

Final Project

Memory Game

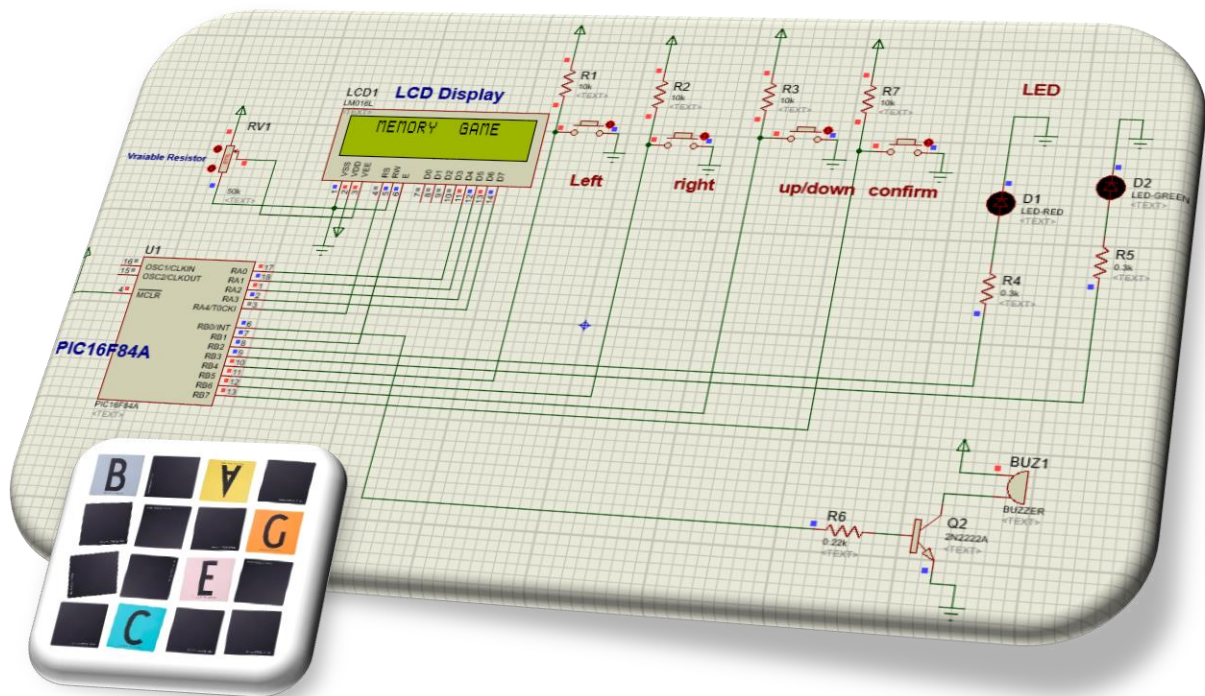


Table of Contents

I.	INTRODUCTION	3
II.	MEMORY GAME OVERVIEW	3
III.	OBJECTIVES	4
IV.	PROJECT DESCRIPTION	4
A.	INPUTS.....	4
B.	OUTPUTS.....	5
C.	STARTUP	5
C1.	<i>Main setting and Initializing.....</i>	<i>6</i>
C2.	<i>Saving The Default Board.....</i>	<i>6</i>
C3.	<i>Setting A Welcome Screen.....</i>	<i>7</i>
C4.	<i>RB interrupt</i>	<i>8</i>
D.	MAIN MENU.....	8
E.	MODE 1.....	9
E1.	<i>Mode 1 Left Push button</i>	<i>10</i>
E2.	<i>Mode 1 Right Push button.....</i>	<i>10</i>
E3.	<i>Mode 1 Up/down Push button</i>	<i>11</i>
E4.	<i>Mode 1 Confirm Push button</i>	<i>11</i>
E5.	<i>Case of non-matching</i>	<i>13</i>
E6.	<i>Non-matching bar meter.....</i>	<i>13</i>
E7.	<i>End game.....</i>	<i>15</i>
F.	MODE 2.....	16
F1.	<i>TMRO interrupt.....</i>	<i>18</i>
G.	MODE 3.....	19
V.	SOFTWARE DESCRIPTION	21
A.	USEFUL HINTS	21
VI.	SOFTWARE SIMULATION	22
VII.	PROGRAMMING THE PIC16F84A	23
VIII.	HARDWARE IMPLEMENTATION.....	24
A.	HARDWARE.....	24
B.	COMPONENTS	25
C.	COMPONENTS DESCRIPTION.....	26
C1.	<i>Power Supply.....</i>	<i>26</i>
C2.	<i>Clock</i>	<i>26</i>
C3.	<i>Buzzer</i>	<i>26</i>
C4.	<i>Switches.....</i>	<i>27</i>
IX.	GRADING CRITERIA AND DELIVERABLES	27
X.	BONUS	28
XI.	IMPORTANT NOTES	28

I. INTRODUCTION

Memory game is card game in which all cards are laid face down. Two cards should be flipped face up over each turn. This game can be played with one or more players. The objective is turn over pairs of matching cards. If non-matching cards were turn face up, then both cards will be flipped face down again. If the two pairs were matching, then the cards will be kept face up and the player will gain a point which will be added to his/her score. Rules of these games may vary from a version to another. They can also have more complex scoring rules.

In our project we will design a small memory game (few number of cards) of three different versions. Each version (mode) will have its own rules such as: how to score or how to end the game.

II. MEMORY GAME OVERVIEW

In our project, we will design a Memory Game (MG) that consists of 6 matching pairs. So, total of 12 cards are available. The cards will be labeled alphabetically from A to F. So, we will have two cards labeled as A, 2 as B, 2 as C, 2 as D, 2 as E, and 2 as F.

In mode 1: the game will end only if a player makes all matching pairs. So, the player will keep playing until he/she makes all the available matches (6 matches in our case). Then, depending on the player's performance (number of non-matching were done during the game), the player will get a feedback as Super, Average, or Weak. In addition to that, a bar-meter will be shown on the screen (which is related to number of non-matching) to show the player performance after each turn. The meter will grow from Super (lower number of non-matching) to Weak (higher number of non-matching) and it will be updated after each turn. More description will be provided later.

In Mode 2: the game will be time dependent. The player should make matching pairs before the end of the game time. Then, the final score will be provided as a combination of the remaining time and the number of matches done. Also, in this mode, the game ends whenever the player makes all available matches or the time of game ends.

In mode 3: the game will have different scoring techniques. An initial score will be set. Then every match, a point will be added to this score. After every non-matching pair, the system will check if the card were revealed before (opened more than one time and failed to do a match), then a point will be deducted from the score for every re-opened revealed card. Here, the game will end if the player makes all the available matches or the player reaches the max number of opened revealed cards.

III. OBJECTIVES

In this project, a group consisting of **three students** will design and implement the “MG” of a given specifications (given to you) with an LCD display using PIC16F84A microcontroller.

