Name _____

Computer Engineering 331, Spring 2020 Exam 1 PRACTICE

Exam time: 5 pre + 70 minutes exam

Test Value: 30 **pts. Total possible points:** 32 **(max score =** 106.67**%)**

Total = /30	P1: /4	P2: /3	P3: /6	P4: /6	P5: /6	P6: /5	P7: /2

Front Left Neighbor # _____	**Front Neighbor #** _____	**Front Right Neighbor #** _____
Left Neighbor # _____	**Your Name (print clearly):** _____	**Right Neighbor #** _____
Rear Left Neighbor # _____	**Rear Neighbor #** _____	**Rear Right Neighbor #** _____

A. You will have 70 minutes to complete this exam. The exam is **closed book, closed notes, no calculators allowed**. Please remove all items from your desk other than pen or pencils, and erasers. Notes have been provided for you on the 2nd sheet of this exam (p3-p4), which can be removed once you have started writing. Additional scratch/answer overflow space is provided at the end of the exam.
B. Please turn off all cell phones, smartwatches, and other mobile devices. Remove all hats & headphones.
C. Place your backpacks, laptops and jackets under your seat.
D. There may be partial credit for incomplete answers; write as much of the solution as you can.
E. Show all of your work for a problem in the space provided on the exam sheet. If you need additional space, you must place a note with the problem indicating where the additional work is located. Answers (for problems requesting that work is shown) that lack derivations are worth no credit.
F. Please note that some problems may have multiple sub-problems and that point allocations are not always directly indicative of difficulty or problem solving time. You may wish to solve problems in an order other than that in which they are presented on the exam.

Recitation Section # (1-6): _____

Email: _____

Student ID Number: _____

I, the undersigned, declare the following: All the work presented on this assessment is my own. I had no prior knowledge of the exam contents, nor will I share the contents with others.

SIGN HERE: _____

Name _____

General instructions

Referring to a register, "current value" means the value in the register at the start of an instruction, which may be different from the "final value", the value in the register after an instruction has completed.

For any code response (e.g. P3-P5) your answer should be in MIPS assembler. You can use pseudoinstructions unless told not to. Comments are not required for code that is not pre-commented, but they may help with partial credit for free-response code sections.

Name _____

Some reminders about MIPS instructions, derived from P&H COD and the MARS help information. "imm$_x$" indicates a signed immediate represented in x bits and "imm$_{xz}$" represents an unsigned immediate. OP and FUNCT are given as decimal values. No pseudoinstructions are listed.

OP	FUNCT	INSTRUCTION	
0	32	add	$rd, $rs, $rt
8		addi	rt, rs, imm_{16}
9		addiu	rt, rs, imm_{16}
0	33	addu	$rd, $rs, $rt
0	34	sub	$rd, $rs, $rt
0	35	subu	$rd, $rs, $rt
28	2	mul	$rd, $rs, $rt
0	24	mult	$rs, $rt
0	25	multu	$rs, $rt
0	26	div	$rs, $rt
0	27	divu	$rs, $rt
0	36	and	$rd, $rs, $rt
12		andi	rt, rs, imm_{16z}
15		lui	rt, imm_{16z}$
0	39	nor	$rd, $rs, $rt
0	37	or	$rd, $rs, $rt
13		ori	rt, rs, imm_{16z}
0	38	xor	$rd, $rs, $rt
14		xori	rt, rs, imm_{16z}
0	0	sll	$rd, $rs, SHAMT
0	3	sra	$rd, $rs, SHAMT
0	2	srl	$rd, $rs, SHAMT
32		lb	rt, imm_{16}$($rs)
36		lbu	rt, imm_{16}$($rs)
35		lw	rt, imm_{16}$($rs)
40		sb	rt, imm_{16}$($rs)
43		sw	rt, imm_{16}$($rs)
0	42	slt	$rd, $rs, $rt
10		slti	$rd, $rs, $rt
11		sltiu	rd, rs, imm_{16z}
0	43	sltu	$rd, $rs, $rt
4		beq	rs, rt, imm_{16}
5		bne	rs, rt, imm_{16}
2		j	imm$_{26z}$
3		jal	imm$_{26z}$
0	9	jalr	$rs, $rd
0	8	jr	$rs

Memory addresses	Segment
0x7ffffefc ↓ $fp $sp	Stack Stack frame Stack top
	open
↑ 0x10040000	Dynamic Data
$gp 0x10000000	Static Data
0x0ffffffc $pc 0x00400000	Text
0x003ffffc 0x00000000	Reserved

Predetermined (word) address boundaries are indicated.

