Lesson 14

Title of the Experiment: Introduction to Microcontroller
(Activity number of the GCE Advanced Level practical Guide – 27)

Name and affiliation of the author:
N W K Jayatissa
Department of Physics, University of Kelaniya

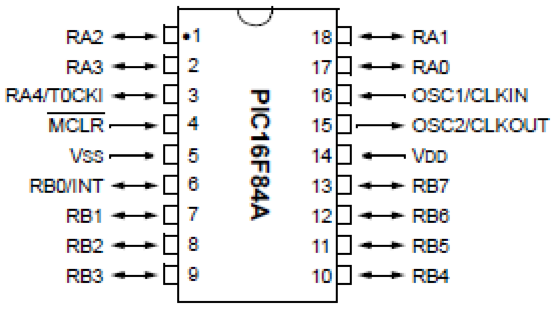
Introduction

A computer consists of a Central Processing Unit (CPU), Read Only Memory (ROM), Random Access Memory (RAM) and Input/Output (IO) ports. The CPU takes instructions from the ROM and executes them. Temporary data is stored in the RAM, and IO ports are used for external communication. A microcontroller is different to a microprocessor because it has all these things on a single chip. There are many sorts of microcontrollers available, but the one used in this syllabus is Peripheral Interface Controller (PIC) chips, manufactured by Microchip. The chip requires only an external clock source, generated from a crystal, to operate. PICs generally have a few Kb of ROM, 256 or less bytes of RAM, 256 bytes of EEPROM and several analogue and digital IO lines. The program you write is stored in flash ROM. Hence, it can be erased and reprogrammed many times. The PIC was the first widely available device to use flash memory, which makes it ideal for experimental work.

The PIC computer follows the Harvard architecture. This means that instructions are held in different memory to data, unlike on a normal PC where instructions and data share the same memory. (This is Von Neumann architecture.) In the PIC, the instruction memory is 14 bits wide, and the data memory is 8 bits.

There are several ways of programming the PIC - using BASIC, C, or Assembly Language. This lesson is focused on teaching how to program a PIC using Assembly Language.

Pin configuration of PIC 16F84 Microcontroller



(Microchip, 2008)

Figure 1: Pin configuration of PIC16F84 microcontroller

PIC 16F84 has two set of bidirectional ports from RA0 to RA3 and RB0 to RB7 (Figure 1). User can configure pins of these ports as inputs (*I*) or outputs (*O*) according to their requirement. Vss and V_{DD} represent the negative and positive power supply pins respectively. Generally Vss is connected to the ground (0V) and V_{DD} is in between 2V (minimum) and 6V (maximum). Pin number 15 and 16 are used for connecting timing devices to the microcontroller. Pin number 4 ($\overline{MCLR}$) is used to clear the memory inside the chip. It is required to reprogram the chip after clearing the memory. Therefore during the operation pin number 4 is connected to the V_{DD} (or positive logic) to prevent clearing the memory. Pin number 6 (RB0/INT) can be used as I/O pin or an Interrupt pin. A status change in outside process could be sensed by this pin and make some operation as required. For example, when the temperature reaches a set value, switch off the heating system and switch on the indicator lamp. Pin 3 (RA4/TOCK1) can be used as I/O pin or another timer input.

It is a good habit to place a PIC chip on an IC socket as shown in figure 2, instead of soldering.

Figure 2: Microcontroller is fitted into a IC base

Block diagram of PIC16F84A

Central processing unit (CPU) is the brain of a microcontroller. That part is responsible for finding and fetching the right instruction which needs to be executed, for decoding that instruction, and finally for its execution. Arithmetic logic unit is responsible for performing operations of adding, subtracting, moving (left or right within a register) and logic operations. Moving data inside a register is also known as 'shifting'. PIC16F84 contains an 8-bit arithmetic logic unit and 8-bit work registers. Figure 3 illustrate the block diagram of PIC16F84A controller. In this figure ALU block and CPU block are illustrated using dotted line and dashed line respectively.

