Name _____ *Formatted For 2-Sided Printing*

Staple This Side

Computer Architecture
LSU EE 4720
Midterm Examination

Wednesday, 3 April 2024 9:30-10:20 CDT

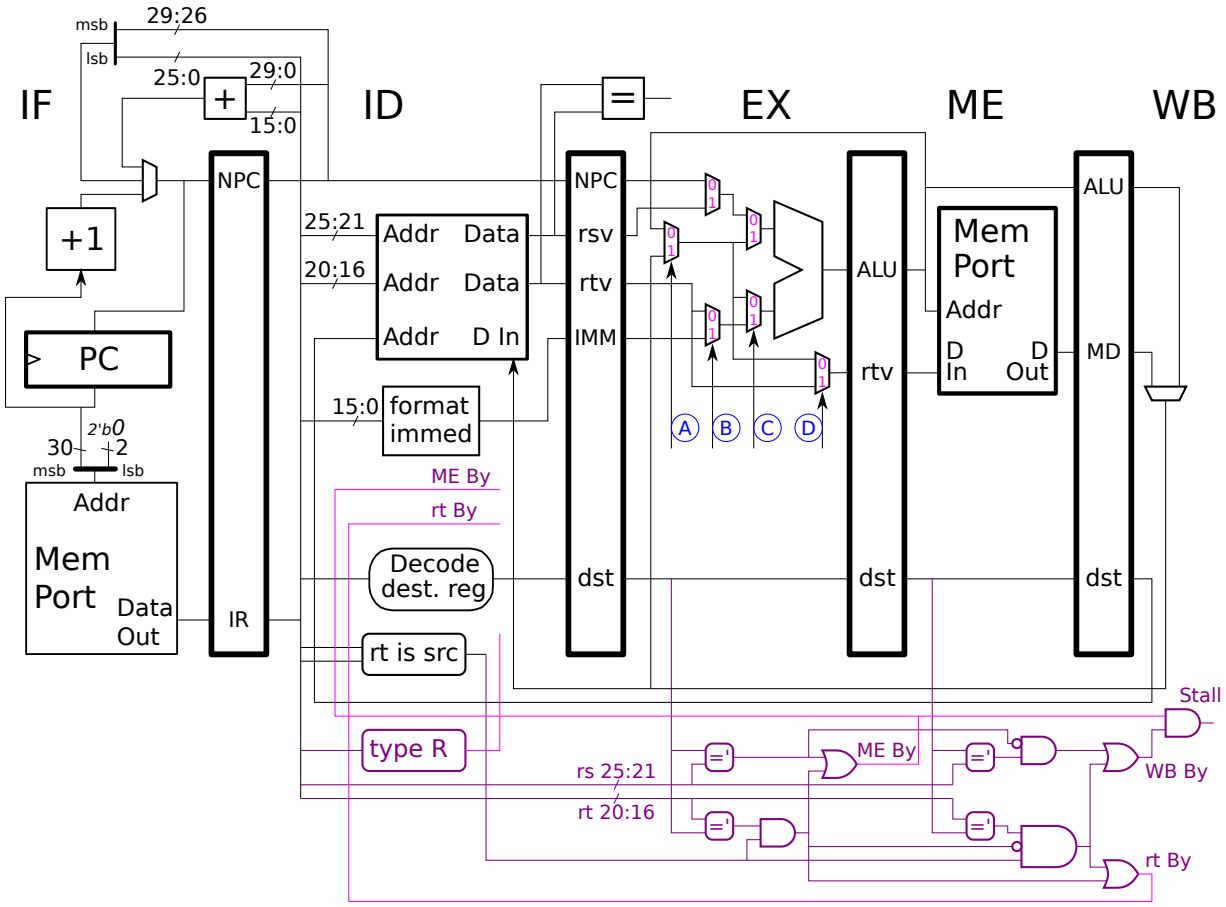
Problem 1	_____	(18 pts)
Problem 2	_____	(17 pts)
Problem 3	_____	(15 pts)
Problem 4	_____	(15 pts)
Problem 5	_____	(20 pts)
Problem 6	_____	(15 pts)
Exam Total	_____	(100 pts)

Alias _____

Good Luck!

Staple This Side

Problem 1: [18 pts] Appearing below is a **changed** version of the MIPS implementation appearing in Homework 3 and the 2020 midterm exam.



