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 #1
2/18/2019 9:00-9:50AM

Directions: There are 5 questions in this exam (on 4 pages). Each question is worth the 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	18	/ 25 points
2	5	/ 25 points
3	30	/ 30 points
4	0	/ 5 points
5	15	/ 15 points
Total	68	/ 100 points

+18

1. **MIPS Assembly and SIMD (25 points).** Consider the two following MIPS code segments.

```
char *a;                                    // assume a in register $a0
...
for (int i=0; i<N; i++) {                   // assume N in register $a1
    sum += a[i];
}
```

Implementation 1 (no SIMD instructions):

```
1
2       add     $t0, $zero, $zero
3       add     $t2, $zero, $zero
4       j       cond
5   loop:
6       addu    $t1, $a0, $t0
7       lb      $t1, 0($t1)
8       addu    $t2, $t2, $t1
9       addiu   $t0, $t0, 1
10  cond:
11      slt     $t1, $t0, $a1
12      bne     $t1, $zero, loop
13  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) (10 points) What is the minimum size of each binary in bytes? Show your work.

Each instruction is 32 bits long / 4 bytes long.

Implementation 1 : 9 lines

9 · 4 =

36 bytes

Implementation 2 : 10 lines

10 · 4 =

40 bytes

(b) (10 points) How many instructions get dynamically executed by each implementation in terms of N? You do not need to simplify, but SHOW YOUR WORK.

3 +

2 + Initial condition

6 · N

=

5 + 6N

3 +

2 + Initial condition

7 · N

=

5 + 7 · N

(c) (5 points) Describe one programming optimization you could perform to implementation 1 to save an additional instruction per iteration.

~~You could combine implementation 2 lines 7 & 8~~

how?

-2

-5

+5

2. **Psuedo-Instructions (25 points).** Consider the case of the MIPS `ulw` instruction that stores a word to an unaligned memory address (e.g., a word stored to the address 0x10010003). `ulw` is not supported in hardware, rather it is a pseudo-instruction.

(a) (15 points) Support `ulw` in software. First, finish labelling the memory diagram provided and shade in the bytes you are intending to load (you may assume either endianness). Second, using only the basic arithmetic, logical, and shift instructions, along with `lw`, provide a sequence of instructions that corresponds to the following `ulw` instruction instance.

```
ulw $a0, 3($t0)     # same operands as lw expects
                    # you may assume $t0's value is
                    # 0x10010000
```

Word Address	Byte 0	Byte 1	Byte 2	Byte 3
...				
				03
0x10010000				
...	04	05	06	

