Name _____ *Formatted For 2-Sided Printing*

Staple This Side

Computer Architecture
LSU EE 4720
Final Examination

Tuesday, 5 May 2026 15:00-17:00 CDT

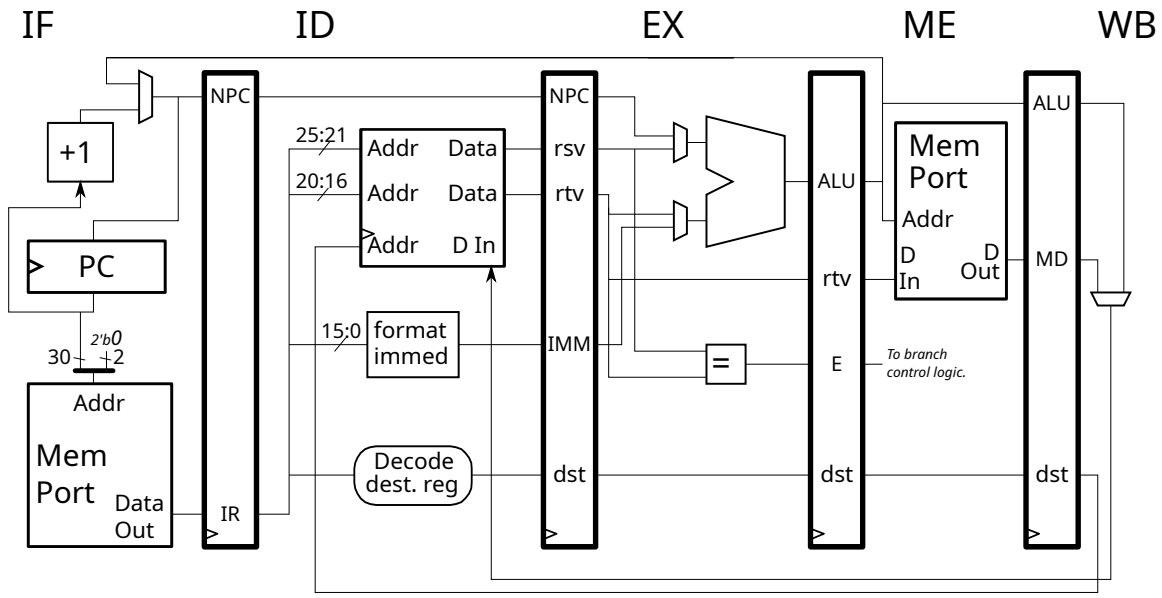
Problem 1	_____	(18 pts)
Problem 2	_____	(15 pts)
Problem 3	_____	(20 pts)
Problem 4	_____	(17 pts)
Problem 5	_____	(14 pts)
Problem 6	_____	(16 pts)
Exam Total	_____	(100 pts)

Alias _____

Good Luck! Thank you for your effort in EE 4720!

Staple This Side

Problem 1: (18 pts) Appearing below are MIPS implementations and similar but not identical code fragments. Show execution (a pipeline execution diagram) of the code on the accompanying implementations.