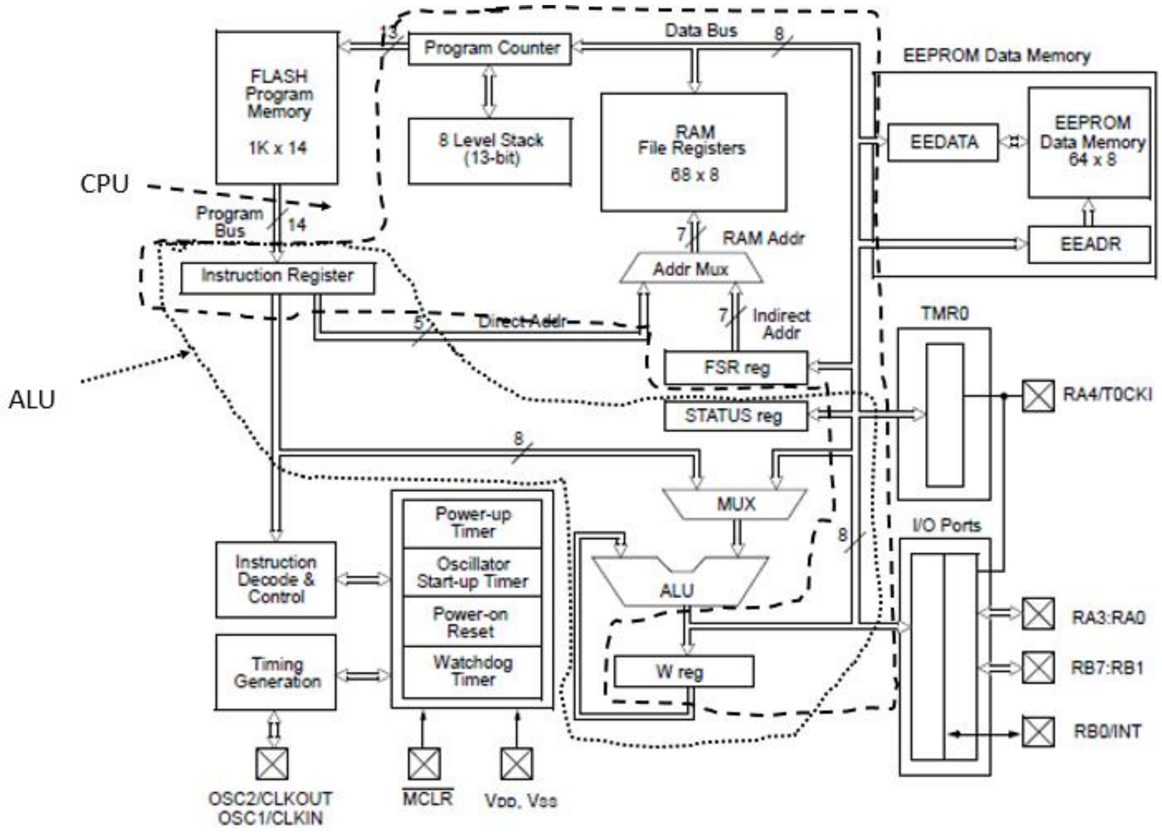


Figure 3: Block diagram of PIC16F84A

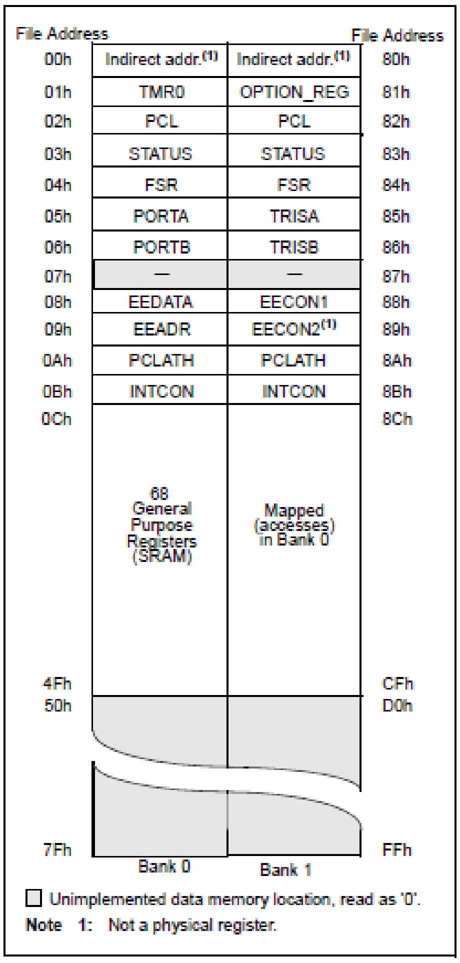
Before discussing further it is good to know a few important things that involve in any good program.

