these are full duplex, because data can travel on the 2 lines at the same time.

Bascom has software built into it for two other communications protocols

- I2C: (pronounced I squared C) this stands for Inter IC bus, it has 1 data line and 1 clock line. Because it has only 1 data line it is half duplex, the sender and receiver take turns, and because it has a clock line it is synchronous.
- Dallas 1-Wire: this is literally 1 wire only, so the data must be half duplex, and asynchronous.

RS 232 Serial Communications

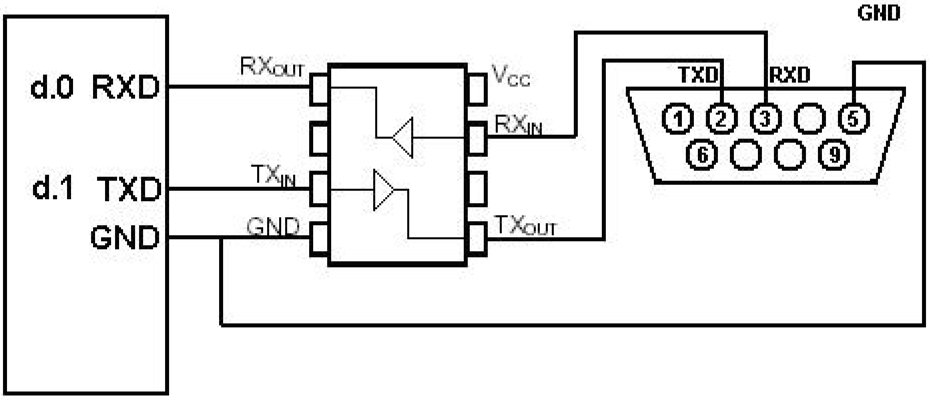
RS232/Serial communications is a very popular communications protocol between computers and peripheral devices such as modems. It is an ideal communication medium to use between a PC and the microcontroller.

The different parts of the RS232 system specification include the plugs, cables, their functions and the process for communications. The plugs have either 9 or 25 pins, more commonly today the PC has two 9 pin male connectors.

There are two data lines one is TXD (transmit data) the other RXD (receive data), as these are independent lines devices can send and receive at the same time, making the system full duplex. There is a common or ground wire and a number of signal wires.

There is no clock wire so the system of communications is asynchronous. There are a number of separate control lines to handle 'handshaking' commands, i.e. which device is ready to transmit, receive etc.

The AVR microcontroller has built in hardware to handle RS232 communications, the lines involved are portd.0 (RXD) and portd.1 (TXD). These two data lines however cannot be directly connected to a PCs RS232 port because the RS232 specification does not use 5V and 0V, but +15V as a zero and -15V as a one. Therefore some interface circuitry is required, the MAX232 and the MAX275 are common devices used for this. A connector (DB9-Female) is required



Research RS232 and find the names of all the pins

Pin 1

Pin 2

Pin 3

Pin 4

Pin 5

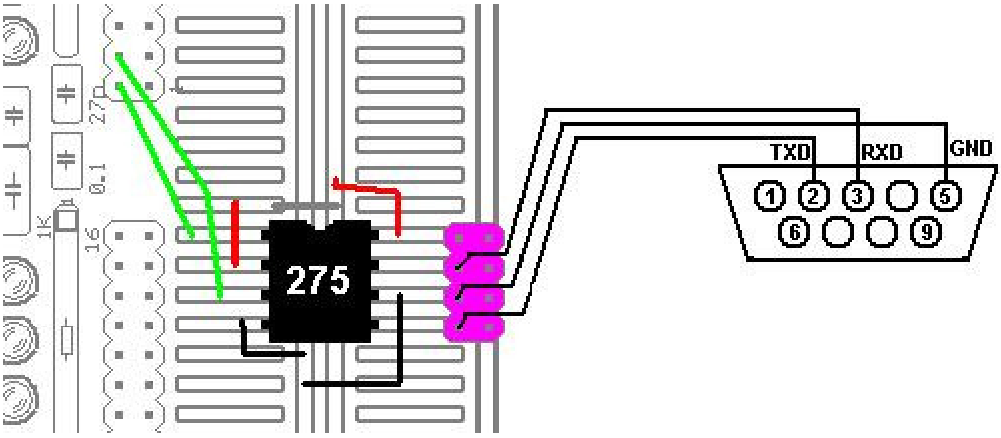
Pin 6

Pin 7

Pin 8

Pin 9

Connect the DS275 as shown.



The DS275 must connect to d.0 and d.1

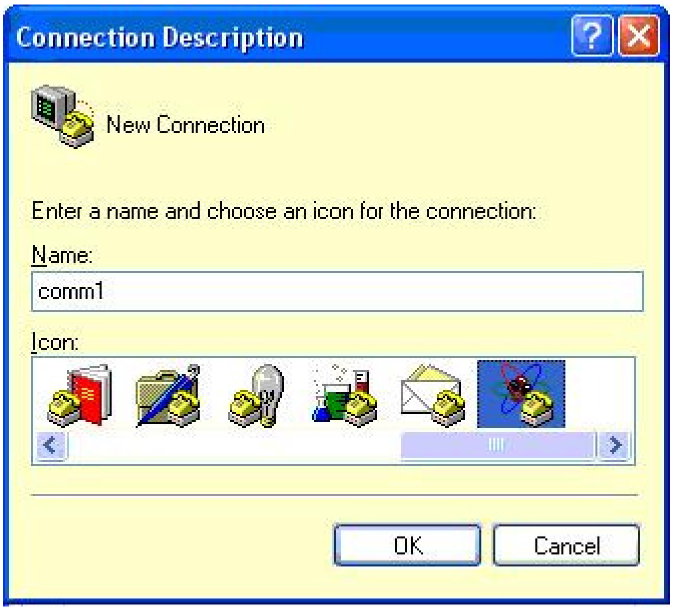
Use 3 header pins on the pcb and a header plug for the cable to the DB9-F connector.

Software

There are several different software options for communicating over rs232 from the AVR, the simplest however is the print statement.

print "hello" will send the ASCII text string to the pc. At the pc end there must be some software listening to the comport, Windows has **HyperTerminal** already built in to do this.

Open HyperTerminal (normally found in programs/accessories/communications).
Start a new connection and name it comm1

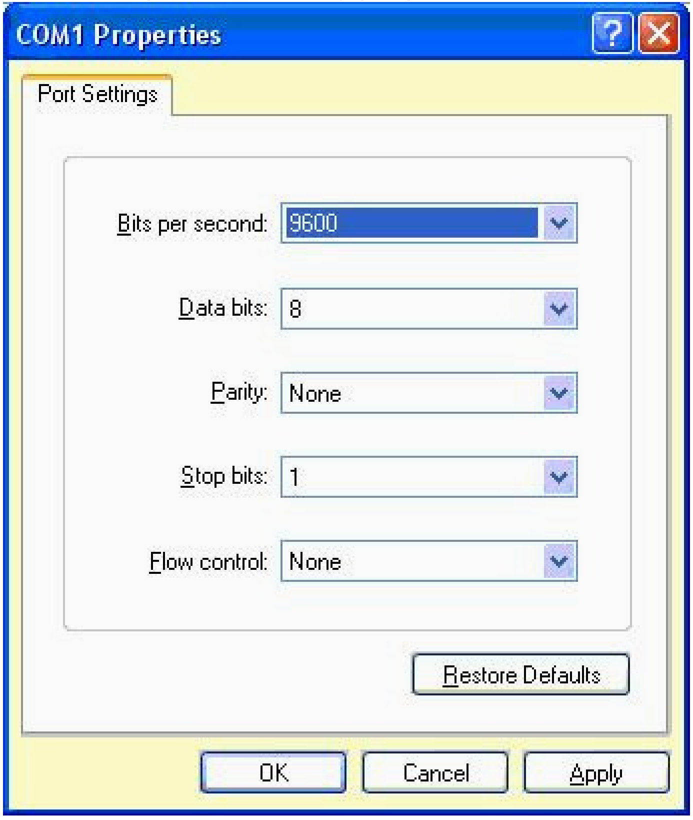


Enter a name and choose an icon for the connection:

Name: comm1

On the next screen make sure you select comm1 as the port.

Then setup the following properties



Bits per second: 9600

Data bits: 8

Parity: None

Stop bits: 1

Flow control: None

When you click on OK HyperTerminal can now send and receive using comm1.

Bascom Program

```
'-----------------------------------------------------------------
' 1. Title Block
' Author: B.Collis
' Date: 22 Aug 03
' Version: 1.0
' File Name: Serialio_Ver1.bas
'-----------------------------------------------------------------
' 2. Program Description:
' This program sends simple text over rs232
' as well as displaying it on the local LCD
'
' 3. Hardware Features:
' DS275 connected to the micro TXD and RXD lines. then wired to a DB9F.
' LCD on portc - note the use of 4 bit mode and only 2 control lines
' 4. Program Features:
' print statement
'-----------------------------------------------------------------
' 5. Compiler Directives (these tell Bascom things about our hardware)
$crystal = 8000000               'the speed of operations inside the micro
$regfile = "m8535.dat"           ' the micro we are using
$baud = 9600 'set data rate for serial comms
'-----------------------------------------------------------------
' 6. Hardware Setups
' setup direction of all ports
Config Porta = Output 'LEDs on portA
Config Portb = Output 'LEDs on portB
Config Portc = Output 'LEDs on portC
Config Portd = Output 'LEDs on portD
Config Lcdpin = Pin , Db4 = Portc.4 , Db5 = Portc.5 , Db6 = Portc.6 , Db7 = Portc.7 , E = Portc.3 , Rs
= Portc.2
Config Lcd = 40 * 2 'configure lcd screen
' 7. Hardware Aliases
' 8. initialise ports so hardware starts correctly
Porta = &B11111111 'turns off LEDs
Portb = &B11111111 'turns off LEDs
Portc = &B11111111 'turns off LEDs
Portd = &B11111111 'turns off LEDs
'-----------------------------------------------------------------
' 9. Declare Constants
Const Timedelay = 500
'-----------------------------------------------------------------
' 10. Declare Variables
Dim Count As Byte
' 11. Initialise Variables
Count = 0
'-----------------------------------------------------------------
' 12. Program starts here
Print "Can you see this"
Do
   Incr Count
   Cls
   Lcd Count
   Print " the value is " ; Count
   Waitms Timedelay
Loop
End 'end program
'-----------------------------------------------------------------
' 13. Subroutines
'-----------------------------------------------------------------
' 14. Interrupts
```

Exercise

Getting text from a PC

```
'-----------------------------------------------------------------
' 1. Title Block
' Author: B.Collis
' Date: 22 Aug 03
' Version: 3.0
' File Name: Serialio_Ver3.bas
'-----------------------------------------------------------------
' 2. Program Description:
' This program prompts for text from the pc over rs232
' and displays it on the local LCD
'
' 3. Hardware Features:
' DS275 connected to the micro TXD and RXD lines. then wired to a DB9F.
' LCD on portc - note the use of 4 bit mode and only 2 control lines
' 4. Program Features:
' input statement
' string variables
'-----------------------------------------------------------------
' 5. Compiler Directives (these tell Bascom things about our hardware)
$crystal = 8000000 'the crystal we are using
$regfile = "m8535.dat" 'the micro we are using
$baud = 9600 'set data rate for serial comms
'-----------------------------------------------------------------
' 6. Hardware Setups
' setup direction of all ports
Config Porta = Output 'LEDs on portA
Config Portb = Output 'LEDs on portB
Config Portc = Output 'LEDs on portC
Config Portd = Output 'LEDs on portD
Config Lcdpin = Pin , Db4 = Portc.4 , Db5 = Portc.5 , Db6 = Portc.6 , Db7 = Portc.7 , E = Portc.3 , Rs = Portc.2
Config Lcd = 40 * 2 'configure lcd screen
' 7. Hardware Aliases
' 8. initialise ports so hardware starts correctly
Porta = &B11111111 'turns off LEDs
Portb = &B11111111 'turns off LEDs
Portc = &B11111111 'turns off LEDs
Portd = &B11111111 'turns off LEDs
Cls
Cursor Noblink
'-----------------------------------------------------------------
' 9. Declare Constants
```

```
Const Timedelay = 2
'-----------------------------------------------------------------
' 10. Declare Variables
Dim Text As String * 15
' 11. Initialise Variables
Text = ""
'-----------------------------------------------------------------
' 12. Program starts here
Print "Can you see this"
Do
   Input "type in something" , Text
   Lcd Text
   Wait Timedelay
   Cls
Loop
End 'end program
'-----------------------------------------------------------------
' 13. Subroutines
'-----------------------------------------------------------------
' 14. Interrupts
```

BASCOM Serial Commands

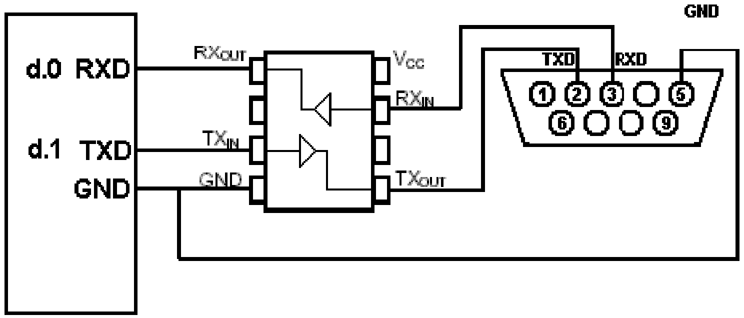
There are a number of different serial commands in Bascom to achieve different functions, find these in the help file and write in the description of each one.

Print
PrintBin
Config SerialIn
Config SerialOut
Input
InputBin
InputHex
Waitkey
Inkey
IsCharWaiting
$SerialInput2LCD
$SerialInput
$SerialOutput
Spc

Some AVRs have more than one UART (the internal serial device) and it is possible to have software only serial comms in Bascom and use
Serin, Serout,
Open
Close
Config Waitsuart

Serial IO using Inkey()

```
'--------------------------------------------------------------------
' 1. Title Block
' Author:    B.Collis
' Date:      22 Aug 03
' Version:   1.0
' File Name: Serialio_Ver1.bas
'--------------------------------------------------------------------
' 2. Program Description:
' This program receives characters from the RS232/comm/serial port of a PC
' it displays them on the LCD
'
' 3. Hardware Features:
' DS275/MAX232 connected to the micro TXD and RXD lines. then wired to a DB9F.
' LCD on portc - note the use of 4 bit mode and only 2 control lines
' 4. Program Features:
' print statement
' serial interrupt and buffer
' inkey reads the serial buffer to see if a char has arrived
' note that a max of 16 chars can arrive before the program
' automatically prints the message
'--------------------------------------------------------------------
' 5. Compiler Directives (these tell Bascom things about our hardware)
$crystal = 8000000                'the crystal we are using
$regfile = "m8535.dat"            'the micro we are using
$baud = 9600                      'set data rate for serial comms
'--------------------------------------------------------------------
' 6. Hardware Setups
' setup direction of all ports
Config Porta = Output             'LEDs on portA
Config Portb = Output             'LEDs on portB
Config Pinb.0 = Input
Config Pinb.1 = Input
Config Portc = Output             'LEDs on portC
Config Portd = Output             'LEDs on portD
Config Pind.2 = Input
Config Pind.3 = Input
Config Pind.6 = Input

Config Lcd = 40 * 2               'configure lcd screen
Config Lcdpin = Pin , Db4 = Portc.4 , Db5 = Portc.5 , Db6 = Portc.6 , Db7 = Portc.7 , E = Portc.3 , Rs = Portc.2

Config Serialin = Buffered , Size = 20  'buffer the incoming data
' 7. Hardware Aliases
Sw_1 Alias Pinb.0
Sw_2 Alias Pinb.1
Sw_3 Alias Pind.2
Sw_4 Alias Pind.3
Sw_5 Alias Pind.6
' 8. initialise ports so hardware starts correctly
Porta = &B11111111                'turns off LEDs
Portb = &B11111111                'turns off LEDs
Portc = &B11111111                'turns off LEDs
Portd = &B11111111                'turns off LEDs
'--------------------------------------------------------------------
' 9. Declare Constants

'--------------------------------------------------------------------
' 10. Declare Variables
Dim Count As Byte
Dim Char As Byte
Dim Charctr As Byte
Dim Message As String * 16
' 11. Initialise Variables
Count = 0

'--------------------------------------------------------------------
' 12. Program starts here
Enable Interrupts                 'used by the serial buff
Print "Hello PC"
Cls
Lcd "LCD is ok"
Wait 3
Do
  Debounce Sw_1 , 0 , Sub_send1 , Sub  'when switch pressed gosub
  Debounce Sw_2 , 0 , Sub_send2 , Sub  'when switch pressed gosub
  Char = Inkey()                  'get a char from the serial buffer
  Select Case Char                'choose what to do with it
    Case 0 :                      'do nothing (no char)
    Case 13 : Gosub Dispmessage   'Ascii 13 is CR so show message
    Case Else : Incr Charctr      'keep count of chars
       Message = Message + Chr(char)   'add new char to message
  End Select
  If Charctr > 15 Then            'if 16 chars received
    Gosub Dispmessage             'display the message straight away
  End If
Loop
End                               'end program
'--------------------------------------------------------------------
' 13. Subroutines
Sub_send1:
  Print "this is hard work"       'send it to comm port
Return

Sub_send2:
  Print "not really"              'send it to comm port
Return

Dispmessage:
  Cls
  Lcd Message
  Message = ""
  Charctr = 0
  Incr Count                      'send some data to the comm port
  Print "you have sent = " ; Count ; " messages"
Return
'--------------------------------------------------------------------
' 14. Interrupts
```

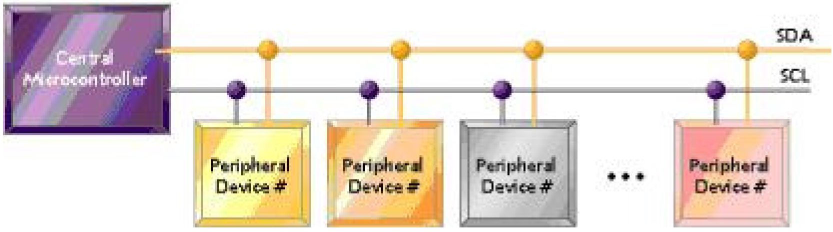


Introduction to I2C

The Inter-IC bus (I2C pronounced "eye-squared-see") was developed by Philips to communicate between devices in their TV sets. It is now popular and is often used when short distance communications is needed. It is normally used within equipment to communicate between pcb's, e.g. main boards and display boards rather than externally to other equipment.

