Name ______________________________

Computer Architecture
LSU EE 4720
Midterm Solve-Home Examination

Tuesday, 14 April 2020 to Friday, 17 April 2020 23:59 CDT

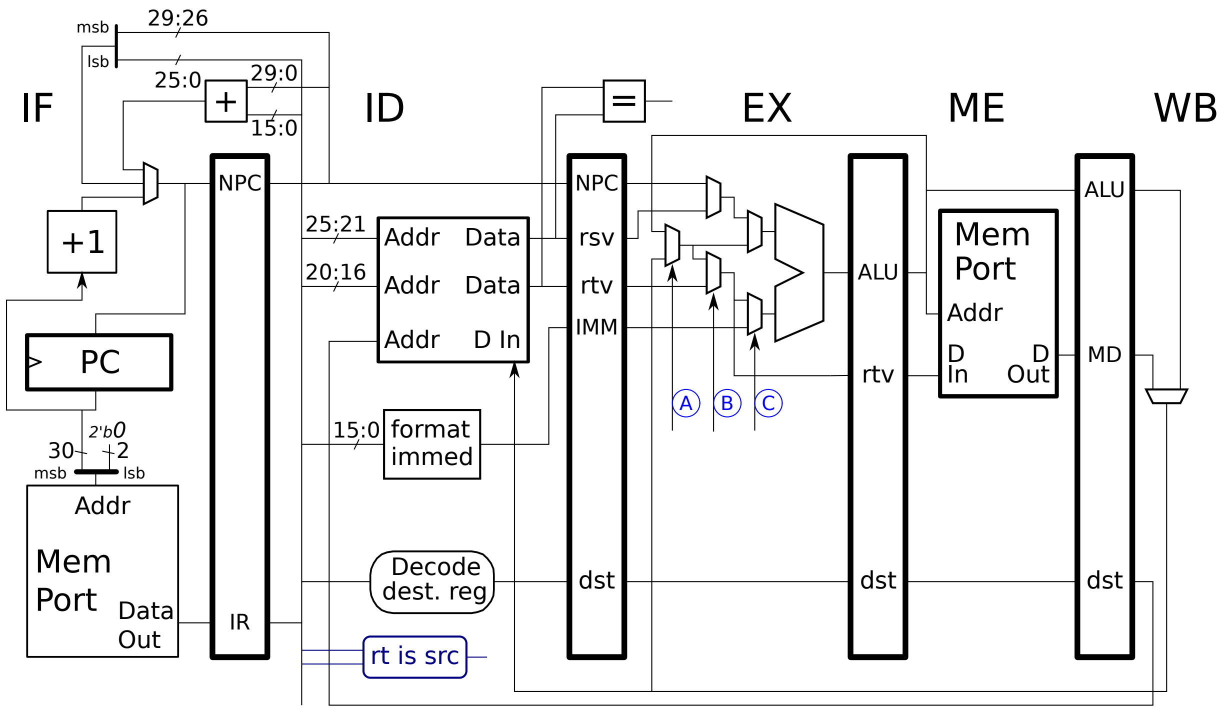
Work on this exam alone. Regular class resources, such as notes, papers, documentation, and code, can be used to find solutions. Do not discuss this exam with classmates or anyone else, except questions or concerns about problems should be directed to Dr. Koppelman.

Problem 1	________	(15 pts)
Problem 2	________	(25 pts)
Problem 3	________	(15 pts)
Problem 4	________	(15 pts)
Problem 5	________	(30 pts)
Exam Total	________	(100 pts)

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

Good Luck! Don't be Foolish!

Problem 1: [15 pts] The pipeline below is a slightly lower cost version of the bypassed MIPS implementation that we've been using. The cost saving is achieved by not allowing an instruction to use a bypassed value from both the ME and WB stage, the value must come from one stage or the other. Select inputs are shown for three of the re-done EX stage multiplexors, they are labeled A, B, and C. For this problem assume that they are connected to properly designed control logic.



(*a*) Show the values on the labeled select signals for an execution of the code below for those cycles in which an instruction below is in the EX stage. If the value on a select signal does not matter, show an X.

