COMPUTER ORGANIZATION
EECS 2021

LAB "B" REPORT
TRANSLATING CODE TO ASSEMBLY

The work in this report is my own. I have read and understood York University academic dishonesty policy and I did not violate the senate dishonesty policy in writing this report.

X ____________________

Camillo John (CJ) D'Alimonte
212754396
Lab Section 2
09/30/14
Professor Aboelaze

Abstract:

This laboratory/experiment was conducted in an attempt to learn the basics behind the process of translating high level Java code into Assembly. By programming and testing using xspim, I will be able to write different Assembly programs that will translate Java code using different operations. The methods used in Assembly consist of arithmetic operations, logical and arithmetic shifting, hexadecimal conversions, and masking. By the end of the lab, I will have a basic understanding on how a high level programming language is converted to lower level Assembly language.

Equipment Used:

This lab was done through the UNIX environment at the Lassonde Lab. I used the Eclipse IDE for the Java applications and the UNIX terminal (pico editor) to write my Assembly code. I used xspim to test my Assembly code on different input.

Methods/Procedures:

For LabB1, I needed to translate a Java application into assembly code (MIPS). This lab introduced me to using the xspim application. The code was given to us in the lab manual.

Pseudo Code	The code was provided to us in the Lab Manual.

For LabB2, I needed to modify my MIPS code from LabB1 by adding a specific header and footer as well as adding some breakpoints in xspim.

Pseudo Code	.globl fini .text main: #--------------------- **LabB1 Code** fini: jr $ra # return

For LabB3, I needed to modify my MIPS code from LabB2 by changing the service number from 1 to 11. I also needed to alter my Java code by adding a cast of a0 to a char → (char) a0.

Pseudo Code	addi $v0, $0, 11 # service #11

For LabB4, I needed to modify my MIPS code in order to print the sum and the difference of $t0 and $t1 separated by a space.

Pseudo Code	To print a space: • change service number to 11 • print the space using *addi* & space character To calculate the difference: • subtract $t0 from $t1, print their difference

	• subtract \$t0 from \$t1, using *sub* and store in \$a0 • change to service 1; use print command

For LabB5, I needed to modify my MIPS code in order to take in an input from the user using service #5, readInt.

Pseudo Code	Receive input from user: • change service # to 5 • use syscall • add input from \$v0 to \$t0

For LabB6, I needed to modify my MIPS code in order to compare two integer values and output their sum if the numbers were equal or their difference if otherwise. I needed to follow this logic:

```
if ($t0 == $t1) branch to XX;
print($t0 - $t1);
jump to YY;
XX: print($t0 + $t1);
YY: rest of program
```

Pseudo Code	Add two values if user input is equal to #t1: • use beq for branching and label code XX for addition • after beq, use j YY to if branching to addition does not happen program can jump to subtraction • after addition, jump to label fini • If user input is not equal to #t1, then use YY for subtraction

For LabB7, I needed to modify my LabB6 so that if \$t0 was less than \$t1, then I would output their sum. If \$t0 was greater, then I would output the difference. The following MIPS code was given:

```
slt $x, $y, $z
```

Pseudo Code	Add two values if \$t0 < \$t1: • use slt to determine if \$t0 < \$t1 • if yes, jump to XX, jump to YY if not • from XX code, add \$t0 and \$t1 • jump to fini • if \$t0 is not less, then subtract at label YY

For LabB8, I had to create a new program from scratch. The code was given to me in the lab manual. I had to replace the 2nd last line (before the jump) with the following: add $a0, $0, 1. No pseudo code is needed as the code was provided to me.

For LabB9, I had to create a program that translated the logic of a Java for loop. The logic provided is below:

```
t5 = 0;
loop: if (! t5 < t0) branch to done;
s0 = s0 + t5;
t5++;
jump to loop;
done: System.out.print(s0);
```

Pseudo Code	Declare s0 and get input from user: • change to service # to 5 • do syscall • take input from v0 and store in t1 • declare loop counter t5 • if t5 is less than the user input, then add t5 to s0, output s0 • if not, then branch out of the loop, output s0

For LabB10, I had to create a program that divided integers using the div, mflo, and mfhi commands. Some code was already provided to me.