The project consists of software and hardware components. The software component includes developing an assembly code that describes the functionality of the “MG” to be downloaded on a microcontroller.

The hardware component requires building the “MG” design by connecting various components including inputs (push buttons), outputs (LCD, buzzer, LED) and the programmed microcontroller (**A PCB Board will be designed for this purpose**). Finally, the design should be tested to validate the design requirements and specifications.

IV. PROJECT DESCRIPTION

The “MG” is equipped with an LCD for display, two LEDs, four push buttons, and a buzzer. The inputs and outputs to the microcontroller are summarized as follows:

A. INPUTS

The inputs of the microcontroller are the inputs of the “MG” itself. We have four inputs;

- A push button (**left**), used to move one step to the left side between the different cards shown on the screen.
 - A push button (**right**), used to move one step to the right side between the different cards shown on the screen. In addition to that, the right push button will be used to switch between different options (modes of the game) in the main menu screen (will be described later).
 - A push button (**up/down**) used to move between cards either by going down (second row) or going up (back to the first row).
 - A push button (**Confirm**), used in all mode to open a card and it is used in the main menu to confirm the choice (mode 1, 2, or 3).
- The inputs of the controller **should** be assigned to the following pins of the PIC16F84A:

Input	PIC pins
Left	RB4
Right	RB5
Up/down	RB6
Confirm	RB7

Table IV-1: Inputs Memory Game

B. OUTPUTS

The outputs of the controller are the LCD, two LEDs, and a buzzer. The LCD should be used in 4-bit mode and hence 4 pins of the microcontroller are needed to connect to the LCD data bus. Furthermore, additional output pins are needed for RS (register select on LCD), LCD Enable, LEDs, and a buzzer. The outputs are summarized in table IV-2.

	Output	PIC pins
LCD	LCD DB(4-7)	RA(0-3)
	RS	RA4
	Enable	RB1
RED LED	LED	RB2
GREEN LED	LED	RB3
Buzzer	Buzzer	RB0

Table IV-2: Outputs of Memory Game

The red LED, the green LED, and the buzzer will be used in different modes to indicate a match, or an error... Follow the description of each mode later.

The LCD is used in all modes and menus. Follow the description.

C. STARTUP

At the startup of your program, you have to set the initial setting of the program. These settings should/ or might include the following:

- Origin of the program
- Origin of the interrupt service routines
- The Main: configuring inputs and outputs, configure some special function register in PIC16F84A, initializing inputs and output, initializing registers, flags...
- Initializing the LCD into 4 bit mode
- Setting a welcome screen
- Saving a default board (the location of each card in memory)
- Setting the main menu screen
- Enabling interrupts
- Construction of an infinite loop

Refer to sections C.1, C.2, C.3, and C.4.

C1. MAIN SETTING AND INITIALIZING

Your program should start with a header section and the declarations section which consists of a number of equate directives that the programmer can use to assign values to symbolic names.

At ORG 0x00, the program should go to the start of your program (START) and at 0x04; the interrupt service routines should be defined. The interrupt service routines are RB vector interrupt and TMRO interrupt.

You should start your program by defining the input and output ports, and then initializing the LCD into 4-bit mode (you should write a Toggle and 40 msec (or less) delay subroutines). Note that the 40 msec delay should be used after power up of the LCD (refer to lab 7), but you can use less delay in the toggle function (ex. 10 msec) in order to write in a faster way on the LCD.

After that you should start initialization routine where you initialize all your registers to be used and all the initial settings.

After the initialization routine, interrupts should be enabled and the corresponding registers should be set (OPTION register and INTCON flags and Enables).

You should write very structured code with multiple routines each of which can act as a separate module on its own that can be called many times when needed. By this modular programming, each block can be tested on its own by software simulation and hardware after downloading the program. Also, this minimizes the number of code lines (instructions) to be written. **Note that PIC16F84A program memory can hold up to 1024 instructions.**

C2. SAVING THE DEFAULT BOARD

We need some scheme of a database in order to save the default board that contains the location of each card. It is advisable, to save this board in 12 different registers. So, each register will contain a certain number of bits that will define the card (the characters from A to F) and some bits of the same register can be kept for later use such as: keep track of opened card, revealed card, matched...The default board should be as follows (**Note that you should have the same default board or your project will not be corrected.**). The board consists of 2 rows. Each row contain 6 cards.

C	A	E	F	B	D
B	D	F	C	A	E

Recall that PIC16F84A is an 8-bit microprocessor. So, all data or General Purpose Register are 8 bit wide. The first register can be used in PIC16F84A is the register at location 12 (which is the first General purpose register GPR). It is preferred to save all these register in sequential order since you might be using **in-direct addressing** later in order to print, check, and compare the status of each register.

In Direct Addressing Mode

Each General Purpose Register (GPR) is 8-bits wide and is accessed either directly or indirectly through the FSR.

FSR is an in-direct Data Memory Address Pointer

INDF is an 8 bit register that uses the contents of FSR to address Data Memory (not a physical register, you cannot watch its value using watch window. You can read this register or you can write on this register). Refer to section 2.5 in PIC16F84A Manual

EXAMPLE 2-1: INDIRECT ADDRESSING

- Register file 05 contains the value 10h
- Register file 06 contains the value 0Ah
- Load the value 05 into the FSR register
- A read of the INDF register will return the value of 10h
- Increment the value of the FSR register by one (FSR = 06)
- A read of the INDF register now will return the value of 0Ah.

EXAMPLE 2-2: HOW TO CLEAR RAM USING INDIRECT ADDRESSING

```
        movlw 0x20    ;initialize pointer
        movwf FSR     ;to RAM
NEXT    clrf INDF     ;clear INDF register
        incf FSR      ;inc pointer
        btfss FSR,4   ;all done?
        goto NEXT     ;NO, clear next
CONTINUE
        :              ;YES, continue
```

C3. SETTING A WELCOME SCREEN

The first screen to be displayed is a “welcome screen” of your choice (or display Memory Game). Make sure that you use minimum characters or use same characters of the other screens (described later in the manual). This is because you have limited lines of code in PIC16F84A and we have to free them for other functions and instructions (**1024 line of code – you can check these lines after building your project using MPLAB and then going to view → program memory**).



Figure C-1: A welcome screen

Don't forget to add some delay after displaying the welcome screen to allow the user read your welcome screen easily. After that, you have to clear the screen (using Clear Display instruction of the LCD- Refer to LAB 7) and go to main menu screen. This screen should include the different options of the game: Mode 1, 2, and 3 (Refer to the figure below)

Note that you should follow the same naming, order, and locations on the LCD as shown in the figure below:

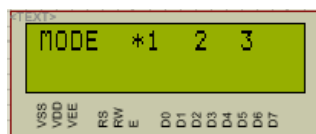


Figure C-2: Main Menu Screen

The word mode should be displaying at the beginning of the first line of the LCD. A pointer should be displayed just before the first choice as shown in the figure above. You can use the * symbol or any other symbol. Numbers 1, 2 and 3 indicate the different modes of the game.