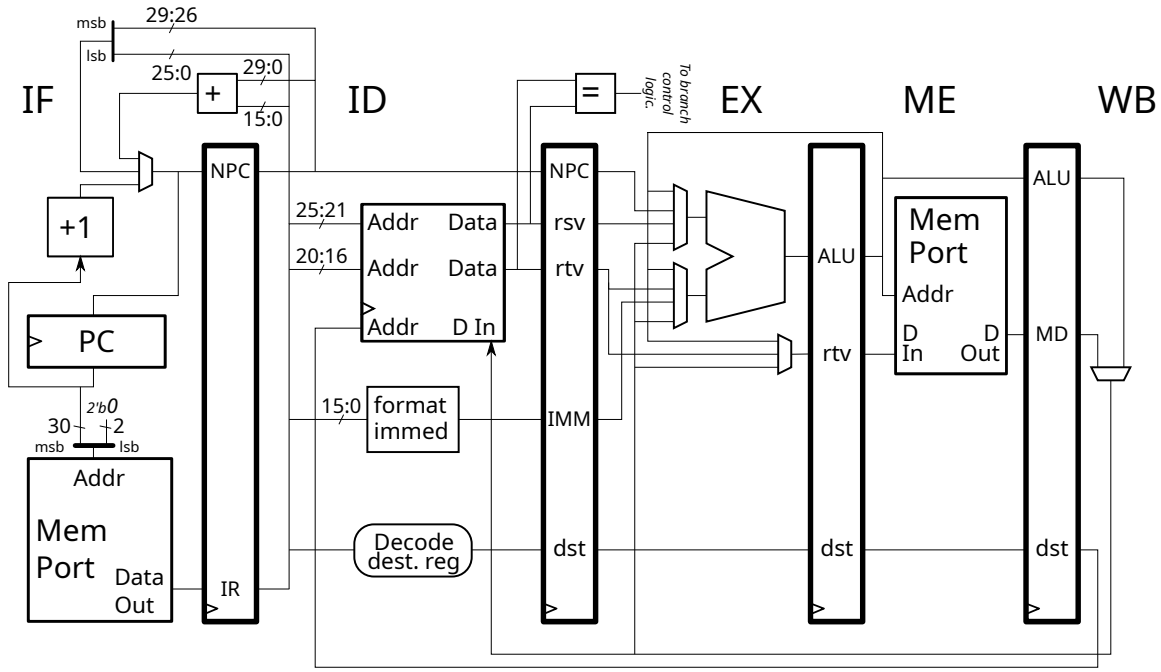


☐ Show execution of the code below on the implementation above ▷ with the branch taken. ☐ Check for dependencies, including ☐ those for the branch.

```
# Cycle            0    1    2    3    4
addi r1, r1, 4     IF   ID   EX   ME   WB


bne r1, r4, SKIP


lw r3, 0(r8)


add r2, r2, r3


sw r4, 0(r5)


SKIP:
andi r6, r1, 0xff


lw r5, 4(r8)


add  r7, r7, r6


or r8, r9, r10
```

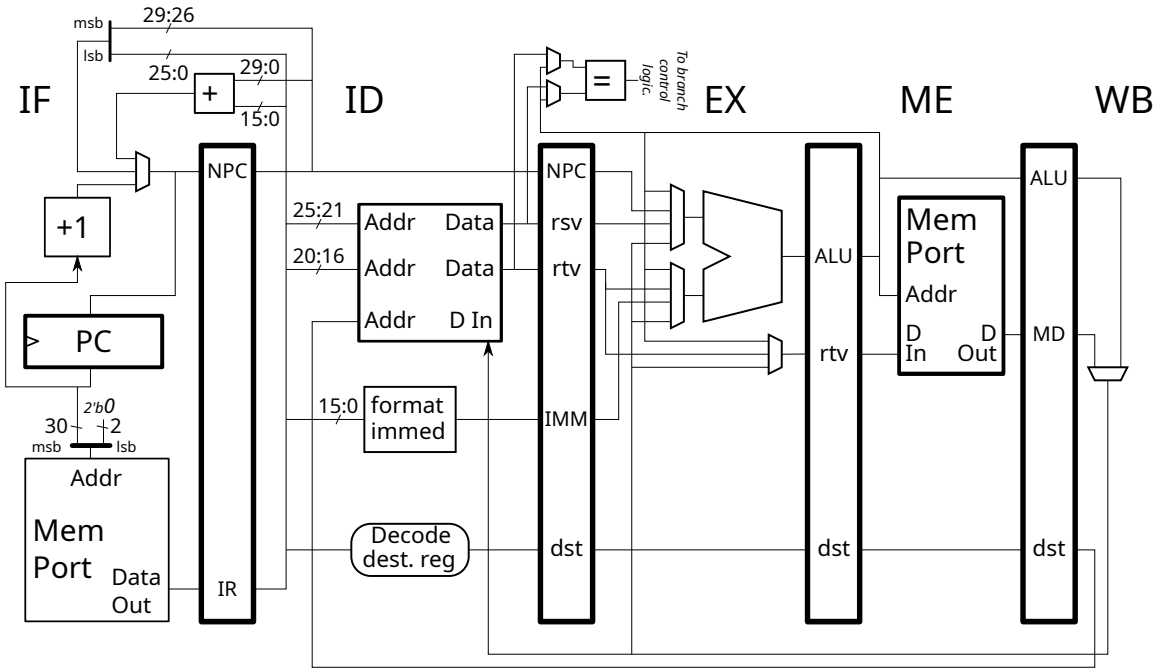

Staple This Side

Staple This Side



☐ Show execution of the code below on the implementation above ▷ with the branch taken. ☐ Check for dependencies, including ☐ those for the branch. Don't overlook store data paths.