Pseudo Code	Modify existing code by doing the following: • change to service #1; print quotient stored in a0 • change to service #11; print a space • store result of mfhi into a0 • change service # to 1; print remainder stored in a0 • multiply t0 & t1 • store result of mflo in a0 • change to service #1; print the product stored in a0

For LabB11, I needed to create a new program that translates Java shift operators into Assembly. Once again, code was given to me and similar to LabA, I had to modify the shift amounts from 1 to 2.

Pseudo Code	Replace comments in given code with: • print a0: change to service #1;syscall • print ' ': change to service#11; syscall • change shift from 1 to 2 on both srl and sll lines

	• set breakpoint after srl line; record value of registers a0 and t0

For LabB12, I needed to write a program that reads an integer x from the user and outputs 18x, (x multiplied by 18) without using mult.

Pseudo Code	Receive input: • shift input value left by 4 and store in t1 • shift input value left by 1 and store in t2 • add t1 & t2 and store in a0 • change service # to 1; syscall

For LabB13, I needed to write a program that determined and outputted the value of bit #10 in $t0 using the following logic:

a0 = t0 << 21;
a0 = a0 >>> 31;

Pseudo Code	Receive input: • shift input value left by 21; store in t1 • shift input value left by 31; store in a0 • change service # to 1; syscall

For LabB14, I needed to write a program that reads an integer x from the user and outputs 18x, (x multiplied by 18) without using mult.

Pseudo Code	Receive input: • shift input value left by 4 and store in t1 • shift input value left by 1 and store in t2 • add t1 & t2 and store in a0 • change service # to 1; syscall

For LabB15, I needed to write a program that clears bit #10 in $t0. The mask is defined as follows:

t5 = 0xffff;
t5 = t5 << 16;
t5 = t5 | 0xfbff;

Pseudo Code	Receive input: • find the OR of 0 and 0xfff and store in t5 • shift t5 left logically by 16 bits to move upper bits • do an OR of t5 and 0xfbff to store in t5 again • AND t5 and input and store in $a0

For LabB16, I needed to modify LabB15 to implement the same mask but using a different algorithm. That algorithm is as follows:

```
t6 = 1024;
t6 = ~t6;
```

Pseudo Code	Receive input: • do the NOR (negation OR) of 0 and mask 1024 to get negation of 1024 & store in t6 • AND the input and t6 and store in a0

For LabB17, I needed to modify LabB15/16 to implement the same mask but using a different command, lui.

Pseudo Code	Receive input: • use lui to store 0xffff in the upper half of the negation 1024 in t7 • find the OR of t7 and 0xfbff to place the lower half of the negation of 1024 in t7 • AND the input and t7 and store in a0

For LabB18, I needed to write a program that implemented the exclusive or command.

Pseudo Code	Receive input: • do the XOR (exclusive or) of input and 1024 • change service # to 1; syscall

<u>Results:</u>

The following table describes the various results I recorded for each question in the lab.

Lab Number	Question	Input or Question Asked	Output or Answer
1	10	How many statements are executed before the first statement in your program?	5
	11	Value of $v0?	1
3	17	What will be the output of our Java app above if we replace its last statement with: System.out.print((char)a0);	C
	20	What is the output for a0?	C was predicted. C was outputted.
4	21	$t0 = 60, $t1 = 7	67 53
5	23	$t0 = 60	C was predicted. C was outputted.
6	24	$t0 = 8	15
		$t0 = 6	13

7	25	$t0 = 12	5
		$t0 = 5	12
8	26	Make a prediction; record your result	Predicted: 01234
	27		Result: 01234
	28		Error: Indefinite Loop
9	29	$t0 = 9	36
10	30	Make a prediction; record your result	8 4
	31		30 120
11	33	Make a prediction; record your result	$t0 = 3c $a0 = 1
	34		15 240
	35		-15 -240
	36/37		-15 -240
12	38	x = 4	72
13	40 & 41	$t0 = 1024	1
14	43	$t0 = 1024	1
	44	$t0 = 0x400	1
15	47	$t0 = 5000	5000
		$t0 = 6000	4976
16	49	$t0 = 5000	5000
		$t0 = 6000	4976
17	52	$t0 = 5000	5000
		$t0 = 6000	4976
18	56	$t0 = 5000	6024