It is a half duplex synchronous protocol, which means that only one end of the link can talk at once and that there are separate data and clock lines. The real strength of this protocol is that many devices can share the bus which reduces the number of I/O lines needed on microcontrollers, increases the number of devices 1 micro can interface to and many manufacturers now make I2C devices.

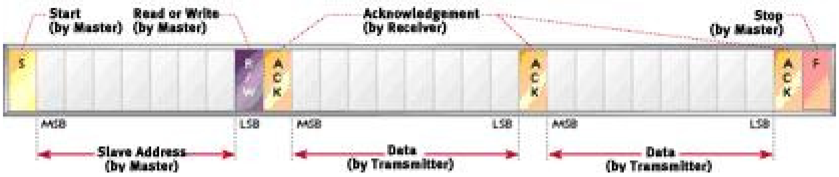
Figure 1: I^{2}C has two lines in total



The two lines are SDA - Serial data and SCL - Serial Clock
Communication

The system of communications is not too difficult to follow, the first event is when the master issues a start pulse causing all slaves to wake up and listen. the master then sends a 7 bit address which corresponds to one of the slaves on the bus. Then one more bit is sent that tells the slave whether it is going to be receiving or sending information. This is then followed by an ACK bit (acknowledge) issued by the receiver, saying it got the message. Data is then sent over the bus by the transmitter.

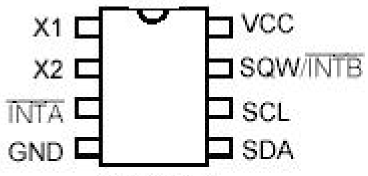
Figure 2: I^{2}C communication



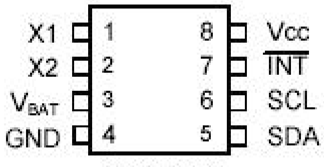
The I2C protocol is not too hard to generate using software; Bascom comes with the software already built in making I2C very easy to use.

I2C Real Time Clocks

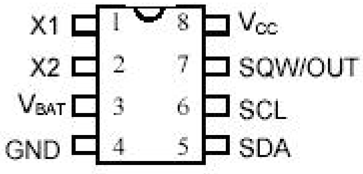
These are fantastic devices that connect to the microcontroller and keep the time for you. Some common devices are the DS1337, DS1678 and DS1307.



DS1337



DS1678



DS1307

All three require an external 32.768KHz crystal connected to X1 and X2, 5Volts from your circuit connected to Vcc, a ground connection (OV) and connection of two interface pins to the microcontroller, SCL (serial clock) and SDA (serial data).

The DS1678 and DS1307 can have a 3V battery connected to them as backups to keep the RTC time going even though the circuit is powered down. This will last for a couple of years and note that it is not rechargeable. There are datasheets on www.maxim-ic.com website for each of these components as well as many other interesting datasheets on topics such as battery backup. Each of these devices has other unique features that can be explored once the basic time functions are operational.

In these RTCs the registers are split into BCD digits. What this means is that instead of storing seconds as one variable it splits the variable into two parts the units value and the tens value.

register 0	Tens of seconds	Units of seconds
register 1	Tens of minutes	Units of minutes
register 2	Tens of hours	Units of hours
register 3	Tens of hours	Units of hours
register ..	Tens of ...	Units of ...

When we want to put the variable onto an LCD we cannot write lcd seconds as the number would not be correct. We must first convert the BCD to decimal using

Seconds = Makedec(seconds).

LCD Seconds

The opposite when writing to the time registers

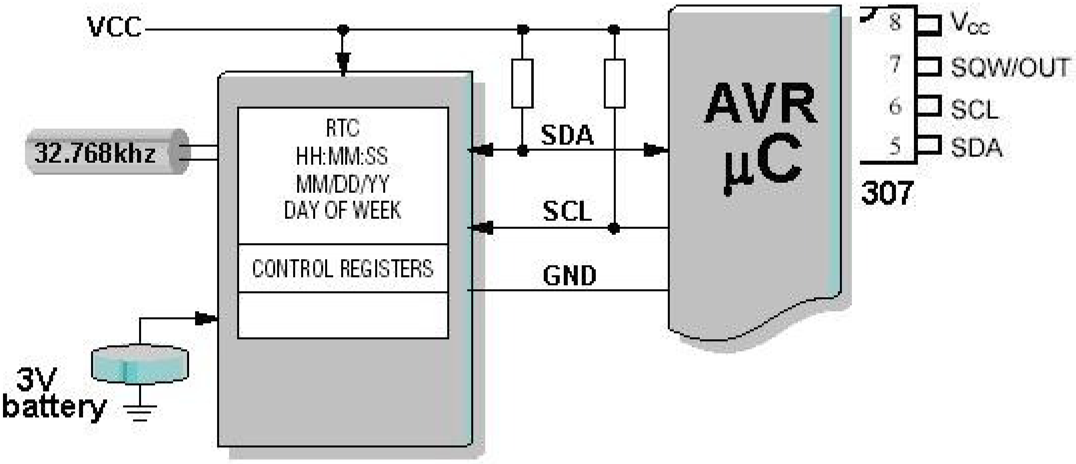
Temp = Makebcd(seconds)

I2cwbyte Temp

Real Time Clocks

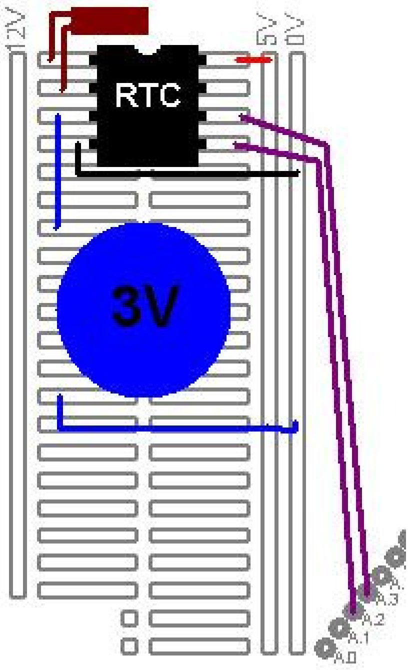
These devices are very common in microcontroller products such as microwave ovens, cellular phones, wrist watches, industrial process controllers etc.

Connecting the RTC



The crystal for the RTC is a 32.768khz crystal. The reason for the strange number is that 32768 is a multiple of 2, so all that is needed to obtain 1 second pulses is to divide the frequency by two 15 times to get exactly 1 second pulses.

32768
/2 = 16384, /2 = 8192, /2 = 4096, /2 = 2048….2 = 8, /2 = 4, /2 = 2, /2 = 1



Connecting the RTC to the board

Take special note about bending the leads and soldering to avoid damage to the crystal. Also fix the crystal to the board somehow to reduce strain on the leads.

The I2C lines SDA and SCL require pull up resistors of 4k7 each to 5V.

The battery is a 3V lithium cell, connect it between 0V and the battery pin of the RTC. If a battery is not used then the battery backup pin probably needs connecting to 0V, but check the datasheet first.

Internal Features

First open the datasheet for the DS1307 RTC

There is a memory within the RTC, firstly all the time and dates are stored individually. The units and the 10s of each number are stored separately.

Here is the layout of the memory within the RTC

ADDRESS	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
00	0	10 Seconds			Seconds			
01	0	10 Minutes			Minutes			
02	0	12/24	AM/PM 10Hr	10Hr	Hour			
03	0	0	0	0		Day of week		
04	0	0	10 Date		Date			
05	0	0	0	10 Mo	Month			
06	10 Year				Year			
07	CONTROL							
08 3F	RAM							

The date and time Sunday, 24 August 2007 21:48:00 are stored as this

ADDRESS	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
00	0				0			
01	4				8			
02	2				1			
03	0				7			
04	2				4			
05	0				8			
06	0				7			
07	2				0			

When we read the RTC we send a message to it,
SEND DATA FROM ADDRESS 0 and it sends
0,48,21,07,24,08,7,20..

DS1307 RTC

Here is the process for communicating with the DS1678 RTC followed by the code for one connected to an 8535.

Step1: configure the hardware and dimension a variable, temp, to hold the data we want to send to/receive from the 1678. Dimension the variables used to hold the year, month, day, hours, etc. Don't forget to configure all the compiler directives and hardware such as the LCD, thermistor, switches etc.

Step2: setup the control register in the RTC, to specify the unique functions we require the 1307 to carry out. This is only ever sent once to the 1307.

Step 3: write a number of subroutines that handle the actual communication with the control and status registers inside the 1307. These routines make use of the Bascom functions for I2C communication.

Step 4: write a subroutine that gets the time, hours, date, etc from the 1307.

step 5 : write a subroutine that sets the time, hours, date, etc from the 1307.

step 6: write a program that incorporates these features and puts the time on an LCD.

```
'--------------------------------------------------------------
' 1. Title Block
' Author:   B.Collis
' Date:     26 Mar 2005
' File Name: 1307_Ver4.bas
'--------------------------------------------------------------
' 2. Program Description:
' use an LCD to display the time
' has subroutines to start clock,write time/date to the rtc,
' read date/time from the rtc, setup the SQW pin at 1Hz
'added subroutines to read and write to ram locations
' LCD on portc - note the use of 4 bit mode and only 2 control lines
' DS1307 SDA=porta.2 SDC=porta.3
'--------------------------------------------------------------
' 3. Compiler Directives (these tell Bascom things about our hardware)
$crystal = 8000000                'the crystal we are using
$regfile = "m32def.dat"            'the micro we are using
'--------------------------------------------------------------
' 4. Hardware Setups
' setup direction of all ports
Config Porta = Output              '
Config Portb = Output              '
Config Portc = Output              '
Config Portd = Output              '
' config 2 wire I2C interface
'Config I2cdelay = 5               ' default slow mode
Config Sda = Porta.2
Config Scl = Porta.3
'Config lcd
```

```
Config Lcdpin = Pin , Db4 = Portc.4 , Db5 = Portc.5 , Db6 = Portc.6 , Db7 = Portc.7 , E = Portc.3
, Rs = Portc.2
Config Lcd = 16 * 2                    'configure lcd screen
'5.Hardware Aliases

'6. Initialise ports so harware starts correctly
Cls                                'clears LCD display
Cursor Off                           'no cursor

'--------------------------------------------------------------
' 7. Declare Constants

'--------------------------------------------------------------
' 8. Declare Variables
Dim Temp As Byte
Dim Year As Byte
Dim Month As Byte
Dim Day As Byte
Dim Weekday As Byte
Dim Hours As Byte
Dim Minutes As Byte
Dim Seconds As Byte
Dim Ramlocation As Byte
Dim Ramvalue As Byte
' 9. Initialise Variables
Year = 5
Month = 3
Weekday = 6
Day = 26
Hours = 6
Minutes = 01
Seconds = 0

'--------------------------------------------------------------
' 10. Program starts here
Waitms 300
Cls

'these 3 subroutines should be called once and then commented out
'Gosub Start1307clk
'Gosub Write1307ctrl
'Gosub Write1307time

'Gosub Clear1307ram          'need to use once as initial powerup is undefined
'Gosub Writeram
Gosub Readram

'Ramvalue = &HAA
'Call Write1307ram(ramlocation , Ramvalue)

Do
  Gosub Read1307time                'read the rtc
  Locate 1 , 1
  Lcd Hours
  Lcd ":"
  Lcd Minutes
  Lcd ":"
  Lcd Seconds
  Lcd "          "
  Lowerline
  Lcd Weekday
  Lcd ":"
  Lcd Day
  Lcd ":"
  Lcd Month
  Lcd ":"
  Lcd Year
  Lcd "          "
  Waitms 200

Loop



End                                   'end program
'-----------------------------------------------------------------
' 11. Subroutines
Read1307time:                          'RTC Real Time Clock
   I2cstart
   I2cwbyte &B11010000                 'send device code (writing data)
   I2cwbyte 0                      'address to start sending from
   I2cstop
   Waitms 50
   I2cstart
   I2cwbyte &B11010001                 'device code (reading)
   I2crbyte Seconds , Ack
   I2crbyte Minutes , Ack
   I2crbyte Hours , Ack
   I2crbyte Weekday , Ack
   I2crbyte Day , Ack
   I2crbyte Month , Ack
   I2crbyte Year , Nack
   Seconds = Makedec(seconds)
   Minutes = Makedec(minutes)
   Hours = Makedec(hours)
   Weekday = Makedec(weekday)
   Day = Makedec(day)
   Month = Makedec(month)
   Year = Makedec(year)
   I2cstop
Return

'write the time and date to the RTC
Write1307time:
   I2cstart
   I2cwbyte &B11010000                 'send device code (writing data)
   I2cwbyte &H00                     'send address of first byte to access
   Temp = Makebcd(seconds)              'seconds
   I2cwbyte Temp
   Temp = Makebcd(minutes)              'minutes
   I2cwbyte Temp
   Temp = Makebcd(hours)                'hours
   I2cwbyte Temp
   Temp = Makebcd(weekday)              'day of week
   I2cwbyte Temp
   Temp = Makebcd(day)                  'day
   I2cwbyte Temp
   Temp = Makebcd(month)                'month
   I2cwbyte Temp
   Temp = Makebcd(year)                 'year
   I2cwbyte Temp
   I2cstop
Return


Write1307ctrl:
   I2cstart
   I2cwbyte &B11010000                  'send device code (writing data)
   I2cwbyte &H07                     'send address of first byte to access
   I2cwbyte &B10010000                  'start squarewav output 1Hz
   I2cstop
Return


Start1307clk:
   I2cstart
   I2cwbyte &B11010000                  'send device code (writing data)
   I2cwbyte 0                      'send address of first byte to access
   I2cwbyte 0                      'enable clock-also sets seconds to 0
   I2cstop
Return


Write1307ram:
'no error checking ramlocation should be from &H08 to &H3F (56 bytes only)
   I2cstart
   I2cwbyte &B11010000                  'send device code (writing data)
   I2cwbyte Ramlocation                'send address of byte to access
   I2cwbyte Ramvalue                  'send value to store
   I2cstop
Return


'routine to read the contents of one ram location
'setup ramlocation first and the data will be in ramvalue afterwards
'no error checking ramlocation should be from &H08 to &H3F (56 bytes only)
Read1307ram:
   I2cstart
   I2cwbyte &B11010000                  'send device code (writing data)
```

```
   I2cwbyte Ramlocation               'send address of first byte to access
   I2cstop
   Waitms 50
   I2cstart
   I2cwbyte &B11010001                 'device code (reading)
   I2crbyte Ramvalue , Nack
   I2cstop
Return

Clear1307ram:
   Ramvalue = 00
   Ramlocation = &H08
   I2cstart
   I2cwbyte &B11010000                 'send device code (writing data)
   I2cwbyte Ramlocation               'send address of byte to access
   For Ramlocation = &H08 To &H3F
      I2cwbyte Ramvalue              'send value to store
   Next
   I2cstop
Return

Writeram:
   Ramlocation = &H08
   Ramvalue = 111
   Gosub Write1307ram
   Ramlocation = &H09
   Ramvalue = 222
   Gosub Write1307ram
Return

Readram:
   Cls
   Ramlocation = &H08
   Gosub Read1307ram
   Lcd Ramvalue
   Lcd ":"
   Ramlocation = &H09
   Gosub Read1307ram
   Lcd Ramvalue
   Ramlocation = &H0A
   Gosub Read1307ram
   Lcd ":"
   Lcd Ramvalue
   Wait 5
Return

'---------------------------------------------------------------
' 12. Interrupts
```

Arrays

It is easy to dimension variables to store data, however what do you do when you want to store many similar variables e.g. 50 light level readings over a period of time.

Do you create 50 variables e.g. lightlevel1, lightlevel2, lightlevel3 .... lightlevel50 ?
The answer is no because it is so difficult to read and write to 50 different variables.

We create an ARRAY type variable. Arrays are a highly important programming structure in computer science.

e.g Dim lightlevel as byte(50) An array is very easy to read and write in a loop, lightlevel(1) will be the first value and lightlevel(50) will be the last.

In this exercise you will modify the given program which stores 50 lightlevel readings.