☐ Show values of A, B, and C for when EX occupied by code below. ☐ Use X if value does not matter, blank when no insn in EX.

```
#    Cycle         0     1     2     3     4     5     6
 add r1, r2, r3   IF    ID    EX    ME    WB
 sub r4, r5, r1         IF    ID    EX    ME    WB
 sw r6, 8(r1)                 IF    ID    EX    ME    WB

#    Cycle         0     1     2     3     4     5     6
 A


 B


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

(*b*) Show a code fragment that would stall on the implementation above but would not stall on our usual bypassed MIPS (which appears in Problem 3).

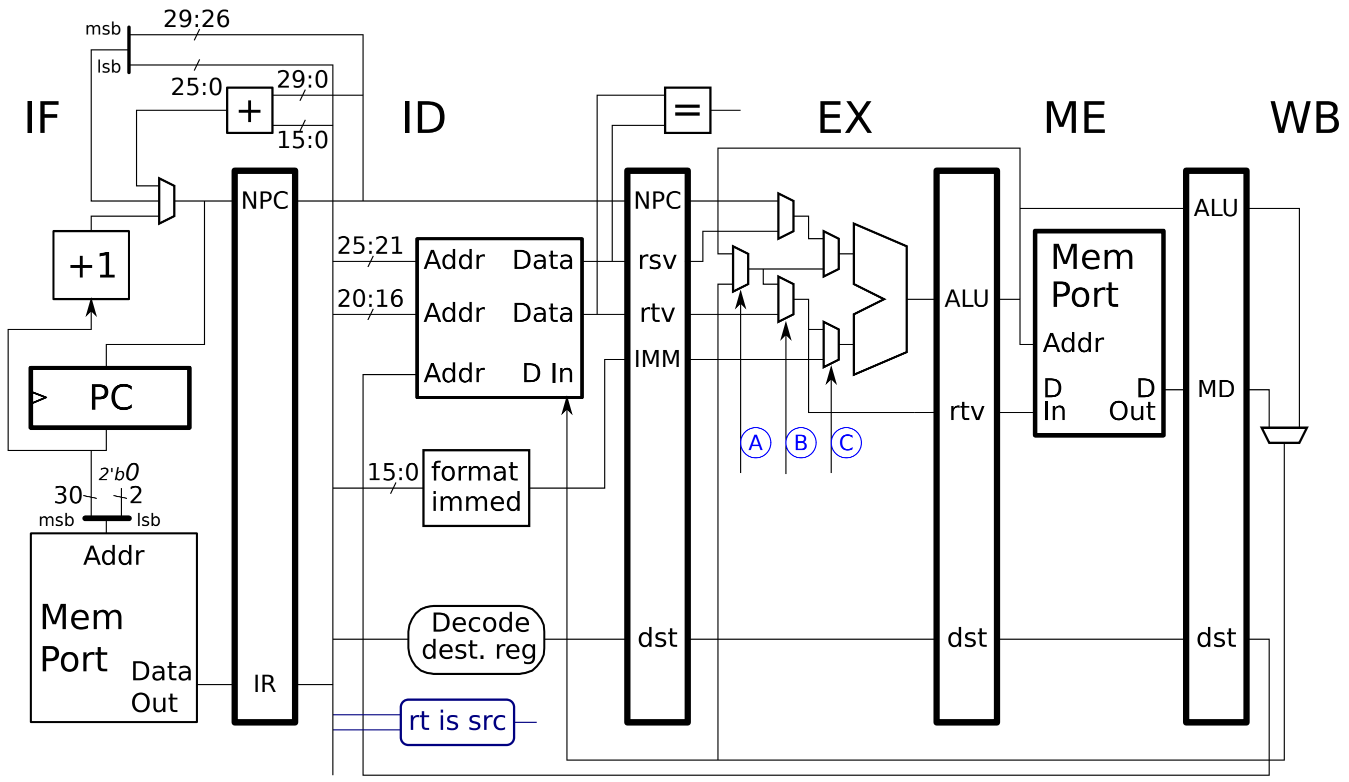
☐ Code fragment that stalls on this implementation, but not our usual 5-stage MIPS.

Problem 2: [25 pts] Appearing below is the lower-cost MIPS implementation from the previous problem. Design the control logic specified below. The output of $\boxed{\text{rt is src}}$ is 1 if