Name _____

Computer Architecture
LSU EE 4720
Midterm Examination

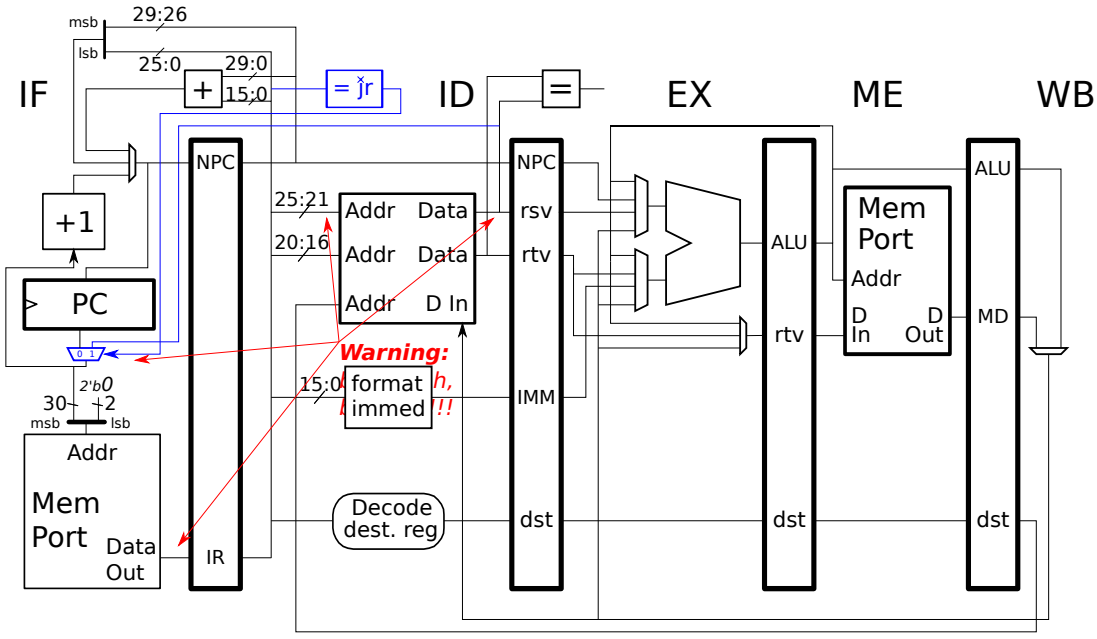
Wednesday, 27 March 2019, 9:30–10:20 CDT

Problem 1	_____	(7 pts)
Problem 2	_____	(17 pts)
Problem 3	_____	(27 pts)
Problem 4	_____	(12 pts)
Problem 5	_____	(12 pts)
Problem 6	_____	(25 pts)
Exam Total	_____	(100 pts)

Alias _____

Good Luck!

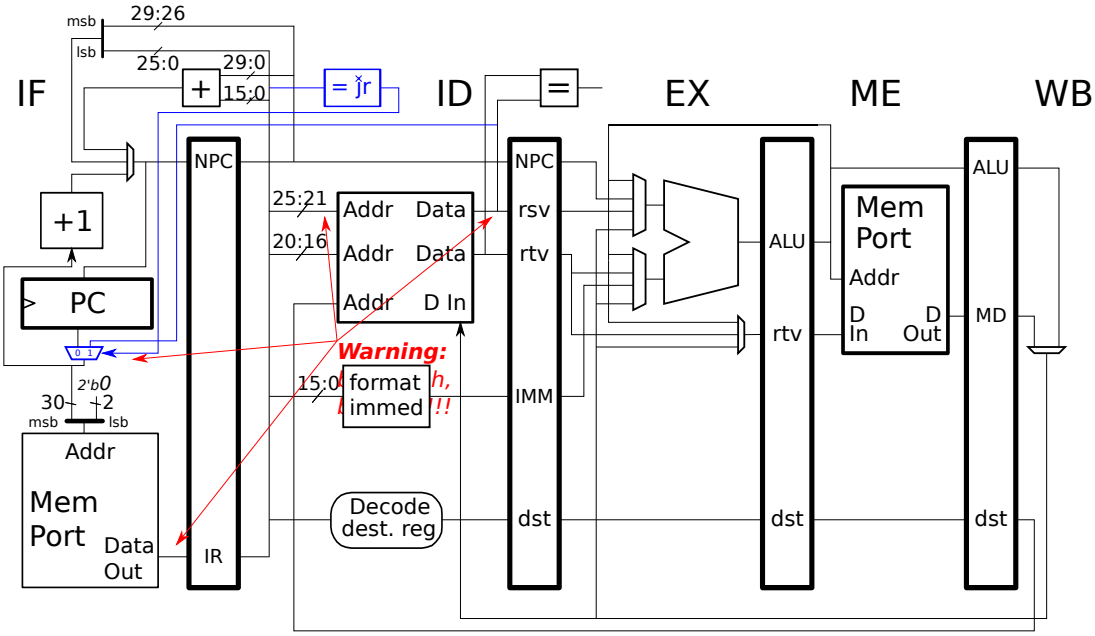
Problem 1: [7 pts] The MIPS pipeline below implements a hypothetical MIPS `jr` instruction, the hardware for `jr` is shown in blue. Don't confuse `jr` with the existing MIPS `jr` instruction.



