Name _____

Computer Architecture
EE 4720
Midterm Examination

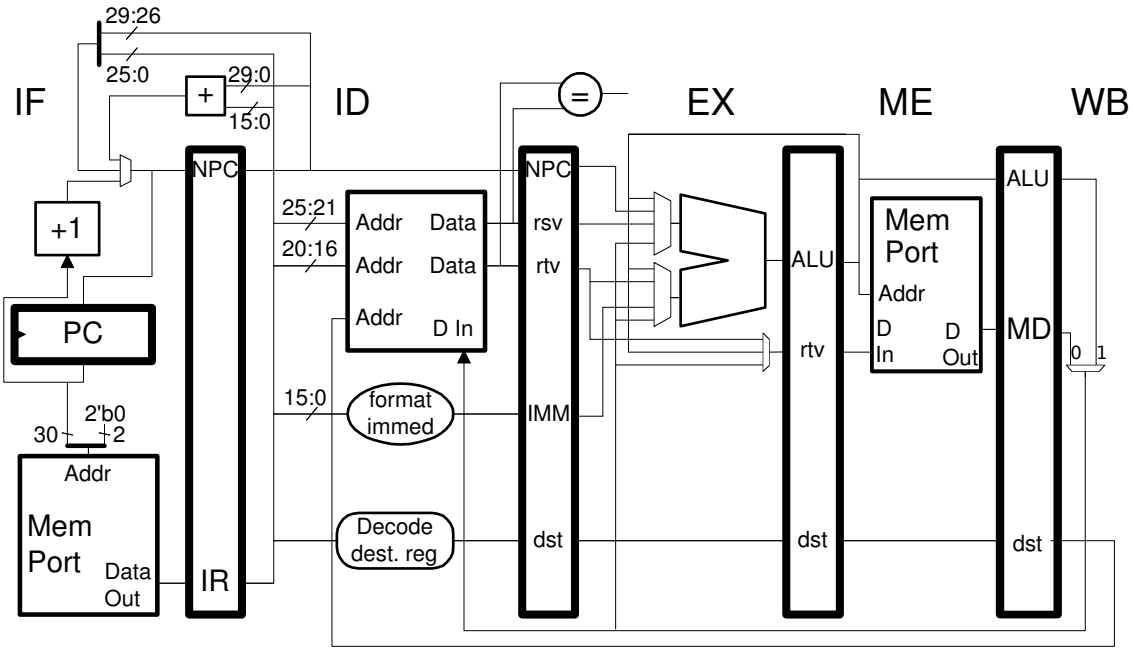
Wednesday, 13 March 2013, 9:30–10:20 CDT

Problem 1	_____	(20 pts)
Problem 2	_____	(15 pts)
Problem 3	_____	(15 pts)
Problem 4	_____	(9 pts)
Problem 5	_____	(11 pts)
Problem 6	_____	(20 pts)
Problem 7	_____	(10 pts)
Exam Total	_____	(100 pts)

Alias _____

Good Luck!

Problem 1: [20 pts] The code below executes on the illustrated MIPS implementation, which is the one usually used in class.



(*a*) Show the execution of the code below until the start of the third iteration.

