COMPUTER ORGANIZATION
EECS 2021

LAB "C" REPORT
TRANSLATING UTILITY CLASSES

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
10/10/14
Professor Aboelaze

Abstract:

This laboratory/experiment was conducted in an attempt to learn the basics behind the process of translating high level Java code, specifically utility classes into Assembly. By programming and testing using xspim and by familiarizing myself with important MIPS operations, I will be able to write different Assembly programs that will translate Java code. By the end of the lab, I will have improved my growing understanding of 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 LabC1, I needed to create and run two separate Java classes, one of which was a utility class. I then had to convert the IntegerMath class into MIPS and then investigate some of the different operations and what their functions were in xspim. The source code was given to us in the lab manual.

For LabC1Client, I needed to write a MIPS program that would translate my Java code from my IntegerMathClient class. Once again, the source code was provided to us in the lab manual.

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

For LabC2, I needed to modify my MIPS code from LabC1 by changing the SIZE from 32 to -32.

For LabC2Client, I needed to investigate the lb and lh commands which store a piece of data represented in less than 32 bits in a register that is 32-bit wide. Since my code's output was unexpected, I needed to make some necessary adjustments in order to fix any bugs in my program.

Pseudo Code	SIZE = .byte set to -32
Pseudo Code (Client)	lw $4 = 0x10010000 print MAX SIZE = lb unsigned $a0 print SIZE

For LabC3, no changes were made to the original program from LabC2.

For LabC3Client, I needed to modify the client so that it functioned the same as before (and produced the same output) yet it did not refer to the SIZE symbol. The following code was provided to us:

```
lb $a0, SIZE($0)
addi $t0, $0, 4
lb $a0, MAX($t0)
```

Pseudo Code	SIZE = .byte set to -32
Pseudo Code (Client)	lw $4 = 0x10010000 print MAX $t0 = 4 MAX $t0 = lb at $a0 print SIZE

For LabC4, we were given two newly modified Java classes to convert into MIPS code. LabC4 was written based off the code from LabC1. The following code was provided to us:

```
                #-----------------------------
                        .text
getCount:       #-----------------------------
                lw      $v0,    count($0)
                jr      $ra
```

For LabC4Client, I needed to modify my original LabC1Client with some new code that was provided to us. The jal (Jump and Link) instruction is similar to j except it stores PC (the program counter register) in $ra prior to jumping.

```
jal getCount
add $a0, $0, $v0
addi $v0, $0, 1
syscall
```

Pseudo Code	MAX = .word set to 2147483647 SIZE = .byte set to 32 count = .word set to 0 count = lw $v0
Pseudo Code (Client)	sp = sw $ra sp -= sp MAX = lw $a0 print MAX stored in $a0 SIZE = lb unsigned $a0 print SIZE stored in $a0 JLA to getCount $a0 = $v0 print the $a0 sp += 4

	sp = lw $ra

For LabC5 and LabC5Client, I needed to translate the setCount() mutator method using some of the code provided to us in the lab manual. This is the first time we used the println command in MIPS.

Pseudo Code	MAX = .word set to 2147483647 SIZE = .byte set to 32 count = .word set to 0 count = lw $v0 count = sw $a0
Pseudo Code (Client)	sp = store word $ra sp -= 4 MAX = lw $a0 print MAX stored in $a0 SIZE = lb unsigned $a0 print SIZE stored in $a0 JLA to getCount $a0 = $v0 print $a0 v0 is reading an integer from the input $a0 = $v0 JLA to setCount sp -= 4 sp = lw $ra

For LabC6 and LabC6Client, I needed to add to our utility a method similar to one in Java's Integer class (the signum method). I needed to use the count attribute as a hit counter for the method; i.e. the method should increment count whenever it is invoked:

Pseudo Code	MAX = .word set to 2147483647 SIZE = .byte set to 32 count = .word set to 0 count = lw $v0 count = sw $a0 <u>SIGNUM IMPLEMENTATION</u> $t0 = i<0 if $t0 != i {$v0 = -1} else if i == 0 {$v0=0} else {$v0=1}

	count= lw $t1 $t1 += 1 count= sw$t1
Pseudo Code (Client)	sp = store word $ra sp -= 4 MAX = lw $a0 print MAX SIZE = lb unsigned $a0 print SIZE JLA to getCount $a0 = $v0 print $a0 $v0 is reading an integer from input $a0 = $v0 JLA to setCount JLA to signum $a0 = $v0 print $a0 JLA to getCount $a0 = $v0 print $a0 sp += 4 sp = lw $ra

For LabC7, I needed to translate the Java code of the isPrime, isFactor methods. LabC7Client is a slight modification of LabC6Client.