' File Name: arrayV1.bas

```
' 5. Compiler Directives (these tell Bascom things about our hardware)
$crystal = 8000000                    'the speed of the micro
$regfile = "m8535.dat"                 'our micro, the ATMEGA8535-16PI
'----------------------------------------------------------------
' 6. Hardware Setups
' setup direction of all ports
Config Porta = Output                  'LEDs on portA
Config Portb = Output                  'LEDs on portB
Config Portc = Output                  'LEDs on portC
Config Portd = Output                  'LEDs on portD
'config inputs
Config Pina.0 = Input                 ' ldr
Config Pind.2 = Input                 'switch A
Config Pind.3 = Input                 'switch B
Config Pind.6 = Input                 'switch C
Config Pinb.1 = Input                 'switch D
Config Pinb.0 = Input                 'switch E
'LCD
Config Lcdpin = Pin , Db4 = Portc.4 , Db5 = Portc.5 , Db6 = Portc.6 , Db7 = Portc.7 , E =
Portc.3 , Rs = Portc.2
Config Lcd = 40 * 2                   'configure lcd screen
'ADC
Config Adc = Single , Prescaler = Auto , Reference = Internal
Start Adc

' 7. Hardware Aliases
Sw_a Alias Pind.6
Sw_b Alias Pind.3
Sw_c Alias Pind.2
Sw_d Alias Pinb.1
Sw_e Alias Pinb.0

' 8. initialise ports so hardware starts correctly
Porta = &B11111100                    'turns off LEDs ignores ADC inputs
Portb = &B11111100                    'turns off LEDs ignores switches
```

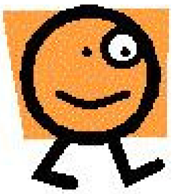
```
Portc = &B11111111                    'turns off LEDs
Portd = &B10110011                    'turns off LEDs ignores switches
Cls                            'clear lcd screen
'-----------------------------------------------------------------
' 9. Declare Constants
Const Reading_delay = 100
'-----------------------------------------------------------------
' 10. Declare Variables
Dim Opmode As Byte
Dim Reading As Word
Dim Lightlevel(50) As Word
Dim Cntr As Byte
' 11. Initialise Variables
Opmode = 0
'-----------------------------------------------------------------
' 12. Program starts here
Do
  Debounce Sw_a , 0 , Mode_select , Sub
  Debounce Sw_b , 0 , Enter_button , Sub
  Debounce Sw_c , 0 , Prev , Sub
  Debounce Sw_d , 0 , Nxt , Sub
  Select Case Opmode
    Case 0 : Gosub Display_welcome
    Case 1 : Gosub Collect_data
    Case 2 : Gosub Display_data
    Case 3 : Gosub Cont_reading
    Case Else : Gosub Display_mode
  End Select
Loop
End                             'end program

'-----------------------------------------------------------------
' 13. Subroutines

Mode_select:
  Cls                          'when mode changes clear the lcd
  If Opmode < 10 Then
    Incr Opmode
  Else
    Opmode = 0
  End If
Return

Display_welcome:
  Locate 1 , 1
  Lcd " Data Collector "
  Lowerline
  Lcd "  version 1.0   "
Return

Collect_data:
    Locate 1 , 1
    Lcd " press enter to "
```

```
     Lowerline
     Lcd "start collection"
Return

Enter_button:
   If Opmode = 1 Then Gosub Start_collecting
Return

Start_collecting:
  Cls
  For Cntr = 1 To 50
     Reading = Getadc(0)            'read lightlevel
     Locate 1 , 1
     Lcd Cntr                       'display the counter
     Locate 2 , 1
     Lcd Reading ; "    "             'diplay the reading
     Lightlevel(cntr) = Reading       ' store reading in array
     Waitms Reading_delay
  Next
  Opmode = 0
Return

Display_data:
   Locate 1 , 1
   Lcd Cntr ; " "
   Locate 2 , 1
   Lcd Lightlevel(cntr) ; "     "
Return

Cont_reading:
   Locate 1 , 1
   Lcd "continous readings"
   Locate 2 , 1
   Reading = Getadc(0)
   Lcd Reading ; "   "
Return

Prev:
   Decr Cntr
Return
Nxt:
   Incr Cntr
Return

Display_mode:
   Locate 1 , 1
   Lcd Opmode
Return
```

1. Fix the bugs with the prev and nxt routines so that they dont go below 0 or above 50.

Computer Programming detail

We refer to programming languages as either **HIGH LEVEL** or **LOW LEVEL** languages.



High Level Languages include Basic, C, Java, Haskell, Lisp, Prolog, C++, and many more.

High level languages are written using text editors such as wordpad or within an **IDE** like Bascom. These languages are typically easy for us to understand, however microcontrollers do not understand these words they only understand binary numbers which are called **Machine Code**. A computer program is ultimately a file containing machine code. Commands written in high level languages must be **compiled** into these binary codes.

Low Level Languages:

Machine code for **all** microcontrollers and microprocessors (all computers) are groups of binary digits (bits) arranged in bytes (8 bits) or words of 16, 32 or 64 bits.

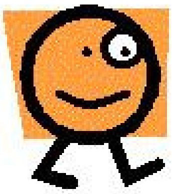
Understanding a program in machine code is not at all easy. The AVR machine code to add the numbers in 2 memory registers is 0001 1100 1010 0111.

To make machine code a little easier to understand we can abbreviate every 4 bits into hexadecimal numbers; HEX uses numbers 0 to 9 and the letters from a to f.

It is easier on the eyes than machine code but still very difficult to read. It looks like this **1CA7** which is easier to read than is 0001 1100 1010 0111**,** but no easier to understand!

Program code for micros is not written directly in machine code, abbreviations are used to refer to the commands, these abbreviations are known as assembly language, assembler or assembly code which is a representaton of the machine code using mnemonics (abbreviated words), these are more readable, for example:

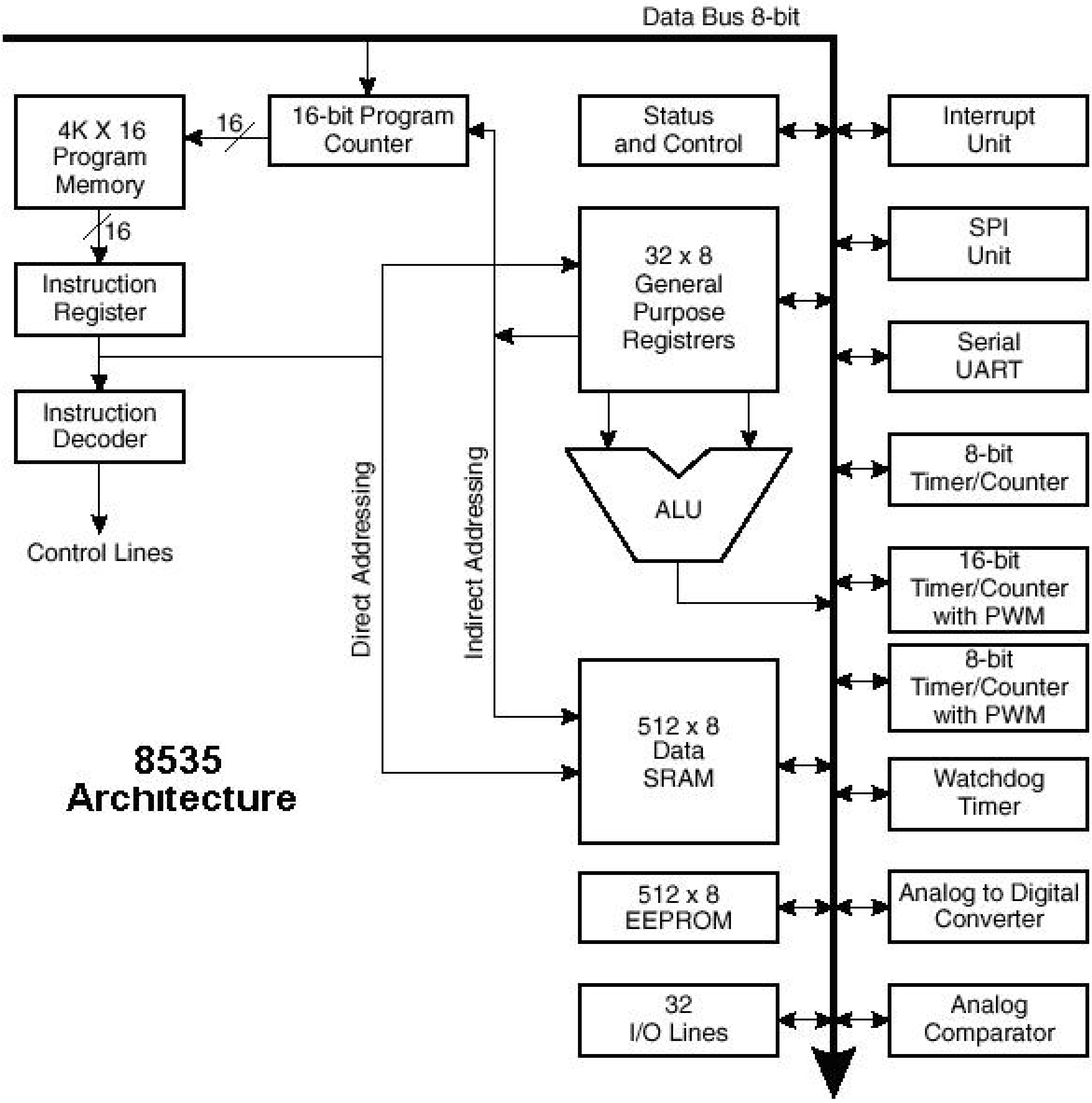
add r12 , r7 instead of **1C A7**



Assembler is much easier to understand than machine code and is in very common use for programming microcontrollers, however It does take more effort to understand the microcontroller internals when programming in assembler.

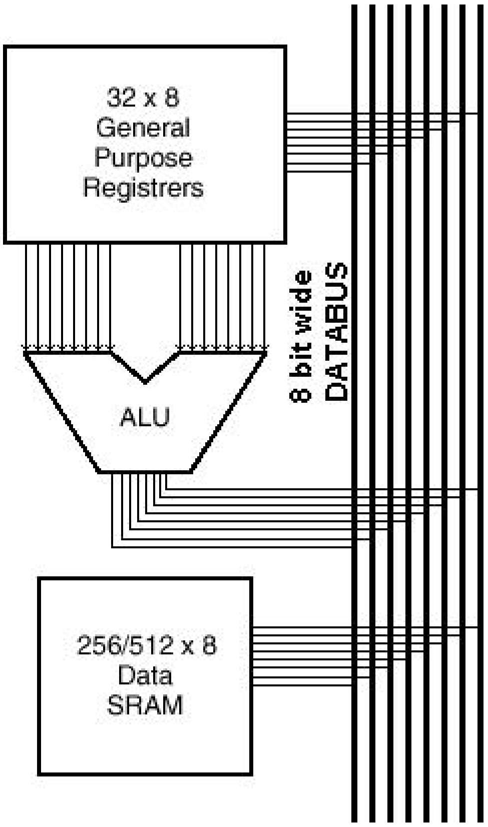
AVR Internals – how the microcontroller works

The AVR microcontroller is a complex integrated circuit, with many features as shown in this block diagram of the AVR's internal architecture.



There are memory, calculation, control and I/O components.

1. The 8bit data bus



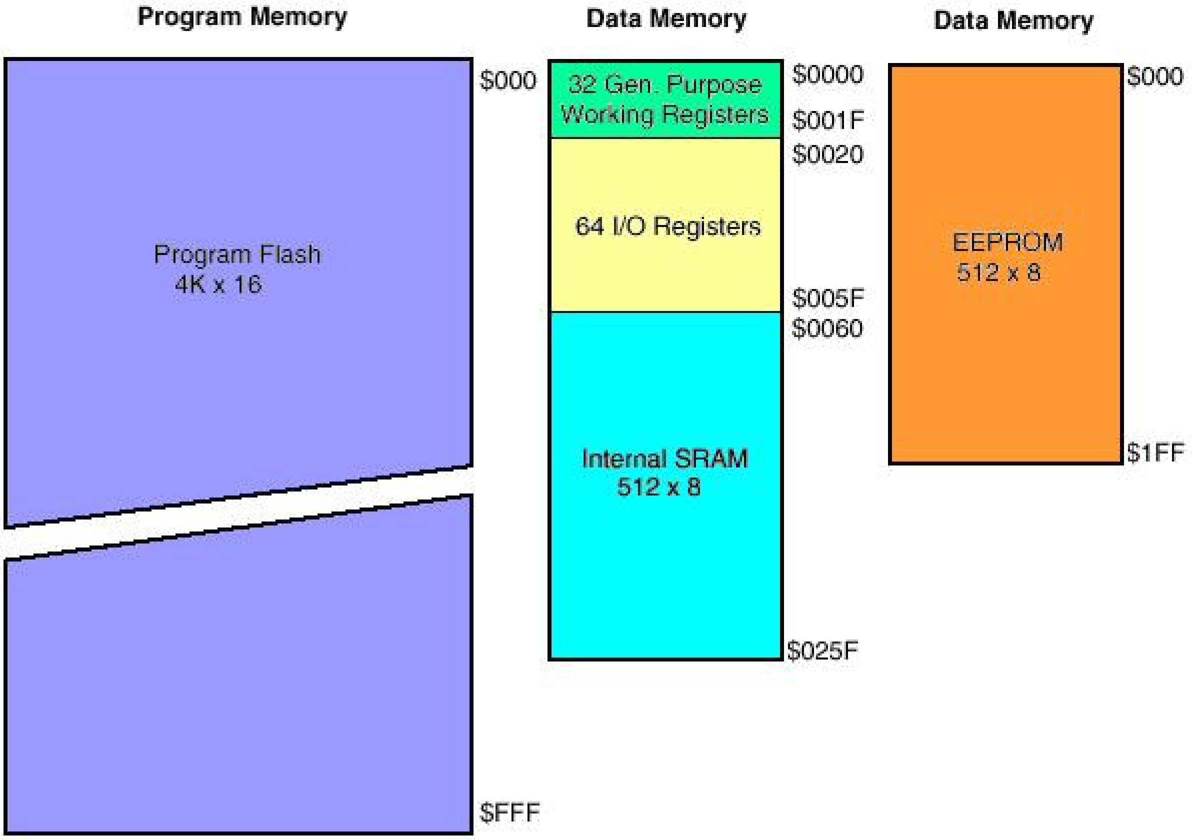
This is actually 8 parallel wires that interconnect the different parts within the IC. At any one time only one section of the 8535 is able to transmit on the bus.

Each device has its own address on the bus and is told when it can receive and when it can transmit data.

Note that with 8 bits (1 byte) only numbers up to 255 may be transmitted at once, larger numbers need to be transferred in several sequential moves.

2. Memory

There are three separate memory areas within the AVR, these are the Flash, the Data Memory and the EEPROM.



In the 8535 the Flash or program memory is 4k of words (8k bytes) of program. The AVR stores program instructions as 16 bit words. Flash Memory is like a row of lockers or pigeon holes. When the micro starts it goes to the first one to fetch an instruction, it carries out that instruction then gets the next one.

The Static RAM is a volatile store for variables within the program.

The EEPROM is a non-volatile store for variables within the program.

The 32 general purpose registers are used by your programs as temporary storage for data while the microcontroller is working on it (in many micros these are called accumulators).

If you had a line on your code to add 2 numbers e.g. z=x+y. The micro will get the contents of ram location X and store it in a register, it will get the contents of ram location Y and puts it into a second register, it will then add the 2 numbers and result will go into one of the registers, it then writes the answer from that register into memory location Z.

The 64 I/O registers are the places where you access the ports, ADC etc and their control them.

3. Special Function Registers

There are several special high speed memory registers within the microcontroller.

* Program counter: 16 bits wide, this keeps track of which instruction in flash the microcontroller is carrying out. After completing an instruction it will be incremented to point at the next location.
* Instruction register: As a program instruction is called from program memory it is held here and decoded.
* Status Register: holds information relating to the outcome of processing within the microcontroller, e.g. did the addition overflow?

4. ALU

The arithmetic logic unit carries out mathematical operations on the binary data in the registers and memory, it can add, subtract, multiply, compare, shift, test, AND, OR, NOR the data.

A simple program to demonstrate the AVR in operation

Lets take a simple program in Bascom then analyse the equivalent machine code program and then what happens within the microcontroller itself.
This program below configures all of portc pins as outputs, then counts binary in a never ending loop on the LEDs on portc.

```
Config Portc = Output    'all of portc pins as outputs
Dim Temp As Byte         'set memory aside
Temp = 0                 'set its initial value to 0
Do
   Incr Temp             'increment memory
   Portc = Temp          'write the memory to port c
Loop                     'loop forever
```

```
End
```

This is compiled into machine code, which is a long line of binary numbers. However we don't normally view the numbers as binary, it is shorter to use hexadecimal notation.

Equivalent machine code to the Bascom code above is:

```
EF0F            (1110 1111 0000 1111)
BB04
E000
BB05
9503
CFFD
```

These program commands are programmed into the microcontroller starting from the first address of the FLASH (program memory). When the micro is powered up (or reset) it starts executing instructions from that first memory location.

