Name _____

Computer Architecture
LSU EE 4720
Solve-Home Final Examination

Wednesday, 6 May 2020 to Saturday, 9 May 2020 5:00 (5 AM) CDT

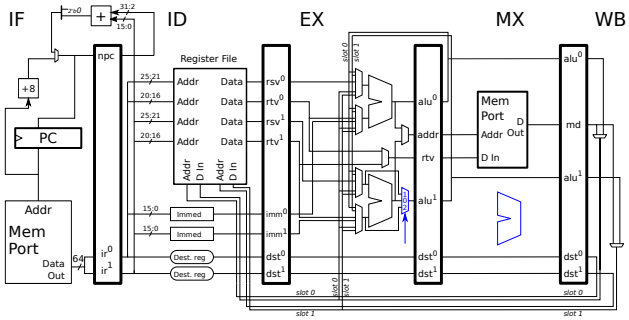
Work on this exam alone. Regular class resources, such as notes, papers, documentation, and code, can be used to find solutions. Outside material that covers the same topics, such as MIPS tutorials, digital logic design guides, and computer architecture references can also be used. Do not try to directly seek out solutions to any question here. That is, don't Web-search the text of a problem. Do not discuss this exam with classmates or anyone else, except questions or concerns about problems should be directed to Dr. Koppelman.

Problem 1	_____	(30 pts)
Problem 2	_____	(25 pts)
Problem 3	_____	(15 pts)
Problem 4	_____	(30 pts)
Exam Total	_____	(100 pts)

$r \geq 2\,\mathrm{m} \quad \Rightarrow \quad R_e < 1$

Good Luck! Help Keep Everybody Safe!

Problem 1: (30 pts) The two-way superscalar MIPS implementation below has an ALU in the `MX` (née `ME`) stage, call it the *second-chance ALU*. For this problem the second-chance ALU will be connected so that two stall situations are avoided. *A slightly similar problem appeared on the Spring 2006 Midterm Exam in Problem 2. It's okay to look at the problem and solution.*



Yes, it's small! Use the next page for the solution.

(*a*) The `sub` in the code below suffers a stall due to a dependence with the other instruction in the same fetch group. Connect the second-chance ALU so that the stall is avoided. The changes must not break existing functionality and must not result in stalls for unrelated code. In particular note that the `add` does not stall in either version.

```
# Cycle           0  1  2  3  4  5  6  7   # Unmodified Implementation
ori R3, r1, 0xff  IF ID EX ME WB
xor R2, r8, r9    IF ID EX ME WB
add R1, R2, R3       IF ID EX ME WB
sub R4, R1, r5       IF ID -> EX ME WB
and r7, r8, R4          IF -> ID EX ME WB
# Cycle           0  1  2  3  4  5  6  7

# Cycle           0  1  2  3  4  5  6  7   # With second-chance ALU.
ori R3, r1, 0xff  IF ID EX MX WB
xor R2, r8, r9    IF ID EX MX WB
add R1, R2, R3       IF ID EX MX WB        # No stall in either implementation.
sub R4, R1, r5       IF ID EX MX WB        # No stall due to second-chance ALU!
and r7, r8, R4          IF ID -> EX MX WB  # Stall due to second-chance ALU.
# Cycle           0  1  2  3  4  5  6  7
```

☐ Connect second-chance ALU to avoid the stall by the `sub` and allowing code to execute as in the sample above. ☐ The connections should work for any pair of dependent, ALU-using, non-memory instructions.

☐ Pay attention to cost. Assume that a pipeline latch bit costs twice as much as a multiplexor bit.

☐ Do not add unneeded bypass paths. ☐ Don't break existing functionality.

(*b*) The code below suffers a load/use stall. Add the minimum number of connections to the second-chance ALU so that such load/use stalls (in which the using instruction is in slot 1) can be avoided.

```
add r3, r2, r3    IF ID EX ME WB
lw r4, 5(r1)      IF ID EX ME WB
ori r6, r1, 0xff     IF ID EX ME WB
sub r5, r3, r4       IF ID -> EX ME WB
```

☐ Add connections to the second-chance ALU to avoid load/use stalls when the using instruction (such as the `sub` in the example) is in slot 1.

☐ Pay attention to cost, use the same cost assumption as given in the previous part.

