Name _____

Computer Architecture
EE 4720
Midterm Examination

Monday, 19 March 2018, 9:30–10:20 CDT

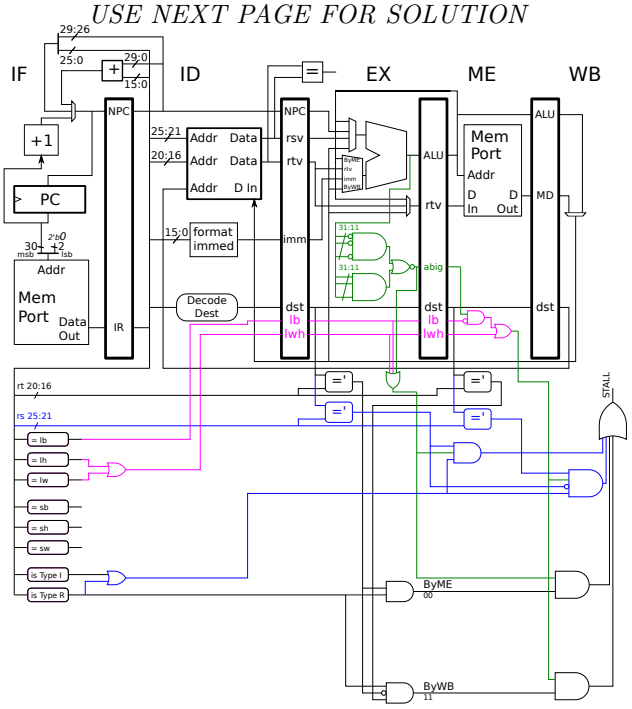
Problem 1	_____	(25 pts)
Problem 2	_____	(20 pts)
Problem 3	_____	(15 pts)
Problem 4	_____	(40 pts)
Exam Total	_____	(100 pts)

Alias _____

Good Luck!

Problem 1: [25 pts] Appearing below is the solution to Homework 4, in which additional control logic is added for the 12-bit bypass paths. The illustrated hardware generates stall signals for a load/use dependence and for cases in which the value that needs to be bypassed is unknown or too wide for the 12-bit bypass paths.