The equivalent assembly language to the above machine code

```
EF 0F   SER   R16        set all bits in register 16
BB 04   OUT   0x14,R16   store register 16 at address 14        (portc = output)
E0 00   LDI   R16,0x00   load immediate register 16 with 0       (temp=0)
BB 05   OUT   0x15,R16   store register 16 at address 15         (port C = temp)
95 03   INC   R16        increment register 16                   (incr temp)
CF FD   RJMP -0x0003     jump back 3 steps in the program        (back to BB05)
```

1. The microcontroller powers up and the program counter is loaded with address &H000, the first location in the flash (program memory). The first instruction is EF 0F and it is transferred into the instruction register. The program counter is then incremented by one to 0x01. The instruction is decoded and register 16 is set to all ones.
2. The next cycle of the clock occurs and BB 04 is moved from the flash into the instruction register. The program counter is incremented by one to 0x02. The instruction is decoded and R16 contents are copied to address 0x14 (0x means hex), this is the i/o register that controls the direction of port c, so now all pins of portc are outputs.
3. The next cycle of the clock occurs and E0 00 is moved into the instruction register from the flash. The program counter is incremented by one (to 0x03). The instruction is decoded and Register 16 is loaded with all 0's.
4. The next cycle of the clock occurs and BB 05 is moved into the instruction register from the flash. The program counter is incremented by one (to 0x04). The instruction is decoded and the contents of register 16 (0) are copied to address 0x15 this is the i/o register address for portc itself – so all portc goes low.
5. The next cycle of the clock occurs and 95 03 is moved into the instruction register from the flash. The program counter is incremented by one (to 0x05). The instruction is decoded and the contents of register 16 are incremented by 1 (to 01). This operation requires the use of the ALU as a mathematical calculation is involved.
6. The next cycle of the clock occurs and CF FD is moved into the instruction register from the flash. The program counter is incremented by one (to 0x06). CF FD is decoded and the program counter has 3 subtracted from it (It is 0x06 at the moment so it becomes 0x03). The sequence jumps back to number three causing a never ending loop.

Interrupts

Microcontrollers are sequential devices, they step through the program code one step after another faithfully without any problem, it is for this reason that they are used reliably in all sorts of environments. However what happens if we want to interrupt the usual program because some exception or irregular event has occurred and we want our micro to so something else briefly.

For example, a bottling machine is measuring the drink being poured into bottles on a conveyor. There could be a sensor connected to the conveyor which senses if the bottle is not there. When the bottle is expected but not there (an irregular event) the code can be interrupted so that drink is not poured out onto the conveyor.

All microcontrollers/microprocessors have hardware features called interrupts. There are two interrupt lines on the AVR, these are pind.2 and pind.3 and are called Int0 and Int1. These are connected to switches on the development pcb. When using the interrupts the first step is to set up the hardware and go into a normal programming loop. Then at the end of the code add the interrupt subroutine (called a handler)

The code to use the interrupt is:

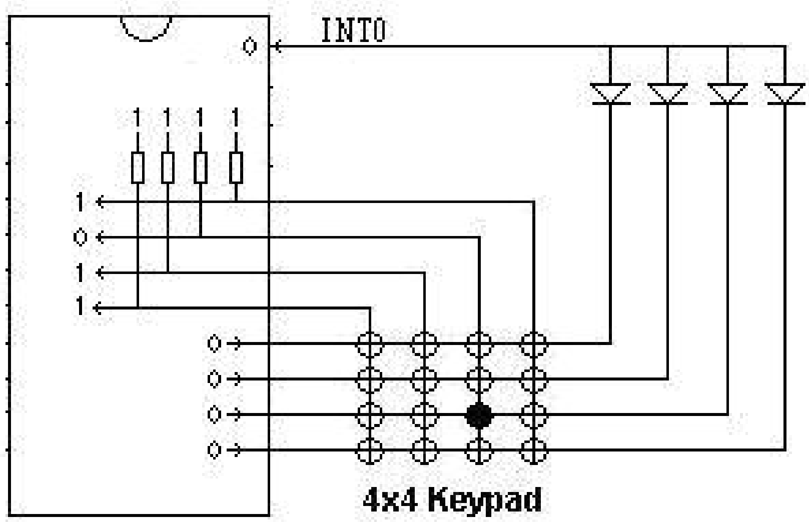
```
'--------------------------------------------------------------
' 1. Title Block
' Author: B.Collis
' Date: 9 Aug 2003
' Version: 1.0
' File Name: Interrupt_Ver1.bas
'--------------------------------------------------------------
' 2. Program Description:
' This program rotates one flashing led on portb
' when INT0 occurs the flashing led moves left
' when INT1 occurs the flashing led moves right
' 3. Hardware Features
' Eight LEDs on portb
' switches on INT0 and INT1
' 4. Software Features:
' do-loop to flash LED
' Interrupt INT0 and INT1
'--------------------------------------------------------------
' 5. Compiler Directives (these tell Bascom things about our hardware)
$crystal = 8000000            'the speed of operations inside the micro
$regfile = "m8535.dat"        ' the micro we are using
'--------------------------------------------------------------
' 6. Hardware Setups
' setup direction of all ports
Config Porta = Output 'LEDs on portA
Config Portb = Output 'LEDs on portB
Config Portc = Output 'LEDs on portC
Config Portd = Output 'LEDs on portD
Config Pind.2 = Input 'Interrupt 0
Config Pind.3 = Input 'Interrupt 1
On Int0 Int0_handler 'if at anytime an interrupt occurs handle it
On Int1 Int1_handler 'if at anytime an interrupt occurs handle it
Enable Int0 Nosave 'enable this specific interrupt to occur
Enable Int1 Nosave 'enable this specific interrupt to occur
Enable Interrupts 'enable micro to process all interrupts
' 7. hardware Aliases
' 8. initialise ports so hardware starts correctly
Porta = &B11111111 'turns off LEDs
Portb = &B11111111 'turns off LEDs
Portc = &B11111111 'turns off LEDs
Portd = &B11111111 'turns off LEDs
'----------------------------------------------------------------
' 9. Declare Constants
'----------------------------------------------------------------
' 10. Declare Variables
Dim Pattern As Byte
Dim Direction As Bit
' 11. Initialise Variables
Pattern = 254
Direction = 0
'----------------------------------------------------------------
' 12. Program starts here
Do
   If Direction = 1 Then
       Rotate Pattern , Left
       Rotate Pattern , Left
   Else
      Rotate Pattern , Right
      Rotate Pattern , Right
   End If
   Portb = Pattern 'only 1 led on
   Waitms 150
   Portb = 255 ' all leds off
   Waitms 50
Loop
'----------------------------------------------------------------
' 13. Subroutines
'----------------------------------------------------------------
' 14. Interrupts
Int0_handler:
   Direction = 1
Return

Int1_handler:
   Direction = 0
Return
```

Note that enabling interrupts is a 2 step process both the individual interrupt flag and the global interrupt flag must be enabled.

Exercise
Change the program so that only one interrupt is used to change the direction.
With the other interrupt change the speed of the pattern.

Polling versus interrupt driven architecture

With the previous keypad circuits we have had to poll (check them often) to see if a key has been pressed.



Knowing what you know about scanning keypads and interrupts how would this circuit work?

What would the code look like in the interrupt routine? (Refer back to the keypad commands)

Timer/Counters

The microcontroller has a number of pre-dimensioned variables (registers in the datasheet) that have special functions. Three of these variables are Timer0, Timer1, and Timer2.

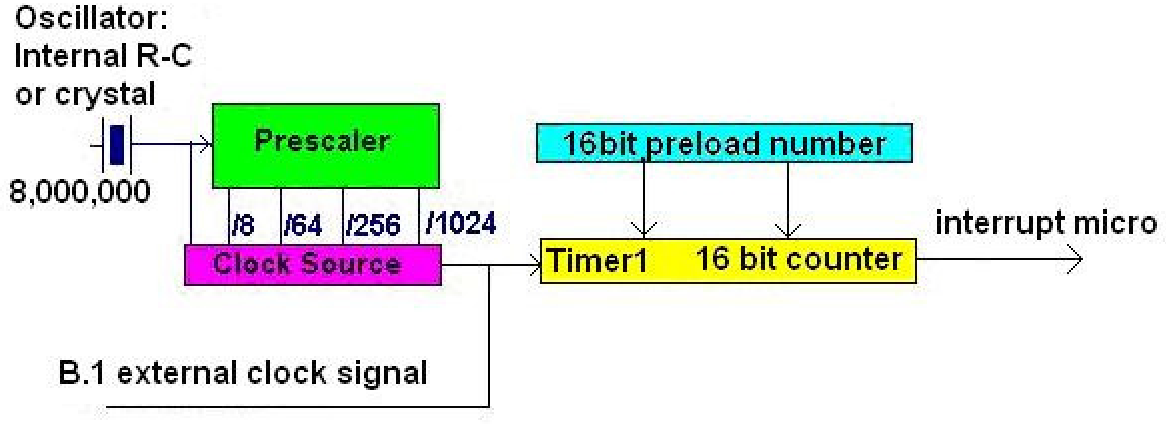
Timer0 is 8 bits so can count from 0 to 255
Timer1 is 16 bits so can count from 0 to 65535
Timer2 is 8 bits so can count from 0 to 255

The timer/counters can be written to and read from just like ordinary RAM but they have so much more to offer a designer,

- Timers can count automatically; you just give the microcontroller the command to start i.e. **enable timer1** or to stop i.e. **disable timer1**.
- You don't even have to keep track of the count in your program because when a timer overflows it will call an interrupt subroutine for you, i.e. **on ovf1 tim1_isr** (on overflow of timer1 do the subroutine called tim1_isr), remember that overflow occurs when a variable goes from its maximum value back to 0.
- The rate of counting can be from the microcontrollers internal oscillator, i.e. **timer1 = timer,** or it can count pulses from an external pin i.e. **timer1 = counter** (which is pin B.1 for timer1).
- When counting from the internal oscillator it will count at the R-C/Crystal rate or at a slower rate we can select such as the oscillator/8 or /64 or /256 or /1024, i.e. **prescale = 64** (which is 8,000,000/64 = 125,000 counts per second) or **prescale = 1024** (which is 8,000,000/1024 = 7,812 counts per second)
- The timer doesn't have to start counting from 0 it can be preloaded to start from any number less than 65535 i.e. **timer1 = 58836**, so that we can program very accurate time periods.

There are over 60 pages in the datasheet describing all the neat things timers can do!

Here is a block diagram of some of Timer1's features



Configuring the counter for use

```
Config Timer1 = Timer, Prescale = 1
On Ovf1 Tim1_isr          'on counter overflow go to Tim1_isr routine
Enable Timer1             ' enable the timer1 individual interrupt
Enable Interrupts        'allow interrupts to occur
dim preload as word
preload = 58836
```

When the 16bit counter overflows (from 65535 to 0) the micro executes the subroutine tim1_isr then returns to where it left off in the main program.

```
Tim1_isr:
  Timer1 = preload    'need to preload the counter every time
  piezo = Not piezo               'toggle the piezo to make sound
Return
```

Connect a piezo between portd.7 and ground

Example Program

```
'------------------------------------------------------------------
' 1. Title Block
' Author:        B. Collis
' Date:          18 March 2008
' File Name:   TimerV3.bas
'------------------------------------------------------------------
' 2. Program Description:
' This program uses a timer to create simple tones on a piezo
'
' 3. Hardware Features:
‘Piezo between portd.7 and ground
‘
' 4. Program Features:
' DO-LOOP to control the program repeating for ever
' use of the Timer1 to generate interrupts for sound timing
'------------------------------------------------------------------
' 5. Compiler Directives (these tell Bascom things about our hardware)
$crystal = 8000000                'the speed of operations inside the micro
$regfile = "m32def.dat"           ' the micro we are using
'------------------------------------------------------------------
' 6. Hardware Setups
' setup direction of all ports
Config Porta = Output 'Led’s on Porta
Config Portb = Output 'Led’s on Portb
Config Portc = Output 'Led’s on Portc
Config Portd = Output 'Led’s on Portc
'Configure internal timer1
Config Timer1 = Timer, Prescale = 1
On Ovf1 Tim1_isr

' 7. Hardware Aliases
Piezo Alias Portd.7 'refer to piezo not PORTd.7

' 8. initialise hardware so it starts correctly
Porta = &B11111111 'turns off LEDs
Portb = &B11111111 'turns off LEDs
Portc = &B11111111 'turns off LEDs
Portd = &B11111111 'turns off LEDs
Reset Piezo ' power off the piezo
'------------------------------------------------------------------
' 9. Declare Constants
Const Tonedelay = 350 ‘delay between tone changes
```

```
'---------------------------------------------------------------
'10. Declare Variables
Dim Preload As Word              ' size word can go up to 65535
' 11. Initialise Variables
Preload = 65
'---------------------------------------------------------------
' 12. Program starts here
Timer1=preload ‘start from required count not 0
Enable Timer1 ' enable the timer interrupt
Enable Interrupts ' allow interrupts to occur
Do
   Preload = 65              '55hz
   Waitms Tonedelay
   Preload = 650             '57hz
   Waitms Tonedelay
   Preload = 30000           '108hz
   Waitms Tonedelay
   Preload = 40000           '147hz
   Waitms Tonedelay
   Preload = 50000           '238hz
   Waitms Tonedelay
   Preload = 60000           '640hz
   Waitms Tonedelay
   Preload = 61000           '790hz
   Waitms Tonedelay
   Preload = 62000           '1020hz
   Waitms Tonedelay
   Preload = 64000           '2270hz
   Waitms Tonedelay
   Preload = 65000           '6000hz
   Waitms Tonedelay
   Disable Timer1            ' stop the sound
   Reset Piezo     'make sure power to the piezo is off
   Wait 5
   Enable Timer1 'restart the sound
Loop                          ' keep going forever

End
'---------------------------------------------------------------
' 13. Subroutines
'---------------------------------------------------------------
' 14. Interrupt subroutines
Tim1_isr:
   Timer1 = Preload            ' reload the counter (how long to wait for)
   Piezo = Not Piezo         ' toggle piezo     pin to make sound
Return
```

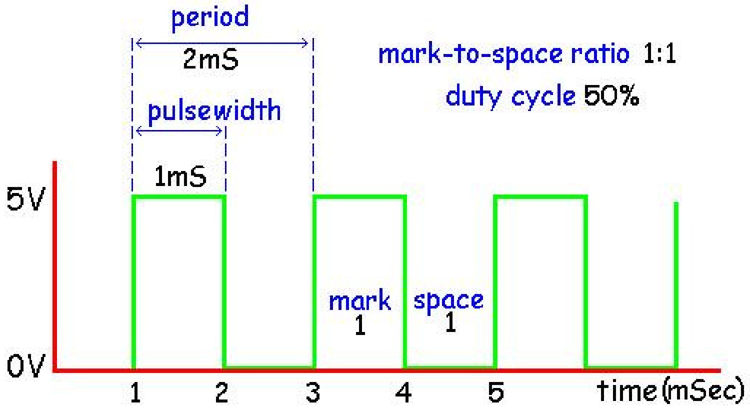
Exercise

Modify the above code to make a simple siren, use a for-next, do-loop-until or while-wend to control the changing frequency not lots of separate steps as in the above program

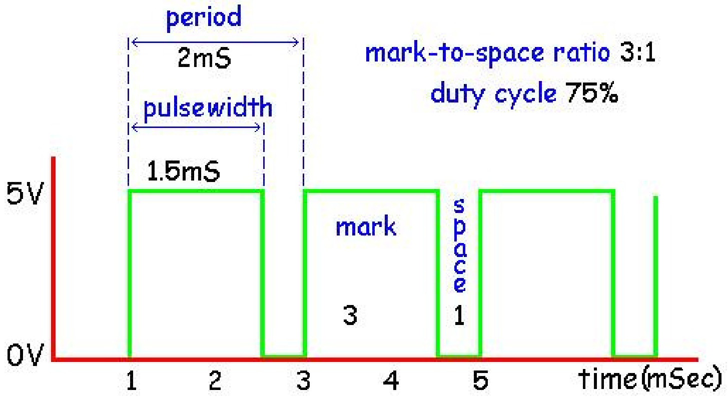
PWM - Pulse Width Modulation

To control the brightness of an LED or speed of a dc motor we could reduce the voltage to it, however this has several disadvantages especially in terms of power reduction; a better solution is to turn it on and off rapidly. If the rate is fast enough then the flickering of the LED or the pulsing of the motor is not noticeable.

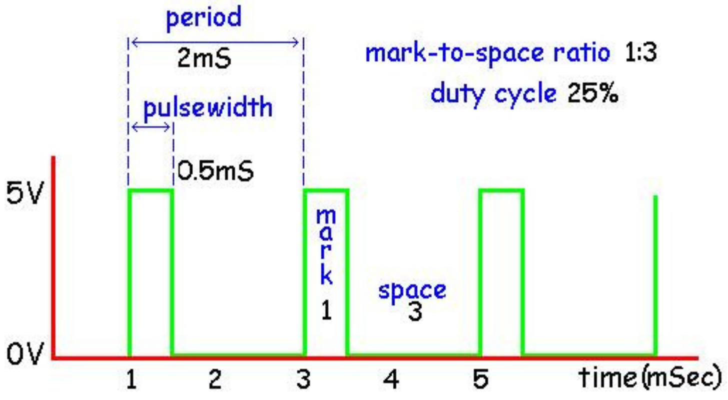
If this waveform was applied to a motor it would run at half speed.



If this waveform were applied to an LED it would be ¾ brightness



If this waveform were applied to an motor it would be run at ¼ speed



The AVR timer/counters can be used in PWM mode where the period of the wave or frequency is kept the same but the pulse width is varied. This is shown in the 3 diagrams, the period is 2mS for each of the three waveforms, yet the pulsewidth (on time) is different for each one (other modes do exist however these will not be described yet).