The compiler will ignore anything after the semicolon (;) until the carriage returns. This helps to add comments in the program. Assign meaningful names to constants and variables such as COUNT, TIME etc. Finally it is good to place a heading and program description before the start of the program with the help of semicolons as follows.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Author :                                    ;
; Date :                                      ;
; Title:                                      ;
; Description:                                ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

Registers

A register is a place inside the PIC that a programmer can write to or read from. The register file map is divided into two parts known as BANK0 and BANK1 (Figure 4).



File Address	Bank 0	Bank 1	File Address
00h	Indirect addr.(1)	Indirect addr.(1)	80h
01h	TMR0	OPTION_REG	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(1)	89h
0Ah	PCLATH	PCLATH	8Ah
0Bh	INTCON	INTCON	8Bh
0Ch–4Fh	68 General Purpose Registers (SRAM)	Mapped (accesses) in Bank 0	8Ch–CFh
50h–7Fh			D0h–FFh

☐ Unimplemented data memory location, read as '0'.

Note 1: Not a physical register.

(Microchip, 2008)

Figure 4: Register file map PIC 16F84

In a simple way BANK0 is used to manipulate data and BANK1 is used to control the operation. For example, to light up an LED connected to PORT A0, you first need to setup RA0 as an output pin using BANK 1 and the move to BANK0 and set the status of the pin to HIGH (logic 1).

STATUS register is used to walk between BANK0 to BANK1. The 8-bit STATUS register is shown in figure 5.

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C
bit 7							bit 0

Figure 5: STATUS register

The 5th bit or RP0 bit is known as the register bank select bit. Setting bit 5 of the STATUS register to '1' select BANK 1 and clearing this bit to '0' select BANK0. The STATUS register is located at 03h address. Tri state registers TRISA and TRISB located in BANK1 used to configure status of the I/O pin. TRISA and TRISB registers are used to set input and output status of PORTA and PORTB pins respectively.

Setting and clearing pins of tri state registers (TRISA & TRISB) can be done by sending '1' or '0' to the appropriate register. Consider PORTA as an example, it has 5 bits (RA0 to RA4) and sending 10101 binary pattern (15h) to TRISA make pin 17, 1 and 3 as inputs and pin 18 and 2 as outputs.

There are another two important registers named W and F. All the calculations and logical manipulations such as addition, subtraction, $\cap$ (&) and $\cup$ (OR) are executed via W register. In order to move data from A to B, the data has to move from A to W and then from W to A. The W register is also known as the working register. The register F represents any location in the internal RAM regardless whether those are special or general purpose registers.

Instructions

There are 35 instructions in the PIC16F84 instruction set consisting of an opcode and operand/s. Basically, the opcode specifies what to do and the operand/s specify how or where. These instructions are split into several groups as follows.