USE NEXT PAGE FOR SOLUTION



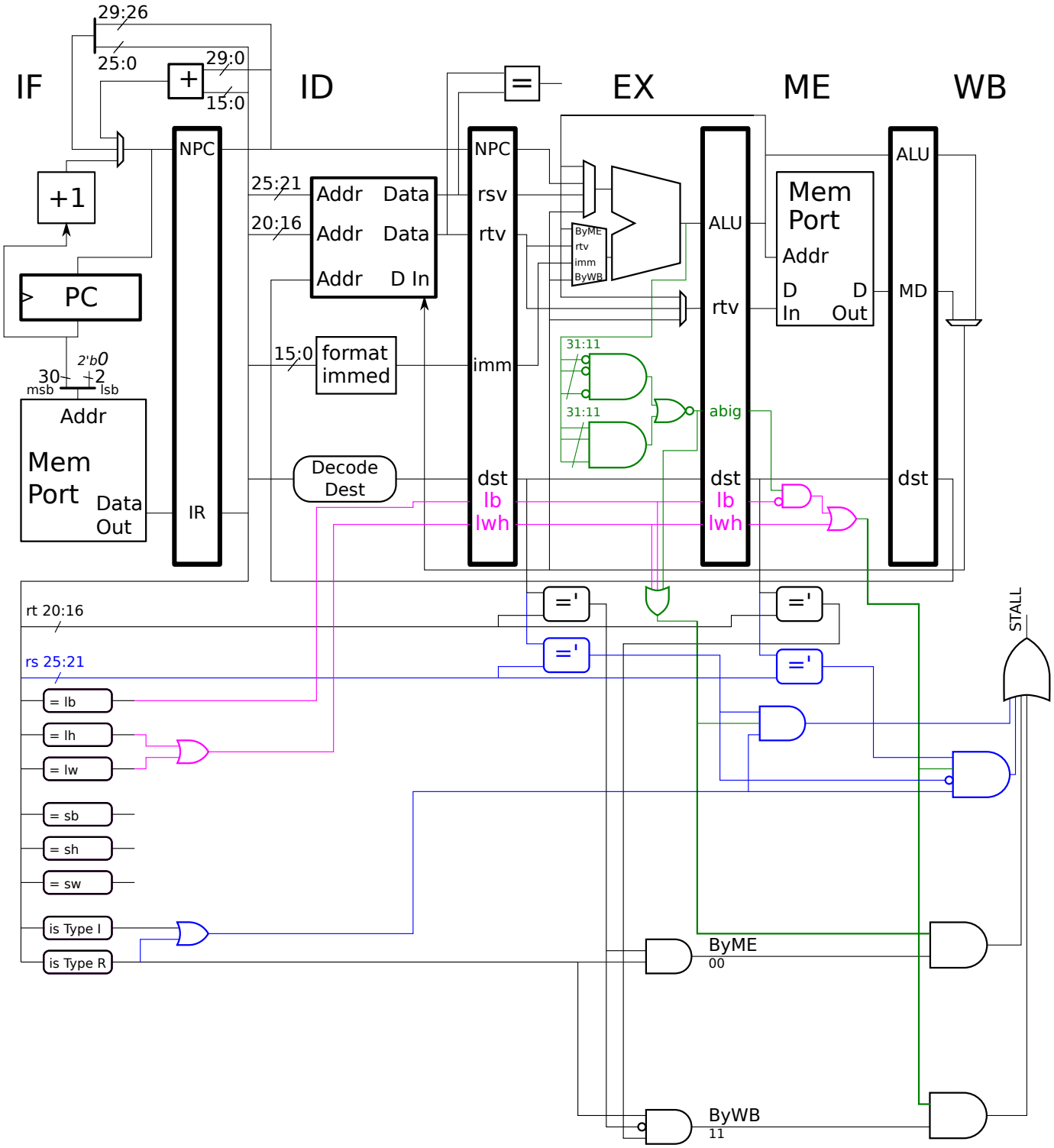
USE NEXT PAGE FOR SOLUTION

(*a*) Suppose, on further consideration, it was decided that full-sized, 32-bit bypass paths to the upper ALU mux were needed. Elsewhere 12-bit bypass paths would be retained. Modify the hardware so that a stall signal would no longer be generated for such too-big values to the upper ALU mux. Do so **without** affecting stalls for the 12-bit bypass paths to the lower ALU mux and without affecting stalls for load/use dependencies.

```
# Should not stall anymore, regardless of size of r1.
add $r1, $r2, $r3
lw $r4, 0($r1)

# Should still stall.
lw $r5, 0($r6)
addi $r7, $r5, 9

# Should still stall if r1 too big for 12-bit bypasses.
add $r1, $r2, $r3
sub $r5, $r6, $r1
```

☐ Remove hardware generating stall due to values too large for upper ALU bypass paths.

☐ **Do not** change stalls for load/use cases and bypasses to lower ALU mux.