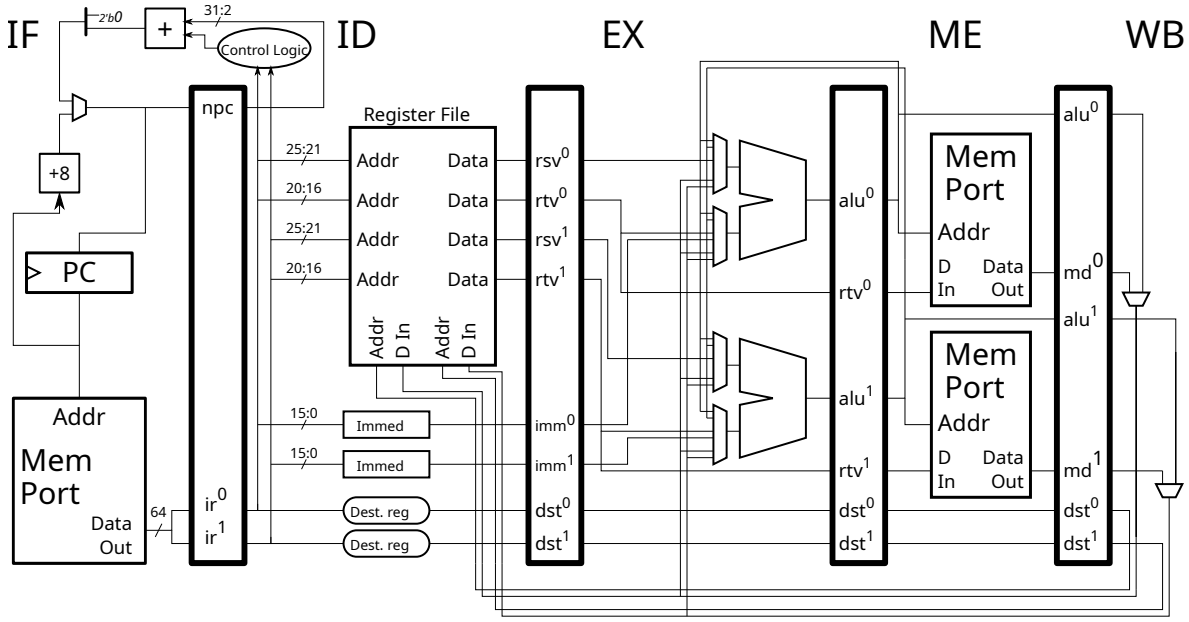
```
# Cycle            0    1    2    3    4
addi r1, r1, 4     IF   ID   EX   ME   WB


bne r1, r4, SKIP


lw r3, 0(r8)


add r2, r2, r3


sw r4, 0(r5)


SKIP:
lw r8, 0(r3)


andi r6, r8, 0xff


add  r7, r7, r6


sw   r7, 4(r8)
```



☐ Show execution of the code below on the implementation above ▷ with the branch taken. ☐ Check for dependencies, including ☐ those for the branch. ☐ Don't overlook the new branch condition data paths.