(*c*) In the code below the `sub` does not stall due to the second-chance ALU but the `and` does stall. Add control logic to generate a stall signal for cases such as this.

```
# Cycle           0  1  2  3  4  5  6  7
ori R3, r1, 0xff  IF ID EX MX WB
xor r2, r8, r9    IF ID EX MX WB
add R1, r2, R3       IF ID EX MX WB
sub R4, R1, r5       IF ID EX MX WB        # No stall due to second-chance ALU!
and r7, r8, R4          IF ID -> EX MX WB  # Stall due to second-chance ALU.
# Cycle           0  1  2  3  4  5  6  7
```

☐ Add logic to generate a stall signal for the situation described above. ☐ The logic should work for any instruction dependent on an instruction using the second-chance ALU.

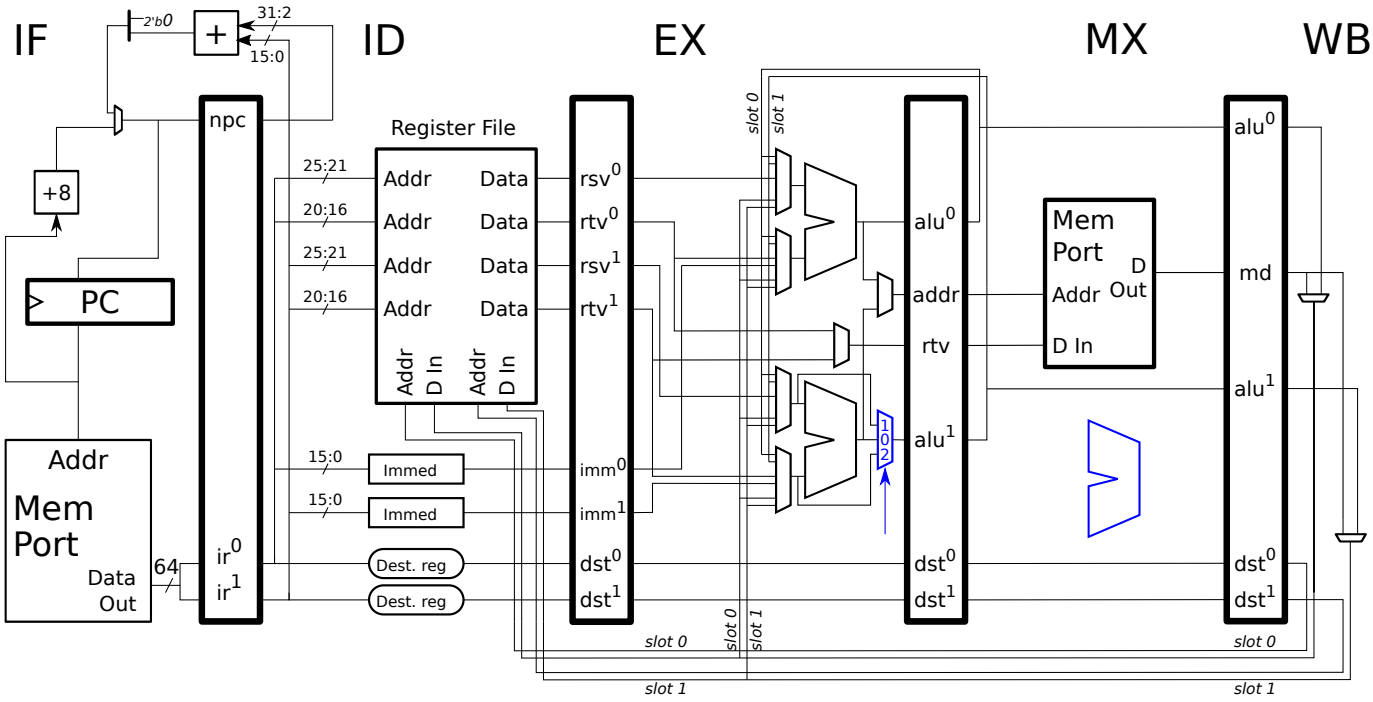
☐ Pay attention to cost, use the same cost assumption as given in the previous part.

(*d*) Generate the select signal for the `EX` stage multiplexor shown in blue. The control logic should work for the intra-group dependence case. (The control logic does not need to work for the load/use case.)

