Name _____ *Formatted For 2-Sided Printing*

Staple This Side

Computer Architecture
LSU EE 4720
Final Examination

Monday, 9 May 2022 10:00-12:00 CDT

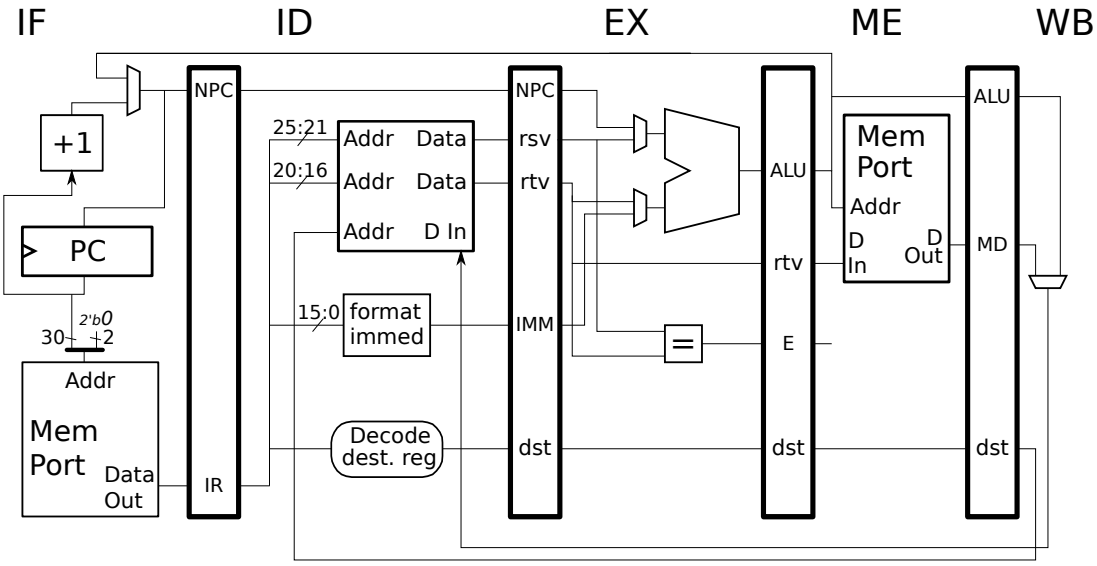
Problem 1	_____	(20 pts)
Problem 2	_____	(20 pts)
Problem 3	_____	(20 pts)
Problem 4	_____	(20 pts)
Problem 5	_____	(20 pts)
Exam Total	_____	(100 pts)

Alias _____

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

Staple This Side

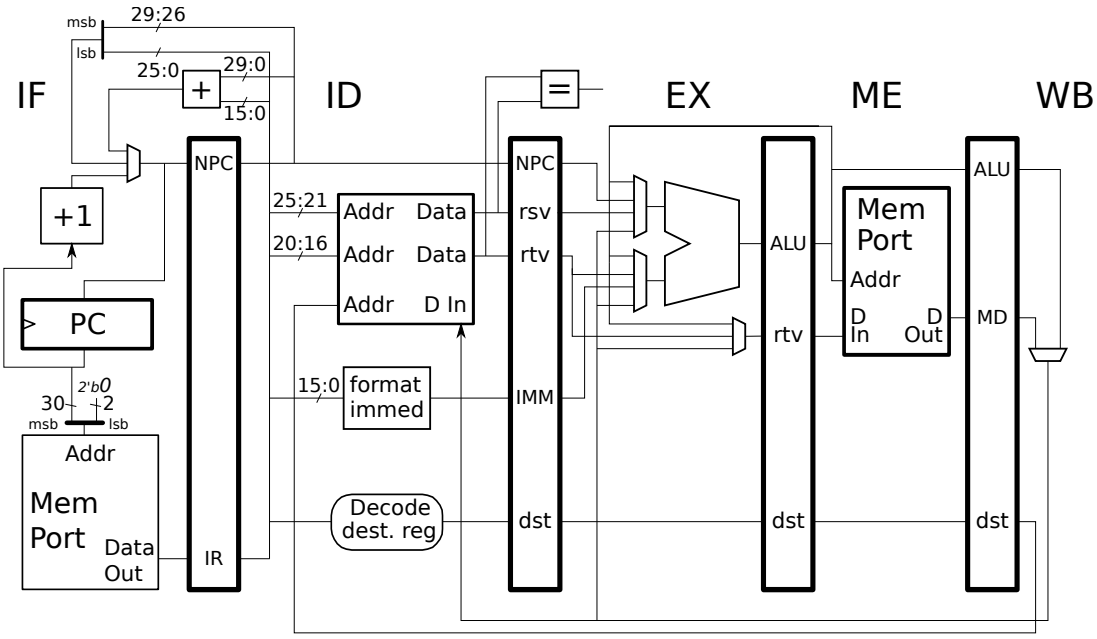
Problem 1: (20 pts) Show the execution of the code fragments on the following implementations for enough iterations to determine the instruction throughput (IPC). **As always, base the behavior of branches and the availability of bypasses on the implementations. Also, don't forget that MIPS branches have a delay slot.** Sorry for yelling, but I hate it when students miss things.