At this step, you can enable interrupts of **RB47 port** and go to an **infinite loop**. The purpose of this loop is to keep the microcontroller running and waiting for interrupts.

C4. RB INTERRUPT

We will be using RB4 – RB7 as a source of interrupts that will be connected to 4 push buttons as listed in **section A**. Because we are using mechanical switches, then you have to **De-bounce** any switch before taking any action. Then you have to check the source of interrupt (RB4, 5, 6, or 7). This can be done by checking the corresponding bit if set or cleared. Note that RB4-7 interrupt occurs on any change of any port 4, 5, 6, or 7. Also, it occurs on both rising and falling edge of the button. As per our hardware connections (described later), the initial value of any push button is high. When the user press on the push button, it will generate a low signal (zero). So, in your program, you should respond to falling edge on ports 4 to 7 and discard the rising edges.

D. MAIN MENU

At the main menu, the user will be allowed to choose between the three different modes of the game. To do so, the user will be using two push buttons. The push button “right” will be used to flip between the three choices and the push button confirm will be used to confirm the choice.

Pressing “right” push button, should move the pointer (* character or any other character) from the initial location on the screen (just before 1) to the new location just before 2. **Note that you should follow the same naming, order, and locations on the LCD.**

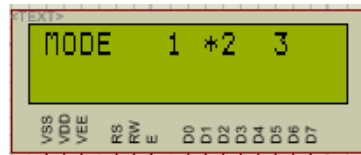


Figure D-1: After pressing right for the first time

Another press of “right” push button, the pointer should move from second to third position (just before 3).

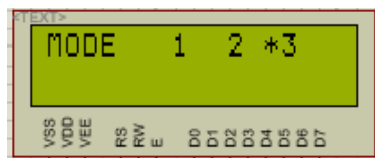


Figure D-2: After pressing right for the second time

Now, pressing “right” push button again, the pointer should go back to the initial position (just before 1) and the user can loop again between the choices indefinitely.

The next step after selecting the mode, the user should press the “confirm” push button in order to confirm the choice and go to the next screen. Confirming the choice of any mode will terminate the main menu screen and go the next screen (Mode 1, 2, or 3 screen).

Also note the user can directly press on confirm push button without moving the pointer. In this case, mode 1 will be confirmed.

Note that the other two push button (left and up/down) should be **disabled** at this step. So, pressing on any of these push buttons should be discarded. This can be done using a certain flag that is set or cleared.

You can always use some flags to keep track of different menus, modes and choices done especially because we will be using the same push button in different menus or modes and for different functions. A flag can be any bit of temporary register that can be set or cleared (use BSF or BCF).

E. MODE 1

The initial screen of this mode should be as follows:

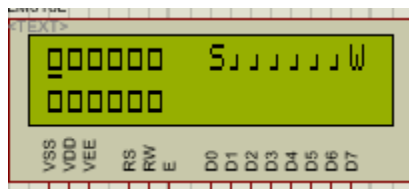


Figure E-1: Mode 1 screen

12 squares representing the cards (6 on the first line starting from the first address of the screen and the other 6 on the second line). The bar-meter which contains: S representing Super and W representing Weak. 6 symbols J as shown in the figure above between S and W. This meter will be updated by replacing the symbol J by the symbol ■ (filled square) starting from the first symbol near S till the last symbol (just before W). This bar-meter depends on the number of non-matching pairs (described in detail later).

The cursor of the LCD should be ON in this mode and it is at the first location of the screen.

Hint: Even at this step printing these squares can be done using a function that print the cards from memory. This function can print the square symbol if the card is face down (closed) and print the character symbol if the card is face up (opened). This function can be used a lot in your project, and this will minimize the number of lines to be coded.

Now, the player can use all the four push buttons to play the game.

E1. MODE 1 LEFT PUSH BUTTON

As described earlier, the left push button will be used to move between the cards one step to the left. Since initially the cursor is at the first position (position 1, address 0 on the LCD), pressing left should generate an error since the cursor is at the first position and it cannot be moved left.

E.1.I CASE OF ERROR OF LEFT

If the user press on left push button and the cursor is at position 1 of the first or second line, then an error should be generated. The error will be generating a small buzzer sound.

E.1.II NORMAL CASE OF LEFT

The normal case of left is to move the cursor of the LCD one step to the left side. This should be applied on both line 1 and line 2. Note that it is very important to create a function that will keep track of the address of the LCD. Later this address can be related to memory (for example address 0 on the LCD is the card saved at address XX in the memory of the PIC16F84A). The Address function can take the value of any temporary register holding the address and then updating this address on the LCD accordingly. It also can take into consideration the difference between the address line 1 or 2.

E2. MODE 1 RIGHT PUSH BUTTON

As described earlier, the right push button will be used to move between the cards one step to the right. Since initially the cursor is on the first position (position 1, address 0 on the LCD), pressing right should move the cursor to the next position.

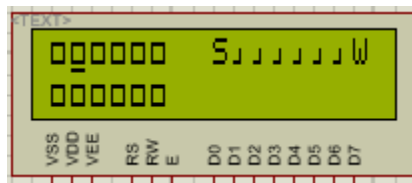


Figure E-2: after pressing right for the first time

Pressing right again and again will keep moving the address one step to the right till it reaches the boundary of the board on the right side. In this case, if the user press right again it should generate an error. The same type of error indicator will be used as in the left function which is generating a short buzzer sound.

Also, note that right function have the same concept if the cursor is at line 2 of the LCD.

E3. MODE 1 UP/DOWN PUSH BUTTON

The up/down push button will be used to toggle between the lines of the LCD (flipping between line 1 and 2). So, if the player press on this button and the cursor were on line 1, then the cursor should be update to line 2 (the address just below the previous address). For example, if the cursor is at position 2 in line 1 and then the player press on the up/down push button, the new cursor position will be on line 2 position 2 (figure below).

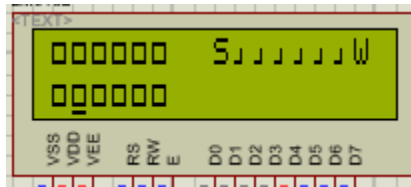


Figure E-3: pressing up/down button

Now pressing up/down button again, the cursor will go back to line 1, position 2 (same as old).

Note that this will be applied to all address in line 1 and 2 and the above description is only an example of specific address.

The up/down function cannot have any error. The player will be always allowed to toggle between the two lines 1 and 2.

E4. MODE 1 CONFIRM PUSH BUTTON

The confirm push button is responsible for opening a card in the playing mode.

