UNIVERSITY OF TORONTO

Faculty of Arts and Science

April 2018 Examinations

CSC258H1S: Computer Organization

Duration: 3 hours

Permitted Aids: one ruler, one highlighter

Last Name:	
First Name:	
Student Number:	
Instructors:	Steve Engels (L0101, L0201)
	Rabia Bakhteri (L5101)

Instructions:

- **Write your name on every page of this exam.**
- **Do not open this exam until you hear the signal to start.**
- Have your student ID on your desk.
- No aids permitted other than writing tools. Keep all bags and notes far from your desk before the exam begins.
- There are 6 questions on 19 pages. When you hear the signal to start, make sure that your exam is complete before you begin.
- Read over the entire exam before starting.
- If you use any space for rough work or have to user the overflow page, clearly indicate the section(s) that you want marked.
- <u>Important:</u> CSC258 has an minimal exam condition for passing the course. You must get at least 40% on this exam to pass the rest of the course.

Mark Breakdown	
Part A:	/ 33
Part B:	/ 20
Part C:	/ 28
Part D:	/ 39
Part E:	/ 20
Part F:	/ 40
Bonus:	/ 1
Total:	**/ 180**

Part A: Short Answer (33 marks)

Answer the following questions in the space provided. When providing a written answer, write **as clearly and legibly as possible**. Marks will not be awarded to unreadable answers.

1. While writing to memory, not all of the data bits were written into the destination address before the write signal was turned off. Which memory delay(s) should have been longer? Circle all that apply. **(1 mark)**

   a) t_{SA} = Address Setup Time .

   b) t_{AA} = Address Access time.

   c) t_{OHA} = Output Hold time.

   d) t_{SD} = Data Setup to Write End.

   e) t_{HD} = Data Hold from Write End.

2. While writing to memory, the data bits were written into the destination address correctly, but then incorrect data bits were written at the end before the write signal was turned off. Which memory delay(s) should have been longer? Circle all that apply. **(1 mark)**

   a) t_{SA} = Address Setup Time .

   b) t_{AA} = Address Access time.

   c) t_{OHA} = Output Hold time.

   d) t_{SD} = Data Setup to Write End.

   e) t_{HD} = Data Hold from Write End.

3. While writing to memory, the data bits were written into two different addresses by mistake. Which memory delay(s) should have been longer? Circle all that apply. **(2 marks)**

   a) t_{SA} = Address Setup Time .

   b) t_{AA} = Address Access time.

   c) t_{OHA} = Output Hold time.

   d) t_{SD} = Data Setup to Write End.

   e) t_{HD} = Data Hold from Write End.

4. True or False? When a K-Map is used to find the boolean expression of a function in "Sum of Minterms" notation, at least one "don't care" output needs to be in a group. **(1 mark)**

   True **False**

5. How many total inputs does a 6-to-1 multiplexer have? **(1 mark)**

6. How many total inputs does a 1-to-6 demultiplexer have? **(1 mark)**

7. On the processor datapath diagram, how many bits wide are the following inputs to the register file? **(3 marks)**

Write reg: ______ Write data: ______ RegWrite: ______

8. How many bits are stored in each of the following registers, given the architecture we discussed in class? **(2 marks)**

Pregram Counter: ______ Instruction Register: ______

Memory Data Register: ______ ALUOut: ______

9. True or False? A half-adder has no "carry out" bit. **(1 mark)**

True False

10. True or False? For a 3-input device, F = m1 + m2 + m3 + m5 and G = M0 · M4 · M6 · M7 express the same function. **(1 mark)**

True False

11. True or False? The ALU in the MIPS datapath is capable of performing increment and decrement operations. **(1 mark)**

True False

12. True or False? Writing to memory is faster than reading from the registers. **(1 mark)**

True **False**

13. True or False? Writing to the registers is faster than reading from memory. **(1 mark)**

True **False**

14. What are the O() costs for each of the following multiplication implementations? **(3 marks)**

Multiplier Circuit:	Time: ________	Space: ________
Accumulator Circuit:	Time: ________	Space: ________
Booth's Algorithm:	Time: ________	Space: ________