☐ Add logic for the select signal for the intra-group dependence. ☐ The logic should work for any pair of dependent, ALU-using, non-memory instructions.

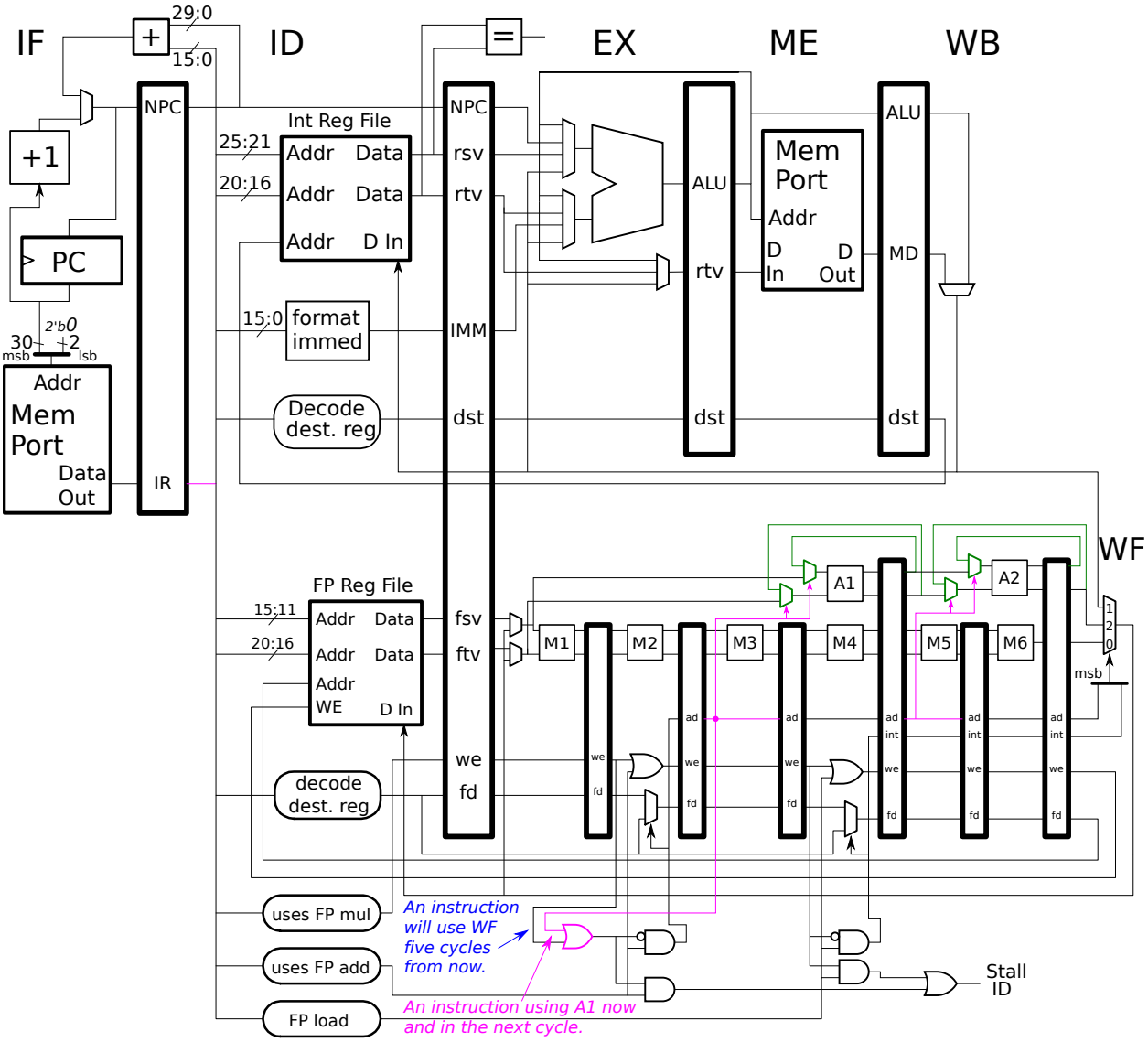
☐ Pay attention to cost, use the same cost assumption as given in the previous part.

The Inkscape SVG source is at `https://www.ece.lsu.edu/ee4720/2020/fe-p1-v2-ss.svg`.