E.4.1 OPENING FIRST CARD

When a player presses on confirm push button, the card must be opened if it is closed (face down). So, first we need to check if the card is opened before as a result from a previous match. Then, if the card was closed, you need to open it by printing the corresponding character in the corresponding position on the board. Then, you need to reset the address (cursor) to the first position of the board (first line, first position). You need to make sure that the card is marked as opened, so the user cannot open it again when it is shown on the screen. Hint: you can do any relation between the address of the LCD and the memory location of the corresponding card. Here In-direct addressing will help a lot. Also, it is very important to set a certain flag that will keep track if this is the first time confirm is pressed (first card of a pair to be opened).

Below is an example: The cursor now is at the same position as in the previous figure (second line, second position). The player press on confirm push button. Now since this card is not opened, then we need to open it and go back to the first address. Next figure shows the result on the LCD display (Letter D is displayed and Cursor at first position):

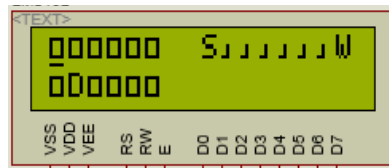


Figure E-4: opening second card in line 2 of the board

E.4.I OPENING SECOND CARD

Now, suppose that the player needs to open a second card. In your program at this stage, you should have updated a certain flag that indicated first card is opened and next confirm is for second card. Suppose that the player goes to the same location as before (second line, second character) and press on confirm. Now, since D is opened as a result of the first confirm, your program should generate an error. The error will be a short buzzer sound. In addition to that, you need to make sure that this “confirm press” is not taken into consideration as a second confirm since it did not open any card. Now suppose that the user moves to any other location and press on confirm:

If the card was closed then we need to open the card by printing the corresponding character in the corresponding location on the board (like before). Then, the address of the LCD (cursor) should go back to the initial location.

Now, since this is the second confirm (second card is opened), then we need to check if the opened pairs right now are matching or non-matching pairs.

E.4.II CASE OF MATCHING

After doing the comparison between the two cards, if the cards were matching pairs then you need to keep the screen (keep the characters printed) and you need to increment the number of matches by 1. This number is to keep track if we match all the board pairs. So, if we reach 6 matches, then the game will end. Ending the game will be described later. Also, a buzzer sound should be generated after each successful match.

Back to our example, suppose that the user opens the last slot in line 1. Then the result on the screen will be as follows:

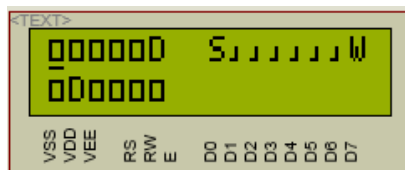


Figure E-5: opening a second card that matches the first one

E5. CASE OF NON-MATCHING

In case that the second card is non-matching pair, then we need to close again the two cards. The RED LED should be turned ON as a first step. A delay should be called to show the user that the second card is not matching and then you need to close the two cards by printing the square symbol in the corresponding positions on the board. Don't forget to set a flag that will indicate that the two pairs are now closed. The number of non-matching should be incremented by 1 and the non-matching bar meter should be updated.

Finally, the address of the LCD should go back to the initial position.

E6. NON-MATCHING BAR METER

The non-matching bar meter should be updated after every non-matching case. After incrementing the number of non-matching, the bar meter should be updated as follows:

If number of matches is less than 3, then print a full square character in the first location of the bar-meter (just after the character S. The filled square character is the last character shown in the ROM memory of the LCD and its 8-bit code is "11111111". Refer to LAB 7 manual).

Back to our example: if the player opens the first card after opening card D (on the second line second location), then the screen should show C on the first location for a certain time and then both characters C and D should be replaced by square symbol since the pairs are non-matching. Then the first slot of the bar-meter should be replaced by a filled square.

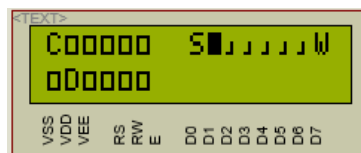


Figure E-6: opening the first non-matching pairs

After the delay, the screen will be updated as follow:

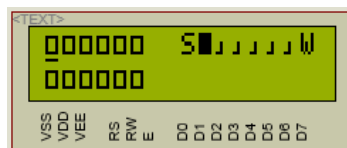


Figure E-7: closing the pairs after non-matching

Now suppose the user continue playing and doing non-matching pair. At the third non-matching pair, the second slot of the bar meter will be updated.

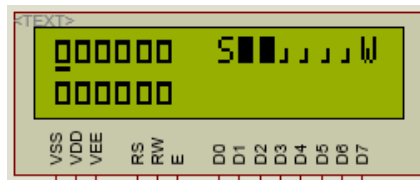
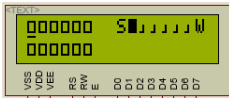
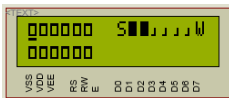
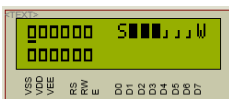
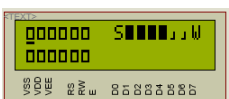
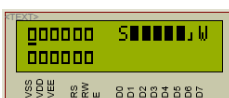
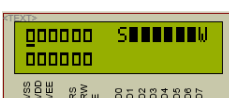


Figure E-8: after doing 3 non-matching pairs

This means that doing one or 2 non-matching pairs (opening 2 or 4 cards) the bar meter will show only first slot updated. At the third non-matching pair, the second slot will be updated. Follow the table below to see how to update the bar-meter with every non-matching pair.

Number of non-matching	Bar meter update
1-2	
3-4	
5-6	
7-8	
9-10	
Greater than 10	

E7. END GAME

In this mode the game ends only when the player is able to do all the 6 matching pairs. Then a feedback will be generated depending on the player performance. If the player do less than 5 non-matching pairs (between 0 and 4), then SUPER should be printed below the bar-meter on the second line. Also, any flashing pattern of your design for both Red and Green LED should be generated.

If the number of non-matching pairs is above 4 and below 9 (between 5 and 8), then AVG (average) should be printed. The green LED should be turned ON.

Finally, if the number of non-matching pairs is 9 or above, then WEAK should be printed and the Red LED should be turned ON. Below is an example of each case.

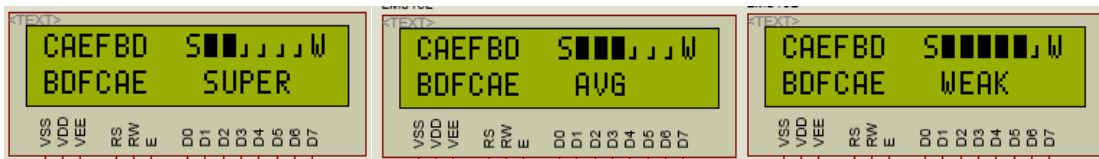


Figure E-9: three different ending game messages

After displaying the feedback and controlling the corresponding LED(s), the game should go back to the main menu. Don't forget to add a certain **delay** to make sure that the user is able to see the feedback message and LEDs. Whenever, you go back to main menu, you have to make sure to reset all variables used so that the user will be able to select mode 1 again and play the game again and again.

