WJEC CBAC

Learning Objectives:

At the end of this topic you will be able to;

- ☑ recall the architecture of a PIC microcontroller, consisting of CPU, clock, data memory, program memory and input/output ports, connected by buses;
- ☑ explain, and give examples of, the use of interrupts to allow an external device to be serviced on request;
- ☑ draw a circuit diagram to show how an external device can be connected to a PIC microcontroller to cause an interrupt;
- ☑ use a vector address to point to an interrupt service routine;
- ☑ write and analyse given code to configure the ports as input or output ports, using the file registers called TRISA and TRISB;
- ☑ configure the INTCON file register to enable an external interrupt;
- ☑ recognise the need to protect the contents of the working register when writing an interrupt service routine;
- ☑ devise and analyse code using the following instructions:
  bcf, bsf, btfss, call, clrf, goto, movf, movlw, movwf, retfie;
- ☑ incorporate given subroutines into program code.

The requirements of the syllabus for this topic are precise and concise. However, to improve understanding of programming techniques and of how the PIC chip responds to them, the notes on this topic are extensive, and go beyond what is examinable.

To aid revision, areas that are examinable are identified by a vertical line in the left margin. Where there is no vertical line, the notes are aimed only at improving understanding.

Review of ET3

We met the PIC microcontroller in the AS coursework module ET3, as an example of a control system that uses a low-level programming language. Here is a reminder of some of that work, with additional background information.

Programs and instructions

A microprocessor processes digital data, following a sequence of instructions given in a program. The program is an instruction/data sandwich. The basic structure is:

Instruction – do 'this' with the following data
Data
Instruction – now do 'this' with the next item of data
Data
and so on...

The microprocessor is designed to process data in the form of binary numbers. It 'understands' a limited number of instructions, which it recognises because each is given a unique binary number.

The same idea can be used in some restaurants to order a meal:
"I'll have a number 45, followed by a number 19 with a number 68 to finish, please." This works very efficiently, because the chef has a recipe to follow for each of the numbered items on the menu.

In the same, way, the programmer tells the microprocessor to carry out a numbered instruction, such as 101001_2 . The built-in instruction decoder lists the tasks to carry out to complete this instruction. The collection of commands which can be used with a microprocessor is known as the instruction set, and it differs from one type of microprocessor to another, (in the same way as the numbering of items on a menu differs from restaurant to restaurant.)

In reality, programmers rarely write the program as a list of binary numbered instructions and data. Instead they use a mnemonic for each instruction – a word or abbreviation that suggests what the instruction does, for example goto, clrf (**cl**ea**r** **f**ile) etc.

Programming languages

In the world of microprocessors, there are a number of programming languages. Like all languages, these consist of a vocabulary (the mnemonics) and syntax (the 'grammar' used to link the mnemonics and data together to build a program of instructions.) A program will work only if the vocabulary and syntax are totally error-free.

A complication – PIC microcontrollers are an extensive and important group of microcontrollers, present in a wide range of devices from DVD players to engine management systems. There are a number of 'dialects' of their programming language. In particular, MPASM and TASM are common versions used to write programs for PICs. The instruction set is the same, but the syntax has minor differences, (rather like the versions of the English language spoken by Americans and by British.) For the module test, you need to be able to interpret code given in MPASM. In questions where you write the code yourself, it does not matter whether you use MPASM or TASM.

Types of memory

The program and the data are stored in electronic memory. Items like the instruction decoder are stored during manufacture in permanent memory, called ROM (read-only memory.) Later, the user stores his/her program in volatile memory, called RAM (read-and-write memory). The contents of volatile memory are lost when the power is turned off. There is a third type of memory, mid-way between ROM and RAM. With this type, the user can write information into it, but the contents are retained even when the power is turned off. There are a number of ways to implement this. The PIC chips which we use have this type of memory, known as EEPROM (electrically erasable programmable read-only memory.)

Ports

Microprocessors control real world devices such as motors, heaters and lights. They monitor external conditions like temperature, speed and light intensity. They must have the means to input data from, and output data to, the outside world. This is done through electronic sub-systems called ports.

A subsystem which imports data from the outside world to the microprocessor is called an input port. An output port sends data from the microprocessor to the outside world.

Microcontrollers vs microprocessors

A microprocessor system usually consists of at least three ICs, linked by a number of buses, (bundles of wires, usually copper tracks on a printed circuit board.)

IC number 1: At its heart is the **CPU** (central processing unit) which does the data processing. This IC also contains a number of registers, to store data temporarily. (A register is just an electronic number store. An 8-bit register stores an eight bit number, and so on...) The most important of these, by far, is the **Working register** (known as 'W'). Numbers loaded into the working register can be 'operated on' by arithmetic and logic functions. The CPU IC also contains the **Program Counter**. This keeps track of where the processor is, within a program. The program instructions are stored in electronic memory. The Program Counter stores the memory address of the next instruction. When the processor reads an instruction, the Program Counter increments, and points to the next instruction. The CPU IC also contains an Instruction Decoder, an ALU (arithmetic and logic unit), and other subsystems, including some ROM.

IC number 2: It communicates to the outside world via a second IC which contains a number of **ports**.

