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	xxxxxxx

Contents

1 Introduction

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

2 Design

2.1 Part A

part a Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

2.2 Part B

partb Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

3 Testing

3.1 Part A

part a Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas

eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

3.2 Part B

partb Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

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?" answer goes here

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? " answer goes here

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

conclusions 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:                                       **/
/*                                                            **/
/************************************************************/

/*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*/

err:                                              /* if the inval is
   equal to or above "g" then the code will naturally continue here*/
 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 the
    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 must be 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*/
/*TODO*/
littleathruz:                           /* inval is between "a" and "z
   "*/
sub.l #0x20, %d2                        /* subtracts the hex difference
     between "a" and "A", making 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*/
/*TODO*/
err:                                                /* if the inval is not
    a valid character then the code will naturally continue here*/
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 program ************************************************/

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