PWM control

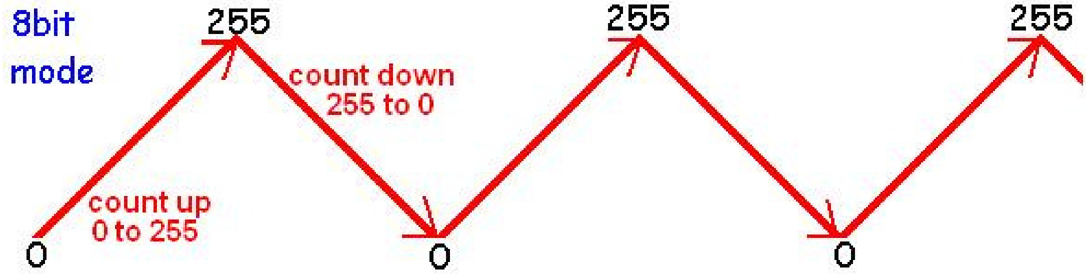
In the 8535 there are two PWM output pins attached to Timer1, these are:

- OC1A (portD.5)
- OC1B (portD.4)

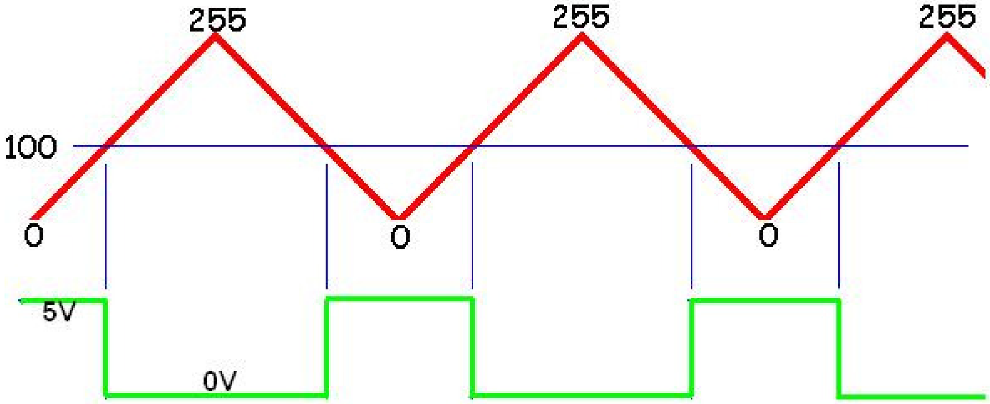
Each PWM output has independent settings for the pulse width however both will run at the same frequency.

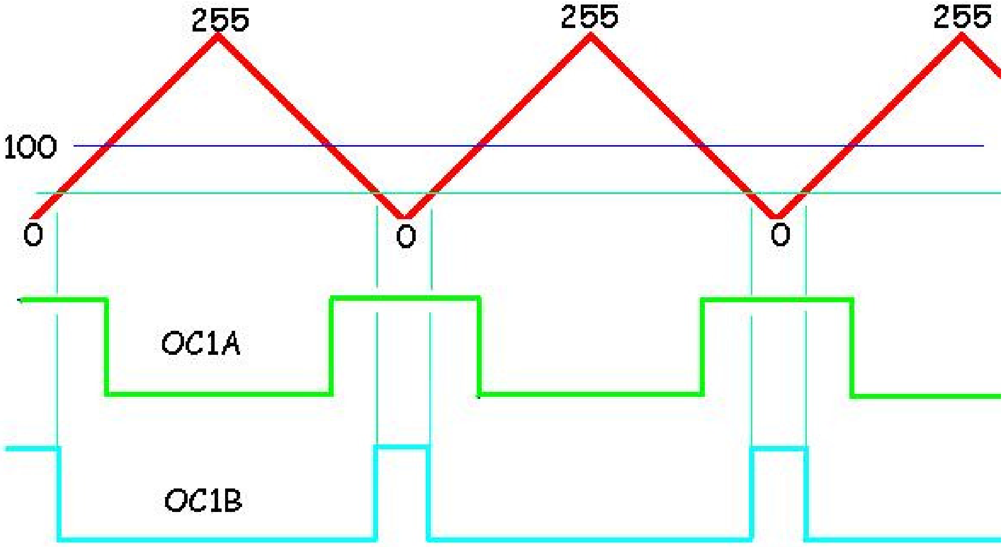
The 3 PWM modes for timer1 discussed here are the 8, 9 & 10 bit mode.

- In 8 bit mode the counter counts from 0 to 255 then back down to 0.
- In 9 bit mode the counter counts from 0 to 511 then back down to 0.
- In 10 bit mode the counter counts from 0 to 1023 then back down to 0.



The programmer sets a point from 0 to 255 at which the output will change from high to low. If the value were set to 100 then the output pulse on portd.5 (OC1A) would switch from 0Volts (0) to 5 Volts (1) as in the next picture.





The lines of code to get the above waveforms on OC1A and OC1B would be

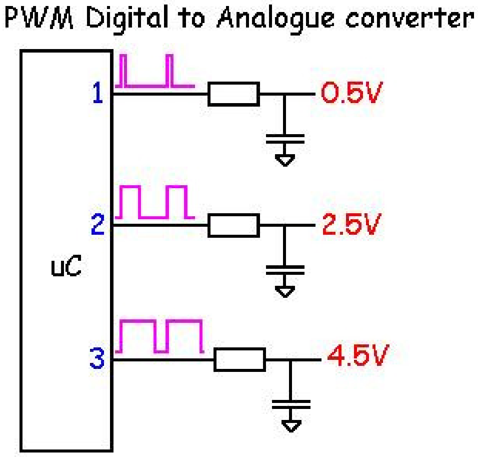
- Config Timer1 = Pwm , Pwm = 8 , Compare A Pwm = Clear Up , Compare B Pwm = Clear up , Prescale = 1024
- Compare1a = 100
- Compare1b = 10

Different values for frequency based upon input crystal and prescale value

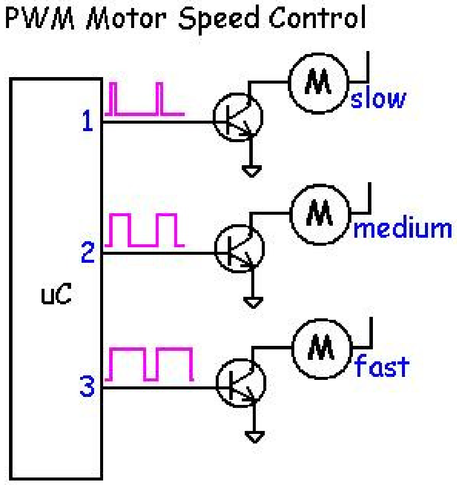
OUTPUT FREQUENCY (Hz) for a crystal frequency of 7,372,800

		Prescale Value				
		1	8	64	256	1024
	8 Bit	14,456	1,807	226	56	14
PWM	9 Bit	7,214	902	113	28	7
	10 Bit	3604	450	56	14	4

Uses for PWM



A pulse is used to charge a capacitor through a resistor, when the pulse is high the capacitor will charge, when it is low the capacitor will discharge, the wider the pulse the longer the capacitor charges and the higher the voltage will be.



The width of the pulse determines the average DC voltage getting to the motor which in turn slows or speeds up the motor. the advantage of using PWM rather than reducing the actual voltage is that torque (power) of the motor maintained at low speeds.

Period - the time from one point in the waveform to the same point in the next cycle of the waveform.

Frequency - the inverse of the period, if period = 2mS the frequency = 1/0.002 = 500 Hz (Hertz).

Pulse width - the length of time the pulse is high or on. The 'mark' time.

Duty cycle - the on time of the pulse as a proportion of the whole period of the waveform.

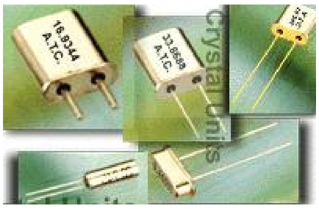
AVR Clock/Oscillator

The AVR executes instructions at the rate set by the system clock (oscillator). There are a number of different ways that this clock can be set up using either internal components of the micro or external components. These are:

- Internal Resistor-Capacitor (lesser accuracy)
- External RC
- External Ceramic Resonator
- External Crystal (more accuracy)

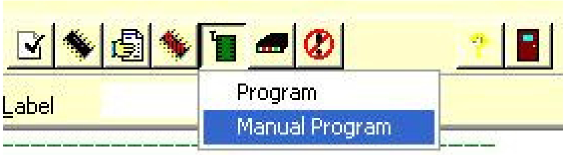
ceramic resonator

crystals



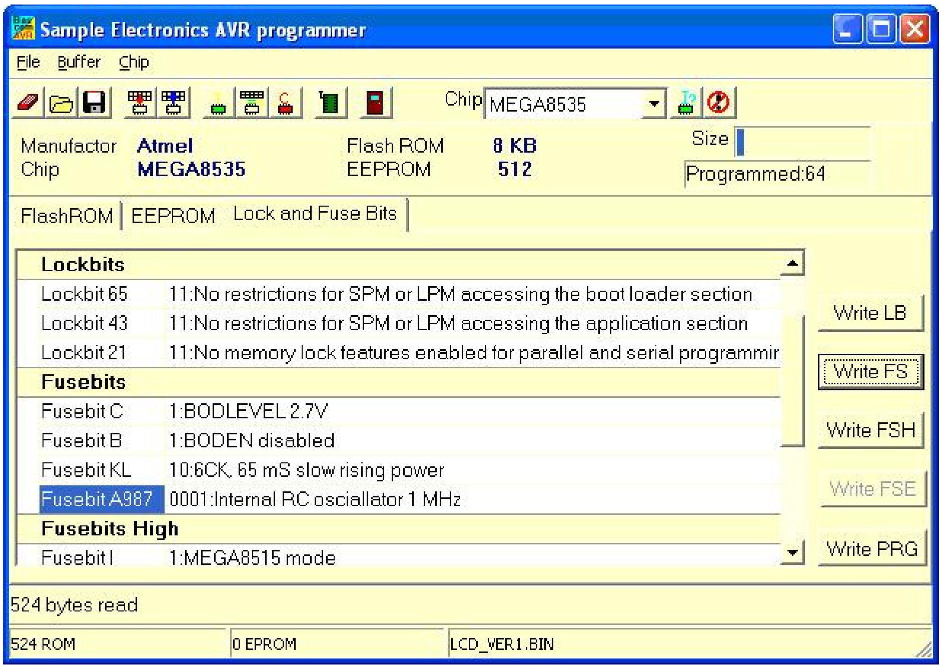
Within the micro reprogrammable fuse links (just like the links on a computer motherboard but set via software) are used to determine which method is used.

The ATMega8535-16PI clock can range up to 16MHz, however initially it is configured to run from the internal RC clock at a 1MHz rate.



In BASCOM when the micro is connected and powered up the settings can be changed by selecting MANUAL PROGRAM.

From the window that appears select the LOCK AND FUSE BITS tab. Bascom will then read the current settings.



Manufactor Atmel
Chip MEGA8535
Flash ROM 8 KB
EEPROM 512
Programmed:64

Lockbits	
Lockbit 65	11:No restrictions for SPM or LPM accessing the boot loader section
Lockbit 43	11:No restrictions for SPM or LPM accessing the application section
Lockbit 21	11:No memory lock features enabled for parallel and serial programmir
Fusebits	
Fusebit C	1:BODLEVEL 2.7V
Fusebit B	1:BODEN disabled
Fusebit KL	10:6CK, 65 mS slow rising power
Fusebit A987	0001:Internal RC osciallator 1 MHz
Fusebits High	
Fusebit I	1:MEGA8515 mode

524 bytes read

524 ROM | 0 EPROM | LCD_VER1.BIN

The Internal RC oscillator may be changed to 8MHz by selecting the line in the window and using the drop down that appears to change it to 8MHz.

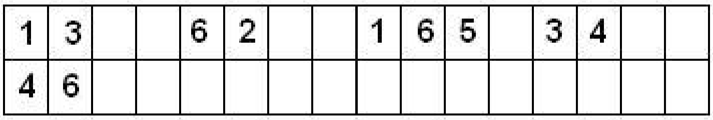
After changing the Fusebit settings select the Write FS button. After it has programmed the fusebits, select the FlashRom tab before exiting

(YOU MAY NEED TO DISABKLE THE JTAG SETTING AS WELL)

DO NOT CHANGE ANYTHING ELSE, YOU RISK STUFFING UP YOUR MICRO!

Assignment – Maths In The Real World

5 numbers are to be entered into memory via the 5 buttons and then displayed on the LCD. Press btn A to move between the 5 numbers. Btn B to increment the number, btn C to decrement the number. The maximum number will be 255, the minimum number will be 1. The display looks like this.



1	3			6	2			1	6	5		3	4		
4	6														

The current code is listed below, load it into your microcontroller to see how it works. Then go onto the next exercise.

```
'-----------------------------------------------------------------
' 1. Title Block
' Author: B.Collis
' Date: 1 June 2005
' File Name: numberentryV0.1.bas

'-----------------------------------------------------------------
' 2. Program Description:
' enters 5 numbers into variables A,B,C,D,E and display them
' 3. Hardware Features:
' LEDS
' LDR, Thermistor on ADC
' 5 switches
' LCD
' 4. Program Features
' do-loop to keep program going forever
' debounce to test switches
' if-then-endif to test variables

'-----------------------------------------------------------------
' 5. Compiler Directives (these tell Bascom things about our hardware)
$crystal = 8000000
$regfile = "m8535.dat"

'-----------------------------------------------------------------
' 6. Hardware Setups
' setup direction of all ports
Config Porta = Output 'LEDs on portA
Config Portb = Output 'LEDs on portB
Config Portc = Output 'LEDs on portC
Config Portd = Output 'LEDs on portD
'config inputs
Config Pina.0 = Input ' ldr
Config Pind.2 = Input 'switch A
Config Pind.3 = Input 'switch B
Config Pind.6 = Input 'switch C
Config Pinb.1 = Input 'switch D
Config Pinb.0 = Input 'switch E


'LCD
Config Lcdpin = Pin , Db4 = Portc.4 , Db5 = Portc.5 , Db6 = Portc.6 , Db7 = Portc.7 , E = Portc.1 , Rs
= Portc.0
Config Lcd = 40 * 2 'configure lcd screen

' 7. Hardware Aliases
Led3 Alias Portd.4
Sw_c Alias Pind.2
Sw_b Alias Pind.3
Sw_a Alias Pind.6

Spkr Alias Portd.7 'refer to spkr not PORTd.7
Cursor Off
' 8. initialise ports so hardware starts correctly
Porta = &B11111100 'turns off LEDs ignores ADC inputs
Portb = &B11111111 'turns off LEDs activate pullups switches
Portc = &B11111111 'turns off LEDs
Portd = &B11111111 'turns off LEDs activate pullups switches
Cls 'clear lcd screen

'-------------------------------------------------------------------
' 9. Declare Constants
Const Btndelay = 15

'-------------------------------------------------------------------
' 10. Declare Variables
Dim State As Byte
Dim A As Byte
Dim B As Byte
Dim C As Byte
Dim D As Byte
Dim E As Byte
Dim Sum As Byte
' 11. Initialise Variables
State = 0
'-------------------------------------------------------------------
```

```
' 12. Program starts here
Cls
        Do
                Debounce Sw_a , 0 , Swa_press , Sub
                Debounce Sw_b , 0 , Swb_press , Sub
                Debounce Sw_c , 0 , Swc_press , Sub
        Loop
End
'----------------------------------------------------------------
' 13. Subroutines
Disp_numbrs:
        Locate 1 , 1
        Lcd A
        Locate 1 , 5
        Lcd B
        Locate 1 , 9
        Lcd C
        Locate 1 , 13
        Lcd D
        Locate 2 , 1
        Lcd E
Return
Swa_press:
        If State < 5 Then
        Incr State
        Else
        State = 1
        End If
        Gosub Disp_numbrs
Return
Swb_press:
        Select Case State
                Case 1 : Incr A
                Case 2 : Incr B
                Case 3 : Incr C
                Case 4 : Incr D
                Case 5 : Incr E
        End Select
Gosub Disp_numbrs
Return
Swc_press:
        Select Case State
        Case 1 : Decr A
        Case 2 : Decr B
        Case 3 : Decr C
        Case 4 : Decr D
        Case 5 : Decr E
        End Select
        Gosub Disp_numbrs
Return
```

Math Assignment - Part 1

The program as given to you has a few bugs for you to fix

1. After the power is applied the lcd is blank it should display the 5 numbers.
Write your code here that fixes this

2. The display does not blank any zeros when the numbers go from 100 to 99 and 10 to 9. Fix this and explain here how you did it.

3. The numbers start at 0, they need to start at 1, fix this and explain here how you did it

4. Make the maximum number that can be entered 200, Write the code here that fixes this.

Math Assignment - Part 2

At the moment the user must press the button to increment or decrement the numbers one at a time. There is no auto-repeat feature included in the debounce function. Add some form of repeat feature so that the user can hold a button and after a short delay the numbers will increase/decrease until the button is released.

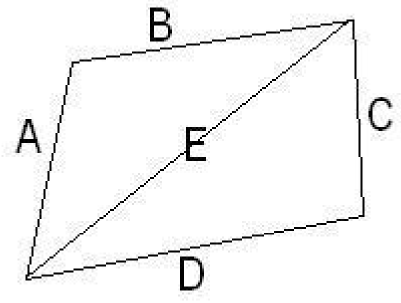
You may want to try and do this using if pin=0 then..... rather than debounce.