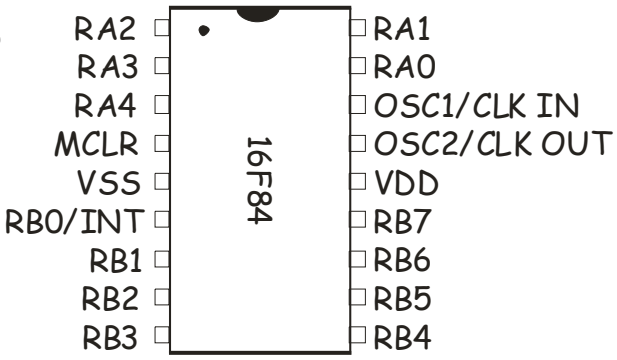
IC number 3: The user program and associated data is stored in a third (and possibly fourth or more) **memory** IC.

The operations of the CPU are timed and synchronised across these ICs by an astable circuit called a **clock**. This may be found in a separate IC, or built into the CPU.

A microcontroller requires these same components, CPU, clock, memory and ports, but they are all housed in a single IC. It is not a replacement for a microprocessor system; it just does a different job. It has less memory than most microprocessor systems, and the processor usually runs more slowly. However, as a single chip solution, it is usually smaller, cheaper and more compact.

The PIC 16F84 microcontroller

This is one of the 18 pin PIC microcontroller range. The pin out is shown opposite:



There are two ports.

- Port A has five bits (RA0 to RA4);
- Port B has eight bits (RB0 to RB7.)

The remaining five bits are:

- V_{SS} and V_{DD}, the power supply connections.
  (The IC runs on a power supply voltage between 4V and 5.5V.)
- MCLR is the master reset pin. It is active low, meaning that to reset the microcontroller, it is necessary to pull this pin down to 0V. Usually, this pin is connected to the positive supply rail, V_{DD} through a resistor.
- OSC1/CLK IN and OSC2/CLK OUT are used to set up one of four oscillator modes to provide clock pulses for the microcontroller.

Pin 6 will be of particular interest to us later. As well as serving as the least significant bit of PORT B, it can also be used to trigger an external hardware interrupt. We will look at this in more detail later in these notes.

Memory Organisation

In the PIC microcontroller, program instructions and data are stored in separate areas of memory.

Program instructions

- These are stored in **F**lash memory (in the PIC16**F**84). This allows the chip to be re-programmed, (given a different set of instructions to carry out,) and means that the program is not lost when power is removed from the PIC chip. Flash is a form of EEPROM.

- The 16F84 Flash memory has 1024 locations, addressed as 000h to 3FFh. (The 'h' indicates that the number is hexadecimal, or base 16. In decimal, 3FFh = $(3 \times 16^2) + (15 \times 16^1) + (15 \times 16^0) = 768 + 180 + 15 = 1023$.So, in decimal, the memory locations start at location number 0 and run up to location number 1023, giving 1024 locations.)

- Location 000h is reserved for the Reset vector address. A vector address is one that contains a pointer to another address. Whenever a reset happens, on power-up, for example, the processor automatically reads the instruction stored in the Reset vector address location. Usually, this instruction sends the processor to the start address of the program.

- Location 004h is reserved for the interrupt vector address. Whenever an interrupt happens, the processor automatically reads the instruction stored in this location. Usually, this instruction sends the processor to the start address of the interrupt service routine, a separate little program which reacts to whatever caused the interrupt. (We will look at interrupts in more detail later.)

Data

- This is stored in RAM or in EEPROM memory.
  - The EEPROM area is used to store 'permanent' data, and is not readily accessible;
  - The data stored in RAM is lost whenever power is removed from the PIC chip. It is split into two sections, Special Function Registers (SFRs) and General Purpose Registers (GPRs). In the PIC 16F84, the SFRs occupy memory locations from 00h to 0Bh. The GPRs start at address 0Ch. This can be seen in the diagram on the next page.
  - In an application such as a combination lock, it would be better to store the correct combination in the EEPROM area, rather than in a GPR so that it is not lost if power is removed.

- The SFRs are further divided into two areas of memory, called 'banks', Bank 0 and Bank1. The registers in Bank1 tend to be those that control the flow of data to and from the corresponding SFR in Bank 0. Although these have different addresses, there is a strong link between them. A register with an address in Bank 0 of 0xxxxxxx, (where 'x' is any binary number, either 0 or 1,) has a corresponding control register in Bank 1 with address 1xxxxxxx. For example, the SFR called PORTA is found in Bank 0 at address 0000 0101_2 (=05h). It is controlled by a register called TRISA, found in Bank 1 at address 1000 0101_2(=85h). Selecting a bank is done by setting (for Bank 1) or clearing (for Bank 0) the RP0 bit of the STATUS register, (which is, in effect, the most significant bit of the register address.) Setting it makes it logic 1, and so addresses Bank1. Clearing RP0 makes the msb logic 0, and so addresses Bank 0. The GPRs are used by the programmer to store information temporarily, such as count totals, lap counts, elapsed time etc.

- The GPRs span both memory banks. A particular GPR, say that at 0Ch, will be accessed regardless of whether Bank 0 or Bank 1 is selected. (It will be accessed using an address of 8Ch as well as 0Ch.)

- Humans work better with names rather than numbers. A GPR can be given a name by using the equate command, e.g.

  total equ 0Ch;

  The effect of this instruction is that whenever the processor encounters the character string 'total', it will replace it with the hex. number 0C.

The structure of the data area of RAM is shown in the table. Don't worry about the names of these registers. We go into more detail for some of them later.