15. If the binary number `01100101` is the input to a barrel shifter with a shift value of `4`, what is the shifter's output? **(1 mark)**

16. Which assembly shift operator can turn a negative number into a positive number? **(1 mark)**

17. Which of the following is true about jump instructions? **(1 mark)**

a) The new PC value and the old PC value will have the same first four bits

b) The new PC value and the old PC value will have the same first six bits

c) The new PC value and the old PC value will have the same first fourteen bits

d) The new PC value and the old PC value will have the same first sixteen bits

d) None of the above.

18. Rank the following interrupts in order of priority, from 1 (highest) to 4 (lowest). **(3 marks)**

Address error exception	_____
Arithmetic overflow	_____
Bus error	_____
External interrupt	_____

19. In the diagram below, show how data is laid out in memory if we store `0xDEC0DEFA` at address `0x400000` in little endian **(2 marks)**

0x400000	0x400001	0x400002	0x400003

20. True or False? All I-type ALU instructions have an R-type counterpart. **(1 mark)**

True **False**

21. True or False? All J-type instructions are jumps. **(1 mark)**

True **False**

22. True or False? All R-type instructions involve the ALU. **(1 mark)**

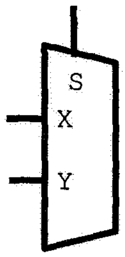
True **False**

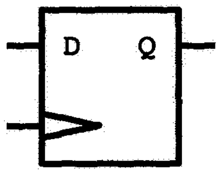
23. Consider the assembly code on the right. What value ends up in `$d`? **(2 marks)**

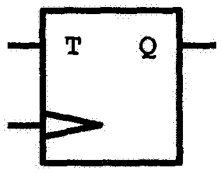
```
sra     $at, $s, 31
nor     $d, $s, $at
sub     $d, $d, $at
```

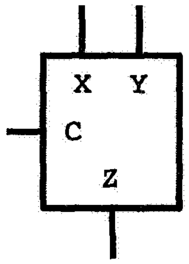
Part B: Design and Analysis (20 marks)

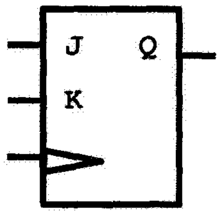
1. Draw lines to connect the device diagrams on the left with the Verilog modules on the right. **(12 marks)**

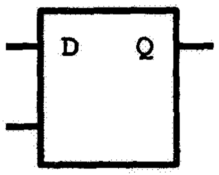












```
module ay(X,Y,Z);
input X,Y;
output Z;
always @ (Y)
   Z = X;
endmodule
```

```
module bee(X,Y,Z);
input X,Y;
output Z;
always @ (posedge Y)
   Z = X;
endmodule
```

```
module cee(W,X,Y,Z);
input W,X,Y;
output Z;
always @ (*)
   Z = ~W&X || W&Y;
endmodule
```

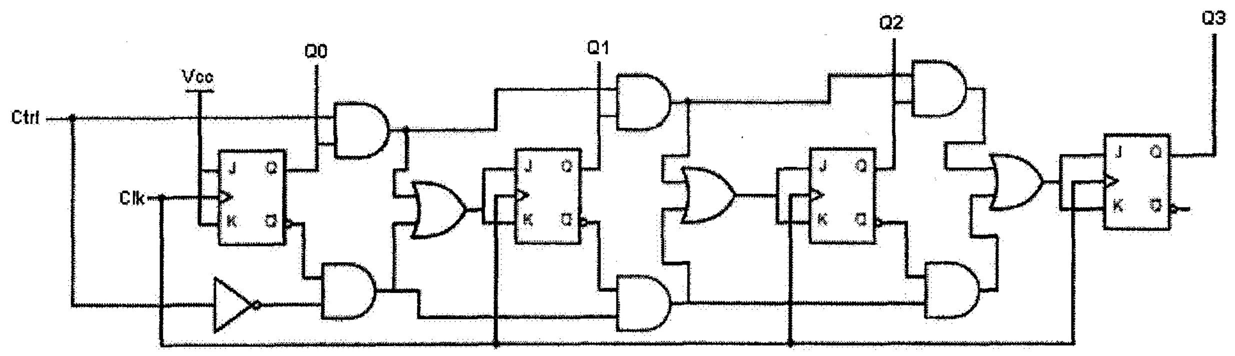
```
module dee(W,X,Y,Z);
input W,X;
output Y,Z;
always @ (*)
   Y = W&X;
   Z = W^X;
endmodule
```

```
module ee(X,Y,Z);
input X,Y;
output Z;
always @ (posedge Y)
   if (X)
      Z = ~Z;
endmodule
```