Pseudo Code	MAX = .word set to 2147483647
	SIZE = .byte set to 32
	count = .word set to 0
	count = lw $v0 #getCount
	count = sw $a0 #setCount
	$t0 = true
	
	<u>IS PRIME IMPLEMENTATION</u>
	
	for $t1 = 2; $t0 && $t1 < $a0; $t1++
	{$t0 != isFactor($a0, $t1
	return $t0
	}
	
	<u>IS FACTOR IMPLEMENTATION</u>
	
	$t0 = ($a0 % $a1) == 0
	
	return $t0

Pseudo Code (Client)	sp = store word $ra sp -= 4 MAX = lw $a0 print MAX SIZE = lb print SIZE $v0 is reading an integer from the input $a0 = $v0 JLA to isPrime $a0 = $v0 print $a0 sp += 4 sp = lw $ra

For LabC8, I needed to add a method that printed out an integer in decimal vertically (one digit per line) starting with its most significant digit. Specifically, I needed to translate the following recursive method:

```
public static void printVertical(int n)  {
if (n > 0)  {
printVertical(n / 10);
System.out.println(n % 10);  }   }
```

Pseudo Code	MAX = .word set to 2147483647
	SIZE = .byte set to 32
	count = .word set to 0
	count= lw $v0 #getCount
	count= sw $a0 #setCount
	
	<u>PRINT VERTICAL IMPLEMENTATION</u>
	
	if $a0 > 0
	{ printVertical($a0 / 10
	print ($a0 % 10) }

Pseudo Code (Client)	sp = sw $ra sp -= 4 $v0 is reading an integer from input $a0 = $v0 JLA to printVertical sp += 4 sp = lw $ra

Results:

The following table describes the various results I recorded for each question in the lab. Each individual observation question is answered in the Discussion section.

Lab Number	Input or Question Asked	Output or Answer
1	No Input was Given to the Program	2147483647 32
2	No Input was Given to the Program	2147483647 224
3	No Input was Given to the Program	2147483647 -32
4	No Input was Given to the Program	2147483647 320
5	No Input was Given to the Program	2147483647 320 11
6	5	32 0 1 6
7	5	1
	6	0
8	2014	2 0 1 4

Discussion:

LabC begins as a continuation of Lab B, as we continue to build on our knowledge of Assembly/MIPS. For LabC, we need to implement static utility features, in this specific case, the IntegerMath utility. In LabC1, we explore the data pane in MIPS, and the different values of the stored data variables. After investigating what happens when we change the values of the three attributes, I found that it changed the order in which our values are outputted. My program output of 2147483647 32 was the expected values as they represent the values of SIZE and MAX.

In LabC2, I must change the value of the size from 32, to -32. The output from my program with this change was 2147483647 and 224. This was not the expected output. This is because we begin by changing 32 to binary, then we end up flipping it around and adding a 1 to it. To fix the problem in Q20, we have to add a u in front of the lb and the lh commands (i.e. ulb, ulh). Examining the first lw, we wonder why the text pane shows that SPIM did not handle it as a single instruction but rather as two. The reason for this "split" is that a non-jump instruction can only be 16-bit wide, thus causing it to break into two pieces. SPIM used $1 for the replacement but could have used other registers; however, programming conventions dictate that $1 is the best choice. For LabC3, we built upon the previous exercise but this time we needed to modify it slightly so it did not directly refer to the size symbol. The program's output after the modification was -32. Furthermore after modifying it to load a word from an address, I observed that it does not contain 0x0000ff20 as a value. In LabC4, we needed to implement the static getCount method. This was the first time using the jal command. We also learned how to implement the Stack pointer register, which grows backwards. After running the code, I received an output of 320 which is what I expected. All in all, the first four labs were fairly straight forward and did not prove to be a considerably difficult.

In LabC5, we needed to implement the static method setCount (a mutator). We had to use the sw and lw functions to load and store the data. This allowed us to store the instructions onto the stack, as well as have the ability to quickly remove it from the stack. After implementing the changes given in the lab manual, the program outputted 320 and 11, the expected value of SIZE. For LabC6, I had to build upon my program from Lab5, I needed to implement a test function that read an integer from the user and then printed it's signum as well as a count value. I needed to save registers using the caller and callee commands. These are always carried out on a stack, which allows for the increase of smaller addresses, with its last location pointing pointed at by $sp. We also learnt how to implement the integer function signum through this lab. This lab proved to be a little tricky to implement, especially with the saving of the registers.