Address	Bank 0	Bank 1	Address
00h	INDF	INDF	80h
01h	TMR0	OPTION	81h
02h	PCL	PCL	82h
03h	STATUS	STATUS	83h
04h	FSR	FSR	84h
05h	PORTA	TRISA	85h
06h	PORTB	TRISB	86h
07h			87h
08h	EEDATA	EECON1	88h
09h	EEADR	EECON2	89h
0Ah	PCLATH	PCLATH	8Ah
0Bh	INTCON	INTCON	8Bh
0Ch	GPR registers 68 bytes		8Ch
to			to
4Fh			CFh

Notice that some SFRs, such as the STATUS register and INTCON register, appear in both banks and can be accessed from either Bank 0 or Bank 1.

'Include' file

We have just seen that an equate statement can be used to create a name for a numbered register. However, we don't do this for Special Function Registers, like PORTA, TRISB and STATUS. The names of these SFRs are equated to the corresponding file register number in the 'include' file for that PIC chip, e.g. P16C84.inc. This file contains the equate statements for all the standard register names, and is included when the program is compiled. Parts of this file are shown below.

```
; P16F84.inc Standard Header File, Version 2.00 Microchip Technology Inc.
; This header file defines configurations, registers, and other useful bits of
;information for the PIC16F84 microcontroller.
;=========================================================
========
;      Register Definitions
;=========================================================
========
                      W  equ 0h
                      F  equ 1h
;----- Register Files------------------------------------------------------
                 STATUS  equ 3h
                  PORTA  equ 5h
                  PORTB  equ 6h
                 INTCON  equ Bh
            OPTION_REG  equ 81h
                  TRISA  equ 85h
                  TRISB  equ 86h
;----- STATUS Bits --------------------------------------------------------
                    RP1  equ 6h
                    RP0  equ 5h
                      Z  equ 2h
;----- INTCON Bits --------------------------------------------------------
                    GIE  equ 7h
                   INTE  equ 4h
                   INTF  equ 1h
;----- OPTION Bits --------------------------------------------------------
                 INTEDG  equ 6h
```

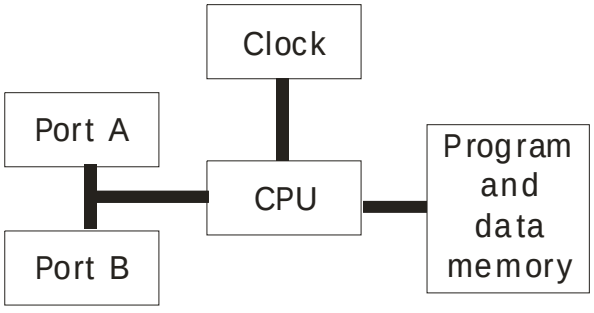
The contents of a typical .inc file

The lines that begin with a ';' are known as **remarks**. The semicolon tells the system to ignore them, as they are not instructions, but merely comments that help humans to keep up with what is going on. Notice that the important lines are the ones that contain the equate (equ) command. These do not start with a semicolon!

Architecture of the PIC microcontroller

Two "normal" forms of architecture are used in modern microprocessors, Von Neumann and Harvard architecture:

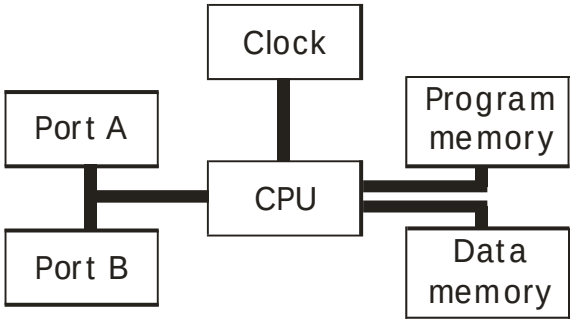
1. Von Neumann architecture –
   - is the more common design, used from the Z80 right through to the Pentium;
   - stores the program instructions and any data being used in the same address space;
   - has three distinct signal pathways, called buses
     - data bus – is used to carry both data and instructions between the processor and peripheral subsystems;
     - address bus - is used to identify which peripheral or storage location is the designated source of, or destination for, the data;
     - control bus - carries a variety of signals that, for example, set the direction of the data transfer, or synchronise it;
   - means that the CPU is either reading an instruction *or* reading/writing data from/to the memory. Both cannot occur at the same time since the instructions and data use the same signal pathways (data bus) and memory space. As a result, some instructions take four or five clock cycles to execute.
   -



Block diagram of Von Neumann architecture

2. Harvard Architecture –

- PIC microcontrollers use the Harvard architecture.
- Program instructions and data are stored separately, and have their own buses;
- The CPU can read both an instruction and data from memory at the same time, as they travel down separate routes;
- As a result, most instructions take only one clock cycle to execute;
- This design offers more efficient instruction queuing.
  Since data and instructions travel down different buses, there is no need for them to be the same bit-width (contain the same number of bits.). For example, often the instruction will be only six bits long, while the data is usually eight bits long.



Block diagram of Harvard architecture

Programming concepts

1. The Instruction Set

The complete instruction set for the PIC16F84 microcontroller comprises 35 instructions. For examination purposes, you should be able to use and interpret the subset of those instructions given in the following table:

Mnemonic	Operands	Description
bcf	f, b	Clear bit b of file f
bsf	f, b	Set bit b of file f
btfss	f, b	Test bit b of file f, skip the next instruction if the bit is set. This is a conditional branch instruction.
call	k	Call subroutine k
clrf	f	Clear file f
goto	k	Unconditional branch to label k
movf	f, d	Move file f (to itself if d = 1, or to working register if d = 0)
movlw	k	Move the literal k to the working register
movwf	f	Move working register to file f
retfie		Return from the interrupt service routine and set the global interrupt enable bit GIE.