- **Check** and **double-check** the code for **dependencies.**

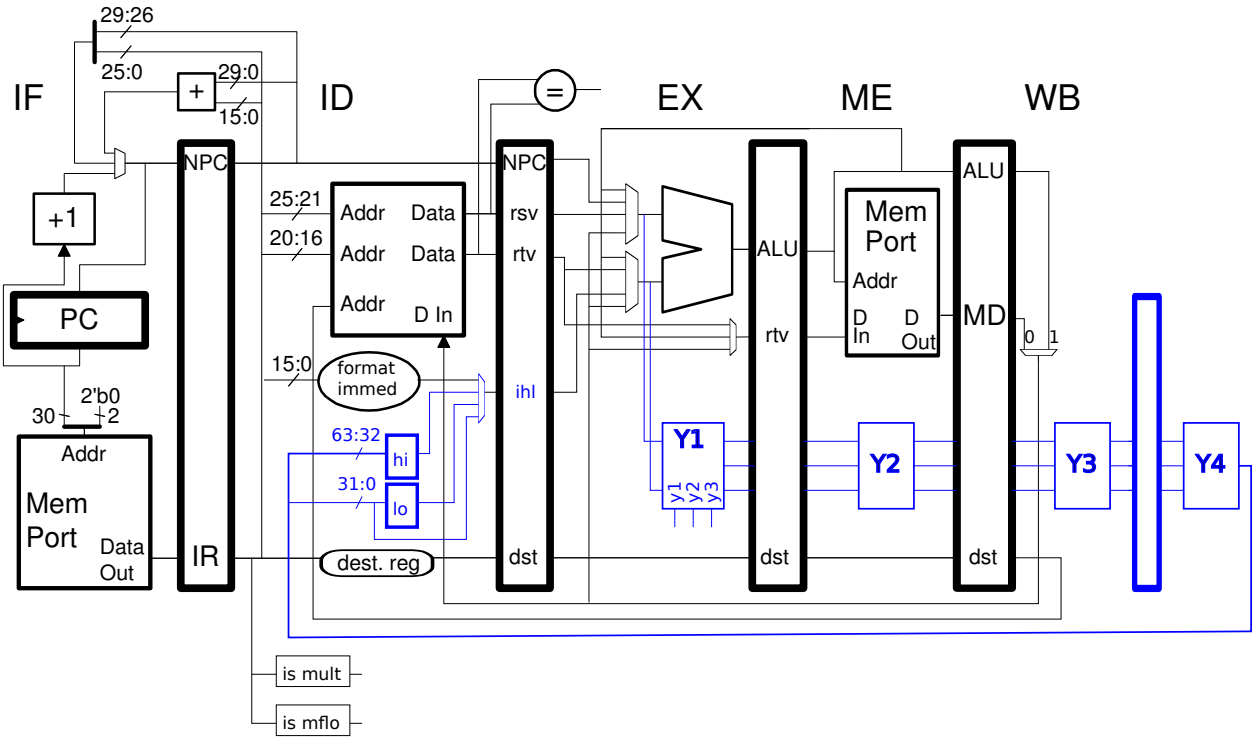
☐ Show execution to start of third iteration.

```
LOOP:
 add r1, r2, r3

 sw r4, 5(r1)

 bne r1, r6, LOOP

 lw r2, 7(r8)
```

(*b*) Compute the CPI assuming a large number of iterations. *Hint: It's important to consider up to the third iteration.*

☐ CPI is:

Problem 2: [15 pts] Illustrated below is our familiar five-stage MIPS implementation with a multiply unit added. Unlike the multiplier in Homework 4, this unit has four stages.



(*a*) Show the execution of the code below on the illustrated implementation.

☐ Show execution of code.

```
mult r1, r2

mflo r3

mfhi r4
```

(*b*) Explain why execution of the following code would be different.

