UNIVERSITY OF TORONTO

CSC258: Computer Organization

Final Assignment

Due: April 10th, 2020 11:59pm

Last Name: Ng

First Name: Isaac

Student Number: 1004817184

Section (circle): **LEC0101** / **LEC0201** / **LEC5101**

Submission via Quercus:

i) **For #1-#8, please submit 1 single PDF file, without changing the format and spacing of the original paper.**

ii) **For #9, please submit 3 ASCII text files, named "9a.s", "9b.s" and "9c.s". Use only ANSI/ASCII encoding. Do not use UTF-8 or Unicode.**

Questions no. 1 to 6 cover processor architecture and instructions, and should be easier to start with after a quick review of slides. Make sure to briefly describe how you come up with your answers for each question.

Questions 7 to 9, the MIPS Assembly language, are covered gradually during the corresponding lectures. We recommend that you start by questions 7 and 8, which are covered by the first third of the lecture set.

The programming assignments, question 9.a to 9.c, get gradually harder. So, our recommendation is to start by 9.a first to get familiar with MARS simulator, data and code segments, labels, if-else, and loops. Then on 9.b, you get exposed to function calls, parameter passing, and returning values. Feel free to start from the examples given in the lectures as well as those on Quercus, and customize them to the question asked. Finally, 9.c covers concepts on recursion and recursive function calls. To more easily answer these questions, you had better first write a recursive function for the corresponding question in a high level language you know, and make sure it works properly. Then try to convert that program to assembly language. Again, feel free to take inspirations, or even start off from, the codes in lecture slides or Quercus.

Processor Operations (12 marks)

1. The stack diagram on the left illustrates the top four bytes of a stack, and the empty spaces above them for a 32-bit processor. What would this stack look like after the decimal integer `513` has been pushed onto it? Draw the result on the diagram on the right (using binary values for the contents).

   For this example, assume little endian byte storage, and draw an arrow to illustrate what `$sp` points to after this operation is complete. **(2 marks)**

2. When Booth's Algorithm is performed on the 4-bit binary inputs A= **−5** and B= **−3**, the values for A and P change at each step of the algorithm. The framework is provided below, with a few values filled in for you. Fill in the rest, according to the steps shown in class. **(6 marks)**

Initial Values: A= 1011 B= 1101 −B= 0011

Step #1:

A = 10110 **Initial P value =** 0000 0000

P value before shift = 0011 0000

Step #2:

A = 11011 **Initial P value =** 0001 1000

P value before shift = 0001 1000

Step #3:

A = 11101 **Initial P value =** 0000 1100

P value before shift = 1101 1100

Step #4:

A = 11110 **Initial P value =** 1110 1110

P value before shift = 0001 1110

Final P value (binary) = 0000 1111 **Final P value (decimal) =** 15

3. Consider the datapath diagrams below. For each of the following steps in an operation, highlight the path that the data needs to take, from start to finish. **(4 marks)**

a) Branch forward to 10 instructions ahead.

1) From the instruction register, I want to read the offset we want
   For branch instructions, this is in the last [15-0] part of instruction
2) i type so sign extend, 4 bytes per instruction so shift left 2
3) PC into A, immediate value (after shifting) into B and add them together
4) ALU goes directly back to PC to update the counter

b) Give address ($s1 + 0x100) to memory for a load or store operation.

1) Instructions are for a load/store opcode whose order is [opcode, read register, destination register, offset]
2) First register is being read ($s1) onto A
3) Write register goes to memory write since we want to load/store to memory
4) The offset is sign extended and sent to B
5) The ALU adds them and sends address to memory where it loads/stores

Processor Instructions (12 marks)

4. For each assembly language instruction below, fill in the blanks with the corresponding 32-bit machine code instruction. **(3 marks)**

a) `j trgt` *(where current PC is 128 and the "trgt" label is at address 100.*

0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0

b) `beq $s0, $s2, top` *(where the "top" label occurs 256 bytes earlier)*

0	0	0	1	0	0	1	0	0	0	0	1	0	0	1	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0

c) `sh $s0, 15($fp)`

1	0	1	0	0	1	1	1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

5. Given the machine code instructions below, write the corresponding assembly language instruction in the space below each machine code instruction. **(3 marks)**

a) `00111011111000101111111111111100`

xori $v0, $ra, -4

b) `00000011101000000000000011001000`

jr $sp

c) `00000000000001110101000111000011`

sra $t2, $a3, 7

6. For each of the processor tasks below, indicate what the values of the following control unit signals will be by filling in the boxes next to each signal with the signal values. **(6 marks)**

- If a control signal doesn't affect the operation, fill in its value with an `X`.
- For `ALUOp`, full marks will only be given for binary values. If you don't know what the values are, just write what kind of operation is taking place instead.

Jump to the address in $s0.

jr $s0
1) Instruction read from memory for jr opcode
2) 25-21 are register $s0, last 6 tell ALU to simply pass A
3) passed into Address where that memory address is read

PCWrite	X	PCWriteCond	X	IorD	1	MemRead	1	MemWrite	0
MemToReg	X	IRWrite	1	PCSource	X	ALUOp	000 (Pass A)		
ALUSrcA	1	ALUSrcB	X	RegWrite	0	RegDst	X		

Move the top 4 bytes of the stack into register $t4.