Discussion:

Overall, the first four labs proved to be very simple. Most of the code was given and the labs were more of an introduction to Assembly language than anything else. Setting breakpoints or modifying the service numbers turned out to be pretty straight forward. In LabB5, we were required to add a few changes to our code from LabB2. We needed to either add or subtract the two integers depending on if they were equal to each other. Again, this lab was straightforward and provided no difficulties. All expected input was correct.

LabB6-B9 focused on common Java structures such as conditional statements and loops. I learned the process of "jumping" commands (with XX, YY) as well as how to use branch statements such as beq and bne. Beq allows you to continuing branching if input is equivalent. When it's not, you're directed to a different statement. Bne provides the same functionality, but in reverse. Overall, all results were expected and once I got the hang of it, it wasn't difficult at all.

The lab manual did a very good job in helping me through LabB10. The multiplication and division functions were explained very thoroughly. Using the lo (to set the quotient) and hi (to set the remainder) commands, we were able to multiply and divide various inputs. All expected input was correct.

LabB11 and LabB12 revolved around shift operators. Once again, the shift operators proved to be difficult to understand (similar to LabA). It was difficult tracking what the shift operators were doing and predicting what the output may be. Shift operators are an area that I will have to continue reading about.

LabB13 through B16 focused once again on masking and the #10 bit. Bitwise operators were the main focus of the four labs. Although not that difficult, I focused more on trial and error to get through the masking techniques. LabB17 and Lab18 worked as expected and in general, were pretty straightforward. These labs were basically reviewing the same concepts as LabA. The lui command in B17 was a new concept but was simple to implement. The xor command is familiar from LabA and previous courses.

All in all, once I started getting used to the Assembly language syntax, the lab proved to be not that difficult.

Conclusion:

I was able to better my understanding on how code is converted into Assembly language. I now know the different operations that are used in the conversion process. I was introduced to how Java operations (i.e. conditional/loop statements, basic arithmetic) are converted into similar structures in Assembly (i.e. how to jump commands). I want to continue reading more references on how to translate shift operators as I found that concept difficult to grasp.

References:

No external references were used for this lab.

Appendix:

The source code for all 18 experiments is included below.

LabB1:

```
public class LabB1 {
	public static void main(String[] args) {
		int t0 = 60;
		int t1 = 7;
```

```
                int t2 = t0 + t1;
                //-------------------
                int a0 = t2;
                System.out.print((char)a0);
        }  }


.text
main:   #---------------------
        addi        $t0, $0, 60         # t0 = 60
        addi        $t1, $0, 7          # t1 = 7
        add         $t2, $t0, $t1       # t2 = t0+t1
        #---------------------
        addi        $v0, $0, 1          # service #1
        add         $a0, $0, $t2        # printInt
        syscall                         # do print
#---------------------
        jr          $ra                 # return
```

<u>LabB2:</u>

```
        .globl fini
        .text
main:   #---------------------
        addi        $t0, $0, 60         # t0 = 60
        addi        $t1, $0, 7          # t1 = 7
        add         $t2, $t0, $t1       # t2 = t0+t1
        #---------------------
        addi        $v0, $0, 1          # service #1
        add         $a0, $0, $t2        # printInt
        syscall                         # do print
        #---------------------
fini:   jr          $ra                 # return
```

<u>LabB3:</u>

```
        .globl fini
        .text
main:   #---------------------
        addi        $t0, $0, 60         # t0 = 60
        addi        $t1, $0, 7          # t1 = 7
        add     $t2, $t0, $t1       # t2 = t0+t1
        #---------------------
        addi    $v0, $0, 11         # service #11
        add     $a0, $0, $t2        # printInt
        syscall                     # do print
#---------------------
fini:   jr      $ra                 # return
```