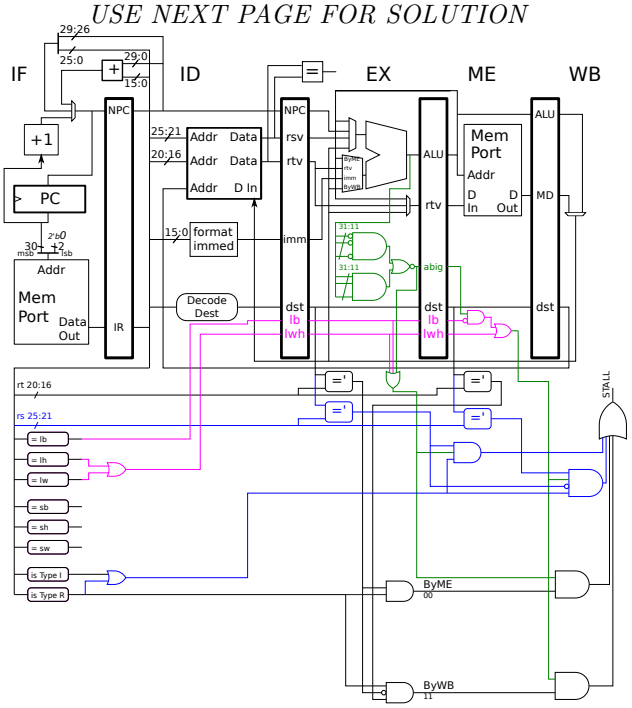


Problem 1, continued:

(*b*) Suppose that the bypass paths to the `EX`-stage `rtv` mux were also just 12 bits wide. Modify the control logic on the next page so that a stall signal would be generated when appropriate for store instructions.

Note that a `lb` produces a value small enough for the 12-bit bypass paths and that the store value needed by a `sb` is always small enough for the 12-bit bypass paths. See the examples below.

