UNIVERSITY OF TORONTO

CSC258: Computer Organization

Final Assignment

Due: April 10th, 2020 11:59pm

Last Name: Singh

First Name: Shantanu

Student Number: 1005218514

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.

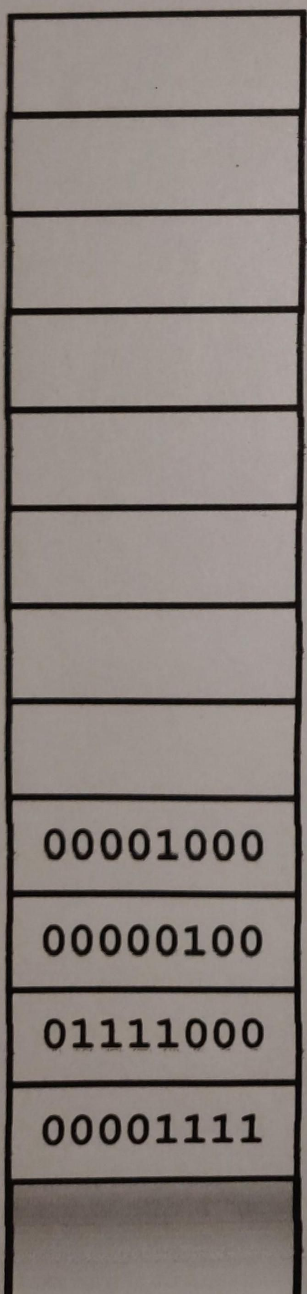
The programming assignments, question 9.a to 9.c, get gradually harder. So, our recommendation is to start by 9.a first to get familiar with MARS simulator, data and code segments, labels, if-else, and loops. Then on 9.b, you get exposed to function calls, parameter passing, and returning values. Feel 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.

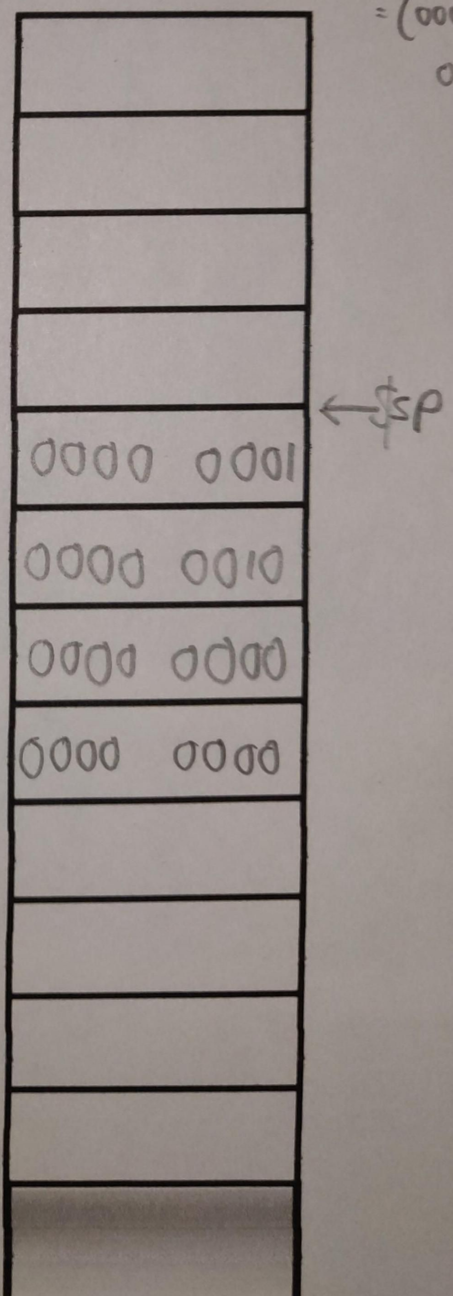
Processor Operations (12 marks)

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)**

$513 = 2^9 + 1$
$= (0000\ 0010$
$0000\ 0001)$





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' = 10 → −

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

P value before shift = 0011 0000

Step #2:

A' = 11 → shift

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

P>>1 = 0000 1100

A>>>1 = 11101

P value before shift = 0001 1000

Step #3:

A' = 01 − +

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

P>>1 = 1110 1110

A>>>1 = 11110

P value before shift = 1101 1100

Step #4:

A' = 10 → −

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

carry

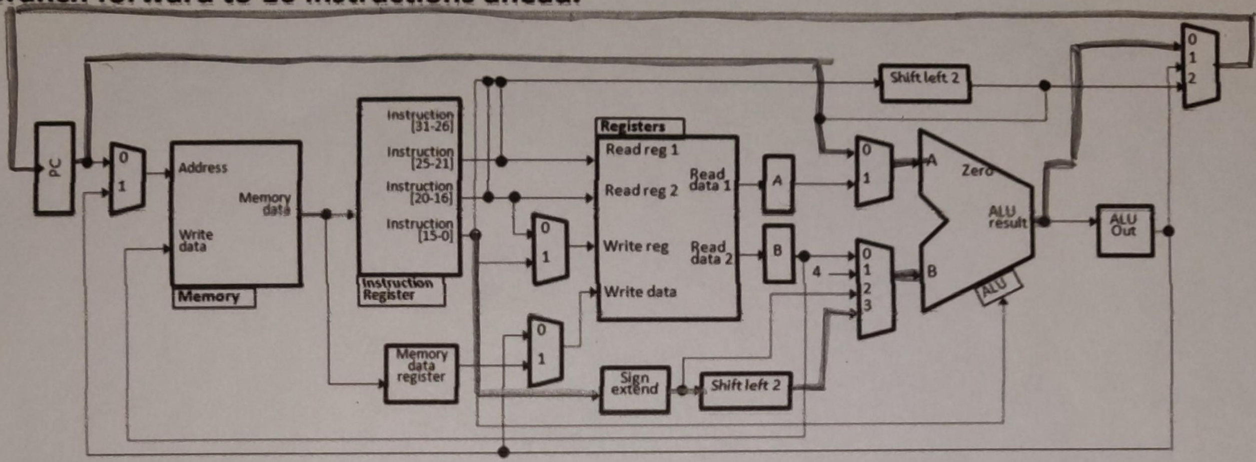
P value before shift = 1 0000 1110 P>>>1 = 0000 1111

Ans

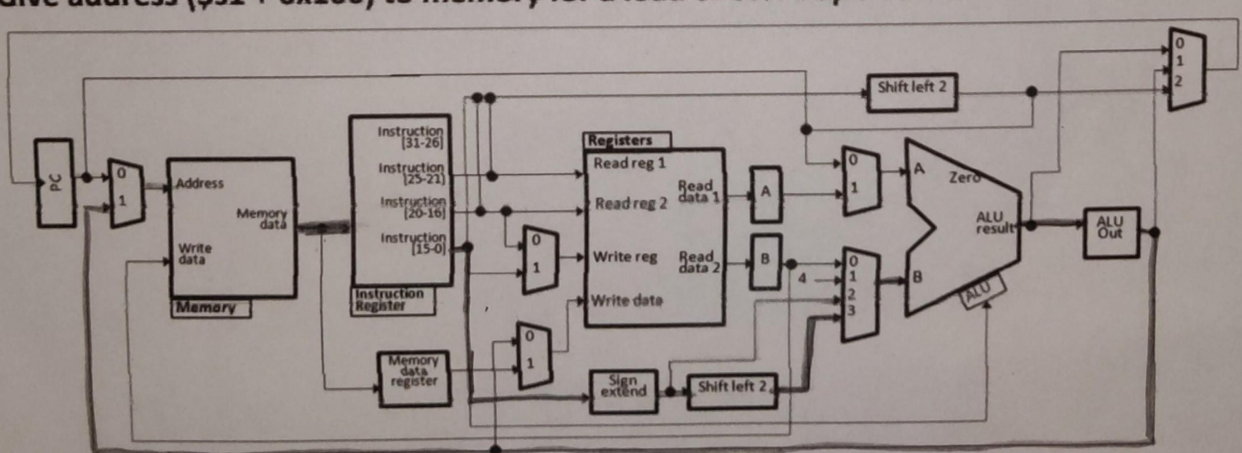
Final P value (binary) = 0000 1111 **Final P value (decimal) =** 15

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.



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



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)**

100/4 = 25
25 : 11001

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

-256 (offset) 28

I-Type **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	0	0	0	0	0	0	0	0

I-Type offset base

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

1	0	1	0	0	1	1	1	1	1	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)**

I-Type

a) `00111011111000101111111111111100`

XORI $v0, $ra -4

Not I,J, R $sp 0 JR

b) `00000011101000000000000000001000`

JR $sp

SRA

c) `00000000000001110101000111000011`

Not I,J,R 0 a3 t2 7

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.

$s0

Jump to the address in $s0. ← j

PCWrite	0	PCWriteCond	0	IorD	X	MemRead	1	MemWrite	0
MemToReg	1	IRWrite	0	PCSource	X	ALUOp	001		
ALUSrcA	0	ALUSrcB	10	RegWrite	1	RegDst	0		

Move the top 4 bytes of the stack into register $t4. ← lw $t4, 0(sp)

PCWrite	0	PCWriteCond	0	IorD	1	MemRead	1	MemWrite	0
MemToReg	1	IRWrite	0	PCSource	X	ALUOp	001		
ALUSrcA	1	ALUSrcB	10	RegWrite	1	RegDst	0		

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

$(2^{16}-1)$

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 $s0, $0, i
mult $s0, $t

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

lw $t, 0($sp)
addiu $sp, $sp, 4
addi works too

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

andi $t0, $t, 1
bnez $t0, dest