Make your routine as generic or portable as possible, so that it could be easily transferred to other programs.

Explain how your auto-repeat code works.

Math Assignment - Part 3

This program is going to be used by a groundsman to calculate the area of a piece of land so that he can work out the amount of grass seed to buy. He will use your program and pace out the 4 sides: a,b,c,d, and the diagonal e.



the formulae to work out the area of a triangle is:
s= (a+b+e)/2
Area of first triangle = sqroot(s(s-a)(s-b)(s-e))

t= (c+d+e)/2
Area of second triangle = sqroot(t(t-c)(t-d)(t-e))

1. All the calculations must be in one subroutine.
2. You will also need to dimension some temporary variables to help you, e.g.
   dim sngl1 as single, sngl2 as single, sngl3 as single
3. Bascom can only do one arithmetic equation per line so you will need to break up each equation into individual parts.

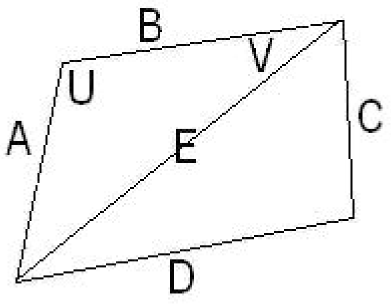
Here is half of the routine.

```
calcarea:
        s= a+b
        s=s+e
        s=s/2
        singl1=s-a
        s=s*singl1          's(s-a)
        singl2=s-b
        s=s*singl2          's(s-a)(s-b)
        singl3=s-e
        s=s*singl3          ' s(s-a)(s-b)(s-e)
        area=sqr(s)        'area of the first triangle
return
```

1. You complete the rest of the equation to work out the area of the second triangle and then work out the total area for the whole shape.
2. Modify your program to automatically update the lcd with the calculated area as the grounds man enters the data for each variable. Explain where in your code you put the changes to make this update happen all

Math Assignment - Part 4

When the groundsman gets back to the office, he needs to draw a plan of the area. To do this he needs the angles within the shape.



Using the cosine rule we can calculate these for him.

U is the angle opposite side E
$E^2 = A^2 + B^2 - 2AB\cos(U)$

V is the angle opposite side E
$A^2 = E^2 + B^2 - 2EB\cos(V)$

1. calculate each of the 6 angles
2. U will be in radians, convert each angle to degrees.
3. display them on the LCD

Write the code for calculating one of the angles below.

Math Assignment - Part 5

When the groundsman has calculated the area and angles, the data must be stored into eeprom so that it will be there when he goes back to his office.

1. To do this you must declare some new variables e.g. eep_a, eep_b, ... and dimension these **dim eep_A as eram byte.**
2. add a state and subroutine to your program which copies the variables A,B,C.etc into the corresponding eeprom variables eep_a, eep_b, eep_c etc. Write it below (you may want to change the fuselink in the AVR that causes the EEPROM to be cleared every time the AVR is reprogrammed)

3. add a state and subroutine to your program that reads the eeprom variables and copies them into the ram variables. Copy the subroutine here

Math Assignment - Part 6

Create a simple menu that allows the groundsman to select the operation to perform

- enter 5 lengths
- calculate and view the area
- calculate and view the angles
- store the values into eeprom
- read the values from eeprom

You must use a state variable to manage the program flow. Explain your code below.

Extension exercise

Give the groundsman the option to store multiple areas of land

Bascom Keyword Reference

1WIRE

1Wire routines allow you to communicate with Dallas 1wire chips.
1WRESET , 1WREAD , 1WWRITE , 1WSEARCHFIRST , 1WSEARCHNEXT ,1WVERIFY , 1WIRECOUNT

Conditions

Conditions execute a part of the program depending on the condition
IF-THEN-ELSE-END IF , WHILE-WEND , ELSE , DO-LOOP , SELECT CASE - END SELECT , FOR-NEXT

Configuration

Configuration command initialize the hardware to the desired state.
CONFIG , CONFIG ACI , CONFIG ADC , CONFIG BCCARD , CONFIG CLOCK , CONFIG COM1 , CONFIG COM2 , CONFIG DATE , CONFIG PS2EMU , CONFIG ATEMU , CONFIG I2CSLAVE , CONFIG GRAPHLCD , CONFIG KEYBOARD , CONFIG TIMER0 , CONFIG TIMER1 , CONFIG LCDBUS , CONFIG LCDMODE , CONFIG 1WIRE , CONFIG LCD , CONFIG SERIALOUT , CONFIG SERIALOUT1 , CONFIG SERIALIN , CONFIG SERIALIN1 , CONFIG SPI , CONFIG LCDPIN , CONFIG SDA , CONFIG SCL , CONFIG DEBOUNCE , CONFIG WATCHDOG , CONFIG PORT , COUNTER0 AND COUNTER1 , CONFIG TCPIP

Conversion

A conversion routine is a function that converts a number or string.
BCD , GRAY2BIN , BIN2GRAY , BIN , MAKEBCD , MAKEDEC , MAKEINT , FORMAT , FUSING , BINVAL , CRC8 , CRC16 , CRC32 , HIGH , HIGHW , LOW

DateTime

Date Time routines can be used to calculate with date and/or times.
DATE , TIME , DATE$, TIME$, DAYOFWEEK , DAYOFYEAR , SECOFDAY , SECELAPSED , SYSDAY , SYSSEC , SYSSECELAPSED

Delay

Delay routines delay the program for the specified time.
WAIT , WAITMS , WAITUS , DELAY

Directives

Directives are special instructions for the compiler. They can override a setting from the IDE.
$ASM , $BAUD , $BAUD1 , $BGF , $BOOT , $CRYSTAL , $DATA , $DBG , $DEFAULT , $EEPLEAVE , $EEPROM , $EEPROMHEX , $EXTERNAL , $HWSTACK , $INC , $INCLUDE , $INITMICRO , $LCD , $LCDRS , $LCDPUTCTRL , $LCDPUTDATA , $LCDVFO , $LIB , $LOADER , $LOADERSIZE , $MAP , $NOINIT , $NORAMCLEAR , $PROG , $PROGRAMMER , $REGFILE , $ROMSTART $SERIALINPUT, $SERIALINPUT1 , $SERIALINPUT2LCD , $SERIALOUTPUT , $SERIALOUTPUT1 , $SIM , $SWSTACK , $TIMEOUT , $TINY , $WAITSTATE , $XRAMSIZE , $XRAMSTART , $XA

File

File commands can be used with AVR-DOS, the Disk Operating System for AVR.
BSAVE , BLOAD , GET , VER , , DISKFREE , DIR , DriveReset , DriveInit , , LINE INPUT , INITFILESYSTEM , EOF , WRITE , FLUSH , FREEFILE , FILEATTR , FILEDATE , FILETIME , FILEDATETIME , FILELEN , SEEK , KILL , DriveGetIdentity , DriveWriteSector , DriveReadSector , LOC , LOF , PUT , OPEN , CLOSE

Graphical LCD

Graphical LCD commands extend the normal text LCD commands.
GLCDCMD , GLCDDATA , SETFONT , LINE , PSET , SHOWPIC , SHOWPICE , CIRCLE

I2C

I2C commands allow you to communicate with I2C chips with the TWI hardware or with emulated I2C hardware.
I2CINIT , I2CRECEIVE , I2CSEND , I2CSTART,I2CSTOP,I2CRBYTE,I2CWBYTE

IO

I/O commands are related to the I/O pins of the processor.

ALIAS , BITWAIT , TOGGLE , RESET , SET , SHIFTIN , SHIFTOUT , DEBOUNCE , PULSEIN , PULSEOUT

Micro

Micro statements are highly related to the micro processor.

IDLE , POWERDOWN , POWERSAVE , ON INTERRUPT , ENABLE , DISABLE , START , END , VERSION , CLOCKDIVISION , CRYSTAL , STOP

Memory

Memory functions set or read RAM , EEPROM or flash memory.

WRITEEEPROM , CPEEK , CPEEKH , PEEK , POKE , OUT , READEEPROM , DATA , INP , READ , RESTORE , LOOKDOWN , LOOKUP , LOOKUPSTR , CPEEKH , LOAD , LOADADR , LOADLABEL , LOADWORDADR , MEMCOPY

Remote Control

Remote control statements send or receive IR commands for remote control.

RC5SEND , RC6SEND , GETRC5 , SONYSEND

RS-232

RS-232 are serial routines that use the UART or emulate a UART.

BAUD , BAUD1, BUFSPACE , ECHO , WAITKEY , ISCHARWAITING , INKEY , INPUTBIN , INPUTHEX , INPUT , PRINT , PRINTBIN , SERIN , SEROUT , SPC

SPI

SPI routines communicate according to the SPI protocol with either hardware SPI or software emulated SPI.

SPIIN , SPIINIT , SPIMOVE , SPIOUT

String

String routines are used to manipulate strings.

ASC , UCASE , LCASE , TRIM , SPLIT , LTRIM , INSTR , SPACE , STRING , RTRIM , LEFT , LEN , MID , RIGHT , VAL , STR , CHR , CHECKSUM , HEX , HEXVAL

TCP/IP

TCP/IP routines can be used with the W3100/IIM7000/IIM7010 modules.

BASE64DEC , BASE64ENC , IP2STR , UDPREAD , UDPWRITE , UDPWRITESTR , TCPWRITE , TCPWRITESTR , TCPREAD , GETDSTIP , GETDSTPORT , SOCKETSTAT , SOCKETCONNECT , SOCKETLISTEN , GETSOCKET , CLOSESOCKET , SETTCP , GETTCPREGS , SETTCPREGS

Text LCD

Text LCD routines work with the normal text based LCD displays.

HOME , CURSOR , UPPERLINE , THIRDLINE , INITLCD , LOWERLINE , LCD , LCDAT , FOURTHLINE , DISPLAY , LCDCONTRAST , LOCATE , SHIFTCURSOR , DEFLCDCHAR , SHIFTLCD , CLS

Trig & Math

Trig and Math routines worj with numeric variables.

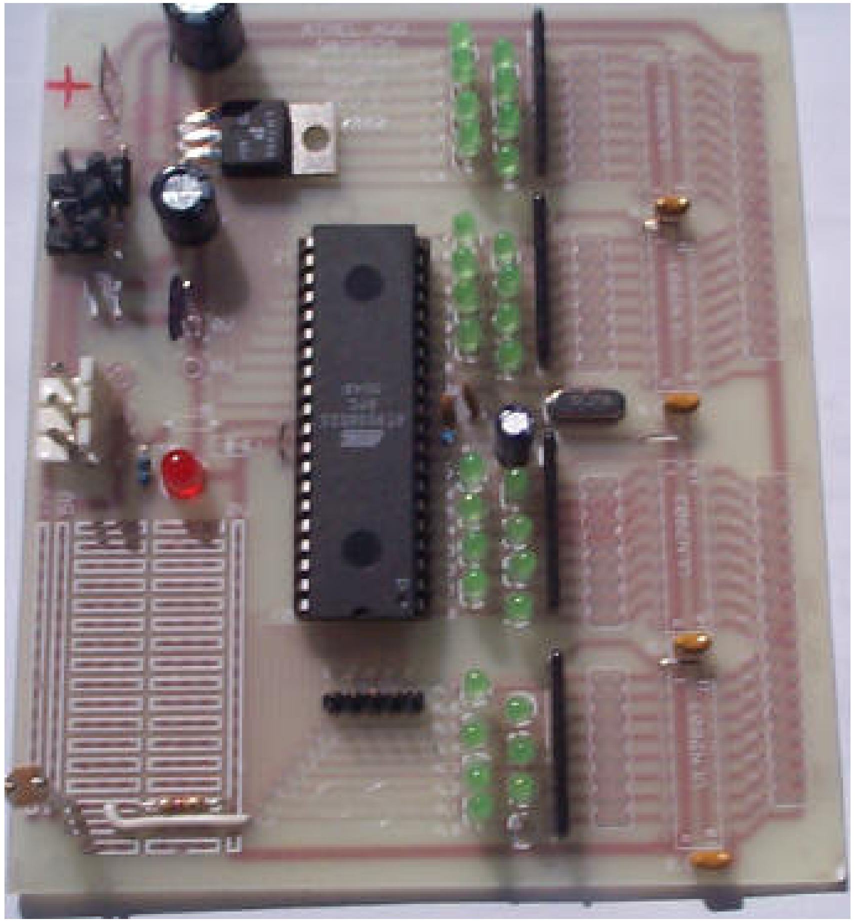
ACOS , ASIN , ATN , ATN2 , EXP , RAD2DEG , FRAC , TAN , TANH , COS , COSH , LOG , LOG10 , ROUND , ABS , INT , MAX , MIN , SQR , SGN , POWER , SIN , SINH , FIX , INCR , DECR , DEG2RAD

Various

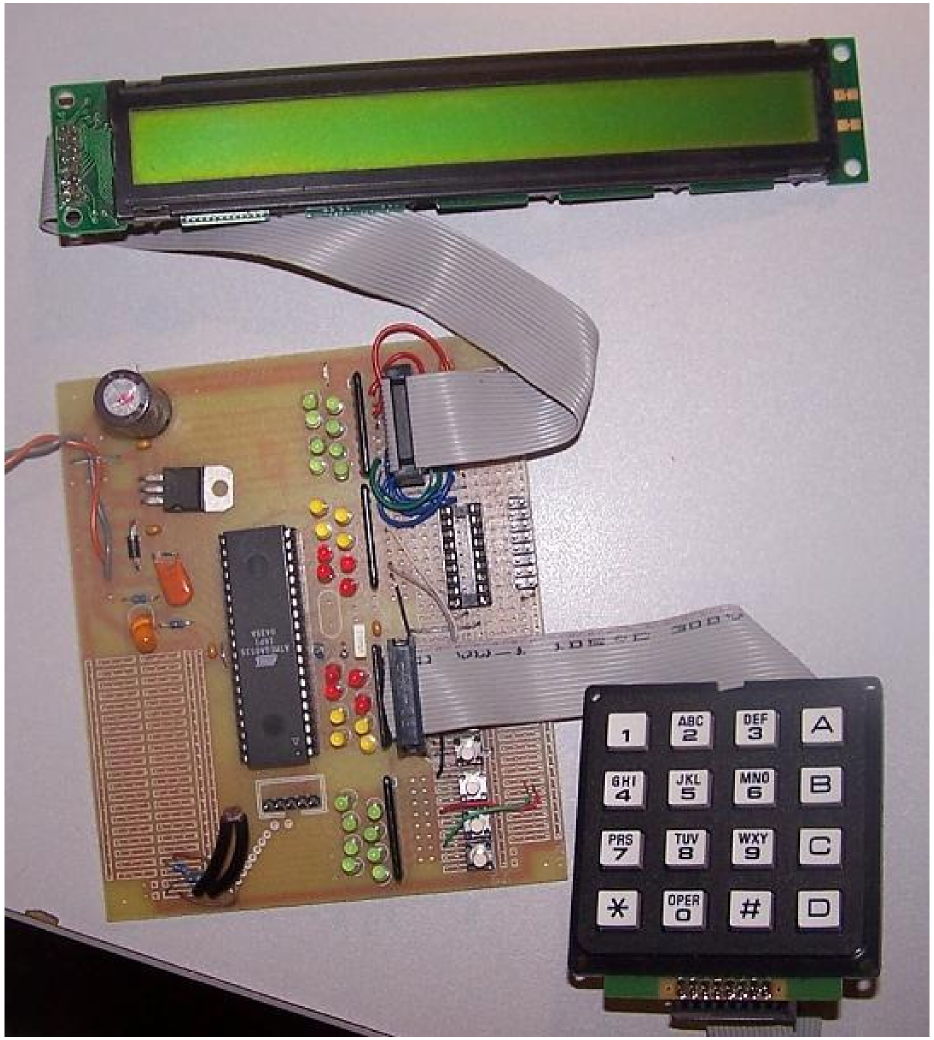
This section contains all statements that were hard to put into another group

CONST , DBG , DECLARE FUNCTION , DECLARE SUB , DEFXXX , DIM , DTMFOUT , EXIT , ENCODER , GETADC , GETKBD , GETATKBD , GETRC , GOSUB , GOTO , LOCAL ,ON VALUE , POPALL , PS2MOUSEXY , PUSHALL , RETURN , RND , ROTATE , SENDSCAN , SENDSCANKBD , SHIFT , SOUND , STCHECK , SUB , SWAP , VARPTR , X10DETECT , X10SEND , READMAGCARD , REM , BITS , BYVAL , CALL , #IF , #ELSE , #EN