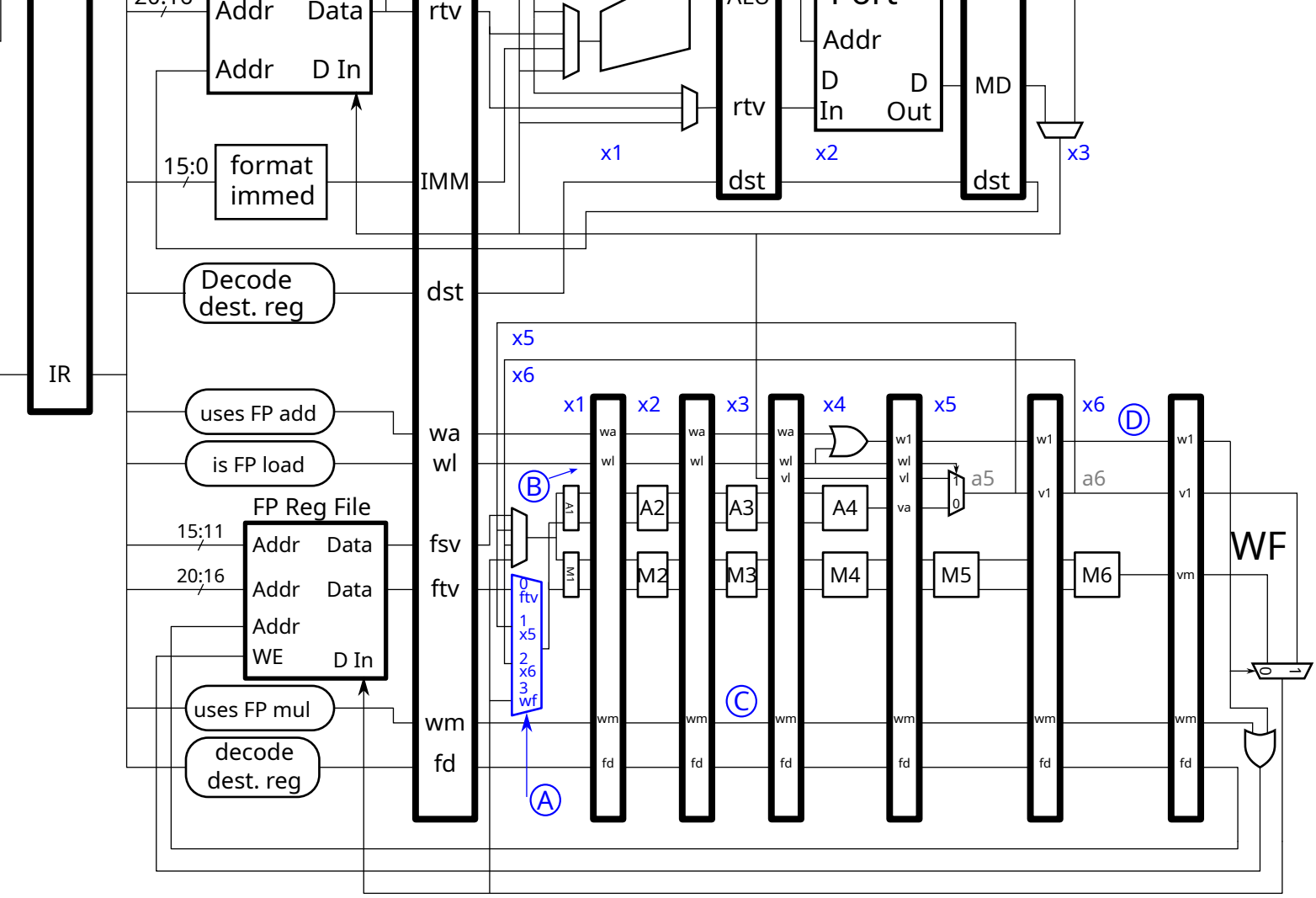
```
# Cycle            0    1    2    3    4
addi r1, r1, 4     IF   ID   EX   ME   WB


bne r1, r4, SKIP


xori r9, r8, 9


add r2, r2, r3


sw r4, 0(r5)


SKIP:
add r3, r3, r10


lw r8, 0(r3)


sw r8, 4(r3)
```


Staple This Side



☐ Show execution of the code below on the superscalar implementation above ▷ with the branch taken. ▷ As usual, fetch groups are not aligned. ☐ Check for dependencies. ☐ Don't overlook store data paths.

```
# Cycle            0    1    2    3    4
bne r1, r4, SKIP   IF   ID   EX   ME   WB

lw r3, 0(r8)


add r2, r2, r3


sw r4, 0(r5)


xor r9, r10, r11


slt r12, r13, r14


SKIP:
 addi r3, r3, 1


 andi r6, r3, 0xff


 add  r7, r7, r6


 sw   r7, 4(r8)
```



Problem 2: (15 pts) Appearing above is the FP pipeline and parts of the integer pipeline from the solution to Homework 4, which has a path for `lwc1` instructions.

(*a*) Complete the execution of the code fragments below.

☐ Complete the execution. ☐ Pay close attention to the number of stalls that are needed. ☐ Check for dependencies.

```
# Cycle            0    1    2    3    4    5    6    7    8
lwc1 f1, 0(r1)     IF   ID   EX   ME   x3   x4   x5   x6   WF


add.s f2, f1, f3
```

☐ Complete the execution. ☐ There will be stalls. ☐ Check for dependencies.

```
# Cycle            0    1    2    3    4    5    6    7    8
mul.s f1, f2, f3   IF   ID   M1   M2   M3   M4   M5   M6   WF


add.s f4, f5, f6


add.s f7, f1, f4
```


Staple This Side

(*b*) Appearing below is an execution on the FP pipeline showing stages for all instructions. Labels `x1`, `x2`, etc. are used for stages beyond `ID`, where `x1` can refer to `EX`, `A1`, or `M1`, depending on the instruction. The rows labeled `A`, `B`, `C`, and `D`, show values on the circled blue labels in the diagram. (There is a larger diagram on the next page.)

☐ Fill in the instructions mnemonic (one of `mul.s`, `add.s`, `lwc1`) of the first six instructions. All six can be determined.