☐ Show execution and ☐ determine instruction throughput (IPC) based on a large number of iterations.

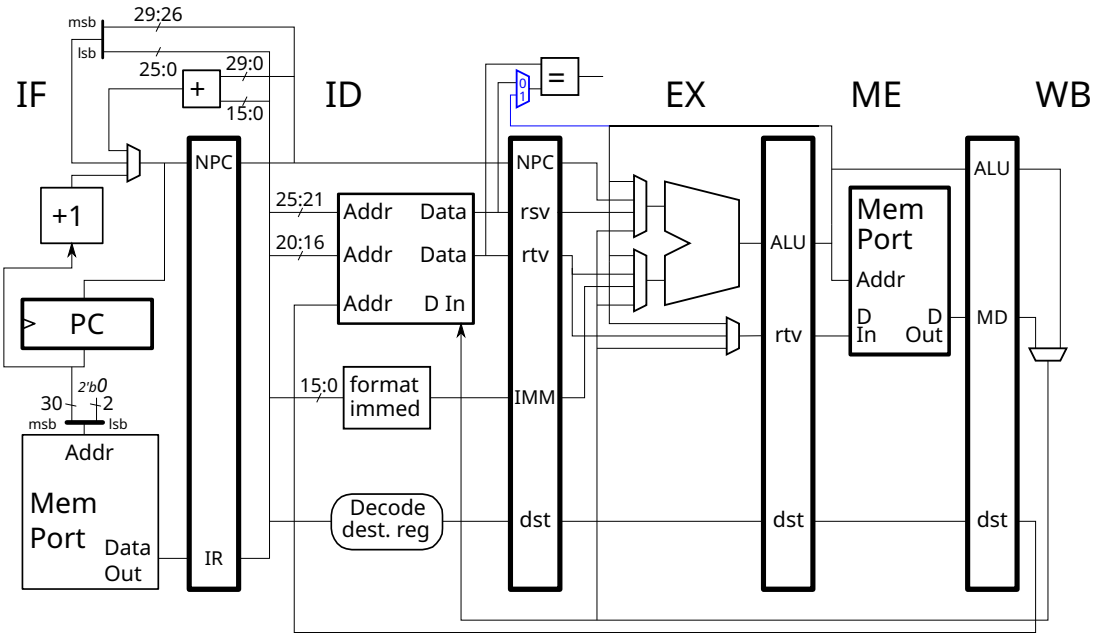
```
LOOP:
 bne r1, r2, LOOP

 addi r1, r1, 4

 xor r5, r6, r7

 sub r8, r9, r10
```


Staple This Side



☐ Show execution and ☐ determine instruction throughput (IPC) based on a large number of iterations.