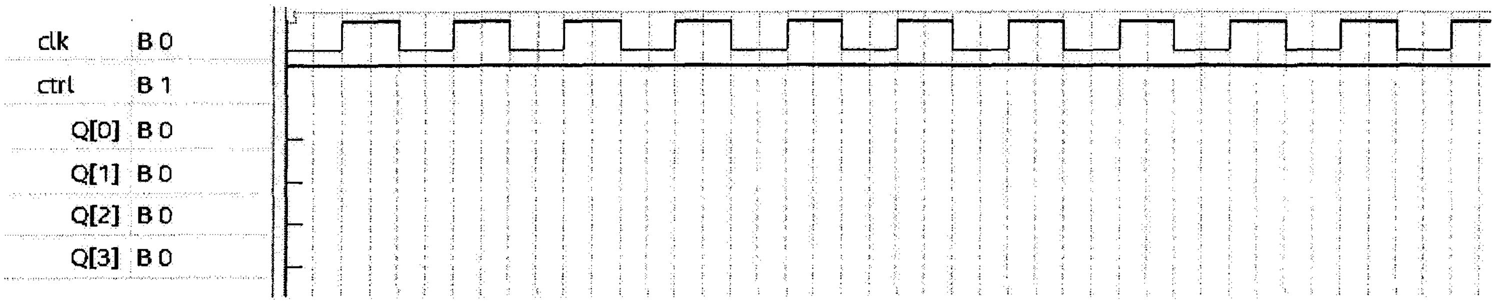
```
module eff(W,X,Y,Z);
input W,X,Y;
output Z;
always @ (posedge Y)
   if (W&~X)
      Z = 0;
   else if (~W&X)
      Z = 1;
   else if (W&X)
      Z = ~Z;
endmodule
```

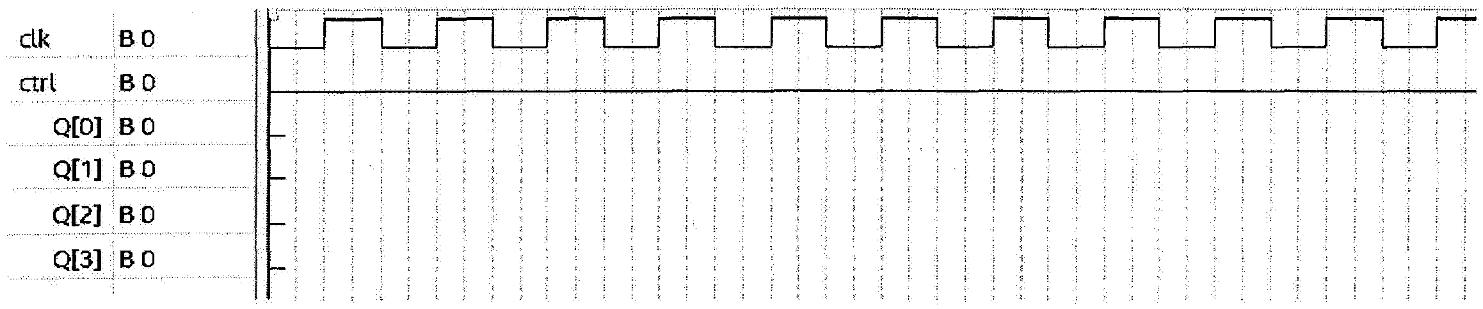
Student Number: ____________________

2. -Consider the sequential circuit shown in the diagram below.