In LabC7, we once again build upon the previous labs in an attempt to implement the static isPrime method. We had to use our previously attained knowledge from LabA and LabB in order to translate this method into Assembly. I found the loading and storing of the word instructions onto the stack a little difficult but all in all, the lab wasn't too difficult. After implementing the method and testing it, I received an output of 1 (true) when I inputted 5 and I received a 0 (false) when I inputted 6. These were the outputs I predicted. For LabC8, we started from scratch in an attempt to implement the static printVertical method, which takes in an input and prints it out vertically. This recursive method was difficult to implement into assembly code. It was a struggle trying to figure out the base case and when and where to recall the method. After taking some time to figure out how to implement this function, I received the following output of 2 0 1 4 vertically after I inputted 2014. This lab proved to be the most difficult of the eight exercises.

Conclusion:

I was able to better my understanding on how code, specifically utility classes are converted into Assembly language. I now know the different operations that are used in the conversion process. I was introduced to how Java utility classes (i.e. IntegerMath, arithmetic classes) are converted into similar structures in Assembly (i.e. how to use jump and link commands). I want to continue reading more references on how to use the signum command as I found that concept difficult to grasp.

References:

No external references were used for this lab.

Appendix:

The source code for all 8 experiments is included below.

LabC1 Java Programs:

```
public class IntegerMath  {

        public static final int MAX_VALUE = 2147483647;
        public static final byte SIZE = 32;
        private static int count = 0;
  }

public class IntegerMathClient   {

        public static void main(String[] args)   {

                System.out.print(IntegerMath.MAX_VALUE);
                System.out.print(IntegerMath.SIZE);
        }
}
```

LabC1:

```
            .globl  MAX
            .globl  SIZE
      #----------------------------
            .data
MAX:          .word         2147483647;
SIZE:        .byte         32;
count:       .word         0
      #----------------------------
```

```
        .text
```

LabC1 Client:

```
        .text
          #-------------------------

main:    lw          $a0,   MAX($0)
         addi        $v0,   $0,    1
         syscall

         addi        $a0,   $0,    10
         addi        $v0,   $0,    11
         syscall

         lbu         $a0,   SIZE($0)
         addi        $v0,   $0,    1
         syscall
         jr          $ra
```

LabC2:

```
      .globl  MAX
      .globl  SIZE
      #----------------------------
           .data
MAX:        .word        2147483647;
SIZE:       .byte        -32;
count:      .word        0
      #----------------------------
      .text
```

LabC2 Client:

```
        .text
      #-----------------------------------
main:    #lw          $a0, MAX($0)
         lw          $4, 0x10010000($0)
         addi        $v0, $0, 1
         syscall
```

```
        addi    $v0, $0, 11
        addi    $a0, $0, ``
        syscall

        addi    $v0, $0, 11
        addi    $a0, $0, 0x0A
        syscall

        lbu     $a0, SIZE($0)
        addi    $v0, $0, 1
        syscall
        jr      $ra
```

LabC3:

```
    .globl  MAX
    .globl  SIZE
    #-----------------------------
        .data
MAX:        .word       2147483647;
SIZE:       .byte       -32;
count:      .word       0
    #-----------------------------
    .text
```

LabC3 Client:

```
.text
        #-----------------
 main:    #lw        $a0,    MAX($0)
        lw      $4,     0x10010000($0)
        addi    $v0,    $0,     1
                syscall

        addi    $v0,    $0,     11
        addi    $a0,    $0,     ' '
                syscall

        addi    $v0,    $0,     11
```

```
        addi    $a0,    $0,     0x0A
                syscall

        #lbu    $a0,    SIZE($0)
        addi    $t0,    $0,     4
        lb      $a0,    MAX($t0)
        addi    $v0,    $0,     1
                syscall

        addi    $v0,    $0,     11
        addi    $a0,    $0,     0x0A
                syscall

        #addi   $t0,    $0,     3
        #lw     $a0,    MAX($t0)
        #addi   $v0,    $0,     1

                jr      $ra
```

LabC4 Java Programs:

```
public class IntegerMath   {

        public static final int MAX_VALUE = 2147483647;
        public static final byte SIZE = 32;
        private static int count = 0;

// Accessor for the count attribute
        public static int getCount()  {
        int result = IntegerMath.count;
        return result;
        }
}

public class IntegerMathClient   {

        public static void main(String[] args)  {
                System.out.print(IntegerMath.MAX_VALUE);
                System.out.print(IntegerMath.SIZE);
                System.out.println(IntegerMath.getCount());
        }
}
```