Table 1: List of Instructions

MOVE INSTRUCTIONS	
MOVF	Move F
MOVLW	Move the literal value to W
MOVWF	Move The Contents Of W Into The Register Address That Follows
CLEAR INSTRUCTIONS	
CLRF	Clear F
CLRW	Clear W
ARITHMATIC INSTRUCTIONS	
ADDWF	Add the contents of the W register and any other register that specified in the program (ADD W & F)
SUBWF	Subtract the content of W from F (syntax is same as the above)
ADDLW	Adds the contents of the W register to a number specified.
SUBLW	Subtract the content of W from the literal

LOGICAL INSTRUCTIONS	
ANDWF	Perform an AND function on the W register and another register.
IORWF	Perform inclusive OR function on W and another register
XORWF	Perform exclusive OR function on W and another register
ANDLW	Perform an AND function with the contents of the W register
IORLW	Inclusive OR literal with W
XORLW	Exclusive OR literal with W
COMF	Provide inversion (compliment) of the bits in the specified register
DECREMENTING AND INCREMENTING INSTRUCTIONS	
DEC	Decrement F
INC	Increment F
BIT SETTING AND CLEARING	
BSF	Set the bit that specified in a register (set bit to 1)
BCF	Clear a bit that specified in a register (set bit to 0)
PROGRAM CONTROL INSTRUCTIONS	
GOTO	Program directed to another address (instruction does exactly what it says)
CALL	Calling a subroutine (CALL followed by the subroutine name)
RETURN	Return from subroutine to the main program
RETLW	Return with a literal value which is placed in W
RETFIE	PC point back to the main program after finishing the interrupt routine
SKIPPING INSTRUCTIONS	
DECFSZ	DECFSZ command will decrement the value stored in the F register by 1. If the result is not 0, then the next instruction will be executed otherwise the next instruction will be skipped.
INCFSZ	INCFSZ command will increment the value stored in the F register by 1. If the result is not 0, then the next instruction will be executed otherwise the next instruction will be skipped.
BTFSC	This instruction will test the bit we specify in the register. If the bit is a 0, then the next instruction will be skipped.
BTFSS	This instruction will test the bit we specify in the register. If the bit is a 1, then the next instruction will be skipped.
ROTATION & SWAP INSTRUCTIONS	
RRF	Move a bit in a register one place to the right. (divide by 2)
RLF	Move a bit in a register one place to the left. (multiplied by 2)

SLEEP & WATCHDOG TIMER	
SLEEP	Instruction does exactly what it says
CLRWDT	Clear watchdog timer
MISCELLANEOUS	
NOP	No operation. Add one cycle delay to the program.
OPTION	Not recommended
TRIS	Not recommended

Introduction to programing

Writing to the Ports

The following code is for sending some data to PORTA.

```
bsf         03h,5       ;Go to Bank 1
movlw       00h         ;Put 00000 into W
movwf       85h         ;Move 00000 onto TRISA – all pins set to output
bcf         03h,5       ;Come back to Bank 0
```

First line of the program uses bit set instruction (bsf). This operation set the 5th bit of the register which is located at 03h address. Figure 4 illustrates the 03h address belonging to the STATUS register. Setting the 5th bit of STATUS register (Figure 3) select the BANK1 where the tri state registers (TRISA & TRISB) are located. In the next line of the program move the 00h literal value into the working register (W). The value stored in the W register moves to 85h address in the next instruction line. Sending all zeros to 85h address (TRISA , refer Figure 4) make all pins to output mode.

Now it is possible to send 5V or 0V to the pins by setting the pin status to logic 1 (high) or logic zero (low) respectively. Let us assume LED is connected to PORTA, Pin 2 and ground. Making Pin2 status high, lit the LED and making it low, off the LED. This can be done by the following code.

Switch on LED

```
movlw       02h         ;Write 02h to the W register.  In binary this is 00010, which
                        ;puts a '1' on bit 2 (pin 18) while keeping the other pins to '0'
movwf       05h         ;Now move the contents of W (02h) onto the PortA, at 05h
```

Switch off LED

```
movlw       00h     ;Write 00h to the W register.
movwf       05h     ;Now move the contents of W (0h) onto the Port A, at 05h
```

In this program code, some numerical values are used to address registers. This can be avoided using "equ' instruction. The 'equ' instruction simply means something equals something else. It is not an instruction for the PIC, but for the assembler. Let us assign some names for constants using 'equ' instruction.

```
STATUS      equ  03h     ;assigns the word STATUS to the value of 03h,
                         ;which is the address of the STATUS register.
TRISA       equ 85h      ; assigns the word TRISA to the value of 85h,
                         ;which is the address of the Tri-State register for PortA
PORTA       equ 05h      ;assigns the word PORTA to 05h which is the
                         ;address of Port A.
.
```

Assigning names to constants make easy to understand the program.