☐ Fill in the destination registers of the first six instructions that can be determined, leave others blank.

```
# Cycle            0   1   2   3   4   5   6   7   8   9   10  11  12  13  14  15  16  17

                   IF  ID  x1  x2  x3  x4  x5  x6  WF

                       IF  ID  x1  x2  x3  x4  x5  x6  WF

                           IF  ID  x1  x2  x3  x4  x5  x6  WF

# Cycle            0   1   2   3   4   5   6   7   8   9   10  11  12  13  14  15  16  17

                               IF  ID  x1  x2  x3  x4  x5  x6  WF

                                   IF  ID  x1  x2  x3  x4  x5  x6  WF

                                       IF  ID  x1  x2  x3  x4  x5  x6  WF

# Cycle            0   1   2   3   4   5   6   7   8   9   10  11  12  13  14  15  16  17
add.s f10, f11, f1                       IF  ID  x1  x2  x3  x4  x5  x6  WF
add.s f12, f13, f2                           IF  ID  x1  x2  x3  x4  x5  x6  WF
# Cycle            0   1   2   3   4   5   6   7   8   9   10  11  12  13  14  15  16  17
mul.s f14, f15, f3                               IF  ID  x1  x2  x3  x4  x5  x6  WF
mul.s f16, f17, f4                                   IF  ID  x1  x2  x3  x4  x5  x6  WF
# Cycle            0   1   2   3   4   5   6   7   8   9   10  11  12  13  14  15  16  17
A                                            ..  2   0   1   3   ..
B                        0   1   0   0   0   1   ..
# Cycle            0   1   2   3   4   5   6   7   8   9   10  11  12  13  14  15  16  17
C                            1   0   0   1   0   0   ..
D                                        0   1   1   0   1   1   ..
# Cycle            0   1   2   3   4   5   6   7   8   9   10  11  12  13  14  15  16  17
```

Problem 3: (20 pts) This is the control logic problem! On the facing page is the FP pipeline from the Homework 4 solution adapted for this exam. To make space, parts of the integer pipeline are off the page. In the lower part of the diagram is the control logic that generates `sftv`, the `ftv` bypass multiplexor select signal.

(*a*) The control logic individually detects the need for the four mux inputs, `ftv`, `WF`, `x6`, and `x5`. The connections to the two blue OR gates are missing.

☐ Connect the four signals, `ftv`, `WF`, `x6`, `x5`, to the blue OR gates so that `sftv` is correct.

☐ Properly connect `sftv` to the mux select signal.

☐ Don't overdo it, just draw wires.

(*b*) Notice that there are four unconnected green wires.