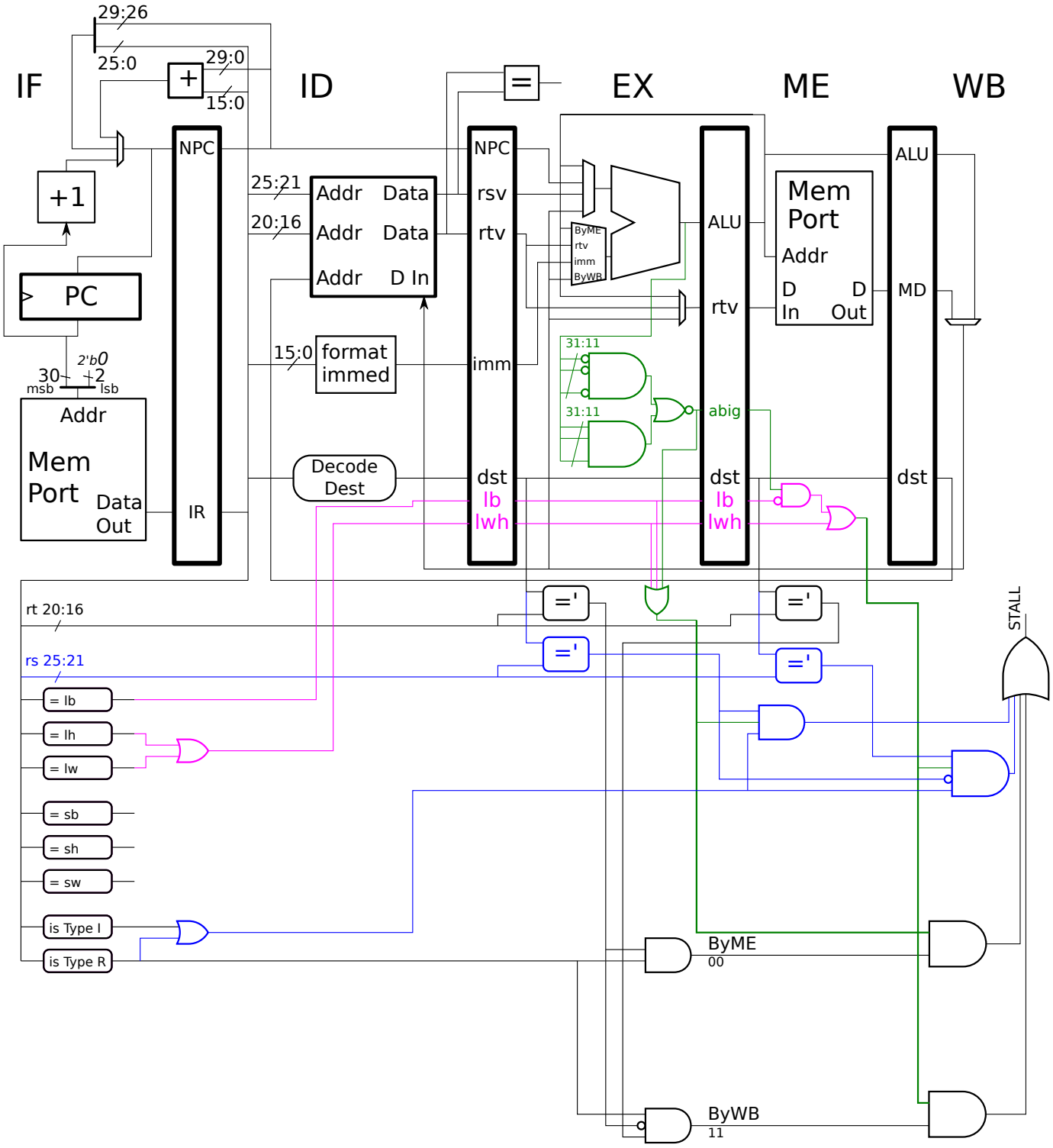
USE NEXT PAGE FOR SOLUTION



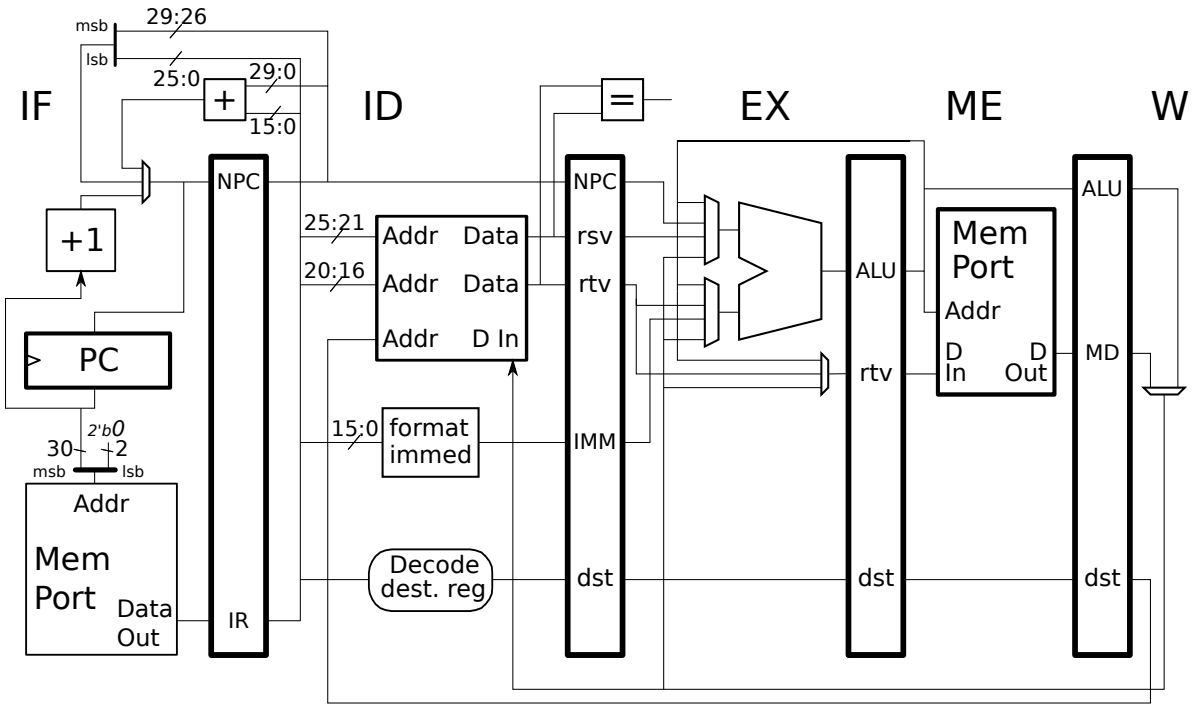
USE NEXT PAGE FOR SOLUTION

```
# sb should not stall for this dependency.
add $r1, $r2, $r3  IF ID EX ME WB
sb $r1, 0($r4)        IF ID EX ME WB

# sw should stall for one cycle.
lb $r1, 0($r5)     IF ID EX ME WB
sw $r1, 0($r4)        IF ID -> EX ME WB

# sh should stall only if r1 is too large.
add $r1, $r2, $r3  IF ID EX ME WB
sh $r1, 0($r4)        IF ID ----> EX ME WB
```

☐ Generate stall for unbypassable value from prior instructions to store value (not store address). ☐ Consider store size and any load size. (See examples on previous page.)



