Hashemite University

Faculty of Engineering and Technology

Computer Engineering Department

Embedded Systems and DIC Lab Manual

110408326

Table of Content

Hashemite University
Faculty of Engineering and Technology
Computer Engineering Department

Experiment 1
Review of PIC Microcontroller & MPLAB IDE

1.1 Objectives:

- Review of the PIC microcontroller.
- Microchip MPLAB Integrated Development Environment (IDE) and the whole process of building a project, writing simple codes, and compiling the project.
- Code simulation.

1.2 Pre-lab Preparation:

- Read the experiment thoroughly BEFORE coming to the lab.

1.3 Equipment:

- Personal computer with MPLAB software installed on it.

1. Review of PIC Microcontroller:

1.1: PIC Features:

- Only 35 single word instructions
- Operating speed: DC - 20 MHz clock input
- 1024 words of program memory
- 14-bit wide instruction words
- 8-bit wide data bytes
- 15 Special Function Hardware registers
- Eight-level deep hardware stack
- I/O pins with individual direction control
- High current sink/source for direct LED drive
- 10,000 erase/write cycles Enhanced FLASH Program memory typical
- Low power, high speed technology

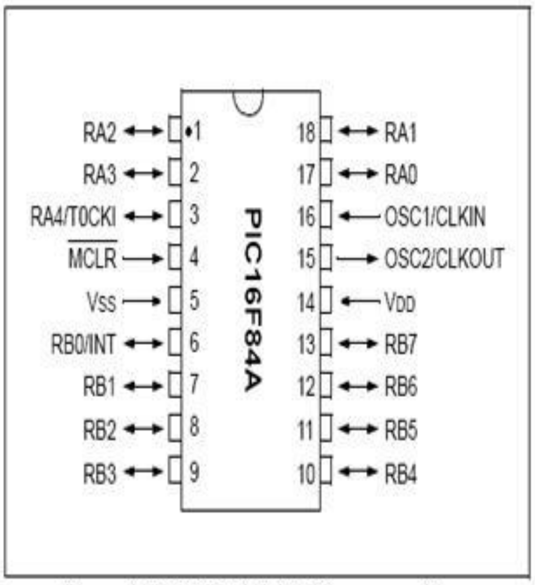


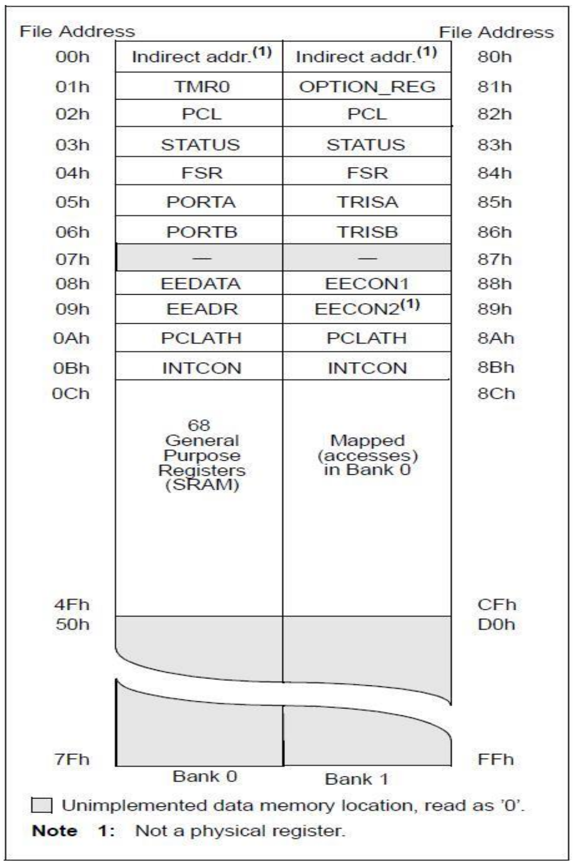
Figure 1: PIC16F84A 8-bit Microcontroller

1.2 Memory Map:

A memory map shows all available registers (in data memory) of a certain PIC along with their addresses, it is organized as a table format and has two parts:

1. **Upper part:** which lists all the Special Function Registers (SFR) in a PIC; these registers normally have specific functions and are used to control the PIC operation

2. **Lower part:** which shows the General Purpose Registers (GPR) in a PIC; GPRs are data memory locations that the user is free to use as he wishes.



File Address			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
	Bank 0	Bank 1	

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

Note 1: Not a physical register.

Figure 1: Data memory organization

Memory Maps of different PICs are different. Refer to the datasheets for the appropriate data map

Notice that the memory map is divided into two columns, each column is called a bank, here we have two banks: bank 0 and bank 1. In order to access bank 1, we have to switch to that bank first and same for bank 0. But how do we make the switch?
Look at the details of the STATUS register in the figure below, there are two bits RP0 and RP1, these bits control which bank we are in:

- If RP0 is 0 then we are in bank 0
- If RP1 is 1 then we are in bank 1

We can change RP0 by using the **BCF**/**BSF** instructions

- BCF STATUS, RP0 RP0 in STATUS is 0 switch to bank 0

- BSF STATUS, RP0 RP0 in STATUS is 1 switch to bank 1

*BCF: **B**it **C**lear **F**ile Register (makes a specified bit in a specified file register a 0)*
*BSF: **B**it **S**et **F**ile Register (makes a specified bit in a specified file register a 1)*

1.2: Starting MPLAB

After Creating new project and writing the code instructions we should build the project, do so by pressing ***build***

An output window should show: BUILD SUCCEDDED

BUILD SUCCEED DOES NOT MEAN THAT YOUR PROGRAM IS CORRECT, IT SIMPLY MEANS THAT THERE ARE NO **SYNTAX** ERRORS FOUND, SO WATCH OUT FOR ANY LOGICAL ERRORS YOU MIGHT MAKE.

Notice that there are several warnings after building the file, warnings do not affect the execution of the program but they are worth reading. This warning reads: "Found opcode in column 1".

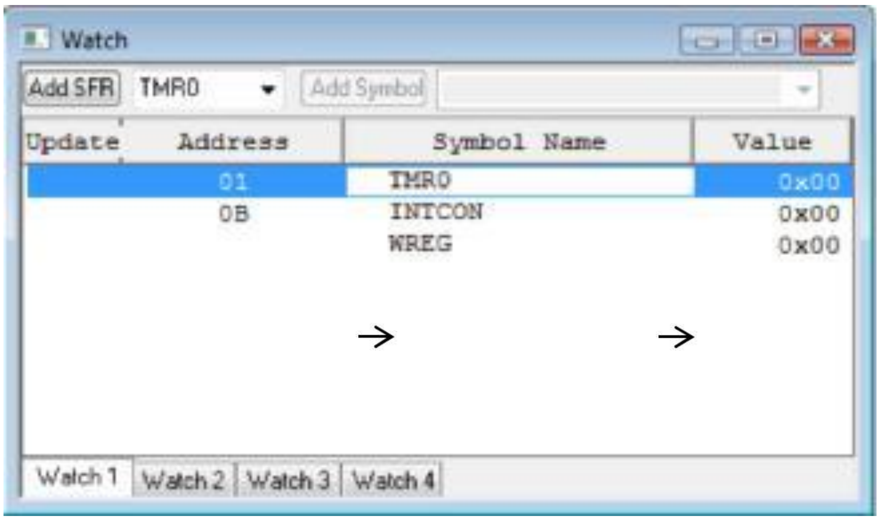
Preparing for simulation

Go to View Menu → Watch

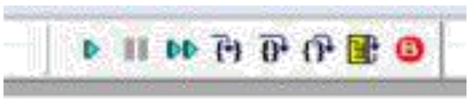
From the drop out menu choose the registers we want to watch during simulation and click ADD SFR for each one
Add the following:

- WREG: working register
- TMR0
- INTCON

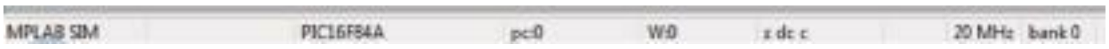
You should have the following:



Update	Address	Symbol Name	Value
	01	TMR0	0x00
	0B	INTCON	0x00
		WREG	0x00



1. To begin the simulation, we will start by resetting the PIC; do so by pressing the yellow reset button. A green arrow will appear next to the first instruction.
The green arrow means that the program counter is pointing to this instruction *which has not been executed yet.*
Notice the status bar below:



MPLAB SIM PIC16F84A pc:0 W:0 z dc c 20 MHz bank 0

Keep an eye on the value of the program counter (pc: initially 0), see how it changes as we simulate the program

2. Press the "Step Into" button one at a time and check the Watch window each time an instruction executes; keep pressing "Step Into" until you reach the NOP instruction then STOP.
Compare the results as seen in the Watch window with those expected.

1.3 PIC Instruction set:

1.3.1: Movement instructions:

Since most of the PIC instructions (logical and arithmetic) work through the working register "W", that is one of their operands must always be the working register "W; therefore we need the following movement operations:

1. Moving constants to the working register (Loading) Using "MOV**LW**" instruction**.**
2. Moving values from the data memory to the working register (Loading) Using "MOV**F**" instruction.
3. Moving values from the working register to the data memory (Storing) Using "MOV**F**" instruction.

Examples:

1. MOVWF 01	; COPIES the value found in W to TMR0
2. MOVLW 10	; moves the constant **16** to the working register.
3. MOVF 2D, 0	; copies the content of the GPR 2D the working register
4. MOVF 05, 1	; copies the content of PORTA to itself

1.3.2 Conditional instructions:

The PIC 16series instruction set has four instructions which implement a sort of conditional statement: ***btfsc*** , ***btfss, decfsz and incfsz*** instructions.
1. **btfsc** checks for the condition that a bit is clear: 0 (***B***it ***T***est ***F***ile, ***S***kip if ***C***lear)
2. **btfss** checks for the condition that a bit is set one: 1 (***B***it ***T***est ***F***ile, ***S***kip if **Set**)
3. Review ***decfsz*** *and* ***incfsz*** *functions from the datasheet*

Example 1:

```
movlw 0x09
btfsc PORTA, 0
movwf Num1
movwf Num2
```

The above instruction tests bit 0 of PORTA and checks whether it is clear (0) or not

- If it is clear (0), the program will **skip "**movwf Num1" and will only execute "movwf Num2" **Only Num2 has the value 0x09**
- If it is set (1), it will not skip but **execute** "movwf Num1" and then **proceed** to "movwf Num2" **In the end, both Num1 and Num2 have the value of 0x09**

You have seen above that **if the condition fails**, the code will continue normally and both instructions will be executed.

1.3.3 Directives:

Directives are not instructions. They are **assembler commands** that appear in the source code but are not usually translated directly into machine code. They are used to control the **assembler**: its input, output, and data allocation. They are not converted to machine code (.hex file) and therefore not downloaded to the PIC.

DIRECTIVES THEMSELVES **ARE NOT CASE-SENSITIVE** BUT THE **LABELS** YOU DEFINE **ARE**. SO YOU MUST USE THE NAME AS YOU HAVE DEFINED IT SINCE IT IS CASE-SENSITIVE.

1- The "END" directive

The "END" command is a directive which tells the MPLAB IDE that we have finished our program, anything which is written after the end command will not be executed and any variable names will be undefined.

2- The "EQU" directive

The equate directive is used to **assign** labels to numeric values. They are used to *DEFINE CONSTANTS* or to *ASSIGN NAMES TO MEMORY ADDRESSES OR INDIVIDUAL BITS IN A REGISTER* and then use the name instead of the numeric address as follow:

Var1 equ 01

3- The "INCLUDE" directive

Suppose we are to write a huge program that uses all registers. It will be a tiresome task to. Therefore we use the include directive. The include directive calls a file which has all Special Function Registers (SFR) and bit namesdefine using "equate" statements for you and ready to use, and its syntax is :
#include "PXXXXXXX.inc" where XXXXXX is the PIC part number
Older version of include without #, still supported.

4- The "Cblock" directive

The cblock defines variables in sequential locations, see the following declaration:

```
Cblock 0x35
        VarX
        VarY
        VarZ
endc
```

Here, VarX has the starting address of the cblock, which is 0x35, VarY has the sequential address 0x36 and VarZ the address of 0x37

5- The Origin "org" directive

The origin directive is used to place the instruction ***which exactly comes after it*** at the location it specifies.

Examples:

```
Org 0x00
Movlw 05          ;This instruction has address 0 in program memory
Addwf TMR0        ;This instruction has address 1 in program memory
Org 0x13          ;WRONG, org only takes even addresses
```

In This Course, Never Use Any Origin Directives Except For Org 0x00 And 0x04, Changing Instructions' Locations In The Program Memory Can Lead To Numerous Errors.

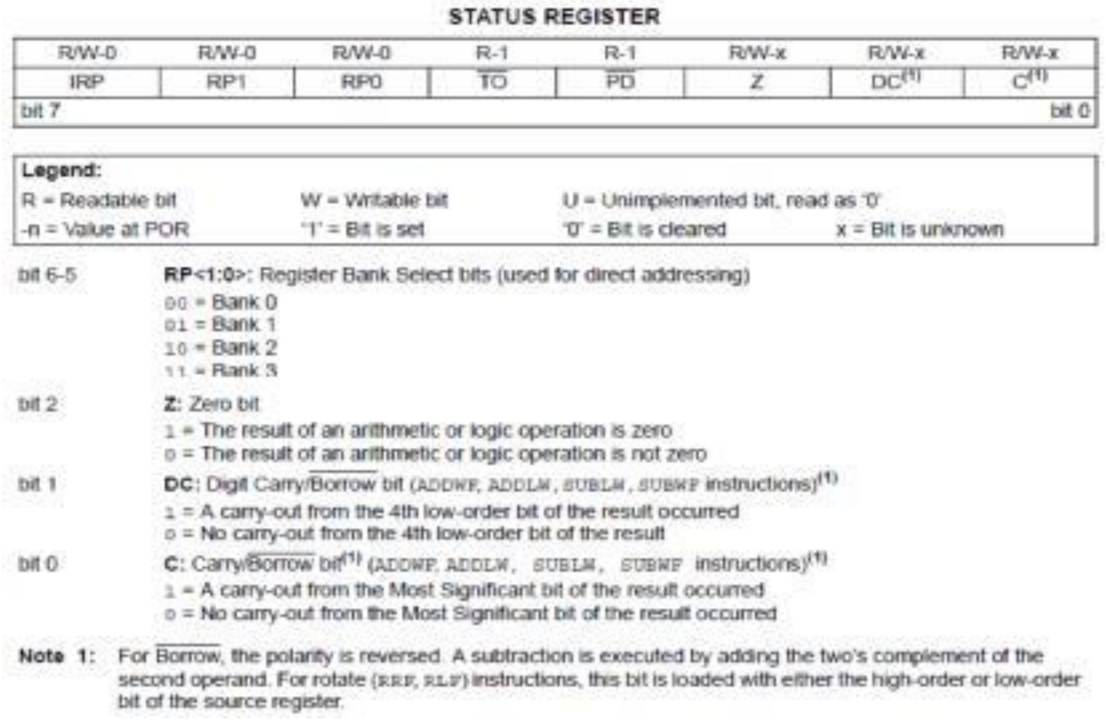
6- The "Banksel" directive

The**BANKSEL** directive is a command to the assembler and linker to generate bank selecting code to set the bank to the bank containing the designated *label*

Example:
BANKSEL TRISA will be replaced by the assembler, which will automatically know which bank the register is in and generate the appropriate bank selection instructions:
Bsf STATUS, RP0
Bcf STATUS, RP1

FLAGS

The PIC 16 series has three indicator flags found in the STATUS register; they are the C, DC, and Z flags. See the description below. Not all instructions affect the flags; some instructions affect some of the flags while others affect all the flags. Refer to the Appendix at the end of this experiment and review which instructions affect which flags.



STATUS REGISTER

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[1]	C[1]
bit 7							bit 0

Legend:

R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 6-5	**RP<1:0>**: Register Bank Select bits (used for direct addressing) `00` = Bank 0 `01` = Bank 1 `10` = Bank 2 `11` = Bank 3
bit 2	**Z**: Zero bit `1` = The result of an arithmetic or logic operation is zero `0` = The result of an arithmetic or logic operation is not zero
bit 1	**DC**: Digit Carry/$\overline{\text{Borrow}}$ bit (`ADDWF`, `ADDLW`, `SUBLW`, `SUBWF` instructions)[1] `1` = A carry-out from the 4th low-order bit of the result occurred `0` = No carry-out from the 4th low-order bit of the result
bit 0	**C**: Carry/$\overline{\text{Borrow}}$ bit[1] (`ADDWF`, `ADDLW`, `SUBLW`, `SUBWF` instructions)[1] `1` = A carry-out from the Most Significant bit of the result occurred `0` = No carry-out from the Most Significant bit of the result occurred

Note 1: For $\overline{\text{Borrow}}$, the polarity is reversed. A subtraction is executed by adding the two's complement of the second operand. For rotate (`RRF`, `RLF`) instructions, this bit is loaded with either the high-order or low-order bit of the source register.

Types of Logical and Arithmetic Instructions and Result Destination

The PIC16 series logical and arithmetic instructions are easy to understand by just reading the instruction, for from the name you readily know what this instruction does. There are the ADD, SUB, AND, XOR, IOR (the ordinary ***I***nclusive ***OR***). They only differ by their operands and the result destination.

Many other instructions of the PIC16 series instruction set are of Type II; refer back to the Appendix at the end of this experiment for study.

1.3.4: Modular Programing:

1- Subroutines:

Subroutines are the closest equivalent to functions

- Subroutines start with a **Label** giving them a name and end with the instruction **return.**
  Example:

doMath Instruction 1 Instruction 2 . . Instruction n **return**	**Process** Instruction 1 Instruction 2 . . **Calculate** Instruction 7 Instruction 8 **return** *Note: This is still one subroutine, no matter the number of labels in between*

- Subroutines can be written anywhere in the program after the org and before the end directives
- Subroutines are used in the following way: Call subroutineName
- Subroutines are stored **once** in the program memory, each time they are used, they are executed from that location
- Subroutines alter the flow of the program, thus they affect the stack

2- Macros:

Macros are declared in the following way:

```
macroName    macro
                     Instruction 1
                     Instruction 2
                     .
                     .
                     Instruction n
             endm
```

- Macros should be declared before writing the code instructions. It is not recommended to declare macros in the middle of your program.
- Macros are used by only writing their name: macroName
- Each time you use a macro, it will be replaced by its body, refer to the example below. Therefore, the program will execute sequentially, the flow of the program will not change. The Stack is not affected

3- Look up tables:

Look up tables are a special type of subroutines which are used to retrieve values depending on the input they receive. They are invoked in the same as any subroutine: Call tableName They work on the basis that they change the program counter value and therefore alter the flow of instruction execution The

retlw instruction is a **return** instruction with *the benefit that it returns a value in W* when it is executed.

Syntax:

```
lookUpTableName
        addwf PCL, F        ;add the number found in the program counter to PCL
        nop
        retlw Value         ;if W has 1, execute this
        retlw Value         ;if W has 2, execute this
```

The General structure of a complete program :(Very Important)

```
;Program Title, description of its function, its inputs and outputs and the programmer name
;**************************************************************
;* FUNCTION:
;* PROGRAMED BY: ENG. EZYA KHADER
;* INPUTS:
;* OUTPUTS:
;* *************************************************************
#include "P16F84A.inc"
; ------------------------------------------------------
; Local equates for your own symbolic designators
; ------------------------------------------------------

NUM1        equ 20h        ; The NUM1 number is in File 20h

; ------------------------------------------------------
; General Purpose RAM Assignments (Cblock definitions)
; ------------------------------------------------------
                    cblock 0x30
                          MY_VAR          ; MY_VAR is in File 30h

                    endc
; ------------------------------------------------------
; Macro Definitions
; ------------------------------------------------------
MY_MACROmacro
                    Instruction 1
                    :
                    nop
              endm
; ------------------------------------------------------
; Vector definition
; ------------------------------------------------------
                    org 0x000
                    nop
                    goto Main

INT_Routine         org 0x004
                    goto INT_Routine
; ------------------------------------------------------
; The main Program
; ------------------------------------------------------
Main
      Instruction x
      Instruction y
      Call MY_FUN
      :
      Instruction z
      :
      Instruction n
      goto Finish

; ------------------------------------------------------
; Sub Routine Definitions
; ------------------------------------------------------
MY_FUN
      Instruction w
      Instruction xx
   return
; ------------------------------------------------------
; The main Program end
```

Hashemite University
Faculty of Engineering and Technology
Computer Engineering Department

Experiment 2
I/O Interfacing & the 7-Segment Display

1.1 Objectives:

1. I/O ports of the Microcontroller.
2. Learn how to interface the 7-segment display with PIC Microcontroller.
3. Learn how to distinguish between the common anode and the common cathode of the 7-segment.
4. Learn the Multiplexing technique of more than one digit of the 7-segment display.
5. Stressing software and hardware co-design techniques by introducing the *Proteus* IDE package.

1.2 Pre-lab Preparation:

- Read the experiment thoroughly BEFORE coming to the lab.
- Review the PIC16F84A data sheet chapters 1, 2 especially (2.1, 2.2, 2.3).
- Review Experiment 3.
- Read Appendix B.(*very important*)

1.3 Equipments:

- Personal computer with MPLAB and *Proteus* software installed on it.

1.4 Theoretical backgrounds:

First: Light Emitting Diodes

Light emitting diodes or LEDs are semiconductor light sources used as indicator lamps in many electronic devices. Modern versions are available across the visible, ultraviolet and infrared wavelengths, with very high brightness and come in a variety of shapes and sizes. The physics behind LED operation is covered in the Electronics I course and will not be offered here.

In order to switch a LED on, forward current must pass from the anode to the cathode, but how to determine which pin of the LED is anode and which is cathode, generally, there are two ways:

1. The longer lead is anode, the shorter is cathode.
2. The cathode has a flat surface as shown in Figure 1.

Resistors with values in between 220Ω to 1kΩ are placed in between the voltage source (often 5 to 9V or even more) and the anode to limit the current entering the LED or else it will burn. The lesser the resistor value, the brighter the LED shines (Ohms Law). In this case these resistors are called current limiting resistors.

Figure 2 shows how to interface a LED to the PORTC pin 1

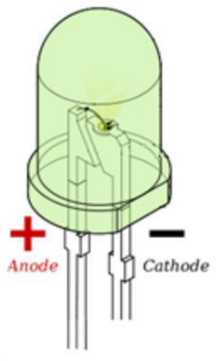


Figure 1

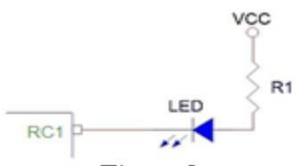
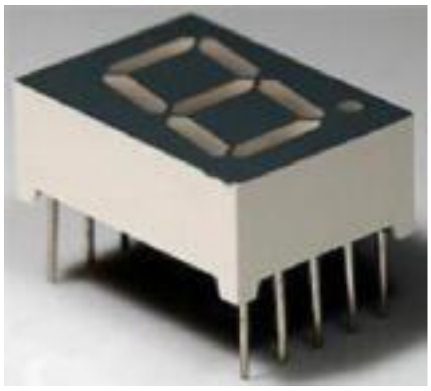
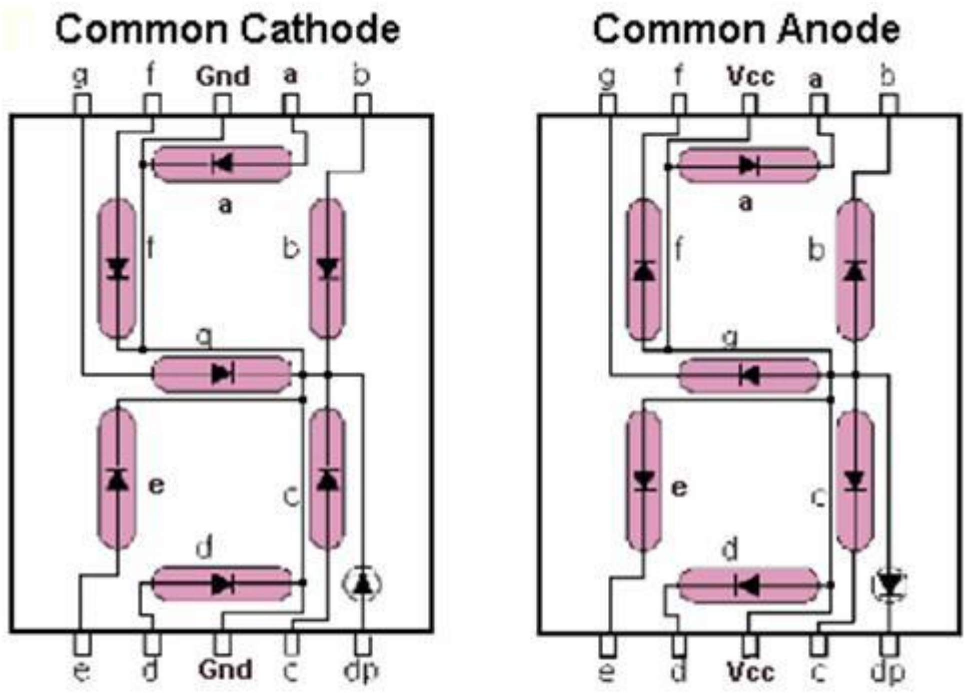


Figure 2

Second: Seven-Segment Display



A Seven-Segment display, as its name implies, is composed of seven segments (or technically of seven LEDs) which can be individually switched on or off. This ability to individually control each segment and the layout in which these segments are distributed allows for the representation of the numerals and some characters. If the anode ends of all LEDs are connected together, it is called common anode display. If the cathodes of all LEDs are connected together, it is called a common cathode display. To switch a LED on in a common cathode configuration, you have to send logic high to the segment pin. Conversely, to switch a LED on in a common anode configuration, you have to send logic low to the segment pin.



Interfacing a Seven-Segment display is independent of the type of the module, whether it is common anode or common cathode, only the logic level sent to the display differs. Finally, since the display is basically LEDs, current limiting resistors are used for each segment.

Multiplexing Technique of the 7-segment display

Multiplexing technique is based on the idea of Persistence of vision of the human eyes. The sample schematic of 3 digits multiplexing is shown below (figure 3).

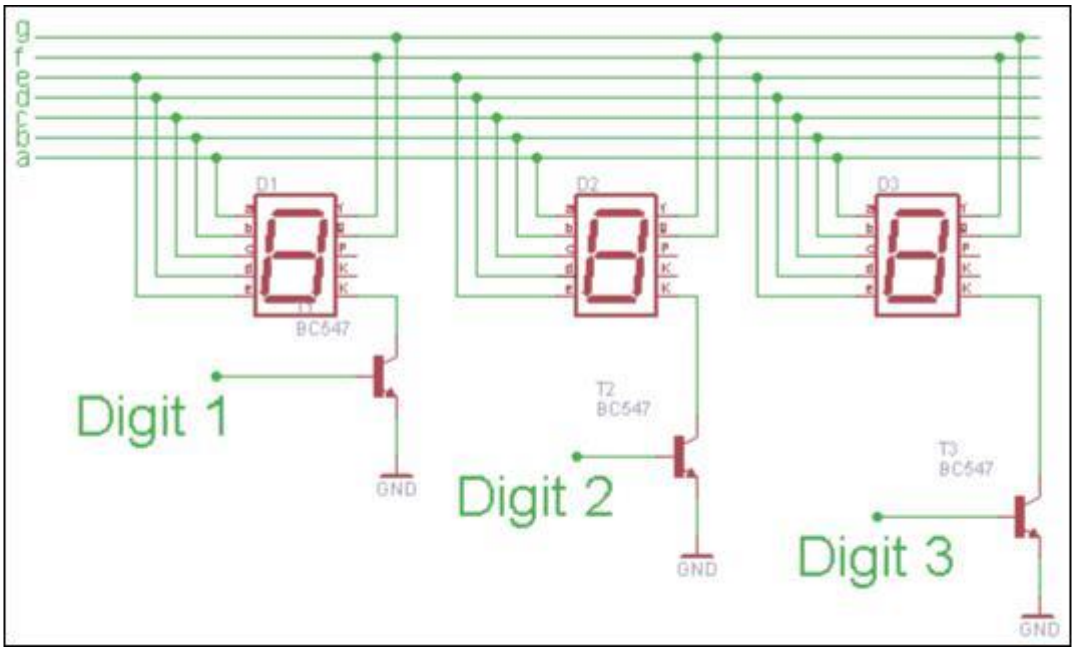
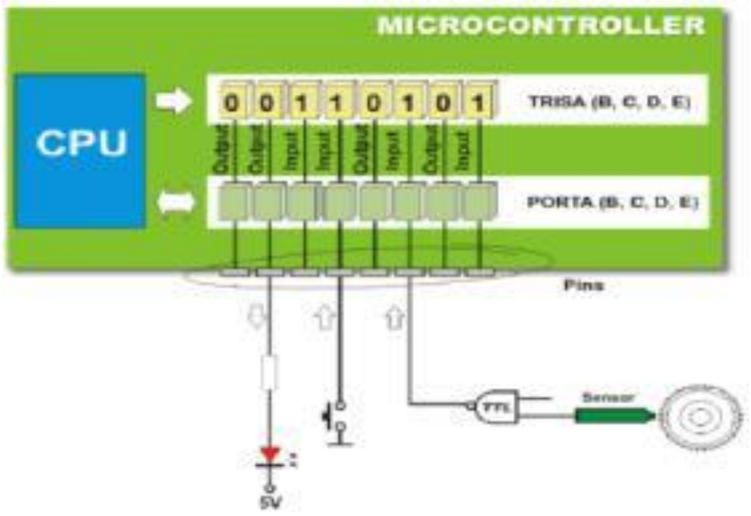


Figure 3: 7-segment multiplexing of 3 digits

Segment a-g of each digit are connected together. Each digit is switched on-off by controlling signal at Digit 1, Digit 2 and Digit 3. For example, if Digit 1 is '1', Digit 1 will be on. If Digit 1 is '0', Digit 1 will be off. People will see all 3 digits display in the same time if each digit switch on and off fast enough.

I/O ports of Microcontroller:

PIC microcontrollers' ports are general-purpose bi-directional digital ports. The state of *TRISx* Register controls the direction of the *PORTx* bits. A logic one in a bit position configures the PICto act as an input and if it has a zero to act as an output. However, a pin can only act as eitherinput or output at any one time but not simultaneously. This means that each pin has a **distinctdirection** state.*See the following figure.*



Examples:

Movlw 0x0F Movwf TRISB	Clrf TRISC	Clrf TRISD Comf TRISD, F	Movlw B'00110011' Movwf TRISB
The high nibble of PORTB is output, low nibble is input	Whole PORTC as output	Whole PORTD as input	Bits 2, 3, 6, 7 as output Bits 0, 1, 4, 5 as input

How to decide whether microcontroller's ports must be configured as inputs or outputs?