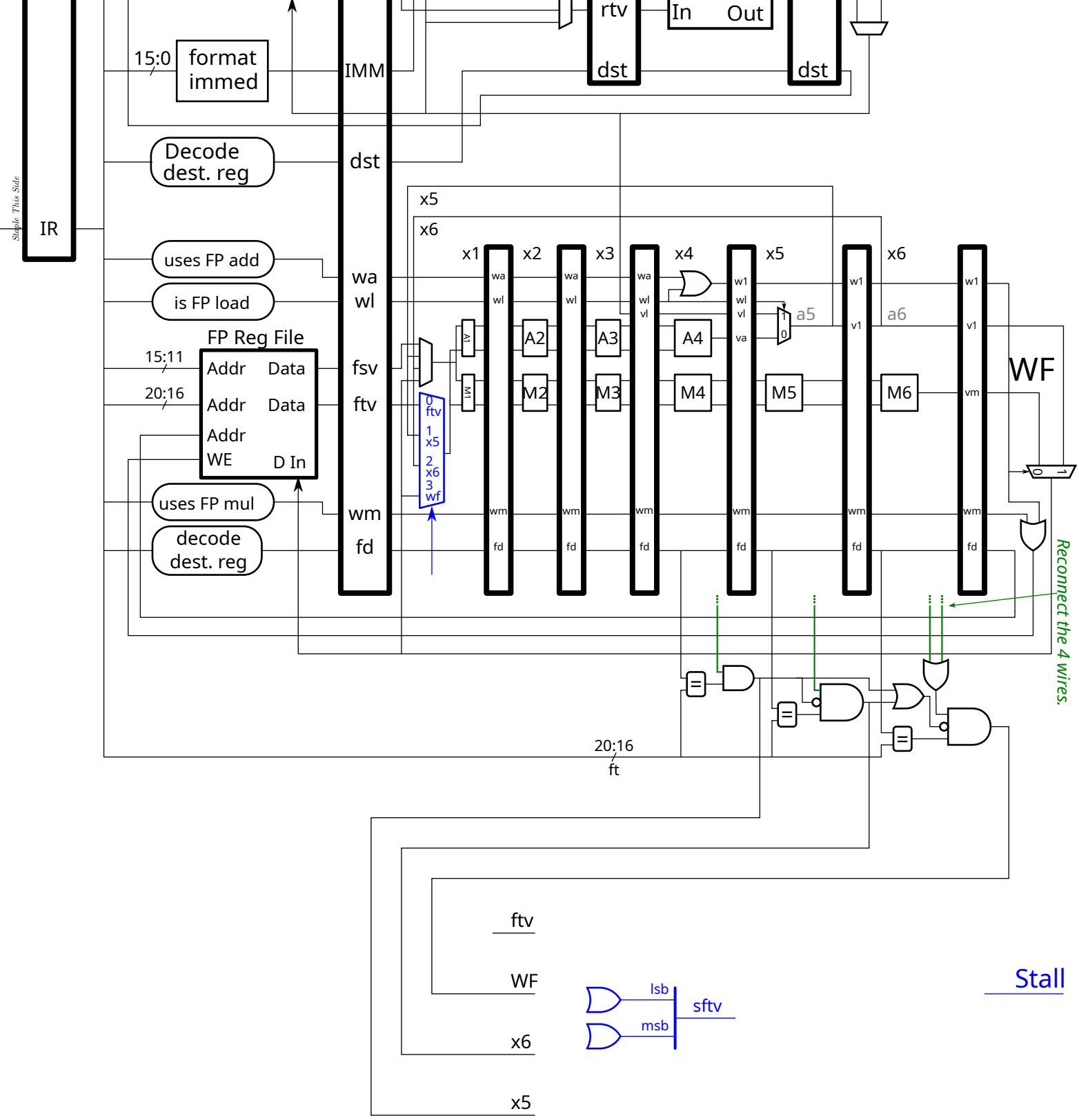
☐ Reconnect the four green wires so that the control logic works properly.

(*c*) (Most of the credit is for this part.) Add logic to set `Stall` to `1` for an unbypassable `mul.s/add.s` dependence such as the one below. Instructions should stall when they are in `ID`. Consider only `ft` sources and the destinations of instructions in stages `x4` and later.

```
# Cycle          0  1  2  3  4  5  6  7  8  9  10 11 12 13 14
mul.s F1, f2, f3 IF ID M1 M2 M3 M4 M5 M6 WF
nop                 IF ID EX ME WB
nop                    IF ID EX ME WB
nop                       IF ID EX ME WB
add.s f4, f5, F1             IF ID ----> A1 A2 A3 A4 a5 a6 WF
# Cycle          0  1  2  3  4  5  6  7  8  9  10 11 12 13 14
```

☐ Set `Stall` to `1` for unbypassable `mul.s/add.s` dependencies of the `add.s ft` source on instructions destinations starting at `x4`.

Staple This Side



Problem 4: (17 pts) Answer the following branch prediction questions.

(*a*) Code producing the branch pattern shown below is to run on several systems (**not** the one on the facing page), each with a different branch predictor. *Commas are used* to emphasize the repeating pattern.

```
B1:  N  N  N  T  T  T  T  T,  N  N  N  T  T  T  T  T,  N  N  N  T  T  T  T  T,
```

