ECE 212 Lab - Introduction to Microprocessors
Department of Electrical and Computer Engineering
University of Alberta

Lab 1: Introduction to Assembly Language.

Student Name	Student
Arun Woosaree	xxxxxxx
Navras Kamal	1505463

Contents

1 Introduction **2**

2 Design **2**
2.1 Part A 2
2.2 Part B 3

3 Testing **3**
3.1 Part A 3
3.2 Part B 4

4 Questions **4**
4.1 Question 1 4
4.2 Question 2 4

5 Conclusion **5**

6 Appendix **6**
6.1 Part A Assembler Code 6
6.2 Part B Assembler Code 9

7 Marking Sheet **12**

1 Introduction

This text is filler text from an ece 210 lab report The purpose of this experiment was to explore the process behind designing and implementing real life design problems using the Xilinx software. A multiplexer (MUX) and demultiplexer (DEMUX) circuit were designed with the goal of sending data to three different 'radio astronomers'. Xilinx was used to create schematics for the circuit, which was then programmed to the Xilinx FPGA development board. The second part of the experiment involved the design and implementation of a lab access control circuit, again using Xilinx and the development board.

A multiplexer is a device that selects a single input from multiple signals, and only transmits one signal. It effectively receives multiple inputs, and only has a single output. The selection occurs based on the values of the 'selection signals'. For a 2x1 (two input signals) multiplexer, only one select signal is needed, but for a 4x1 multiplexer, two select signals are needed.

Multiplexers are often coupled with demultiplexers. DEMUXs receive a single input, and then send this input on to different possible outputs or 'locations', based again on the select signals inputted. The combination of a multiplexer and a demultiplexer allow the transfer of multiple signals over a shared medium or transmission line, in a process known as multiplexing. The signals are combined at the transmitter by the MUX and then split up at the receiving end by the DEMUX.

In the lab, the Xilinx software was utilized to design a three input/one output MUX, and a one input/three output DEMUX using only AND gates, OR gates, and inverters. Initially, Boolean expressions representing the MUX and DEMUX were simplified using a K-Map. From these simplified expressions, two separate designs were created using schematic capture tools, and then were utilized to implement data transmission to engineers. The output of the MUX was wired to the input of the DEMUX. The three outputs from the DEMUX are then sent to the office of radio astronomers. Three additional outputs are used, (designated as 'engineering indicators') to confirm the radio astronomers are receiving data. Only one of these engineering indicators should be turned on, considering the DEMUX selects only one output.

In the second part of the experiment, a circuit was designed and then implemented to control access to two labs: Lab0 and Lab1. Two input signals are accepted (a 2-bit card code and a 3-bit key), with three output signals (Lab0_unlock, Lab1_unlock, and Alarm). A valid card read and correct key entered unlocks the appropriate lab, and invalid input signals will sound the alarm.

2 Design

2.1 Part A

For the first part of the lab, the address register a1 was chosen to initially point to memory location 0x2300000, which is the starting point of where the input data was stored. We used this address register to keep track of the memory location of the next long word of data to be read as an input to our program. Even though one memory location is capable of storing one ASCII character, in this lab, 4 memory locations were used to store one ASCII character, as specified in the lab manual. Next, a2 was selected to initially point at the memory location 0x231000, which is the starting location of where our output for the converted values was. This register was used to keep track of the memory location of where the next long word of our converted data would go. The data register d2 was chosen to temporarily store data so we could do comparisons and process the input data.

The *Setzeros.s* and the *DataSrorage.s* files, which were provided, were used to initialize memory contents.

We started with a loop branch that served as our main looping function. This loop first starts by moving data from a memory location pointed to by a1, to the data register d2 so that we could start comparing the input data to known ASCII values. In the first comparison, the input character is compared to '0x0D', which is the ASCII code for the 'Enter' key. This code is meant to signal the end of the program, so if the input was the ASCII code for 'Enter', we branched to a label that would end our program. The next step was to determine if the input character was valid. For Part A, an input character was valid if it was an ASCII character from 0-9, A-F, or a-f. In other words, the data was accepted if it was in the following ranges: 0x30-0x39, 0x41-0x46, or 0x61-0x66. If the input character was invalid, we brached to a label named 'err', which would put the error code 'FFFFFFFF' in the memory location pointed to by a2, which keeps track of where our output data goes.

If the input character was valid, we had three branches to take care of each of the accepted ranges of input. For example, in the branch that took care of the input range A-F, the ASCII value of 'A' was subtracted from the input value, and the difference was added to the hex value 0xA. The converted value is then moved to the memory location pointed to by a2, which keeps track of where our output data goes. Similar steps were done to convert input characters in the other accepted ranges. After converting the input character and moving the output to the location pointed by a2, we branched to a label 'endloop', which increments the addresses stored in a1 and a2 by 4, and then branches to the loop, where the process is repeated.