Input ports "Get Data" from the outside world into the microcontroller while output ports "Send Data" to the outside world.

- LEDs, 7-Segment displays, motors, and LCDs (write mode) that are interfaced to microcontroller's ports should be configured as output.
- Switches, push buttons, sensors, keypad and LCDs (read mode) that are interfaced to microcontroller's ports should be configured as input.

Software Introduction of QL200 Programming Module

QL200 programming software QL_PROG is what our company specifically developed for programming module QL_PROG of QL200.The software can be procured from the CD-ROM coming along with our product, or you may also download the latest version from our website.

1. Installation of Software

Find file QL-PROGvXX.EXE under "soft" of CD, double click this file to access installation interface of application software, and operate according to prompts to finish the installation.

2. Installation of USB Driver

When you first use the USB of this programmer for communication, please connect PC and programmer hardware through USB cable. Then, PC system will indicate to find new hardware and request installation of driver for this hardware.
You can operate in regular way and designate path to install driver, which is under directory "QL200 USB-driver" of attached CD.
After installation, a serial port will be added to your computer. Now the programmer is connected to the serial port converted from USB.
When there is no system message to find new hardware after programmer hardware is connected, please check:

- Any bad or broken connection for USB cable?
- Does USB of computer works smoothly?
- Maybe there is similar USB driver already installed in your computer. You may connect programmer hardware, right click "My Computer", select "Property" in pop-up list, switch to "Hardware" from "System Property" window, and then click "Device Manager" to check "Port (COM and LPT)". If you can find "USB Serial Port (COMX)", the driver of this programmer has been installed in your computer (this operation requests proper connection of hardware and PC).

3. Running of Software

After proper installation, shortcut of this software will appear on desktop. We can double click it to run the software. (Note: Run the software after proper connection of hardware so as to identify the hardware easily.)

Use of Software

1）**Select Communication Port/Select Programmer Hardware**: Connect hardware and power up, run the software which will automatically search programmer hardware. If search fails and indicates “Have not find board…”, you can set programmer hardware manually. We offer two methods for user to select communication port and programmer type. Directly select the port from the list of “Port Selector” at top right of the software. (Note: Only existing ports of computer are shown in the list. If only COM1 is installed at your computer, there is only COM1 for your choice. If you have connected USB but can’t find it in the list, the reason may be that the software is started too fast. You can shut down the software and restart it.)

2) Select Chip Type:

The control for selection of chip type is at top right of the software.
You may select the type to shorten the chip list for fast selection of proper chip.
If you don’t know exactly the device, please select “All Chip” in the list of “Chip Family”

3) Select Chip

The control for selection of exact chip is at top right of the software, if the desired chip type is not listed, please change the chip Family or select “All Chips” and try again.

4) Load Code

Run【File】-【Load File】, or press “Load” button on programming software panel to load machine code file of the chip you desire to program.
This programmer software supports load of BIN files and HEX files.

5) Download code

Press "Program" button to begin programming. After completion, there will be messages of "PASS" or "ERROR".

Experiment 3
PIC Microcontroller Interrupts

1.2 Objectives:

- Interrupts and interrupt service routines.
- Interrupts control register and its flags.

1.2 Pre-lab Preparation:

- Read the experiment thoroughly BEFORE coming to the lab.
- Read from the data sheet section 2.1 & 2.3 carefully.

1.3 Equipments:

- Personal computer with MPLAB and *Proteus* software installed on it.

1.4 Theoretical backgrounds:

There are 2 methods for communicatingbetween the Microcontroller and the externalsystem:

- Polling
- Interrupts

First: Polling

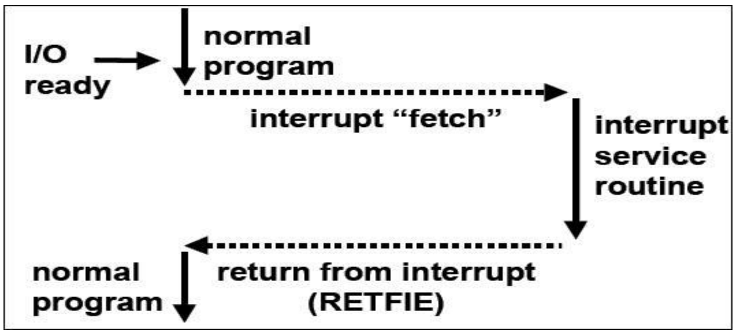
In this method, microcontroller accesses at the exact time interval the external device, and gets the required information, the time periods is determined by user. In fact, you can say that, when using the method - polling, the processor must access the device itself and request the desired information that is needed to be processed.

In fact we see that in this method, there is no independence for the external systems themselves. They depend on the microcontroller. The processor may only access the external device and get from it the information needed.
The main drawback writing program that uses this method is a waste of time. The micro needs to wait and review whether new information arrived.

Second: Interrupts

Interrupts are a mechanism of a microcontroller which enables it to respond to some events at the moment when they occur, regardless of what microcontroller is doing at the time. This is a very important part, because it provides connection between a microcontroller and environment which surrounds it. Generally, each interrupt changes the program flow, interrupts it and after executing an interrupt subprogram (interrupt routine) it continues from that same point on.

So an interrupt is an event that causes the microcontroller to halt the normal flow of the program and execute another program called the interrupt service routine



Interrupts can be thought of as *hardware-initiated subroutine calls.*
Usually, interrupts are generated by I/O devices such as timers or external devices.

General Hardware Structure for Interrupts

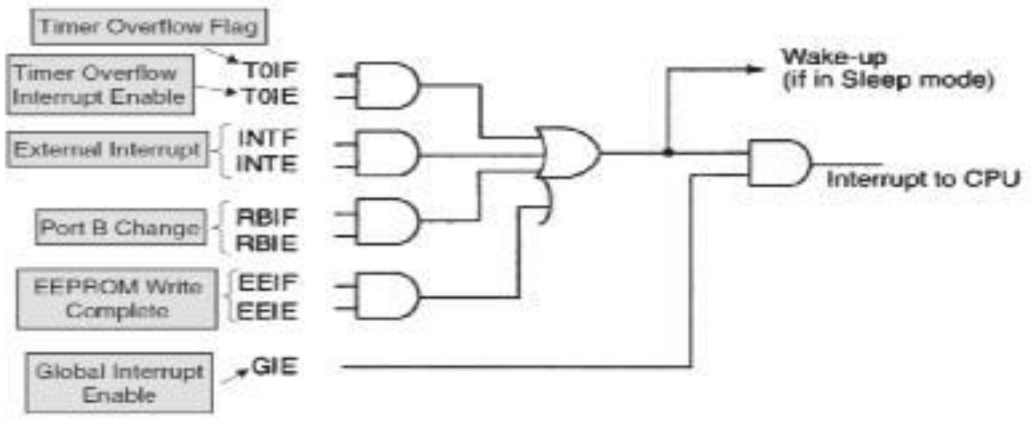
- Interrupts sources can be *external and internal*.
- Two types of interrupts: *maskable*and *non-maskable*.
- Maskable can be enabled/disabled by setting/clearing some bits.
- Non-maskable interrupts can't be disabled and they always interrupt the CPU.
- Usually, each interrupt has a flag (a bit) that is set whenever the interrupt occurs.

Note: No non-maskable interrupts in 16F84A

The 16F84A Interrupt Structure

Sources of interrupts:

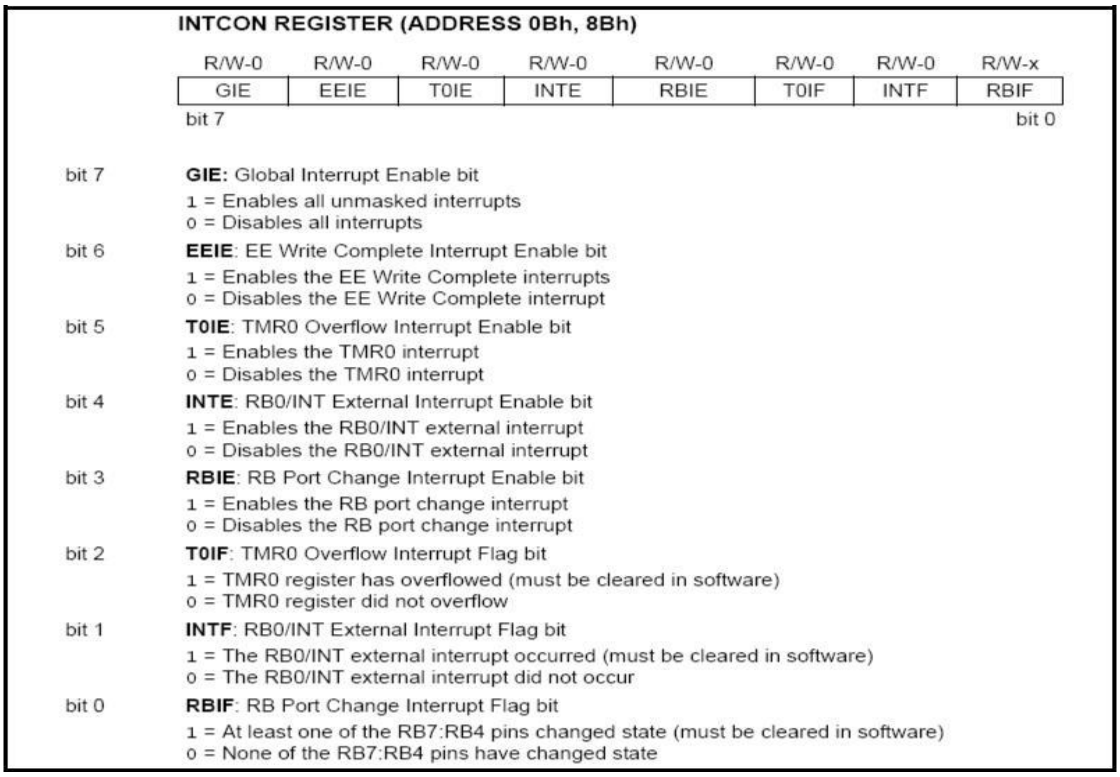
1) ***External interrupt***
    - The only external interrupt input
    - The input is multiplexed with RB0 pin of port B
    - It is edge triggered
2) ***Timer overflow interrupt***
    - It is an internal interrupt that occurs when the 8-bit timer overflows
3) ***Port B interrupt change***
    - An interrupt occurs when a change is detected on any of the upper 4 bits of port B
4) ***EEPROM write complete interrupt***



There are two SFR register we must deal with if we have interrupts;

INTCON REGISTER

The INTCON register is a readable and writable register that contains the various enable bits for all interrupt sources.



INTCON REGISTER (ADDRESS 0Bh, 8Bh)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
bit 7							bit 0

bit 7 **GIE:** Global Interrupt Enable bit
1 = Enables all unmasked interrupts
0 = Disables all interrupts

bit 6 **EEIE**: EE Write Complete Interrupt Enable bit
1 = Enables the EE Write Complete interrupts
0 = Disables the EE Write Complete interrupt

bit 5 **T0IE**: TMR0 Overflow Interrupt Enable bit
1 = Enables the TMR0 interrupt
0 = Disables the TMR0 interrupt

bit 4 **INTE**: RB0/INT External Interrupt Enable bit
1 = Enables the RB0/INT external interrupt
0 = Disables the RB0/INT external interrupt

bit 3 **RBIE**: RB Port Change Interrupt Enable bit
1 = Enables the RB port change interrupt
0 = Disables the RB port change interrupt

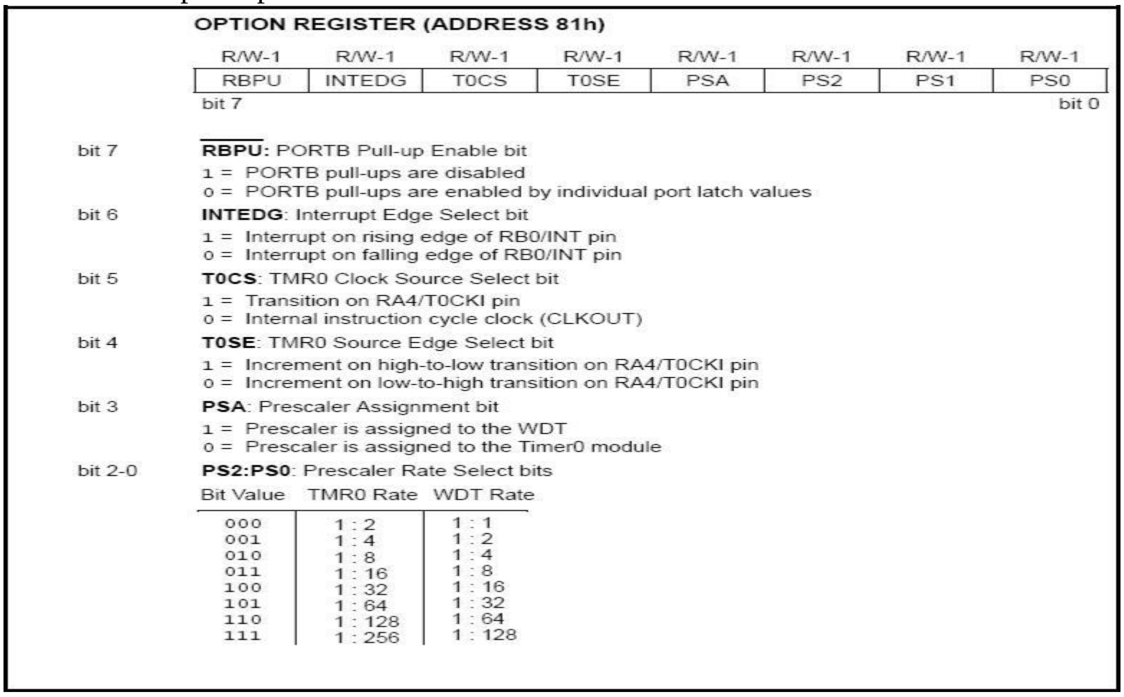
bit 2 **T0IF**: TMR0 Overflow Interrupt Flag bit
1 = TMR0 register has overflowed (must be cleared in software)
0 = TMR0 register did not overflow

bit 1 **INTF**: RB0/INT External Interrupt Flag bit
1 = The RB0/INT external interrupt occurred (must be cleared in software)
0 = The RB0/INT external interrupt did not occur

bit 0 **RBIF**: RB Port Change Interrupt Flag bit
1 = At least one of the RB7:RB4 pins changed state (must be cleared in software)
0 = None of the RB7:RB4 pins have changed state

1) **OPTION REGISTER**

The OPTION register is a readable and writable register which contains various control bits to configure the TMR0/WDT prescaler, the external INT interrupt,TMR0, and the weak pull-ups on PORTB.



OPTION REGISTER (ADDRESS 81h)

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

bit 7 **$\overline{\text{RBPU}}$:** PORTB Pull-up Enable bit
1 = PORTB pull-ups are disabled
0 = PORTB pull-ups are enabled by individual port latch values

bit 6 **INTEDG**: Interrupt Edge Select bit
1 = Interrupt on rising edge of RB0/INT pin
0 = Interrupt on falling edge of RB0/INT pin

bit 5 **T0CS**: TMR0 Clock Source Select bit
1 = Transition on RA4/T0CKI pin
0 = Internal instruction cycle clock (CLKOUT)

bit 4 **T0SE**: TMR0 Source Edge Select bit
1 = Increment on high-to-low transition on RA4/T0CKI pin
0 = Increment on low-to-high transition on RA4/T0CKI pin

bit 3 **PSA**: Prescaler Assignment bit
1 = Prescaler is assigned to the WDT
0 = Prescaler is assigned to the Timer0 module

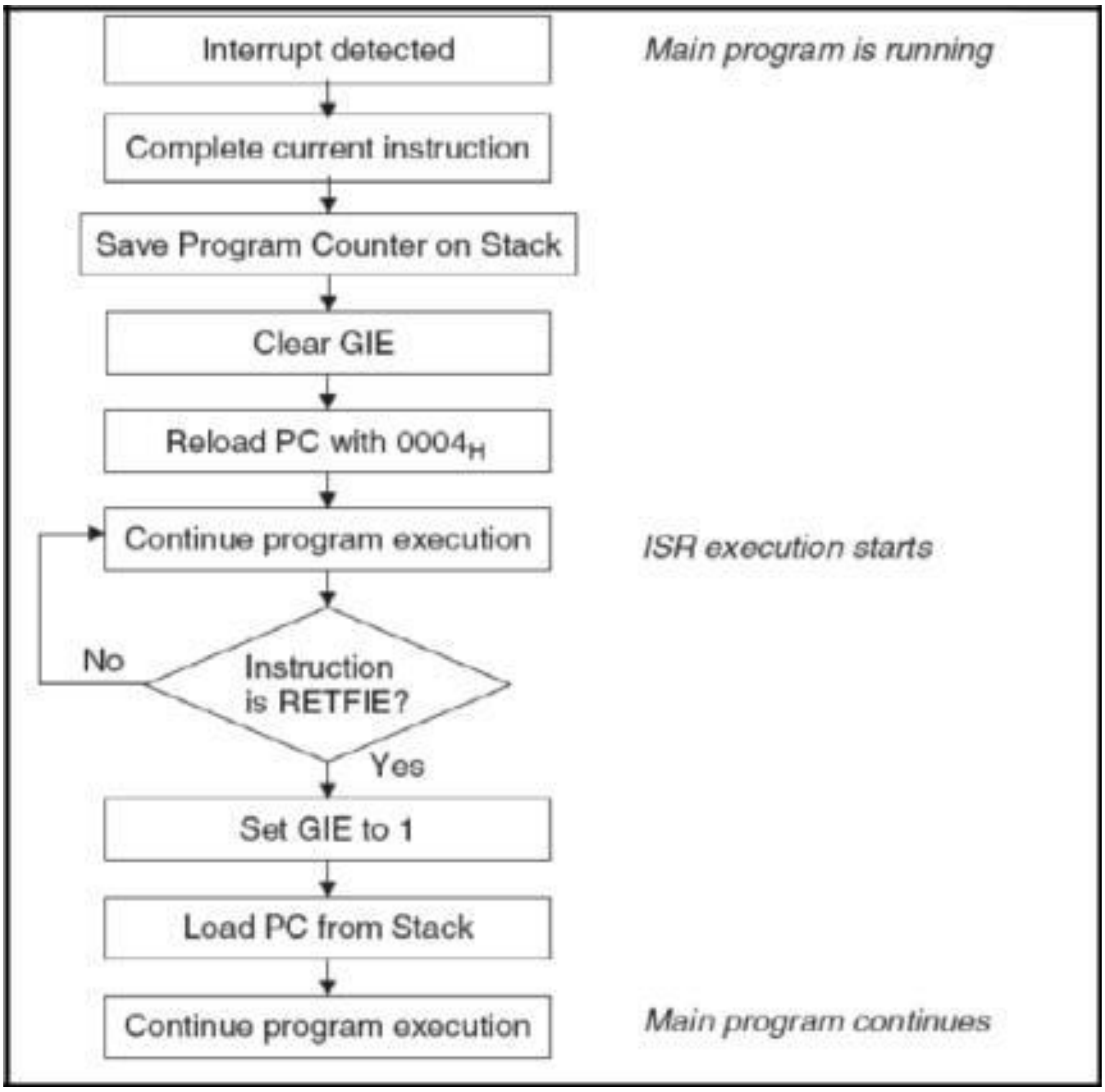
bit 2-0 **PS2:PS0**: Prescaler Rate Select bits

Bit Value	TMR0 Rate	WDT Rate
000	1 : 2	1 : 1
001	1 : 4	1 : 2
010	1 : 8	1 : 4
011	1 : 16	1 : 8
100	1 : 32	1 : 16
101	1 : 64	1 : 32
110	1 : 128	1 : 64
111	1 : 256	1 : 128

How to use interrupts?

1) Start the interrupt service routine at 0x0004
2) Clear the flag of the used interrupt in the INTCON register (if it is not cleared on reset)
3) Enable the corresponding interrupt by setting its bit in INTCON register
4) Enable global interrupts by setting the GIE bit
5) End the interrupt subroutine with **RETFIE**instruction to resume program execution

Interrupt Operation:

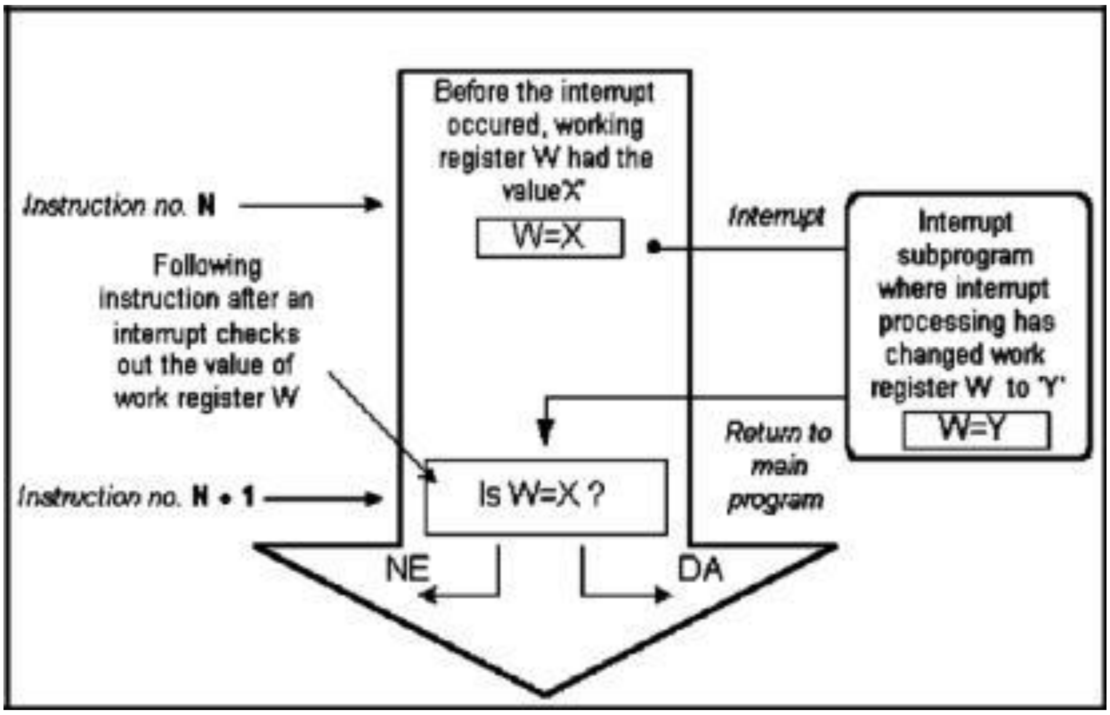


Context Saving;

What if the main program is to preserve the W register and interrupt uses it?

- ✓ Save it temporarily in memory at the beginning of the ISR
  MOVWFTEMP ; push
- ✓ Restore the value at the end of ISR
  MOVFTEMP, W ; pop

Example Error if Registers are not saved before Executing ISR


Before the interrupt occured, working register W had the value'X'
W=X
Instruction no. N
Following instruction after an interrupt checks out the value of work register W
Interrupt
Interrupt subprogram where interrupt processing has changed work register W to 'Y'
W=Y
Return to main program
Instruction no. N + 1
Is W=X ?
NE
DA

Multiple Interrupts;

Note that there is only one interrupt vector for all types of interrupts, In other words, regardless of the interrupt type, the microcontroller will start executing from location 0x0004 on any interrupt.

How to determine the source of interrupt?

Check the interrupt flag bits in the INTCON register at the beginning of the interrupt service routine to determine what source of the interrupt!

Example:

```
Interrupt_SR
btfsc    INTCON,0                  ;test RBIF
goto   portb_int                   ;Port B Change routine
btfsc INTCON,1                     ;test INTF
goto   ext_int                     ;external interrupt routine
btfsc   INTCON,2                   ;test T0IF
goto   timer_int                   ;timer overflow routine
btfsc  EECON1,4                    ;test EEPROM write complete flag
goto   eeprom_int                  ;EEPROM write complete routine
```

Hashemite University
Faculty of Engineering and Technology
Computer Engineering Department

Experiment 4
Controlling the LCD

1.1 Objectives:

1. To become familiar with HD44780 controller based LCDs and how to use them.
2. Knowing the various modes of operation of the LCD (8-bit/4-bit interface, 2-lines/1-line, CG-ROM).
3. Distinguishing between the commands for the instruction register and data register.
4. Stressing software and hardware co-design techniques by using the ***Proteus IDE*** package to simulate the LCD.

1.2 Pre-lab Preparation:

- Read the experiment thoroughly and make a real effort to build the pre-lab circuit.

1.3 Equipments:

- Personal computer with ***Proteus*** simulator installed on it.

1.4 Theoretical background:

What is an LCD?

A ***L***iquid ***C***rystal ***D***isplays (LCD) is a thin, flat display device made up of any number of color or monochrome pixels arrayed in front of a light source or reflector. It is often utilized in battery-powered electronic devices because it uses very small amounts of electric power.

Figure 1: A typical LCD module

LCDs have the ability to display numbers, letters, words and a variety of symbols. This experiment teaches you about LCDs which are based upon the Hitachi HD44780 controller chipset. LCDs come in different shapes and sizes with 8, 16, 20, 24, 32, and 40 characters as standard in 1, 2 and 4–line versions.

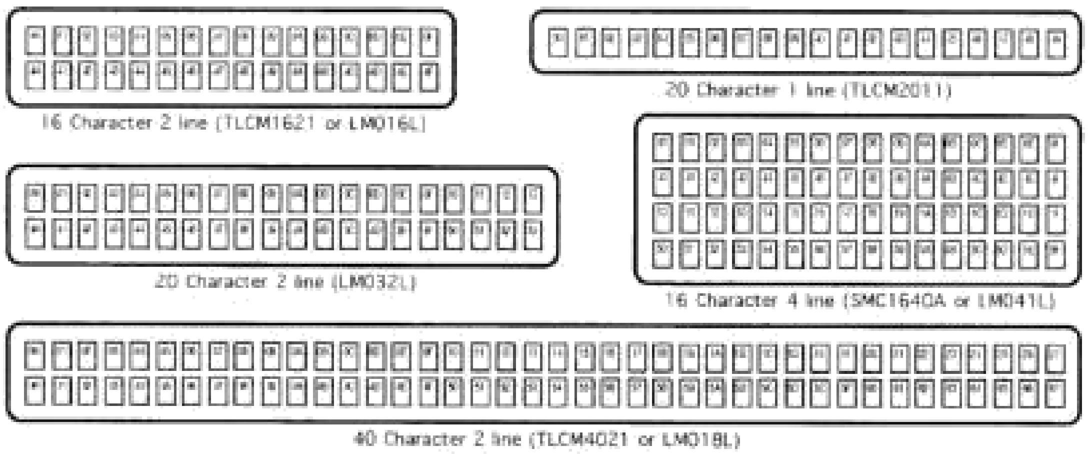
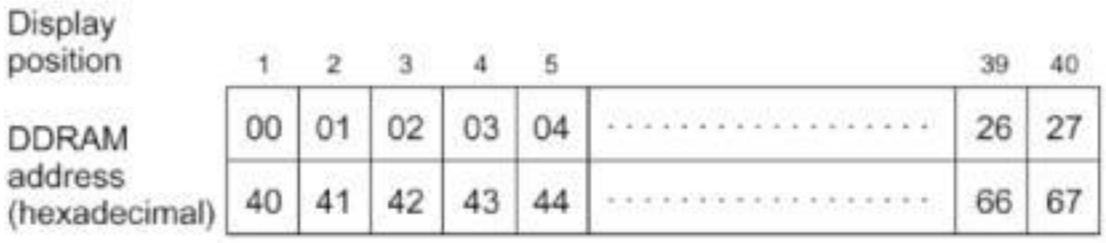

16 Character 2 line (TLCM1621 or LM016L)
20 Character 1 line (TLCM2011)
20 Character 2 line (LM032L)
16 Character 4 line (SMC1640A or LM041L)
40 Character 2 line (TLCM4021 or LM018L)

Figure 2: Different LCD modules shapes and sizes



Display position	1	2	3	4	5		39	40
DDRAM address (hexadecimal)	00	01	02	03	04	....	26	27
	40	41	42	43	44	....	66	67

Figure 3: Display address assignments for HD44780 controller based LCDs

LCD I/O

Most LCD modules conform to a standard interface specification. A 14-pin access is provided having eight data lines, three control lines and three power lines as shown below. Some LCD modules have 16 pins where the two additional pins are typically used for backlight purposes

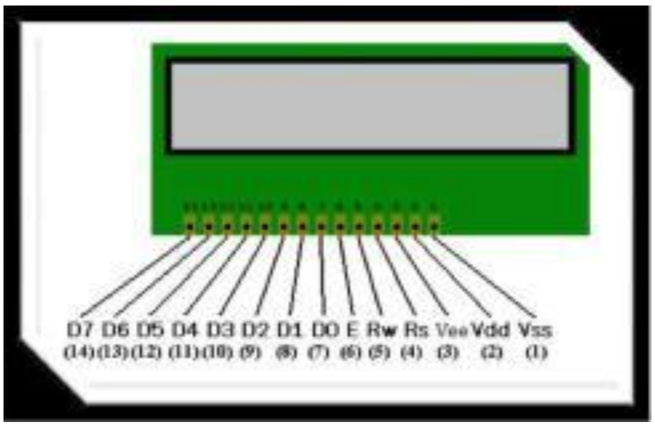
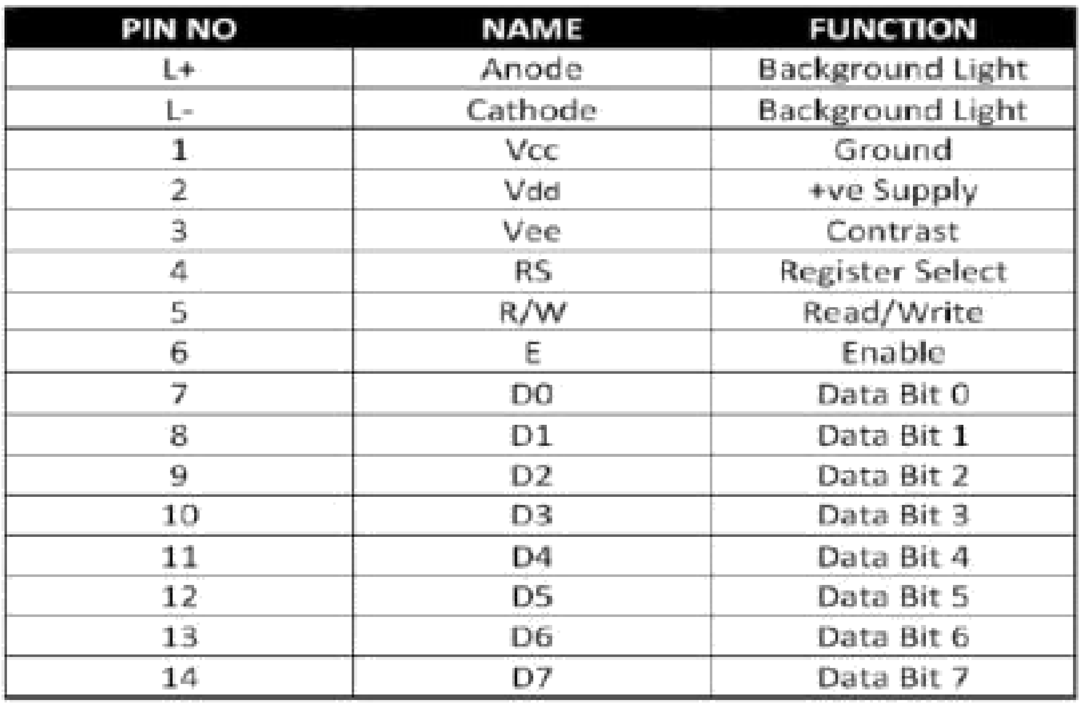


Figure 4: LCD pin-out

Powering up the LCD requires connecting three lines: one for the positive power ***Vdd*** (usually +5V), one for negative power (or ground) ***Vss***. The ***Vee*** pin is usually connected to a potentiometer which is used to vary the contrast of the LCD display. We will connect this pin to the GND.