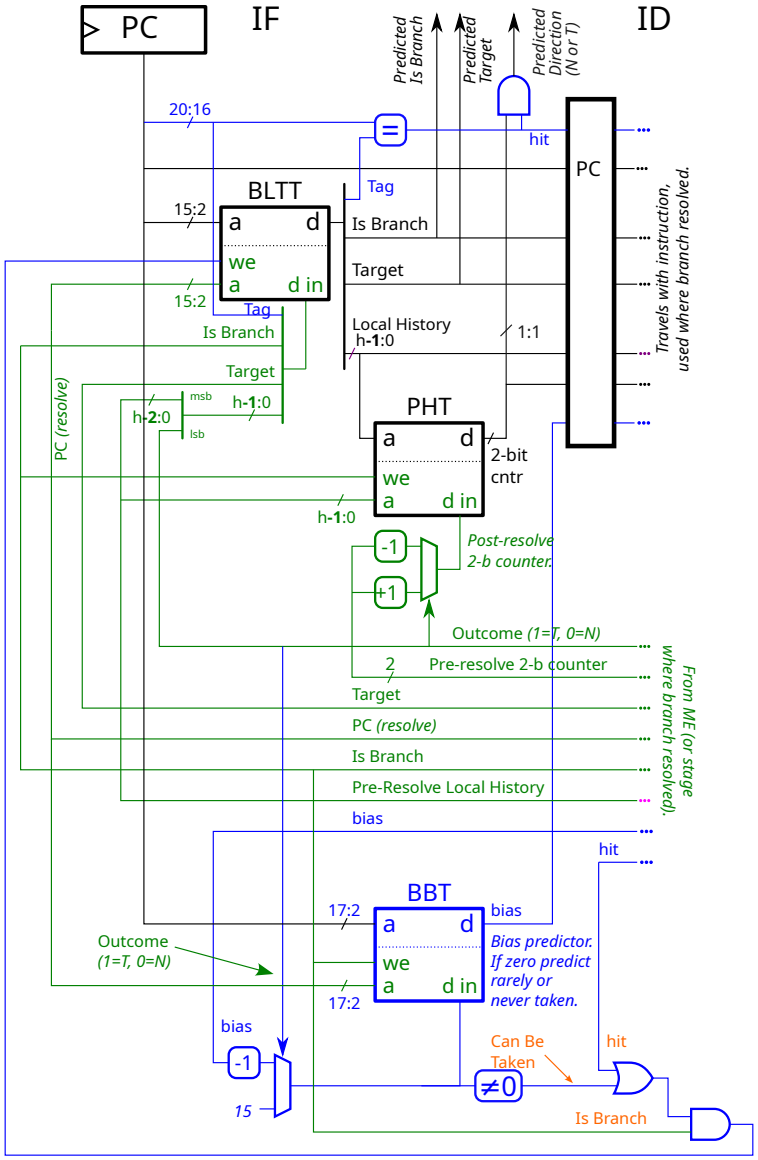
☐ What is the accuracy of the bimodal predictor on branch `B1`? ☐ Be sure to base the accuracy on a warmed-up 2-bit counter and repeating pattern.

☐ What is the accuracy of a 3-outcome local predictor on branch `B1`? ☐ Show the local histories that will be mispredicted. ☐ Indicate the number of times each local history is mispredicted and correctly predicted. (Don't show local histories that are always predicted correctly.)

☐ Find a branch pattern for `B2` that will induce two more mispredictions of `B1` on the 3-outcome local predictor. ☐ Indicate which positions of `B1` are mispredicted due to `B2`. ▷ Branches `B1` and `B2` execute the same number of times. ☐ Pay attention to branch frequency.

```
     1  2  3  4  5  6  7  8,  1  2  3  4  5  6  7  8,  1  2  3  4  5  6  7  8, <- Position
B1:  N  N  N  T  T  T  T  T,  N  N  N  T  T  T  T  T,  N  N  N  T  T  T  T  T,

B2:
```

☐ What is the minimum local history size for which branch `B1` is predicted with 100% accuracy ignoring `B2`.



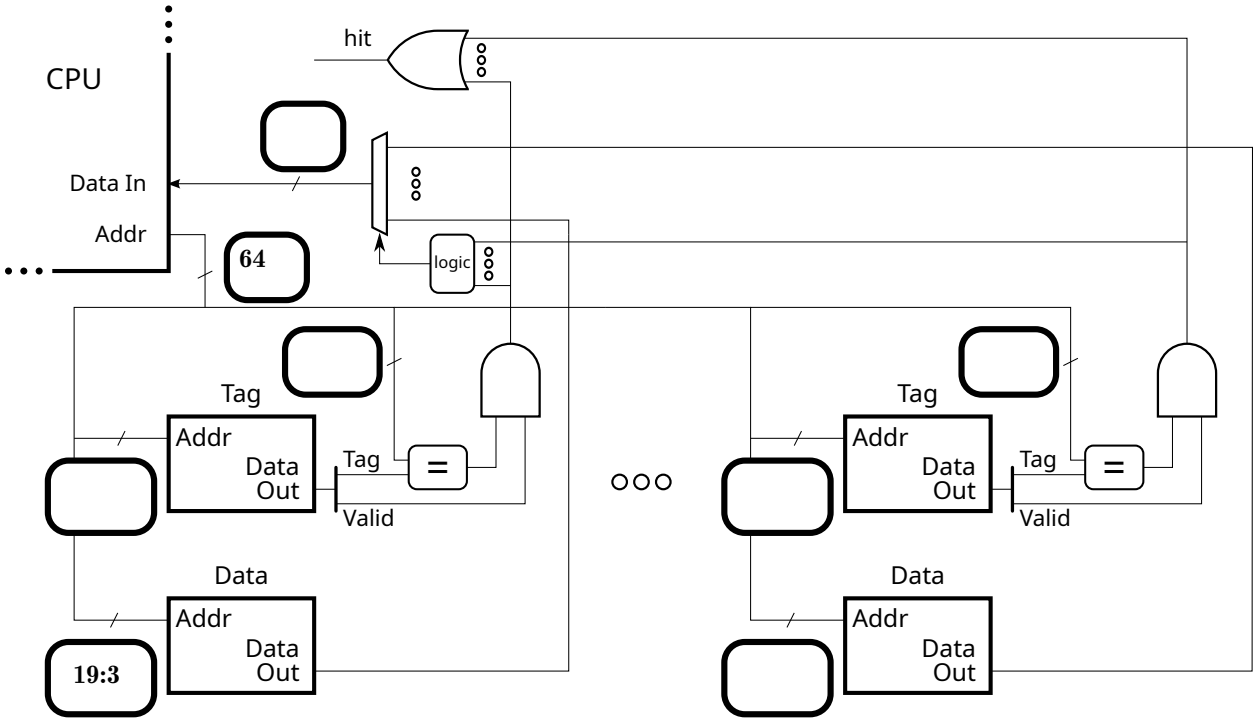
(*b*) Appearing above is the solution to Homework 7 with some wires in the lower part of the diagram labeled in orange.