Please note:

- An instruction consists of a mnemonic and an operand.

For instructions such as call, goto and movlw the operand will be a label. (A label is simply a marker inside the program.)

For example:

```
goto    restart  ;branch (unconditionally) to the part of the
                 ;program labelled 'restart'.
call    second   ;call the subroutine named 'second'
```

- In other cases, clrf and movwf, the operand will be the name (or number) of a file register, either a SFR or a GPR..

For example:

```
movwf   PORTB    ;move the contents of the Working register
                 ;into the file register called PORTB.
clrf    0Fh      ;write logic 0 into all bits of the file (GPR)
                 ;numbered 0Fh.
```

- Instructions such as, bcf, bsf and btfss, will require the operand to specify both a file register name (or number) and also the specific bit of that register that is affected.
  For example:

```
bsf     PORTA,0     ; set bit 0 (the lsb) of PORTA
```

- In the case of instructions like movf, the operand gives the name (or number) of the file register, and the destination, (working register or file register) for the result of the instruction.
  For example:

```
movf    counter,0   ;move the contents of the file called counter
                    ;into the working register.
```

or alternatively

```
movf    counter,1   ; move the contents of the file called counter
                    ;into itself. Although apparently useless, this
                    ;instruction will test whether the file counter
                    ;contains zero. If it does, this instruction will
                    ;set the Zero flag in the STATUS register.
                    ;Otherwise, the Zero flag will not be set.
```

Exercise 1: (Solutions to exercises are given at the end of the topic.)

Observing correct syntax, write the instructions needed to:

(a) set bit 3 of PORTA;

(b) clear bit 1 of PORTB;

(c) clear the file register called testfile;

(d) move the binary number 11011 into the working register;

(e) move the contents of the working register into a file register called cost;

(f) move the contents of the file register called cost into the working register;

(g) branch unconditionally to a point in the program identified by the label repeat;

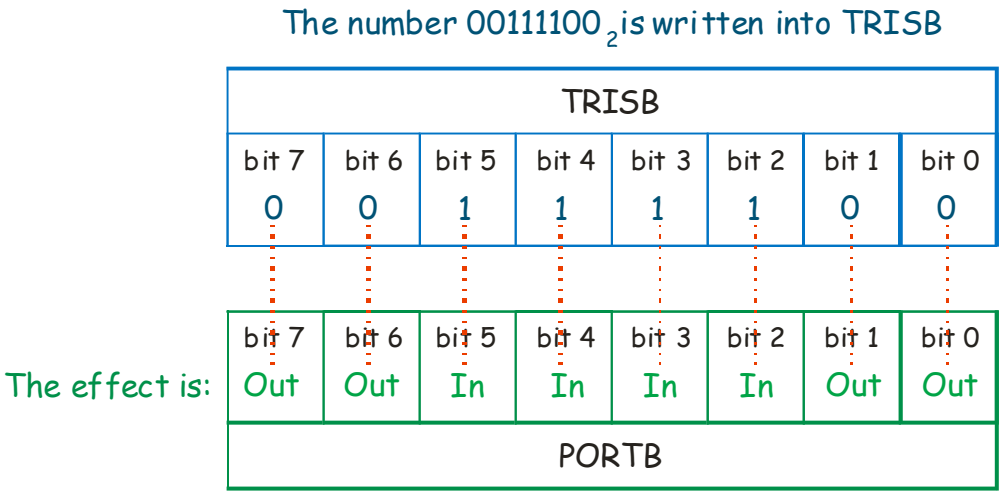
(h) test bit 2 of the file register called input, and skip the next instruction if the bit is set.

2. Configuring the ports:

Data is inputted and outputted through registers, called ports. The number of ports varies from PIC to PIC, but there are usually at least two, called Port A and Port B.

These ports are bi-directional, meaning that they can be set up either to input data or to output it. Control registers, called TRISA, for Port A and TRISB for Port B, determine whether the port is an input or an output, (or a mixture.) Writing logic 1 into a bit of TRISA causes the corresponding bit of Port A to input data. Logic 0 makes the corresponding bit output data. TRISB controls Port B in the same way.

The next diagram illustrates this relationship.



The names TRISA, TRISB, PORTA and PORTB are all defined in the 'include' file referred to earlier. As the diagram on page 8 shows, TRISA and TRISB are found in Memory Bank 1, in locations that sit alongside PORTA and PORTB in Bank 0.

Usually the ports are configured at the beginning of the program by writing the appropriate binary numbers into the registers TRISA and TRISB. As a rule, the ports are then left that way for the rest of the program. In other words, the only time you use TRISA and TRISB is at the beginning of the program. In the PIC 16F84, PORTA (and so TRISA) is a 5-bit port. PORTB (and TRISB) is a 8-bit port.

The registers TRISA and TRISB are located in an area of memory called Bank 1. Normally, programs use the memory located in Bank 0. To move to Bank 1, a bit, called RP0, of the STATUS register (of which more later) is set (has logic 1 written into it.) To move back to Bank 0, that bit is cleared (has logic 0 written into it.)