As you can see from Figure 4, the LCD connects to the microcontroller through three control lines: RS, RW and E, and through eight data lines D0-D7.
With 16-pin LCDs, you can use the L+ and L- pins to turn the backlight (BL) on/off.



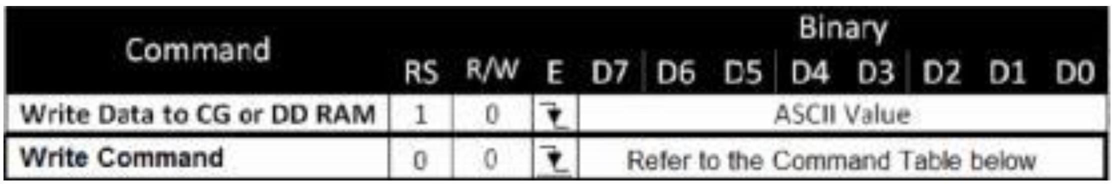
PIN NO	NAME	FUNCTION
L+	Anode	Background Light
L-	Cathode	Background Light
1	Vcc	Ground
2	Vdd	+ve Supply
3	Vee	Contrast
4	RS	Register Select
5	R/W	Read/Write
6	E	Enable
7	D0	Data Bit 0
8	D1	Data Bit 1
9	D2	Data Bit 2
10	D3	Data Bit 3
11	D4	Data Bit 4
12	D5	Data Bit 5
13	D6	Data Bit 6
14	D7	Data Bit 7

Figure 5: LCD pin-out details

When powered up, the LCD display should show a series of dark squares. These cells are actually in their off state. When power is applied, the LCD is reset; therefore we should issue a command to set it on. Moreover, you should issue some commands which configure the LCD. (See the table which lists all possible configurations and the explanation to each field)

Sending Commands/Data to the LCD

Using an LCD is a simple procedure once you learn it. Simply you will place a value on the LCD lines D0-D7 (this value might be an ASCII value (character to be displayed), or another hexadecimal value corresponding to a certain command). So how will the LCD differentiate if this value on D0-D7 is corresponding to data or command? Observe the figure below, as you might see the only difference is in the RS signal (**R**egister **S**elect), this is the only way for the LCD controller to know whether it is dealing with a character or a command!



Command	Binary										
	RS	R/W	E	D7	D6	D5	D4	D3	D2	D1	D0
Write Data to CG or DD RAM	1	0	↓	ASCII Value							
Write Command	0	0	↓	Refer to the Command Table below							

Figure 7: Necessary control signals for Data/Commands

Steps to send character to LCD

1. Place the ASCII character on the D0-D7 lines
2. Register Select (RS) = 1 to send characters
3. "Enable" Pulse (Set High – Delay – Set Low)
4. Delay to give LCD the time needed to display the character

Steps to send a command to LCD

1. Place the command on the D0-D7 lines
2. Register Select (RS) = 0 to send commands
3. "Enable" Pulse (Set High – Delay – Set Low)
4. Delay to give LCD the time needed to carry out the command

Displaying Characters

All English letters and numbers (as well as special characters, Japanese and Greek letters) are built in the LCD module in such a way that it **conforms to the ASCII standard**. In order to display a character, you only need to send its ASCII code to the LCD which it uses to display the character.

Notice that from column 1 to D, the character resolution is 5 characters wide x 7 characters high (5x7) (column 0 is a special case, it is 5x8, but considered as 5x7, more on this later) whereas the character resolution of columns E and F is 5 characters wide x 10 characters high (5x10). We should change the resolution if we are to use characters from different resolution columns, this can be done using a command discussed later.

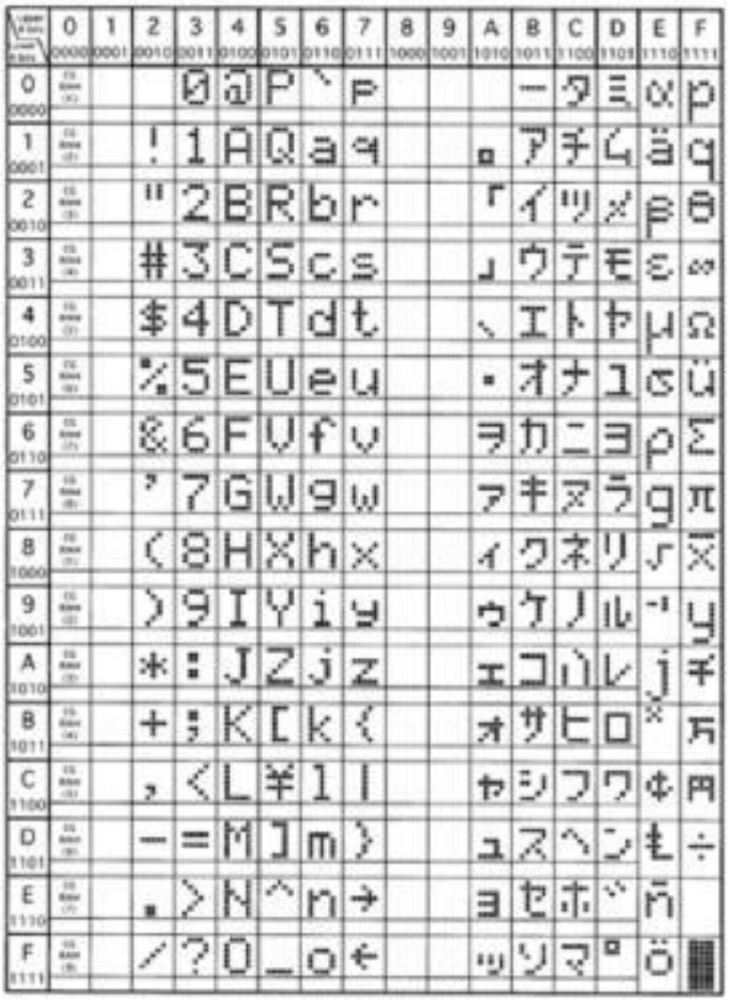
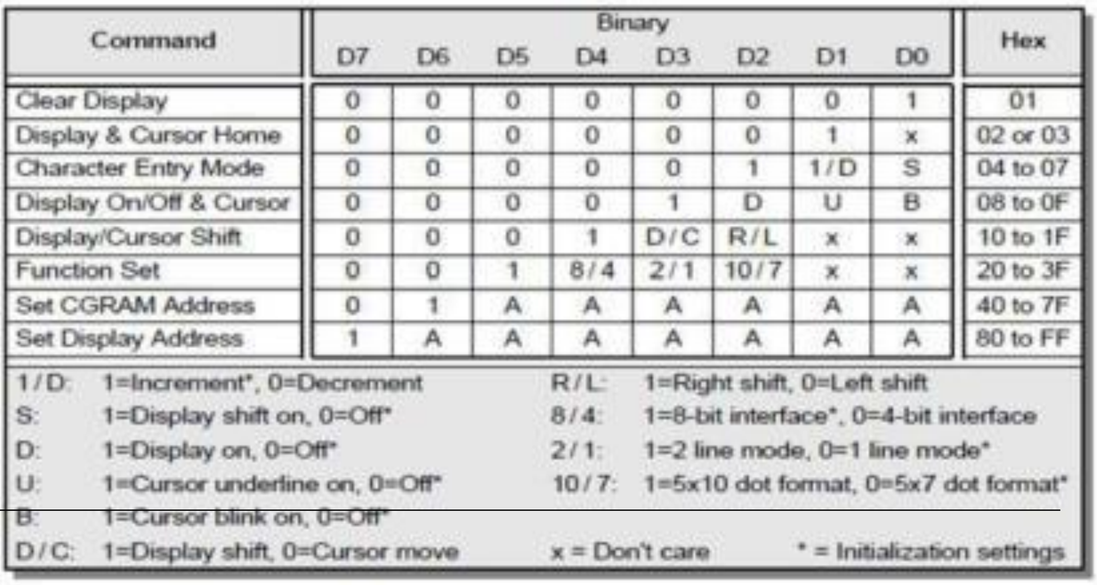


Figure 8: LCD Characters Map

Explaining the commands and their parameters in the LCD command table:



Command	Binary								Hex
	D7	D6	D5	D4	D3	D2	D1	D0	
Clear Display	0	0	0	0	0	0	0	1	01
Display & Cursor Home	0	0	0	0	0	0	1	x	02 or 03
Character Entry Mode	0	0	0	0	0	1	1 / D	S	04 to 07
Display On/Off & Cursor	0	0	0	0	1	D	U	B	08 to 0F
Display/Cursor Shift	0	0	0	1	D / C	R / L	x	x	10 to 1F
Function Set	0	0	1	8 / 4	2 / 1	10 / 7	x	x	20 to 3F
Set CGRAM Address	0	1	A	A	A	A	A	A	40 to 7F
Set Display Address	1	A	A	A	A	A	A	A	80 to FF

1 / D:	1=Increment*, 0=Decrement	R / L:	1=Right shift, 0=Left shift
S:	1=Display shift on, 0=Off*	8 / 4:	1=8-bit interface*, 0=4-bit interface
D:	1=Display on, 0=Off*	2 / 1:	1=2 line mode, 0=1 line mode*
U:	1=Cursor underline on, 0=Off*	10 / 7:	1=5x10 dot format, 0=5x7 dot format*
B:	1=Cursor blink on, 0=Off*		
D / C:	1=Display shift, 0=Cursor move	x = Don't care	* = Initialization settings

Clear Display
Clear the LCD display; however the cursor will remain at its last position, so any future character writes will start from the last location, to reset the cursor position use the Display and Cursor Home command.

Display and Cursor Home
Resets cursor location to position 00 of the LCD screen (Figure 3), future writes will start at the first location of the first line.

Character Entry Mode
This command has two parameters 1/D and S:
1/D: By default, the cursor is automatically set to move from location 00 to 01 and so on (Increment mode). Suppose now that you are to write from right to left (as in the Arabic language), then you have to set the cursor to the Decrement mode.
S: Accompanies the D/C parameter, explained below

Display On/OFF and Cursor
This command has three parameters:
D: Turns on the display (when you see the black dots on the LCD, it means that it is POWERED on, but not yet ready to operate), this command activates the LCD in order to be ready to use.
U: This displays the cursor (in the form of a horizontal line at the bottom of the character) when it is high and turns the cursor off when it is low
B: If the underline cursor option is enabled, this will blink the cursor if high.

Display/Cursor Shift
All LCDs based on the HD44780 format - whatever their actual physical size is - are internally built in to be 40 characters x 2 lines with the upper row having the display addresses 0-27H (27H = 39D → 0-39 = 40 Characters!!) and the lower row from 40H -67H. Now suppose you bought an LCD with the physical size of 20 char. x 2 lines, when you start writing to the LCD and the cursor reaches locations 20D, 21 D, and 22 D …, you will not see them BUT don't worry, they are not lost. They were written in their respective locations but you could not see them because your bought LCD is 20 **visible** Characters wide from the outside and 40 from the inside. All you have to do is shift the display. So all you do is
1. Determine the direction of the shift (R/L)
2. Issue the shift Command D/C
R/L: Determines the direction of the shift, this might be useful if you are writing Arabic characters …
D/C: if this bit has a value of 0, the display is not shifted and the cursor moves the same way it was, if the its value is logic high, the display is shifted once, you might need to issue this command multiple times in order to shift the display by multiple locations!

Function Set
This command has three parameters:
8/4: Eight/Four bits mode
8 – Bit interface: you send the whole command/character (8 bits) in one stage to the D0-D7 lines
4 – Bit interface: you send the command/character in two stages as nibbles to D4-D7 lines.

When to use the 4-bit mode?
1. Interfacing LCD with older devices which have 4-bit wide I/O Bus
2. You don't have enough I/O pins remaining, or you want to conserve the I/O pins for other HW

2/1: Line mode, determines whether you want to use the upper line of the LCD or both lines

10/7: Dot format, based on the LCD built-in characters table, note the following:
* 5x7 format (Default) is used whenever you use the characters found in columns 1 to D
* 5x7 format is also used whenever you use the built in characters in CG-RAM ***(EVEN THOUGH THE CG-RAM CHARACTERs ARE 5X8!!!)***
* 5x10 format is only used when displaying the characters found in columns **E** and **F**

Set Display Address command

Syntax: 1AAAAAAA

This command allows you to move the cursor to whichever location you want, suppose you want to start writing in the middle of the display (assuming the ***visible*** width of the LCD screen is 20), then from Figure 2 you will observe that location 06 is approximately in the middle so you replace the A's with 06: 1*AAAAAAA* →1*0000110*→0x86

Moreover, suppose you wish to move to the second line which starts at location 40, same as above 1AAAAAAA→ 1*1000000* → 0xC0

Set CG-RAM Address command

Syntax: 01AAAAAA

If you give a closer look at Figure 8, you will clearly see that the table only contains English and Japanese characters, numbers, symbols as well as special characters! Suppose now that you would like to display a character not found in the built-in table of the LCD (i.e. an Arabic Character). In this case we will have to use what is called the CG-RAM (Character Generation RAM), which is a reserved memory space in which you could draw your own characters and later display them.

Observe column one in Figure 8, the locations inside this column are reserved for the CG-RAM. Even though you see 16 locations (0 to F), you only have the possibility to use the first 8 locations 0 to 7 because locations 8 to F are mirrors of locations 0 – 7.

So, to organize things, in order to use our own characters we have to do the following:
1. Draw and store our own defined characters in CG-RAM
2. Display the characters on the LCD screen as if it were any of the other characters in the table

Drawing and storing our own defined characters in CG-RAM

As stated earlier, we have eight locations to store our characters in. So how do we choose which location out of these to start drawing and building our characters in?

The answer is quite simple; follow this rule as stated in the datasheet of the HD44780 controller:
1. To write (build/store a character in location 00 (crossing of the row and column)), you send the CG-RAM address command as follows: 01*AAAAAA* → 01*000000* → 0x40
2. However, to write in any location from 01 to 07, you have to skip eight locations (WHY?)

So the CG-RAM address command will send **0x48** (to store a character in location 1**),** **0x50** (to store a character in location 2) and so on...

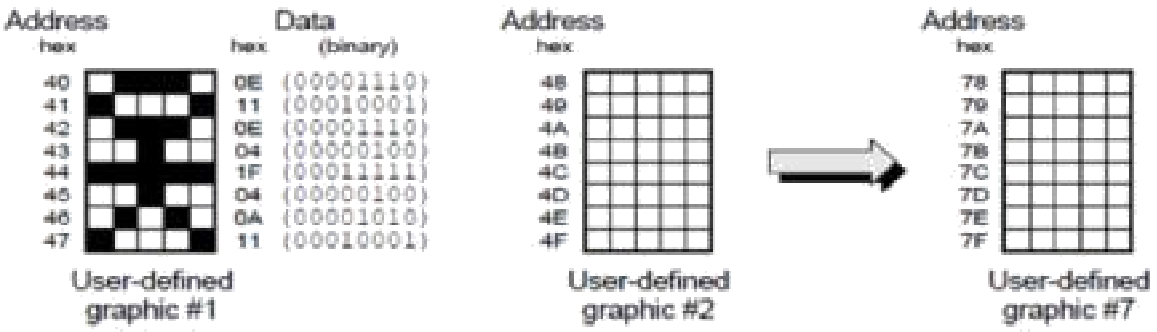
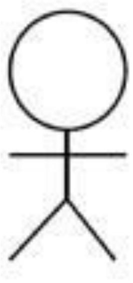
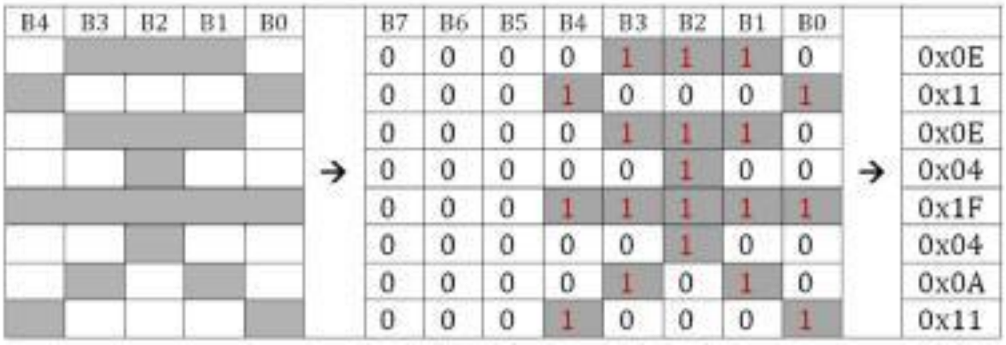


Figure10 Showing how the CGRAM addresses correspond to individual pixels.

So up to this point we have defined **where** to write our characters but not how to build them! This is the fun part, draw a 5x8 Grid and start drawing your character inside, then replace each shaded cell with one and not shaded ones with zero. Append three zeros to the left (B5-B7) and finally transform the sequence into hexadecimal format. This is the sequence which you will fill in the CG-RAM SEQUENTIALLY once you have set the CG-RAM Address before.





B4	B3	B2	B1	B0		B7	B6	B5	B4	B3	B2	B1	B0		
	▨	▨	▨			0	0	0	0	1	1	1	0		0x0E
▨				▨		0	0	0	1	0	0	0	1		0x11
	▨	▨	▨			0	0	0	0	1	1	1	0		0x0E
		▨			→	0	0	0	0	0	1	0	0	→	0x04
▨	▨	▨	▨	▨		0	0	0	1	1	1	1	1		0x1F
		▨				0	0	0	0	0	1	0	0		0x04
	▨		▨			0	0	0	0	1	0	1	0		0x0A
▨				▨		0	0	0	1	0	0	0	1		0x11

Figure 11: calculated values to draw a Stick Man

Hashemite University
Faculty of Engineering and Technology
Computer Engineering Department

Experiment 5
Interfacing the Keypad

1.1 Objectives:

1. To become familiar with keypad Interfacing and usage
2. Stressing software and hardware co-design techniques by using the ***Proteus* IDE** package to simulate a user built keypad

1.2 Pre-lab Preparation:

- Read the experiment thoroughly and make a real effort to build the pre-lab circuit.

1.3 Equipments:

- Personal computer with ***Proteus*** simulator installed on it.

1.4 Theoretical background:

Keypads are essentially large switch arrays which allow data entries (numeric or alphanumeric) into systems. Keypads are widely used in everyday applications such as burglar alarms, cell phones and photocopiers. Keypads come in different shapes and sizes with 4x3, 4x4 buttons as common examples. It is not practical to connect each button in the keypad to its own port input as we previously did with switch and push buttons; therefore keypads are normally constructed in a matrix format. An (n x m) array can be read by (n + m) port bits.

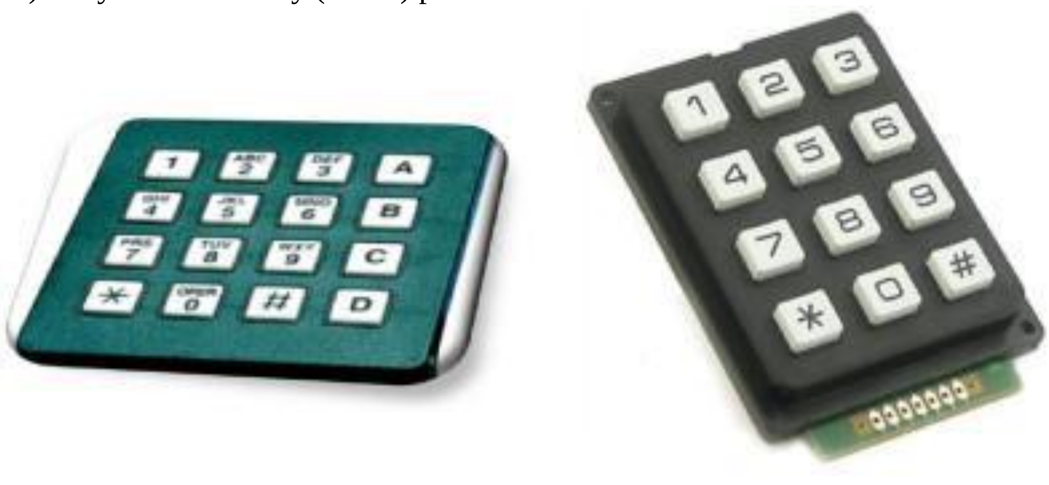


Figure 1: A 4x4 keypad (left) and a 4x3 keypad (right), notice the 7 connections of the keypad (4+3)

General Keypad Operation

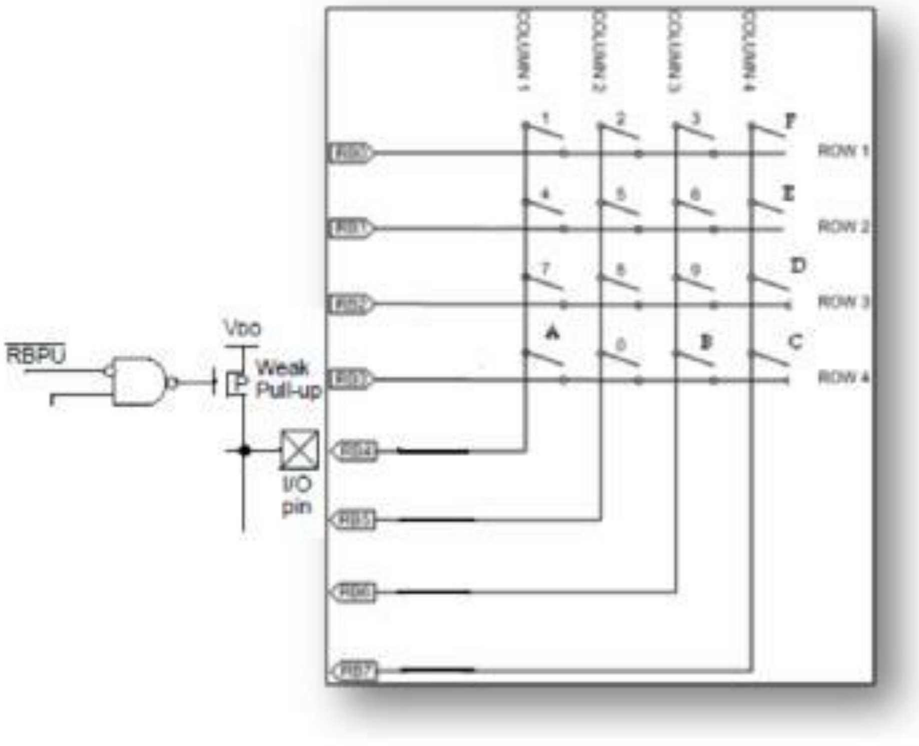
In general, a keypad is interfaced in a way such that initially if no key is pressed you will read a certain logic level and when you press a button a signal with the negative of the original level will be read. You have two cases:

1. Fix the initial button state to be read as logic 1 (through the use of pull-up resistors), when you press a button you will read logic 0.
2. Fix the initial button state to be read as logic 0 (through the use of pull-down resistors), when you press a button you will read logic 1.

- Pull-up and pull-down resistors are used to limit the amount of current and protect the circuit. (not to read a floating state)

- Pull-up and pull-down resistors are normally connected externally, BUT you can make use of the internal pull-up resistors found in Microchip's PIC devices such as those in implemented in PORTB. In this experiment we will use the internal pull-up resistors.

Whether you use internal or external pull-up resistors the keypad will operate in the same way.



Technique

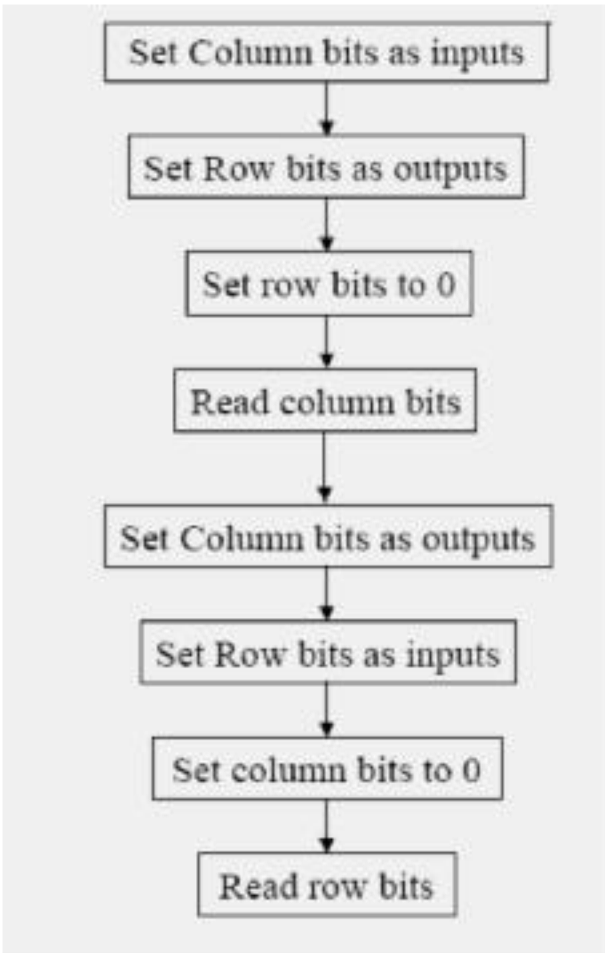
**It does not matter whether you start scanning rows or columns first it depends on your connections; the basic idea is the logic of the scanning technique is the same.*

First the row bits are set to output, with the column bits as input. The output rows are set to logic 0. If no button is pressed all column line inputs will be read as logic 1 due to the action of the pull-up resistors. If, however, a button is pressed then its corresponding switch will connect column and row lines, and the corresponding column line will be read as low.

To detect this logic transition from high to low (that is to know whether a key has been pressed or not), we have to either:

1. Keep pulling the inputs (columns) continuously until 0 is detected.
2. Make use of the interrupt (Here PORTB interrupt- on change will be beneficial)

Yet still, we have identified the column in which the key was pressed but not the button itself. So what we do now is save the column and repeats the same procedure above with the following minor modification:
Secondly the column bits are set to output, with the row bits as input. The output columns are set to logic 0. Since the button is still pressed then its corresponding switch is still connecting column and row lines, and the corresponding row line will be read as low. If, however, the button is released all row line inputs will be read as logic 1 due to the action of the pull-up resistors.
Now we have identified the row.

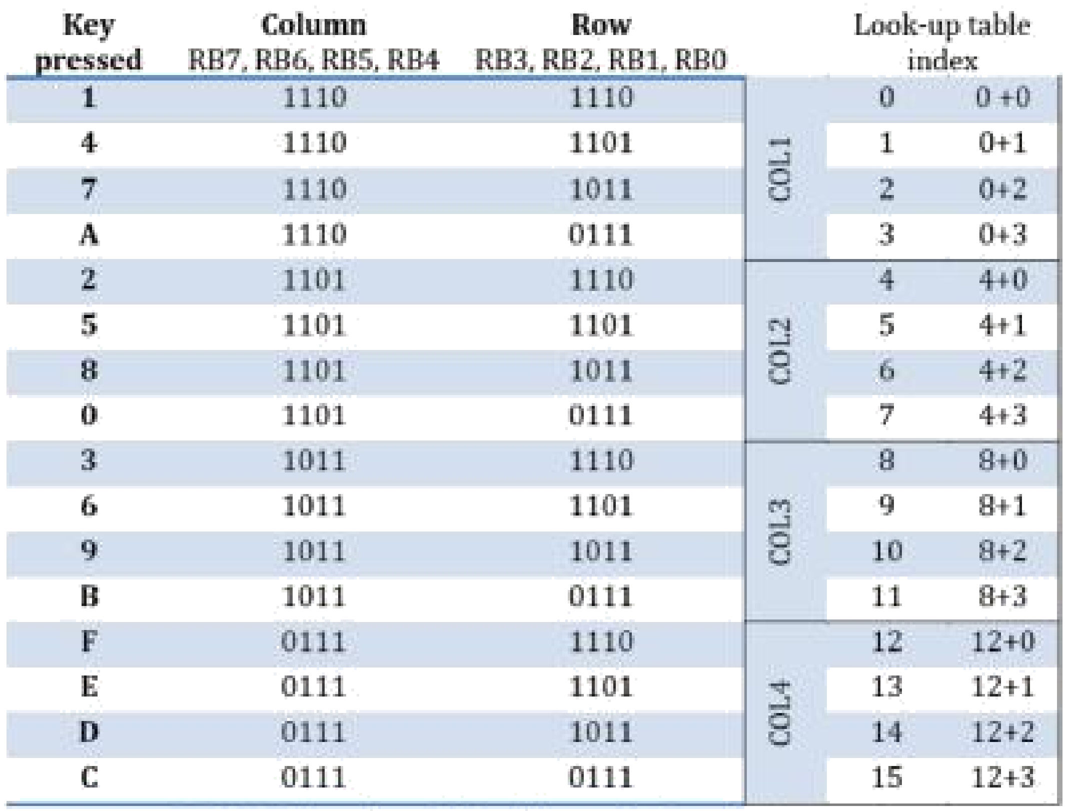


Example

Suppose we connect the columns to PORTB 4-7 as input and the rows to PORTB 0-3 as output (with the value of 0). If we continuously read the inputs they will always be read as 1 because of the internal pull-up resistors on PORTB. If one presses number “7”, this will make us read logic 0 on RB4 (identified that we have pressed a button in the first column).
Now let us exchange the inputs for the outputs that is we connect the rows to PORTB 4-7 as input and the columns to PORTB 0-3 as output (with the value of 0).
If we read the input we will find that RB2 is 0. Now we have identified the location of the pressed button and ready to process what it means.