LabC4:

```
        .globl  MAX
        .globl  SIZE
        #----------------------------
                .data
MAX:            .word           2147483647;
SIZE:           .byte           32;
count:          .word           0
        #----------------------------
                .text
                .globl getCount
        #----------------------------
                 .text
getCount:       #----------------------------
              lw              $v0, count($0)
              jr              $ra
```

LabC4 Client:

```
.text
              #------------------
 main:
              sw              $ra,    0($sp)
              addi            $sp,    $sp,    -4


              lw              $a0,    MAX($0)
              addi            $v0,    $0,     1
              syscall

              addi            $v0,    $0,     11
              addi            $a0,    $0,     ' '
              syscall

              addi            $v0,    $0,     11
              addi            $a0,    $0,     0x0A
              syscall

              lbu             $a0,    SIZE($0)
```

```
        addi    $v0,    $0,     1
        syscall

        jal     getCount

        #--------------------

        add     $a0,    $0,     $v0
        addi    $v0,    $0,     1
        syscall

        addi    $sp,    $sp,    4
        lw      $ra,    0($sp)

        jr      $ra
```

LabC5:

```
  .globl  MAX
        .globl  SIZE
        .globl  getCount
        .globl  setCount
        .globl  println
    #-----------------------------------
        .data
MAX:        .word   2147483647;
SIZE:       .byte   32;
count:      .word   9
nextline:   .ascii  "\n"
    #-----------------------------------
            .text

getCount:   lw      $v0,    count($0)
            jr      $ra
setCount:   sw      $a0,    count($0)
            jr      $ra
println:    la      $a0,    nextline
            jr      $ra
```

LabC5 Client:

```
.text
        #--------------------------
main:   sw      $ra,    0($sp)
        addi    $sp,    $sp,    -4

        jal     getCount
        add     $a0,    $0,     $v0
        addi    $v0,    $0,     1
        syscall


        jal      println
        addi    $v0,    $0,     4
        syscall

        addi    $v0,    $0,     5
        syscall
        add     $a0,    $0,     $v0

        #---------------------------
        jal     setCount

        jal      getCount
        add     $a0,    $0,     $v0
        addi    $v0,    $0,     1
        syscall

        addi    $sp,    $sp,    4
        lw      $ra,    0($sp)
        jr      $ra
```

LabC6:

```
.globl  MAX
        .globl  SIZE
        .globl  getCount
        .globl  setCount
        .globl  println
        .globl  signum
```

```
            #-----------------------------------
                .data
MAX:        .word   2147483647;
SIZE:       .byte   32;
count:      .word   0
nextline:   .ascii  "\n"
            #-----------------------------------
            .text

getCount:       lw      $v0,    count($0)
                jr      $ra
setCount:       sw      $a0,    count($0)
                jr      $ra
println:        la      $a0,    nextline
                jr      $ra
signum:         beq     $a0,    $0,     ZERO
                slt     $t0,    $0,     $a0
                beq     $t0,    $0,     MINUS

            #-----------------------------------
            j ELSE

MINUS:  addi    $a0,    $0,     -1
            j       F
ZERO:       addi    $a0,    $0,     0
            j       F
ELSE:       addi    $a0,    $0,     1
            j       F

F:          lw      $v0,    count($0)
            addi    $v0,    $v0,    1
            sw      $v0,    count($0)
            jr      $ra
```

LabC6 Client:

```
.text
      #---------------------------
```

```
main:   sw    $ra,   0($sp)
        addi  $sp,   $sp,   -4

        jal   getCount
        add   $a0,   $0,    $v0
        addi  $v0,   $0,    1
        syscall


        jal   println
        addi  $v0,   $0,    4
        syscall

        addi  $v0,   $0,    5
        syscall
        add   $a0,   $0,    $v0


        jal   signum
        addi  $v0,   $0,    1
        syscall

        jal   println
        addi  $v0,   $0,    4
        syscall

        jal   getCount
        add   $a0,   $0,    $v0
        addi  $v0,   $0,    1
        syscall

        addi  $sp,   $sp,   4
        lw    $ra,   0($sp)
        jr    $ra
```

LabC7 Java Programs:

```
public static boolean isPrime(int i)   {

        boolean result = true;
```

```
        for (int candidate = 2; result && candidate < i; candidate++)  {
                result = !IntegerMath.isFactor(i, candidate); }
return result;

}

private static boolean isFactor(int n, int factor)  {
boolean result = (n % factor) == 0;
return result;
}
```