Problem 2: (25 pts) Show the execution of the code fragments as requested below.

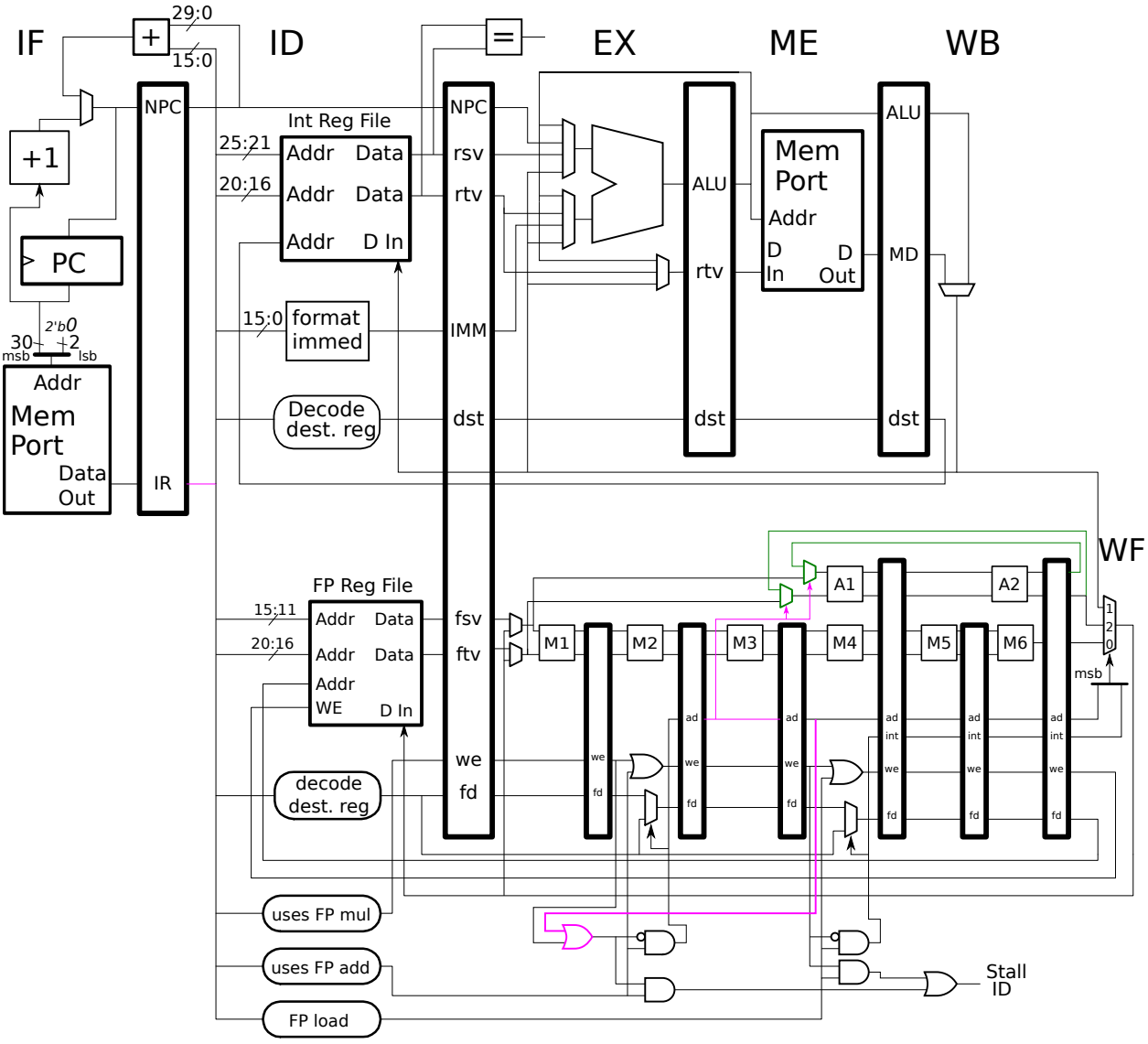
(*a*) Show the execution on the FP pipeline below, note that the adder unit has an initiation interval of 2.