Problem 2: [20 pts] Answer the following questions about two versions of our bypassed MIPS implementation.



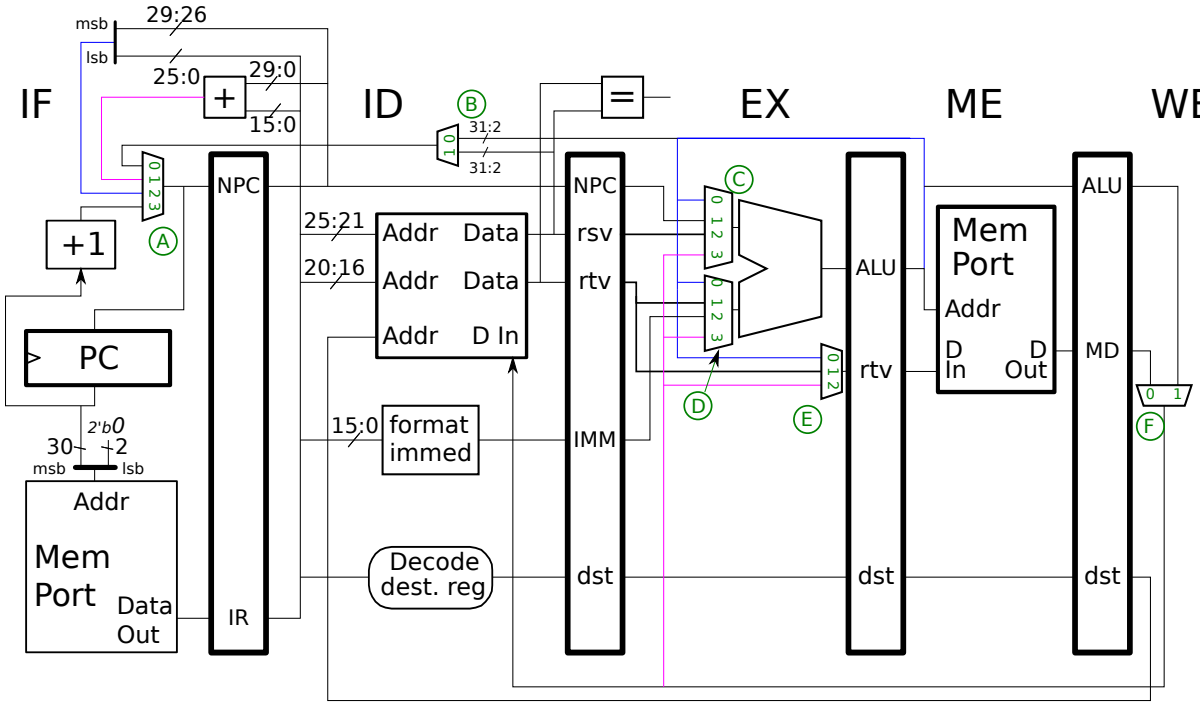
(*a*) Show the execution of the code below on the implementation illustrated above when the branch is taken.

☐ Show Execution.

☐ Check the code for dependencies.

```
lw r2, 0(r4)

addi r1, r2, 3

bne  r1, r3  TARG

ori r1, r9, 10

andi r11, r1, 14

TARG:
xor r20, r11, r22
```

(*b*) Each mux in the implementation below is labeled with a circled letter, and mux inputs are numbered. Some wires are colored to make them easier to follow. Write code sequences that use the mux inputs as requested below. Some code sequences may consist of a single instruction.



☑ Use C0 to carry `r1`, elsewhere `r1` not used.