So what comes next?

The above scanning technique let us know the position of the pressed button (in terms of its row/column intersection) but not the value corresponding to the button. So obviously the next step is to use the location to retrieve the desired value.



Key pressed	Column RB7, RB6, RB5, RB4	Row RB3, RB2, RB1, RB0		Look-up table index	
1	1110	1110	COL1	0	0 +0
4	1110	1101		1	0+1
7	1110	1011		2	0+2
A	1110	0111		3	0+3
2	1101	1110	COL2	4	4+0
5	1101	1101		5	4+1
8	1101	1011		6	4+2
0	1101	0111		7	4+3
3	1011	1110	COL3	8	8+0
6	1011	1101		9	8+1
9	1011	1011		10	8+2
B	1011	0111		11	8+3
F	0111	1110	COL4	12	12+0
E	0111	1101		13	12+1
D	0111	1011		14	12+2
C	0111	0111		15	12+3

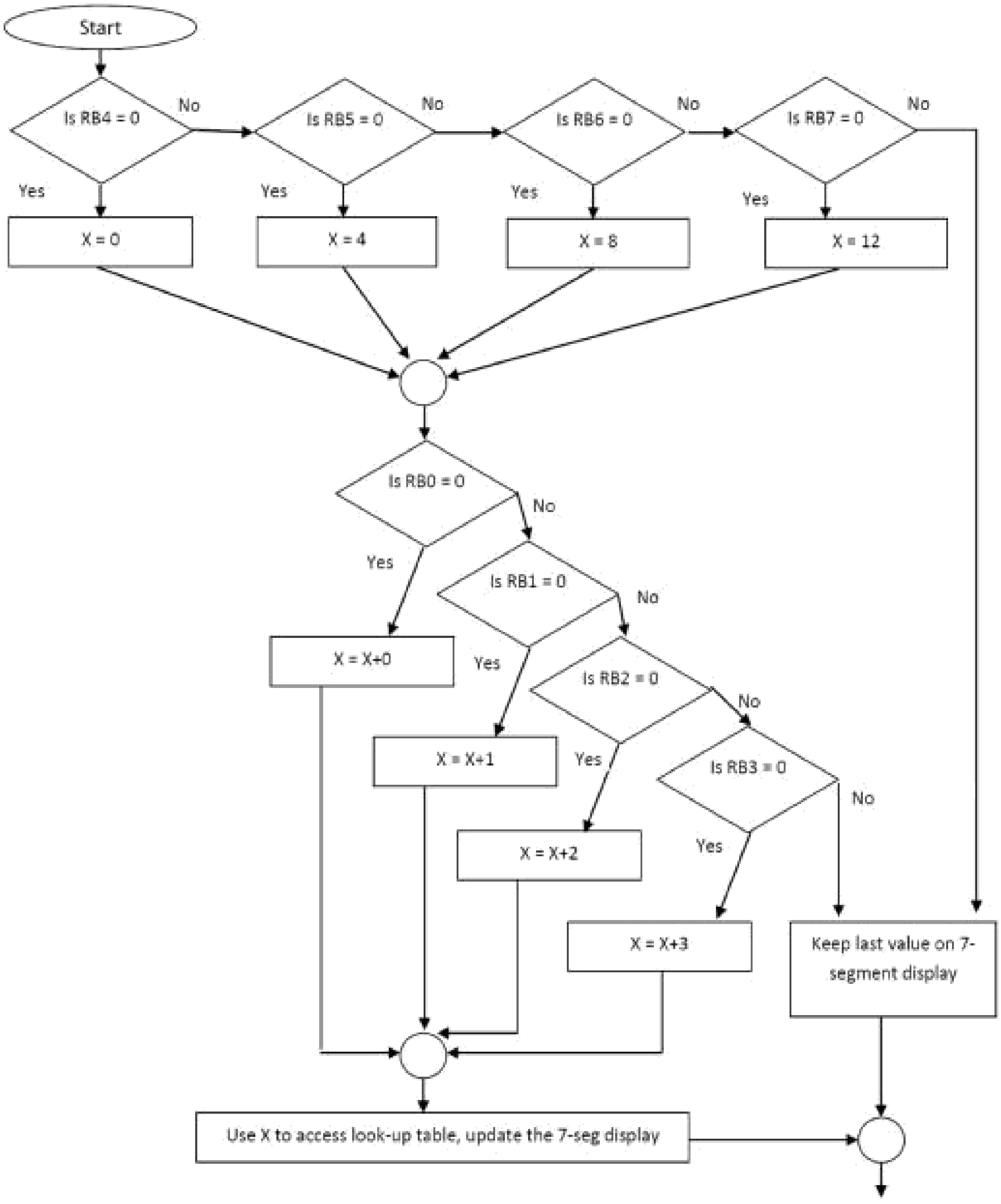
Table 1 - The values which will be read when a key is pressed

Most often, you will need to display the number on a 7-segment display, LCD or use it in binary calculations. Therefore, it is natural to build a look-up table with the 7-segment representations, ASCII code or binary equivalent and use the location pattern which you saved (as in the table above) as an index to the look-up table.

The way one organizes the look-up table entries differs from one person to another, therefore there is no specific way to translate the locations to their corresponding values. One might use a series of **btfsc** or **btfss** instructions, or deduce a relationship and use mathematical operations or a combination of both.

What have we done in this experiment?
Study the following flowchart which is based on the table above.
If the key pressed is in column 1, then X might take the values 0, 1, 2, 3
If the key pressed is in column 2, then X might take the values 4, 5, 6, 7
If the key pressed is in column 3, then X might take the values 8, 9, 10, 11
If the key pressed is in column 4, then X might take the values 12, 13, 14, 15

These values will be added to PCL to retrieve values from the look-up table.
See figure below


Start
Is RB4 = 0
No
Yes
X = 0
Is RB5 = 0
No
Yes
X = 4
Is RB6 = 0
No
Yes
X = 8
Is RB7 = 0
No
Yes
X = 12
Is RB0 = 0
No
Yes
X = X+0
Is RB1 = 0
No
Yes
X = X+1
Is RB2 = 0
No
Yes
X = X+2
Is RB3 = 0
No
Yes
X = X+3
Keep last value on 7-segment display
Use X to access look-up table, update the 7-seg display

Hashemite University
Faculty of Engineering and Technology
Computer Engineering Department

Experiment 6
Introduction to Android

1.1 Objectives:

1. Introduce the Android Operating System and its basic components of Android.
2. Introduce the basic ingredients of an Android Application.
3. Provide a detailed description Environment setup process.
4. Creating simple HelloWorld application.

1.2 Pre-lab Preparation:

- Read the experiment thoroughly and make a real effort to build the pre-lab circuit.

1.3 Equipments:

- Personal computer with ***Eclipse IDE and Android SDK*** installed on it.

2.1 Theoretical Background:

Android is an open source Linux-based Operating System for mobile devices developed by the Open Handset Alliance, led by Google, and other companies, the first beta version of the Android Software Development Kit (**SDK**) was released by Google in 2007 where as the first commercial version, Android 1.0, was released in September 2008.

The developers need to write their codes for Android in general, and their applications should be able to run on different devices powered by Android. Android applications are developed in the Java language using the Android Software Development Kit (**SDK**) also it can be packaged easily and sold out either through a store such as **Google Play** or the **Amazon Appstore**.

2.2 Features of Android:

Android is a powerful operating system competing with Apple 4GS and supports great features. Few of them are listed below:

- **Connectivity**: GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth, Wi-Fi, LTE, NFC and WiMAX.
- **Storage**: SQLite, a lightweight relational database, is used for data storage purposes.
- **Multi-touch**: Android has native support for multi-touch which was initially made available in handsets such as the HTC Hero.

- **Multi-tasking**: User can jump from one task to another and same time various applications can run simultaneously.
- **Media support**: H.263, H.264, MPEG-4 SP, AMR, AMR-WB, AAC, HE-AAC, AAC 5.1, MP3, MIDI, OggVorbis, WAV, JPEG, PNG, GIF, and BMP.
- **Web browser**: Based on the open-source WebKit layout engine, coupled with Chrome's V8 JavaScript engine supporting HTML5 and CSS3.
- **Resizable widgets** :Widgets are resizable, so users can expand them to show more content or shrink them to save space
- **Multi-Language**: Supports single direction and bi-directional text.
- **Wi-Fi Direct**: A technology that lets apps discover and pair directly, over a high-bandwidth peer-to-peer connection.
- **GCM:** Google Cloud Messaging (GCM) is a service that lets developers send short message data to their users on Android devices, without needing a proprietary sync solution.

3. Environment Setup:

All the required tools to develop Android applications are freely available and can be downloaded from the Web. Following is the list of software's you will need before you start your Android application programming.

- Java JDK
- Android SDK
- Eclipse IDE for Java Developers (optional you can use any IDE you want such as Android studio or Inetiliji…etc.).
- Android Development Tools (ADT) Eclipse Plugin (optional)

3.1 Setup Java Development Kit (JDK):

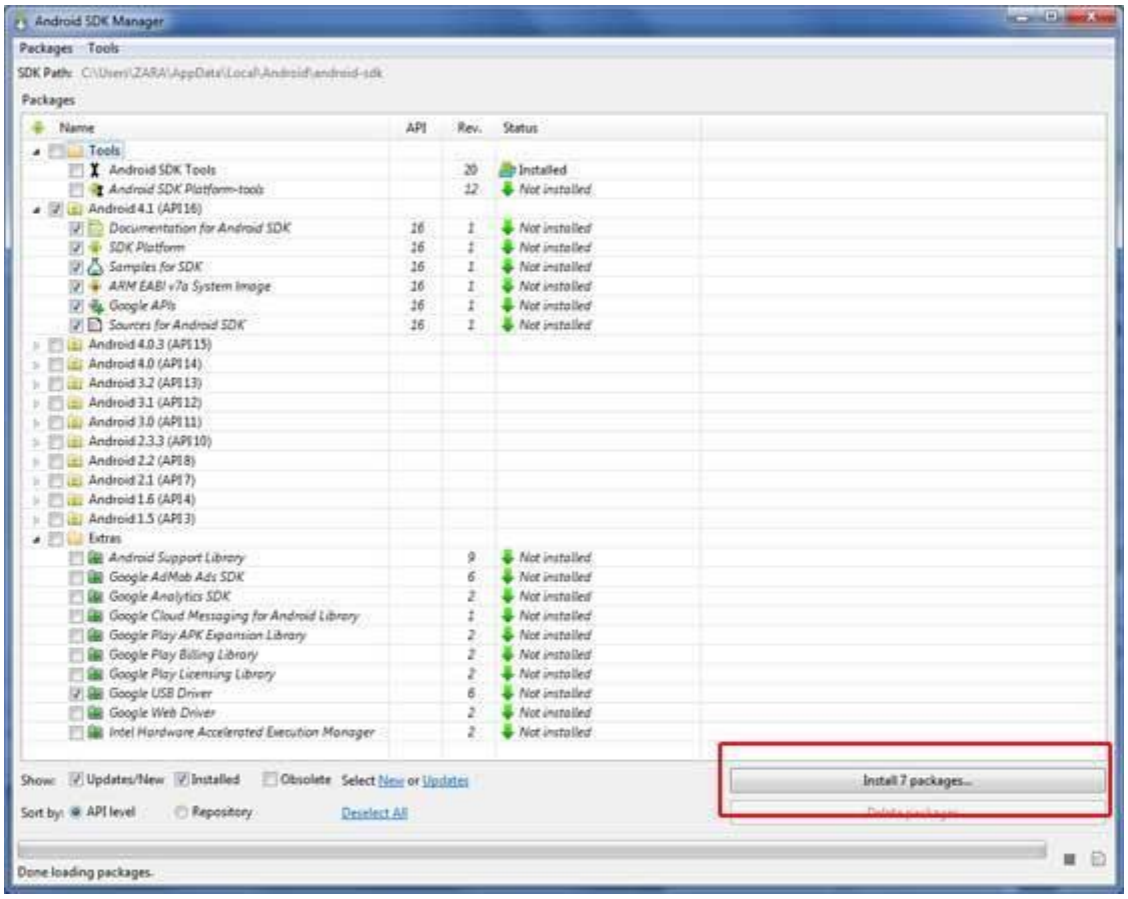
The JDK provides tools, such as the Java compiler, used by IDEs and SDKs for developing Java programs. The JDK also contains a Java Runtime Environment (JRE), which enables Java programs, such as Eclipse, to run on your system.

You can download the JDK from Oracle’s site, you can find the JDK at:
http://www.oracle.com/technetwork/java/javase/downloads/index.html.

The Downloads page will automatically detect your system and offer to download the correct version. The installer you download is an executable file. Run the executable installer file to install the JDK. After installing the JDK you may need to set PATH and JAVA_HOME environment variables to refer to the directory that contains java and javac, typically java_install_dir/bin and java_install_dir respectively.
To guarantee that everything is fine write the following command on the CMD:
javac -version

3.2 Setup Android SDK:

You can download the SDK (named installer_rXX-windows.exe) from the android official website: **http://developer.android.com/sdk/index.html**, so just download and run this exe which will launch Android SDK Tool Setup wizard to guide you through out of the installation, so just follow the instructions carefully. Finally you will have Android SDK Tools installed on your machine.
After installing the SDK launch Android SDK Manager by pressing Start> Programs > Android SDK Tools > SDK Manager, this will give you following window:


Android SDK Manager

SDK Path: C:\Users\ZARA\AppData\Local\Android\android-sdk

Packages

Name	API	Rev.	Status
☐ Tools			
☐ Android SDK Tools		20	Installed
☐ Android SDK Platform-tools		12	Not installed
☑ Android 4.1 (API16)			
☑ Documentation for Android SDK	16	1	Not installed
☑ SDK Platform	16	1	Not installed
☑ Samples for SDK	16	1	Not installed
☑ ARM EABI v7a System Image	16	1	Not installed
☑ Google APIs	16	1	Not installed
☑ Sources for Android SDK	16	1	Not installed
☐ Android 4.0.3 (API15)			
☐ Android 4.0 (API14)			
☐ Android 3.2 (API13)			
☐ Android 3.1 (API12)			
☐ Android 3.0 (API11)			
☐ Android 2.3.3 (API10)			
☐ Android 2.2 (API8)			
☐ Android 2.1 (API7)			
☐ Android 1.6 (API4)			
☐ Android 1.5 (API3)			
☐ Extras			
☐ Android Support Library		9	Not installed
☐ Google AdMob Ads SDK		6	Not installed
☐ Google Analytics SDK		2	Not installed
☐ Google Cloud Messaging for Android Library		1	Not installed
☐ Google Play APK Expansion Library		2	Not installed
☐ Google Play Billing Library		2	Not installed
☐ Google Play Licensing Library		2	Not installed
☑ Google USB Driver		6	Not installed
☐ Google Web Driver		2	Not installed
☐ Intel Hardware Accelerated Execution Manager		2	Not installed

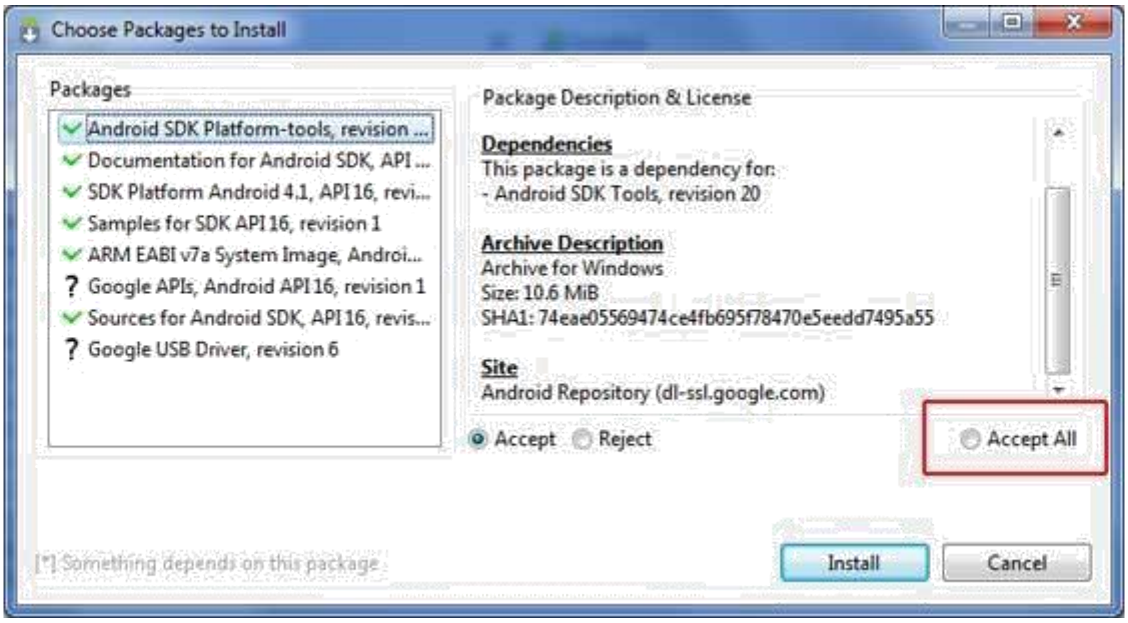
Show: ☑ Updates/New ☑ Installed ☐ Obsolete Select New or Updates

Sort by: ● API level ○ Repository Deselect All

Done loading packages.

Figure 1:SDK Manager

Once you launched SDK manager, it's the time to install other required packages. By default it will list down total 7packages to be installed, but I will suggest to de-select Documentation for Android SDK and Samples for SDK packages to reduce installation time. Next click Install 7 Packages button to proceed, which will display following dialogue box:


Choose Packages to Install

Packages
- ✓ Android SDK Platform-tools, revision ...
- ✓ Documentation for Android SDK, API ...
- ✓ SDK Platform Android 4.1, API 16, revi...
- ✓ Samples for SDK API 16, revision 1
- ✓ ARM EABI v7a System Image, Androi...
- ? Google APIs, Android API 16, revision 1
- ✓ Sources for Android SDK, API 16, revis...
- ? Google USB Driver, revision 6

Package Description & License

Dependencies
This package is a dependency for:
- Android SDK Tools, revision 20

Archive Description
Archive for Windows
Size: 10.6 MiB
SHA1: 74eae05569474ce4fb695f78470e5eedd7495a55

Site
Android Repository (dl-ssl.google.com)

[✓ Accept] Reject Accept All

[*] Something depends on this package

Figure 2: Choose Package

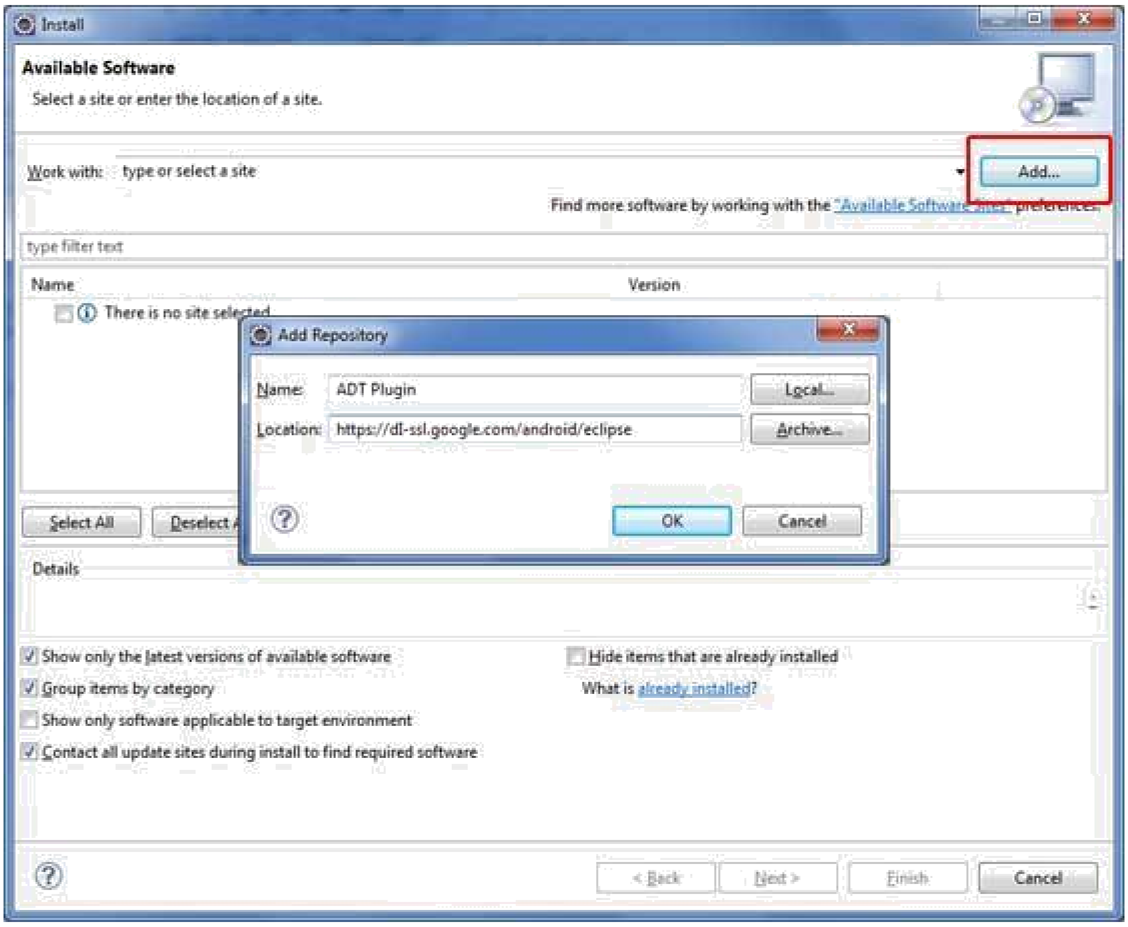
If you agree to install all the packages, select Accept All radio button and proceed by clicking Install button. Now let SDK manager do its work and you wait until all the packages are installed. It may take some time depending on your internet connection. Once all the packages are installed, you can close SDK manager using top-right cross button.

3.3 Setup Eclipse IDE:

You can download the latest Eclipse exe from:
http://www.eclipse.org/downloads/
Once you downloaded the installation, unpack the binary distribution into a convenient location.

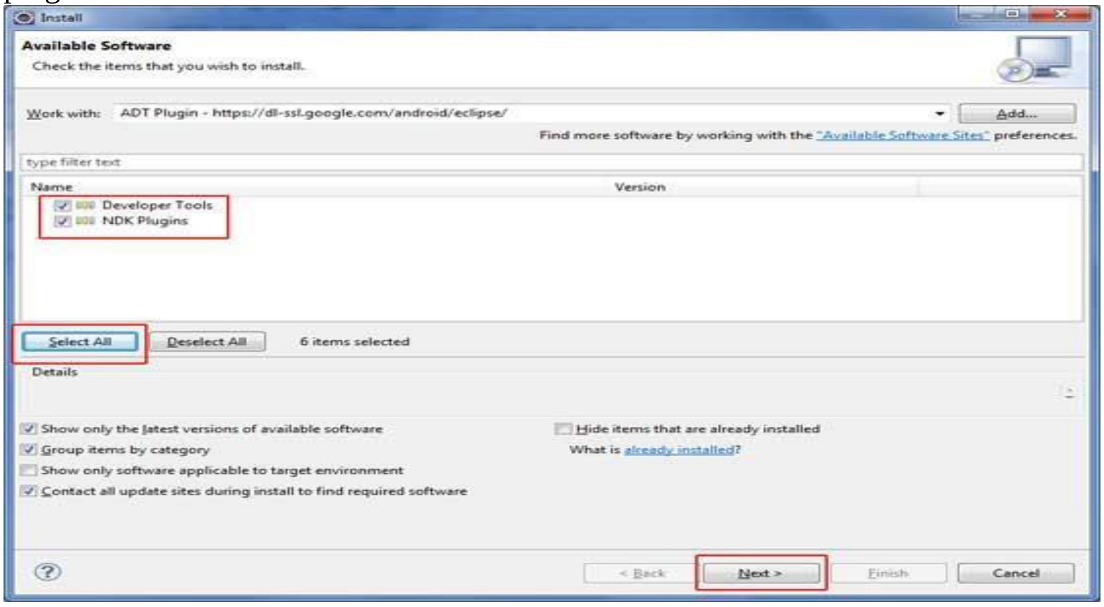
3.3 Setup Android Development Tools (ADT) Plugin:

Launch Eclipse and then, choose Help > Software Updates > Install New Software. This will display the following dialogue box:


Install
Available Software
Select a site or enter the location of a site.
Work with: type or select a site
Add...
Find more software by working with the "Available Software Sites" preferences.
type filter text
Name
Version
There is no site selected
Add Repository
Name: ADT Plugin
Location: https://dl-ssl.google.com/android/eclipse
Local...
Archive...
OK
Cancel
Select All
Deselect A
Details
Show only the latest versions of available software
Group items by category
Show only software applicable to target environment
Contact all update sites during install to find required software
Hide items that are already installed
What is already installed?
< Back
Next >
Finish
Cancel

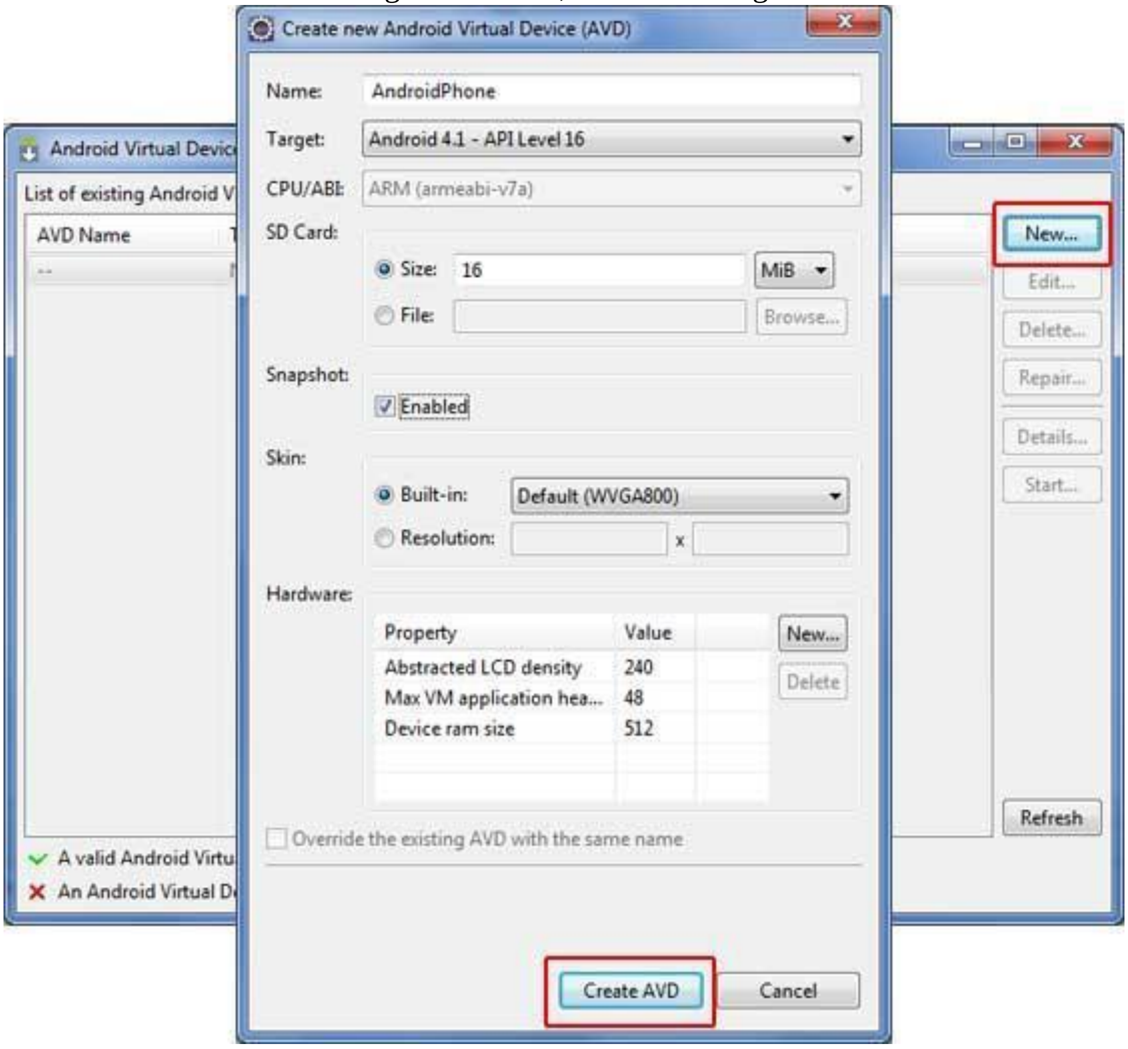
Now use Add button to add ADT Plugin as name and **https://dl-ssl.google.com/android/eclipse/** as the location.
Then click OK to add this location, as soon as you will click OK button to add this location, Eclipse starts searching for the plug-in available the given location and finally lists down the found plugins.

Now select all the listed plug-ins using Select All button and click Next button which will guide you ahead to install Android Development Tools and other required plugins.


Install
Available Software
Check the items that you wish to install.
Work with: ADT Plugin - https://dl-ssl.google.com/android/eclipse/
Add...
Find more software by working with the "Available Software Sites" preferences.
type filter text
Name
Version
Developer Tools
NDK Plugins
Select All
Deselect All
6 items selected
Details
Show only the latest versions of available software
Group items by category
Show only software applicable to target environment
Contact all update sites during install to find required software
Hide items that are already installed
What is already installed?
< Back
Next >
Finish
Cancel
Figure 4: Install Plugin

3.4 Create Android Virtual Device:

To test your Android applications you will need a virtual Android device. So before we start writing our code, let us create an Android virtual device. Launch Android AVD Manager using Eclipse menu options Window > AVDManager> which will launch Android AVD Manager. Use New button to create a new Android Virtual Device and enter the following information, before clicking Create AVD button:



Create new Android Virtual Device (AVD)

Name: AndroidPhone
Target: Android 4.1 - API Level 16
CPU/ABI: ARM (armeabi-v7a)
SD Card: Size: 16 MiB
Snapshot: Enabled ✓
Skin: Built-in: Default (WVGA800)

Hardware:

Property	Value
Abstracted LCD density	240
Max VM application hea...	48
Device ram size	512

Figure 5: Create AVD

If your AVD is created successfully it means your environment is ready for Android application development.