☐ Show execution. ☐ Pay attention to how the FP add unit should operate. ☐ Don't forget to check for dependencies.

```
lwc1 f2, 0(r1)

add.s f0, f2, f4

add.s f1, f2, f5

add.s f3, f1, f6
```

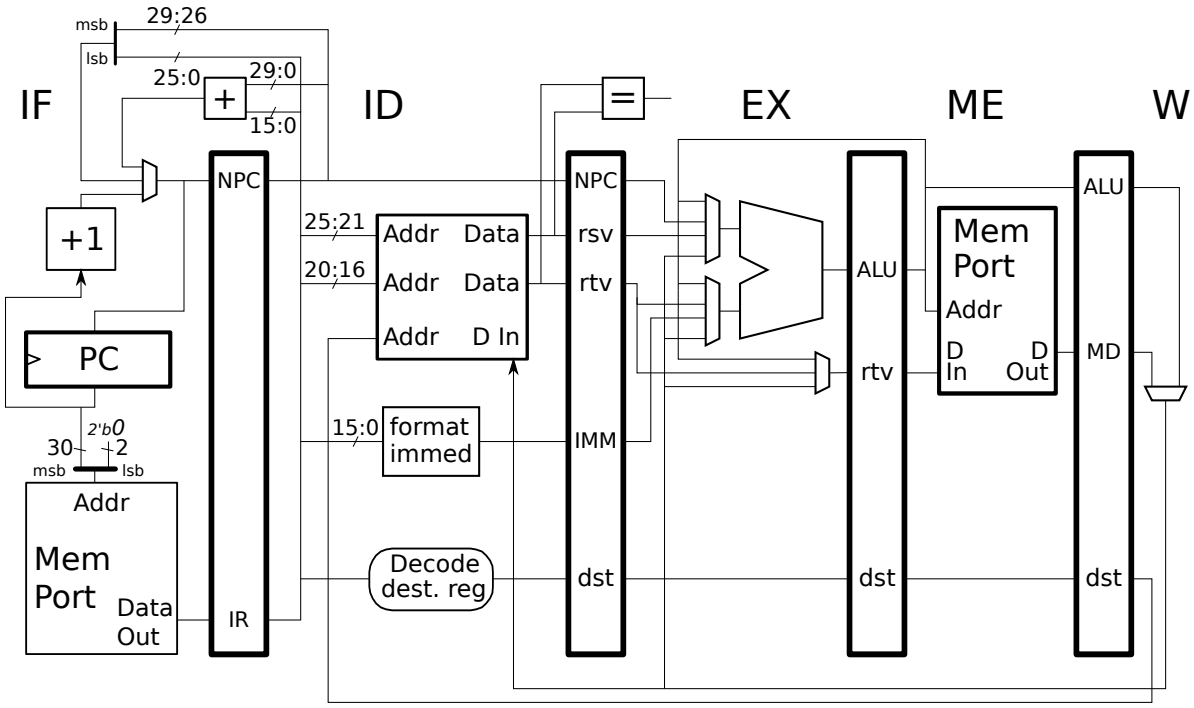
(*b*) Show the execution on the FP pipeline below, note that the adder unit is different than the previous problem and from other examples covered in class.



☐ Show execution. ☐ Pay attention to how the FP add unit should operate. ☐ Don't forget to check for dependencies.

```
lwc1 f2, 0(r1)

add.s f0, f2, f4

add.s f1, f2, f5

add.s f3, f1, f6
```