Complete the following simulation waveforms provided below, one for when `ctrl` is high, and one for when `ctrl` is low. **(8 marks total, 4 marks each)**





Part C: Processor Operations (28 marks)

1. When Booth's Algorithm is performed on the 4-bit binary inputs A= **−3** and B= 7, 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. **(8 marks)**

Initial Values: A= 11010 B= -B=

Step #1:

A = 11010 **Initial P value =** 0000 0000

P value before shift = 1001 0000

Step #2:

A = **Initial P value =**

P value before shift =

Step #3:

A = **Initial P value =**

P value before shift =

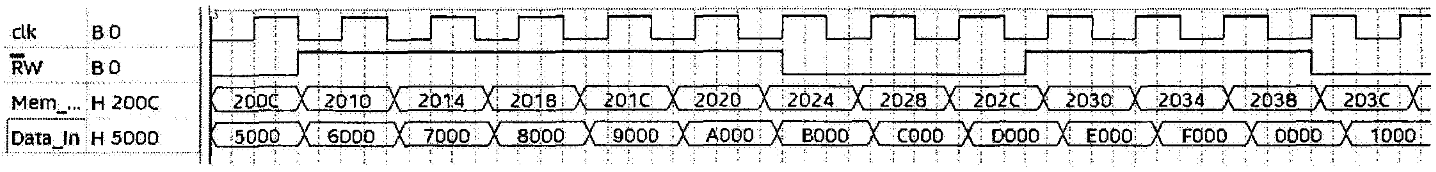
Step #4:

A = **Initial P value =**

P value before shift =

Final P value (binary) = **Final P value (decimal) =**

2. The following waveform shows the write cycle for a RAM memory. The control signal $\overline{\text{Read}}$/Write is represented by $\overline{\text{R}}$W in the waveform.

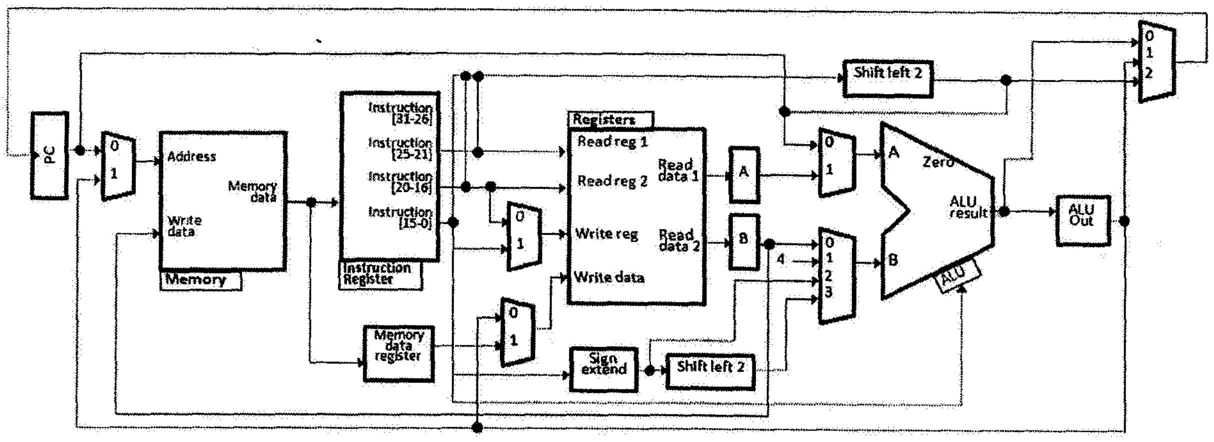

clk B 0
RW B 0
Mem_... H 200C
Data_In H 5000
200C 2010 2014 2018 201C 2020 2024 2028 202C 2030 2034 2038 203C
5000 6000 7000 8000 9000 A000 B000 C000 D000 E000 F000 0000 1000