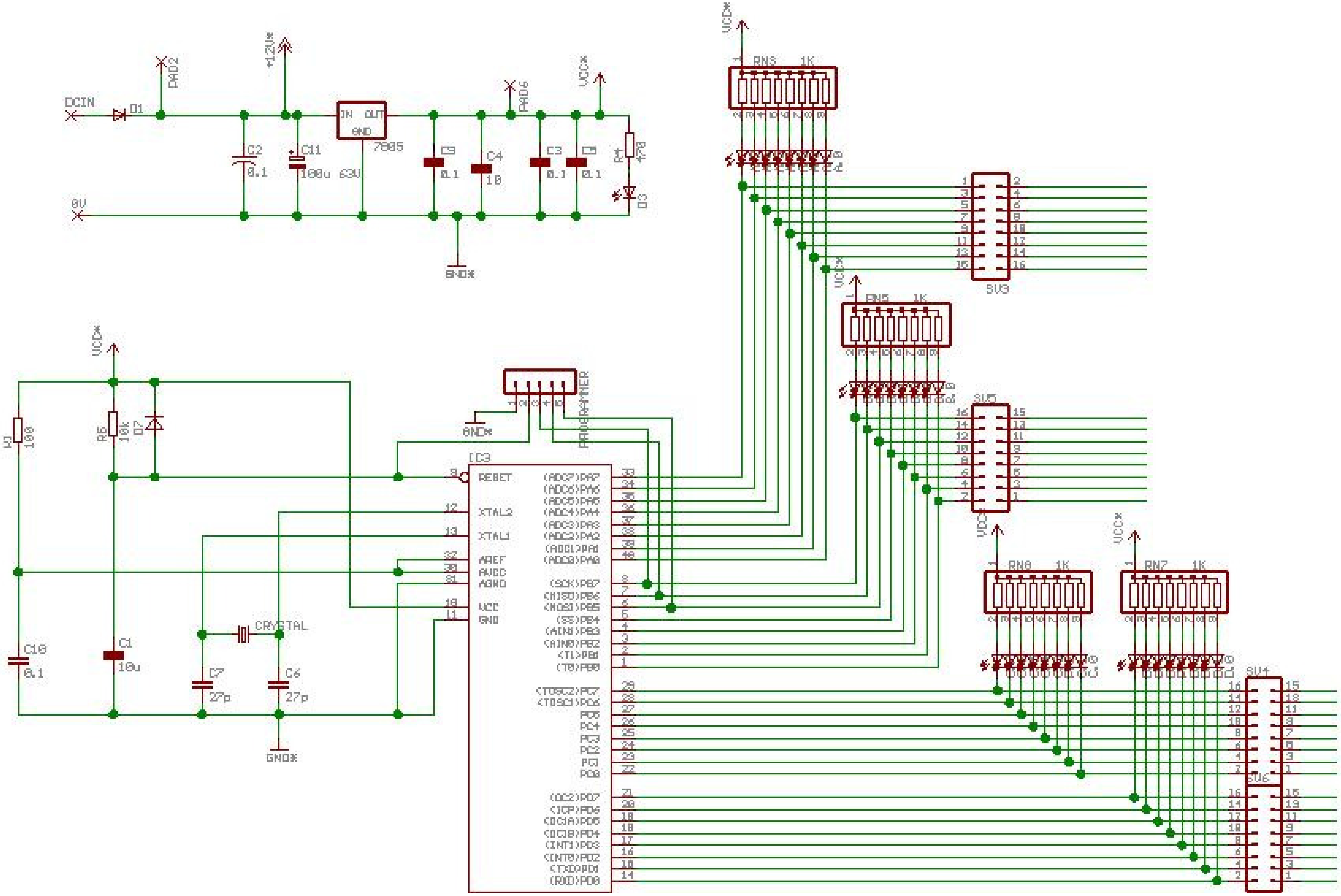
AVR Development Boards we can use



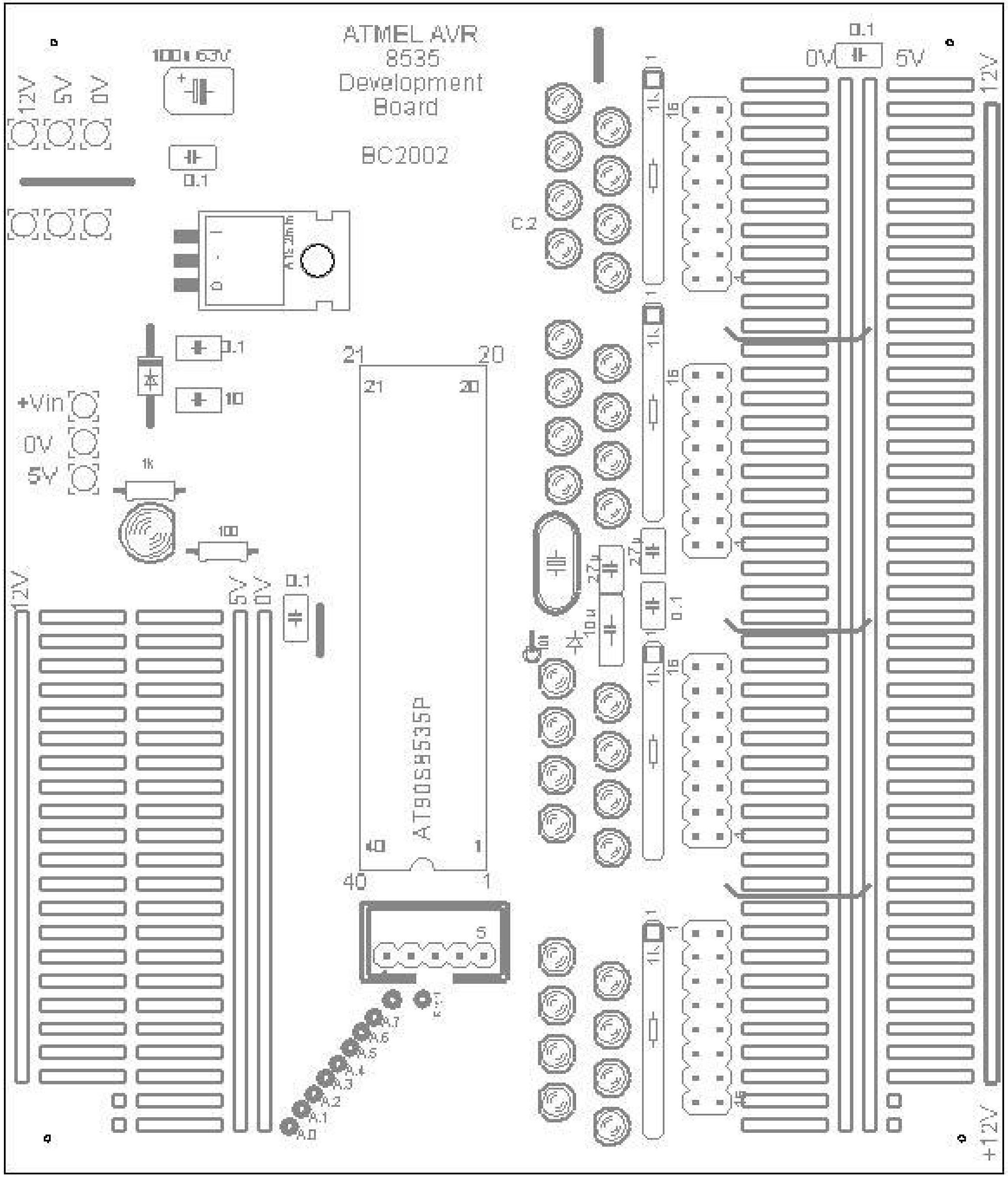
8535 Version 1



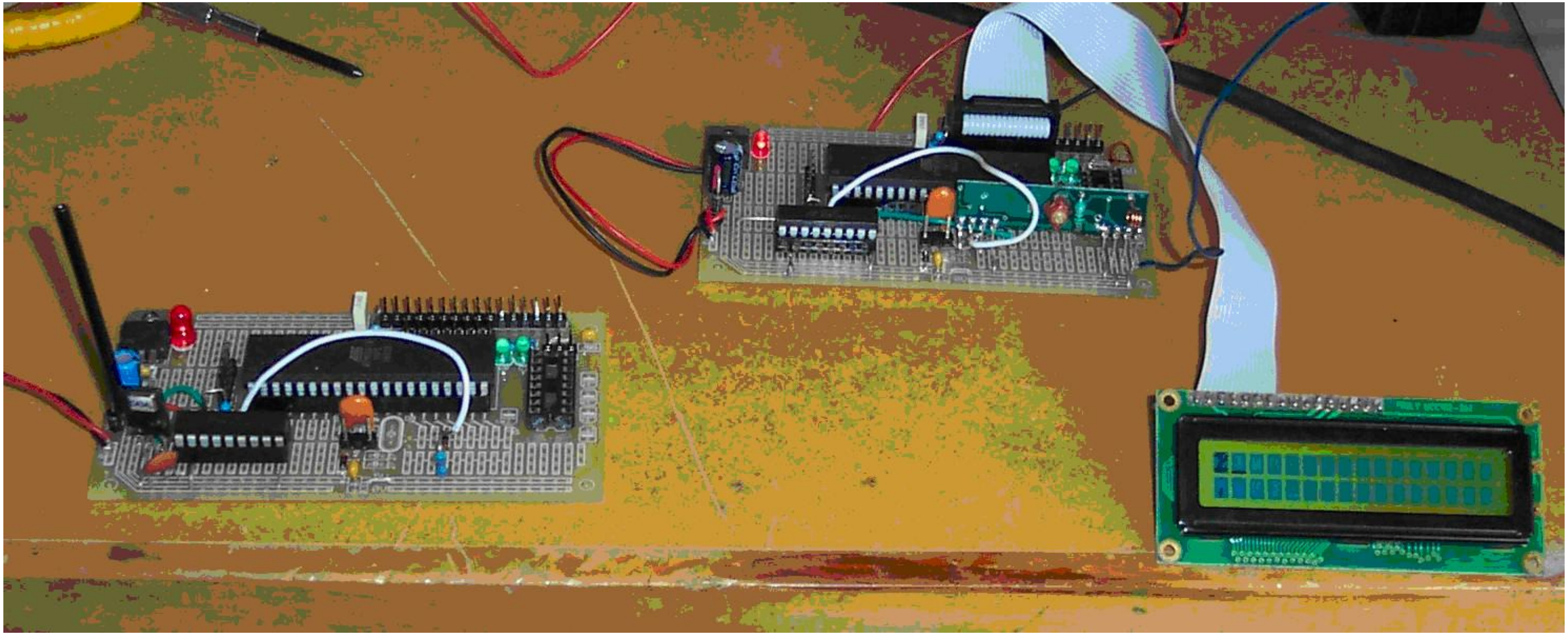
8535 Version 1A

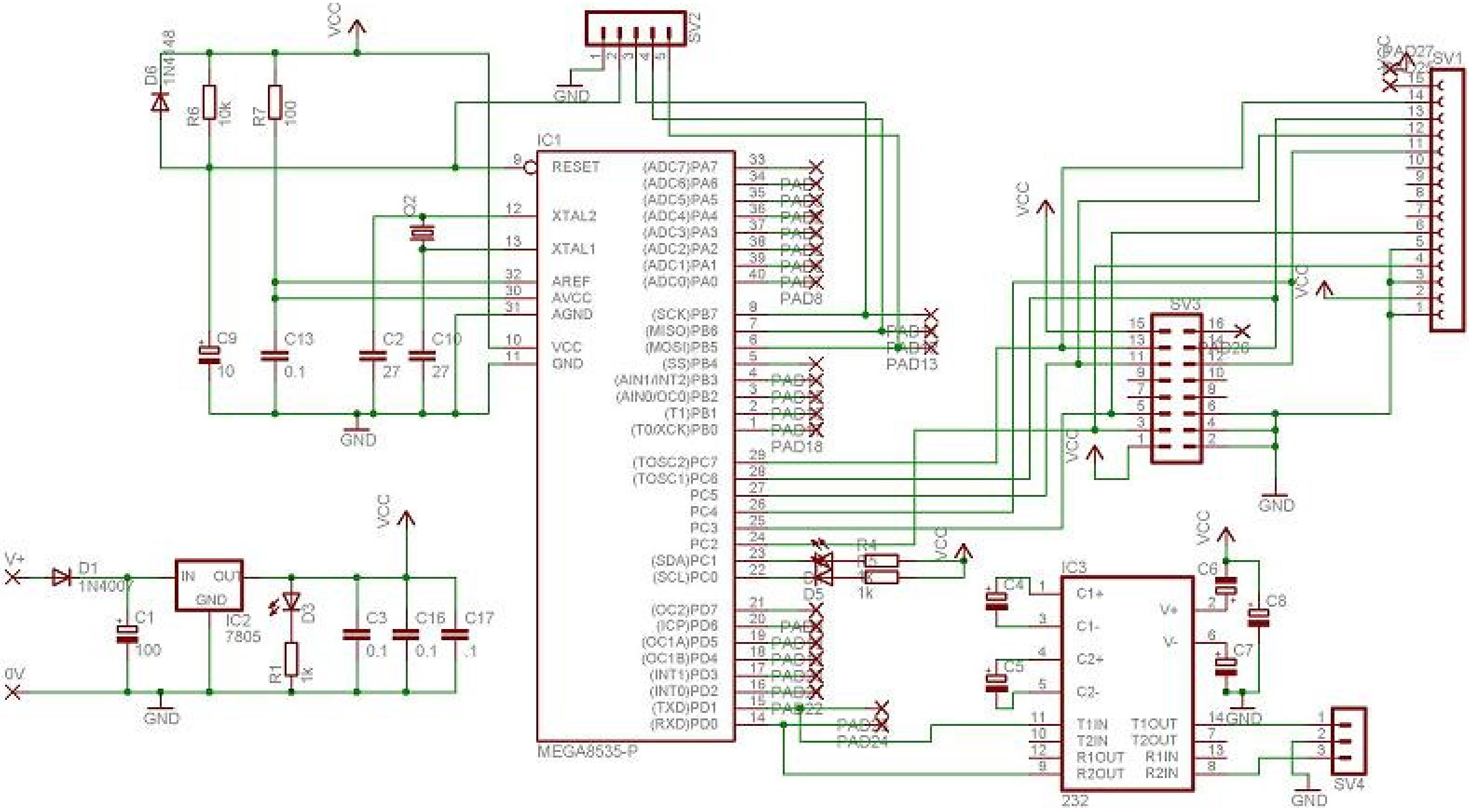


DCIN
D1
PAD2
+12V*
IN
OUT
GND
7805
PAD6
VCC*
C2
0.1
C11
100u 63V
C9
0.1
C4
10
C3
0.1
C
0.1
R4
470
D3
0V
GND*
VCC*
RN3
1K
SV3
RN5
1K
SV5
RN6
1K
RN7
1K
SV4
SV6
R1
100
R5
10k
D7
C10
0.1
C1
10u
C7
27p
C6
27p
CRYSTAL
GND*
PROGRAMMER
GND*
IC3
RESET
XTAL2
XTAL1
AREF
AVCC
AGND
VCC
GND
(ADC7)PA7
(ADC6)PA6
(ADC5)PA5
(ADC4)PA4
(ADC3)PA3
(ADC2)PA2
(ADC1)PA1
(ADC0)PA0
(SCK)PB7
(MISO)PB6
(MOSI)PB5
(SS)PB4
(AIN1)PB3
(AIN0)PB2
(T1)PB1
(T0)PB0
(TOSC2)PC7
(TOSC1)PC6
PC5
PC4
PC3
PC2
PC1
PC0
(OC2)PD7
(ICP)PD6
(OC1A)PD5
(OC1B)PD4
(INT1)PD3
(INT0)PD2
(TXD)PD1
(RXD)PD0

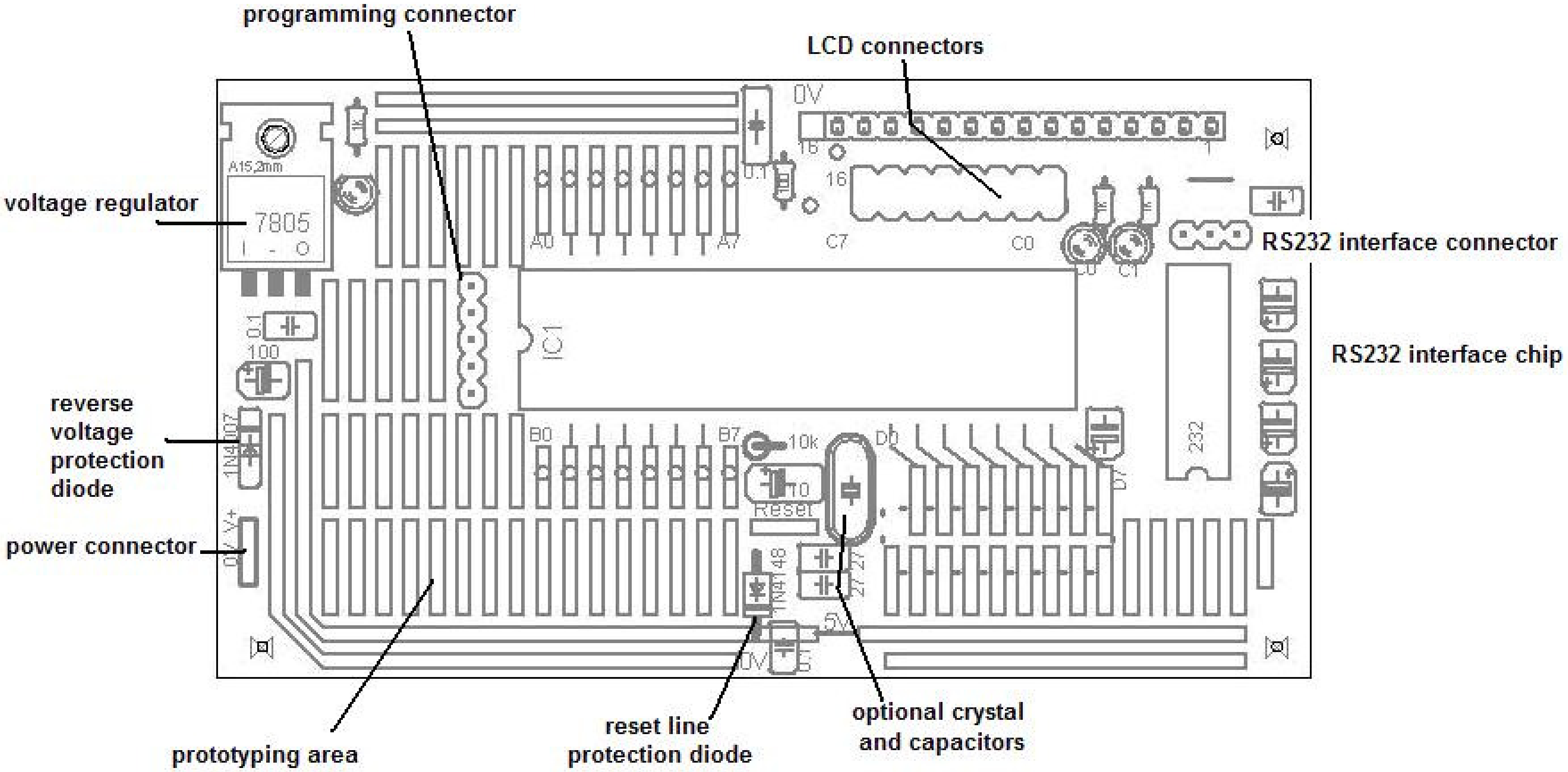

ATMEL AVR
8535
Development
Board
BC2002
12V
5V
0V
100u 63V
0.1
C2
21
20
40
1
AT90S8535P
+Vin
0V
5V
1k
100
10
0.1
12V
5V
0V
16
1k
27p
10u
5
A.0
A.1
A.2
A.3
A.4
A.5
A.6
A.7
+12V