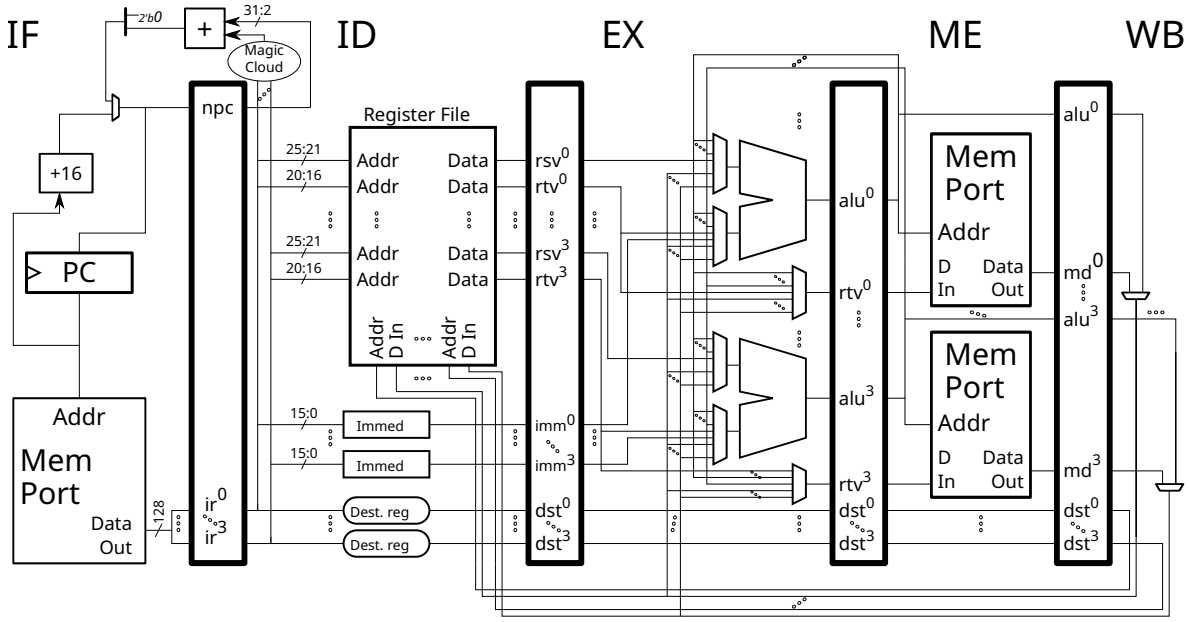
```
LOOP:
 bne r1, r2, LOOP

 addi r1, r1, 4

 xor r5, r6, r7

 sub r8, r9, r10
```



☐ Show execution and ☐ determine instruction throughput (IPC) based on a large number of iterations.

```
LOOP:
 bne r1, r2, LOOP

 addi r1, r1, 4

 xor r5, r6, r7

 sub r8, r9, r10
```