F. MODE 2

The game in mode 2 will depend on time. A timer will be generated with an initial value 90 seconds. Every 10 seconds, the timer value should be updated on the screen. The initial screen of mode 2 is shown in the figure below:

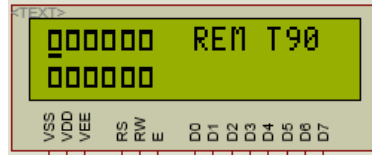


Figure F-1: Initial screen of Mode 2

The screen will show the board (like in mode 1) which is 6 squares on the first line and 6 squares on the second line. It will also display the remaining time by typing REM (stands for remaining) and T (stands for Time) and a two digit timer value (initially 90).

In order to implement the Timer, you need to use TMR0 interrupt. Every 10 seconds, the new value of the Timer should be updated (initially 90). So, the Timer display will go from 90 to 80, 70, 60, 50, 40, 30, 20, 10, and finally 00. **Refer to TMR0 section below.**

Hint: you can implement a function that will take a value from a specific register and print it on the screen. This function can be used later in your project (printing the score and printing of mode 3).

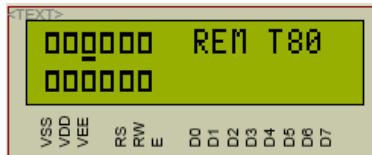


Figure F-2: Timer updated after 10 seconds

Note that the cursor is initially at first position of the board. Don't forget to return the cursor the old position (set by user) when you finish updating the timer value. In the figure above, the cursor was at the third position in line 1 before updating the timer from 90 to 80. So, the cursor should be loaded again to the same position on the LCD after the update and **NOT** to the initial position (position 1 in line 1).

The player in this mode will also be using all the push button left, right, up/down, and confirm. These buttons will have the same function as in mode 1. So, cursor will be moved left, right, and up/down using the first three buttons. Confirm will be opening the specified cards.

This mode does not have the non-matching bar meter. The score of this mode depend only on the remaining time and the number of matches done. The final score will be remaining time/10 + number of matches. This will happen only if the player was able to do all the 6 matching pairs before the end of the time. If the timer reaches zero, then the score will be 0 + number of matches.

So, this mode will end in two ways: timer becomes 00 or the player was able to do all matching pairs. When the game ends, you should print out the score on the second line as follow:

The word score and two digit score xx (xx stands for the score of the player at the end of the game).

Below is an example of a player who was able to do all matching pairs and the time left is 40 seconds. Note that will not take the actual time to calculate the score, we will always taking the timer printed on the screen. In our example below the actual remaining time can be any number between $30 < \text{REM T} \leq 40$. When the timer reaches 30, the update will be done on the screen.

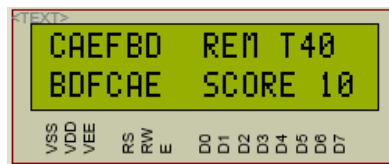


Figure F-3: calculating score example 1

So, in the above example the total score is $40/10 + 6 \text{ matches} = 4 + 6 = 10$.

Another example for calculating the score is when the time reaches 00 and the player failed to find all the matching pairs.

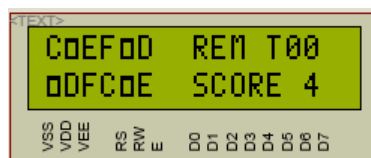


Figure F-4: calculating score example 2

In the second example: the remaining time is 00 and the player did 4 matching pairs. So, the final score is $00/10 + 4 = 4$.

After displaying the score for a certain time, the game should go back to the main menu and the user will able to choose any mode to play 1, 2, or 3. Don't forget to disable TMRO at this stage and reset all variables and flags used in mode 2.

F1. TMRO INTERRUPT

F.1.I TMRO

TMRO is an eight-bit, free-running timer/counter that is part of the PIC16F84A. In order to enable TMRO interrupts, both the global interrupt enable flag (GIE – bit 7 of the INTCON register) and the TMRO interrupt enable flag (TOIE – bit 5 of the INTCON register) must be set. This will cause an interrupt to occur every time the counter overflows from 0xFF to 0x00. Note that the counter will continue to be incremented after the interrupt occurs. Also note that it is the responsibility of the programmer to clear the TMRO interrupt flag (TOIF – bit 2 of the INTCON register) in the interrupt service routine, since failure to do so will result in an interrupt occurring as soon as global interrupts are re-enabled.

The frequency at which TMRO operates depends on the source of its clock. When TMRO is used as a timer, it runs on an internal clock that operates at a quarter of the frequency of the PIC16F84A's external clock. The timer can be slowed down further by using a pre-scalar, which effectively divides the internal clock frequency by a user-selectable power-of-two constant.

On the other hand, when TMRO is used as a counter, it operates from an external signal that is applied to pin 3 of the PIC16F84A (RA4/T0CKI). When operating in this mode, TMRO can be configured to count on either the rising or falling edge of the input signal. The pre-scalar can also be used to slow the counter by ensuring it counts every p th pulse, where p is a power-of-two constant.

TMRO can be configured by setting the appropriate bits of the OPTION register. Bits 0 to 2 (PS0, PS1, and PS2) are used to select the pre-scalar ratio. A value of 000 divides the clock by a ratio of 1:2, while a value of 111 divides the clock by a ratio of 1:256. Bit 3 (PSA) is used to assign the pre-scalar to either TMRO or the watchdog timer. The pre-scalar can only be assigned to one of them at a time. Bit 4 (TOSE) is used to specify whether the counter will be incremented on a rising or falling edge. Finally, Bit 5 (TOCS) is used to select the source of the TMRO clock.

F.1.II FUNCTION OF TMRO

In this interrupt you need to create 10 seconds delay used in mode 2. Every 10 seconds, the timer displayed on the screen should be updated as described earlier. When the timer reaches 00, the game will end and the score should be displayed as described in mode 2. Note that you should enable TMRO only when the user confirms Mode 2 in the main menu. In all other cases (outside Mode 2) TMRO should be disabled.