PCWrite	X	PCWriteCond	X	IorD	1	MemRead	1	MemWrite	0
MemToReg	1	IRWrite	1	PCSource	X	ALUOp	001 (A + B)		
ALUSrcA	1	ALUSrcB	2	RegWrite	1	RegDst	0		

lw $t4, 0($sp)
1) Instruction read from memory for lw opcode
2) [25-21] passed to read reg1, [20-16] passed to write reg, instruction part is sign extended
3) A is from register ($sp), B is from sign extended instruction (0) and added in ALU
4) Passed back into Memory
5) loaded to memory data register and into write data

Assembly Language (66 marks)

7. In the spaces provided below, write the assembly language instruction(s) that perform the following tasks. ***Full marks will only be given for one-instruction answers***! **(3 marks total)**

a) Multiply the value stored in `$t4` by 4 and stored it back in `$t4`. **(1 marks)**

sll $t4, $t4, 2

b) Store the remainder of dividing `$t3` by 32 in `$t9`. **(1 marks)**

andi $t9, $t3, 31

c) Compute 1's complement (inverting all the bits) of the lower half of `$t6` without changing its upper half, and store the result in `$t7`. **(1 marks)**

xori $t7, $t6, 65535

8. We need to extend the MIPS assembler to support the following new pseudo instructions. For each new pseudo-instruction, write the real MIPS instructions that will perform that operation. Only implementations that use at most 2 operations will get full marks. **(3 marks total)**

a) `multi $t,i` Multiply `$t` by the constant `i` and store result in `lo` and `hi`. **(1 marks)**

```
addi $at, $zero, i
mult $t, $at
```

b) `popw $t` Pop the top word of the stack and store it in `$t`. **(1 marks)**

```
lw $t, 0($sp)
addi $sp, $sp, 4
```

c) `bod $t, dest` Branch to label `dest` if the value in `$t` is odd. **(1 marks)**

```
andi $t, $t, 1
bne $0, $t, dest
```

9. Write below programs in MIPS assembly language. **(60 marks)**

a. Complete below program to find the maximum value in the array `list` which has `len` elements, and return it in one of the return-value registers. **(10 marks)**

```
            .data
len:        .word     5
list:       .word     -4, 6, 7, -2, 1
            .text
main:
```

<u>Please submit a separate text file named "9a.s".</u>

b. Write a function `get_n` that takes `n` as an argument, and returns the `n`-th word of global array `list` which has `len` words. Also write code at label `main` to call `get_n` function. If n is less than 1 or bigger than `len`, your function should return -1. **(20 marks)**

```
            .data
len:        .word     5
list:       .word     -4, 6, 7, -2, 1

            .text
get_n:

main:
```

<u>Please submit a separate text file named "9b.s".</u>

c. Write a recursive function `get_min` to return the smallest element in an array of words. Call the function in your main program and pass it an array of 10 elements. **(30 marks)**

<u>Please submit a separate text file named "9c.s".</u>

Reference Information

ALU arithmetic input table:

Select		Input	Operation	
S_1	S_0	Y	C_{in}=0	C_{in}=1
0	0	All 0s	G=A	G=A+1
0	1	B	G=A+B	G=A+B+1
1	0	B	G=A-B-1	G=A-B
1	1	All 1s	G=A-1	G=A

Register assignments:

Register values : Processor role

- Register 0 ($zero): reserved value.
- Register 1 ($at): reserved for the assembler.
- Registers 2-3 ($v0, $v1): return values
- Registers 4-7 ($a0-$a3): function arguments
- Registers 8-15, 24-25 ($t0-$t9): temporaries
- Registers 16-23 ($s0-$s7): saved temporaries
- Registers 28-31 ($gp, $sp, $fp, $ra)

Instruction table:

Instruction	Type	Op/Func	Syntax
add	R	100000	$d, $s, $t
addu	R	100001	$d, $s, $t
addi	I	001000	$t, $s, i
addiu	I	001001	$t, $s, i
div	R	011010	$s, $t
divu	R	011011	$s, $t
mult	R	011000	$s, $t
multu	R	011001	$s, $t
sub	R	100010	$d, $s, $t
subu	R	100011	$d, $s, $t
and	R	100100	$d, $s, $t
andi	I	001100	$t, $s, i
nor	R	100111	$d, $s, $t
or	R	100101	$d, $s, $t
ori	I	001101	$t, $s, i
xor	R	100110	$d, $s, $t
xori	I	001110	$t, $s, i
sll	R	000000	$d, $t, a
sllv	R	000100	$d, $t, $s
sra	R	000011	$d, $t, a
srav	R	000111	$d, $t, $s
srl	R	000010	$d, $t, a
srlv	R	000110	$d, $t, $s
beq	I	000100	$s, $t, label
bgtz	I	000111	$s, label
blez	I	000110	$s, label
bne	I	000101	$s, $t, label
j	J	000010	label
jal	J	000011	label
jalr	R	001001	$s
jr	R	001000	$s
lb	I	100000	$t, i ($s)
lbu	I	100100	$t, i ($s)
lh	I	100001	$t, i ($s)
lhu	I	100101	$t, i ($s)
lw	I	100011	$t, i ($s)
sb	I	101000	$t, i ($s)
sh	I	101001	$t, i ($s)
sw	I	101011	$t, i ($s)
trap	I	001100	i
mflo	R	010010	$d