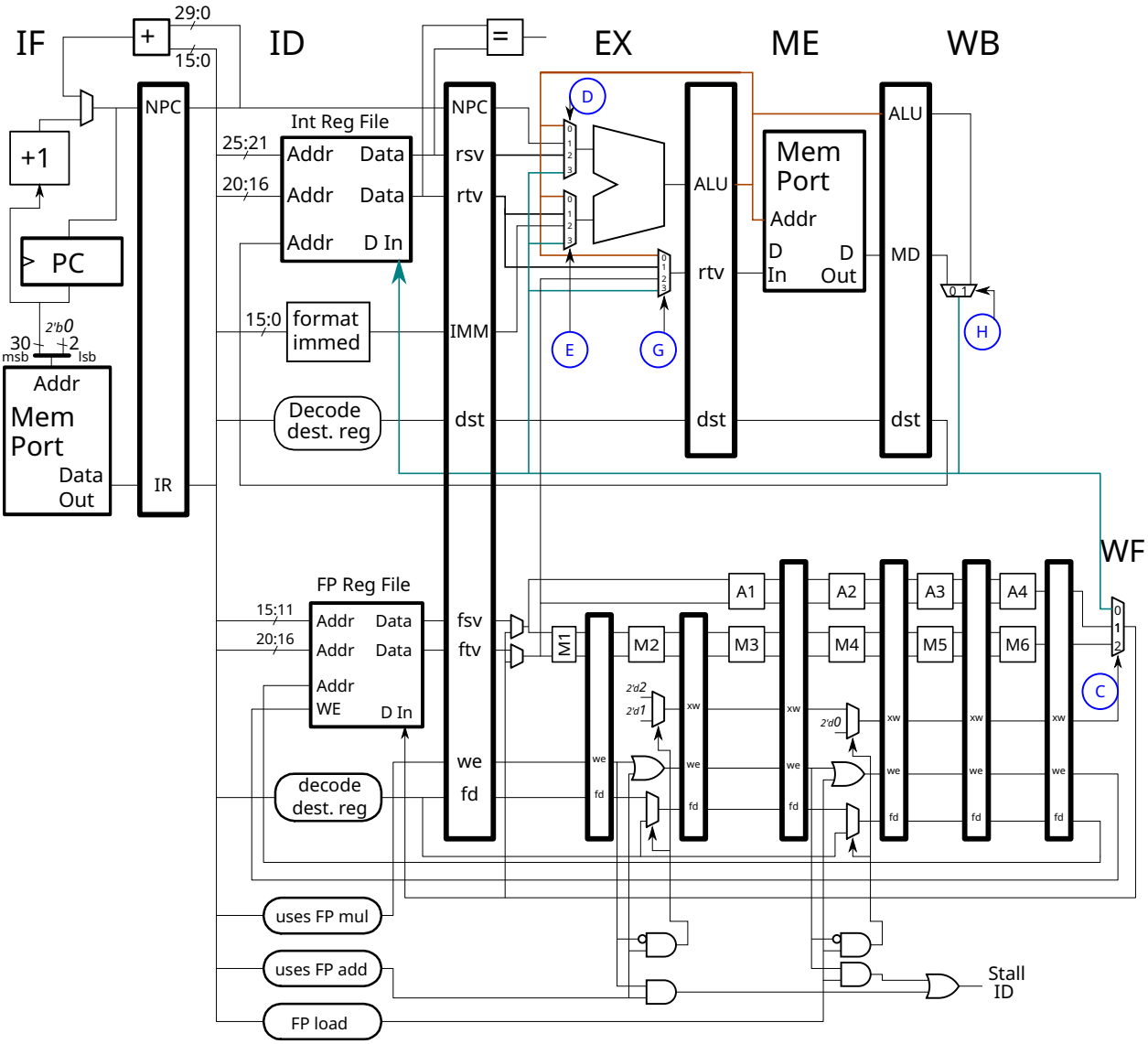

Staple This Side



☐ Show execution until the fetch of the `lw r1` in the second iteration. ☐ Show instruction throughput (IPC) assuming a large number of iterations.

```
LOOP:
 lw r1, 0(r2)

 lw r3, 4(r2)

 add r4, r1, r3

 sw r4, 0(r6)

 sub r5, r1, r3

 sw r5, 4(r6)

 addi r6, r6, 4

 bne r2, r9, LOOP

 addi r2, r2, 8
```

Problem 2: (20 pts) Appearing below is our MIPS implementation with a floating-point pipeline. The select inputs of some multiplexors are labeled with a letter. Also, the inputs to some multiplexors have been colored to make them easier to follow.



Staple This Side

Staple This Side

☐ Show the values on the select inputs (D, E, and H) expected from the execution shown below. ☐ Leave a signal **blank** if it does not affect execution.

```
# Cycle         0    1    2    3    4    5    6

D:

E:

H:
```

```
# Cycle         0    1    2    3    4    5    6
add R3, r5, r6  IF   ID   EX   ME   WB
addi r2, R3, 4       IF   ID   EX   ME   WB
lw r1, 0(R3)              IF   ID   EX   ME   WB
# Cycle         0    1    2    3    4    5    6
```

☐ Show instructions that could have produced the select input (D,E,G, and H) values shown below. ☐ Take dependencies into account when choosing register numbers.

```
# Cycle                 0    1    2    3    4    5    6
D:                                2    2    3
E:                                2    1    2
G:                                          0
H:                                          0    1
# Cycle                 0    1    2    3    4    5    6
                        IF   ID   EX   ME   WB

                             IF   ID   EX   ME   WB

                                  IF   ID   EX   ME   WB

# Cycle                 0    1    2    3    4    5    6
```

☐ Show the values on the select inputs expected from the execution shown below. ☐ Leave an input **blank** if it does not affect execution.