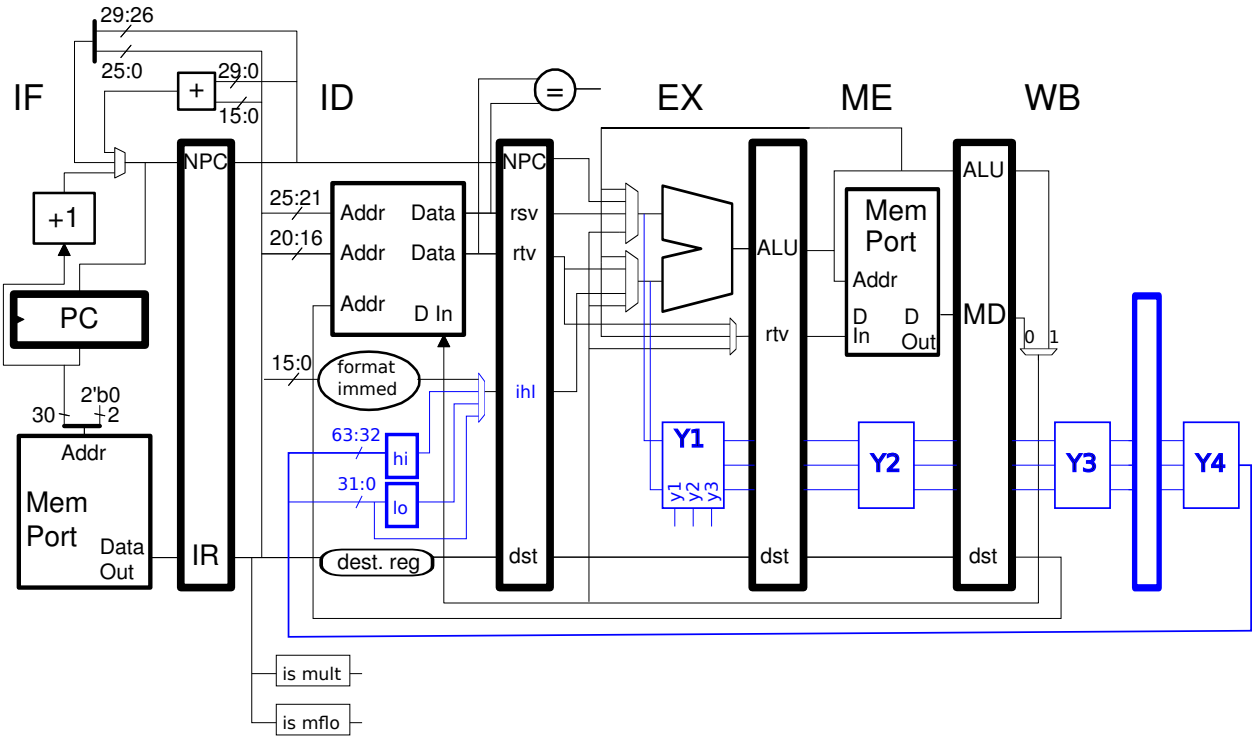
```
mult r1, r2
mfhi r4
mflo r3
```

☐ Execution of the code above would be different because:

(*c*) Design the control logic needed to generate a stall signal for dependencies between a `mult` and a `mflo`, something the execution above should have experienced. For this problem do not assume any bypass paths for the product other than those illustrated. Note that logic to identify the `mult` and `mflo` instructions has already been added to the `ID` stage.

☐ Control logic to generate stall signal.

Problem 3: [15 pts] In this problem the multiply unit has been designed to produce a product sooner for special cases, such as when one of its inputs is zero. To indicate when the product will be available `Y1` has three one-bit outputs along the bottom, `y1`, `y2`, and `y3`. Signal `y1` is `1` if the center output of `Y1` will have a completed product at the end of the cycle, `y2` is `1` if the center output `Y2` will have a completed product at the end of the cycle in which the `mult` is in `Y2`, output `y3` works similarly. If `y1` is `1`, then `y2` and `y3` are also `1`, if `y2` is `1` then `y3` is also `1`. In other words, if an early product is available at the output of `Y1`, it will be at the outputs of `Y2` and `Y3` in subsequent cycles.




```
# Cycle             0  1  2  3  4  5  6  7
    mult r10, r11   IF ID Y1 Y2 Y3 Y4
a:  mflo r1            IF ID EX ME WB
b:  mflo r2               IF ID EX ME WB
c:  mflo r3                  IF ID EX ME WB
# Cycle             0  1  2  3  4  5  6  7
```

(*a*) Add bypass paths so that the code executes above without a stall assuming an early product is available.

☐ Show bypass paths for each of the `mflo` instructions above.

☐ Label the bypass paths with `a`, `b`, and `c`, based on the instruction (above) that uses them.

Problem 4: [9 pts] Show the encoding of each of the instructions below. Be sure to handle the branch target correctly. Fill in as many fields as you can. Use the value "lookup" for any field who's value can't be determined by inspection.

