Name Sean Gordon

ISU ID # 495 762 295

Lab Section (circle one—your exam will be given a 0 if you do this incorrectly):

A (W 10-Noon), B (R 10-Noon), [✓ C] (F 10-Noon), D (W 4-6)

CprE 381
Computer Organization and Assembly Level Programming

Exam #2
4/1/2019 9:00-9:50AM

Directions: There are 4 questions in this exam. Each question is worth points indicated. You should roughly spend 1 minute for every two points—plan accordingly. If a problem appears to be hard, move on and come back. Please read the questions carefully. Show your work, including any assumptions you need to use to solve the problems.

Calculators should NOT be used.

Problem	Score
1	15 / 15 points
2	19 / 25 points
3	35 / 35 points
4	23 / 25 points
Total	92 / 100 points

15

1. **FUNctional Unit Design (15 points).**
Currently the rotate left (**`rol`**) instruction (see the below excerpt from an ISA manual) is a pseudoinstruction. You will consider implementing the hardware needed to support left rotates for a ***4-bit data path*** (this means that data elements such as general-purpose registers and ALU components are only four bits wide).

Rotate left

```
rol  rdest, rsrc1, rsrc2        pseudoinstruction
```

Rotate value in register rsrc1 left by the distance indicated by rsrc2 and put the result in register rdest.

Example:

```
    rol     $t1, $t0, 1
```

is assembled as

```
    sll     $at,$t0,1
    srl     $t1,$t0,3
    or      $t1,$t1,$at
```

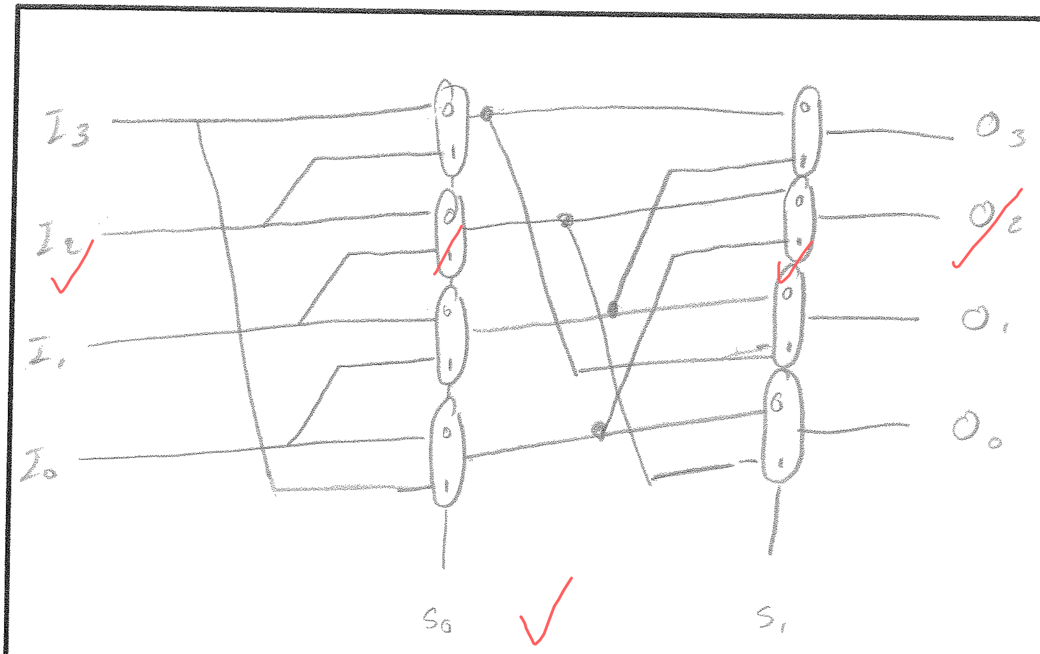
Implement a functional unit that does both rotate left.