Based on the waveform above, fill in the following diagram with memory addresses and their correct contents. **(8 marks)**

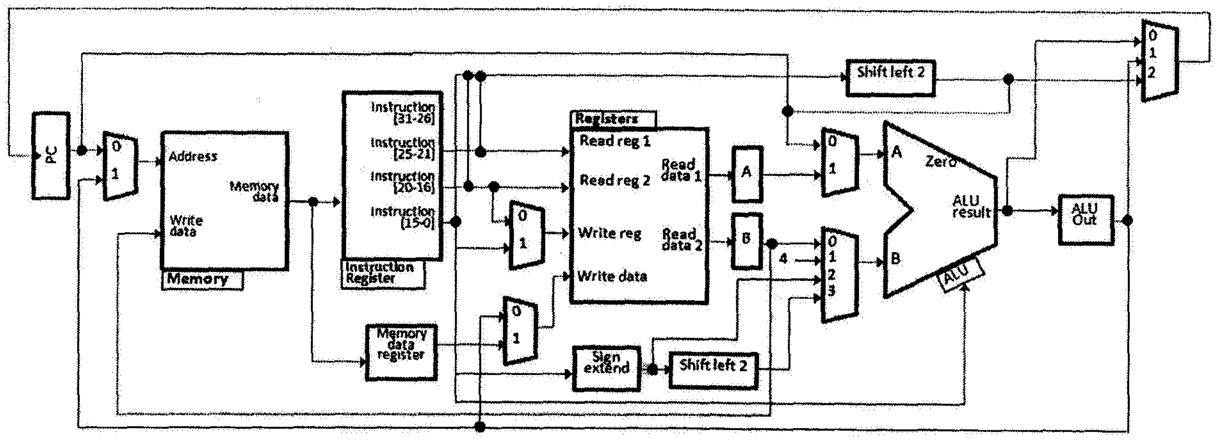
Memory Address	**Memory Contents**
0x200C	
0x2010	
0x2014	
0x2018	
0x201C	
0x2020	
0x2024	
0x2028	
0x202C	
0x2030	
0x2034	
0x2038	
0x203C	

3. Consider the datapath diagrams below. For each of the following steps in a function call operation, highlight the path that the data needs to take, from start to finish. **(12 marks)**

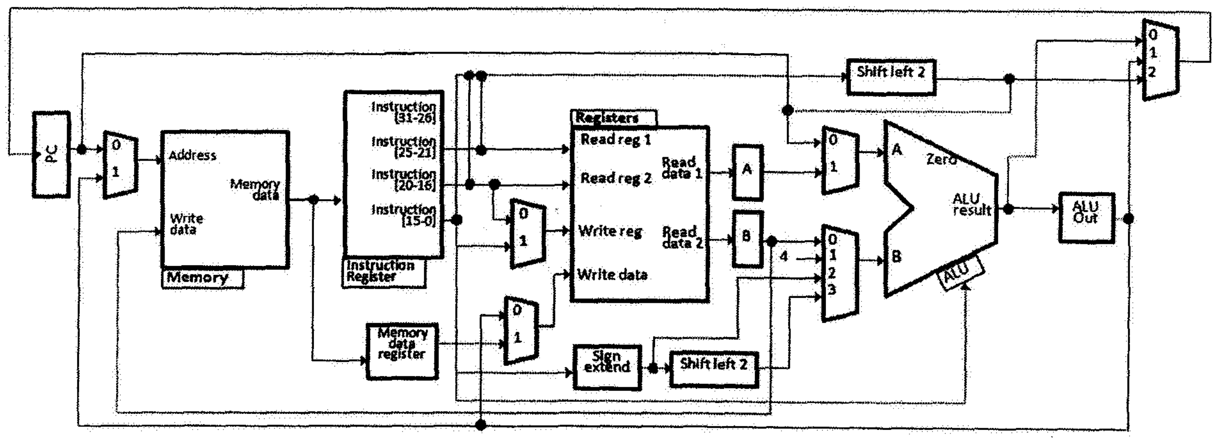
a) `and $t3, $t1, $t2`



b) `sw $t1, 2($s1)`



c) `j main`



Part D: Processor Instructions (39 marks)