4. Architecture Overview:

Android operating system is a stack of software components which is roughly divided into five sections and four main layers as shown below in the architecture diagram:

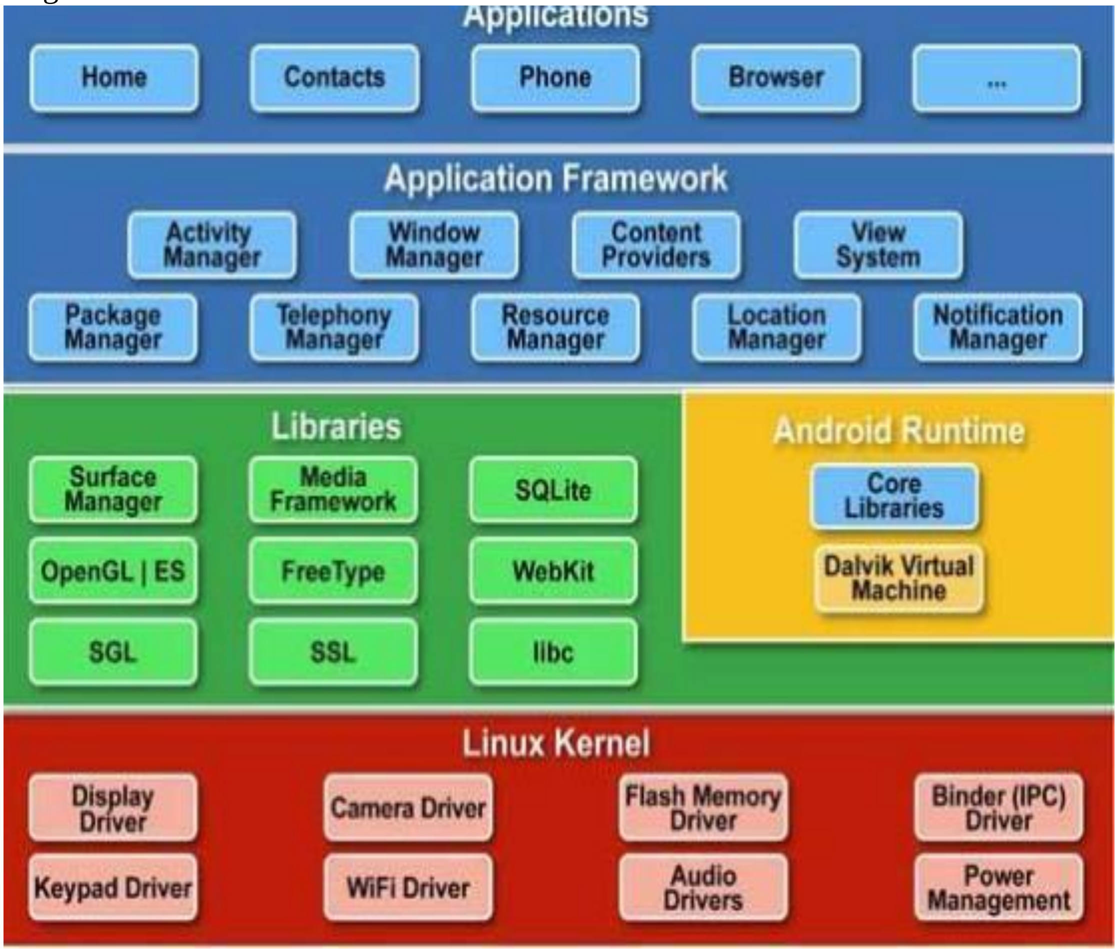


Figure 5: Create Androids components Stack

Linux kernel

At the bottom of the layers is Linux - Linux 2.6 with approximately 115 patches. This provides basic system functionality like process management, memory management, device management (like camera, keypad, display etc). Also, the kernel handles all the things that Linux is really good at such as networking and a vast array of device drivers, which take the pain out of interfacing to peripheral hardware.

Libraries

On top of Linux kernel there is a set of libraries including open-source Web browser engine WebKit, well known library libc, SQLite database which is a useful repository for storage and sharing of application data, libraries to play and record audio and video, SSL libraries responsible for Internet security etc.

Android Runtime

This is the third section of the architecture and available on the second layer from the bottom. This section provides a key component called Dalvik Virtual Machine which is a kind of Java Virtual Machine specially designed and optimized for Android. The Dalvik VM makes use of Linux core features like memory management and multi-threading, which is intrinsic in the Java language. The Dalvik VM enables every Android application to run in its own process, with its own instance of the Dalvik virtual machine. The Android runtime also provides a set of core libraries which enable Android application developers to write Android applications using standard Java programming language.

Application Framework

The Application Framework layer provides many higher-level services to applications in the form of Java classes. Application developers are allowed to make use of these services in their applications.

Applications

You will find all the Android application at the top layer. You will write your application to be installed on this layer only. Examples of such applications are Contacts Books, Browser, and Games etc.

5. Application Components

Application components are the essential building blocks of an Android application. These components are loosely coupled by the application manifest fileAndroidManifest.xml that describes each component of the application and how they interact.

The following four components are the main components that can be used within an Android application:

- ***Activities***

An activity represents a single screen with a user interface. For example, an email application might have one activity that shows a list of new emails, another activity to compose an email. If an application has more than one activity, then one of them should be marked as the activity that is presented when the application is launched. An activity is implemented as a subclass of Activity class as follows:

```
public class MainActivity extends Activity {

}
```

- ***Services***

A service is a component that runs in the background to perform long-running operations. For example, a service might play music in the background while the user is in a different application. A service is implemented as a subclass of Service class as follows:

```
public class MyService extends Service {

}
```

- ***Broadcast Receivers:***

They handle the communication between Android OS and applications. A broadcast receiver is implemented as a subclass of BroadcastReceiver class and each message is broadcasted as an Intent object.

```
public class MyReceiver extends BroadcastReceiver {

}
```

- ***Content Providers:***

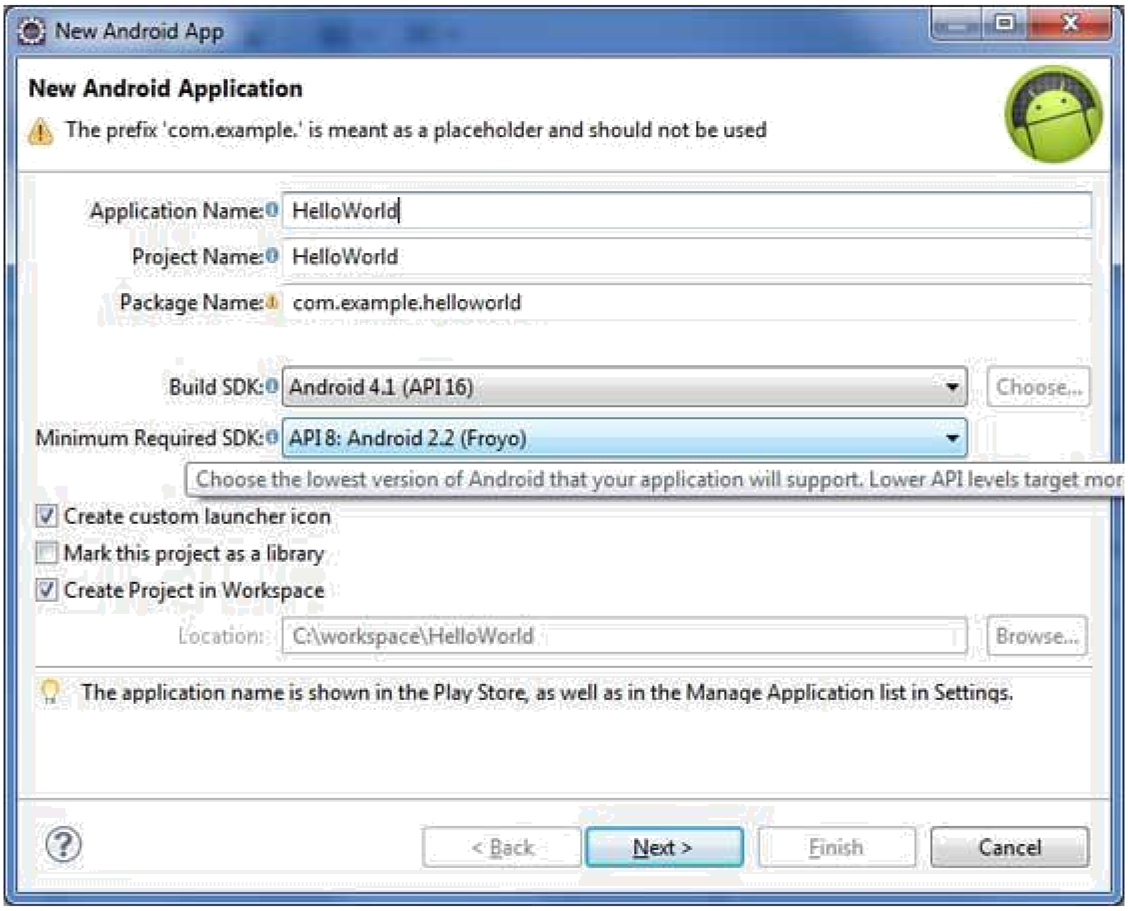
They handle data and database management issues. A content provider is implemented as a subclass of ContentProvider class and must implement a standard set of APIs that enable other applications to perform transactions.

```
public class MyContentProvider extends ContentProvider {

}
```

There are additional components which will be used in the construction of above mentioned entities, their logic, and wiring between them. These components are:

Component	Description
Fragments:	Represents a behavior or a portion of user interface in an Activity.
Views:	UI elements that are drawn onscreen including buttons, lists forms etc.
Layouts:	View hierarchies that control screen format and appearance of the views
Intents:	Messages wiring components together.
Resources:	External elements, such as strings, constants and drawables pictures.
Manifest:	Configuration file for the application.

6. Hello World Example:


New Android App

New Android Application

The prefix 'com.example.' is meant as a placeholder and should not be used

Application Name: HelloWorld

Project Name: HelloWorld

Package Name: com.example.helloworld

Build SDK: Android 4.1 (API 16)

Minimum Required SDK: API 8: Android 2.2 (Froyo)

Choose the lowest version of Android that your application will support. Lower API levels target mor

[✓] Create custom launcher icon

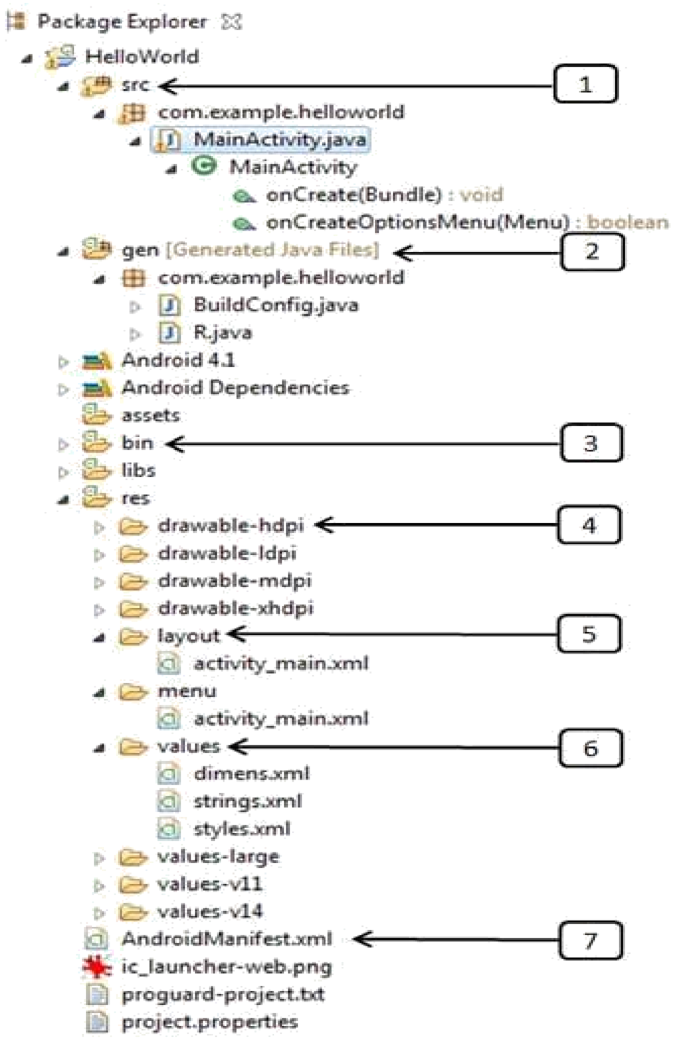
[ ] Mark this project as a library

[✓] Create Project in Workspace

Location: C:\workspace\HelloWorld

The application name is shown in the Play Store, as well as in the Manage Application list in Settings.

Next, follow the instructions provided and keep all other entries as default till the final step. As your project is created successfully, let's take a look at the directories and files in the Android project.in the left of your screen there is a Package explorer that is shown in the figure below:



1- src: This contains the .java source files for your project. By default, it includes an MainActivity.java source file having an activity class that runs when your app is launched using the app icon.

2- gen: This contains the .R file, a compiler-generated file that references all the resources found in your project. You should not modify this file.

3- bin: This folder contains the Android package files .apk built by the ADT during the build process and everything else needed to run an Android application.

4- res/drawable-hdpi: This is a directory for drawable objects that are designed for high-density screens. everything else needed to run an Android application.

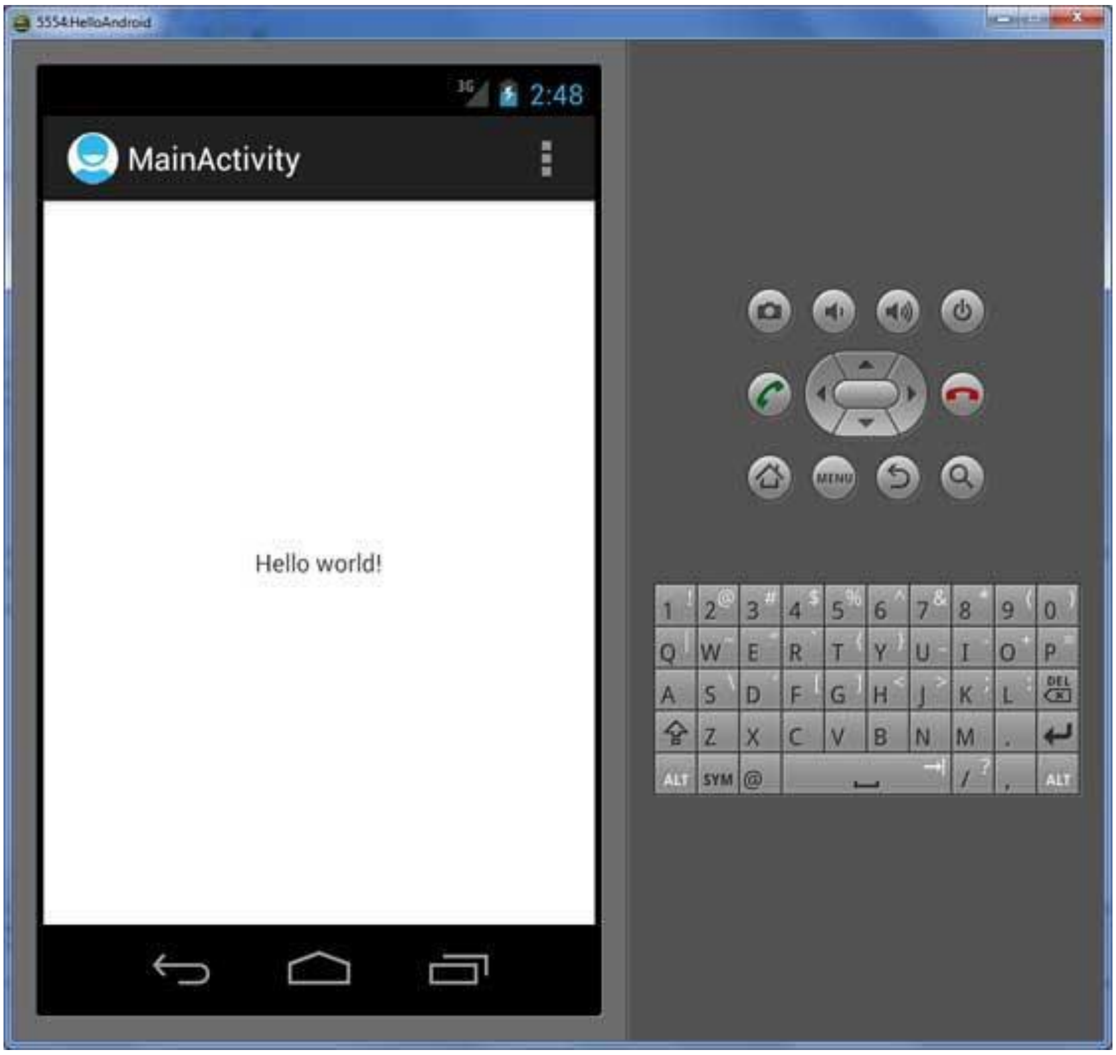
5- res/layout: This is a directory for files that define your app's user interface.

6- res/values: This is a directory for other various XML files that contain a collection of resources, such as strings and colors definitions.

7- AndroidManifest.xml: This is the manifest file which describes the fundamental characteristics of the app and defines each of its components.

Running the Application:

To run the app from Eclipse, open one of your project's activity files and click Run icon from the toolbar. Eclipse installs the app on your AVD and starts it and if everything is fine with your setup and application, it will display following Emulator window:



Hashemite University
Faculty of Engineering and Technology
Computer Engineering Department

Experiment 7
Adding UI Components and Events

1 Objectives:

At the end of this lab you will be expected to know:

- How to use Events and Event Listeners.
- What Views Buttons and text views.
- How to deal with What Views.
-

1.2 Pre-lab Preparation:

- Read the experiment thoroughly and make a real effort to build answer the pre-lab quiz.

1.3 Equipments:

- Personal computer with ***Eclipse IDE and Android SDK*** installed on it.

2 Theory:

2.1 The Layout File:

Every single screen (with a user interface) in the Android Application is represented by an Activity, there are two files related to each screen in the application the first file is Layout file which is a .XML that represent the UI Components of this screen and the second file is the Activity which is java file handles the user interaction to the screen of the smart phone. The activity_main.xml is a layout file available in res/layout directory that is referenced by the application when building its interface. We will modify this file very frequently to change the layout of your application.

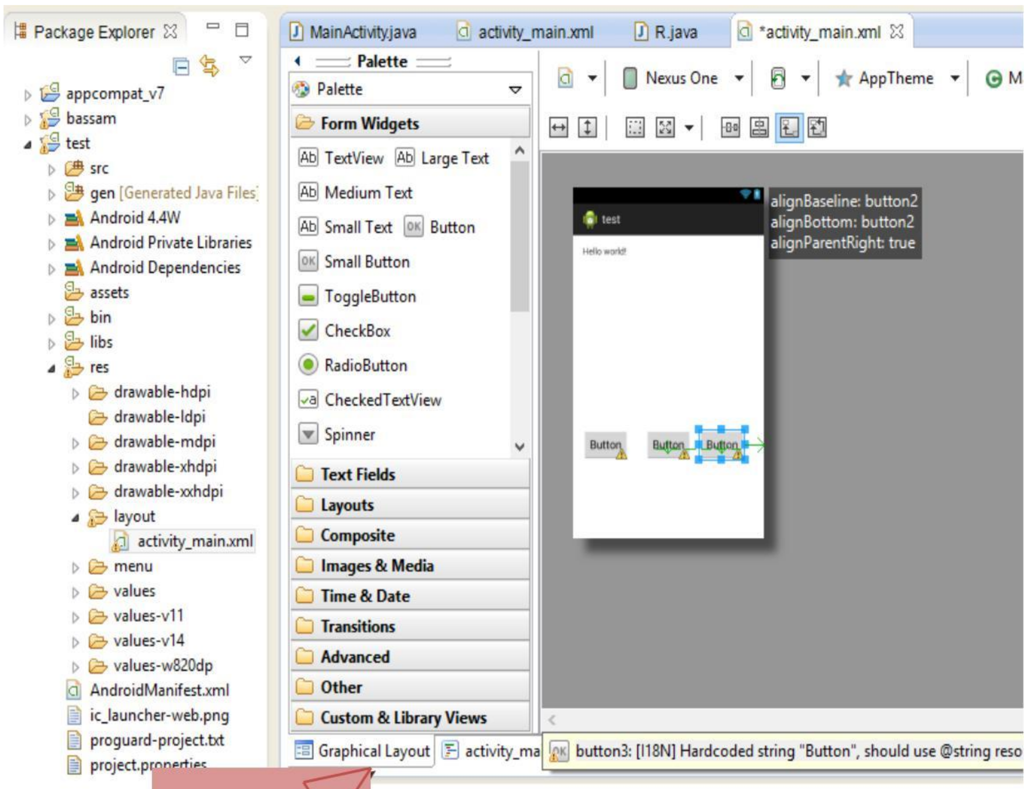
For The "Hello World!" application, this file will have following content related to default layout:

```
<RelativeLayout
xmlns:android=http://schemas.android.com/apk/res/android
xmlns:tools=http://schemas.android.com/tools
android:layout_width="match_parent"
android:layout_height="match_parent" >
    <TextView
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:layout_centerHorizontal="true"
         android:layout_centerVertical="true"
         android:padding="@dimen/padding_medium"
         android:text="@string/hello_world"
          tools:context=".MainActivity" />
</RelativeLayout>
```

We can add UI components to the screen by writing the XML that represents the UI we want and specify all its properties in this file, for example to add a button to any activity we write the following XML in the corresponding layout file under res/layout directory:

```xml
<Button xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/button_send"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_send"
    android:onClick="sendMessage" />
```

However we can do the same job using the graphical interface by just dragging the component from the toolbox in the left of the screen to the graphical designer of the screen as the following figure describes:



alignBaseline: button2
alignBottom: button2
alignParentRight: true

button3: [I18N] Hardcoded string "Button", should use @string reso

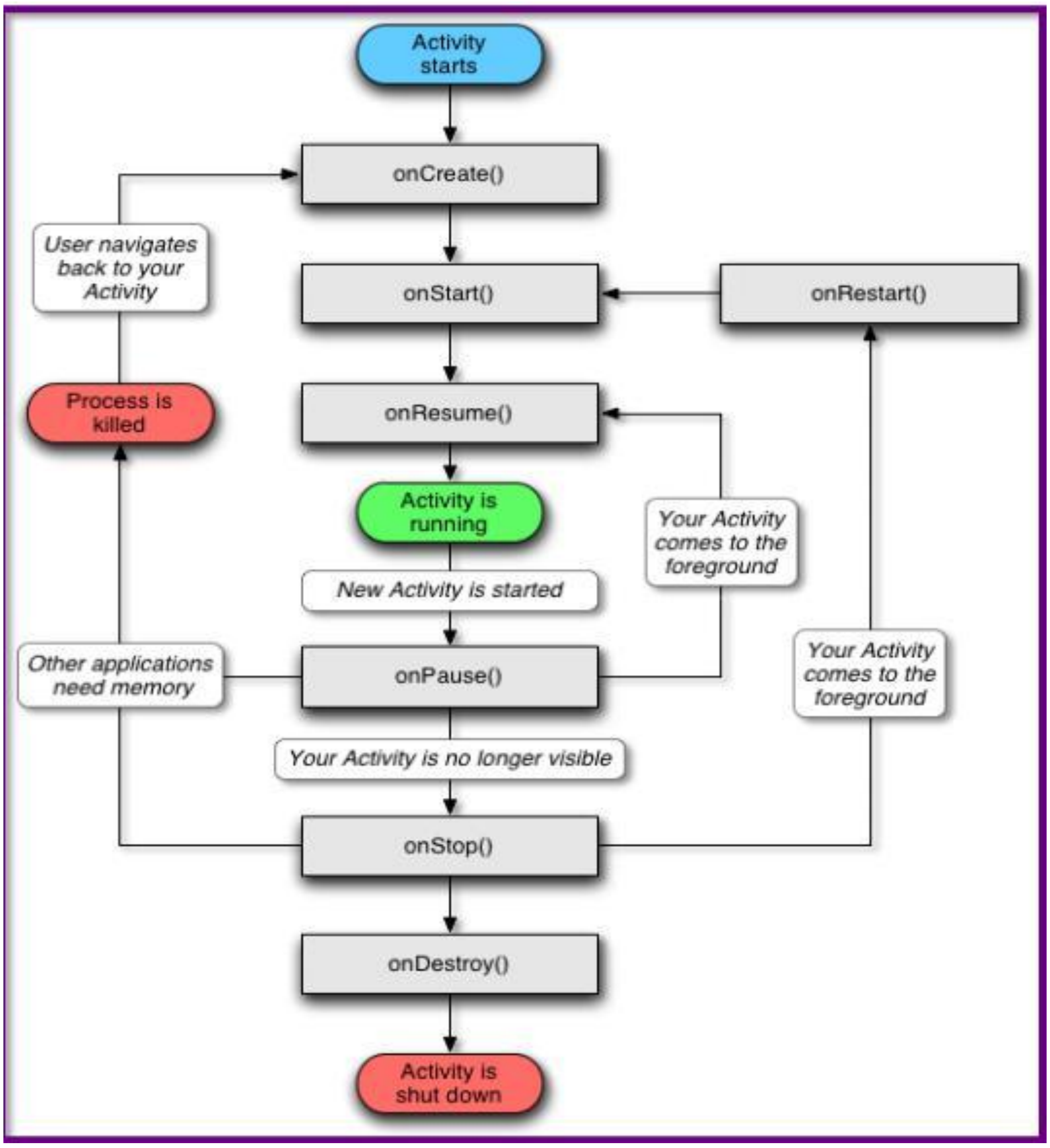
Figure 1

To switch between the Graphical designer and the XML file.

2.2 Activity Life cycle:

The Activity base class defines a series of events that govern the life cycle of an activity. The Activity class defines the following events:

- onCreate() — Called when the activity is first created
- onStart() — Called when the activity becomes visible to the user
- onResume() — Called when the activity starts interacting with the user
- onPause() — Called when the current activity is being paused and the previous activity is being resumed
- onStop() — Called when the activity is no longer visible to the user
- onDestroy() — Called before the activity is destroyed by the system (either manually or by the system to conserve memory
- onRestart() — Called when the activity has been stopped and is restarting again



Activity starts
onCreate()
User navigates back to your Activity
onStart()
onRestart()
Process is killed
onResume()
Activity is running
Your Activity comes to the foreground
New Activity is started
Other applications need memory
onPause()
Your Activity comes to the foreground
Your Activity is no longer visible
onStop()
onDestroy()
Activity is shut down

3 Adding Event handlers:

3.1 Using onClick attribute:

To define the click event handler for a button, add the android:onClick attribute to the <Button> element in your XML layout. The value for this attribute must be the name of the method you want to call in response to a click event. The Activity hosting the layout must then implement the corresponding method.

For example, here's a button with click event handler using android:onClick:

```
<Button xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/button_send"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_send"
    android:onClick="sendMessage" />
```

Within the Activity that hosts this layout, the following method handles the click event:

```
/** Called when the user touches the button
*/ public void sendMessage(View view) {
    // Do something in response to button click
}
```

The method you declare in the android:onClick attribute must have a signature exactly as shown above. Specifically, the method must:

- Be public
- Return void
- Define a View as its only parameter (this will be the View that was clicked)

3.2 Using an OnClickListener:

You can also declare the click event handler programmatically rather than in an XML layout. This might be necessary if you instantiate the Button at runtime or you need to declare the click behavior in a Fragment subclass.

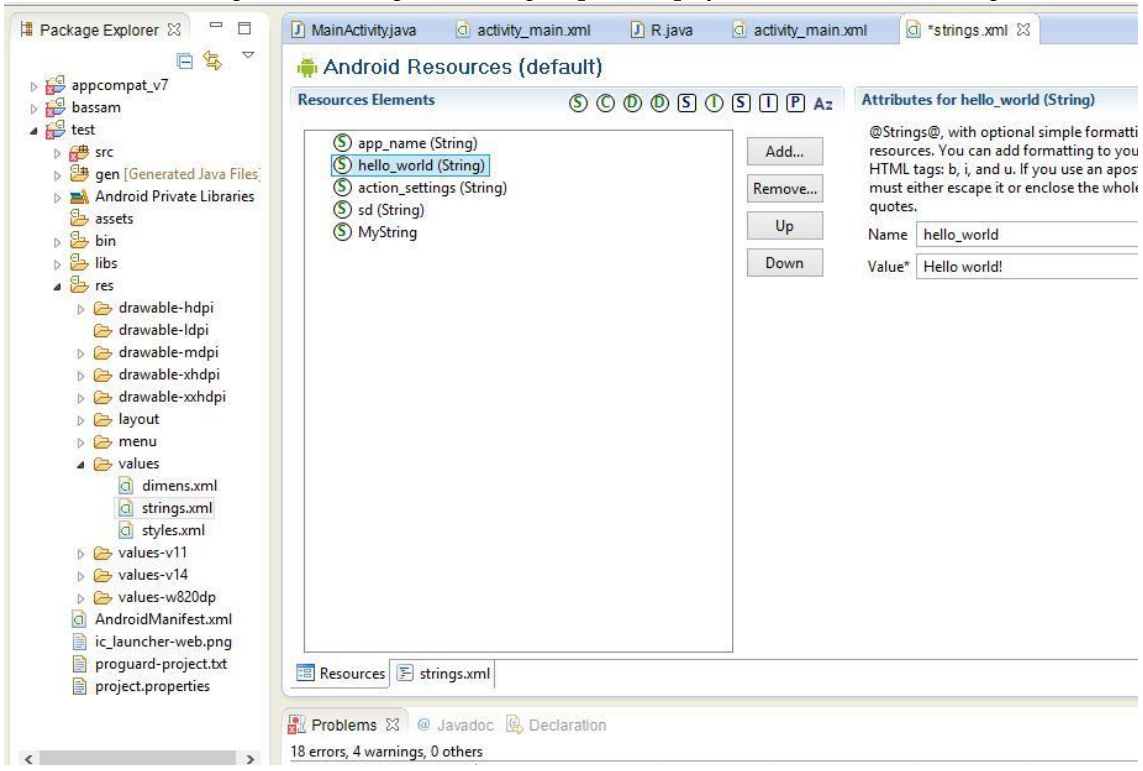
To declare the event handler programmatically, create an View.OnClickListener object and assign it to the button by calling setOnClickListener(View.OnClickListener). As follow:

```
Button button = (Button) findViewById(R.id.button_send);
button.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        // Do something in response to button click
    }
});
```

3.2 *Using strings.xml file:*

If you look to the attribute of the button we added in the previous steps you notice the attribute `android:text="@string/button_send"` this attribute corresponds to the text that will appear on the button when the user run the application, but what does the `"@string/button_send"` means? Well this means that there is a string defined inside the **string.xml** file called **button_send** and has some value to be assigned to the button, pay attention that **"button_send"** is just the name of the string and the value of that string is deferent form the name of it.