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

G:

H:

C:
```

```
# Cycle         0    1    2    3    4    5    6    7    8    9
lwc1 f1, 0(r5)  IF   ID   EX   ME   WF
swc1 f2, 0(r7)       IF   ID   EX   ME   WB
mtc1 f3, r8               IF   ID   EX   ME   WF
add.s f4, f5, f6               IF   ID   A1   A2   A3   A4   WF
# Cycle         0    1    2    3    4    5    6    7    8    9
```

Staple This Side

Problem 3: (20 pts) Appearing to the right is the *early writeback* 2-way superscalar implementation from the 2021 Final Exam and 2022 Homework 6. Recall that in this implementation if slot 1 contains a load instruction then slot 1 writes back when it reaches `WB` but slot 0 writes back early, in `ME/MW`. If slot 1 does not contain a load instruction slot 0 writes back when it reaches `WB` and slot 1 writes back in `ME/MW`. This is illustrated in the execution below.

```
# Cycle          0  1  2  3  4  5  6
add r2, r3, r4   IF ID EX ME WB    # Slot 0 uses WB since slot 1 isn't a load.
and r1, r5, r6   IF ID EX MW
or r10, r9, r7      IF ID EX MW    # Slot 0 uses MW since slot 1 is a load.
lw r6, 8(r11)       IF ID EX ME WB
# Cycle          0  1  2  3  4  5  6
```

(*a*) Consider the execution of the code below:

```
# Cycle          0  1  2  3  4  5
add r2, r3, r4   IF ID EX ME WB
sub r1, r2, r5   IF ID -> EX MW
```

The `sub` stalls because it needs to wait for `r2` from the `add`. Add control logic to generate a stall when slot 1 depends on slot 0. The output of rs src and rt src are `1` when the slot-1 instruction uses the `rs` and `rt` registers as sources. Use these in your solution.

☐ Add hardware to generate a stall (see the big OR gate) when slot 1 depends on slot 0.

(*b*) In the first code fragment below the `lw r5` stalls, but because the `lw` has a zero immediate it could have just used the value computed by the `add` instruction (since there is no need to add anything to it). In the second execution Mux A is used to perform a *lateral bypass* from the `add` to the `lw` during cycle 2, avoiding the stall.

Modify the control logic so that a lateral bypass will be performed when there is a zero-immediate load in slot 1 that depends on the slot-0 instruction. Other code, such as the examples at the top of this problem, should continue to work correctly.

```
# Cycle          0  1  2  3  4  5
add r2, r3, r4   IF ID EX ME WB
lw r5, 0(r2)     IF ID -> EX ME WB