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

☐ Use D3 to carry `r1`, elsewhere `r1` not used.

☐ Use E0 to carry `r1`, elsewhere `r1` not used.

☐ Use A1.

☐ Use B0.

☐ Use C1.

Problem 3: [15 pts] The floating point code fragment below computes `f2 = f3 * f0 + f1`, where `f3` is a call argument and `f0` and `f1` are loaded from a table. A total of eight instructions are used to load the constants into `f0` and `f1`. Re-write the code so that fewer instructions are used. It is possible to load both registers using a total of two instructions.

☐ Load constants into `f0` and `f1` using two or three instructions.

☐ The constants must be loaded from the table.

```
        .data
famous_constants: # 0x10010300
        .float 2.718282818
        .float 3.141592654
        .text
pie:
        # Load f0 with first element of table.
        lui $t0, 0x1001
        ori $t0, $t0, 0x300
        lw $t1, 0($t0)
        mtc1 $t1, $f0

        # Load f1 with second element of table.
        lui $t0, 0x1001
        ori $t0, $t0, 0x304
        lw $t1, 0($t0)
        mtc1 $t1, $f1

        # Don't modify the code below this line.
        mul.s $f2, $f3, $f0
        add.s $f2, $f2, $f1
```

Problem 4: [40 pts] Answer each question below.

(*a*) In class we said that MIPS-I lacks an instruction like `bgt` (branch greater-than) because the magnitude comparison would take a little too long. MIPS-I does have `beq` and `bne` instructions that compare two registers. However, SPARC v8 has a `bgt` instruction, but it is done in such a way that here is no risk of critical path impact.

☐ How is the actual SPARC `bgt` different than a hypothetical MIPS `bgt`?

☐ How does that difference avoid critical path impact in resolve-in-ID implementations?

☐ Explain why a `bgt r1, r2, TARG` would not have a big critical path impact if it were resolved in EX.

(*b*) In the MIPS `add` instruction the `sa` field must have a value of zero. Consider a future version of MIPS in which the `sa` field would hold a scale factor, `s`. The result of the add would be `rsv + rtv * s`. Suppose that analysis of users' programs found that such an instruction would be **very useful** and that it could **easily be implemented in hardware**. Should the `add` be extended in that way? If not, suggest another way of providing the scaled add.

☐ Should `add` be extended to compute `rsv + rtv*s`?

☐ Explain a possible objection and suggest an alternative way of including the instruction.

(*c*) The MIPS-I assembly instruction below is invalid. Explain why and replace it with one that correctly adds two double-precision values.

```
add.d $f0, $f1, $f2
```

(*d*) What is the difference between a dependency and a hazard?

☐ The difference between a dependency and a hazard is:

(*e*) Identify the type of dependence between each pair of instructions below, and indicate the corresponding hazard.

☐ The type of dependence is:_____. The corresponding hazard is:_____

```
add r1, r2, r3
sub r1, r5, r6
```

☐ The type of dependence is:_____. The corresponding hazard is:_____

```
add r1, r2, r3
sub r4, r1, r6
```

☐ The type of dependence is:_____. The corresponding hazard is:_____

```
add r1, r4, r3
sub r4, r2, r6
```

(*f*) In class we said that the lifetime of an ISA can be decades and so it must be carefully designed to take into account current and future implementation technologies. Is IA-32 (80x86) a good example of this rule? Explain.

☐ IA-32 ◯ *is* ◯ *is not* a good example of this rule because ....

(*g*) Indicate the most appropriate ISA family for each ISA below.

☐ Intel 64 is considered ◯ *RISC* ◯ *CISC* ◯ *VLIW*

☐ Itanium is considered ◯ *RISC* ◯ *CISC* ◯ *VLIW*

☐ MIPS is considered ◯ *RISC* ◯ *CISC* ◯ *VLIW*

☐ SPARC is considered ◯ *RISC* ◯ *CISC* ◯ *VLIW*

☐ VAX is considered ◯ *RISC* ◯ *CISC* ◯ *VLIW*