Middle East Technical University | Department of Computer Engineering

CENG 336

Introduction to Embedded Systems Development

Spring 2015-2016

Take Home Exam 2

Due date: 17 April 2016, Thursday, 23:55

1 Objective

This assignment concerns programming with interrupts and timers. This will be done with the implementation of a single player memory game. Clarifications and revisions on the specifications and requirements will be posted on the *metu.ceng.course.336* newsgroup.

Keywords: *Timer interrupt, Button interrupts*

2 Scenario

The main goal of this assignment is to implement a game in which the player is expected the memorize a sequence of cells whose location generated randomly by using the *Timer0* interrupt.

2.1 Game Play

The game you will implement for this assignment consists of five states. These states are called **ready**, **encoding**, **play**, **wait** and **end**. These states, and transitions between them explained in proceeding sections.

2.1.1 The Ready State

Game should be in this state when board is powered on. LEDs should be turned off, and 7-Segment Display should be turned off, when game is in this state. Game should only accept *RA4* key press and ignore other key presses. When player presses *RA4* button you should start the game with 1st level, and game state should move to *the encoding state*.

2.1.2 The Encoding State

Game should show randomly selected cells to user in this state. You should look at to *Cell Selection* section for randomly selected cells. If player plays nth level, then n many cells should be shown in this level. Game should ignore any key press in this state. Counter should be initialized into 3 seconds, and game should move to *the play state*, when counter reaches to 0.

2.1.3 The Play State

Player should select cells shown in *the encoding state* during this state. Player will enter the location of cells using *RB4-7* buttons and they will use *RA4* button for selection. When player presses to *RA4* button, selected cell should be on until game moves to either *wait state* or *ready state*. Counter should be initialized into 90 seconds. There are multiple possible transition from this state which are given at the below.

- If player selects a cell which is not shown in *the encoding state*, then player loses and game advances into *the end state*.
- If player selects a cell which is already selected, then player loses and game advances into *the end state*.
- If counter reaches to 0 before user can select every cell shown in *the encoding state*, then player loses and game advances into *the end state*.
- Otherwise, players wins current level and game advances into next level and game state becomes *the wait state* if current level is not 3, if it is 3, then player wins the game and game state becomes *the end state*.

2.1.4 The Wait State

Game should wait until user presses to *RA4* button and ignore any other key press. When *RA4* button is pressed game enters into *the encoding state*.

2.1.5 The End State

Game enters to this state when player either loses or wins the game. During this state 7-Segment Display should show "End X" where X is the last level which player wins. If player loss the game at first level, then X should be 0. Game stays in this state until user presses to *RA4* button and game should ignore any other key press. When *RA4* button is pressed game enters into *the ready state*.

2.2 Counter

Counter will be used to measure specified number of seconds. You should use *Timer1* for the counter with 1 second intervals. Current value of counter should be shown on rightmost 2 cells of 7-Segment Display (D1, D0). Please notice that counter can be used to count at most 99 seconds.

2.3 Cell Selection

Cells will be represented with LEDs. Game screen is a 6x4 table with upper left most cell being RC0 and lower right most cell being RF5. A cell will be specified using a row index and a cell index. You may consult into *LEDs* section for exact location of cells. You will select a cell whenever user presses to *RA4* when game is not in *the end state*. You will generate row index and column index as described in *random number generation* section. Please notice that when player completes nth level successfully, (s)he should press *RA4* button n times, and 1 more time in order to start n+1 th level. You should use these n+1 press button for selecting n+1 cells for n+1 th level.

Player will select cells with row index and column index in *the play state*. Row index will be displayed in the left most cell of 7-Segment Display (D3), and column index will be displayed in the second left most cell of 7-Segment Display (D2). Row and column index will be controlled with RB key presses as described below.

- If RB7 button is pressed, then row index should be incremented. Row index should be wrapped around in case of overflow, i.e if RB7 pressed when row index is 5, then row index should be 0.
- If RB6 button is pressed, then row index should be decremented. Row index should be wrapped around in case of overflow, i.e if RB7 pressed when row index is 0, then row index should be 5.
- If RB5 button is pressed, then column index should be incremented. Column index should be wrapped around in case of overflow, i.e if RB7 pressed when column index is 3, then column index should be 0.
- If RB4 button is pressed, then column index should be decremented. Column index should be wrapped around in case of overflow, i.e if RB7 pressed when column index is 0, then column index should be 3.