# Cycle          0  1  2  3  4  5
add r2, r3, r4   IF ID EX MW      # add executes normally.
lw r5, 0(r2)     IF ID EX ME WB   # Lateral Bypass: lw uses slot-0 alu value.
```

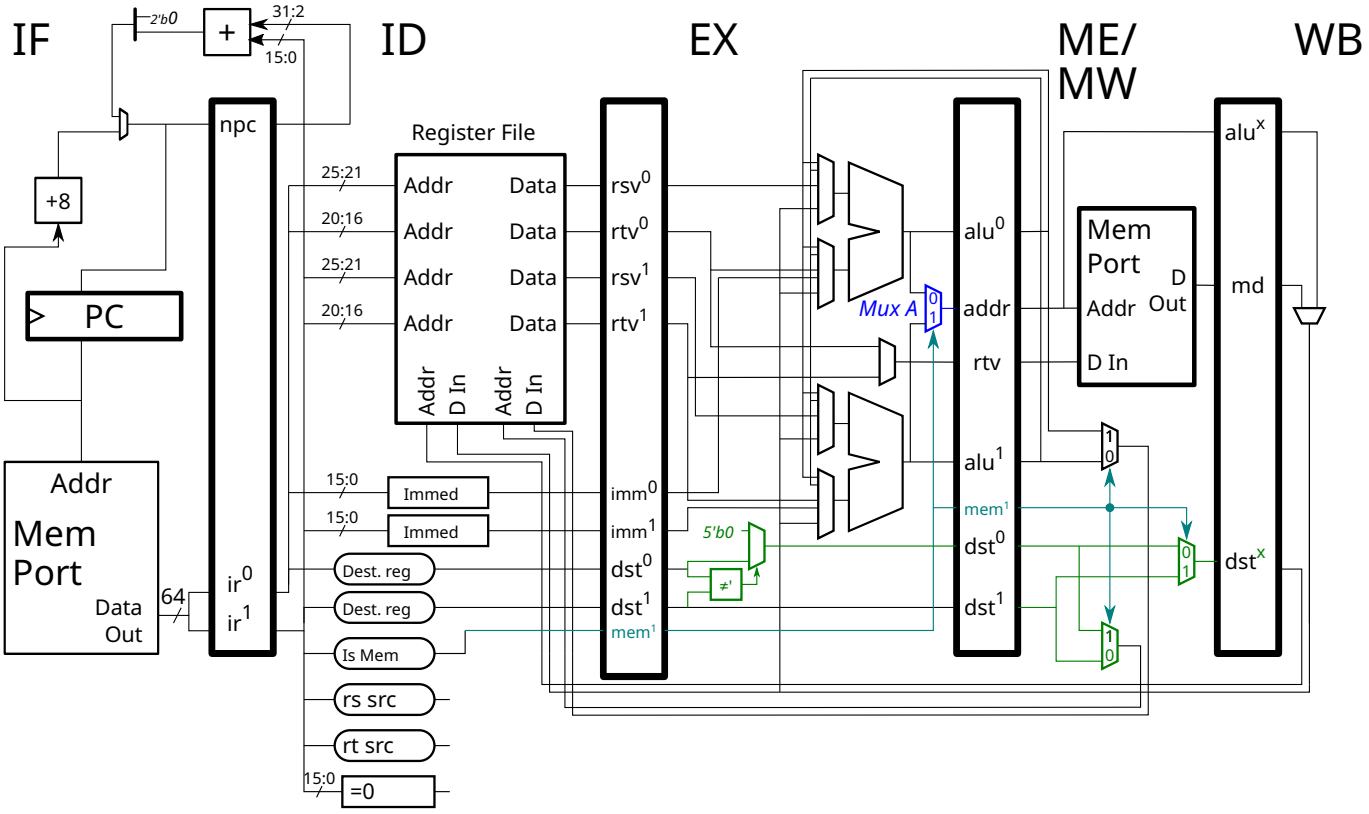
☐ Add logic to detect whether a lateral bypass is possible, and if so, suppress the stall from part a.

☐ Modify the control logic to implement a lateral bypass, in part using Mux A.

☐ Make sure that ☐ early writeback continues to work correctly in other cases and ☐ that the destination of the slot 0 and slot 1 instructions are written to the correct registers.

☐ As engineers always do, pay attention to cost and performance.

Staple This Side



Problem 4: (20 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. One system has a bimodal predictor and the other system has a local predictor with an 6-outcome local history.

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

```
B1:  T  N  N  T  T  N  T    T  N  N  T  T  N  T    T  N  N  T  T  N  T      <- Outcome