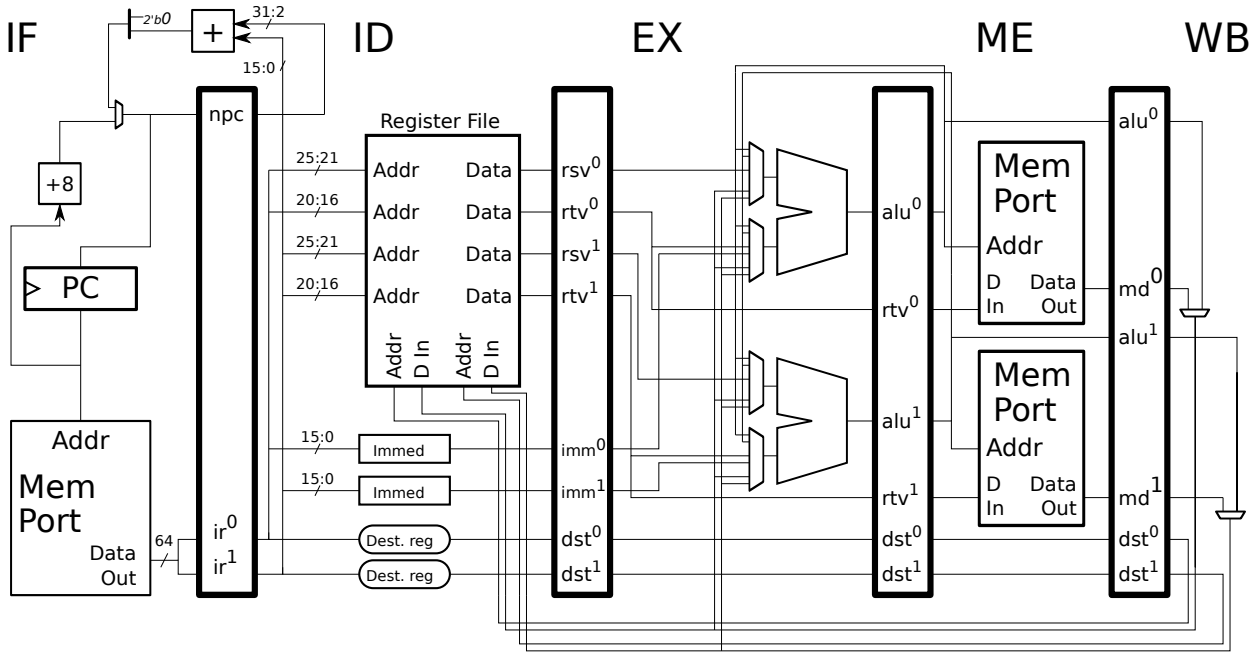
(*c*) Show the execution of the code on the implementation below. Find the CPI for a large number of iterations.



☐ Show execution on the illustrated implementation with the branch taken. ☐ Find the CPI for a large number of iterations.

```
LOOP:
 addi r2, r2, 16

 lw r1, 8(r2)

 sw r1, 12(r3)

 bne r3, r4, LOOP

 addi r3, r3, 32

 sub r10, r3, r2
```

(*d*) Show the execution of the code on the 2-way superscalar MIPS implementation illustrated below, and find the CPI for a large number of iterations. This is not the same as the implementation from Problem 1. Instruction fetch is of aligned groups.



☐ Show execution on the illustrated implementation. ☐ Find the CPI for a large number of iterations.

☐ Take aligned fetch into account, the address of `LOOP` is `0x1000`. ☐ Pay attention to available bypass paths.

```
LOOP:
 addi r2, r2, 16

 lw r1, 8(r2)

 sw r1, 12(r3)

 bne r3, r4, LOOP

 addi r3, r3, 32

 sub r10, r3, r2
```

(*e*) The code fragment below is the same as the one from the previous problem and is to run on the same superscalar system. Re-write the code so that it runs with fewer stalls (and of course does the same thing), and compute the CPI for a large number of iterations. Extra instructions can be added before or after the loop. Do not unroll the loop.

☐ Re-write code to minimize stalls on the superscalar implementation.

☐ Compute the CPI of the re-written code for a large number of iterations.

```
LOOP:
 addi r2, r2, 16
 lw r1, 8(r2)
 sw r1, 12(r3)
 bne r3, r4, LOOP
 addi r3, r3, 32
 sub r10, r3, r2
```

Problem 3: (15 pts) Answer the following branch prediction questions.

(*a*) Code producing the branch patterns shown below is to run on several systems, each with a different branch predictor. All systems use a 2^{12} entry BHT. One system has a bimodal predictor and the other systems have a local predictor, the length of the local history is given in the questions below.

Answer each question below, the answers should be for predictors that have already warmed up. Show work or provide brief explanations.

```
B1:   N  N  N  N  T  T    N  N  N  N  T  T   ...

B2:      T  T  T  T  T  T    T  T  T  T  T  T   ...
```

☐ What is the accuracy of the bimodal predictor on branch `B1`?