AVR Development Board 2

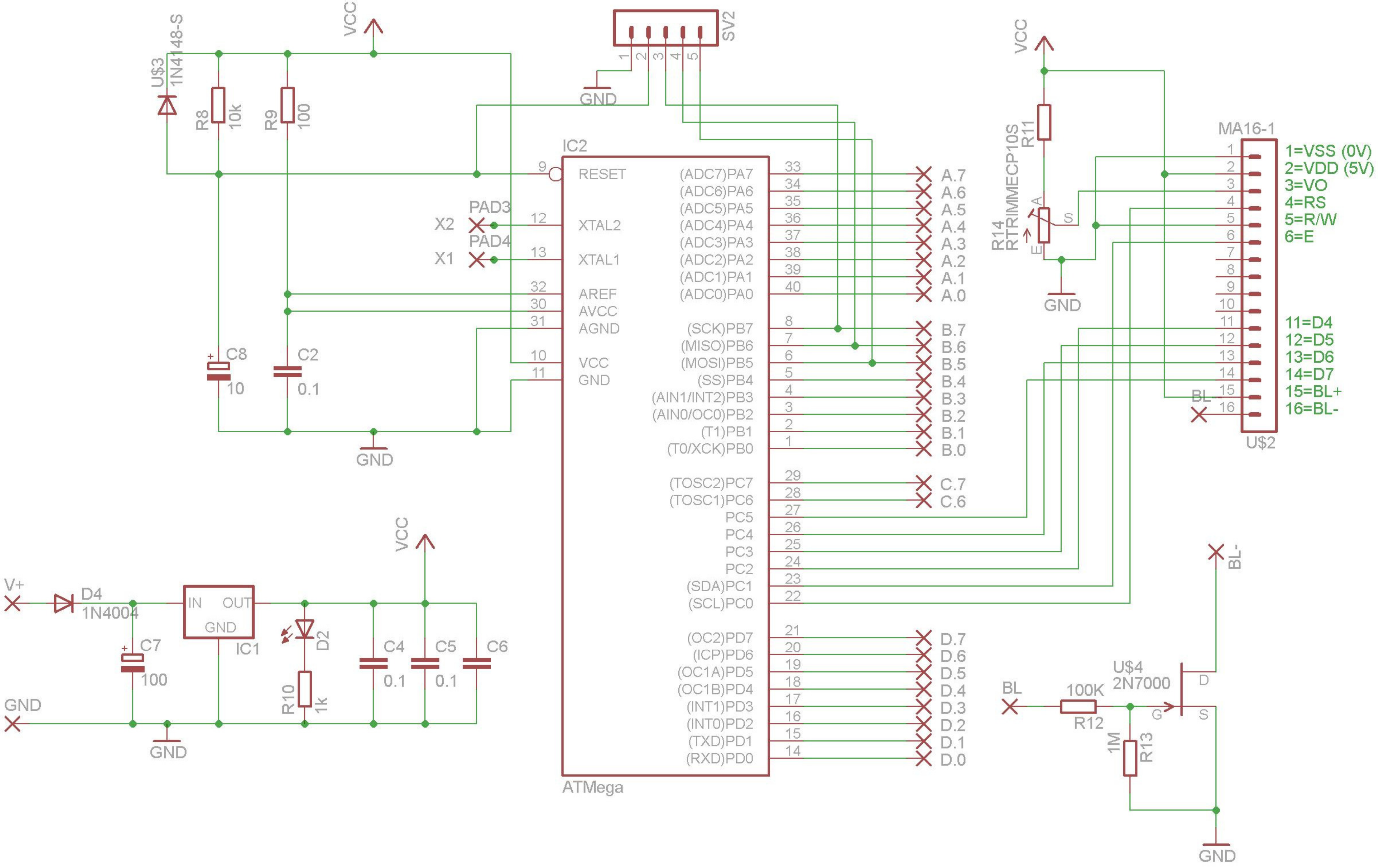


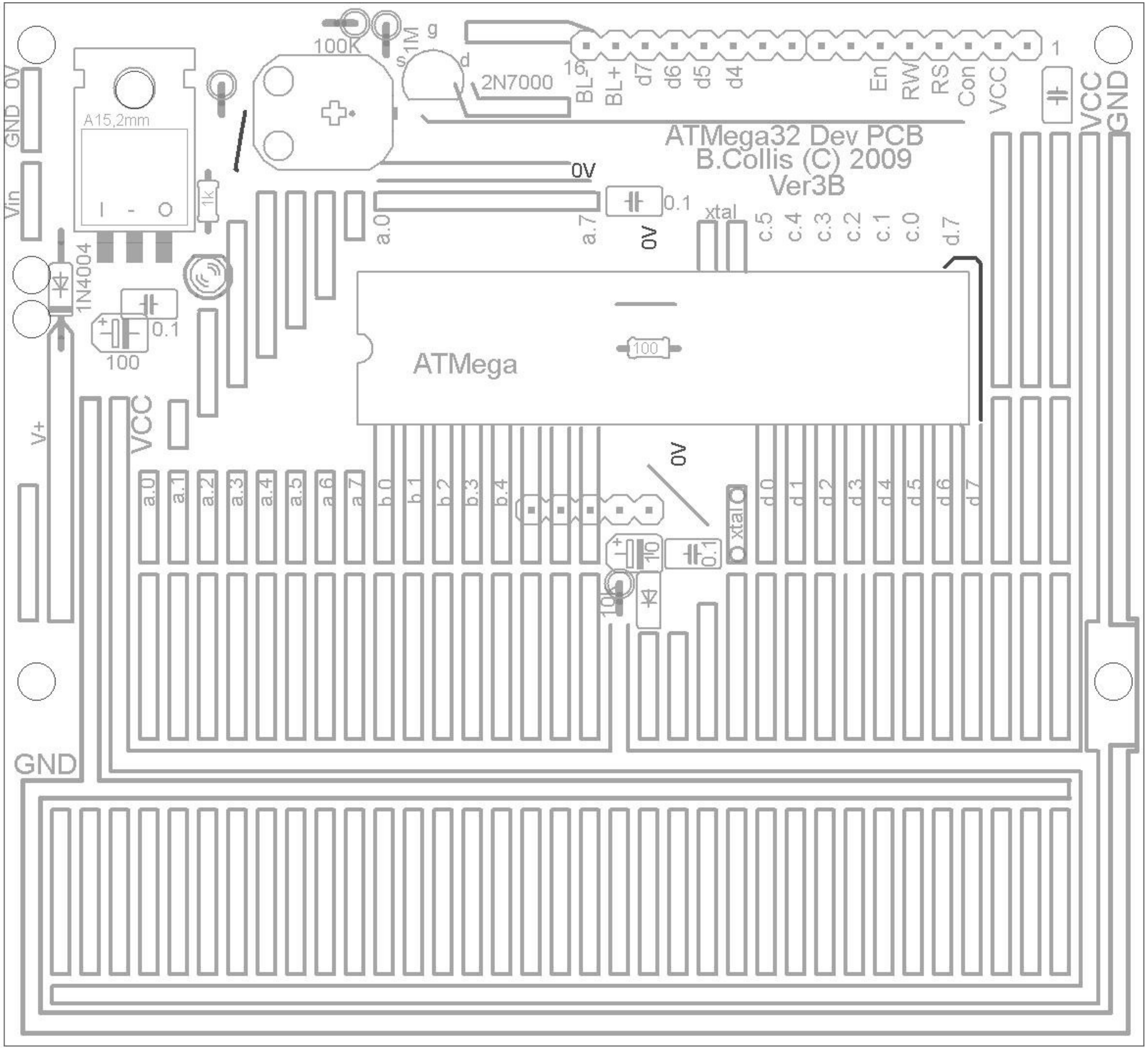


VCC
D6
1N4148
R6
10k
R7
100
SV2
GND
IC1
RESET
XTAL2
XTAL1
AREF
AVCC
AGND
VCC
GND
Q2
C9
10
C13
0.1
C2
27
C10
27
GND
(ADC7)PA7
(ADC6)PA6
(ADC5)PA5
(ADC4)PA4
(ADC3)PA3
(ADC2)PA2
(ADC1)PA1
(ADC0)PA0
(SCK)PB7
(MISO)PB6
(MOSI)PB5
(SS)PB4
(AIN1/INT2)PB3
(AIN0/OC0)PB2
(T1)PB1
(T0/XCK)PB0
(TOSC2)PC7
(TOSC1)PC6
PC5
PC4
PC3
PC2
(SDA)PC1
(SCL)PC0
(OC2)PD7
(ICP)PD6
(OC1A)PD5
(OC1B)PD4
(INT1)PD3
(INT0)PD2
(TXD)PD1
(RXD)PD0
MEGA8535-P
PAD8
PAD13
PAD18
PAD22
PAD24
PAD26
PAD27
SV1
SV3
VCC
GND
R4
R5
1k
1k
D4
D5
V+
0V
D1
1N4007
C1
100
IN
OUT
GND
IC2
7805
D3
R1
1k
C3
0.1
C16
0.1
C17
.1
IC3
C1+
C1-
C2+
C2-
V+
V-
T1IN
T2IN
R1OUT
R2OUT
T1OUT
T2OUT
R1IN
R2IN
232
C4
C5
C6
C7
C8
SV4
GND


programming connector
LCD connectors
voltage regulator
RS232 interface connector
RS232 interface chip
reverse voltage protection diode
power connector
prototyping area
reset line protection diode
optional crystal and capacitors
7805
IC1
0V
16
1
A0
A7
B0
B7
C7
C0
D0
D7
10k
Reset
1N4007
1N4148
232
0.1
100
27
5V
0V V+

ATMEGA Development Board 3





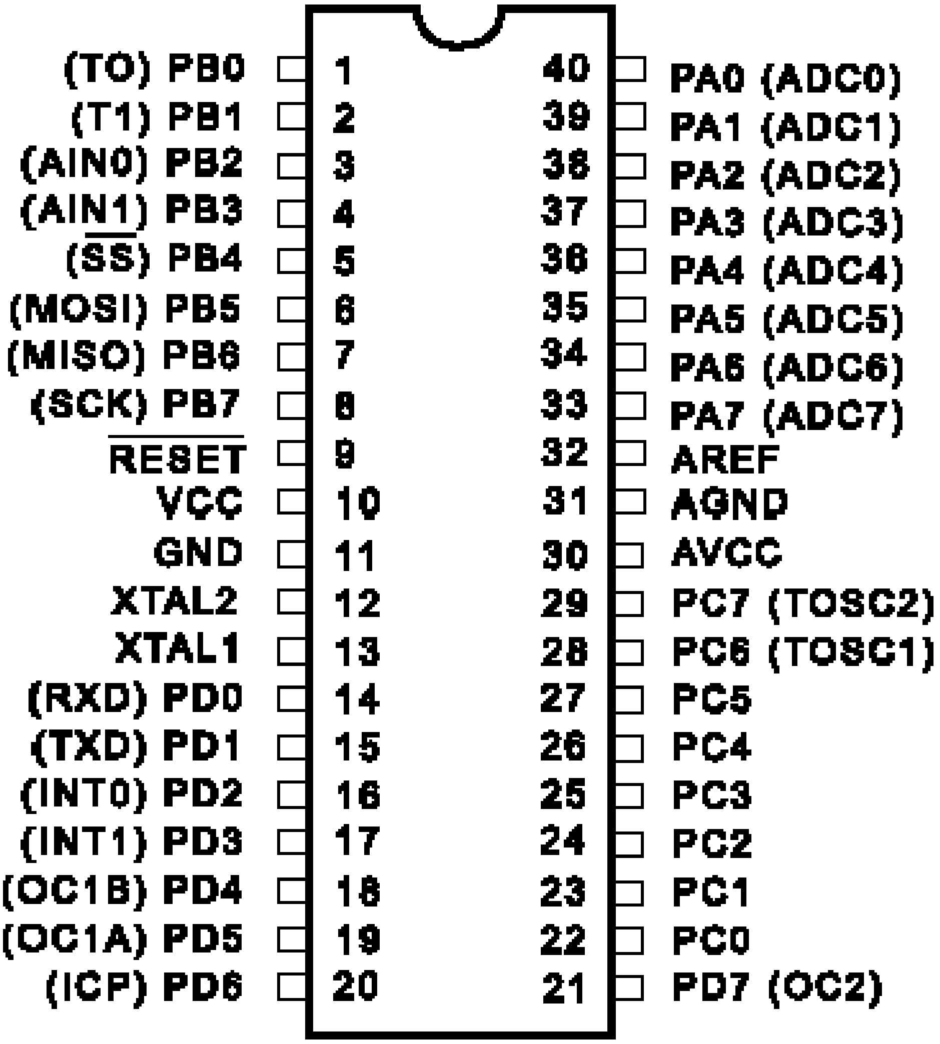
ATMega32 Dev PCB
B.Collis (C) 2009
Ver3B
ATMega
100K
1M
g
s
d
2N7000
16
BL-
BL+
d7
d6
d5
d4
En
RW
RS
Con
VCC
1
VCC
GND
0V
GND
A15,2mm
Vin
1k
1N4004
0.1
100
0V
0.1
xtal
a.0
a.7
0V
c.5
c.4
c.3
c.2
c.1
c.0
d.7
100
V+
VCC
0V
a.0
a.1
a.2
a.3
a.4
a.5
a.6
a.7
b.0
b.1
b.2
b.3
b.4
xtal
d.0
d.1
d.2
d.3
d.4
d.5
d.6
d.7
10
0.1
10k
GND

ATMEGA16/32 Microcontroller Pin Functions and Connections

Although each port of the large development board is connected to an LED, many of them have alternative functions and they have other devices connected to them

Port Pin	Second Function	Direction	Connected to	To control/sense
A.0	ADC 0	In		
A.1	ADC 1	I / O		
A.2	ADC 2	I / O		
A.3	ADC 3	I / O		
A.4	ADC 4	I / O		
A.5	ADC 5	I / O		
A.6	ADC 6	I / O		
A.7	ADC 7	I / O		
B.0	Timer0	Input		
B.1	Timer1	Input		
B.2		I / O		
B.3		I / O		
B.4		I / O		
B.5	MOSI-Prog	I / O		
B.6	MISO-Prog	I / O		
B.7	SCK-Prog	I / O		
C.0		I / O		
C.1		I / O		
C.2		Output		
C.3		Output		
C.4		Output		
C.5		Output		
C.6	xtal	Output		
C.7	xtal	Output		
D.0		I / O		
D.1		I / O		
D.2	Int0	Input		
D.3	Int1	Input		
D4		I / O		
D.5		I / O		
D.6	ICP	Input		
D.7		I / O		

ATMEGA16/32 40pin DIP package– Pin Connections


(TO) PB0
1
(T1) PB1
2
(AIN0) PB2
3
(AIN1) PB3
4
(SS) PB4
5
(MOSI) PB5
6
(MISO) PB6
7
(SCK) PB7
8
RESET
9
VCC
10
GND
11
XTAL2
12
XTAL1
13
(RXD) PD0
14
(TXD) PD1
15
(INT0) PD2
16
(INT1) PD3
17
(OC1B) PD4
18
(OC1A) PD5
19
(ICP) PD6
20
40
PA0 (ADC0)
39
PA1 (ADC1)
38
PA2 (ADC2)
37
PA3 (ADC3)
36
PA4 (ADC4)
35
PA5 (ADC5)
34
PA6 (ADC6)
33
PA7 (ADC7)
32
AREF
31
AGND
30
AVCC
29
PC7 (TOSC2)
28
PC6 (TOSC1)
27
PC5
26
PC4
25
PC3
24
PC2
23
PC1
22
PC0
21
PD7 (OC2)