The boundary between Dynamic Data and Static Data is fixed, but its location depends on the program.

Assume that `$gp` is set to `0x10008000`, and does not change.

`$pc` is initialized to `0x00400000`.

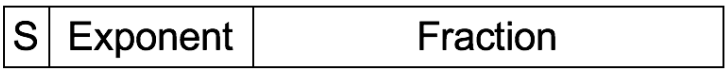
Name _____

Register Name	Register Number	Usage
$zero	0	Constant 0
$at	1	Reserved for assembler
$v0, $v1	2, 3	Function return values
$a0 - $a3	4 – 7	Function argument values
$t0 - $t7	8 – 15	Temporary (caller saved)
$s0 - $s7	16 – 23	Temporary (callee saved)
$t8, $t9	24, 25	Temporary (caller saved)
$k0, $k1	26, 27	Reserved for OS Kernel
$gp	28	Pointer to Global Area
$sp	29	Stack Pointer
$fp	30	Frame Pointer
$ra	31	Return Address

Single precision		Double precision		Object represented
Exponent	Fraction	Exponent	Fraction	
0	0	0	0	0
0	Nonzero	0	Nonzero	± denormalized number
1–254	Anything	1–2046	Anything	± floating-point number
255	0	2047	0	± infinity
255	Nonzero	2047	Nonzero	NaN (Not a Number)

single: 8 bits
double: 11 bits

single: 23 bits
double: 52 bits



$$x = (-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$$

Bias – Single: 127; Double: 1023
(in both cases, $2^{\text{ExponentBits}-1} - 1$)

Name _____

1. (4pts; 0.5 each) **General MIPS Knowledge part 1**
True or False. Circle T or F.

T F a. The MIPS architecture defines exactly 32 32-bit registers.

T F b. The MIPS `add` instruction operates on register values as if they were signed integers in 2's complement format.

T F c. The MIPS `addiu` instruction uses sign-extension on its immediate input.

T F d. The MIPS logical instructions on immediates (`ori`, `andi`, etc.) all use zero-extension.

T F e. The MIPS-32 ABI procedure call conventions require that a function can only access registers and the memory locations between the stack pointer and the frame pointer.

T F f. The Program Counter's name comes from the fact that it indicates how many instructions have been executed.

T F g. The MIPS assembler instruction (not pseudoinstruction) `addi $t1, $t2, 65536` is properly formulated (all fields are in required locations and are valid).

T F h. MIPS is a load-store architecture, which means that only load and store instructions access memory.

Name _____

2. (3pts; 1 each) **General MIPS Knowledge part 2**
Multiple choice. Circle only one of 1, 2, 3, 4 in each part.

a. What is the final value of `$pc` after an `add` instruction?
   1. the current value of `$pc` (it didn't change)
   2. the current value of `$pc + 4`
   3. the current value of `$pc + 8`
   4. none of the above

b. Which value is stored in `$ra` by the `jal` instruction?
   1. the current value of `$pc (address of jal instruction)`
   2. the current value of `$pc + 4`
   3. the current value of `$pc + 8`
   4. none of the above

c. Which of these would **not** be valid as the address of a MIPS instruction?
   1. `0x00400000`
   2. `0x00400298`
   3. `0x004002fc`
   4. none of the above – they are all valid instruction addresses, assuming the program is long enough