(a) How many levels/stages will be needed? Why? (5 points)

As all bits can only be shifted left by a combination of 1+2 bits before it becomes useless, there will be 2 stages to cover that criteria.

(b) Draw a schematic of the rotate unit below. First, label the inputs and outputs including their widths. Second, draw the required levels of MUXs. Third, hook up the components—make sure the signals are clearly labeled. (10 points)

This is a barrel roll rotate, right?



-19

2. **Processor Design (25 Points).**
Modify the single-cycle datapath on the following page to include the **minc** instruction. A **minc** instruction reads the word pointed to by the src register, increments it by one, and stores it back into the src register. Specifically, it performs the following rtl:

```
R[rs] ← M[R[rs]] + 1
```

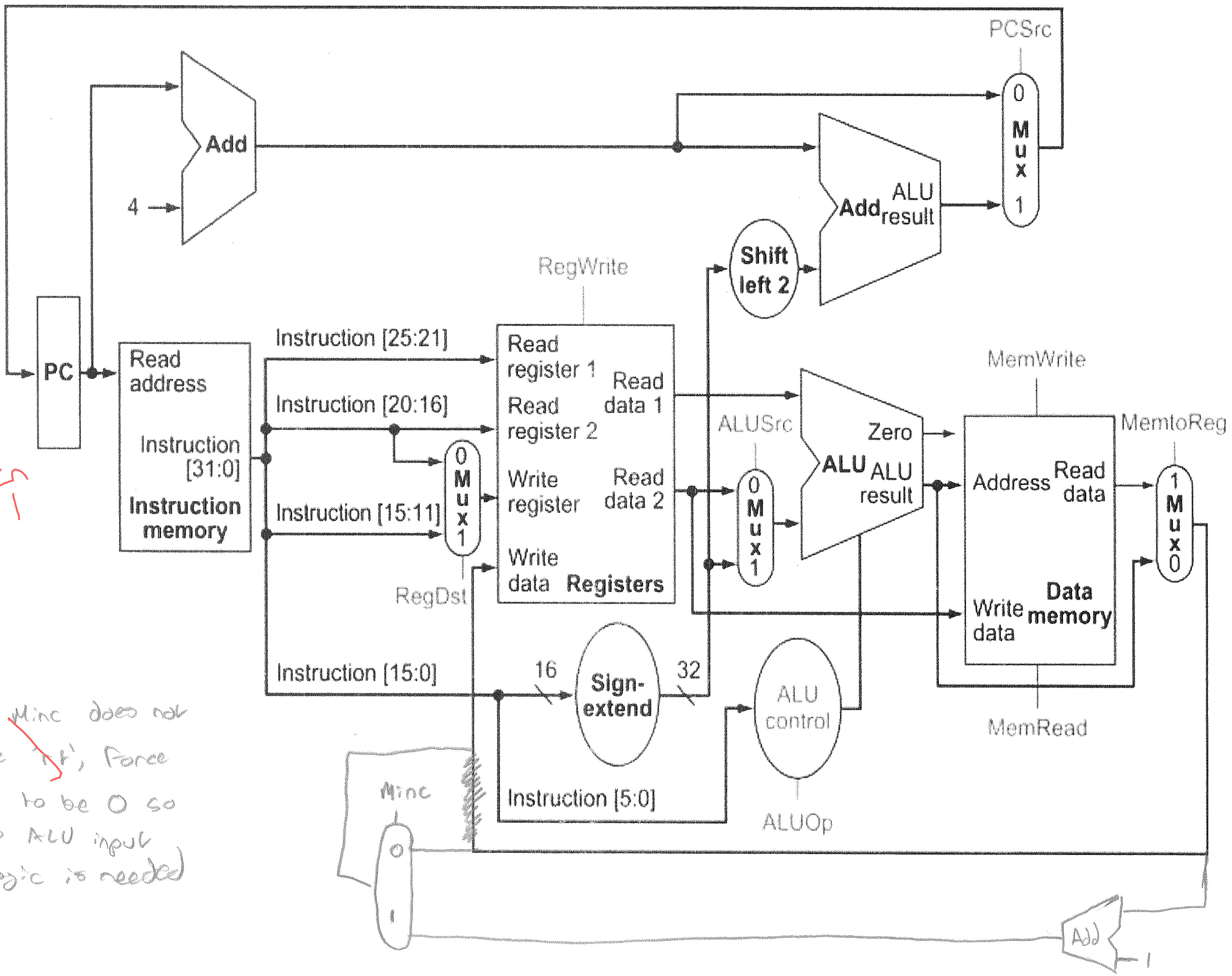
Assume that **minc** uses the I-format. Your modifications can include: severing wires, assigning wires values, adding mux inputs, adding wires, and duplicating any component in the below diagram (and adding corresponding control signals). Assign values to each control signal (both old and new).