(*a*) The diagram shows a warning in red with lots of arrows and an explanation. Alas, the explanation is covered by the `format immed` box. **Assume that the hardware for `jr` is correct.** Then what can the warning be about?

☐ Reason for warning:

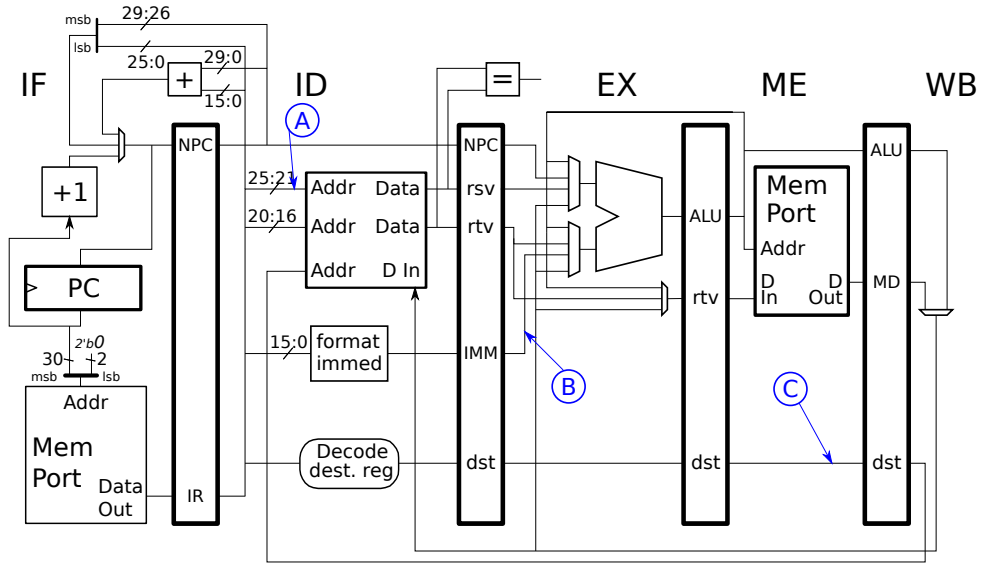
Problem 1, continued: (*b*) *Note: This part did not appear on the exam because the exam was already long enough.* There are two differences between ~~`jr`~~ and `jr`. Fragment A, below, uses `jr`. Complete Fragment B so that it uses ~~`jr`~~, making changes to account for these two differences. Fragment B must jump to the same location and perform the same computation as Fragment A. Register `r9` can be used for intermediate values. *Hint: The differences are when and where.*



☐ Complete code, or for partial credit explain two differences.

```
# Fragment A -- Uses jr. Don't modify it.
lw r1, 0(r6)
jr r1
addi r2, r2, 4
xor r3, r4, r5


# Fragment B --  ☐  Finish code below, use ~~jr~~

lw r1, 0(r6)

addi r2, r2, 4

xor r3, r4, r5
```

Problem 2: [17 pts] The code below executes on the illustrated implementation. The implementation has hardware that enables the `bne` to avoid the stall, but that hardware is not shown.



(*a*) The illustration has several circled letters pointing to wires. In the diagram below show the values on those wires on each cycle the value is used.

☐ Show values for ☐ A, ☐ B, and ☐ C, show these ☐ for each cycle used.

```
LOOP: # Cycle    0    1    2    3    4    5    6    7    8    9    10   11
 lhu r1, 8(r2)   IF   ID   EX   ME   WB
 addi r2, r2, 2       IF   ID   EX   ME   WB
 add r3, r1, r3            IF   ID   EX   ME   WB
 sw r3, 12(r6)             IF   ID   EX   ME   WB
 slt r5, r3, r4                 IF   ID   EX   ME   WB
 bne r5, r0  LOOP               IF   ID   EX   ME   WB  # No stall? Next prob.
 addi r6, r6, 4                      IF   ID   EX   ME   WB

#       Cycle    0    1    2    3    4    5    6    7    8    9    10   11

A:

B:

C:

#       Cycle    0    1    2    3    4    5    6    7    8    9    10   11
```

(*b*) For each dependency below highlight, on the illustration, the multiplexor input that provides the bypassed value. Also indicate the cycle in which the bypass occurs.