Name _____

3. (6 pts; 2 each). **Potpourri: Short answer**

a. Some MIPS instructions, like divu, take multiple cycles to execute, and changing the number of cycles required could have an impact on overall clock cycle time. If, on some **machine A** implementing the MIPS ISA and running a fixed workload, **W%** of dynamically executed MIPS instructions take **X** cycles to complete and all others take **1 cycle** to complete, then use the **Execution Time Equation** and **Amdahl's Law** to express the speedup relative to **A** for a **machine B** that reduces the number of cycles for each multi-cycle instruction by a factor of **Y** but increases the cycle time by a factor of **Z**.

b. Consider the following code with a branch instruction such as

```
        bne  $t0, $t1, L7
        addi $s0, $s0, 1
L7:     addi $a0, $a0, 4
```

When the assembler generates machine code for the branch instruction, it must replace the symbol `L7` by a number.

i) What is the value of the immediate used in the above branch instruction?

ii) Describe how this number is calculated.

c. Write a sequence of MIPS instructions that implements the pseudoinstruction ble $t0,$t1, L2 (branch less than or equal to). [Note, the register $at is available as a temporary, but you cannot perturb any other registers]

Name _____

4. (6pts; 2 each) **Mapping high-level code behavior to MIPS**
No more than 3 instructions each – longer responses will receive ½ credit.

Assume these declarations in C/C++:
```
int32_t Func(int32_t a);  int32_t Array[100];  int32_t b;
```

a. Assuming that `b` is currently mapped to `$s0` provide code that implements `b = Func(28);`

b. Load the value of `Array[j]` into register `$t1`. Assume the base address of Array is already in register `$s0`, `j` is already in register `$t0`, and `j` is a valid index into `Array`.

c. Assuming that `&Array[j+1]` is in `$t2`, provide code for the expression `Array[j] = Array[j+1]`

Name _____

5. (6pts) **Function calling in MIPS**
Complete the implementation of the following function, described below, in MIPS. You may use pseudoinstructions.

Consider the function `int sum(unsigned int i);`
that returns the sum of all numbers from 0 to i, implemented recursively as:

```
int  sum (int n) {
   if (0 == n) return 0;
   else return (n + sum(n-1));
}
```

Fill in the missing instructions below to correctly implement this function.
Not all _____ fields will necessarily need to be used (your implementation length may be shorter than the space provided)

```
.text
.globl sum


sum:
_____________________ # check if argument is non-zero
_____________________ # if was zero, set return value to 0
jr $ra                # return to caller
rSum:
_____________________ # reserve stack space
_____________________ # preserve registers across function call
_____________________ #
_____________________ #
_____________________ # compute n-1
_____________________ # call sum on n-1
_____________________ #
_____________________ # restore preserved registers
_____________________ #
_____________________ #
_____________________ # compute return value
_____________________ # release stack space
_____________________ #
jr $ra                # return to caller
```