          All N's          All T's             All N's          All T's
      ---------------  --------------      ---------------  --------------  ...
B2:    N  N  ...  N  N  T  T  ... T  T      N  N  ...  N  N  T  T  ... T  T  <- Outcome
       1  2       7  8  9 10        16      1  2       7  8  9 10        16  <- Position
```

☐ What is the accuracy of the bimodal predictor on branch B1? ☐ Be sure to base the accuracy on a repeating pattern.

☐ What is the accuracy of the local predictor on B1 ignoring B2.

☐ What is the accuracy of the local predictor on B2 ignoring B1.

☐ What is the accuracy of the bimodal predictor on branch B2?

☐ What is the shortest history size for which the local history predictor is better than the bimodal predictor on branch B2?

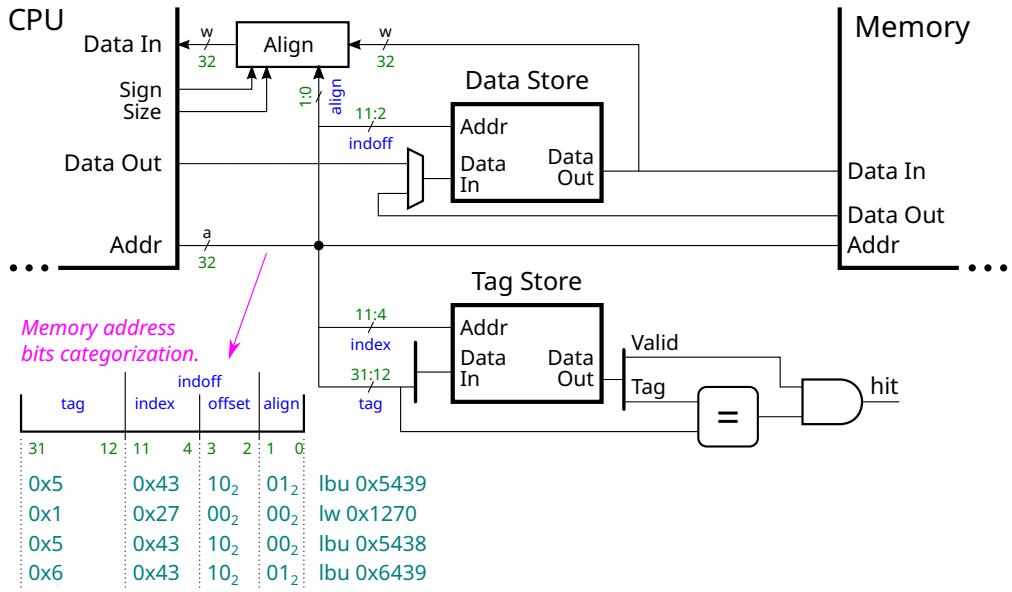
Staple This Side

This page intentionally left blank, except for this notice.

Staple This Side

Problem 5: (20 pts) Answer each question below.

(*a*) The diagram below shows a simple direct-mapped cache and the address bit categorization of four lookup addresses (`0x5439, 0x1270, ..`).



Two kinds of memory are used in the diagram above, fast/expensive and slow/cheap.

☐ On the diagram above show which blocks are fast and which blocks are slow.

Suppose that the cache is initially cold (there is nothing in the cache). Show the outcome, hit or miss, of each of the four lookups.

☐ Show outcome, hit or miss, on diagram above.

Find the addresses requested below.

☐ After the four lookups, what is the smallest address that will hit the cache.

☐ After these four lookups, what is the largest address that will hit the cache.

Staple This Side

(*b*) Show the encoding for the `beq` and `lw` as used in the code below. Be sure to include the immediate value.

```
 addi r6, r0, 10
 beq r2, r6, SKIP
 lw r1, 4(r3)
 add r1, r1, r5
SKIP:
 and r9, r9, r1
```

☐ Encoding of `beq`. ☐ Be sure to show a value for the immediate field.

☐ Encoding of `lw`. ☐ Be sure to show a value for the immediate field.

(*c*) Answer the following about ISA families.

☐ Which style of implementation are RISC ISAs designed for?

☐ Which style of implementation are VLIW ISAs designed for?

(*d*) Some early RISC ISAs omitted useful magnitude-comparison branch instructions such as `bgt r1, r2, TARG` in which the branch is taken if `r1 > r2`. As branch prediction became more common magnitude-comparison branch instructions were added to RISC ISAs. One might argue that with branch prediction the cost and performance impact of magnitude-comparison instructions was lower.

☐ Explain how the cost of implementing `bgt` is lower with branch prediction than without.

☐ Explain how the performance impact of implementing `bgt` is lower with branch prediction than without.

Staple This Side