2.2 Part B

The design for Part B was very similar to Part A. The address register a1 still initially points to 0x2300000, but this time, a2 now points to 0x2320000, which is where the converted data is stored.The same data register d2 was used to temporarily store the input and process the data. Just like in Part A, the *Setzeros.s* and the *DataSrorage.s* files were used to initialize memory contents as well.

Once again, we had a loop branch that initially loads input data from the location pointed to by a1 into data register d2, and the ASCII 'Enter' code still terminates the program, as before. In Part B, valid inputs are the ASCII characters A-Z and a-z. (0x41-0x5A and 0x61-0x7A) Error handling was also similar to Part A, where the value 0xFFFFFFFF was stored at the memory location pointed to by a2. We had two branches to handle the two ranges of accepted characters.

For example, if an input character was in the range a-z, a difference was taken relative to the ASCII character 'a', and added to the ASCII character 'A'. The converted value is then stored at the memory location pointed to by a2, and the addresses stored in a1 and a2 are incremented by 4 before the loop starts over.

3 Testing

3.1 Part A

Designs for the MUX and DEMUX schematics were created in Xilinx ISE, as outlined in Figures 2 and 3 in Section III. From the designs created, two custom symbols for the MUX and DEMUX were made, as shown in Figure 4, which allows for the use of modular design, which has the advantage of allowing components to be re-used, or allowing changes made to the component to be automatically transferred throughout the design hierarchy. The custom symbols can be created by expanding the Design Utilities menu under the Processes window, and double clicking on Create Schematic Symbol. Next, the synthesis and pin assignment steps were done. Input signals were wired to separate switches on the development FPGA, and output signals were wired to separate LEDs on the board. The design was verified using ISim by first creating a VHDL script and coding the expected outputs, as determined by the circuit design and prelab. A picture of the DEMUX test can be found in Figure 6. Finally, a programming file was created and the bitstream was uploaded to the FPGA board using Adept.

3.2 Part B

A schematic for the control circuit was created using Xilinx ISE according to the design in the lab manual, and developed in the prelab. Like in part one, the synthesis and pin assignment steps were done, with inputs being wired to separate switches, and the outputs being wired to separate LEDs. A VHDL script was created, with the truth table in Table 3 coded in to test the outputs, and the design was tested using ISim (Figure 7). Finally, a programming file was generated, and the bitstream was uploaded to the development board FPGA using Adept.

4 Questions

4.1 Question 1

"What happens when there is no exit code 0x0D provided in the initialization process? Would it cause a problem? Why or why not?"

A: Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

4.2 Question 2

"How can our code be modified to provide a variable address range? For example, what if I only wanted to convert the first 10 data entires? "

A: Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Donec odio elit, dictum in, hendrerit sit amet, egestas sed, leo. Praesent feugiat sapien aliquet odio. Integer vitae justo. Aliquam vestibulum fringilla lorem. Sed neque lectus, consectetuer at, consectetuer sed, eleifend ac, lectus. Nulla facilisi. Pellentesque eget lectus. Proin eu metus. Sed porttitor. In hac habitasse platea dictumst. Suspendisse eu lectus. Ut mi mi, lacinia sit amet, placerat et, mollis vitae, dui. Sed ante tellus, tristique ut, iaculis eu, malesuada ac, dui. Mauris nibh leo, facilisis non, adipiscing quis, ultrices a, dui.

5 Conclusion

Morbi luctus, wisi viverra faucibus pretium, nibh est placerat odio, nec commodo wisi enim eget quam. Quisque libero justo, consectetuer a, feugiat vitae, porttitor eu, libero. Suspendisse sed mauris vitae elit sollicitudin malesuada. Maecenas ultricies eros sit amet ante. Ut venenatis velit. Maecenas sed mi eget dui varius euismod. Phasellus aliquet volutpat odio. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Pellentesque sit amet pede ac sem eleifend consectetuer. Nullam elementum, urna vel imperdiet sodales, elit ipsum pharetra ligula, ac pretium ante justo a nulla. Curabitur tristique arcu eu metus. Vestibulum lectus. Proin mauris. Proin eu nunc eu urna hendrerit faucibus. Aliquam auctor, pede consequat laoreet varius, eros tellus scelerisque quam, pellentesque hendrerit ipsum dolor sed augue. Nulla nec lacus.