To add a string to the string.xml file go open it up, you will see something like this:



@Strings@, with optional simple formatti
resources. You can add formatting to you
HTML tags: b, i, and u. If you use an apos
must either escape it or enclose the whole
quotes.

Name hello_world

Value* Hello world!

18 errors, 4 warnings, 0 others

To add a string just click the Add button write a name and a value for your string in the boxes in the right side of the screen.

Hashemite University
Faculty of Engineering and Technology
Computer Engineering Department

Experiment 8
Multi Activity Applications, Intents and Filters

1.1 Objectives:

At the end of this lab you will be expected to know:

- How to use Events and Event Listeners.
- What Views Buttons and text views.
- How to deal with What Views.
-

1.2 Pre-lab Preparation:

- Read the experiment thoroughly and make a real effort to build answer the pre-lab quiz.

1.3 Equipments:

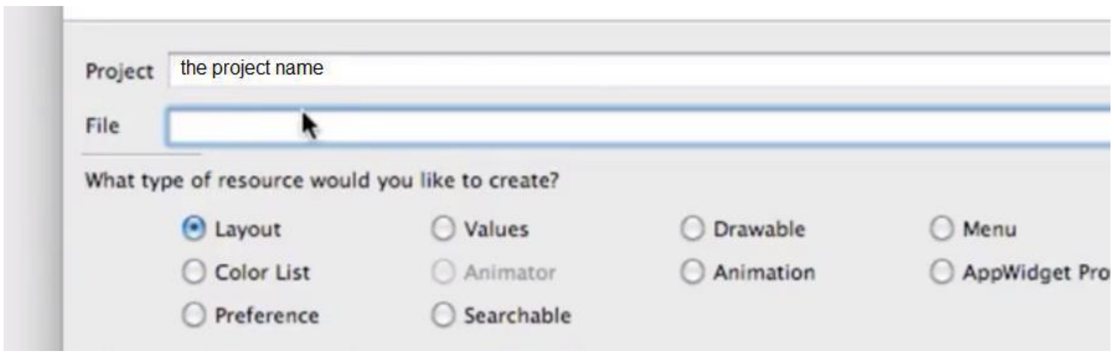
- Personal computer with ***Eclipse IDE and Android SDK*** installed on it.

2.1 Theory:

Most Android applications need to have more than one screen (and every screen needs a Layout file for the user interface and an Activity file for the code behind) so in this experiment we will learn how to define our own activity and how to move between "screens".

2.2 Adding Layout

The first thing we want to do is to add the layout file which will represent the UI of our screen, to do so we will right click the Layout folder in the Package Explorer to in left side of the Eclipse window then choose New > Other > Android XML file (under Android section), you will see screen like this :



Project: the project name
File:
What type of resource would you like to create?
[✓ Layout] Values Drawable Menu
Color List Animator Animation AppWidget Pro
Preference Searchable

In the write the layout name in the "File" box and hit finish, make sure that the Layout option is checked.

Now you added your layout file to the application so you can add the component you want to that file but you should notify your application about this layout and how to call it to be displayed to the user.

2.3 Intents and Filters

An Android Intent is an object carrying an intent i.e. message from one component to another component with-in the application or outside the application. The intents can communicate messages among any of the three core components of an application - activities, services, and broadcast receivers. An Intent object, is a passive data structure holding an abstract description of an operation to be performed.

In this lecture we will use the Intent and the filter to move from one activity to the other and to achieve that we need an Intent to the "caller" Activity and a Filter to be added to the target one but the question is how to do that and where exactly to but the code of the Intent and Filter and that will be described in the following parts.

2.4 Altering the Manifest file

Once the layout xml file is created it's the time to add it to the manifest file so other activities can use it to do that, open the AndroidManefiest.xml file, you will see xml tags like this:

```
 <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="21" />
```

It's clear that this tag specify the target and the minimum SDK that the application will run on. The important tag that we will describe is this:

```
<activity
      android:name =".MainActivity"
      android:label="@string/app_name" >
      <intent-filter>
         <action   android:name="android.intent.action.MAIN" />
         <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
</activity>
```

Clearly this tag (and sub tags) deal with an Activity, the first attribute `android:name` specifies the name of the java file of that activity proceeded with "." pay attention it is case sensitive.
The second one specify the text that will be displayed on the title of the activity when it displayed to the user (note that its references a string.xml file).

The sub-tag is the core of our interest here it's the `<intent-filter>` tag which determine the Intents that this activity will respond to, the `<intent-filter>` has two sub tags the first one is `<action>` which determines the action name which the "caller" activity should be call this one with. A good way to keep track of all your action names is to make the action name as follow `="package.name.CLASSNAME"`

The last tag we will talk about here is the `<category>` which specify the category of the Intent that this filter will respond to. There are a lot of intent categories we will describe two of them:

- *`android.intent.category.LAUNCHER:`* means that this activity should be displayed in the top-level launcher, you will find it with the activitymain.xml that will be created with the application.

- *`android.intent.category.DEFAULT:`* this type what we will use for any activity we will add to our application.

2.5 Setup the Activity

You need to add the java file by right clicking the **src** folder and choose add > class then specify the class name and click finish.
Once we have this class added we need to do some modifications, first of all let us make this class represent an Activity by extending the Activity class by writing extends Activity between the class name and the starting brace of the class body "{" we need to import the activity library by writing import android.app.Activity; before the class definition. Now right click anywhere inside the class and choose Source > Override Implement Methods, from the list under Activity choose onCreate(Bundle) method then you will have it setup for you, now in side this method write the following line :

setContentView(R.layout.yourlayout);

in this line we called a built in method which will tie this java code to an xml file that represent the user interface for this activity, as you see this method needs to know which xml file is the chosen one so we used the R file in that way to I identify the exact xml file to be used.

Now let us call our activity first we need to declare the Intent that will call it like this:

```
Intent intentname = new Intent ("the.intent.name");
```

Finally we will use this intent to bring up the wanted activity as follow :

```
stratActivity(intentname);
```

After the previous line of code being executed the previous screen will disappear and the new one will show up instead, and by the same way you can add as many Activities as you need to your application and you can move between them using the Intent and the Filters.

Hashemite University
Faculty of Engineering and Technology
Computer Engineering Department

Experiment 9
Introduction to PSPICE

1.1 Objectives:

At the end of this lab you will be expected to know:

- How to use Events and Event Listeners.
- What Views Buttons and text views.
- How to deal with What Views.
-

1.2 Pre-lab Preparation:

- Read the experiment thoroughly and make a real effort to build answer the pre-lab quiz.

1.3 Equipments:

- Personal computer with ***Eclipse IDE and Android SDK*** installed on it.

2.1 Background:

SPICE (Simulation Program with Integrated Circuit Emphasis) was originally developed in the 1970s at Berkeley. It solves the nonlinear differential equations describing components such as transistors, resistors, capacitors, and voltage sources.

PSPICE is commercial version with a free limited student version that allows the user to simulate a circuit and extract key voltages and currents all versions of SPICE read an input file and generate a list file with results, warnings, and error messages. The input file is often called a SPICE deck and each line a card because it was once provided to a mainframe as a deck of punch cards. The input file contains a 'netlist' consisting of components and nodes. It also contains simulation options, analysis commands, and device models. The 'netlist' can be entered by hand or extracted from a circuit schematic or layout in a CAD program.

2.2 How to write SPICE Deck:

The first line of a SPICE deck must be a comment, typically indicating the title of the simulation. The comments is those lines that starts with (*).The last statement of a SPICE deck must be (.end) the lines starts with dot (.) is Control statements. Here is some control statements that is commonly used:

- **.plot** command generates a textual plot of the node variables specified
- **.print** statement prints the results in a multicolumn table

- **.option** post command generates a file (.tr0)containing the results of the specified (transient) analysis

- **.include** reads another SPICE file from disk.
- **.dc** command varies certain voltage source DC voltage within a range in increments of the amount specified by the user.
- **.tran** command specify the step size and the duration that the transient analysis will be performed with when plotting the response of some circuit.

The lines that describes the circuit components should start whit specific letter followed by a user-defined name and the component parameters separated by spaces the following tables show the representation of the common SPICE elements and SPICE unites.

Letter	Element
R	Resistor
C	Capacitor
L	Inductor
K	Mutual inductor
V	Independent voltage source
I	Independent current source
M	MOSFET
D	Diode
Q	Bipolar transistor
W	Lossy transmission line
X	Subcircuit
E	Voltage-controlled voltage source
G	Voltage-controlled current source
H	Current-controlled voltage source
F	Current-controlled current source

Table 9.1: Spice elements

Letter	Unit	Magnitude
a	atto	10^-18
f	femto	10^–15
p	pico	10^–12
n	nano	10^–9
u	micro	10^–6
m	milli	10^–3
k	kilo	10^3
x	mega	10^6
g	giga	10^9

Table 9.2: Spice unites

As mentioned before the parameters of the component should be specified in the line that represent that component, below is the parameters needed for every component.

Parameter Syntax

Resistor:
R<name> [+ node] [- node] [value]

Capacitor:
C<name> [+ node] [- node] [value] [IC = <initial value>, optional]

Inductor:
L<name> [+ node] [- node] [value] [IC = <initial value>, optional]

Independent Sources:
I<name> [- node] [+ node] [value] [type] [transient spec]
V<name> [+ node] [- node] [value] [type] [transient spec]

Dependent Sources:
VCVS: **E**<name> [+ node] [- node] [+controlling node] [-controlling node] [gain]
CCCS: **F**<name> [+ node] [- node] [Vbranch] [gain]
VCCS: **G**<name> [+ node] [- node] [+controlling node] [-controlling node] [gain]
CCVS: **H**<name> [+ node] [- node] [Vbranch] [gain]

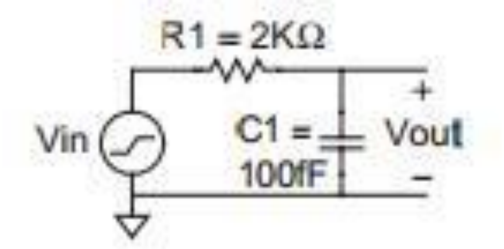
For example to represent a Resistor the following line should be written:

```
        R1 gnd abc 1k
```

This line represent a 1000 ohm resistor named R1 from ground (and) to node called “abc”, notice that every node in the circuit should be named (user defined name) the gnd is a special node name defined to be the 0 V reference.

2.3 Using OrCAD PSpice:

Now consider the following circuit:



This is a typical form of the RC circuits consists of voltage source (Vin), a resistor (R1), and a capacitor (C1). Let’s call the node between the Vin and R “in” and the node between the R and the C1 “out” so the value of the Vout will be the difference between the “out” node and the ground (gnd). In addition we will consider that the voltage source (Vin) is defined as a piecewise linear (PWL) source.

To write SPICE NetList that represent this circuit should be as follow:

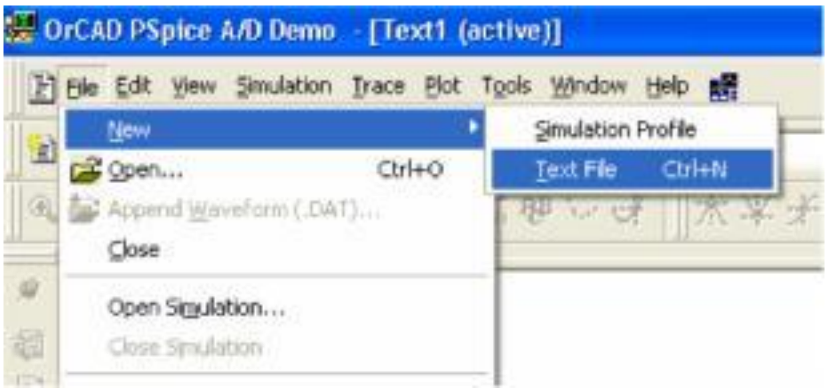
```
Vin in gnd pwl 0ps 0 100ps 0 150ps 1.0 1ns 1.0

R1 in out 2k

C1 out gnd 100f
```

The previous lines are the netlist only i.e. if we want to collect some other data about the response of the circuit we should write the appropriate commands that will command the PSpice to do so.
Suppose that we want to plot the values of the Vin and the Vout which is the result that will be achieved by the following code.

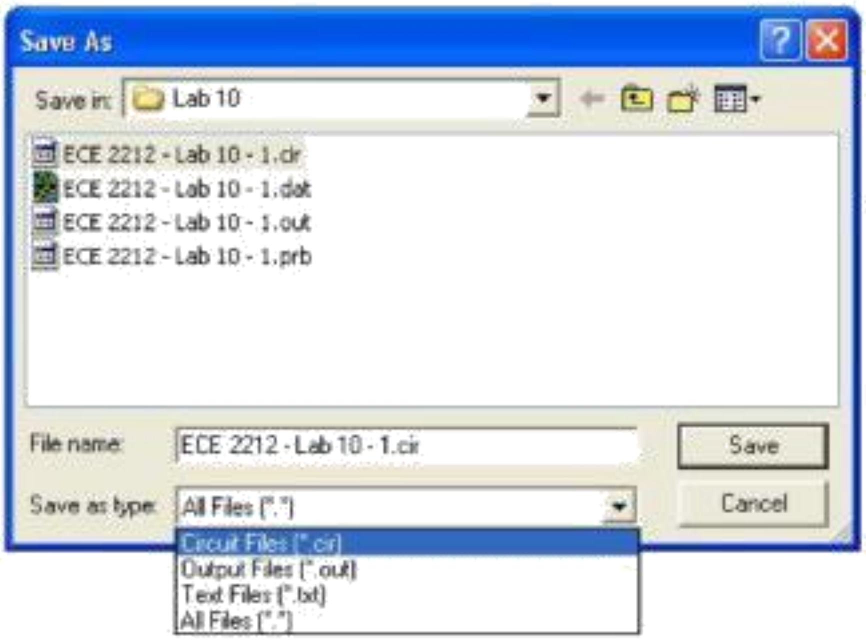
```
* rc.sp
* Find the response of RC circuit to rising input
*------------------------------------------------------------------
* Parameters and models
*------------------------------------------------------------------

.option post

*------------------------------------------------------------------
* Simulation netlist
*------------------------------------------------------------------

Vin in gnd pwl 0ps 0 100ps 0 150ps 1.0 1ns 1.0
R1 in out 2k
C1 out gnd 100f

*------------------------------------------------------------------
* Stimulus
*------------------------------------------------------------------

.tran 20ps 1ns
.plot v(in) v(out)
.end
```

Now the final question how to write this code in the application and how to run it to show the desired result, and here is the steps to get the job done:

- Open **PSPICE AD Student**
- Goto **File -> New -> Text File** or Click on the **New Icon** and then **New Text File**


OrCAD PSpice A/D Demo - [Text1 (active)]
File Edit View Simulation Trace Plot Tools Window Help
New
Open... Ctrl+O
Append Waveform (.DAT)...
Close
Open Simulation...
Close Simulation
Simulation Profile
Text File Ctrl+N

- Write your PSPICE circuit. Make sure to have a label with a title, include all circuit elements and include a .END statement.
- Goto **File -> Save As** and save the file with a **.cir** circuit extension.



Save As

Save in: Lab 10

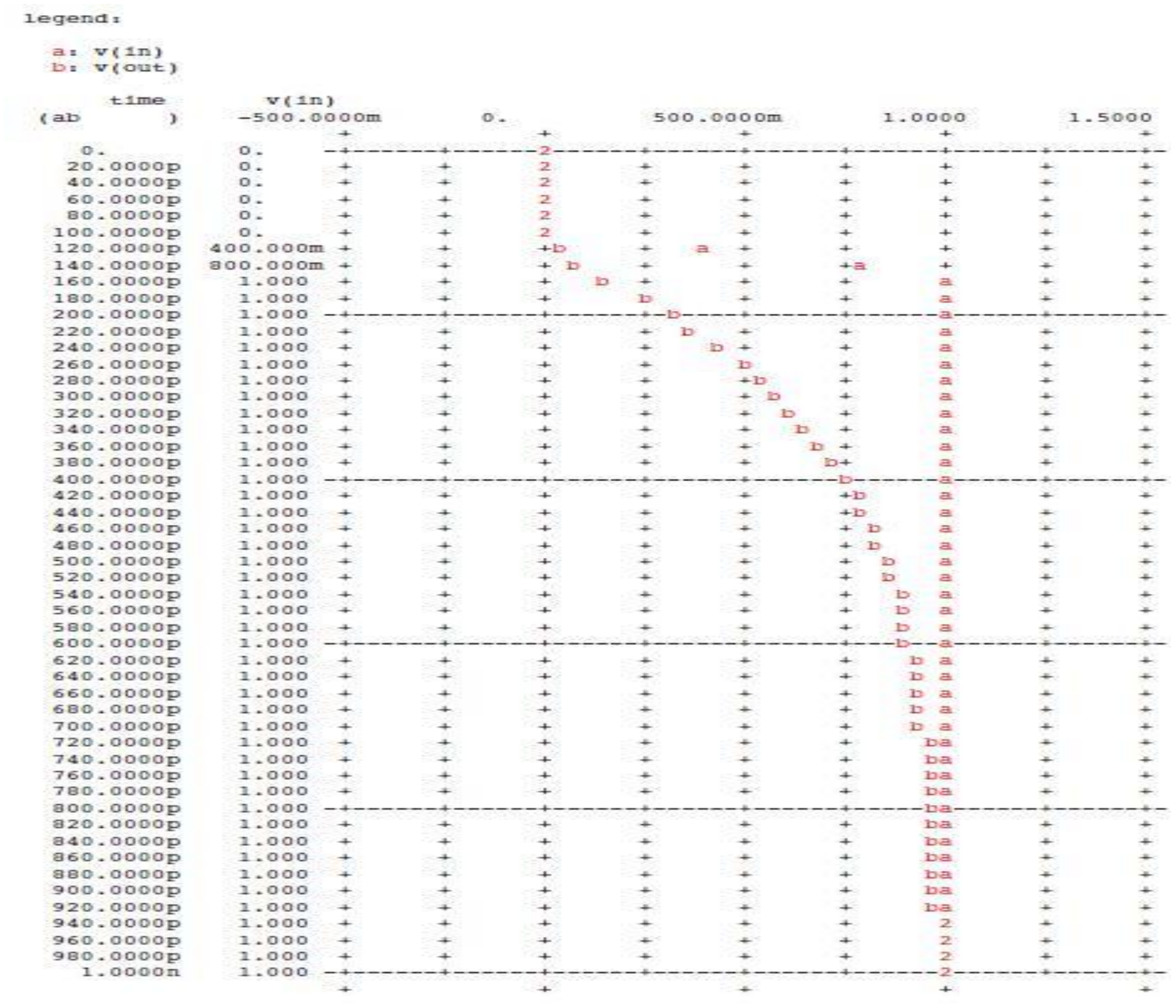
ECE 2212 - Lab 10 - 1.cir
ECE 2212 - Lab 10 - 1.dat
ECE 2212 - Lab 10 - 1.out
ECE 2212 - Lab 10 - 1.prb

File name: ECE 2212 - Lab 10 - 1.cir

Save as type: All Files (*.*)

- Goto **Simulation -> Run Filename.** If the option is not available you may need to close the file and reopen it.

Once the program has run the output file may appear. If it doesn't you can go to **View -> Output File** or click on the **Output File Icon** on the left sidebar. The output should look like this:



```
legend:

  a: v(in)
  b: v(out)

      time       v(in)
 (ab      )    -500.0000m        0.          500.0000m       1.0000       1.5000
                       +             +             +             +             +
    0.         0.      -+------+------2------+------+------+------+------+------+-
   20.0000p    0.       +      +      2      +      +      +      +      +      +
   40.0000p    0.       +      +      2      +      +      +      +      +      +
   60.0000p    0.       +      +      2      +      +      +      +      +      +
   80.0000p    0.       +      +      2      +      +      +      +      +      +
  100.0000p    0.       +      +      2      +      +      +      +      +      +
  120.0000p  400.000m   +      +      +b     +   a  +      +      +      +      +
  140.0000p  800.000m   +      +      + b    +      +      +a     +      +      +
  160.0000p    1.000    +      +      +   b  +      +      +      a      +      +
  180.0000p    1.000    +      +      +      b      +      +      a      +      +
  200.0000p    1.000   -+------+------+------+-b----+------+------a------+------+-
  220.0000p    1.000    +      +      +      +  b   +      +      a      +      +
  240.0000p    1.000    +      +      +      +    b +      +      a      +      +
  260.0000p    1.000    +      +      +      +      b      +      a      +      +
  280.0000p    1.000    +      +      +      +      +b     +      a      +      +
  300.0000p    1.000    +      +      +      +      + b    +      a      +      +
  320.0000p    1.000    +      +      +      +      +  b   +      a      +      +
  340.0000p    1.000    +      +      +      +      +   b  +      a      +      +
  360.0000p    1.000    +      +      +      +      +    b +      a      +      +
  380.0000p    1.000    +      +      +      +      +     b+      a      +      +
  400.0000p    1.000   -+------+------+------+------+------b------a------+------+-
  420.0000p    1.000    +      +      +      +      +      +b     a      +      +
  440.0000p    1.000    +      +      +      +      +      +b     a      +      +
  460.0000p    1.000    +      +      +      +      +      + b    a      +      +
  480.0000p    1.000    +      +      +      +      +      + b    a      +      +
  500.0000p    1.000    +      +      +      +      +      +  b   a      +      +
  520.0000p    1.000    +      +      +      +      +      +  b   a      +      +
  540.0000p    1.000    +      +      +      +      +      +   b  a      +      +
  560.0000p    1.000    +      +      +      +      +      +   b  a      +      +
  580.0000p    1.000    +      +      +      +      +      +   b  a      +      +
  600.0000p    1.000   -+------+------+------+------+------+---b--a------+------+-
  620.0000p    1.000    +      +      +      +      +      +    b a      +      +
  640.0000p    1.000    +      +      +      +      +      +    b a      +      +
  660.0000p    1.000    +      +      +      +      +      +    b a      +      +
  680.0000p    1.000    +      +      +      +      +      +    b a      +      +
  700.0000p    1.000    +      +      +      +      +      +    b a      +      +
  720.0000p    1.000    +      +      +      +      +      +     ba      +      +
  740.0000p    1.000    +      +      +      +      +      +     ba      +      +
  760.0000p    1.000    +      +      +      +      +      +     ba      +      +
  780.0000p    1.000    +      +      +      +      +      +     ba      +      +
  800.0000p    1.000   -+------+------+------+------+------+-----ba------+------+-
  820.0000p    1.000    +      +      +      +      +      +     ba      +      +
  840.0000p    1.000    +      +      +      +      +      +     ba      +      +
  860.0000p    1.000    +      +      +      +      +      +     ba      +      +
  880.0000p    1.000    +      +      +      +      +      +     ba      +      +
  900.0000p    1.000    +      +      +      +      +      +     ba      +      +
  920.0000p    1.000    +      +      +      +      +      +     ba      +      +
  940.0000p    1.000    +      +      +      +      +      +      2      +      +
  960.0000p    1.000    +      +      +      +      +      +      2      +      +
  980.0000p    1.000    +      +      +      +      +      +      2      +      +
    1.0000n    1.000   -+------+------+------+------+------+------2------+------+-
                       +             +             +             +             +
```

Experiment 10: Design and Conduct an Experiment

1. *Objectives :*

- Design and Conduct an Experiment to verify that ….
- The ability to interpret and analyze given situation and plan solutions for it.
- The ability to choose the appropriate tools and instruments that suit the desired job.
- The ability to use the chosen tools and instruments to achieve the goals of the experiment.

2. *Student proposal*

You have to submit a proposal before starting your experiment. Your proposal will describe the objective or the goal of the experiment in addition to the proposed experimental set-up, instruments, tools, governing equations, etc.

Remember that your work does not based on the existing well-defined experimental procedure (manual).

You have to develop a new technique in the lab, so that you can conduct a new experiment to achieve the goal. In addition, you have to report the results. Students can use any of lab equipment, tools, instrument to setup their own experiments.
The design of an experiment can be integrated in the lab through different approaches:

- Design-build-test approach.
- Modifying an existing experimental setup.
- Utilizing instrumentations from other experiment to study a certain phenomenon.

3. *Deliverables:*

Each group (2-3 students) must deliver a report written using a word processor. The final report should include:

- **Objectives:** Statement of what you are going to achieve.
- **Designing experiments**: develop a methodology which will produce high quality data that can be used to evaluate specific process or parameter.
- **Experimental setup:** the apparatus, devices, and instruments used to conduct the experiment should be clearly specified [figures or photos may be added].
- **Theoretical background:** the theory related to the experiment, including all assumptions and equations.
- **Conduct the experiment:** clear procedure should be specified.

- **Experimental results:** represent all data.

- **Analyze and interpret data:** develop, if you can, a mathematical model or computer simulation to correlate or interpret experimental results.

- **Discussion and conclusions:** list and discuss several possible reasons for deviations between predicted and measured results from an experiment.

Appendix A
Introduction to Proteus

The PROTEUS Environment:

Proteus PIC Bundle is the complete solution for developing, testing and virtually prototyping your system designs based around the Microchip Technology series of Microcontroller. This software allows you to perform *schematic capture* and to *simulate* the circuits you design.

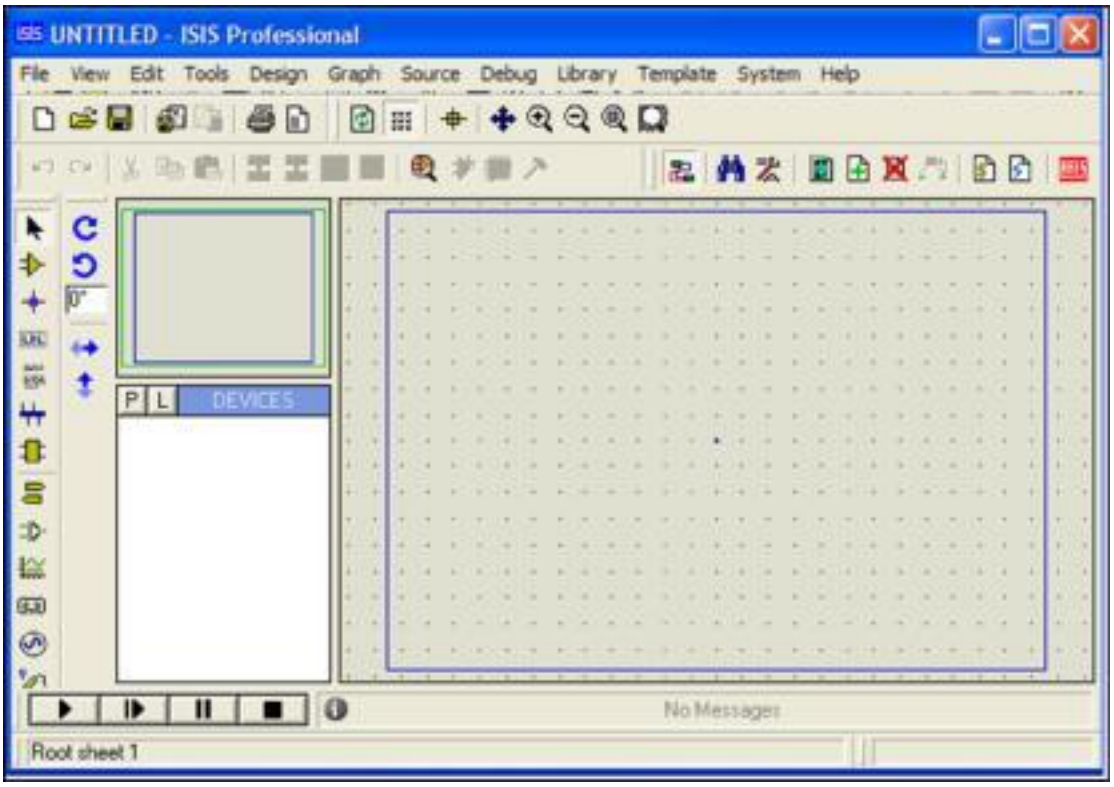

UNTITLED - ISIS Professional
File View Edit Tools Design Graph Source Debug Library Template System Help
P L DEVICES
No Messages
Root sheet 1

Figure 1: A screen shot of the Proteus IDE

Proteus How to Start Drawing the Circuit

Start a fresh design, select New Design from File menu then the Create New Design dialogue now appears as shown in Figure 2 and 3. Select Default and press OK .

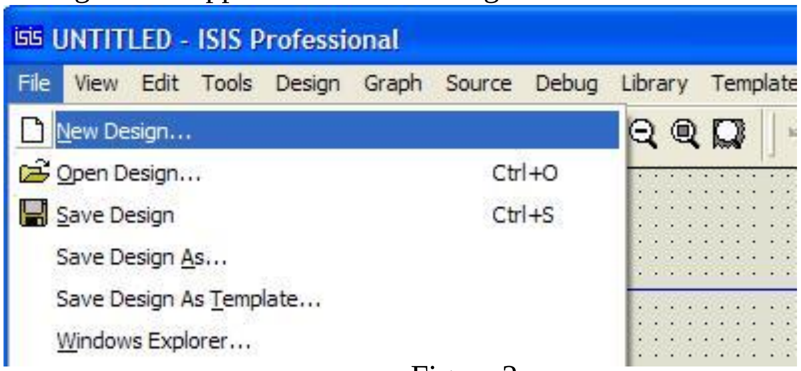

UNTITLED - ISIS Professional
File View Edit Tools Design Graph Source Debug Library Template
New Design...
Open Design... Ctrl+O
Save Design Ctrl+S
Save Design As...
Save Design As Template...
Windows Explorer...