G. MODE 3

In mode 3, the game is similar to the mode 1 but with different techniques of calculating the score. The initial screen of mode 3 will be as follows:

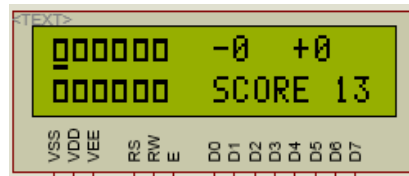


Figure G-1: initial screen of mode 3

- The screen of mode 3 contains: 6 squares on the first line and 6 squares on the second line to display the board as in mode 1 and 2.
- Number of matching pairs and number of revealed cards are also displayed. Number of revealed cards is the first number – 0. So, this number will be decremented from the initial score 13 (displayed on the second line). The number of matching pairs + 0 will be added to the initial score after every matching pair. Clearly, the matching pairs are when the user is able to open 2 matching cards like in mode 1 and 2. Whereas the number of revealed cards is the number of how many times the card is opened after it was shown for the first time. A detailed example is provided below:

Suppose that the player opened the first two cards:

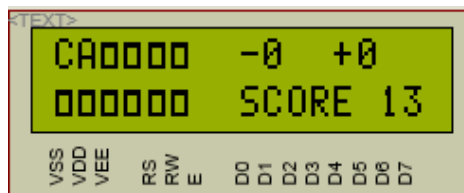


Figure G-2: example of revealed cards

At this step, you should mark Card 1 "C" and card 2 "A" are revealed. They were shown for the first time. But no action on the score will be taken since it is the first time we open these two cards.

Now if the player opens first card again and third card:

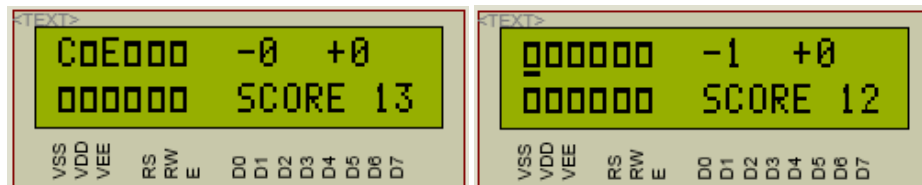


Figure G-3: opening a revealed card

Since C was opened before, the score should be update by -1. Off course the two cards will go face down again after certain time since they are non-matching pairs. Now we have 3 revealed cards. If the user re-opens two of them (suppose A and E):

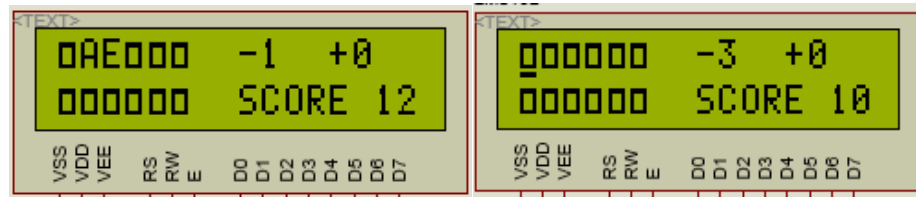


Figure G-4: opening two revealed cards

Now since both A and E were opened before, 2 points penalty will be deducted from the score. So, total opened revealed cards number is 3 and the score is $13 - 3 = 10$. Now suppose that the player makes a matching pair.



Figure G-5: making a matching pair

As we can see that the total number of revealed cards did not change even though A was opened before. This means this number should only be updated in case the pairs are non-matching. Clearly the matching pairs' number is incremented by 1. So, the score is $13 - 3 + 1 = 11$.

The game can end in this mode in two cases:

- Number of matching pairs is 6 (matches all board) and number of revealed less than 13
- Number of revealed cards is greater than or equal to 13.

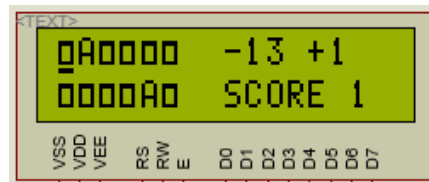


Figure G-6: after opening 13 revealed cards

In other words, the player is given the chance to open 13 revealed cards before ending the game. If the player was able to finish all the matching pairs before reaching the 13 revealed cards, then the game will end with a better score.

Note that the number of matching pairs is maximum 6, the number of revealed cards is max 13 (or 14 if two revealed cards were opened at the last step), and the maximum score is $13 + 6 \text{ matches} = 19$.

Also note that the score should not go below zero. You need to test that in your code. This can happen if the user opens 14 revealed cards without doing any match (which is opening 7 pairs). This is because we will be updating the number of revealed cards after checking matching pairs or non-matching pairs.

After ending the game, the score should be kept on the screen for a certain time.

- if the number of revealed cards exceeded, then a short buzz sound should be generated
- If the player was able to do all the matching pairs before exceeding the allowed opened revealed cards, then a flashing pattern of LEDS should be generated.

Then the game will go back to main menu (in both cases list above) and the player will be able to choose any mode to play.

Note that most of the functions implemented in mode 1 and mode 2 can be used easily to build mode 3.

V. SOFTWARE DESCRIPTION

You should write very structured code with multiple routines each of which can act as a separate module on its own that can be called many times when needed. By this modular programming, each block can be tested on its own by software simulation and hardware after downloading the program.

A. USEFUL HINTS

Always try to write in efficient way. There are different subroutines: 40 msec delay for de-bouncing, initializing the LCD, writing on the LCD, changing the address of the LCD, writing a number to LCD, Buzzer ... Note: Some sub-routines can also call sub-routines.

Useful Instructions: SWAP; Logic such as ANDWF, XORWF... Arithmetic functions such as SUBWF, SUBLW, ADDLW, ADDWF ...

Note that you can use STATUS register to do comparisons (check if zero, check if negative). Refer to the manual of PIC16F84A more information.

VI. SOFTWARE SIMULATION

Write your program step by step; do not include all modules and complications at the same time. Write every module separately, simulate it on software by watching various registers (watch window), single stepping your code, and measuring time using the stopwatch and setting the appropriate oscillator frequency. If everything is correct, then you can either continue writing software or download the program and test it on hardware. You can test on software (Proteus) and hardware every mode, and then include all modes together. Structured code with multiple subroutines is required to simplify debugging and testing.

To build the circuit using Proteus, use the following components (Refer to figure VI-1):

Component	Proteus Component		
	Name	Category	Notes
PIC16F84A	PIC16F84A	Microprocessors ICs	Double click on the device: Change frequency to 4MHZ, and load the hex file.
LCD	LM016L	Optoelectronics	
Variable Resistor	POT-HG	Resistors/Variables	
Fixed Resistor	RES	Resistors/Generic	
NPN Transistor	2N2222A	Transistors	
Buzzer	Buzzer (DC operated buzzer)	Speakers and Sounders	Double click on the device: Change operating voltage to 5V, you can also change the frequency.
LED	RED/ Green	Optoelectronics	
Push Button	Button	Switches and Relays	