Because I-Format has access to another 'rt' reg, but doesn't use it, Force rt to be 0, and nothing needs to be added to ALU input logic

Control Signal	Value
RegDst	0 ← would write to $0 ⇒ Nop
RegWrite	1 ✓
ALUSrc	0 ✓
ALUOp	Add ✓
PCSrc	0 ✓
MemWrite	0 ✓
MemRead	1 ✓
MemtoReg	1 ✓
Minc	1 ✓

adding $rs, $0

-1



write reg?

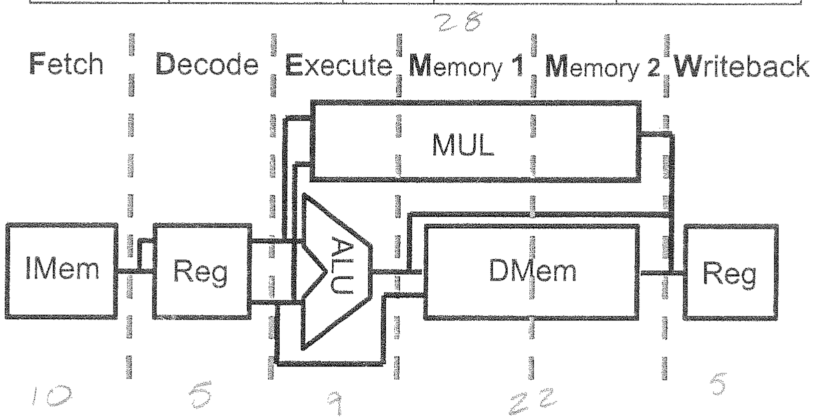
−5

As Minc does not use 'rt', force rt to be 0 so no ALU input logic is needed

+35

3. **Pipelining (35 points).**
We have developed a new 381 pipeline that includes a larger (and thus longer-latency) data memory component and a tightly-integrated multiplier functional unit. The latencies of each of the functional units are tabulated below. Because of the long latencies of the new functional units, the pipeline has been redesigned as shown below (note this is a high-level representation and doesn't include all of the control signals and MUXs).

Imem	Reg Read	ALU	MUL	DMem
10ns	5ns	9ns	28ns	22ns



(a) What is the cycle time of this processor? You may assume that pipeline registers and MUXs have negligible latencies. Why? (10 points)

Cycle time is set by component with the most latency, in this case being DMem. Breaking DMem up into two even 11ns chunks is the greatest latency found in the circuit, and therefore it sets the cycle time. ✓

Cycle time = 11 ns ✓

(b) What is the maximum CPI of this new processor? Why? (5 points)

Each instruction takes 6 cycles, so as every instruction takes the same # cycles CPI = 1

ok ∴ given "max"

(c) Complete the table of read after write *data dependencies* in the following MIPS assembly code. You only need to include data dependencies through registers. (10 points)

```
1: mul     $t0, $s0, $s1    # Assume this to be a hardware
                            implementation of the MIPS mul
                            pseudoinstruction

2: sw      $t1, 0($t2)
3: lw      $t1, 4($t2)
4: addiu   $t2, $t0, 4
5: xor     $t0, $t1, $t2
6: sllv    $t3, $t0, $t1
```