2.4 Random Number Generation

Two random number should be generated for row and column index, whenever user presses to *RA4* button and game is not on *the end state*. You should store randomly generated numbers until user starts to play next level. When user starts to play next level, you should use stored numbers for cells of that level, and you should store new set of numbers using *RA4* presses on until next level. Lets denote the value of Timer0 with xy where x and y are four bit numbers. You should generate row and column index using following strategies. You should use the first generated pair which is different from any stored pair.

- Row index should be $x\%6$ and Column index should be $y\%4$.
- Row index should be $\overline{x}\%6$ and Column index should be $y\%4$.
- Row index should be $x\%6$ and Column index should be $\overline{y}\%4$.
- Row index should be $\overline{x}\%6$ and Column index should be $\overline{y}\%4$.

where $\overline{z}$ is bitwise complement of z, and $z\%T$ is the remainder of division of z by T.

3 Specifications

- You will use 8 ports in this assignment: *PORTA*, *PORTB*, *PORTC*, *PORTD*, *PORTE*, *PORTF*, *PORTH* and *PORTJ*.
- *PORTA* will be used as an input port to receive *RA4* button interactions.
- *PORTB* will be used as input port to configure the numbers on 7-Segment Display.
- *PORTC*, *PORTD*, *PORTE* and *PORTF* will be used as output ports in this assignment. These ports will be used to display game. Remember that you will use only 6 LEDs on each port.
- *PORTH* and *PORTJ* will be used as output ports to show numbers on 7-Segment displays. You can find some useful information on **Hints** section.
- You should use high priority interrupts for timer interrupts, and low priority interrupts for *PORTB*.
- When the PIC is powered on, *Timer0* should be started, *RA4* should be able to take input from the player, and all LEDs should be turned off.
- Also, you should configure the *Timer0* to work in 8-bit mode. You can find the necessary configuration settings in **PIC18F8722** data sheet.
- When any of the buttons is pressed/released by the player, other components are still expected to work properly. Any flicker or turn off state **will not be accepted**.

4 Implementation and Regulations

- You will code your program using PIC assembly language.
- Your program should be written for **PIC18F8722** working at **40 MHz**.
- Usage of polling is prohibited for *RB** buttons in this assignment. You have to use *interrupt-on-change* on feature of *PORTB*. You have to use polling to control *RA4* button. control buttons. You have to use button interrupts to control buttons. **All button actions should be triggered on button release**. Make sure to use proper debouncing logic in your code to prevent a single button press triggering multiple events.
- You should use the *Timer0* and *Timer1* interrupts as written in **Scenario** section. You can also refer to the **Grading** section to check the significance of this requirement.
- When you are writing your code, you can use the lecture notes on Input/Output and Interrupts, Recitation04 and Recitation06 documents. It is also highly recommended that you make extensive use of the PIC data sheet, MCDEV Kit User Manual and Programming Manual which are available on COW. Please consult these resources first before you ask any questions to the assistants or on the newsgroup.

5 Hand-In Instructions

- You should submit your code as single file named as the *the2_##.asm* through COW where ## represents your group number. Do not forget the replace ## with your group number.
- At the top of *the2_##.asm*, you should write ID, name and surname of both group members as commented out.

6 Hints

6.1 LEDs

To displays cells on LEDs you should use *PORTC*, *PORTD*, *PORTE* and *PORTF*. You should use following table to determine correct led for given (row, column) tuple.

		Column value			
		0	1	2	3
Row value	0	RC0	RD0	RE0	RF0
	1	RC1	RD1	RE1	RF1
	2	RC2	RD2	RE2	RF2
	3	RC3	RD3	RE3	RF3
	4	RC4	RD4	RE4	RF4
	5	RC5	RD5	RE5	RF5

6.2 7-Segment Display

There are four common cathode 7-Segment displays mounted on the development board. *PORTJ* and *PORTH* are used as data bus and selection pins, respectively, as illustrated in figure below. Notice that *PORTH* pins are connected to all displays.