<u>LabB4:</u>

```
        .globl fini
        .text
main:     #---------------------
        addi    $t0, $0, 60         # t0 = 60
        addi    $t1, $0, 7          # t1 = 7
        add     $t2, $t0, $t1       # t2 = t0+t1
        sub     $t3, $t0, $t1       # t3 = t0-t1
        #---------------------
        addi    $v0, $0, 1          # service #1
        add     $a0, $0, $t2        # printInt
        syscall                     # do print
        add     $v0, $0, 11         # service 11
        addi    $a0, $0, ' '        # print space
        syscall



        add     $v0, $0, 1          # service 1
        add     $a0, $0, $t3        # printInt
        syscall
#---------------------
fini:   jr      $ra                 # return
```

<u>LabB5:</u>

```
        .globl fini
        .text
main:     #---------------------
        addi    $v0, $0, 5          # v0 = readInt
        syscall
        add     $t0, $0, $v0
        addi    $t1, $0, 7          # t1 = 7
        add     $t2, $t0, $t1       # t2 = t0+t1
        #---------------------
        addi    $v0, $0, 1          # service #1
        add     $a0, $0, $t2        # printInt
        syscall                     # do print
#---------------------
fini:   jr      $ra                 # return
```

<u>LabB6:</u>

```
        .globl fini
        .text
main:     #---------------------
        addi    $v0, $0, 5          # v0 = readInt
        syscall
        add     $t0, $0, $v0
        addi    $t1, $0, 7          # t1 = 7
        add     $t2, $t0, $t1       # t2 = t0+t1
        sub     $t3, $t0, $t1       # t3 = t0-t1
        #---------------------
        addi    $v0, $0, 1          # service #1
        beq     $t0, $t1, XX
        add     $a0, $0, $t3
        j       YY
XX:     add     $a0, $0, $t2        # printInt
YY:     syscall                     # do print
#---------------------
fini:   jr      $ra                 # return
```

<u>LabB7:</u>

```
        .globl fini
        .text
main: #---------------------
          addi  $v0, $0, 5          # v0 = readInt
          syscall
          add   $t0, $0, $v0
          addi  $t1, $0, 7          # t1 = 7
        add     $t2, $t0, $t1       # t2 = t0+t1
        sub     $t3, $t0, $t1       # t3 = t0-t1
        #---------------------
        addi    $v0, $0, 1          # service #1
        slt     $t4, $t0, $t1
        beq     $t4, 1,XX
        add     $a0, $0, $t3        # printInt
        j       YY
XX:     add     $a0, $0, $t2
YY:     syscall                             # do print
        #---------------------
fini:   jr      $ra
```

<u>LabB8:</u>

```
       .globl fini
       .text
main:   #---------------------
       addi     $v0, $0, 1
       add      $a0, $0, $0
       #---------------------
loop:  slti     $t9, $a0, 5
       beq      $t9, $0, fini
       syscall              # do print
       addi     $a0, $a0, 1
       j loop
#--------------------
fini:  jr       $ra                 # return
```

<u>LabB9:</u>

```
       .text
       .globl fini
main:  #---------------------
            addi     $s0, $0, 0
            addi     $v0, $0, 5
            syscall
            add      $t0, $0, $v0
            addi     $v0, $0, 1
            add      $t5, $0, $0
       #------------------
```

```
loop:       slt         $t9, $t5, $t0
            beq         $t9, $0, done
            add         $s0, $s0, $t5
            addi        $t5, $t5, 1
            j loop
done:       add         $a0, $0, $s0
            syscall
      #-------------------
fini:       jr          $ra
```

LabB10:

```
      .text
      .globl fini
main: #--------------------
            addi        $t0, $0, 60
            addi        $t1, $0, 7
            div         $t0, $t1        # div t0/t1
            addi        $v0, $0, 1
            mflo        $a0
            syscall
            mfhi        $a0
            syscall
            #--------------------
            mult        $t0, $t1        # 64 bit product
            mflo        $a0             # move from lo
            syscall
            mfhi        $a0             # move from hi
            syscall
            #--------------------
fini:       jr          $ra
```