Delay Loops

Adding delays to various parts of the program is an essential part. Therefore it is required to understand the clock frequency and timing for one instruction cycle. This PIC chip requires 4 clock cycles to complete one instruction cycle. In other words using a 4MHz oscillator, each instruction will take 1µS to complete. Therefore our LED blinking program finished in less than 10 µS time. This is far too fast for us to see, and it will appear that the LED is permanently on. To observe the blink it is required to introduce a delay between turning the LED on and turning the LED off.

Generally a delay is introduced using up counting or down counting until it reaches zero. Once it reaches zero, program counter (PC) exits from the delay loop and continue through the main program.

Reading from the Ports

It is clear that sending zeros (0) or ones (1) to tri state registers (TRISA or TRISB) made the pin status to output or input respectively. The following code shows setting up a bit 2 (pin 20) of the PORTA as input.

```
movlw       02h         ;load litereal 02
movwf       TRISA       ;Set the Port A RA1  to input.
bcf         STATUS,5    ;Switch back to Bank 0
```

The above code made the RA1 pin as input. Now it is possible to check the status of the input continuously and change the program when it is low or high.

According to the instruction set (Table 1), status of a bit can be checked by BTFSC or BTFSS commands. The BTFSC command checks the status of the bit (input is high or low) and continues checking until the bit is clear (low input). Program counter exit from the loop and do the rest when the bit becomes logic low. In a similar way BTFSS command check the status of the bit and exit from checking when the bit set to logic high.

```
loop
BTFSS        PortA,1        ;Get the value from PORT A, bit1 (skip loop if bit is set, otherwise loop)
Goto loop
Carry on here (skip from loop)
```

MPLAB® IDE

MPLAB Integrated Development Environment (IDE) is a comprehensive editor, project manager and design desktop for application development of embedded designs using Microchip PICmicro® microcontrollers.

Initially run the ***MPLAB.IDE***, then select ***File > New*** to create blank edit window. It is possible to type code in this window or copy and paste on this window. The new file save in a new directory named ***C:\MyProject*** as ***LEDBlink.asm***.

To start a new project select ***Project > Project Wizard*** and press ***Next*** in the Welcome screen. In the next screen select the PIC micro used in the project (16F84A) from the pull down menu. In the next screen full path to the MPASM assembler executable should appear. If this box is empty click browse to locate ***mpasmwin.exe*** in the machine. The next screen requires to enter project name and directory to save the project. Project name would be any meaningful name and browse the ***C:\MyProject*** for the directory. Add the source file into the project in the next screen (select ***LEDBlink.asm*** and press Add). Click ***Next*** and check the summary before you finish the wizard.

Now complete the work required to build the project. In this process Microchip MPASM toolsuite assemble the source code. Select ***Project > Build All*** to start this process. Output window deliver the message "BUILD SUCEEDED" for successfully assemble files. In this process generates a ***LEDBlink.hex*** file.

Programing the device

There are several programing devices to program microcontrollers. Programmer or programing device is used to transfer the hex file into the real device. Initially it is required to establish the communication between programmer and the computer. The hex file is then opened using programmer and transfer the program into the microcontroller. Once the programing is finished it is possible to remove the microcontroller from the programmer. Now the loaded program may run when it is powered up.

Experiment 1- Blinking LED (Sending data to ports)

Apparatus:

Demonstration board, PIC programmer, PIC 16F84A microcontroller, DC power supply

Theory:

Changing the output pin status from low to high and high to low continuously. Connecting LED to the above pin makes the status change visible. The LED on and off times are important to identify the change in the pin status. Fast switching or blinking cannot be identified by the human eye. Therefore it is required to add a delay in between on and off states. The demonstration board is connected to a 4 MHz crystal hence evaluate the time for one instruction cycle before writing the delay loop.