For example:
To configure bits 0 to 3 of Port B bits as input bits, and bits 4 to 7 as output bits, use the following code:

```
bsf     STATUS,RP0  ;the following instructions will refer to
                    ;Memory Bank1, and so affect the TRIS
                    ;registers rather than the Ports themselves
movlw   b'00001111' ;0 = output bit, 1 = input bit
movwf   TRISB       ;store number 00001111 in TRISB
bcf     STATUS, RP0 ;the rest of the program will refer to
                    ;Memory Bank0, and so operate on the Ports
                    ;themselves rather than the TRIS registers.
```

The STATUS register

This is one of the Special Function Registers, SFRs, discussed on page 67. It contains a number of selection bits, and 'flags'. A flag is a marker to show that some event, or condition, has occurred. For example, the zero flag is used to spot when the processor, in subtracting one number from another, produces a result of zero.

The STATUS register has three main jobs:

- It allows us to select the memory bank used in the instructions that follow using bit 5, the selection bit named as RP0 in the 'include' file on page 9.
- It shows the results of arithmetic and logic operations, using the zero flag, the carry flag and the digit carry flag.
- It identifies the cause of a device reset, on the PD and TO flags..

The structure is shown below

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
IRP	**RP1**	**RP0**	**TO**	**PD**	**Z**	**DC**	**C**

Key: **IRP** - Register Bank Select bit,
(not used in PIC16F84 and so should contain logic 0.)

RP1, RP0 - Memory Bank Select bits. (Each bank contains 128 bytes of memory.)
00 = Bank0 (00h – 7Fh);
01 = Bank1 (80h – FFh);
10 = Bank2 (100h – 17Fh);
11 = Bank3 (180h – 1FFh);
- RP1 is not used in PIC16F84 and should contain logic 0

TO - Time-out flag

PD - Power-down flag

Z - Zero flag (Logic 1 means that the result of the previous arithmetic or logic operation was zero. Logic 0 means that the result was not zero.)

DC - Digit Carry / Borrow bit

C - Carry / Borrow bit

Exercise 2: (Solutions to exercises are given at the end of the topic.)

(a) Complete the following code in order to configure bits 0 to 3 of Port B bits as input bits, and bits 4 to 7 as output bits.

```
……      STATUS,…….  ;the following instructions will refer to
                    ;Memory Bank1, and so affect the TRIS
                    ;registers rather than the Ports themselves
movlw   b'……';0 = output bit, 1 = input bit
……      TRISB       ;move the number from the working register
                    ;to TRISB
……      STATUS, …… ;the rest of the program will refer to
                    ;Memory Bank0, and so operate on the Ports
                    ;themselves rather than the TRIS registers.
```

(b) Write the section of code needed to:

- select Bank 1;
- configure the lsb and the msb of PORTA as input bits, and the other three bits as output bits.
- select Bank 0.

Interrupts

An interrupt is an event that forces the processor to jump from its current activity to a specific point in the program, and then carry out a special program called the Interrupt Service Routine (ISR). Interrupts are designed to be special events whose arrival cannot be predicted precisely.

A non-PIC example: You are holding a party. You are not sure when the guests will arrive. One option is to open the front door every couple of minutes to see if anyone is there. This is very time-consuming, and means that you can not really get on with other jobs. In programming, this approach is called *polling*. Alternatively, you can wait for the doorbell to ring. Then you can stop whatever you are doing to open the front door. This is like using an *interrupt*.

Interrupts allow the microcontroller to respond immediately to an external event, such as a switch being pressed, or a sensor output changing state. These are called hardware interrupts.

For example, an environmental control system for a building may have a number of functions. It could:

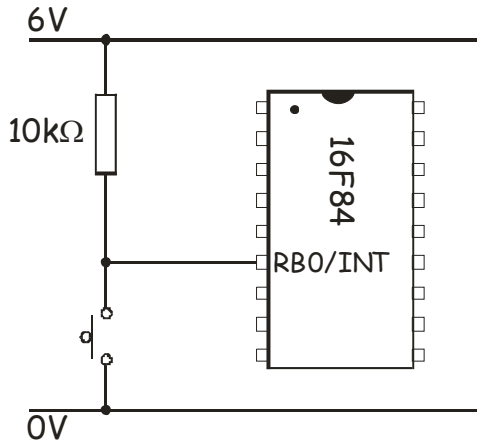
- control air conditioning, heating and ventilation;
- detect fires in the building.

The first function requires that the control program **poll** a number of sensors throughout the building periodically, and switch heating equipment and fans on and off for various time intervals accordingly.

However, if a fire breaks out, an instant response is needed. If the control program polled smoke detectors, then it may take time for the program to get round to that particular sensor. It is much better to connect the smoke sensors to the PIC IC so that they cause hardware **interrupts** when there is a fire.

Hardware Interrupt Circuit

With the PIC 16F84, pin 6, called RB0/INT, must be momentarily pulled down to logic 0 in order to cause a hardware interrupt. The circuit diagram shows how this is achieved, using a switch.



Interrupt Service Routine

When an interrupt is detected, the processor completes the current instruction and then jumps to a section of the program called the **Interrupt Service Routine (ISR)**.

This takes place in stages:

- The processor completes the instruction it is currently executing. Then it stores the contents of the Program Counter on the **stack**. This is a specialized area of memory. In the PIC16F84, it cannot be accessed directly in the way that other memory locations can. Its job is to store return addresses so that the processor knows where to resume the main program after running either an interrupt service routine or a subroutine.
- The interrupt automatically causes the Program Counter to load the address 004h, the **Interrupt Vector Address**. The CPU then executes the instruction found there.
- You can write the ISR directly, starting at memory location 4.
  More commonly, you write a **goto** instruction in memory location 4, redirecting the processor to the location of the ISR.
  Usually you mark the start of the ISR with a label such as **inter**.