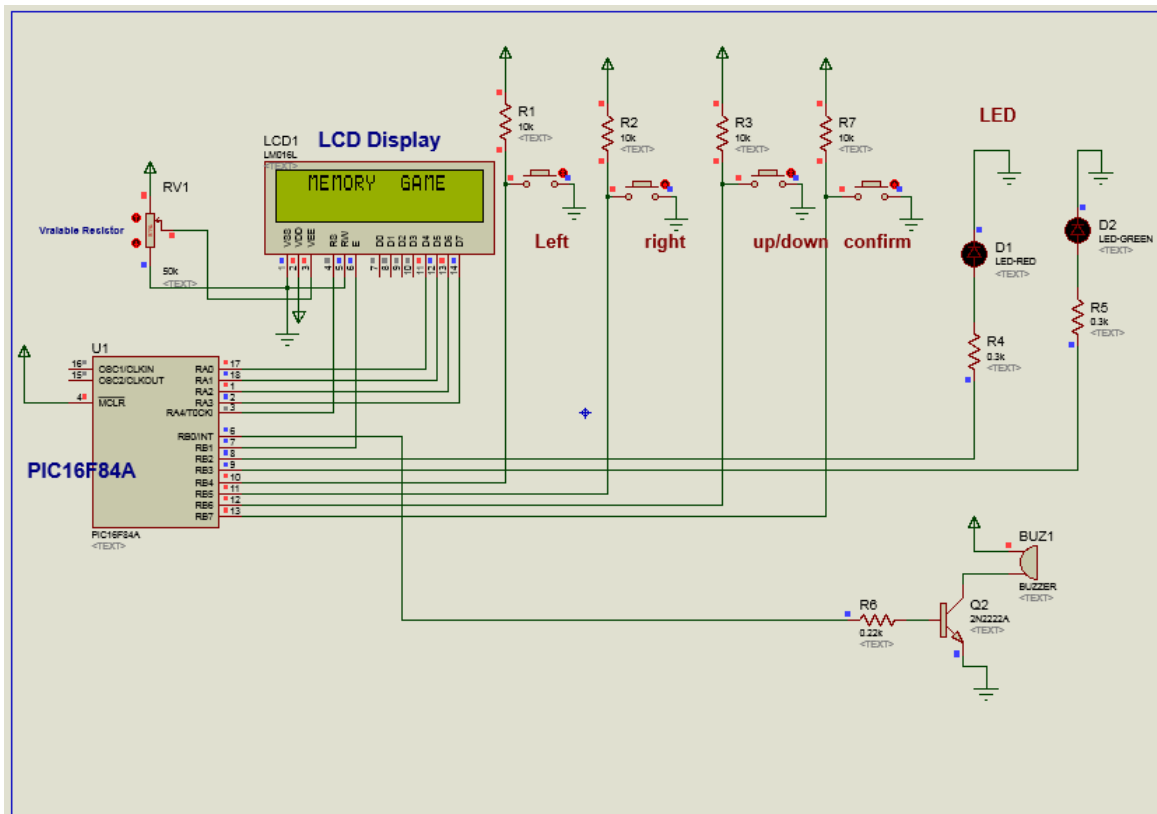


Figure A-1: Proteus Circuit Diagram

VII. PROGRAMMING THE PIC16F84A

Once you have finished developing and testing your program in the MPLAB IDE, you are ready to program the PIC16F84A. Since instruction memory in the PIC16F84A is built around flash memory technology, new programs can be loaded onto the microcontroller in the same way data can be saved on a flash memory stick.

Prior to programming the PIC16F84A, you must first ensure it will be properly configured. To do so, add the following line to the header section of your program following the list and include directives: `__config_XT_OSC & _WDT_OFF & _PWRTE_ON & _CP_OFF`. These configuration flags are used to specify the type of the external clock (crystal oscillator), disable the watchdog timer, enable the power-up timer, and disable program memory protection, respectively.

Once your program builds successfully, you are ready to program the PIC16F84A with the generated hex file. The hex file, which has a suffix of *.hex, is automatically generated each time you build your program.

To program the PIC16F84A from the MPLAB IDE,

- Choose the programmer device by selecting “Programmer → PICSTART PLUS”.
- Connect PICSTART Device to PC (through COM cable), plug the power cable, and place the PIC chip
- Choose “Programmer → Settings→ Communications tab”. Then choose the corresponding communication port.

- ## VIII. HARDWARE IMPLEMENTATION

The following circuit will be implemented on a PCB and used for testing. Many boards will be fabricated and you are welcome to do the testing any time as scheduled (will be posted).

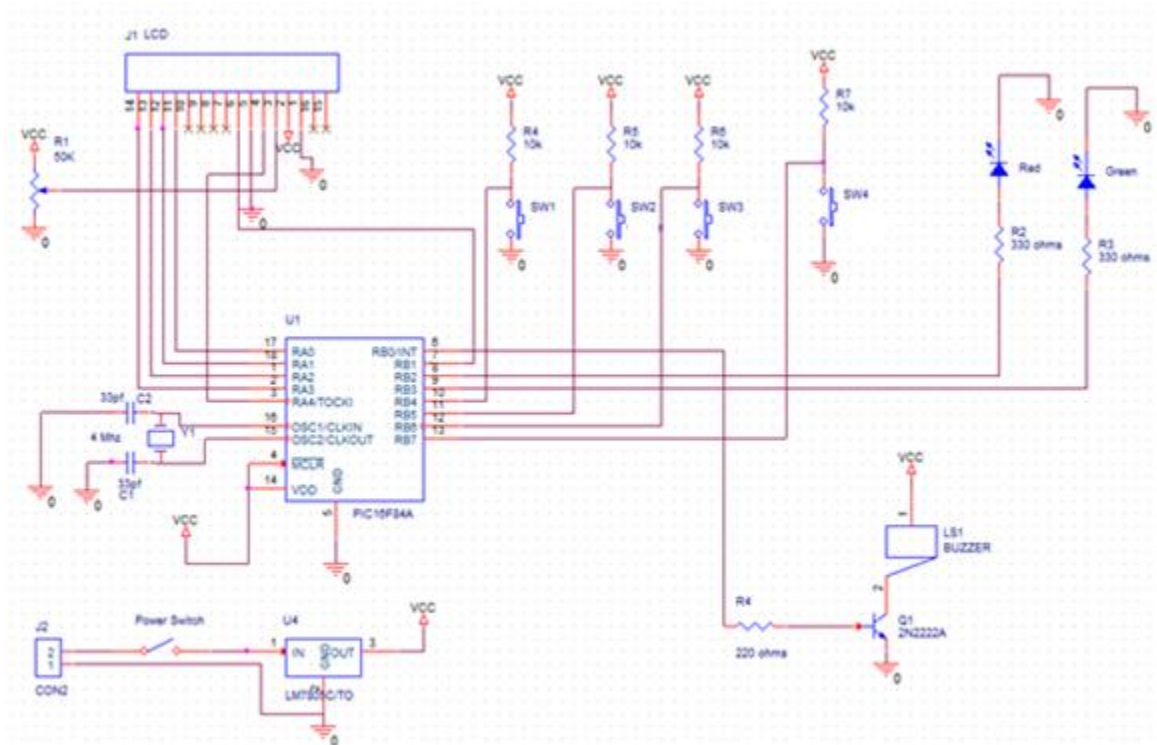


Figure VIII-1 shows the circuit diagram of “MG” to be designed. In this project, we will use push button switches which are normally open (NO) and active low. Hence the button is active when grounded. A variable resistor is used to vary the contrast of the LCD.