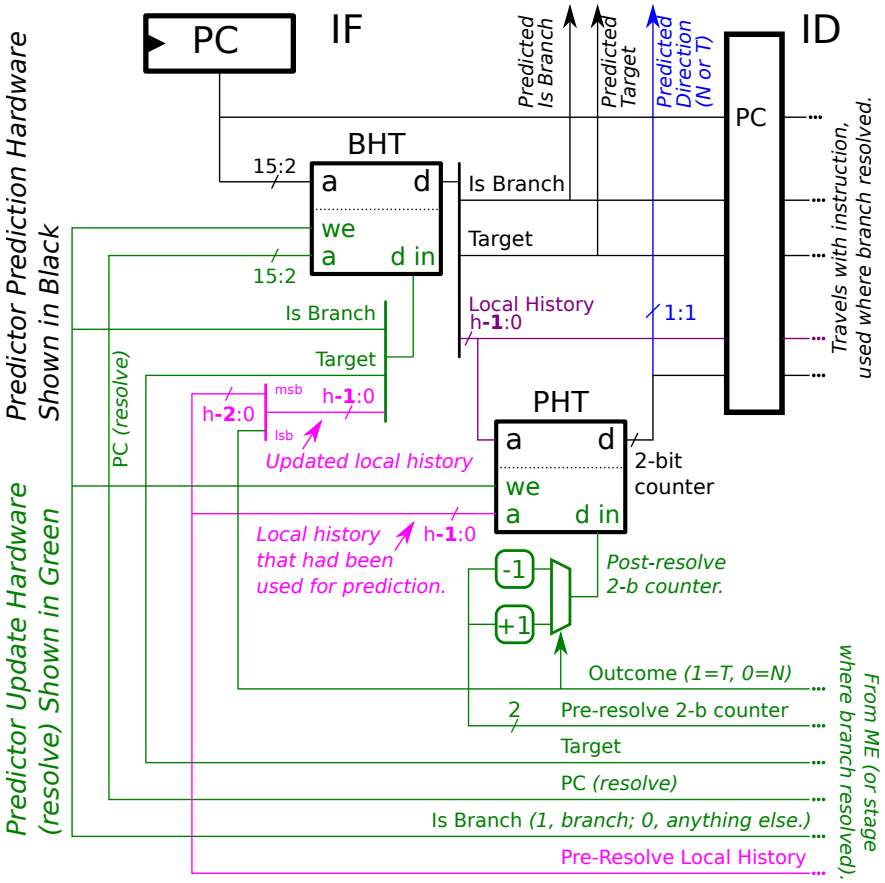
☐ What is the accuracy of a local predictor with a 12-outcome local history on branch `B1` and ignoring `B2`.

☐ What is the accuracy of a local predictor with a 2-outcome local history on branch `B1` and ignoring `B2`.

☐ What is the accuracy of a local predictor with a 2-outcome local history on branch `B1` and taking into account `B2`.

☐ What is the minimum local history size so that branch `B1` is predicted with 100% accuracy, taking into account `B2`.

(*b*) Appearing below is a diagram of a local predictor. The local predictor illustrated has a specific BHT size and an h-outcome local history size. The BHT size does not necessarily match the BHT in the prior part. Determine the amount of storage, in bits, used by the BHT and PHT for a 16-bit local history. Assume that Target is stored efficiently.



☐ Amount of storage for PHT is:

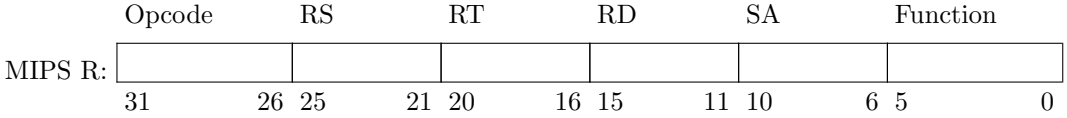
☐ Assumption about Target:

☐ Amount of storage for BHT is:

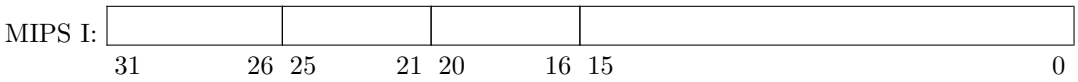
Problem 4: (30 pts) Answer each question below.