```
srl  $t1, $a0, 12     # Get most significant nibble
lw   $t1, 0($t0)      # load new 03
sll  $t1, $a0, 4      # Get 3 lower nibbles
lw   $t1, 1($t0)      # load new 04-06
```

ok... I see the way you labelled

still unaligned address

values never used

-10

(b) (10 points) Why does the MIPS ISA not implement a `ulw` instruction? Provide a *technical* reason. *[Hint: think about how you'd have to modify your Lab 4 datapath to support this.]*

Loading on an unaligned memory address would not be particularly useful, as the mips infrastructure is not equipped to handle something that is not aligned, and as such the stored data would be difficult to use

both already load such data

-10

+30

3. **MIPS Formats (30 points).** MIPS has three base instruction formats (see the R-, I-, and J-formats on the reference sheet attached to your exam). However, domain-specific experts with tunnel-vision always want more. Consider the four newly proposed formats below.

Format C (Compact):

Opcode		rs		rt		rd	
15	9	8	6	5	3	2	0

Format M (Multiple immediate):

Imm1		rt		Imm2		Opcode	
31	21	20	16	15	6	5	0

Format P (Partial) :

Opcode		rs		Imm	
31	26	25	21	20	0

Format E (Extended operands):

Opcode		rs		rt		rd		Funct	
31	26	25	21	20	16	15	11	10	0
Imm									
63									32

(a) Which formats are incompatible with the three base formats? Briefly explain why any total fail formats won't work.

Formats C + E are immediately incompatible, as each instruction is classically 32 bits in length. Format C as well can only access regs 0-7,

Format M has the opcode placed incorrectly, and as such could not reliably be identified

(b) Of the remaining formats, identify any additional hardware complexities that arise from the format. *[Hint: think about how you'd have to modify your Lab 4 datapath to support such operations.]*

These formats would require a good amount of additional overhead.

In the case of P, this would necessitate an additional muxe for the new imm with its longer bundle size

4. **ISA vs ABI Twitter War (5 Points).** In 140 chars or less (total): what is the primary purpose of an ISA? of an ABI? +0

The primary purpose of an ISA is to provide a medium to translate high level code into machine code.

An ABI does what an ISA does, but with hardware designed for that specific job.

5. **Processor Design Preparedness (15 Points).** +15

(a) (10 points) Dissassemble the following MIPS machine instruction (i.e., list the instructions mnemonic and the types and decimal values of the operand fields). Show your work!

0x00000000

R-Type: OP Rs Rt Rd Shamt Func.

000000 00000 00000 00000 00000 000000

sll $0, $0, 0

(b) (5 points) True or False: I am ready to design a processor. If False, why?

True (although not especially well). Oh whoops

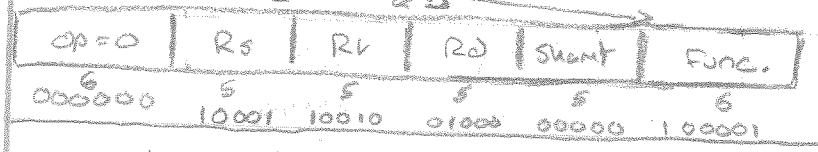
I have learned the building blocks of a processor, and using machine code, am able to appropriately direct data from memory or immediate.

(END OF EXAM)

381 Notes Sheet

Sean Gordon
Sgordon4
C/10:00

R: addu $t0, $s1, $s2



Used for Register/ALU ops.

I: lw $t2, 0($t1)

Load/Store
ALU ;imm.
Conditional Branch

J: j L1 Used if L1 is very far, too big for a branch or something

Jump
Jump & link

PC Relative Addressing When branching to an address is actually PC+4+4·imm

```
...
-1·4  0x1000  bne  $t2, $s3, Exit
 0·4  0x1004  addu ... ... ...
 1·4  0x1008  lw   ... ... ...
 2·4  0x100c  Exit: addiu $s2, $s1, 10
```

PC
0x1000
↓
0x100c

Exit == 2 because
PC+4+4·imm = 100c =>
100c - PC - 4 = 4·imm =>
100c - 1000 - 4 = 4·imm

J-type example; what if "L1" is very far away too far for the 16 bits?

```
beq $s0, $s1, L1
```

- can be rewritten as -

```
bne $s0, $s1, L2    # Inverted
j L1                # Unconditional, can use 16 bits
L2:
```

$1 or $at is reserved for the compiler

```
blt  $t0, $t1, L1   # Goto L1 if $t0 < $t1
 ⇓
slt  $at, $t0, $t1  # set $at = 1 if $t0 < $t1
bne  $at, $0, L1    # Goto L1 if $at != 0
```

Name	Reg #	Usage
$0, $zero	0	Constant 0
$1, $at	1	Reserved for pseudo instructions
$v0 - $v2	2-3	Values for results
$a0 - $a3	4-7	arguments/parameters
$t0 - $t7	8-15	temps.
$s0 - $s7	16-23	saved regs
$t8 - $t9	24-25	temps 2
$gp	28	Global pointer
$sp	29	Stack pointer
$Fp	30	Frame pointer
$ra	31	return address

$s0 - $s7 guaranteed to be preserved over a function call (convention)

$t0 - $t9 not guaranteed

$gp points to the center of the 64k block of memory in static data

$sp points to the last location on stack

$fp points to beginning of stack, not really