CprE 381 Homework 2

[Note: This homework gives you practice with MIPS assembly language. When you are asked to provide a program, try running it on MARS to confirm it works. MARS can also check your understanding of code tracing, etc. Plus you get more practice with the tools.]

1. MIPS Control Flow

   a. Assume $t0 holds the value 0x0000FFFF, $t1 holds the value 0x80000000, and $t2 holds the value 0x00000001. What is the value of $t2 after the following instructions?

```
            lui $t0, 0xFFFF
            slt $t1, $t1, $t0
            bne $t1, $0, SOMEPLACE
            addiu $t2, $t1, -1
            j EXIT
        SOMEPLACE:
            add $t2, $t2, $t1
        EXIT:
```

```
lui - $t0 = 0xFFFF0000 # lui clears lower bits
slt - $t1 = 1 # t1 is less than t0
bne - if t1 != 0, goto Someplace, t1 equals 1 goto someplace
add - $t2 = 0x00000001 + 1 = 0x000000002
Exits
So, $t2 = 2
```

   b. Translate the following C-style loop into MIPS assembly, assuming the following variables to register mappings:

int *a	$s0
int min	$s1
int i	$s2
int N	$s3
int *b	$s4

   All variables are 4 byte words. How many instructions are executed if N=1, N=10, N=100, N=1000?

```
int min = a[0];
  for (i=1; i<N; i++) {
   a[i] = (b[i]/8) + 11; // do NOT use div
                        // instruction
}

#int min = a[0];
#for (i=1; i<N; i++) {
#a[i] = (b[i]/8) + 11; // do NOT use div
# // instruction
#}
#int *a $s0
#int min $s1
#int i $s2
#int N $s3
#int *b $s4
.data
a_arr: .word 2 3 5 8
b_arr: .word 8 16 32 64
.globl main

.text
main:

la $s0, a_arr
la $s4, b_arr
addiu $s3, $zero, 4

-------Where actual code implemented------

lw $s1, 0($s0) #int min = a[0]

addiu $s2, $zero, 1 # i = 1
j conditional

loopbody:
sll $t9, $s2, 2 # i *= 4, this for word size
addu $t0, $s4, $t9 #[*b + i]
lw $t1, 0($t0) #$t1 = b[i]
srl $t1, $t1, 3 #t1 /= 8
addiu $t3, $t1, 11 # t1 += 11

addu $t2, $s0, $t9 #[*a + i]
sw $t3, 0($t2) # a[i] = $t1

increment:
addiu $s2, $s2, 1 # i++
conditional:
slt $t0, $s2, $s3 # t0 = 1 if i < N
bne $t0, $zero, loopbody #if t0 == 1 don't loop


Loop runs 10 commands + 5 constant (2 initial and 3
for condition start)

N starts at 1 soIt run N - 1 times

N = 1, 5 times (never enters loop)
N = 10, 90 + 5 constant
N = 100, 990 + 5 constant
N = 1000, 9990 + 5 constant
```

c. Write MIPS assembly for the following switch statement. Assume `score` is in `$a0` and `grade` is in `$v0`. Do NOT use a jump table as a compiler might, but rather use conditional branches.

```
if (score >= 90) {
 grade = ‘A’;
} else if (score >= 80) {
 grade = ‘B’;
} else if (score >= 70) {
 grade = ‘C’;
} else if (score >= 60) {
 grade = ‘D’;
} else {
 grade = ‘F’;
}

#if (score >= 90) {
# grade = ‘A’;
#} else if (score >= 80) {
# grade = ‘B’;
#} else if (score >= 70) {
# grade = ‘C’;
#} else if (score >= 60) {
# grade = ‘D’;
#} else {
# grade = ‘F’;
#}
#score $s0
```

```
#grade $v0

addiu $s0, $zero, 65

blt $s0, 90, elseif1
addiu $v0, $zero, 'A' #grade = 'A'
j exit

elseif1:
blt $s0, 80, elseif2
addiu $v0, $zero, 'B' #grade = 'B'
j exit

elseif2:
blt $s0, 70, elseif3
addiu $v0, $zero, 'C' #grade = 'C'
j exit

elseif3:
blt $s0, 60, else
addiu $v0, $zero, 'D' #grade = 'D'
j exit

else:
addiu $v0, $zero, 'F' #grade = 'F'

exit:
```