- Once the processor has completed the ISR, the return address is copied from the stack into the Program Counter, and the main program continues from where it left off when the interrupt was called.

These features can be seen in the ET3 project program template:

```
;The PIC16C84 vectors live at the bottom of memory (0000h-0007h)

        org     0000h       ;Reset vector for PIC16C84 is at 0000h
        goto    start       ;Go to main program start

        org     0004h       ;Interrupt vector for PIC16C84 is at 0004h
        goto    inter       ;Go to start of ISR

        org     0008h       ;first location available to programs
```

Notice that you **goto** the ISR – you don't **call** the ISR. Using the **call** instruction would store another return address on the stack. After completing the ISR, the processor would then go back to location 0004h.

(Notice also the use of the **Reset Vector Address**, at memory location 0000h, with its **goto** command directing the processor to the start of the main program, identified by the label **start**. In the same way, memory location 0008h is identified as the first location in which user programs should be written.)

Exercise 3: (Solutions to exercises are given at the end of the topic.)

Modify the ET3 project program template so that:

- on power-up it directs the processor to the main program which begins at the label launch.
- in the event of an interrupt, it directs the processor to an interrupt service routine identified by the label problem1.

The INTCON register

There are four types of event that can trigger interrupts

- completion of writing data to the EEPROM memory;
- overflow of the TMR0 timer (changing from FFh to 00h);
- change of state of any of the pins RB4, RB5, RB6 or RB7;
- change of state of pin RB0/INT to logic 0.

The syllabus requires consideration of only the last one, a hardware interrupt triggered by use of the RB0/INT pin on the PIC chip.

Interrupts are controlled by a Special Function Register, SFR, called **INTCON**. It allows the user to make the CPU respond to any/all of the four types of interrupt source, and also indicates which kind of interrupt has taken place. Its structure is shown below.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
GIE	**EEIE**	**TOIE**	**INTE**	**RBIE**	**TOIF**	**INTF**	**RBIF**

Key:

GIE - Global Interrupt Enable bit
- logic 1 enables all interrupt sources, logic 0 disables them.

EEIE - EEPROM Write Complete Interrupt Enable bit
- logic 1 enables, logic 0 disables, it;

TOIE - TMR0 Overflow Interrupt Enable bit
- logic 1 enables, logic 0 disables, it;

INTE - External Interrupt Enable bit
- enables interrupt on RB0/INT pin
- logic 1 enables, logic 0 disables, it;

RBIE - RB Port Change Interrupt Enable bit
- enables interrupt caused by change of state of pins RB4, RB5, RB6 or RB7
- logic 1 enables, logic 0 disables, it;

TOIF - TMR0 Overflow Interrupt Flag
- logic 1 - TMR0 counter output changed from FFh to 00h;
- logic 0 – no overflow occurred.

INTF - External Interrupt Flag
- logic 1 – RB0/INT pin pulled low, causing interrupt;
- logic 0 – no interrupt occurred.

RBIF - RB Port Change Interrupt Flag
- logic 1 – one of pins RB4, RB5, RB6 or RB7changed state;
- logic 0 – no change of state occurred.

- The Global Interrupt Enable bit (GIE) must be set to make the processor respond to *any* type of interrupt. You can clear this bit if you don't want the processor to be interrupted in a particularly important part of the program.
- Bits 3, 4, 5 and 6 are used to select which type(s) of interrupt are active. Bit 4 is called the **External Interrupt Enable (INTE)** bit. It makes the processor respond to interrupts triggered by the RB0/INT pin, (but only if the GIE bit is also set.)
- The INTCON register also contains three flags, bits 0, 1 and 2, to show the processor which type of interrupt has taken place. They operate even if the interrupts themselves are not enabled.
- On power-up, the INTCON register contains the binary number 0000 000x, where x is unknown. This means that all interrupts are disabled by default.
- When you write the ISR, you must clear bit 1, the **INTF** bit, which flags up that an interrupt happened on the RB0/INT pin, so that the program returns to normal operation. Otherwise the program will constantly go into interrupt routine.
- The GIE bit is cleared automatically when an interrupt occurs. This means that an ISR can't be interrupted. On returning to the main program, the GIE bit must be set once more to make the processor respond to further interrupts. This is done automatically when you use the **retfie** command, (**ret**urn **f**rom **i**nterrupt and **e**nable {GIE}). Using the **return** command does not set the GIE bit.

The following code enables RB0/INT interrupts:

```
movlw   b'10010000'  ;enables GIE and INTE bits only
movwf   INTCON       ;and clears all interrupt flags.
```

As the INTCON register sits in both Bank 0 and Bank 1, this code can be written when the ports are configured (using Bank1) or in the main program (using Bank 0.)

The following code clears the INTF bit:

```
bcf     INTCON,1   ;clears bit 1 (INTF) of the INTCON register.
```

Exercise 4: (Solutions to exercises are given at the end of the topic.)

(a) Write the section of code needed to:

- select Bank 1;
- configure bits 0,1 and 2 of PORTA, and bits 0, 1, 2, 3 and 4 of PORTB as input bits, and all other bits as output bits.
- enable RB0/INT interrupts;
- select Bank 0.