(*a*) Appearing below are the three integer MIPS I instruction formats. Consider a modified form of MIPS in which there are 64 rather than 32 integer registers. A goal is compatibility with MIPS-I code and to use as few new opcodes and function field values as possible. Modify each format so that it can use 64 registers and explain what new opcodes (if any) are needed and any assumptions about existing MIPS-I instructions. *Hint: For one case there's nothing to do, for one case many opcodes will be needed, for one case only a few.*

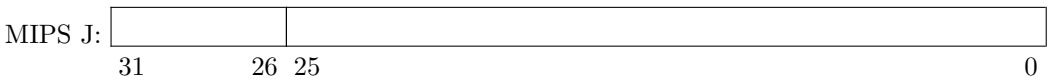
☐ Modification for 64-register MIPS. ☐ Describe what new opcode/func values are needed for, if any.



☐ Modification for 64-register MIPS. ☐ Describe what new opcode/func values are needed for, if any.



☐ Modification for 64-register MIPS. ☐ Describe what new opcode/func values are needed for, if any.



(*b*) Chip A has five 4-way superscalar cores. Chip B has 20 scalar cores. The cores are similar to our pipelined MIPS implementations. All cores use a 1 GHz clock. *Yes, up until this comment the question is identical to one asked on the Spring 2019 final exam.* The SPECcpu benchmarks are run on each chip. Recall that SPECcpu can be run to compute a speed score and a rate score. (Don't confuse speed/rate with base/peak or int/FP.) Feel free to visit the SPEC site to help answering this question.

☐ Which chip would likely score higher (better) on the SPECspeed2017int benchmarks,
◯ *Chip A* or ◯ *Chip B* . ☐ Explain.

☐ Which chip would likely score higher (better) on the SPECrate2017int benchmarks,
◯ *Chip A* or ◯ *Chip B* . ☐ Explain.

(*c*) Our goal is to build a machine that can execute eight floating-point operations per cycle. Two machines are under consideration, an 8-way superscalar system implementing ISA I, and a 2-way superscalar system with an 8-lane vector unit implementing ISA IV, which is like I but with vector instructions. Both machines run at 1 GHz, and both can sustain eight billion floating-point operations per second.

☐ Which machine is likely to be more expensive? ☐ Explain.

☐ Which machine is likely to be faster on typical code? ☐ Explain with ☐ a code example.

(*d*) In MIPS and many other RISC ISAs memory accesses must be aligned. For example, a `lw` instruction, which loads a four-byte value, must load from an address that is a multiple of 4. The execution of a `lw` loading from an address that is not a multiple of 4 will result in an exception (and on Linux system resulting in the Bus Error signal handler being called). As we pointed out in class, integer instructions, and especially load and store instructions, in any reasonable ISA would be required to raise precise exceptions. MIPS is certainly reasonable in this respect.

Suppose that a program uses non-aligned addresses in memory accesses, but is otherwise correct. That is, the program would run correctly if the load could handle a non-aligned address. (After all, CISC ISA load instructions can do it.) But on MIPS it raises an exception as soon as a non-aligned load or store is attempted. Suppose further that re-writing the program is not feasible.

☐ Explain how we can take advantage of precise exceptions so that this program would run correctly. A code example would be nice but not necessary.

☐ Explain why it would be impossible if loads only raised deferred exceptions. (Assume that aligned accesses work fine with such loads.)

(*e*) MIPS has a `slt` (set less than) instruction, but doesn't have a `sge` (set greater than or equal to) instruction. Why not?

☐ Why doesn't MIPS have an `sge` instruction?

(*f*) *Note: The following question was asked about two months after in-person classes were ended for the CoViD-19 pandemic.*

Perhaps many of us are wishing we could go back in time. (Not to warn people, that's obviously futile.) Wish granted. You are in a meeting (in person, not Zoom) with future Turing Prize winners discussing which features to put into their new [airquotes] "RISC" ISA, MIPS.

One attendee is advocating for the inclusion of magnitude comparison branch instructions such as `bgt r1,r2 TARG` (branch if `r1` greater than `r2`). But many others oppose the idea because it would have too much critical path impact. "We can include `bgt` with zero critical-path impact if we use a surprisingly simple but effective technique called branch prediction," you say.

☐ Explain how branch prediction can remove the critical path impact that was a concern at the meeting.

☐ Was the phrase *branch prediction* an anachronism at that fictional meeting? Web-search freely to answer this question.