Learning outcome:

At the end of the experiment, the student will be able to demonstrate skills on identifying microcontroller pins and sending data to microcontroller ports.

Procedure:

Part 1

Read the above description carefully and write the assembly program to blink the LED in the following diagram (Figure 6). Identify the LED's and crystal oscillator in the demonstration board.

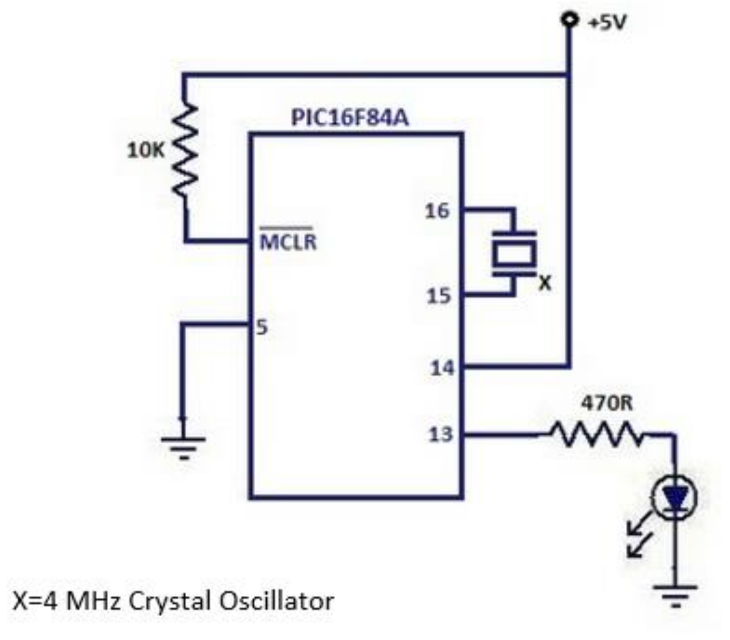


Figure 6: Circuit diagram for Experiment 1

Build the project using MPASM.

Transfer the hex file to the microcontroller using a given programmer.

Remove the chip from the programmer and fix it to the demo board.

Make the required connection according to the diagram

Observations:

Conclusion:

Part 2

Change the parameters in the delay loop and observe the difference.

Observations:

Conclusion:

Experiment 2- Push button operation (Reading data from ports)

Apparatus:

Demonstration board, PIC programmer, PIC 16F84A microcontroller, DC power supply

Theory:

Microcontroller I/O pins can be configured as either input or output. Input configuration is useful when connecting external sensors to the system. Change of the sensor or any other device connected to the input pin can be sensed by the microcontroller and perform the necessary action.

Learning outcome:

At the end of the experiment, the student will be able to demonstrate skills on receiving signals from sensors.

Procedure:

Part 1

Push button or a sensor that can be used to connect to the microcontroller as shown in figure 7. Write the assembly code to operate the LED in experiment 1 when the push button is pressed.

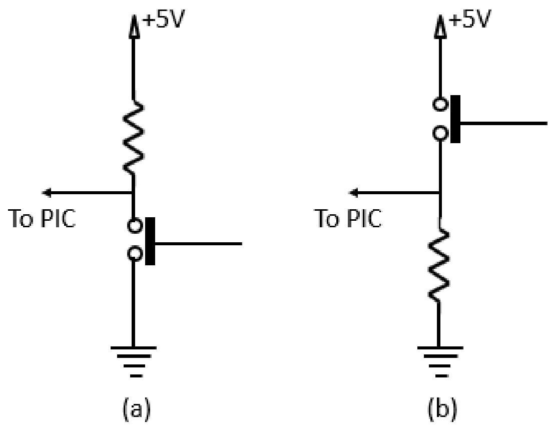


Figure 7: Push button configurations

Download the file to the microcontroller and run the program.

Observations:

Conclusion:

PART 2

Rearrange the program to identify the switch position. In this case one LED is required to indicate the push position and another LED is required to indicate button release position.

Observations:

Conclusion:

References

Microchip. (2008). PIC 16F84A datasheet.