B. COMPONENTS

To complete this project, you will need the following components:

Component	Value	Items
PIC16F84A Microcontroller	PIC16F84A	1
4 MHz Crystal Oscillator (2 legs)	4 MHz	1
Capacitors (Ceramic)	33 nf	2
Miniature Push-Button (NO) Switches		4
5/6-Volt Buzzer		1
2N2222 NPN Transistor		1
Resistor	220 Ω	1
Resistor	10k Ω	4
Resistor	330 Ω	1
LED	Green,RED	2
Variable resistor	50k Ω	1
LCD WH1602A (Preferable with ordered pins from 1 to 16)***		1
LCD connector pins		1
18-pin chip socket for PCB		1
9V Battery	9V	1
9Volt Battery Connector		1
7805 Voltage Regulator (9v to 5 v)		1
Miniature ON/OFF Switch		1
PCB (FR4 or glass epoxy "single sided 10x15")		1

You have to get the PIC16F84A only. Try to get more than one in case you damage the chip while programming.

At the end of the project, if any one hopes to have PCB design, you are welcome to come and fabricate it with our help. Then, you have to bring all the components listed above.

These components may be purchased from a number of electronics supply stores in and around Beirut. Check the list posted on Moodle.

Data sheets for the PIC16F84A and LCD WH1602A are available from the Download Area on the Moodle.

C. COMPONENTS DESCRIPTION

C1. POWER SUPPLY

Figure C1 shows how to supply power to the PIC16F84A. The 7805 is used to provide a clean +5 volt supply, which can also be used to power the other components of the system.

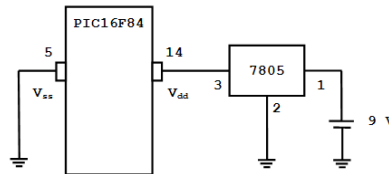


Figure C-1: Supplying Power to the PIC16F84A

C2. CLOCK

Figure C2 shows how to connect the crystal oscillator to the PIC16F84A. The oscillator is used to provide the PIC16F84A with a 4 MHz square wave clock signal.

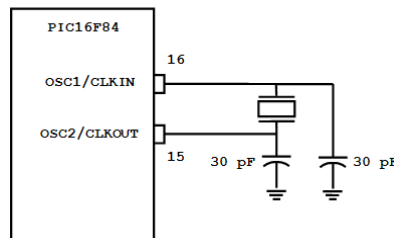


Figure C-2: Connecting the Crystal Oscillator to the PIC16F84A

C3. BUZZER

Figure C-3 shows how to connect the buzzer to the PIC16F84A. The 2N2222 transistor operates as a switch.

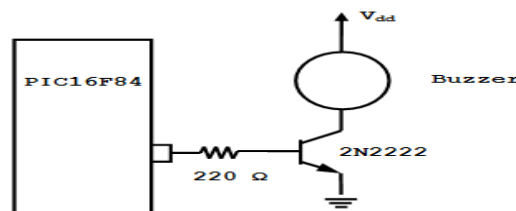


Figure C-3: Connecting the buzzer to PIC16F84A

C4. SWITCHES

Figure C-4 shows two alternative ways of connecting a switch to the PIC16F84A. The choice depends on whether you want your input to be active high or low, respectively. Due to the mechanical nature of a switch, a phenomenon known as bouncing typically occurs when the switch is closed. This is due to the contacts bouncing against each other for a very short time before the switch closes completely. Since switch bouncing may result in an incorrect value being read, a switch should be de-bounced before its value is read. This can be done in software by ensuring the value of the switch remains constant for at least 40 msec before its value is read.

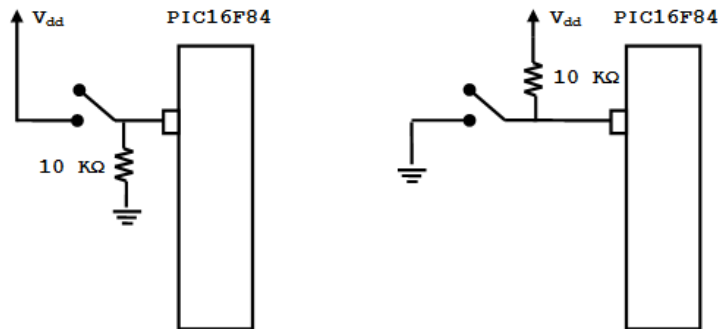


Figure C-4: Connecting switches to PIC16F84A

IX. GRADING CRITERIA AND DELIVERABLES

You must present your work in a short power point presentation that clearly explains the way your code works. It should focus on how your code performs the **specified** and the **important** functions. Pay particular attention to explaining timing issues, how you manage interrupts from the inputs, and how you manage the buzzer, the display registers and the LCD display. Finally, briefly explain how you tested your code and your hardware. To demonstrate your project and do the presentation, a Moodle activity “demo location and time slots” will be posted on Moodle. You will be asked in the presentation to explain some parts of the codes and do software simulations.

The grade for this project will be based on several criteria: how well you present and understand your work and how well your code is organized. Please make sure there are comments (**have a part of grade**) in your code and that your code is structured and contains independent subroutines that represent functions rather than large blocks of code.

Every single function and specification will be tested. The total grade of the project will be divided as follows:

- Coding and functionality
- Coding organization and commenting
- Presentations

X. BONUS

You might add any creative ideas to the project in addition to the specified specifications above (but only after you meet all the requirements). This will give you bonus points but I will judge if the new ideas are counted as bonus.

Some bonus functions can be like:

- Give the user the option to enter the board card and then play it
- Give the user the option to enter maximum number of revealed cards allowed in mode 3 and then play the game
- Add a penalty to Mode 3: if you failed to match and you should have been able to match then 2 will be deducted from the score.

XI. IMPORTANT NOTES

Read carefully the entire document before starting your project.

- Dates of the demo and presentations: Tuesday April 30, Thursday May 2, and Friday May 3.
- You should submit your code (zip the whole project), and the presentation by Monday April 29 at 10:00 p.m.

Start now; do not postpone your work on the project to the last days; Schedule your time.

Presenting your own incomplete work is better than adopting someone else's work.

Serious measures will be taken for cheating in the project (ZERO grade and/or Dean's warning). All codes will be corrected and checked. You need to have comments in your codes (grades will be deducted from un-commented codes). In addition, in the presentation different group members will be asked individually for detailed information in their code. Members of the same group will not get the same grade on the project.

GOOD LUCK AND HOPE YOU HAVE ENJOYED THE COURSE