☐ What would be mispredicted if `hit`, and `Is Branch` were stuck at the value `1` (rather than the values from `ME`)? ☐ Explain.

☐ What might be mispredicted if `Can Be Taken` were stuck at `1`, which would not be mispredicted if it had its intended value? ☐ Explain.

Problem 5: (14 pts) The diagram below is for a 16 MiB, x-way set-associative cache with a line size of 64 B. Helpful facts: 16 MiB $= 2^{24}$ B, $64 = 2^6$.

(a) Answer the following, formulæ are fine as long as they consist of grade-time constants.



☐ Show the number of bits or bit ranges in the unfilled boxes above.

☐ Complete the address bit categorization below. Label the sections appropriately. (Index, Offset, Tag.)

Address: | | | |

☐ What is the associativity of the cache:

Staple This Side

Staple This Side

(*b*) The code in the problem below runs on a cache with a line size of 64 bytes (characters). The code fragment starts with the cache cold (empty); consider only accesses to the array.

```
double sum = 0;
bf16 *a = 0x2000000; // sizeof(bf16) == 2
int ILIMIT = 1 << 14;    // = 2^14

for ( int i=0; i<ILIMIT; i++ ) sum += a[ i ];
```

☐ What is the hit ratio running the code above? Show a formula and briefly justify.

(*c*) The tag used in a cache serves the same purpose as a tag used some branch predictors. But in a cache the tag must be of a particular size while in a branch predictor its size can be chosen to balance cost and performance.

☐ Why can't the tag in a cache be made smaller to save cost?

Staple This Side

Problem 6: (16 pts) Answer each question below.

(*a*) Results for the SPECcpu suite can be reported several ways.

☐ What is the difference between SPECcpu base and peak tuning levels?

☐ Everyone agrees (say) SPECcpu base tuning results are useful. Provide an argument that results at SPECcpu peak tuning are not useful.

(*b*) Branches in more recent RISC ISAs lack a delay slot.

☐ Why is a delay slot useful in five stage pipelines?

☐ Why isn't a delay slot very useful in superscalar implementations?

(*c*) VLIW ISAs fetch in units of bundles.

☐ Briefly explain what a bundle is.

☐ RISC ISAs were motivated by pipelined implementations. What kind of implementation motivated VLIW ISAs?

(*d*) Compiler command line arguments can specify the ISA and implementation being compiled for.

☐ A compiler is told to compile for the wrong ISA, for example MIPS32 instead of RV-32i. What happens: ◯ *The code won't run.* ◯ *The code runs slowly.* ◯ *The code runs normally.* ☐ Explain.

☐ A compiler is told to compile for the wrong ISA version, for example MIPS32 instead of MIPS-I or RISC-V RV-32I instead of RISC-V RV-32IAM. What happens: ◯ *The code won't run.* ◯ *The code may not run.* ◯ *The code will run but slowly.* ◯ *The code runs normally.* ☐ Explain.

☐ A compiler is told to compile for the wrong implementation. What happens: ◯ *The code won't run.* ◯ *The code runs slowly.* ◯ *The code runs normally.* ☐ Explain.