the `rt` field of the instruction specifies a source value, as it does in most type R but only a few type I, such as `sw`. The Inkscape SVG source for the image below can be found at `https://www.ece.lsu.edu/ee4720/2020/mt-p1.svg`.

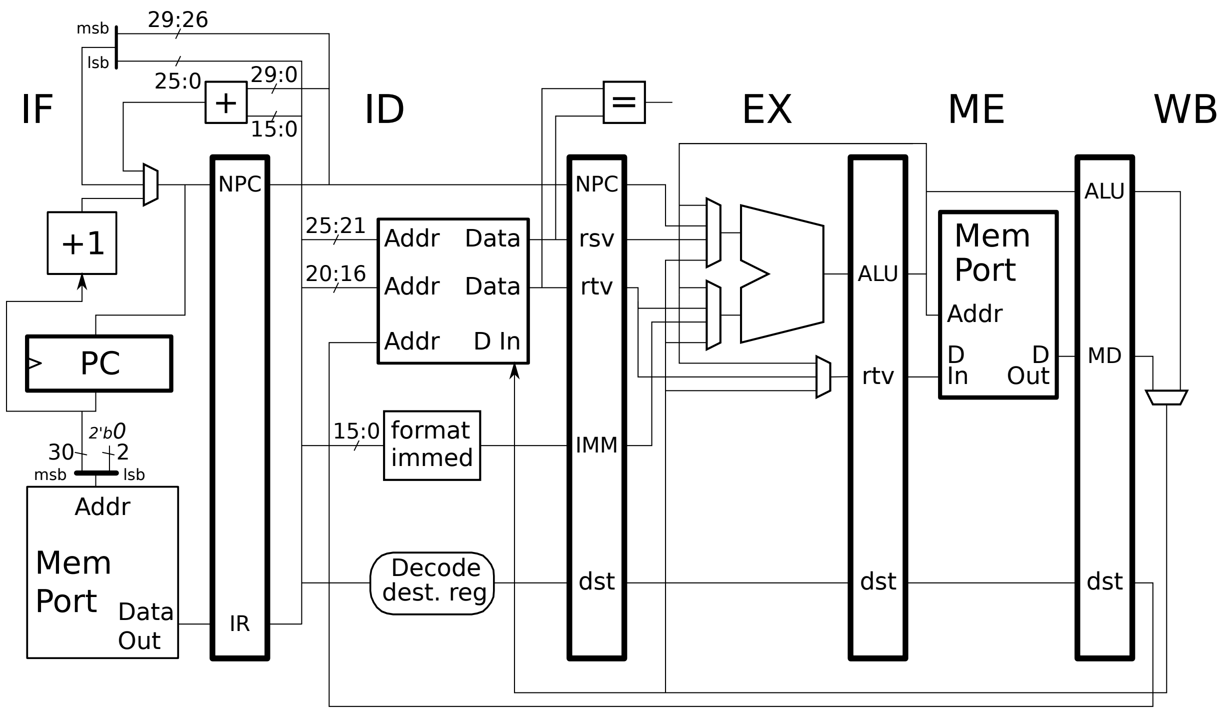
☐ Design control logic for the labeled multiplexor select signals, A, B, and C.

☐ Design control logic to generate a stall signal when a bypass would have been from both `ME` and `WB`.

☐ Pay attention to the usual stuff: ☐ Cost and critical path. ☐ The stage that instructions are in when the select signals are computed and the stage in which they are used.



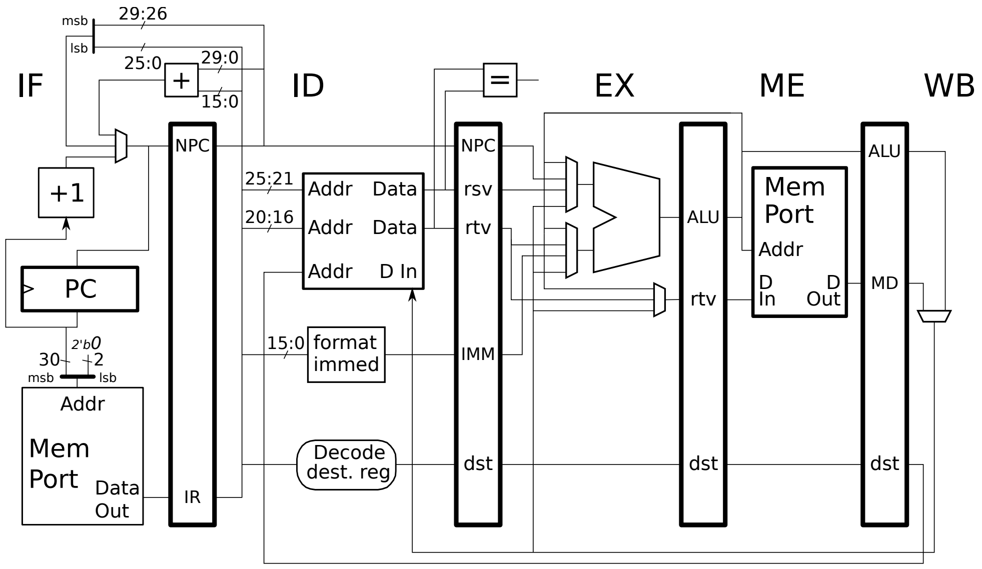
Problem 3: [15 pts] Show the execution of the code fragment below on the illustrated implementation.



☐ Show execution. ☐ Note that the branch is taken. ☐ Pay attention to the timing of the branch. ☐ Check for dependencies, ☐ including for the branch.