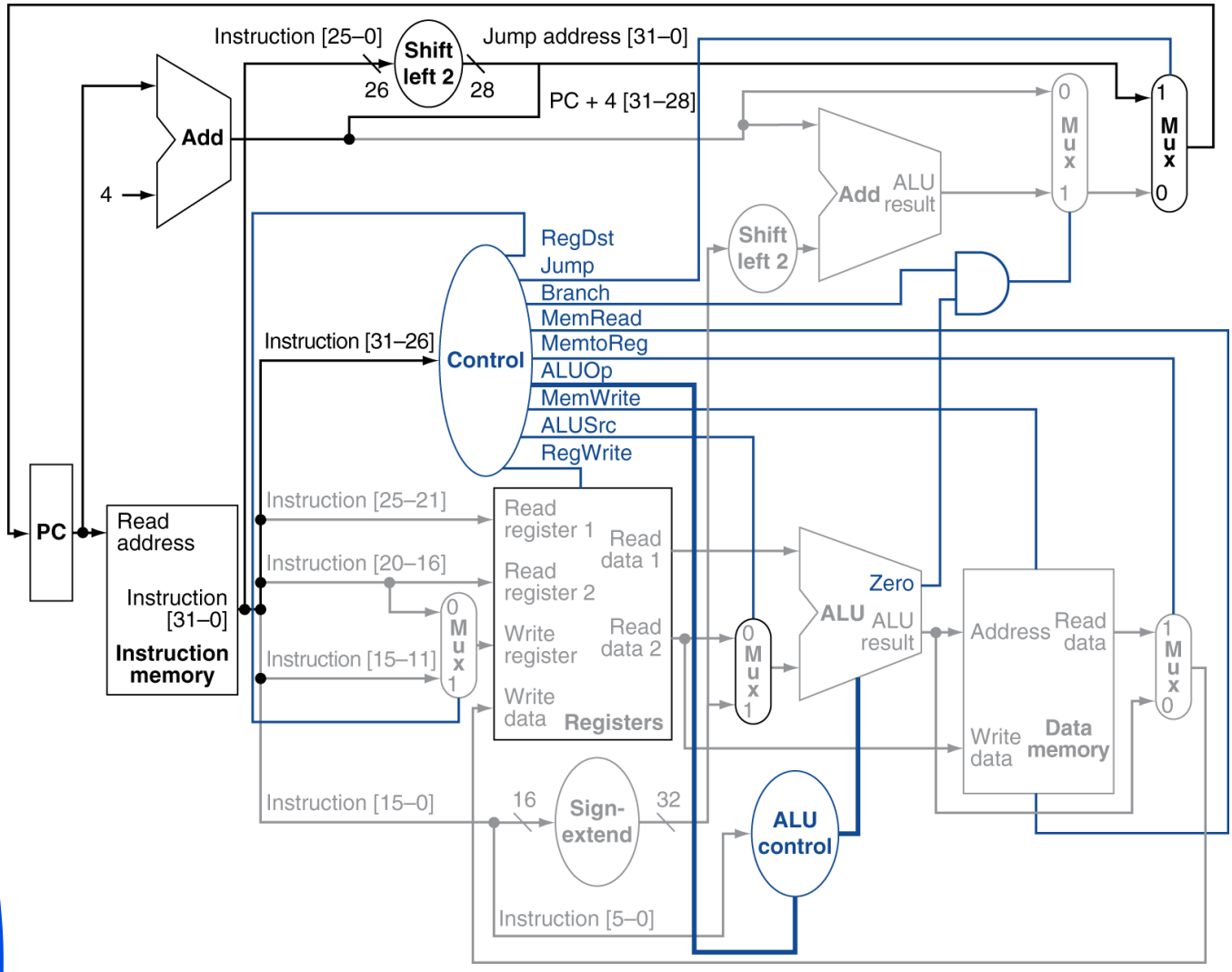
Name _____

6. (5pts; 4pts and 1 pt, respectively) **Implementing a MIPS processor.**

a) (4pts) Modify the below diagram to implement the (fictional) instruction `jSum $rs, $rt`