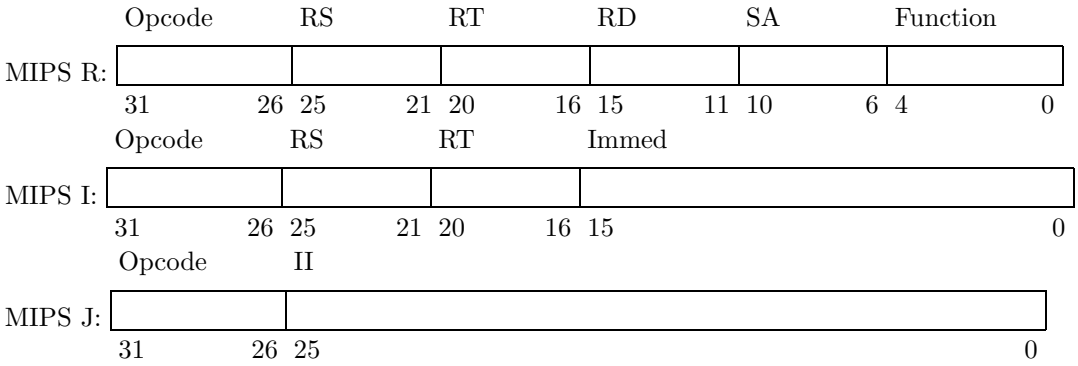
```
 0x1000 beq r1, r2  TARG
 0x1004 lui r1, 0x1234
TARG:
 0x1404 sw r3, 4(r5)
```

☐ Encoding for `beq` as used above:

☐ Encoding for `lui` as used above:

☐ Encoding for `sw` as used above:

The formats appearing below are for reference.



Problem 5: [11 pts] Answer each question below.

(*a*) MIPS and many other RISC ISAs have a special zero register, `r0` in MIPS. VAX, a CISC ISA, lacks a zero register, but VAX has an `mneg` (negate) instruction (`dest = -source`), which MIPS and other RISC ISAs lack.

☐ Why does MIPS' zero register make it unnecessary to have a negate instruction?

☐ What can VAX use in place of a zero register? (The answer is not `mneg` or any other instruction.)

☐ If VAX can use its zero register replacement in the same way MIPS uses its zero register, what is the advantage for VAX in having a negate instruction? A quantitative answer will get full credit.

(*b*) Why might it be possible to have a higher clock frequency in an implementation of an ISA that uses condition code registers (such as SPARC) compared to one which allows branch instructions to compare two registers? Consider the types of implementations covered in class so far.

☐ Higher clock frequency possible with condition code registers because:

Problem 6: [20 pts] Answer each question below.

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

☐ Difference between dependency and hazard:

(*b*) How do we know that the benchmarks in SPECcpu were not unfairly chosen to favor an influential company's processors.

☐ Can trust choice of SPECcpu benchmarks because:

(*c*) When using profiling for optimization a program needs to be compiled twice. Explain what happens in each compilation related to profiling.

☐ During the first compilation:

☐ During the second compilation:

(*d*) Who would benefit the most if each new processor implemented a new, one-of-a-kind, ISA: computer engineers, compiler back-end writers, software developers, computer buyers? Explain.

☐ Group that would benefit most (circle): computer engineers, compiler back-end writers, software developers, computer buyers.

☐ Reason that the group would benefit the most:

Problem 7: [10 pts] Compiler and microarchitecture experts at a computer company are preparing the optimization flags for a SPECcpu run. They have come up with a different set of optimization settings for each benchmark in the suite.

(*a*) Based on the SPEC rules it is okay to use these individualized sets for the peak tuning runs but not the base tuning runs. Why?

☐ Reason individualized settings are not allowed for base:

(*b*) The compiler writer figures out a way to have the compiler itself, given only the `-O3` flag, choose the same individualized sets as the expert did. The compiler expert makes these changes to the compiler, which the company will start selling. It is easy to do this in a way which goes against the spirit of the SPECcpu rules. It would require great skill to do this in a way consistent with the spirit of the SPECcpu rules.

☐ Easy way to have `-O3` pick correct set of optimizations, but against spirit (goals) of rules. Explain.

☐ Hard way to have `-O3` pick correct set of optimizations, but in spirit (goals) of rules. Explain.