☐ In the table show the select signal values expected for the execution shown below. ☐ Use `X` for select signals that don't matter (that can be either `0` or `1`). ☐ **Don't forget** to check for dependencies

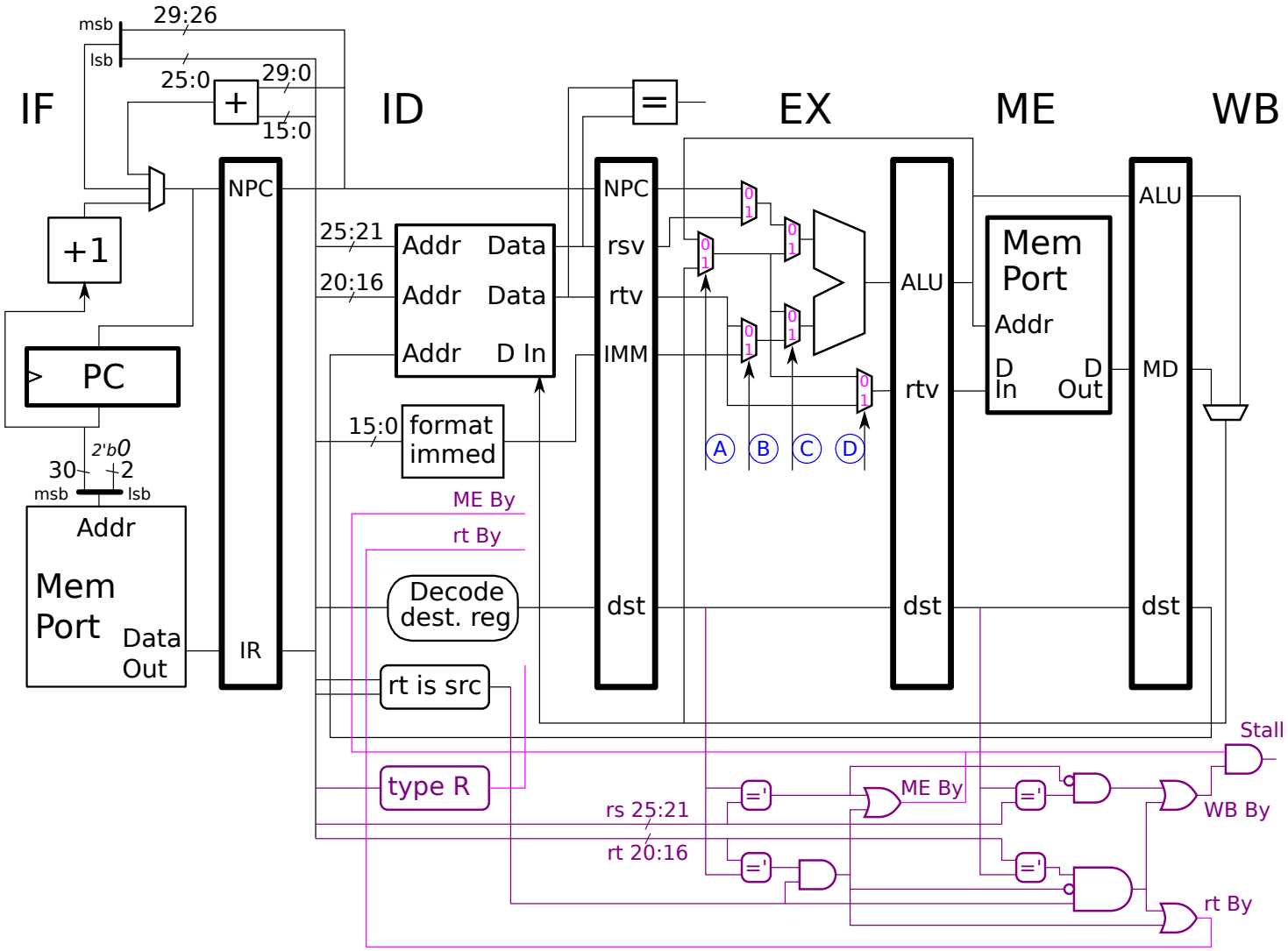
```
# Cycle          0     1     2     3     4     5     6     7
add r1, r2, r3   IF    ID    EX    ME    WB
ori r7, r1, 9          IF    ID    EX    ME    WB
sub r8, r9, r7               IF    ID    EX    ME    WB
sw r7, 5(r6)                       IF    ID    EX    ME    WB
# Cycle          0     1     2     3     4     5     6     7

A

B

# Cycle          0     1     2     3     4     5     6     7

C

D

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


Staple This Side