1. For each assembly language instruction below, fill in the blanks with the corresponding 32-bit machine code instruction. For bits that could be either a **0** or a **1**, fill in the blank with an **X** value instead. **(12 marks)**

a) `subu $v1, $a1, $s1`

b) `jal start` *(where the "start" label occurs at address 0x0000FF00)*

c) `sra $t7, $t8, -4`

2. Given the machine code instructions below, write the corresponding assembly language instruction in the space below each machine code instruction. **(12 marks)**

a) 00100001000111011111111111000000

b) 10010100010001000000000000001100

c) 00000011111001110111111111001001

3. 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. **(15 marks)**

- Assume that the instruction for these operations is already in the instruction register.
- 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.

Set `$ra` to the address 10 instructions after the current instruction location.

PCWrite		PCWriteCond		IorD		MemRead		MemWrite	
MemToReg		IRWrite		PCSource		ALUOp			
ALUSrcA		ALUSrcB		RegWrite		RegDst			

Jump to the address stored in the `ALUOut` register if `$t0` and `$t1` are equal.

PCWrite		PCWriteCond		IorD		MemRead		MemWrite	
MemToReg		IRWrite		PCSource		ALUOp			
ALUSrcA		ALUSrcB		RegWrite		RegDst			

Set `$s0` to the exclusive or of `$s1` and `$s2`.

PCWrite		PCWriteCond		IorD		MemRead		MemWrite	
MemToReg		IRWrite		PCSource		ALUOp			
ALUSrcA		ALUSrcB		RegWrite		RegDst			

Part E: Verilog (20 marks)

Consider the piece of Verilog code on the right.

1. In one sentence or less, describe the operation performed by this module. **(6 marks)**

2. Based on your answer above, what are the functions of the following input and output bits? **(5 marks)**

s: ______________________

a: ______________________

b: ______________________

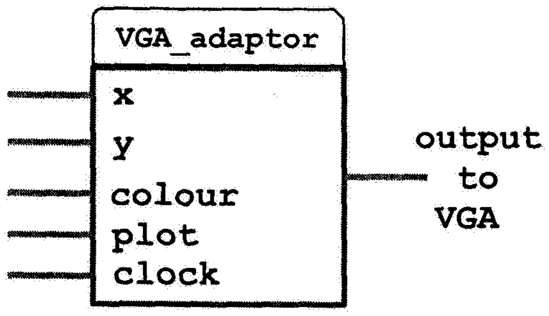
c: ______________________

d: ______________________

```
module foo(q, s, x, y, a, b, c, d);

  input [31:0]  x, y;
  input  s;
  output reg [31:0] q;
  output  a, b, c, d;

  always @(*)
    begin
      if (s)
        begin
          {a,q} <= x - y;
          b <= x[31]&~y[31]&~q[31] |
                 ~x[31]&y[31]&q[31];
        end
      else
        begin
          {a,q} <= x + y;
          b <= x[31]&y[31]&~q[31] |
                 ~x[31]&~y[31]&q[31];
        end
      c <= q[31];
      d <= ~| q;
    end

endmodule
```

3. On the right is a simplified version of the VGA adaptor from Lab 7. Assume that you will use this module as a component of your solution, similar to the way that lab was implemented.

In the space below, complete the Verilog that would draw a **single white line** (one pixel high and 160 pixels wide) along the top of the screen. The other lines on the screen should remain unchanged.