Pipeline length = 6

	Reading Instruction Mnemonic	Register	Writing Instruction Mnemonic
1	xor	$t2	addiu
2	addiu	$t0 ✓	mul
3	xor	$t1 ✓	lw
4	sllv	$t0 ✓	xor
5	sllv	$t1 ✓	lw

Mul
Sw
lw
addiu
xor
sllv

(d) For the sequence of instructions in (c), identify and ***list*** any data hazards for the pipeline described in (a). Demonstrate these hazards with a pipeline diagram (you must show where the hazard is). Assume no forwarding or stalling is implemented yet, but the register file will read the new value from a register that is written in the same cycle. (10 points)

Mnemonic	**Cyc 1**	**Cyc 2**	**Cyc 3**	**Cyc 4**	**Cyc 5**	**Cyc 6**	**Cyc 7**	**Cyc 8**	**Cyc 9**	**Cyc 10**	**Cyc 11**	**Cyc 12**	**Cyc 13**
`mul`	F	D	E	M1	M2	W							
sw		F	D	E	M1	M2	W						
lw			F	D	E	M1	M2	W					
addiu ✓				F	D	E	✓ M1	M2	W				
xor					F	D	E	M1	M2	W			
sllv						F	D	E	M1	M2	W		

+23

4. **Performance (25 points).**
You are designing an embedded system that takes pictures of your cat and automatically generates a cat meme. You are considering two different processors. The first processor is a traditional MIPS processor, while the second one also has SIMD (single instruction, multiple data) support. The processors have the following frequencies and CPIs:

Machine	CPU Speed	ALU	SIMD ALU	Load/Store	Branch/Jump
Original	2 GHz	2	-	3	1
w/ SIMD	???	1	1	2	2

Consider two familiar software implementations for summing all of the elements of a byte array (this operation, known as a "sum reduction," is common in matrix-vector applications such as neural network inference). Assume that register $a1contains the value N.

Implementation 1 (no SIMD instructions):

```
     add    $t0, $zero, $zero
     add    $t2, $zero, $zero
     j      cond
loop:
     addu   $t1, $a0, $t0
     lb     $t1, 0($t1)
     addu   $t2, $t2, $t1
     addiu  $t0, $t0, 1
cond:
     slt    $t1, $t0, $a1
     bne    $t1, $zero, loop
exit:
```

Implementation 2 (SIMD):

```
     add    $t0, $zero, $zero
     add    $t2, $zero, $zero
     j      cond
loop:
     addu      $t1, $a0, $t0
     lw        $t1, 0($t1)
     raddu.qb  $t1, $t1
     addu      $t2, $t2, $t1
     addu      $t0, $t0, 4
cond:
     slt    $t1, $t0, $a1
     bne    $t1, $zero, loop
exit:
```

(a) What is the CPI of the two processors on the above implementations (you should choose the appropriate implementation for each processor)? Since N may be very large, we can assume that the CPI is roughly the same as that of one iteration of the loop. SHOW YOUR WORK. (15 points)

add => ALU 2

Lb => Ld/Str 3

addu => ALU 2

addiu => ALU 2

SLt => ALU 2

BNE => B/J 1

6 instr   12 cycles

2 CPI ✓

addu ALU 1

lw Ld/Str 2

raddu.qb SIMD 1

addu ALU 1

addu ALU 1

slt ALU 1

Bne B/J 2

7   9

9/7 CPI ✓

(b) (continuation of 4) What is the performance of ***the SIMD processor*** assuming that both processors operate at the same frequency? Hint: Remember why SIMD is beneficial to the number of instructions executed. SHOW YOUR WORK. (10 points)

Execution = IC · CPI ✓ / clk

6 · 2 / 2GHz vs

9 · (9/7) / clk

↑ depends on N

-2

(END OF EXAM)