2. MIPS Assembly Language Design

a. P&H(2.21) <§2.6>. Provide a minimal set of MIPS instructions that may be used to implement the following pseudoinstruction *[We've talked about pseudoinstructions before with* ***mov****. Effectively they are assembly instructions that aren't actually in the ISA of hardware, so the assembler has to translate them into another machine instruction or series of machine instructions.]*:

```
not $t1, $t2 # bit-wise invert
```

```
nor $t1, $t2, $zero
```

b. You are tasked with adding a new pseudo-instruction that performs a right rotational shift.

```
ror $t0, $t1, imm # rotates $t1 imm bits right
```

Give a reasonably minimal implementation (i.e., don't use control flow instructions). Should this instruction produce any exceptions (i.e., can it produce any unexpected behaviors)? If so, what are they and does your version cause the same exceptions or introduce any more?

```
sll $at, $t1, 29 # Put first 3 bits into last 3 bits, put in temp reg
srl $t0, $t1, 3 # Open up last 3 bits
or $t0, $t0, $at # fill last 3 bits with temp reg
```

If the number is signed it won't work correctly.

c. Why does MIPS not have **add** `label_dst,label_src1,label_src2`, instructions in its ISA (there are at least two reasons for this)? *[Read this instruction's function as M[label_dst] = M[label_src1] + M[label_src2].]* Provide a concrete technical justification—you should have ideas both from lab and lecture.

Mips is register to register, not memory to memory.

The instruction length in MIPS is 32 bits and Memory is 32 bits.

3. MIPS Programming *[I suggest you actually run these programs to confirm that they work. Use MARS for MIPS runtime simulation.]*

a. Write a simple (you do not need to optimize—just use the direct implementation) C code snippet that implements the strncpy function from string.h (https://cplusplus.com/reference/cstring/strncpy/)

```
char* strncpy( char* destination, const char* source, size_t num)
{
        char curChar = '0';
        int seenNull = 0;
        for(int i = 0; i < num; i++)
        {
                if(!seenNull)
                {
                        curChar = source[i];
                }
                if(curChar == '\0')
                {
                        curChar = '0';
                        seenNull = 1;
                }
                destination[i] = curChar;
        }
        return destination;
}
```

```
char * strncpy(char * dst, const char * src, size_t num);
```

b. Translate your answer to part a into MIPS assembly. Make sure you use correct System-V ABI conventions as described in lecture (https://refspecs.linuxfoundation.org/elf/mipsabi.pdf). All variables in your C code should be initialized within your code.

```
strncpy:
# Put the address of array label in $s3
addiu $t0, $zero, 5 #t0 curChar, set to some arbitrary non zero value
addiu $t1, $zero, 0 #t1 seenNull

addiu $t2, $zero, 0 #i = 0
j conditional

loop:
        if1:
        seq $t3, $t1, 0 # if (seenNull == 0), t3 = 1
        bne $t3, 1, endif1 # if seenNull != 0 don't enter if

        #inside if 1
        addu $t4, $a1, $t2 # [source + i], dealing with chars no *4
        lb $t0, 0($t4) #curChar = source[i]

        endif1:
        if2:
        seq $t3, $t0, $zero #t3 = 1 if(curChar == '\0')
        bne $t3, 1, endif2 # if(curChar != '\0') goto end of if2

        #inside if 2
        addiu $t0, $zero, 48 # curChar = 0, ascii 0
        addiu $t1, $t1, 1 # seenNull = 1

        endif2:

        addu $t3, $a0, $t2 # [destination + i]
        sb $t0, 0($t3) # #destination[i] = curChar

increment:
addiu $t2, $t2, 1

conditional:
```

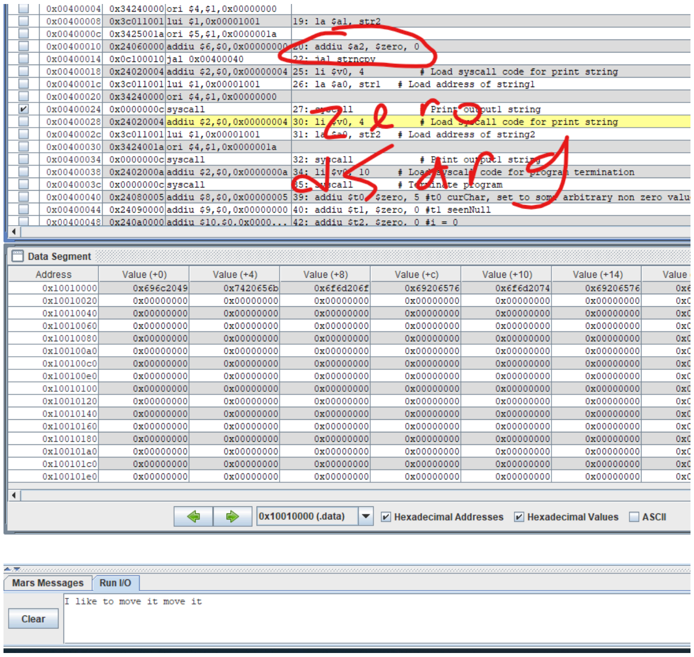
```
slt $t3, $t2, $a2 #t0 = t2 < num
beq $t3, 1, loop #t2 < num go to loop
#end loop

addu $v0, $a0, $zero #put destination as return value
jr $ra #jump back to function call
```

c. Add code that *calls* **`strncpy`** and prints the result for testing purposes. Provide *three* reasonable test cases for your MIPS assembly (inputs and expected outputs) and justify why you have included each one.