(b) Write an Interrupt Service routine, identified by the label inter, that:

- clears the INTF bit;
- lights three LEDs connected to bits 5, 6 and 7 of PORTB, by outputting logic 1 to these bits;
- calls the delay subroutine called fivesec;
- switches off the three LEDs connected to bits 5, 6 and 7 of PORTB;
- returns to the main program and reset the Global Interrupt Enable bit at the same time.

Protecting registers during interrupts

When an interrupt happens, the contents of the Program Counter are automatically stored (on the Stack). Once the ISR is completed, the processor retrieves the Program Counter contents from the Stack, and so can continue with the main program from where it left off.

No other registers are protected in this way. This will cause problems when the contents of a register are changed during the execution of the ISR. When the ISR is completed, and the processor returns to the main program, the changed contents of the register may cause undesirable effects.

Particularly vulnerable in this respect are the STATUS register and the Working register. It is good practice to save the contents of both these registers at the beginning of the ISR, and restore them at the end.

The syllabus requires that candidates can protect the **Working** register. (The principle is exactly the same for protecting the STATUS register.)

To begin with, in the **Register Usage** section of the program, define the file register which will be used to store the contents of the Working register. This is done by using an **equate** statement. This tells the processor to convert the label on the left of the equ instruction to the number on the right.

The code in this case is:

```
W_temp   equ      10h    ;the temporary storage place is now called
                         ;W_temp, and is file register number 10h
```

At the beginning of the ISR, which is called inter in the following code, store the contents of the Working register in W_temp:

```
inter    movwf   W_temp
```

At the end of the ISR, before returning to the main program, restore the original contents of the Working register:

```
         movf    W_temp,0     ;move the file W_temp into the Working
                               ;register (shown by the ' ,0 ')
         retfie               ;return to the main program
```

Exercise 5: (Solutions to exercises are given at the end of the topic.)

(a) Write a statement associate the name Workstore with the GPR at address 0Fh.

(b) Write an Interrupt Service routine that:

- is identified by the label panic;
- protects the Working register by transferring its contents to the file register Workstore at the beginning of the ISR;
- clears the INTF bit;
- outputs logic 1 to four sirens connected to bits 4, 5, 6 and 7 of PORTB;
- calls the subroutine called thirtysec;
- recovers the contents of the Working register;
- returns to the main program and reset the Global Interrupt Enable bit at the same time.

The OPTION Register:

Another feature of this interrupt source is that it can be set to occur either on the rising or falling edge of the signal pulse from the RB0/INT pin. This makes use bit 6 of another SFR, the **OPTION** register. Its structure is shown below:

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
RBPU	**INTEDG**	**T0CS**	**T0SE**	**PSA**	**PS2**	**PS1**	**PS0**

Key:

RBPU - PortB weak Pull-up Enable bit
INTEDG - Interrupt Edge Select bit
- logic 1 - interrupt occurs on the rising edge of the RB0/INT pulse;
- logic 0 - interrupt occurs on the falling edge of the RB0/INT pulse;

T0CS - TMR0 Clock Source Select bit
T0SE - TMR0 Source Edge Select bit
PSA - Pre-scaler Assignment bit
PS2, 1, 0 - Prescaler Rate Select bits

- On power-up, the OPTION register will contain the binary number 1111 1111. This means interrupts occur on the rising edge of the RB0/INT pulse by default.
- In examination questions, candidates will not have to manipulate the contents of the OPTION register.

For further information:

www.epemag.wimborne.co.uk
– for occasional tutorials about PIC

www.mikroe.com/en/books/picbook/picbook.htm
- for a free online PIC course that goes way beyond the requirements of the syllabus

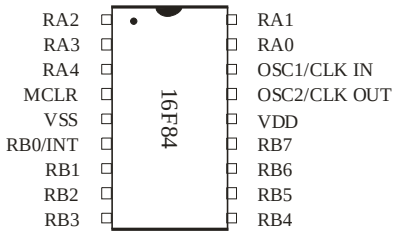
www.mstracey.btinternet.co.uk/index.htm
- see tutorials 11 and 12

www.microchip.com
- datasheets and full technical information on PIC chips.

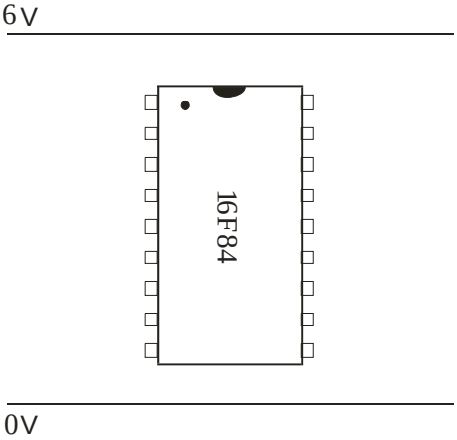
Module ET5
Electronic Systems Applications.

Practice Exam Questions:

1. *(a)* Here is the pin out for a PIC 16F84 microcontroller.