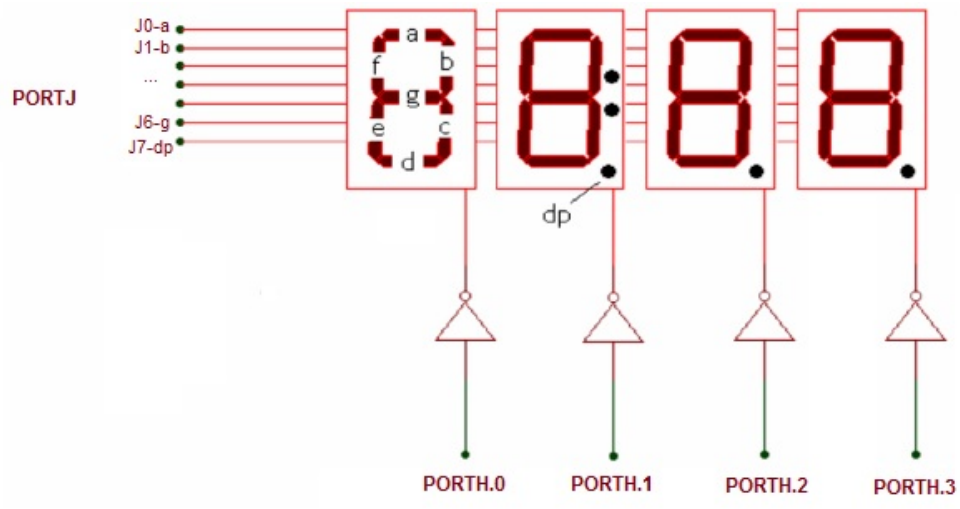

Figure 1: Connections for the 7-Segment displays on the development board.

As an example, if you want to show number "4" on leftmost display (*DISP0*), you should first select it by setting *RH0* and clearing *RH1*, *RH2* and *RH3*, then send binary "01100110" to *PORTJ*. Hence, *a*, *d*, *e* segments on *DISP0* will be turned off, and *n*, *c*, *f*, *g* segments will be turned on. Note that *RJ7* pin is used for *dp* on *DISP1* which run simultaneously.

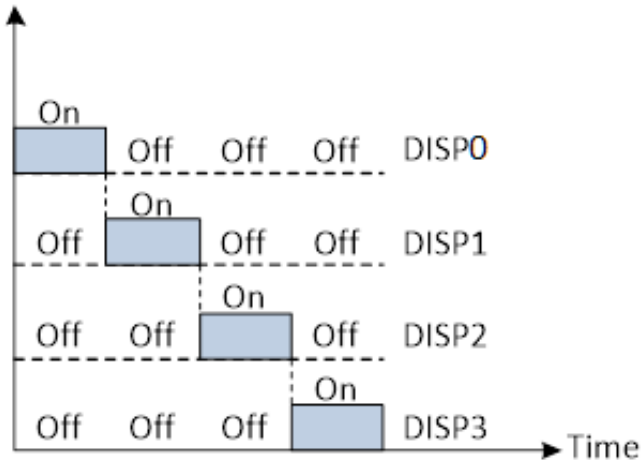

Figure 2: Graphical illustration to show characters on each 7-Segment displays simultaneously.

If you want to show, for instance some value (1234) on the displays, you should select only *DISP0* by using *PORTH*, write the byte necessary to turn on segments to *PORTJ*, and wait for a while. Then you should do the same for *DISP1*, *DISP2* and *DISP3*. This is illustrated in figure below. If you adjust "on" times properly and repeat on-off cycles continuously, you show your values on the displays in a smooth manner without flicker.

6.3 Timer 0 & Timer 1

It is beneficial to avoid function call in Timer ISRs. You may use some counters or flags and do the function calls or calculations in your main routine. For example, if you call a display subroutine in ISR, measuring the correct time interval in ISR will become harder.

7 Cheating

We have zero tolerance policy for cheating. People involved in cheating will be punished according to the university regulations. Cheating Policy: Students/Groups may discuss the concepts among themselves or with the in- structor or the assistants. However, when it comes to doing the actual work, it must be done by the student/group alone. As soon as you start to write your solution or type it, you should work alone. In other words, if you are copying text directly from someone else - whether copying les or typing from someone else's notes or typing while they dictate - then you are cheating (committing plagiarism, to be more exact). This is true regardless of whether the source is a classmate, a former student, a website, a program listing found in the trash, or whatever. Furthermore, plagiarism even on a small part of the program is cheating. Also, starting out with code that you did not write, and modifying it to look like your own is cheating. Aiding someone else's cheating also constitutes cheating. Leaving your program in plain sight or leaving a computer without logging out, thereby leaving your programs open to copying, may constitute cheating depending upon the circumstances. Consequently, you should always take care to prevent others from copying your programs, as it certainly leaves you open to accusations of cheating. We have automated tools to determine cheating. Both parties involved in cheating will be subject to disciplinary action. [Adapted from`http://www.seas.upenn.edu/cis330/main.html`]