LabC7:

```
            .globl  MAX
            .globl  SIZE
            .globl  getCount
            .globl  setCount
            .globl  println
            .globl  signum
            .globl  isPrime
            .globl  isFactor

      #------------------------------------
            .data
MAX:          .word   2147483647;
SIZE:         .byte   32;
count:        .word   0
nextline:     .ascii  "\n"
true:         .asciiz "true"
false:        .asciiz "false"
      #------------------------------------
            .text
getCount:     lw      $v0,    count($0)
            jr      $ra
setCount:     sw      $a0,    count($0)
            jr      $ra
println:      la      $a0,    nextline
            jr      $ra

      #---------------------------------------
signum:   slt     $t0,    $a0,    $0
            beq     $t0,    $0,     MINUS
```

```
        beq     $a0,    $0,     ZERO

        j ELSE


    MINUS:      addi    $a0,    $0,     -1
                j       F
    ZERO:       addi    $a0,    $0,     0
                j       F
    ELSE:       addi    $a0,    $0,     1
                j       F
    F:          lw      $v0,    count($0)
                addi    $v0,    $v0,    1
                sw      $v0,    count($0)
                jr      $ra

    #---------------------------------------------

isPrime:        sw      $ra,    0($sp)
                addi    $sp,    $sp,    -4
                add     $t5,    $0,     $a0     #t5 =   input
                la      $t0,    true            #result = true
                la      $t3,    true            #t3 = true
                la      $t4,    false           #t4 = false
                addi    $t1,    $0,     2       #t1 = candiate = 2
    Loop:       bne     $t0,    $t3,    Done
                slt     $t2,    $t5,    $t1
                bne     $t2,    $0,     Done
                add     $s0,    $0,     $t5     #parameter1 = t5
                add     $s1,    $0,     $t1     #parameter2 = t1

        jal isFactor

        #----------------------------------------
            beq     $a0,    $t3, False
            la      $t0, true
            addi    $t1,    $t1,    1

            j Loop
```

```
False:      la      $t0, false
            addi    $t1,    $t1,    1

            j Loop

Done:       add     $a0,    $0,     $t0
            addi    $sp,    $sp,    4
            lw      $ra,    0($sp)
            jr      $ra


isFactor:   div     $s0,    $s1
            mfhi    $t6
            bne     $t6,    $0,     False2
            la      $a0,    true
            j       $ra
False2:     la      $a0, false
            j $ra
```

LabC7 Client:

```
        .text
    #------------------------------

main:       sw      $ra,    0($sp)
        addi    $sp,    $sp,    -4

        addi    $v0,    $0,     5
        syscall
        add     $a0,    $0,     $v0

        jal     isPrime
        addi    $v0,    $0,     4
        syscall

        addi    $sp,    $sp,    4
        lw      $ra,    0($sp)
        jr      $ra
```

LabC8:

```
        .text
        #----------------------------
main:   sw          $ra,    0($sp)
        addi        $sp,    $sp,    -4

        addi        $v0,    $0,     5
        syscall
        add         $s0,    $0,     $v0

        #----------------------------
        jal         printVertical

        addi        $sp,    $sp,    4
        lw          $ra,    0($sp)
        jr          $ra
```

LabC8 Client:

```
            .globl  MAX
            .globl  SIZE
            .globl  println
            .globl  printVertical
        #-----------------------------------
            .data
MAX:          .word   2147483647;
SIZE:        .byte   32;
count:       .word   0
nextline:     .ascii  "\n"
        #-----------------------------------
            .text
println:     la     $a0,    nextline
            jr      $ra

printVertical:
            sw      $ra,    0($sp)
            addi    $sp,    $sp,    -4
            add     $t0,    $0,     $s0
            sgt     $t1,    $t0,    $0
```

```
        beq     $t1,    $0,     Done
        addi    $t2,    $0,     10
        div     $t0,    $t2
        mflo    $s0
        mfhi    $t3

        sw      $t3,    0($sp)
        addi    $sp,    $sp,    -4

        jal     printVertical

        addi    $sp,    $sp,    4
        lw      $a0,    0($sp)
        addi    $v0,    $0,     1
        syscall

        addi    $v0,    $0,     4
        la      $a0,    nextline
        syscall
#-------------------------------------
        j       Done
Done:   addi    $sp,    $sp,    4
        lw      $ra,    0($sp)
        jr              $ra
```