☐ Dependence from `lhu` to `add r3`. ☐ Cycle when bypass used.

☐ Dependence from `add r3` to `sw`. ☐ Cycle when bypass used.

☐ Dependence from `add r3` to `slt`. ☐ Cycle when bypass used.

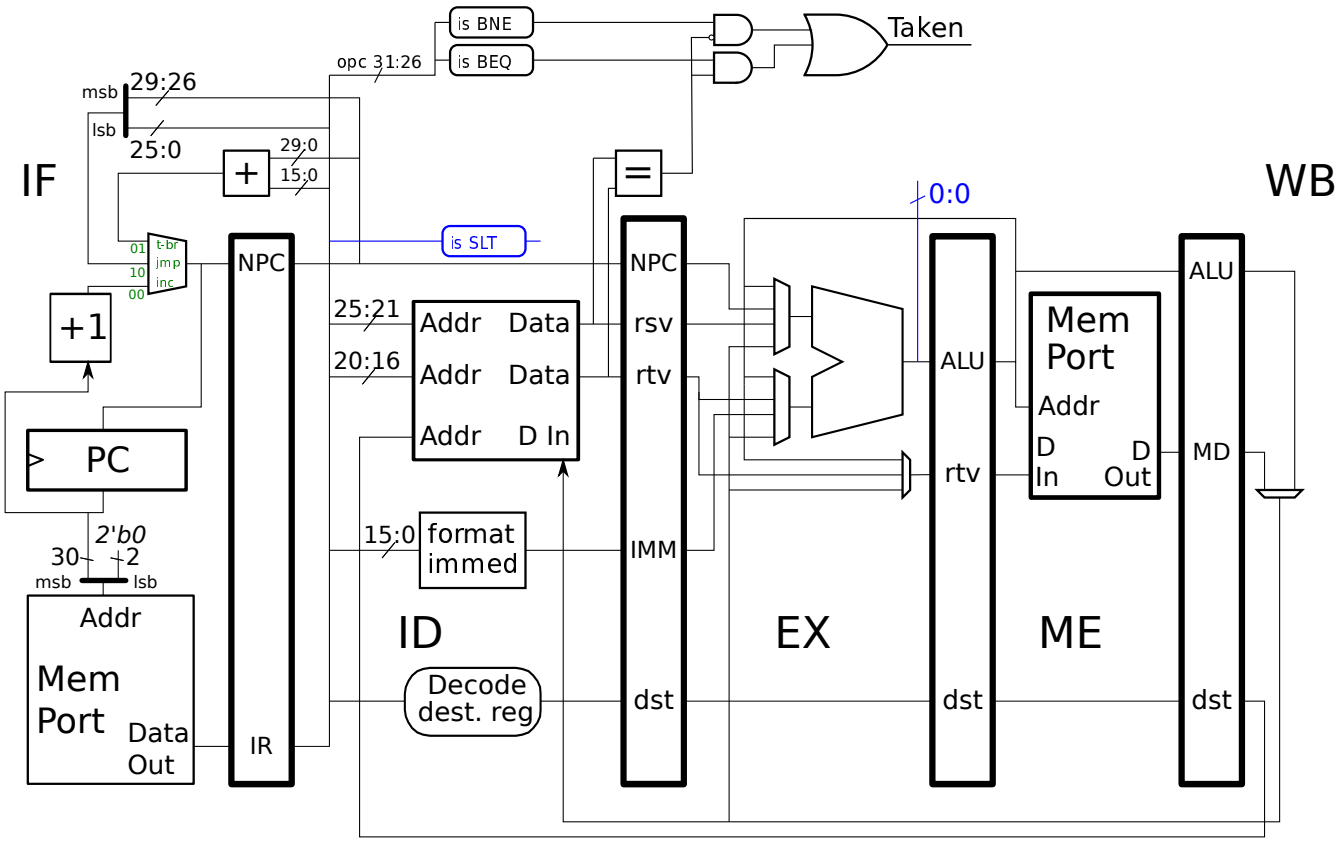
Problem 3: [27 pts] In the previous problem the `bne` did not stall despite a dependence with `slt`. Code with a similar dependence appears below. Add hardware to the implementation below that correctly sets the Taken signal for such `slt rX, rY, rZ` to `bne rX, r0` dependencies. Branches without the dependence should not be effected. Note that the result of `slt` is 0 or 1. Some useful hardware is shown in blue.

```
LOOP: # Cycle      0  1  2  3  4  5  6
 slt r5, r3, r4    IF ID EX ME WB       # Note: r5 is 0 or 1.
 bne r5, r0  LOOP     IF ID EX ME WB
 addi r6, r6, 4          IF ID EX ME WB
```

☐ Add logic to set `Taken` correctly for the dependence described above.