Suspendisse vitae elit. Aliquam arcu neque, ornare in, ullamcorper quis, commodo eu, libero. Fusce sagittis erat at erat tristique mollis. Maecenas sapien libero, molestie et, lobortis in, sodales eget, dui. Morbi ultrices rutrum lorem. Nam elementum ullamcorper leo. Morbi dui. Aliquam sagittis. Nunc placerat. Pellentesque tristique sodales est. Maecenas imperdiet lacinia velit. Cras non urna. Morbi eros pede, suscipit ac, varius vel, egestas non, eros. Praesent malesuada, diam id pretium elementum, eros sem dictum tortor, vel consectetuer odio sem sed wisi.

6 Appendix

6.1 Part A Assembler Code

```
/* DO NOT MODIFY THIS ----------------------------------------------*/
.text

.global AssemblyProgram

AssemblyProgram:
lea      -40(%a7),%a7 /*Backing up data and address registers */
movem.l %d2-%d7/%a2-%a5,(%a7)
/*-----------------------------------------------------------------*/

/******************************************************************/
/* General Information *******************************************/
/* File Name: Lab1a.s ********************************************/
/* Names of Students: Arun Woosaree and Navras Kamal            **/
/* Date: 1/29/2018                                               **/
/* General Description:                                          **/
/* Converts ASCII letters 0-9, a-f, A-F to hex value 0x0-0xF     **/
/******************************************************************/

/*Write your program here*****************************************/

movea.l #0x2300000, %a1              /* save input  address to a1*/
movea.l #0x2310000, %a2              /* save output address to a2*/

/* let a value in quotation marks be the ASCII value of the character
   enclosed by the quotation marks*/

loop:                /* the looping function                      */
 move.l (%a1), %d2   /* move the value at address a1 to d2,       */
                     /* call this 'inval' from henceforth         */

 cmp.l #0x0D, %d2    /* Check if the inval is the enter code      */
 beq end             /* if it is, go to the end of the program    */
                     /* (breaking the loop)                       */

 cmp.l #0x2F, %d2    /* compare inval to the hex value of "0"     */
 blt err             /* if inval is less than ASCII zero          */
                     /* it is not valid, throw an error           */

 cmp.l #0x3A, %d2    /* compare the inval to the hex value of ":", */
                     /* which is one ASCII value higher than "9"  */
 blt zeronine        /* if it is less than the value of ":"       */
```

```
                    /* then it must be a value between "0" and "9"*/
                    /* thus go to the proper part of the code to  */
                    /* handle this value                          */

 cmp.l #0x41, %d2   /* compare the inval to "A"                   */
 blt err            /* if it is less than the "A" than it is      */
                    /* invalid, throw an error                    */

 cmp.l #0x47, %d2   /* compare the inval to "G"                   */
 blt bigathruf      /* if it is less than the value of "G" then   */
                    /* it must be in the range "A" through "F"    */
                    /* thus go to the part of the code to handle  */
                    /* these values                               */

 cmp.l #0x61, %d2   /* compare the inval to "a"                   */
 blt err            /* if it is in this range it is invalid,      */
                    /* thus throw an error                        */

 cmp.l #0x67, %d2   /* compare the inval to "g"                   */
 blt littleathruf   /* if it is less than "g" then it must be in  */
                    /* the range "a" through "F"                  */
                    /* thus go to the part of the code to handle  */
                    /* these values                               */

/*    if inval is above "f" then the code will continue here      */

err:
 move.l #0xFFFFFFFF, (%a2)   /* throw the error code to the output*/
                             /* address location                  */
 bra endloop                 /* go to the end of the loop before  */
                             /* restarting the loop               */

zeronine:           /* inval is between "0" and "9"               */
 sub.l #0x30, %d2   /* subtract the hex value of "0" from inval,  */
                    /* which will leave a value from 0x0 to 0x9,  */
                    /* for "0" to "9" respectively                */
 move.l %d2, (%a2)  /* move this calculted hex value to the output*/
                    /* address location                           */
 bra endloop        /* go to the end of the loop before restarting*/
                    /* the loop                                   */

bigathruf:          /* inval is between "A" and "F"               */
 sub.l #0x41, %d2   /* subtracts the hex value of "A" d2. This is */
                    /* the difference between d2 and the character*/
                    /* and "A"                                    */
 add.l #0xA, %d2      /* adds the value of "A" to d2, which will    */
                      /* make it into the hex representation of the */
                      /* original ASCII value                       */
 move.l %d2, (%a2)    /* move this value to the output address      */
                      /* location                                   */
 bra endloop          /* go to the end of the loop before restarting*/
                      /* the loop                                   */

littleathruf:         /* inval is between "a" and "f"               */
 sub.l #0x61, %d2     /* subtracts the hex value of "a" d2.         */
                      /* This is the difference between d2          */
                      /* and the character and "a"                  */
 add.l #0xA, %d2      /* adds the value of "a" to d2, which will    */
                      /* make it into the hex representation of     */
                      /* the original ASCII value                   */
 move.l %d2, (%a2)    /* move this value to the output address      */
                      /* location                                   */
 bra endloop          /* go to the end of the loop before restarting*/
                      /* the loop                                   */

endloop:              /* handles code to be executed before the     */
                      /* start of a new loop                        */
 add.l #0x4, %a1      /* increment the input address by 4           */
 add.l #0x4, %a2      /* increment the output address by 4          */
 bra loop             /* restart the loop                           */

end:                  /* end the custom part of the program         */

/*End of program **********************************************/

/* DO NOT MODIFY THIS ------------------------------------------*/
movem.l (%a7),%d2-%d7/%a2-%a5 /*Restore data and address registers */
lea      40(%a7),%a7
rts
/*-------------------------------------------------------------*/
```