Draw the circuit diagram for a push switch, and any other components needed, connected to the PIC chip so that, when the switch is pressed, the PIC microcontroller jumps to the interrupt service routine, using the RB0/INT pin, which is active low. [1]



(b) The INTCON register contains the enable bits for all interrupt sources.
Complete the instructions below, to configure the INTCON register so that pressing the switch triggers an interrupt, and so that all other interrupt sources are disabled. [1]

```
        movlw       .............
        .............  .............
```

(c) The interrupt service routine is given below:

```
inter   movlw       b'11110000'
        movwf       PORTB
        call        tensec      ; call the ten second delay subroutine.
        retfie
```

The interrupt vector address is 04. Write the instruction that must be included at that address to jump to the interrupt service routine. [2]

```
04      .........   .........
```

(d) Complete the instructions to configure all bits of PORTA, and the four lowest significant pins of PORT B, as inputs and the four highest significant pins of PORTB as outputs, [3]

```
bsf     STATUS,RP0
movlw   b'.....'
movwf   .....
movlw   b'.....'
movwf   .....
bcf     STATUS,RP0
```

(e) A PIC microcontroller is used to control the operation of a lift in a multi-storey car park. If a passenger in the lift presses the Alarm switch, the interrupt service routine warns the lift operator by flashing an LED on the control panel and sounding a buzzer. The LED flashes on and off at a frequency of 2Hz until the operator presses a reset switch.

(i) Explain why it is better to have the Alarm switch cause an interrupt rather than include a section of code in the main program to poll the switch to see whether it has been pressed or not? [1]

……

……

(ii) The Interrupt Service Routine:

- is labelled inter;
- protects the contents of the working register, using a file register called workstore;
- makes use of a 0.25 second time delay subroutine, called quart, which is already written;
- controls an LED attached to bit 0 of PORT B;
- controls a buzzer attached to bit 1 of PORTB;
- monitors the Alarm switch attached to PORTA bit 0.

Write the code for the interrupt service routine. [5]
You should make use of commands only from the following list. You will not need to use all of these commands.

bcf bsf btfss call clrf goto movf movlw movwf retfie

```
inter   .....       .....
loop    .....       .....
        .....       .....
        .....       .....
        .....       .....
        .....       .....
        .....       .....
        .....       .....
        .....       .....
```

.....

..

Solutions to Exercises:

Exercise 1:

(a) set bit 3 of PORTA → bsf PORTA,3

(b) clear bit 1 of PORTB → bcf PORTB,1

(c) clear the file register called testfile → clrf testfile

(d) move the binary number 11011 into the working register
→ movlw b'11011'

(e) move the contents of the working register into a file register called cost
→ movwf cost

(f) move the contents of the file register called cost into the working register → movf cost,0

(g) branch unconditionally to a point in the program identified by the label repeat → goto repeat

(h) test bit 2 of the file register called input, and skip the next instruction if the bit is set → btfss input,2

Exercise 2:

(a)
```
bsf     STATUS,RP0  ;the following instructions will refer to
                    ;Memory Bank1, and so affect the TRIS
                    ;registers rather than the Ports themselves
movlw   b'00001111' ;0 = output bit, 1 = input bit
movwf   TRISB       ;move the number from the working register
                    ;to TRISB
bcf     STATUS, RP0 ;the rest of the program will refer to
                    ;Memory Bank0, and so operate on the Ports
                    ;themselves rather than the TRIS registers.
```

(b)
```
bsf     STATUS,RP0  ; select Bank 1;
movlw   b'10001'    ; configure the lsb and the msb of PORTA as
movwf   TRISA       ; input bits, and the other three bits as
                    ; output bits
bcf     STATUS, RP0 ; select Bank 0
```

Exercise 3:

```
org     0000h   ; Reset vector for PIC16C84 is at 0000h
goto    launch  ; On power-up it directs the processor to the
                ; main program which begins at the label launch.
org     0004h   ; Interrupt vector for PIC16C84 is at 0004h
goto    problem1; In the event of an interrupt, it directs the
                ; processor to an interrupt service routine
                ; identified by the label problem1
org     0008h   ;first location available to programs
```

Exercise 4:

(a)
```
bsf     STATUS,RP0  ; select Bank 1
movlw   b'00111'    ; configure bits 0,1 and 2 of PORTA as input
                    ; bits, and all other bits as output bits.
movwf   TRISA
movlw   b'00011111'  ; configure bits 0, 1, 2, 3 and 4 of PORTB as
                    ; input bits, and all other bits as output bits.
movwf   TRISB
bcf     STATUS, RP0 ;select Bank 0
```

(b)

```
inter   bcf     INTCON,1    ; clears the INTF bit.
        movlw   b'11100000' ; lights three LEDs connected to
                            ; bits 5, 6 and 7 of PORTB,
        movwf   PORTB       ; by outputting logic 1 to these bits.
        call    fivesec     ; calls the delay subroutine called fivesec.
        clrf    PORTB       ; switches off the three LEDs connected to
                            ; bits 5, 6 and 7 of PORTB.
        retfie              ; returns to the main program and resets the
                            ; Global Interrupt Enable bit at the same time.
```

Exercise 5:

```
(a) Workstore   equ   0Fh   ; associates the name Workstore with the
                            ; GPR at address 0Fh.
```

(b)

```
panic   movwf   Workstore   ; protects the Working register by
                            ; transferring its contents to file Workstore
        bcf     INTCON,1    ; clears the INTF bit.
        movlw   b'11110000' ; outputs logic 1 to four sirens connected to
        movwf   PORTB       ; bits 4, 5, 6 and 7 of PORTB,.
        call    thirtysec   ; calls the subroutine called thirtysec.
        movf    Workstore,0; recovers the contents of Working register
        retfie              ; returns to the main program and resets the
                            ; Global Interrupt Enable bit at the same time.
```