```
 lw r1, 0(r2)


 slt r3, r1, r4

 # Branch is taken.
 beq r3, r0  SKIP


 addi r2, r2, 4


 xor r5, r5, r9


 or  r6, r6, r9


SKIP:
 addi r7, r7, 4


 sw r1, 0(r7)
```

Problem 4: [15 pts] The code fragment below runs inefficiently. Modify the code so that it runs faster on the implementation below. Instructions can be re-arranged, changed, or removed, and registers can be changed. Don't forget that the modified needs to do the same thing as the original code.

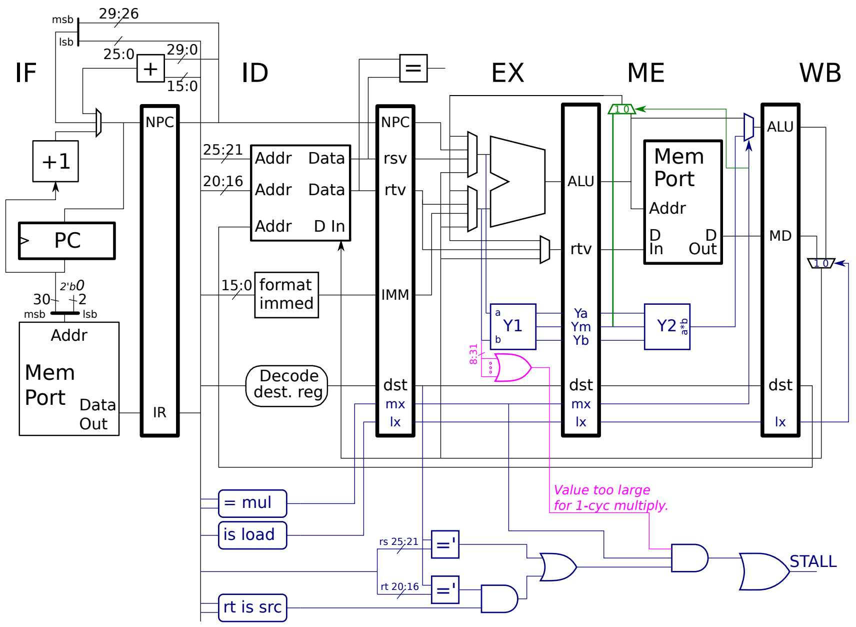


☐ Re-write code so that it is faster but, of course, does the same thing as the original.