LabB11:

```
      .text
      .globl fini
main: #--------------------------
            addi        $t0, $0, 60
            addi        $v0, $0, 1
            sra         $a0, $t0, 1
```

```
                syscall
                addi            $v0, $0, 11
                addi            $a0, $0, ' '
                syscall
                addi            $v0, $0, 1
                sll             $a0, $t0, 1
                syscall
        #-------------------------
fini:           jr              $ra
```

<u>LabB12:</u>

```
        .text
        .globl fini
main: #-------------------------
                addi            $v0, $0, 5
                syscall
                add             $t0, $0, $v0
                addi            $t1, $0, 18
                mult            $t0, $t1
                mflo            $t2
                mfhi            $t4
                addi            $v0, $0, 1
                add             $a0, $0, $t2
                syscall
                add             $a0, $0, $t4
                syscall
                sll             $t5, $t0, 4    # multiplying by 2 to the 4
                sll             $t6, $t0, 1    # multiplying by 2
                add             $a0, $t5, $t6  # adding the two numbers
                syscall
          #-------------------------
fini:           jr              $ra
```

<u>LabB13:</u>

```
        .text
        .globl fini
main:   #----------------------
                addi            $v0, $0, 5
                syscall
                add         $t0, $0, $v0
                addi        $v0, $0, 1
                sll         $a0, $t0, 21     # shift left logically
                srl         $a0, $a0, 31     # shift right logically
                syscall
        #--------------------------
fini:   jr          $ra
```

<u>LabB14:</u>

```
        .text
        .globl fini
main:   #-----------------------
                addi        $v0, $0, 5
                syscall
                add         $t0, $0, $v0
                addi        $v0, $0, 1
                andi        $a0,$t0,0x400 # mask - 1024
                beq         $a0, $0, done
                addi        $a0, $0, 1
        #----------------------------
done:           syscall
        #----------------------------
fini:   jr          $ra
```

<u>LabB15:</u>

```
        .text
        .globl fini
main:   #-----------------------
        addi        $v0, $0, 5
        syscall
        add         $t0, $0, $v0
          addi      $v0, $0, 1
        ori         $t5,$0, 0xffff
        sll         $t5, $t5, 16         # logica shift left
        ori         $t5, $t5, 0xfbff     # used ori instead of addi
        and         $a0,$t0,$t5
        syscall
        #------------------------------
fini:   jr          $ra
```

<u>LabB16:</u>

```
        .text
        .globl fini
main:   #----------------------
                addi        $v0, $0, 5
                syscall
                add         $t0, $0, $v0
                addi        $v0, $0, 1
                ori         $t5,$0, 0xffff
                sll         $t5, $t5, 16
                ori         $t5, $t5, 0xfbff
                and         $a0,$t0,$t5
        #--------------------------------------------
                syscall
                addi        $t6, $0, 1024
                nor         $t6, $0, $t6
                and         $a0, $t0, $t6
                syscall
        #--------------------------------------------
fini:           jr           $ra
```

<u>LabB17:</u>

```
        .text
        .globl fini
main:   #----------------------
                addi        $v0, $0, 5
                syscall
                add         $t0, $0, $v0
                addi        $v0, $0, 1
                ori         $t5,$0, 0xffff
                sll         $t5, $t5, 16        # logical shift left

                ori         $t5, $t5, 0xfbff
                and         $a0,$t0,$t5
                syscall
```

```
            addi    $t6, $0, 1024
            nor     $t6, $0, $t6
            and     $a0, $t0, $t6
            syscall
            lui     $t7, 0xffff     # lower upper immediate
            ori     $t7, $t7, 0xfbff
            and     $a0, $t0, $t7
            syscall
        #---------------------------
fini:       jr      $ra
```

<u>LabB18:</u>

```
        .text
        .globl fini
main:   #--------------------
            addi    $v0, $0, 5
            syscall
            add     $t0, $0, $v0
            addi    $v0, $0, 1
            xori    $a0, $t0, 1024      # xor
            syscall
        #-----------------------
fini:       jr      $ra
```