Hardware Interfacing

😂😂😂

🅱️enjamin Kong — 1573684

Lora Ma ———— 1570935

ECE 212 Lab Section H11

April 8, 2020

Contents

1 Discussion

Due to the COVID-19 pandemic, we were unable to physically complete the hardware component of the lab. As a result, the majority of this lab report will be focused on how the software component (i.e., the assembly language programming) works with the hardware component of the lab. We will discuss various details regarding the hardware and software working together. We will also outline the different parts of the lab and what purposes they each fulfill.

The purpose of this lab was to gain experience using the Netburner board to interface with hardware. We were to use the Netburner board connected with a bufferboard to interface with the GMC2288C 8x8 LED array. Furthermore, we were to use two 3x8 74LS138 decoders, 8 1 kΩ resistors, and two hex-input 74LS04 inverters.

The Netburner board uses a 10-pin ribbon cable to interface with the bufferboard. This 10-pin ribbon cable maps pins 37 to 43 of the Netburner to the bufferboard. The bufferboard is then connected to various parts of the breadboard, which contains the LED array.

We now discuss the software component of the lab, which we were able to complete (despite not being able to test it with the hardware). There were three parts to the lab:

1. WelcomePrompt,
2. Convert, and
3. LedSub.

In the first part of the lab (WelcomePrompt), we wrote a subroutine that prompts the user to enter a single keystroke from the keyboard. We first display a welcome message, then prompt the user for a string. When the user inputs something, we check that the character is valid. An invalid input results in re-prompting the user for input. We then place the character (stored in ASCII format) onto the stack, at which point we pass it on to the Convert subroutine (which is the next part of the lab).

For the second part of the lab (Convert), we take the keystroke from the stack and pass out the pattern address through the stack. This pattern is stored as 2 longwords (2 bytes), and we use the subrountine called convert1 (provided to us) to get the pattern. We then move on to the LedSub subroutine (the last part of the lab).

For the third part of the lab (LedSub), we take the pattern address passed from the stack to display the pattern onto the 8x8 LED array. We then loop through the array and check if each LED should be switched on or off. After we iterate through the subroutine, we do a 300 unit delay such that information the LED array displays is visible.

We have included UML diagrams for each respective part of the lab in the appendix, along with our code.

2 Question

Why were the two 74LS138 decoders used in the circuit instead of directly connecting the pins to the array? Discuss how removing the decoders would affect the function of the LED array in the context of this lab.

If we didn't use the two decoders, we would require a wider ribbon cable (a 16 pin one) from the Netburner board to the bufferboard. This is because each row/column in the LED array has 8 positions; hence, we need 3 bits to access any position in a row/column. Hence without the decoders, we'd need 6 extra input pins. We would have to select LEDs in the LED array directly from the microprocessor; hence, we would have to modify the subroutines in LedSub to select pins directly.

3 Appendix

3.1 Part A UML Diagram

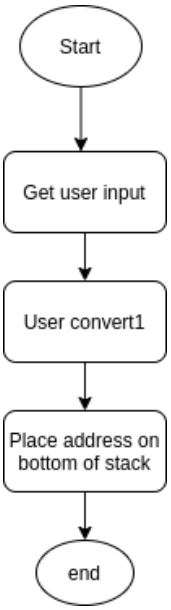

Figure 1: UML diagram for part A.

3.2 Part B UML Diagram

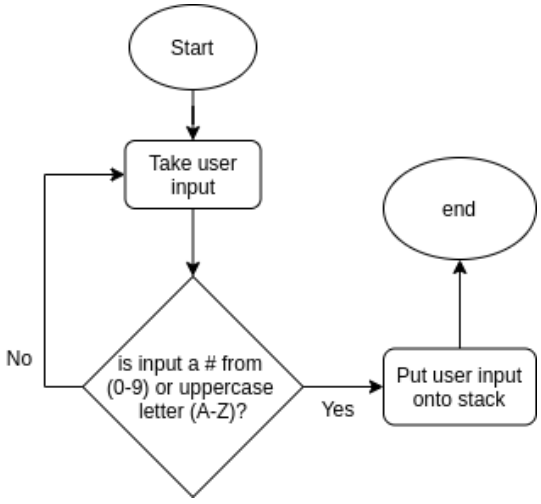

Figure 2: UML diagram for part B.