6.2 Part B Assembler Code

```
/* DO NOT MODIFY THIS ----------------------------------------------------*/
.text

.global AssemblyProgram

AssemblyProgram:
lea      -40(%a7),%a7 /*Backing up data and address registers */
movem.l %d2-%d7/%a2-%a5,(%a7)
/*-----------------------------------------------------------------------*/

/***********************************************************************/
/* General Information *********************************************/
/* File Name: Lab1b.s **********************************************/
/* Names of Students: Arun Woosaree and Navras Kamal                **/
/* Date: 1/29/2018                                                  **/
/* General Description:                                             **/
/*                                                                  **/
/***********************************************************************/

/*Write your program here*******************************************/

movea.l #0x2300000, %a1     /* save input address to a1          */
movea.l #0x2320000, %a2     /* save output address to a2         */

/* let a value in quotation marks be the ASCII value of the         */
/* character enclosed by the quotation marks                        */

loop:                       /* the looping function               */
 move.l (%a1), %d2          /* move the value at address a1 to d2 */
                            /* Call this 'inval' from henceforth  */

 cmp.l #0x0D, %d2           /* Check if the inval is enter code   */
 beq end                    /* if it is, go to the end of the     */
                            /* program (breaking the loop)        */

 cmp.l #0x41, %d2           /* compare the inval to "A"           */
 blt err                    /* if it is less than the "A" than it */
                            /* is invalid, throw an error         */

 cmp.l #0x5B, %d2           /* compare the inval to "["           */
 blt bigathruz              /* if it is less than the value of    */
                            /* "[" then it must be in the range   */
                            /* "A" through "Z"                    */
                            /* thus go to the part of the code to */
```

```
                             /* handle these values             */

 cmp.l #0x61, %d2            /* compare the inval to "a"        */
 blt err                     /* if it is in this range it is    */
                             /* invalid, thus throw an error    */

 cmp.l #0x7B, %d2            /* compare the inval to "{"        */
 blt littleathruz            /* if it is less than "{" then it is */
                             /* in the range "a" through "z"    */
                             /* thus go to the part of the code to */
                             /* handle these values             */

bigathruz:                   /* inval is between "A" and "Z"    */
 add.l #0x20, %d2            /* adds the hex difference between */
                             /* "A" and "a", making it into the */
                             /* lowercase equivalent            */
 move.l %d2, (%a2)           /* move this value to the output   */
                             /* address location                */
 bra endloop                 /* go to the end of the loop before */
                             /* restarting the loop             */

littleathruz:                /* inval is between "a" and "z"    */
 sub.l #0x20, %d2            /* subtracts the hex difference    */
                             /* between "a" and "A", changing it */
                             /* into the uppercase equivalent   */
 move.l %d2, (%a2)           /* move this value to the output   */
                             /* address location                */
 bra endloop                 /* go to the end of the loop before */
                             /* restarting the loop             */

 /* if inval isn't a valid character then code will continue here */
err:
 move.l #0xFFFFFFFF, (%a2)   /* throw the error code to the output */
                             /* address location                */
 bra endloop                 /* go to the end of the loop before */
                             /* restarting the loop             */

endloop:                     /* handles code to be executed before */
                             /* the start of a new loop         */
 add.l #0x4, %a1             /* increment the input address by 4 */
 add.l #0x4, %a2             /* increment the output address by 4 */
 bra loop                    /* restart the loop                */

end:                         /* end of custom program section   */
```

```
/*End of program **********************************************/

/* DO NOT MODIFY THIS ------------------------------------------*/
movem.l (%a7),%d2-%d7/%a2-%a5 /*Restore data and address registers */
lea       40(%a7),%a7
rts
/*-------------------------------------------------------------*/
```

7 Marking Sheet