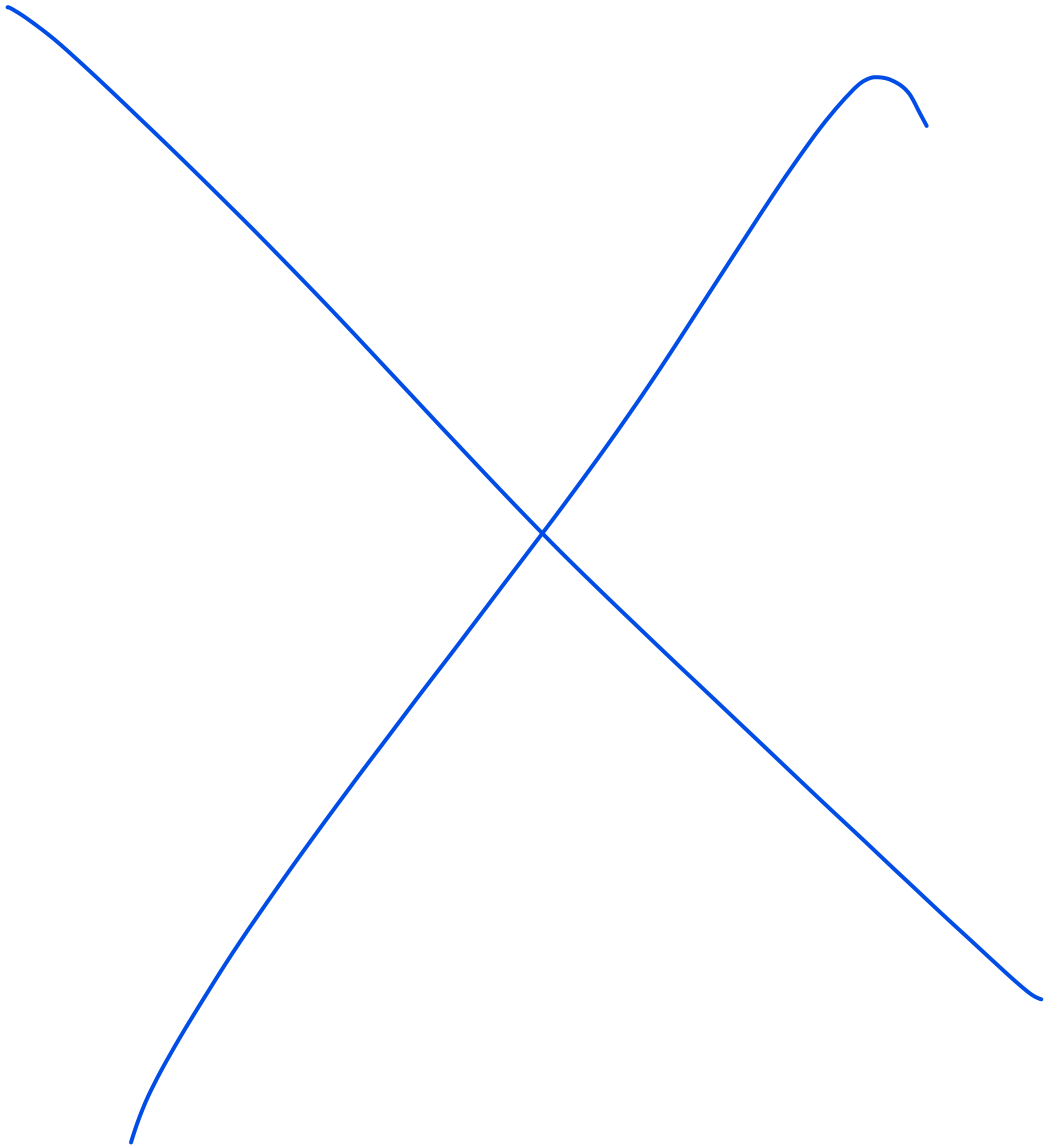
with RTL semantics: `PC ← PC + 4 + REG[$rs] + REG[$rt]`

by adding additional muxes and control lines, as necessary. Specify specific, Boolean values for all outputs of the **Control** module, including any newly added outputs, except for *ALUOp* (which can be presumed to be properly specified to cause addition to happen).



Name _____

b) (1 pt) Datapaths need to select for all possible places an operand may come from, which may not always be a power of 2. Specify a Verilog module that performs 5:1 multiplexing on five 32-bit wide busses.



Name _____

7. (2 pts) **Support for complex arithmetic functionality**

Consider the following five floating point numbers.

A = 1.0000×2^{-1}

B = 1.0000×2^{70}

C = 1.0000×2^{70}

D = 1.0000×2^{100}

E = 1.0000×2^{40}

Assume each is stored as a 32-bit single precision IEEE standard 754 floating point number, and the compiled version of the a-e options below, as C code, will be run on an IEEE 754 floating point compliant floating point unit.

Which one of the following placements of parentheses in expressions a-e will lead to the result of

F = A + B * C / D - E = 1.0000×2^{-1} ?

a) (A+(B*C)/D)-E

b) (A+B*(C/D))-E

c) A+((B*C)/D-E)

d) A+(B*(C/D)-E)

e) All of the above because addition and multiplication are associative

Name ____

SCRATCH PAPER

Name ____

SCRATCH PAPER