Assume that this VGA adaptor exists already and is provided to you as a module called `vga_adaptor`. This module has the following inputs, similar to Lab 7:

- `x` and `y` (both 8 bits wide),
- `colour` (3 bits wide),
- `plot` and `clock` (both 1 bit wide).

It is strongly suggested that you use a `case` statement to colour the VGA buffer in one stage, and then draw it on the screen in a second stage. Again, assume that you don't have to colour the rest of the screen, just the top line. **(9 marks total)**

```
module draw_white_pixel (clock);
   input   clock;
   reg [7:0] x=0;
   reg      plot, state = 0;

















endmodule
```

Part F: Assembly Language (40 marks)

1. 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*.** **(12 marks total)**

a) Divide the contents of register `$t0` by 8. **(3 marks)**

b) Set `$t1` to the remainder that would be left after dividing `$t0` by 8. **(3 marks)**

c) Invert the last 8 bits of `$v0`, leaving the rest of the bits untouched. **(3 marks)**

d) Perform 2's complement on `$a0` (storing the result back in `$a0`) **(3 marks)**

Student Number: ____________________ (continued)

2. We're extending the MIPS assembler to support the following new pseudo instructions. For each pseudo-instruction below, write real MIPS instructions that will perform that operation. For full marks, use the minimal number of operations in your solution. **(16 marks total)**

a) `exp $d, $s` Set `$d` to 2^x, where `x` is stored in `$s`. **(4 marks)**

b) `mean $d, $s, $t` Set `$d` to the mean of `$s` and `$t` **(4 marks)**

c) `xnor $d,$s,$t` Store the result of an XNOR of `$s` and `$t` in `$t`. **(4 marks)**

d) `ji i` Jump to address `i` (sign extended to 32 bits). **(4 marks)**

3. In the spaces provided, write the operation performed by each assembly program. **(12 marks total)**

```
        .data
len:    .word   5
list:   .word   -4, 6, 7, -2, 1
        .text
main:   la $s1, len
        lw $t1, 0($s1)
        la $s0, list
alpha:  lw $t0, 0($s0)
        addi $t1, $t1, -1
        addi $s0, $s0, 4
        lw $t2, 0($s0)
        sw $t0, 0($s0)
        sw $t2, -4($s0)
        blez $t1, beta
        j alpha
beta:   jr $ra
```

```
        .data
len:    .word   5
list:   .word   -4, 6, 7, -2, 1
        .text
main:   la $s1, len
        lw $t1, 0($s1)
        la $s0, list
alpha:  lw $t0, 0($s0)
        addi $t1, $t1, -1
        bgtz $t0, alpha
        sw $t1, 0($s0)
        addi $s0, $s0, 4
        blez $t1, beta
        j alpha
beta:   jr $ra
```

```
        .data
len:    .word   5
list:   .word   -4, 6, 7, -2, 1
        .text
main:   la $s1, len
        lw $t1, 0($s1)
        la $s0, list
alpha:  lw $t0, 0($s0)
        sub $t0, $zero, $t0
        sw $t0, 0($s0)
        addi $t1, $t1, -1
        blez $t1, beta
        addi $s0, $s0, 4
        j alpha
beta:   jr $ra
```

```
        .data
len:    .word   5
list:   .word   -4, 6, 7, -2, 1
        .text
main:   la $s1, len
        lw $t1, 0($s1)
        la $s0, list
alpha:  lw $t0, 0($s0)
        sll $t0, $t0, 2
        sw $t0, 0($s0)
        addi $t1, $t1, -1
        blez $t1, beta
        addi $s0, $s0, 4
        j alpha
beta:   jr $ra
```

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)

Bonus Question: (1 mark)

In the space below, draw (and name) your favorite CSC258 TA.

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
`sltu`	R	101001	$d, $s, $t
`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
`mfhi`	R	010000	$d
`mflo`	R	010010	$d

Student Number: ___________________

(continued)

This page is left blank intentionally for answer overflows.

Total Marks = 180

Total Pages = 19

End of exam