source = “waka”
destination = “I like to move it move it”

num = 0
Expected: “I like to move it move it”



```
☐ 0x00400004 0x34240000 ori $4,$1,0x00000000
☐ 0x00400008 0x3c011001 lui $1,0x00001001       19: la $a1, str2
☐ 0x0040000c 0x3425001a ori $5,$1,0x0000001a
☐ 0x00400010 0x24060000 addiu $6,$0,0x00000000  20: addiu $a2, $zero, 0
☐ 0x00400014 0x0c100010 jal 0x00400040          22: jal strncpy
☐ 0x00400018 0x24020004 addiu $2,$0,0x00000004  25: li $v0, 4          # Load syscall code for print string
☐ 0x0040001c 0x3c011001 lui $1,0x00001001       26: la $a0, str1   # Load address of string1
☐ 0x00400020 0x34240000 ori $4,$1,0x00000000
☑ 0x00400024 0x0000000c syscall                 27: syscall            # Print output1 string
☐ 0x00400028 0x24020004 addiu $2,$0,0x00000004  30: li $v0, 4          # Load syscall code for print string
☐ 0x0040002c 0x3c011001 lui $1,0x00001001       31: la $a0, str2   # Load address of string2
☐ 0x00400030 0x3424001a ori $4,$1,0x0000001a
☐ 0x00400034 0x0000000c syscall                 32: syscall            # Print output1 string
☐ 0x00400038 0x2402000a addiu $2,$0,0x0000000a  34: li $v0, 10     # Load syscall code for program termination
☐ 0x0040003c 0x0000000c syscall                 35: syscall        # Terminate program
☐ 0x00400040 0x24080005 addiu $8,$0,0x00000005  39: addiu $t0, $zero, 5 #t0 curChar, set to some arbitrary non zero valu
☐ 0x00400044 0x24090000 addiu $9,$0,0x00000000  40: addiu $t1, $zero, 0 #t1 seenNull
☐ 0x00400048 0x240a0000 addiu $10,$0,0x0000...  42: addiu $t2, $zero, 0 #i = 0
```

zero

as arg

Data Segment

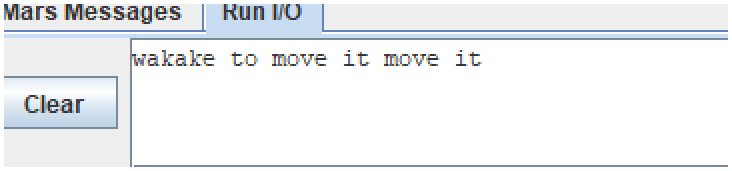
Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value
0x10010000	0x696c2049	0x7420656b	0x6f6d206f	0x69206576	0x6f6d2074	0x69206576	0x6
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x0
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x0
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x0
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x0
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x0
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x0
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x0
0x10010100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x0
0x10010120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x0
0x10010140	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x0
0x10010160	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x0
0x10010180	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x0
0x100101a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x0
0x100101c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x0
0x100101e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x0

0x10010000 (.data) ☑ Hexadecimal Addresses ☑ Hexadecimal Values ☐ ASCII

```
I like to move it move it
```

I chose this as an edge case to test 0 input.

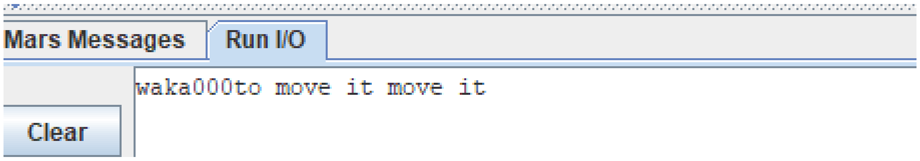
num = 4
Expected: “wakake to move it move it”



```
wakake to move it move it
```

I chose this test case because it was a middle test case.

num = 7
Expected: “waka000to move it move it”



```
waka000to move it move it
```

I chose this because it was a special case for when the num is higher than the source string length.