☐ Avoid costly solutions. No part of the solution should operate on 32 bits.

☐ Check that the second branch register is `r0`. ☐ Branches without the dependence should not be effected.



Problem 4: [12 pts] The loop below writes zeros to a range of memory.

```
 # Call Value: r1 is address of the start of the region to zero.
 # Call Value: r3 is the number of bytes to zero.
 add r2, r1, r3  # Memory location at which to stop.
 addi r2, r2, -1
LOOP:
 sb r0, 0(r1)
 bne r1, r2, LOOP
 addi r1, r1, 1
```

(*a*) Compute the rate that it zeros memory when it runs on our bypassed 5-stage pipeline. Use an appropriate unit.

☐ Rate at which loop above copies data.

(*b*) Apply loop unrolling and make other changes so that the code writes at the rate of two bytes per clock cycle, assuming favorable values of `r1` and `r3`. State those assumptions.

☐ Show unrolled loop and make other changes for 2 byte per cycle copy.

☐ Assumption about `r1`:

☐ Assumption about `r3`:

```
 # Call Value: r1 is address of the start of the region to zero.
 # Call Value: r3 is the number of bytes to zero.
 add r2, r1, r3

 addi r2, r2, -1
LOOP:
```

Problem 5: [12 pts] The MIPS code below adds an integer value loaded from memory to the constant π.

(*a*) Suppose no other instructions needed the value of `a0` that was loaded. Modify the code to use fewer instructions.

☐ Use fewer instructions.

```
        .data
famous_constant: # 0x10010300
        .float 3.141592654
        .text
pie:

        lw $a0, 0($t2)

        lui $t0, 0x1001
        lwc1 $f0, 0x300($t0)

        mtc1 $f1, $a0
        cvt.s.w $f2, $f1
        add.s $f3, $f0, $f2
```

(*b*) Modify the code below so that it would run correctly if `a0` were a floating-point value. Also use fewer instructions.

☐ Fix code so it works correctly if `a0` is FP.

```
        .data
famous_constant: # 0x10010300
        .float 3.141592654
        .text
pie:

        lw $a0, 0($t2)  # a0 is FP!  ☐  Fix code below for FP a0.

        lui $t0, 0x1001
        lwc1 $f0, 0x300($t0)

        mtc1 $f1, $a0
        cvt.s.w $f2, $f1
        add.s $f3, $f0, $f2
```

Problem 6: [25 pts] Answer each question below.

(*a*) SPARC divides the 32 integer registers an instruction can access into four groups, `%l0 to %l7`, `%g0 to %g7`, `%o0 to %o7`, and `%i0 to %i7`. The names reflect how they might be used in programs, and how SPARC's register windowing feature affects them when `save` and `restore` instructions are executed. Explain what the first letter of each group stands for. Explain what happens to the values in those registers when `save` or `restore` instruction is executed.

☐ Word that `l`, `g`, `o`, and `i` each stand for.

☐ What happens to values on a `save` or `restore`.

(*b*) Instructions like `addi r1, r1, 1` occur frequently in programs. For this add-one-to-a-register operation CISC ISAs have a specialized instruction, for example `inc r1`. MIPS lacks such an increment instruction.

☐ Why do MIPS and other RISC ISAs lack such an instruction?

☐ Why do CISC ISAs have such an instruction?

☐ What is the benefit that RISC ISAs don't realize?

(*c*) A CPU design team has an area budget that they can use for bypass paths. There's not enough area for all the bypass paths they'd like. They are simulating these design alternatives to determine which is best.

Explain the role that the compiler people have in this process.

☐ Role for compiler writers in deciding on bypass paths.

(*d*) Provide examples for the following common optimizations. Show code to which this optimization can apply and how it is optimized.

☐ Dead-code elimination. ☐ Example code and code after optimization.

☐ Constant propagation and folding. ☐ Example code and code after optimization.

(*e*) Described below are three tuning levels for C++ programs, two of which are SPECcpu tuning levels the other was just made up. Identify the one which is not a SPEC tuning level, and explain why it isn't.

Level A: *Each C++ program is compiled without optimization.*

☐ Is it a SPEC level? _____ ☐ If so, name it: _____

☐ Indicate the users that will benefit from this level, or why no one would benefit.

Level B: *Each C++ program can have its own set of optimization flags.*

☐ Is it a SPEC level? _____ ☐ If so, name it: _____

☐ Indicate the users that will benefit from this level, or why no one would benefit.

Level C: *All C++ programs must be compiled with the same set of optimization flags.*

☐ Is it a SPEC level? _____ ☐ If so, name it: _____

☐ Indicate the users that will benefit from this level, or why no one would benefit.