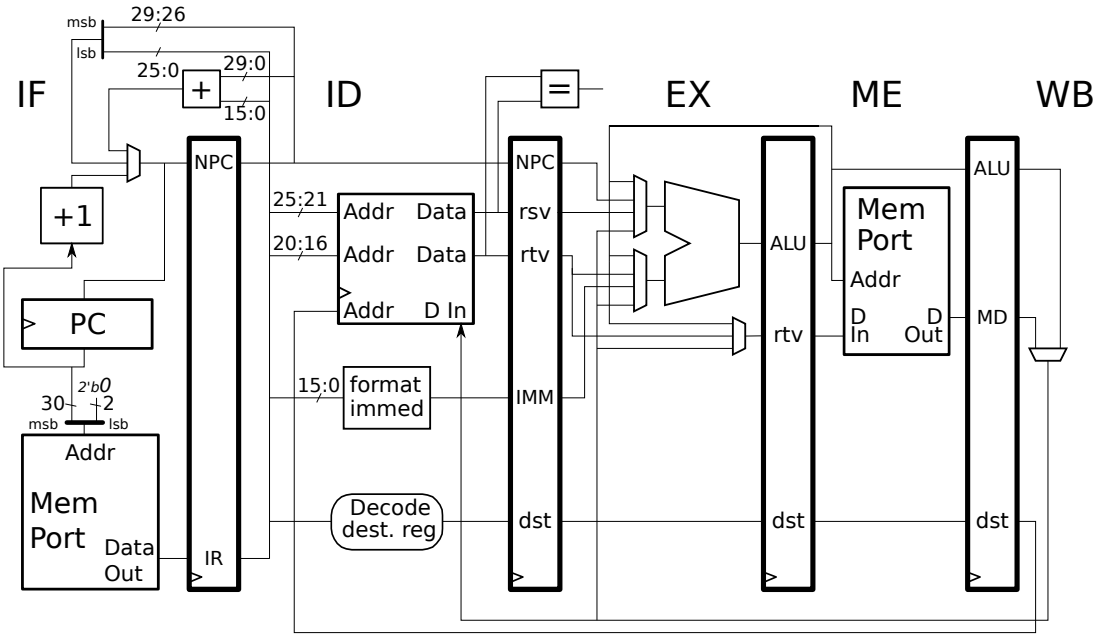
Problem 2: [17 pts] Appearing below is the implementation from the previous problem. It **is not identical** to the Homework 3 implementation. *See the last page of this exam for the Homework 3 Problem 3 solution.*

☐ Design the control logic for the A, B, C, and D select signals.

☐ Take advantage of existing logic, not much more logic is needed. ☐ Make sure that C works for the code fragment in the previous part. ☐ Don't forget that execution is pipelined.



Problem 3: [15 pts] Show the execution of the MIPS code fragments on the implementation.



☐ Show the execution of the fragment below with ☐ the branch taken. ☐ Pay close attention to branch behavior.

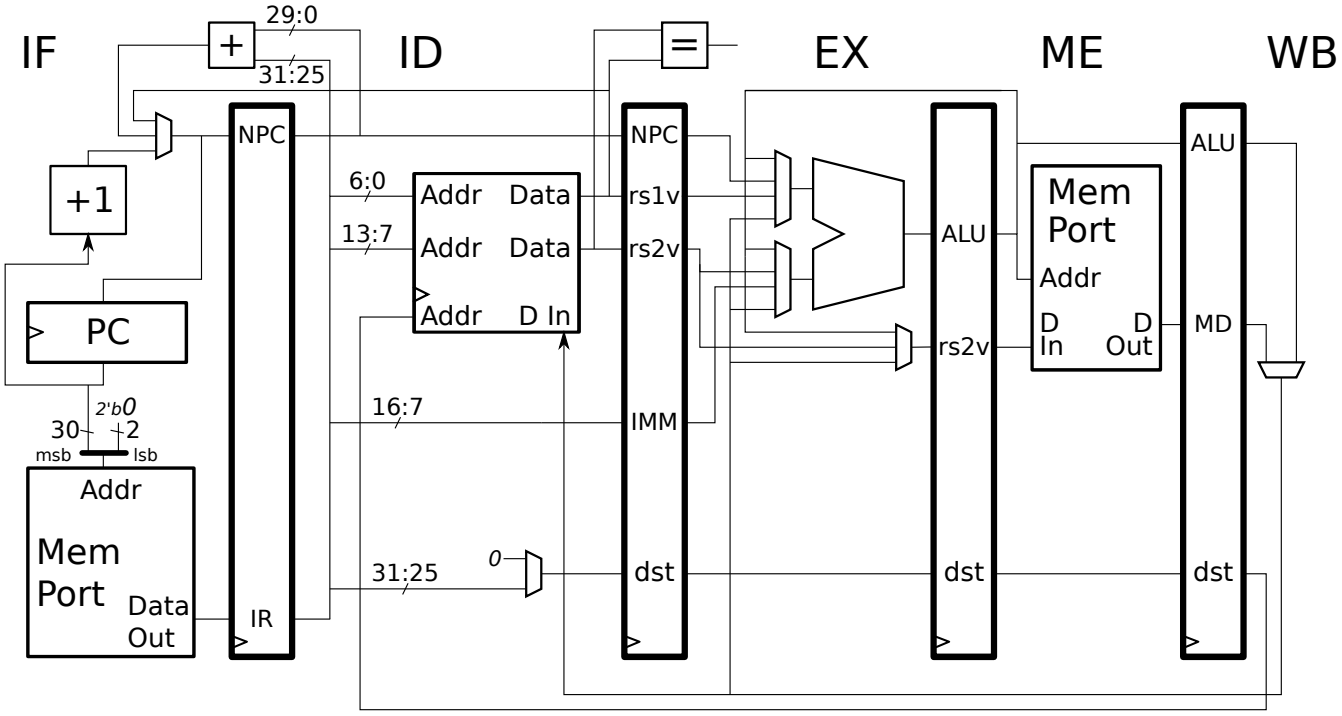
```
beq r1, r1, SKIPA