3.3 Part C UML Diagram

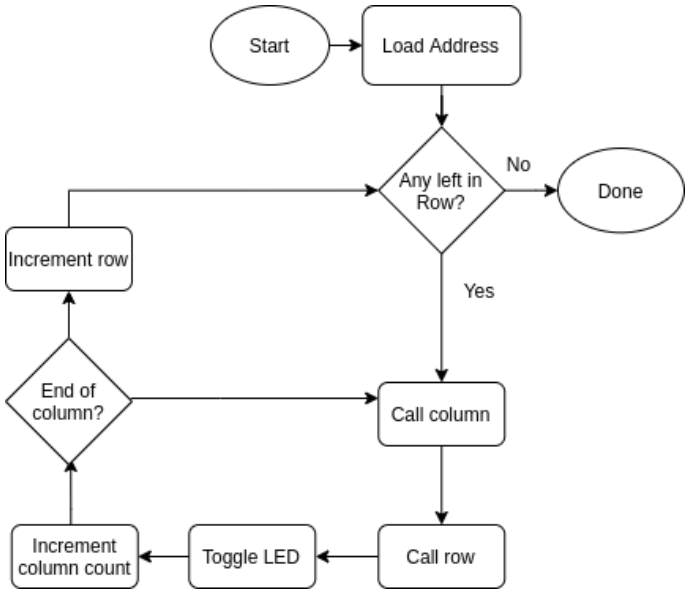

Figure 3: UML diagram for part C.

3.4 Part A Assembler Code

```
1  WelcomePrompt:
2  /*Write your program here*****************************************/
3
4  /* Backup registers */
5  LEA -40(%sp), %sp
6  MOVEM.L %D2-%D5/%A2-%A5, (%sp)
7
8  /* Display Welcome message */
9  pea Welcome
10 jsr iprintf
11 adda.l #4, %sp
12 jsr cr
13
14 /* Prompt for string */
15 PEA Ask
16 JSR iprintf
17 ADDA.L #4, %sp
18 JSR cr
19 BRA GetChar
20
21 Invalid:
```

```
22    PEA InvalidInput
23    JSR iprintf
24    ADDA.L #4, %sp
25    JSR cr
26
27  GetChar:
28    JSR getchr
29    CMPI.L #48, %D0
30    BLT Invalid
31    CMPI.L #57, %D0
32    BGT Check
33
34  Check:
35    CMPI.L  #65, %D0
36    BLT Invalid
37    CMPI.L  #90, %D0
38    BGT Invalid
39
40  /* Move ASCII Keystroke to location */
41  MOVE.L %d0, 44(%sp)
42
43
44  /* Restore registers */
45  MOVEM.L (%sp), %D2-%D5/%A2-%A5
46  LEA     40(%sp), %sp
47
48  rts
49  /*End of Subroutine ************************************************/
50  .data
51  /*All Strings placed here ******************************************/
52
53  Welcome:
54  .string "Welcome to Wing's LED Display"
55
56  Ask:
57  .string "Please enter an UpperCase letter or Number from the keyboard"
58
59  InvalidInput
60  .string "Invalid entry, please enter proper keystroke from keyboard"
61
62  /*End of Strings ***************************************************/
63  /*****************************************************/
```

3.5 Part B Assembler Code

```
convert:
/*Write your program here*****************************************/

/* Backup registers */
LEA -40(%sp), %sp
MOVEM.L %D2-%D5/%A2-%A5, (%sp)

MOVE.L 44(%sp), -(%sp)
JSR convert1
MOVE.L (%sp)+, 44(%sp)

/* Restore registers */
MOVEM.L (%sp), %D2-%D5/%A2-%A5
LEA 40(%sp), %sp

rts

/*End of Subroutine ************************************************/
.data
/*All Strings placed here ******************************************/


/*End of Strings ***************************************************/
```

3.6 Part C Assembler Code

```
LedSub:
/*Write your program here*****************************************/
LEA -40(%A7),%A7
MOVEM.L %D2-%D7/%A2-%A5,(%A7)

MOVEA.L 44(%A7), %A2 /* Load address*/
CLR.L %D3
CLR.L %D2

LoopRow:
CMP.L #8, %D2 /* Check if loop is done */
BGE End
MOVE.B (%A2, %D2*1), %D4 /* Load row */

LoopColumn:
MOVE.L %D3, -(%sp)
JSR Column
LEA 4(%sp), (%sp) /* Clean up stack */

```

```
20  MOVE.L %D2, -(%sp) /* Load row number in to stack */
21  JSR Row /* Call row subroutine */
22  LEA 4(%sp), (%sp) /* Clean up stack */
23  MOVE.L %D3, %D5
24  ADD.L #-7, %D5
25  BTST.B %D5, %D4
26  BEQ TurnOff
27  JSR TurnOnLed /* Turn on if 1 */
28  BRA Loop
29
30  TurnOff:
31  JSR TurnOffLed /* Turn off if 0 */
32  BRA PostLoop
33
34  Loop:
35  ADD.L #1, %D3 /* Increment column */
36  CMP.L #8, %D3
37  BLT LoopColumn
38  CLR.L %D3
39  ADD.L #1, %D2 /* Increment row */
40  BRA LoopRow /* return to row loop */
41
42  End:
43  MOVE.L #300, -(%sp)
44  JSR Delay /* Call delay subroutine */
45  LEA 4(%sp), (%sp) /* Clean up stack */
46
47  MOVEM.L (%A7), %D2-%D7/A2-%A5
48  LEA 40(%A7),%A7
49
50
51  rts
52  /*End of Subroutine ************************************************/
53  .data
54  /*All Strings placed here ******************************************/
55
56
57
58  /*End of Strings ***************************************************/
59  /****************************************************/
```