```
LOOP:
 lw r1, 0(r2)
 andi r1, r1, 0xff
 addi r2, r2, 4
 lw r3, 0(r2)
 srl r3, r3, 24
 add r9, r9, r1
 add r9, r9, r3
 addi r2, r2, 4
 sub r8, r2, r11
 bne r8, r0  LOOP
 nop
```

Problem 5: [30 pts] Answer each question below.

(*a*) The code fragments below are to run on the implementation with the small-multiply bypass from Homework 3 and shown to the right. For each code fragment, indicate whether our small-value bypass feature always eliminates a stall, sometimes, or never? Explain.



☐ Eliminates stall on code below: ◯ *Always* ◯ *Sometimes* ◯ *Never*

```
andi r3, r5, 0x3f
mul r1, r2, r3
add r6, r6, r1
lw r10, 0(r6)
```

☐ Eliminates stall on code below: ◯ *Always* ◯ *Sometimes* ◯ *Never*

```
ori r3, r5, 0x63f
mul r1, r2, r3
add r6, r6, r1
lw r10, 0(r6)
```

☐ Eliminates stall on code below: ◯ *Always* ◯ *Sometimes* ◯ *Never*

```
lbu r3, 0(r4)
mul r1, r2, r3
add r6, r6, r1
lw r10, 0(r6)
```

☐ Eliminates stall on code below: ◯ *Always* ◯ *Sometimes* ◯ *Never*

```
lw r3, 0(r4)
mul r1, r2, r3
add r6, r6, r1
lw r10, 0(r6)
```

(*b*) In typical practice a company decides upon an ISA, and then makes multiple implementations of that ISA. Let H_I and L_I be two implementations of ISA I, H_I is a high-end system and L_I is low-cost. Let ISA E (for expensive) be an ISA designed for high-end systems, and ISA C (for cheap) be an ISA designed for low-cost systems, and let H_E and L_C be their implementations. All three ISAs and all four implementations were designed by skilled engineers with lots of resources.

☐ Why might H_E be better than H_I and why might L_C be better than L_I? The same reason should apply to both. The answer is related to the ISAs used for the implementations.

☐ Even if L_C is better than L_I, why might a user still choose L_I?

(*c*) Consider the preparation of a set of SPECcpu results. For each item below indicate who is responsible, SPEC (the organization) or the tester. Also indicate what would be the problem if it were the other way around. For example, if you answered that SPEC chooses the benchmarks, then explain the disadvantage of having the tester choose the benchmarks.

☐ Choose the benchmarks: ○ *SPEC* or ○ *The Tester*

☐ Problem if it were the other way around:

☐ Choose the benchmark input data: ○ *SPEC* or ○ *The Tester*

☐ Problem if it were the other way around:

☐ Choose the benchmark training data: ○ *SPEC* or ○ *The Tester*

☐ Problem if it were the other way around:

☐ Choose the compiler: ○ *SPEC* or ○ *The Tester*

☐ Problem if it were the other way around:

☐ Choose the compiler optimization flags: ○ *SPEC* or ○ *The Tester*

☐ Problem if it were the other way around:

(*d*) The IA-32 ISA has been described as Intel's golden handcuffs. Who slapped on those handcuffs? What does the gold refer to? What do the handcuffs refer to? *This was discussed in class, but it is okay to use Web searches to answer this question.*

☐ The reason for these handcuffs is:

☐ They are golden because:

☐ They are handcuffs (a restriction) because:

(*e*) Appearing below are some hypothetical instructions. Indicate whether each instruction is a better candidate for a RISC ISA or a CISC ISA. Explain why.

☐ Is the instruction below more ○ *RISC* or ○ *CISC* like? ☐ Explain.

```
addi r1, r2, 0x12345678
```

☐ Is the instruction below more ○ *RISC* or ○ *CISC* like? ☐ Explain.

```
lw r1, (r2+r3)  # Load r1 = Mem[ r2 + r3 ]
```

☐ Is the instruction below more ○ *RISC* or ○ *CISC* like? ☐ Explain.

```
bgt r1, r2, TARG  # Branch if r1 < r2
```

☐ Is the instruction below more ○ *RISC* or ○ *CISC* like? ☐ Explain.

```
add (r1), r2, (r3)   # Mem[r1] = r2 + Mem[r3]
```