Figure 2

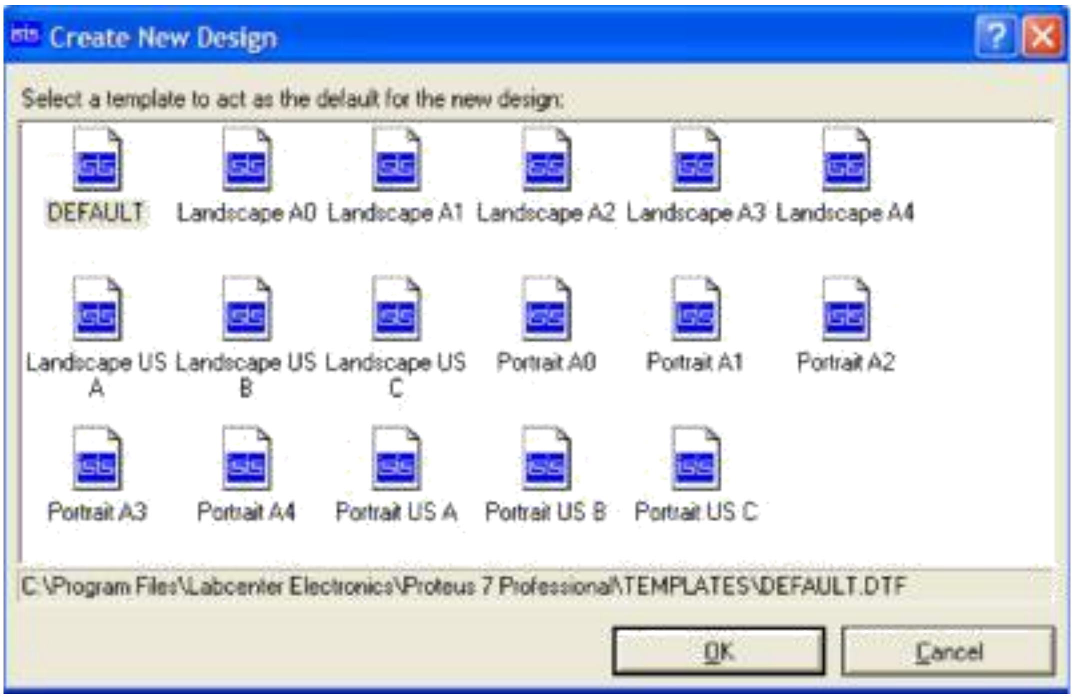


Figure 3

From the Library menu select Pick Device/Symbol see Figure 4 or left click on the letter 'P' above the Object Selector as shown in Figure 5 to launch the Library Browser or Press the 'P' button on the keyboard. The Library Browser will now appear over the Editing Window see Figure 6.

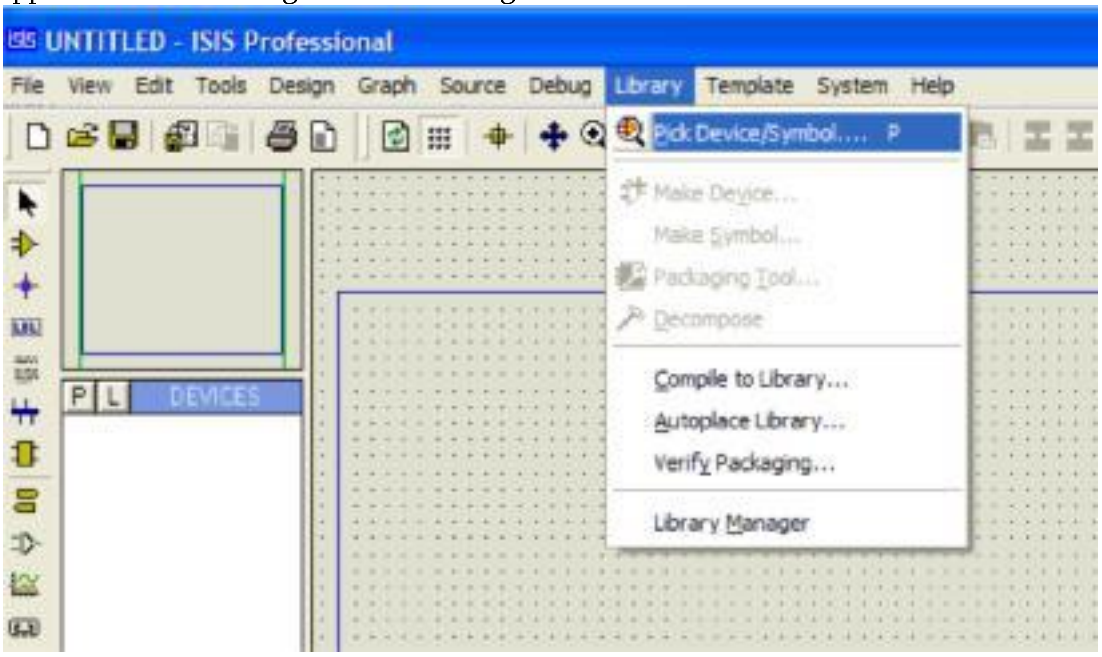


Figure 4

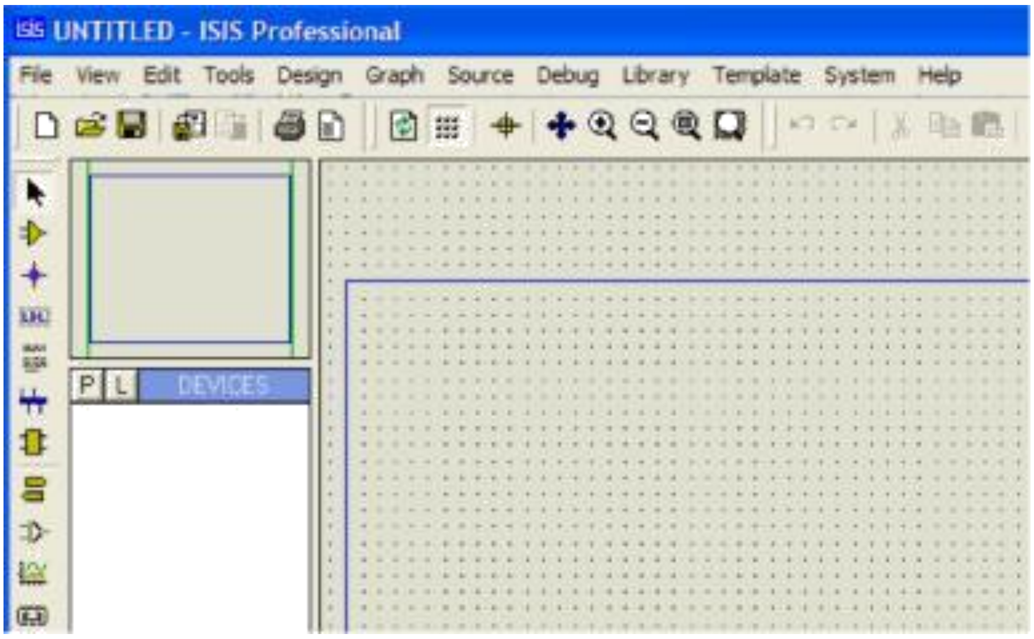
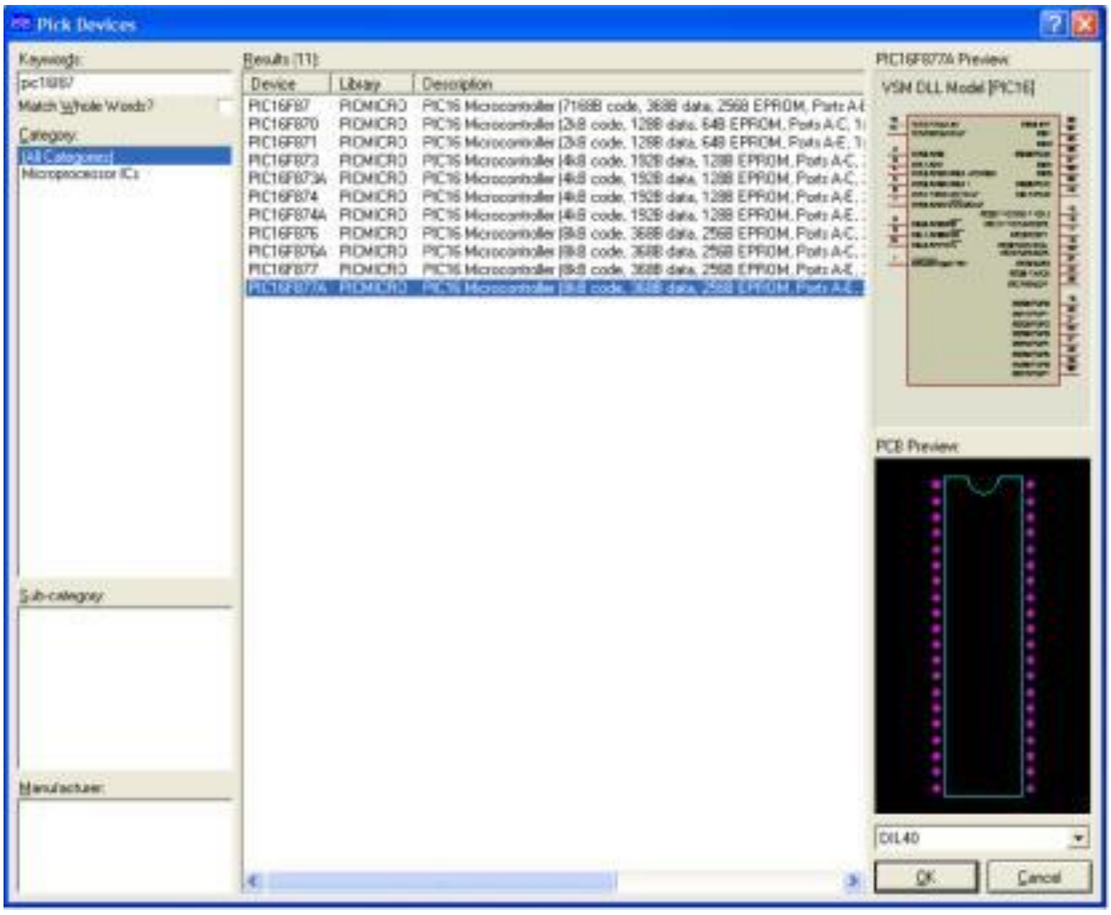

UNTITLED - ISIS Professional
File View Edit Tools Design Graph Source Debug Library Template System Help
P L DEVICES

Figure 5



Keywords: pic16f87

Match Whole Words?

Category:

[All Categories]

Microprocessor ICs

Sub-category:

Manufacturer:

Results (11):

Device	Library	Description
PIC16F87	PICMICRO	PIC16 Microcontroller (7168B code, 368B data, 256B EPROM, Ports A-E
PIC16F870	PICMICRO	PIC16 Microcontroller (2kB code, 128B data, 64B EPROM, Ports A-C, 1
PIC16F871	PICMICRO	PIC16 Microcontroller (2kB code, 128B data, 64B EPROM, Ports A-E, 1
PIC16F873	PICMICRO	PIC16 Microcontroller (4kB code, 192B data, 128B EPROM, Ports A-C,
PIC16F873A	PICMICRO	PIC16 Microcontroller (4kB code, 192B data, 128B EPROM, Ports A-C,
PIC16F874	PICMICRO	PIC16 Microcontroller (4kB code, 192B data, 128B EPROM, Ports A-E,
PIC16F874A	PICMICRO	PIC16 Microcontroller (4kB code, 192B data, 128B EPROM, Ports A-E,
PIC16F876	PICMICRO	PIC16 Microcontroller (8kB code, 368B data, 256B EPROM, Ports A-C,
PIC16F876A	PICMICRO	PIC16 Microcontroller (8kB code, 368B data, 256B EPROM, Ports A-C,
PIC16F877	PICMICRO	PIC16 Microcontroller (8kB code, 368B data, 256B EPROM, Ports A-E,
PIC16F877A	PICMICRO	PIC16 Microcontroller (8kB code, 368B data, 256B EPROM, Ports A-E,

Figure 6

Type in ' PIC16F877A ' in the Key words field and double click on the result to place the PIC16F877A into the Object Selector. Do the same for the LEDs, Buttons, Crystal oscillator, capacitors, 7 SEG-COM-Cathode, Resistors.
Once you have selected all components into the design close the Library Browser and left click once on any component in the Object Selector (This should highlight your selection and a preview of the component will appear in the Overview Window at the

top right of the screen see Figure 7). Now left click on the Editing Window to place the component on the schematic - repeat the process to all components on the schematic.

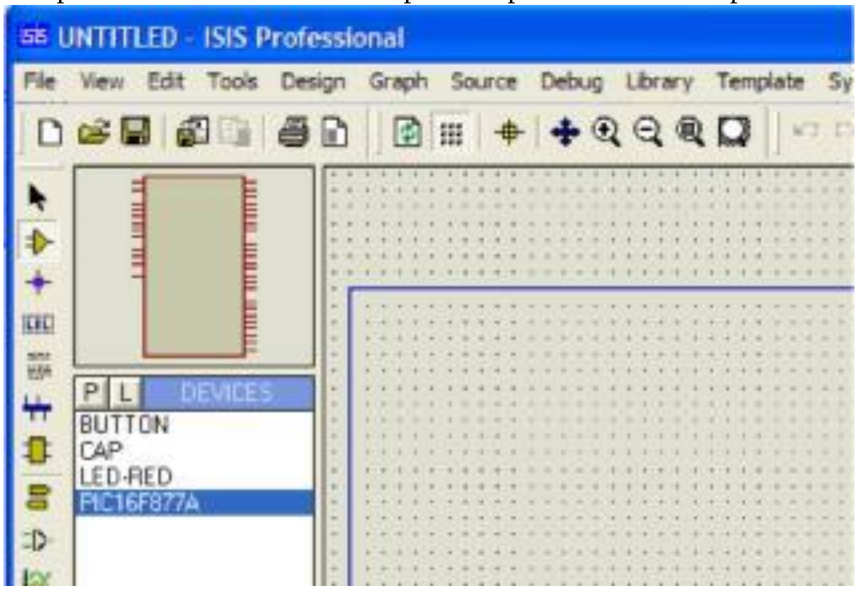

UNTITLED - ISIS Professional
File View Edit Tools Design Graph Source Debug Library Template Sy
P L DEVICES
BUTTON
CAP
LED-RED
PIC16F877A

Figure 7

In order to place ground or 5 voltage right click on the Editing Window, select place then terminal then select ground(0 V)or power (5V).Connect the components to obtain the circuit you need.

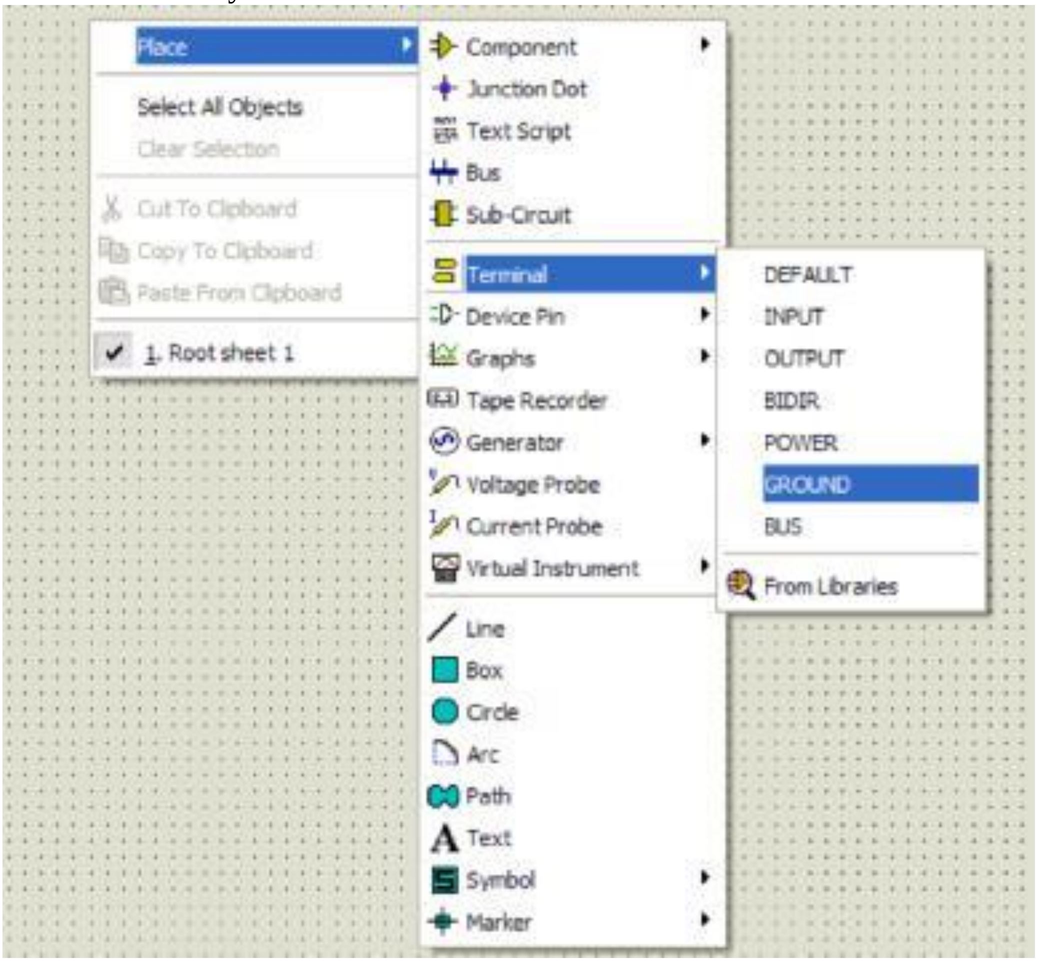


Figure 8

Attaching the Source File

The next stage is to attach the source file to our design in order to successfully simulate the design. We do this through the commands on the Source Menu.

Go to the Source Menu now and select the Add/Remove Source Files Command as shown down.

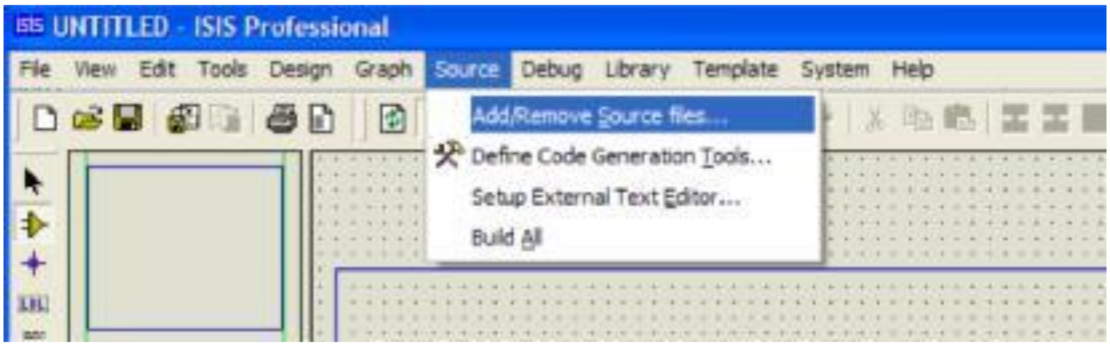

UNTITLED - ISIS Professional
File View Edit Tools Design Graph Source Debug Library Template System Help
Add/Remove Source files...
Define Code Generation Tools...
Setup External Text Editor...
Build All

Figure 9

Click on the New button, browse until you reach the desired directory and select your ASM file.

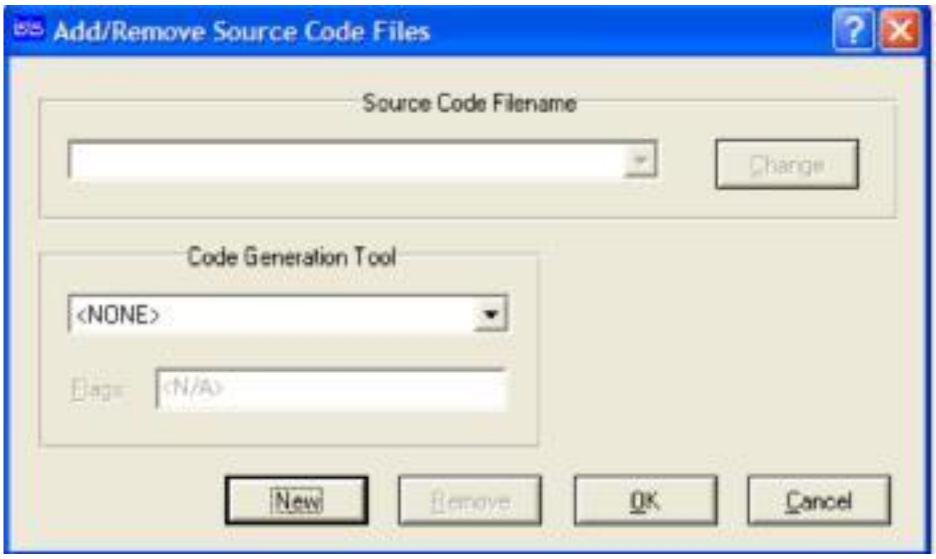


Figure 10

Click open and the file should now appear in the Source Code Filename drop down listbox.

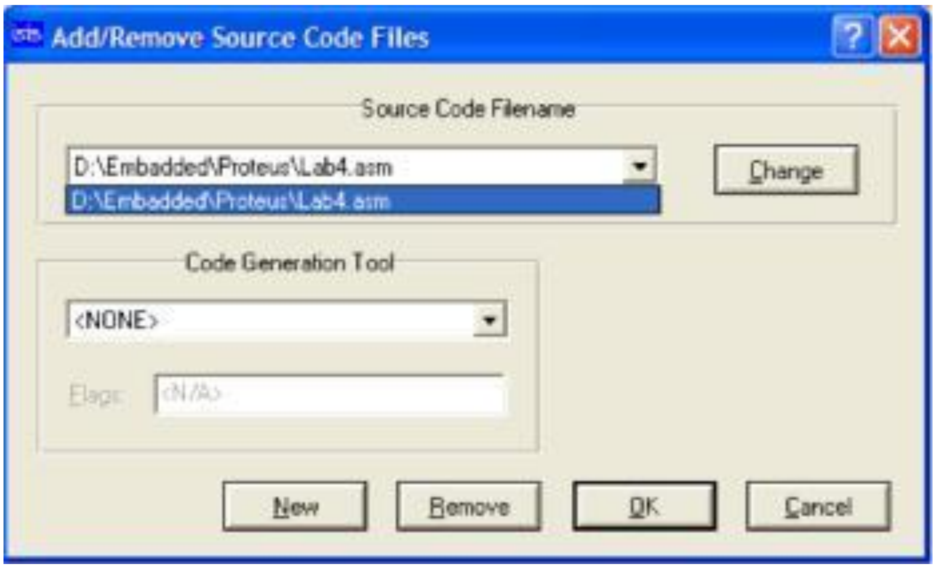
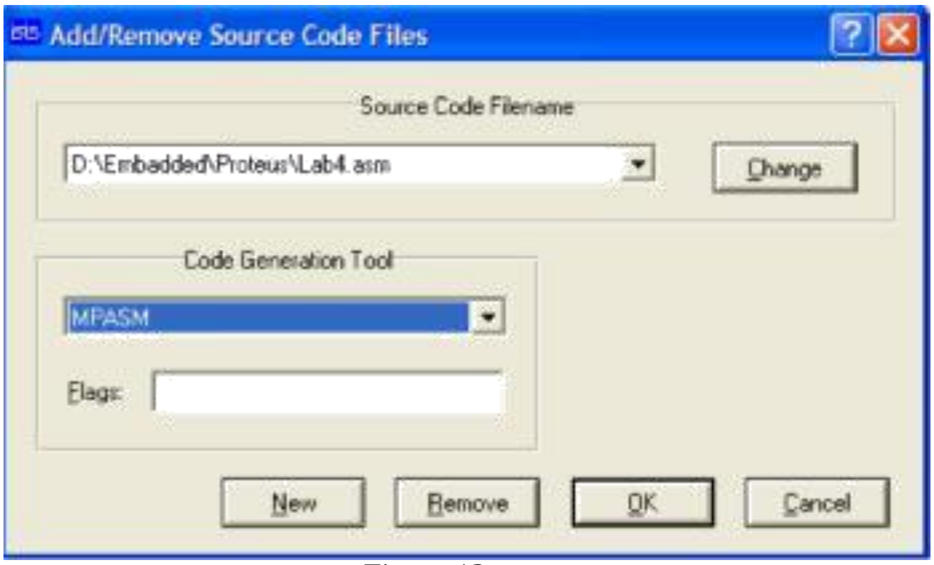

Add/Remove Source Code Files
Source Code Filename
D:\Embedded\Proteus\Lab4.asm
D:\Embedded\Proteus\Lab4.asm
Change
Code Generation Tool
<NONE>
Flags
<N/A>
New
Remove
OK
Cancel

Figure 11



Source Code Filename: D:\Embadded\Proteus\Lab4.asm

Code Generation Tool: MPASM

Flags:

Figure 12

We now need to select the code generation tool for the file. For our purposes the MPASM tool will suffice. This option should be available from the drop down listbox and so left clicking will select it in the usual fashion.

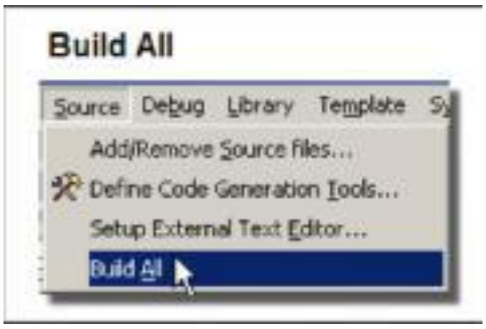
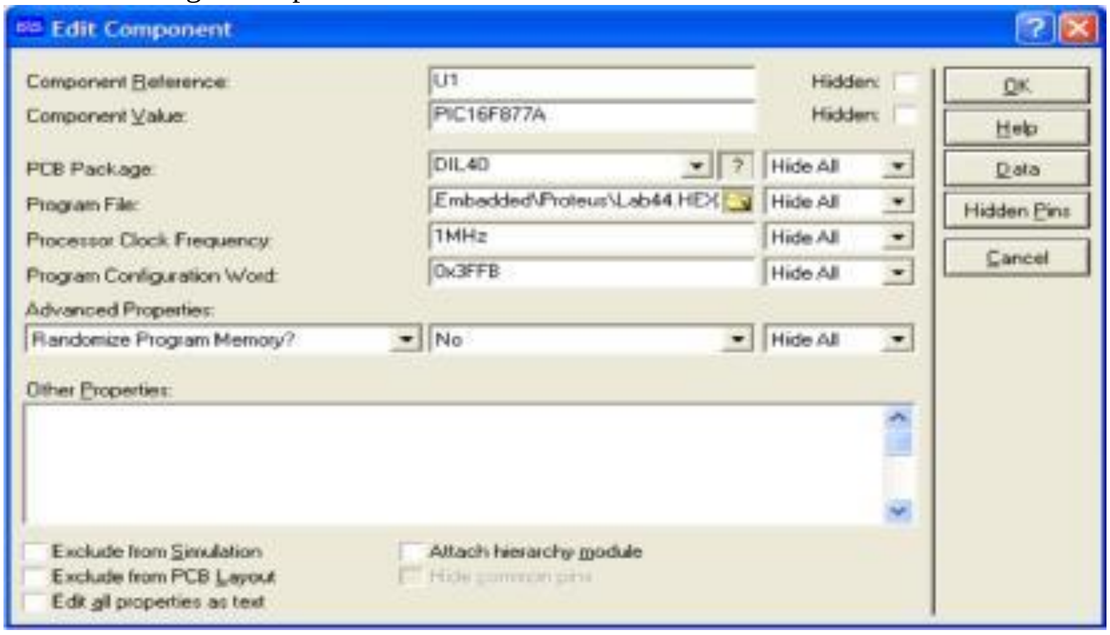

Build All
Source
Debug
Library
Template
Sy
Add/Remove Source files...
Define Code Generation Tools...
Setup External Text Editor...
Build All

Figure 13

Select the Build All command from the source menu as shown in Figure 13, this provides an excellent way to check that everything has built without errors or warnings.

Finally, it is necessary to specify which file the processor is to run. In our example this will befilename.hex (the hex file produced from MPASM subsequent to assembling filename.asm).To attach this file to the processor, right click on the schematic part for the PIC and then left click on thepart. This will bring up the Edit Component dialogue form which contains a field for Program File.If it is not already specified as filename.hex either enter the path to the file manually or browse to the location of the file via the button to the right of the field. Once you have specified the hex file to be run press ok to exit the dialogue form.We have now attached the source file to the design and specified which Code Generation Tool will be used.



Component Reference:	U1	Hidden:
Component Value:	PIC16F877A	Hidden:
PCB Package:	DIL40	Hide All
Program File:	Embadded\Proteus\Lab44.HEX	Hide All
Processor Clock Frequency:	1MHz	Hide All
Program Configuration Word:	0x3FFB	Hide All

Advanced Properties:

Randomize Program Memory?	No	Hide All

Other Properties:

Figure 14

Debugging the Program Simulating the Circuit

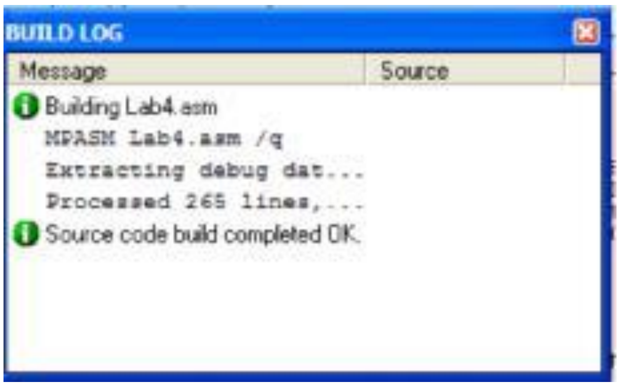
In order to simulate the circuit point the mouse over the Play Button on the animation panel at the bottom right of the screen see Figure 1 and click left. The status bar should appear with the time that the animation has been active for.

Working on your Source Code

To edit a source file:

- Select the source file it will be Filename.asm from the Source menu.
- Edit the source file.
- Select the Build All command from the Source menu.

You should see a small window showing details of the compilation process and the following message "Source code build completed OK".



```
Building Lab4.asm
MPASM Lab4.asm /q
Extracting debug dat...
Processed 265 lines,...
Source code build completed OK.
```

Figure 15

Appendix B
QL200 DEVELOPMENT BOARD

Overview:

QL200 PIC DEVELOPMENT BOARD (hereinafter referred to QL200) is a multifunctional PIC microcontroller development platform designed and developed by Shenzhen QSL Electronics Co., Ltd. based on our experience of many years' development and our efforts of months' design. It integrated the common external resources and simulation interfaces. Particularly it is suitable for self-learning for microcontroller beginners as well as electronic lovers. It has rich and flexible hardware resources and the expansion feature as well as the free resource allocation, which makes it a convenient debugging tool for PIC engineers to use especially in their beginning period of designs.