add r2, r3, r4

sub r5, r6, r7

ori r8, r9, 100

xori r10, r11, 101


SKIPA:
lw r12, 0(r14)
```

☐ Show the execution of the fragment below. ☐ Be sure to check for dependencies.

```
addi r5, r5, 4

lw r2, 0(r5)

add r1, r2, r3

sub r4, r1, r4
```


Staple This Side

Problem 4: [15 pts] Appearing below is the implementation of Another RISC ISA (ARI) and incomplete diagrams for the encoding of its MIPS-like R and I formats.



☐ How many registers does ARI have?

☐ What is ARI's immediate size?

☐ Why is it possible to implement an instruction like `lw r1, 4(r2)` but not an instruction like `sw r1, 4(r2)` on the implementation above?

☐ In the spaces below complete ☐ ARI R and ☐ ARI I instruction formats consistent with the implementation. ☐ Be sure to show the opcode field and any opcode extensions that are needed.

ARI R: ☐ `add r1, r2, r3`

31 0

ARI I: ☐ `addi r4, r5, 6`

31 0

Problem 5: [20 pts] Answer each question below.

(*a*) Show the contents of the destination register after each MIPS I instruction below executes.

```
#           Initially r1 = 0x12345678
sll r2, r1, 16
#                 ☐  r2 =

srl r3, r1, 16
#                 ☐  r3 =

or  r4, r2, r3
#                 ☐  r4 =
```

(*b*) Given the MIPS code below, why might execution never reach the `or` instruction?

```
lw $a0, 0($t0)
jal SOME_CONVENTIONAL_STANDARD_LIBRARY_FUNCTION
addi r31, r31, -8
or $s1, $s1, $v0
```

☐ The `or` instruction won't be reached because:

☐ What will happen instead is:

Staple This Side

(*c*) Register `r9` holds the address of the middle of a large memory allocation, and so all the MIPS `lb` instructions below execute with no problem. Not so for the `lw` instructions.

```
lb r11, 0(r9)  # Will execute correctly.
lb r12, 5(r9)  # Will execute correctly.
lb r13, 10(r9) # Will execute correctly.
lb r14, 15(r9) # Will execute correctly.

lw r1, 0(r9)
lw r2, 5(r9)
lw r3, 10(r9)
lw r4, 15(r9)
```

☐ Why won't the rest of the MIPS code execute to completion?

☐ What are the maximum and minimum number of `lw` instructions that will execute before an error occurs, and ☐ briefly explain how the maximum and minimum number are determined by the exact value of `r9`.

(*d*) Simplify MIPS the code fragment below.

```
lbu r1, 0(r10)
lbu r2, 1(r10)
sll r1, r1, 8
or r1, r1, r2
sh r1, 2(r10)
# Note: r1 and r2 not used again.
```

☐ Simplify the code fragment ☐ without changing what it does.

Staple This Side

Problem 6: [15 pts] Answer each question below.

(*a*) In class we described three families of ISAs, CISC, VLIW, and RISC.

☐ How do VLIW ISAs differ from both RISC and CISC ISAs?

(*b*) Identify the ISA family of the following ISAs:

MIPS: ◯ *CISC* ◯ *VLIW* ◯ *RISC*

Arm A64: ◯ *CISC* ◯ *VLIW* ◯ *RISC*

Itanium: ◯ *CISC* ◯ *VLIW* ◯ *RISC*

Intel 64 /IA-32: ◯ *CISC* ◯ *VLIW* ◯ *RISC*

VAX: ◯ *CISC* ◯ *VLIW* ◯ *RISC*

(*c*) The statement below is wrong.

CISC ISAs can have large immediate values, but at the cost of having large instructions. That is why programs in CISC ISAs are large compared to those in RISC ISAs.

☐ What is correct in the statement above?

☐ What is wrong in the statement above?

Appearing below is part of the solution to Homework 3 Problem 3. It may be helpful in solving Problem 2 in this exam.