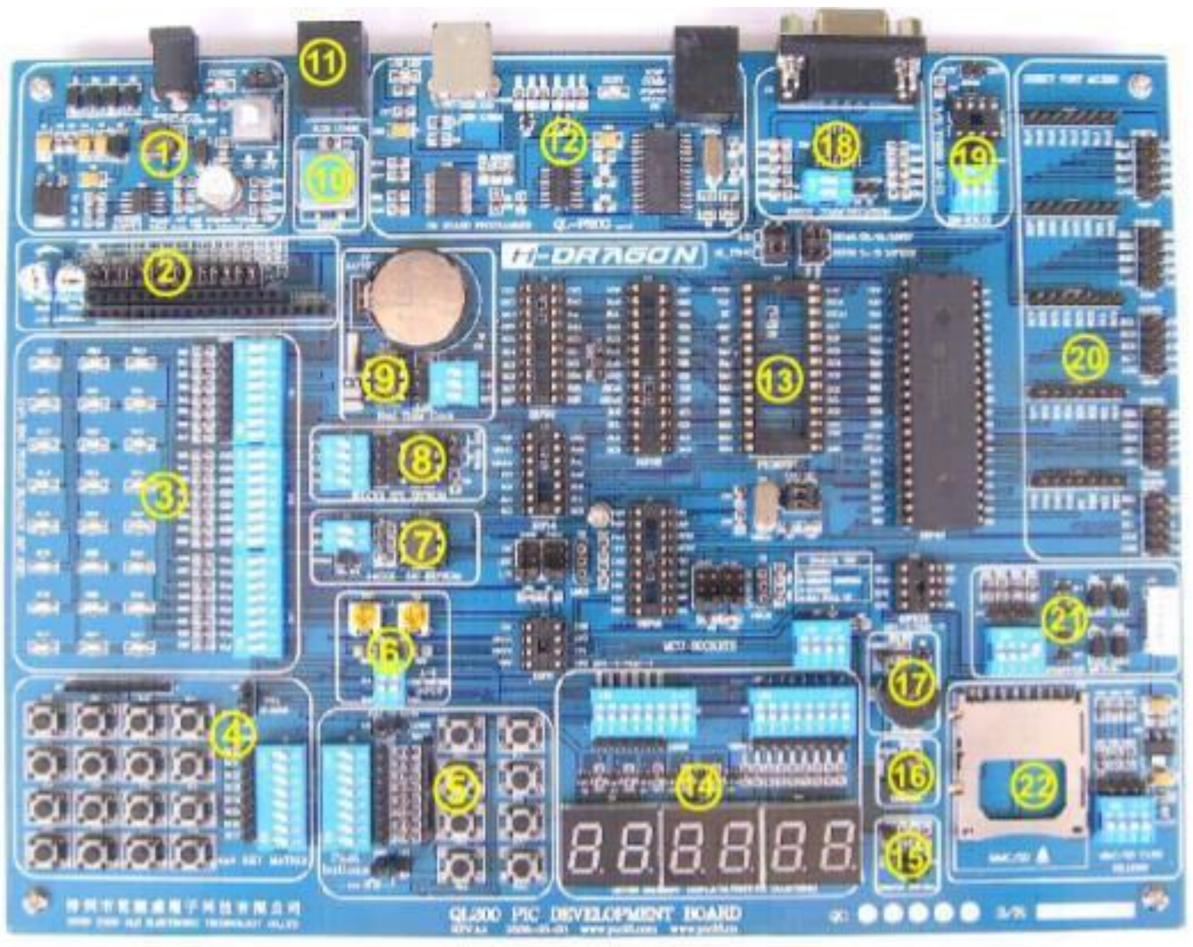


Figure 1:QL200 Board.

1) Power Module.
2) 12864LCD and 1602 LCD module.
3) LED I/O module.
4) 4x4 matrix keyboard module.
5) Button module.
6) A/D converter module.

7) IIC communication module.
8) SPI communication module.
9) DS1302 module.
10) MCU reset module.
11) ICD / MCD online debugging interface.
12) On-board programming module (USB Communication).
13) Chip socket and system clock selection.
14) Digital LED display with 6 bit and 7 sections.
15) Remote control receiver and decoder module.
16) DS18B20 digital temperature measurement module.
17) Beeper experimental module.
18) USART serial communication module.
19) 10-bit D/A converter module.
20) Port output module.
21) Stepper Motor Module.
22) SD/MMC card Read/Write Module.

QL200 programming module QL-PROG:

The QL-PROG and PC are connected through "USB COMM" interface for communication. The indicator BUSY keeps on during the whole process of programming, and after the completion of programming it automatically become off. Through ICSP interface, users can use QL-PROG to program other target microcontrollers in other boards. The programming module gets power supply from USB. During online programming the ICSP may not use external power, but the jumper wires of power module should be set to the USB position. Through jumper wires J5 and J6 to choose whether use QL_PROG to program target MCU or use MCD2 to debug MCU (with modules 11) as shown in Figure 4-3. Through jumper wires J7 and J8 to choose programming/debugging DIP40/28/18/16F57 chips or DIP20/14/8/10FXXX chips (because the PGD and PGC of DIP40 are RB7 and RB6 ports, while the PGD and PGC of DIP20 are RA0 and RA1 ports). As shown in the following Figures:

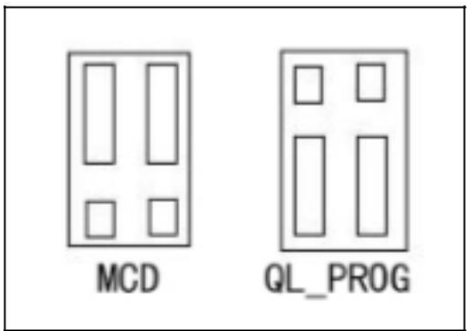


Figure 2: Choose MCD2 or QL-PROG

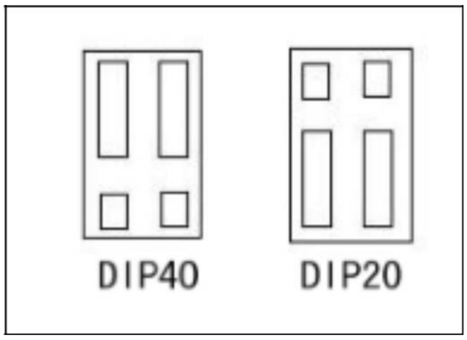


Figure 3: Choose DIP40 or DIP20

MCU Port Resources and System Clock Selection

This module is the core portion of QL200 development board, as shown in Figure 4. This module consists of the following main components:

1) 40-pin chip socket
2) PIC16F57 chip socket
3) 28-pin chip socket
4) 20-pin chip socket
5) 18-pin chip socket
6) 14-pin chip socket
7) 8-pin chip socket
8) PIC10FXXX chip socket
9) Clock source option OSCA (this clock source is used for DIP40/28 and PIC16F57).
10) Clock source option OSCB (this clock source is used for DIP18).
11) Clock source option OSCC (this clock source is used for DIP20/14/8).

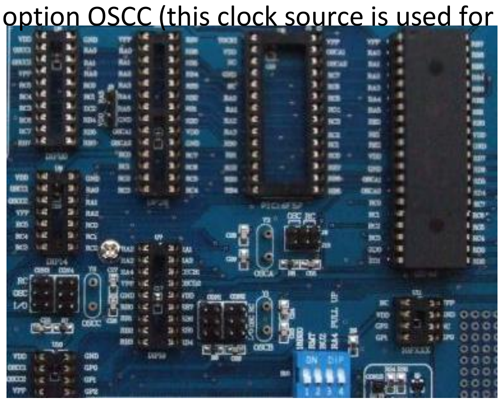


Figure 4: MCU socket and clock sources

About the details of pin resources of each chip please refer to its respective manual, and here we just list some specific pins used in this development board which needs the user to pay attention to.

1) Pin 8 of PIC10FXXX is used for VPP and MCLR, and cannot be used as I/O GP.
2) Pin 4 of DIP8 is used for VPP and MCLR, and cannot be used as I/O GP3.
3) Pin 4 of DIP14 is used for VPP and MCLR, and cannot be used as I/O RA3.
4) Pin 4 of DIP18 is used for VPP and MCLR, and cannot be used as I/O RA5.
4) Pin 4 of DIP20 is used for VPP and MCLR, and cannot be used as I/O RA3.

6) TOCKI PIC16F57 must connect to the power supply.

7) Pin 7 of DIP28 may connect to RA5 or VDD through selection of jumpers (Pin 7 of 18FXX31 is VDD, and Pin 7 of other 28-pin PIC is RA5).

8) Pin 2 and 3 of DIP8/14/20 can not only be used as clock input and output, but they can also be used as I/O port (in the condition that the internal clock mode is chosen at the point of clock selection.)

9) GPIO port of 8-pin chip are connected to other chips.

10) Pin 15 and 16 of DIP20 section can not only be used as clock input, but they can also be used as I/O port (in the condition that the internal clock mode is chosen at the point of clock selection.)

11) There's a 10K up-pull resistor on RA4 (which is controlled by the coding switch S10 next to the LED), for some chips the up-pull resistor must be activated so that the RA4 can output high level voltage.

System Clock Selections:

1. PIC10FXXX clock selection. PIC10FXXX can only use the Internal RC oscillation.
2. DIP8/14/20 clock selection. As shown in Figure 5.

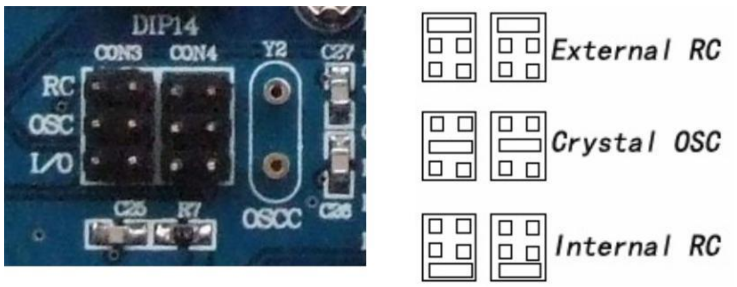


Figure 5: DIP8/14/20 Clock

3. DIP20 clock selection. As shown in Figure 6

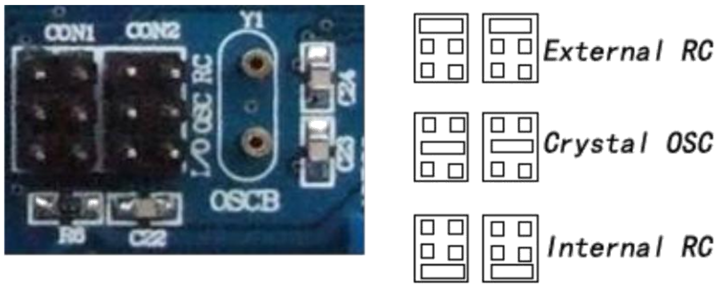


Figure 6: DIP20 clock selection

4. DIP40/28/PIC16F57 clock selection. As shown in Figure 7:

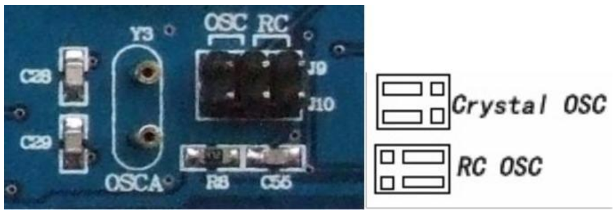


Figure 7: DIP40/28/PIC16F57 clock selection

12864LCD and 1602LCD Modules

12864LCD and 1602LCD modules are shown in Figure 5-7. The modules consist of the following components:

1) Character LCD1602 and contrast adjustment potentiometers.
2) Graphics array LCD12864, and contrast adjustment potentiometers.

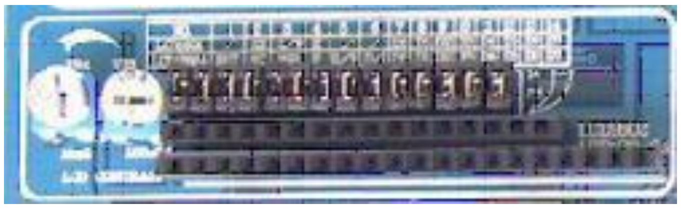


Figure 5-7 12864LCD and 1602LCD

Modules

1) 12864LCD and 1602LCD both use the PORTA as the control bit and the PORTD as data bit.
2) All the pins of this module are jump connected to the pins of MCU without coding switches to control it. It is recommended that the LCD is plugged off the socket.
3) In the experiment of LCD1602 or LCD12864, the up-pull resistor of RA4 required to be activate, i.e., set the S10_4 ON.
4) Datasheets of 12864LCD and 1602LCD are included in the Datasheet directory of CD-

ROM that comes along with product.

5) Examples of 12864LCD and 1602LCD are included in the Example directory of CD-ROM that comes along with product.

Schematics of these modules are shown in Figure 9:

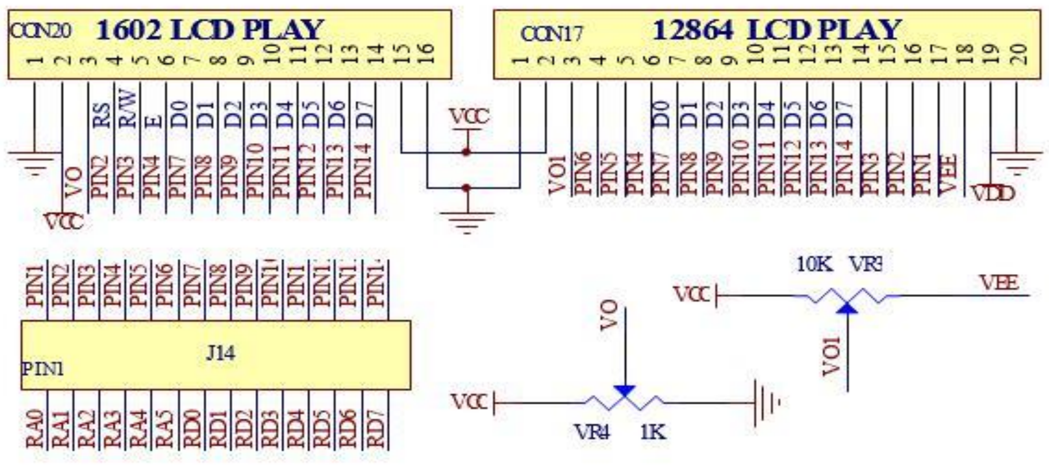


Figure 9: Schematics of 12864LCD and 1602LCD

PORTA, I PORTB and the LED module PORTC

The module shown in Figure 10 and 11 consists of the following components:

1) 8 LED lights on PORTA, and PORTA of PIC16F628 etc. has 8 bits.
2) 8 LED lights on PORTB.
3) 8 LED lights on PORTC.

1) All the three ports have respective 8 bits output
2) All the three ports use separate coding switch to control them. It is recommended that if this module is not in use, the corresponding bits should be shut off to avoid interference with other functions.
3) Through the LED socket of PORTC, you can conduct the experiment of other ports (such as PORTD and PORTE of 40-bit MCU). However, when this is in use, please make sure that the coding switch must be switched off.
4) All LED are grounding with negative pole, and if you want to lit the LED, just output a high level voltage at the corresponding pin of the port.
5) In the CD-ROM examples of "single LED light" and "play-in-turn" are included for reference.

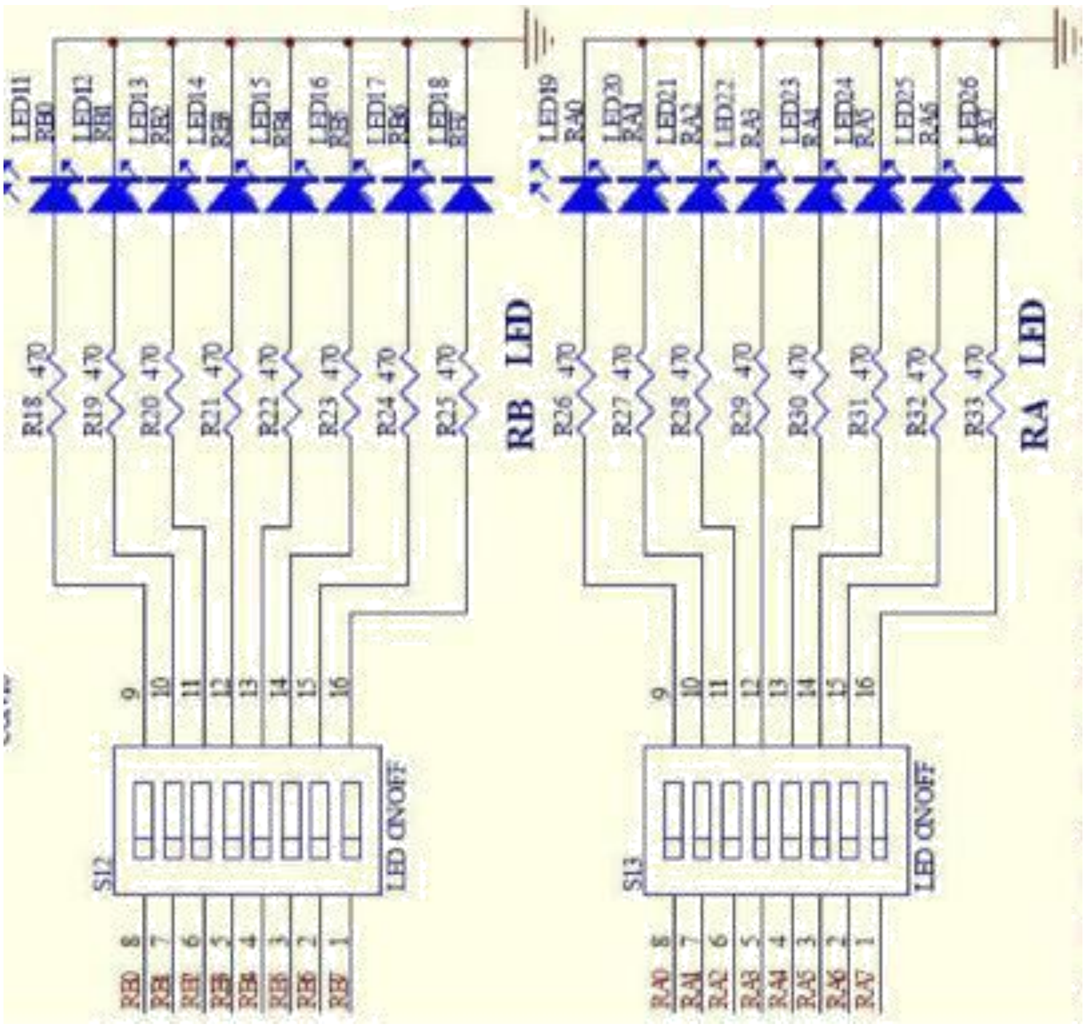


Figure 10: Ports A/B LED schematic

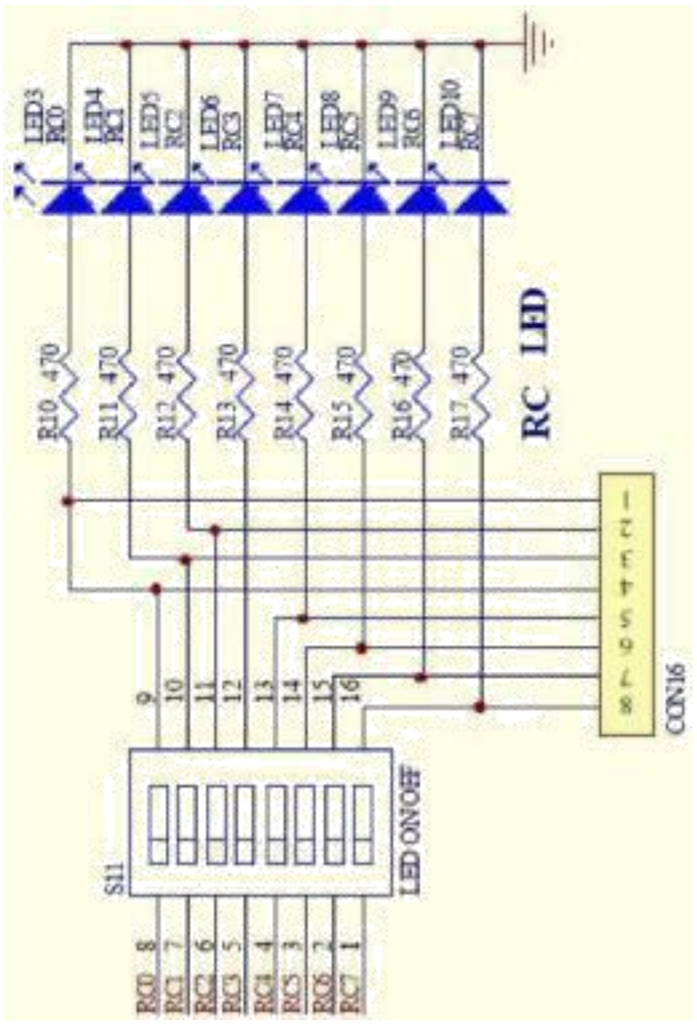


Figure 10: PortC LED schematic

4x4 matrix keyboard

This module is shown in Figure 11. It consists of the following main components:
1) 16 keys.
2) Coding switching.
2) Interface socket.
2) Up-pull resistors.

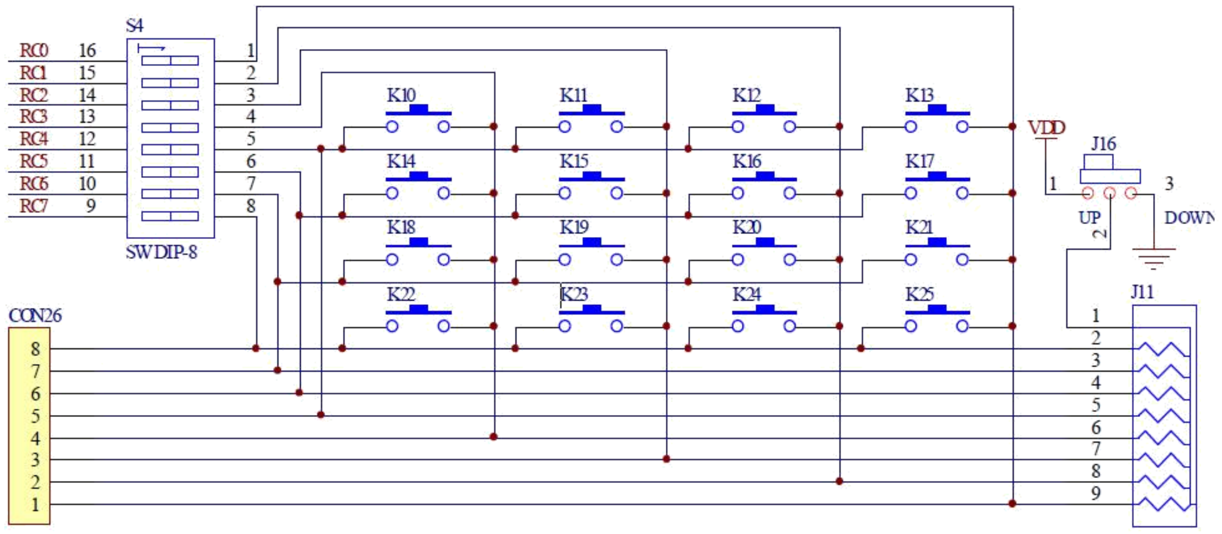


Figure 11: The schematic of the 4x4 matrix keyboard

Descriptions about this module are as the following:
1) Keys are connected to the 8 bits of PORTC in 4 x4 array
2) This module uses coding switch to control it. It is recommended that if this module is not in use, the corresponding bits should be shut off to avoid interference with other modules.
3) Through the interface socket you may practice the other matrix keyboards (you must make sure the coding switch is off).
4) There's a 10K up-pull or down-pull resistors to ensure the stability of voltage level of data online.

PORTA0~3 and PORTB0~3 Key Module

This module is about single key functions.
This module consists of the following components:
1) 4 keys on PORTA0 ~ 3;
2) 4 keys on PORTB0 ~ 3;
3) 10K up-pull or down-pull resistors on 8 pins, and jumper select.
4) Coding switch
5) Interface socket

Descriptions of this module are as the following:
1) 8 keys are connected to the PORTA0 ~ 3 and PORTB0 ~ 3.
2) When the keys are pressed, the output level of voltage is LOW or HIGH (jumper J17 select)
3) The 8 keys are controlled by the coding switches to decide whether they are connected to the pins of MCU, and when this module is not in use please make sure the coding switches are not connected to avoid the effect on other modules.
4) Through the interface socket, you can do other output experiment (with the coding switches disconnected).
5) In case the keys on PORTB0 ~ 3 are used, if the internal up-pull resistors are activated, you may use the jumper wire to switch them off to disconnect them from the internal up-pull resistors. But for the keys on PORTA0~3, in order to keep the stable voltage level, the jumper wires must always be ON.

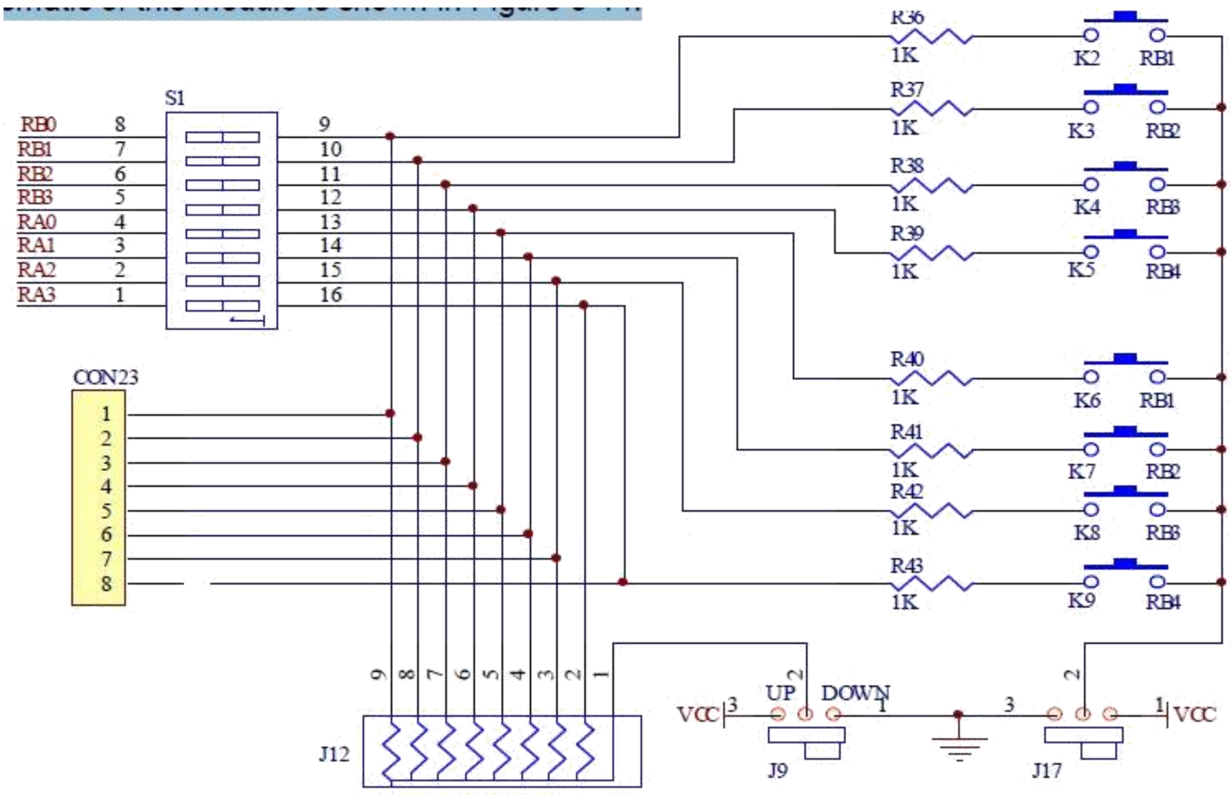


Figure 12: Schematic of the keys

Six Digital LED Module

This module introduces the use of a multi digital LED, the module consists of the following main components:
1) 6 digital LED
2) Coding switches for bit control and section control
3) Driving circuit
4) Interface socket

Descriptions of this module are as the following:
1) The section control of LED is connected to PORTD of MCU through coding switch.
2) The bit control of LED is connected to PORTA of MCU through coding switch. When this module is in use please make sure that the coding switch are on, and when this module is not in use please make sure the coding switches are off to avoid affect the other modules.
4) By using the interface socket, it is also possible to achieve the display of LED with other MCU chips. (Please make sure that the coding switches are off.)
5) The six digital are connected to a common anode pole.

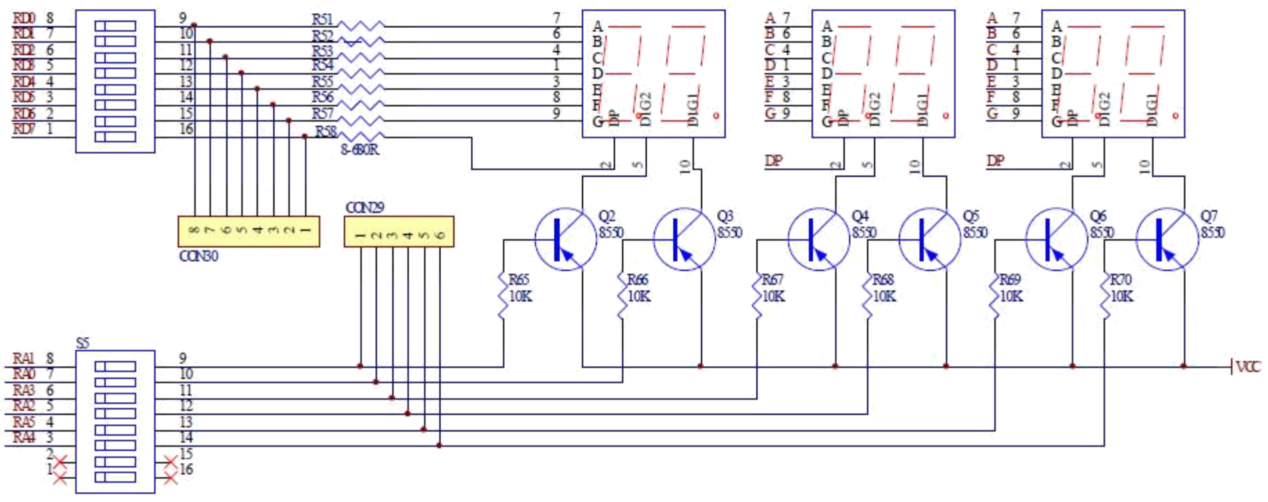


Figure 12: 7-segments schematic