Name _____

Computer Architecture
EE 4720
Midterm Examination

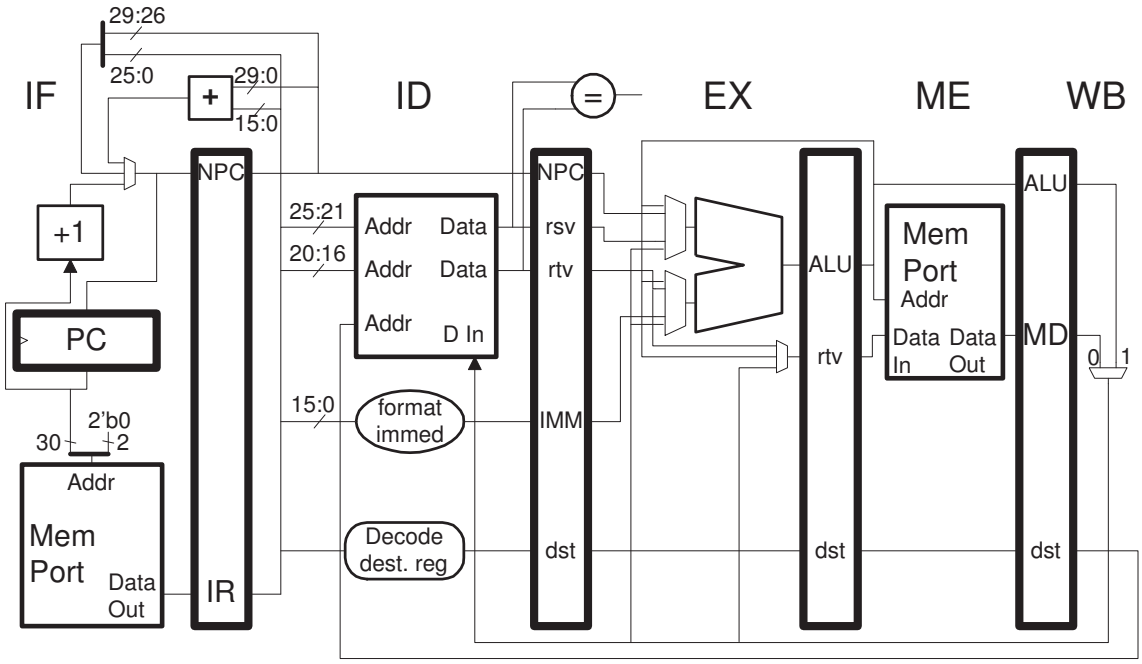
Wednesday, 3 November 2010, 10:40–11:30 CDT

Problem 1	_____	(30 pts)
Problem 2	_____	(15 pts)
Problem 3	_____	(15 pts)
Problem 4	_____	(23 pts)
Problem 5	_____	(10 pts)
Problem 6	_____	(7 pts)
Exam Total	_____	(100 pts)

Alias _____

Good Luck!

Problem 1: Show the execution of the following code fragments on the illustrated MIPS implementations.



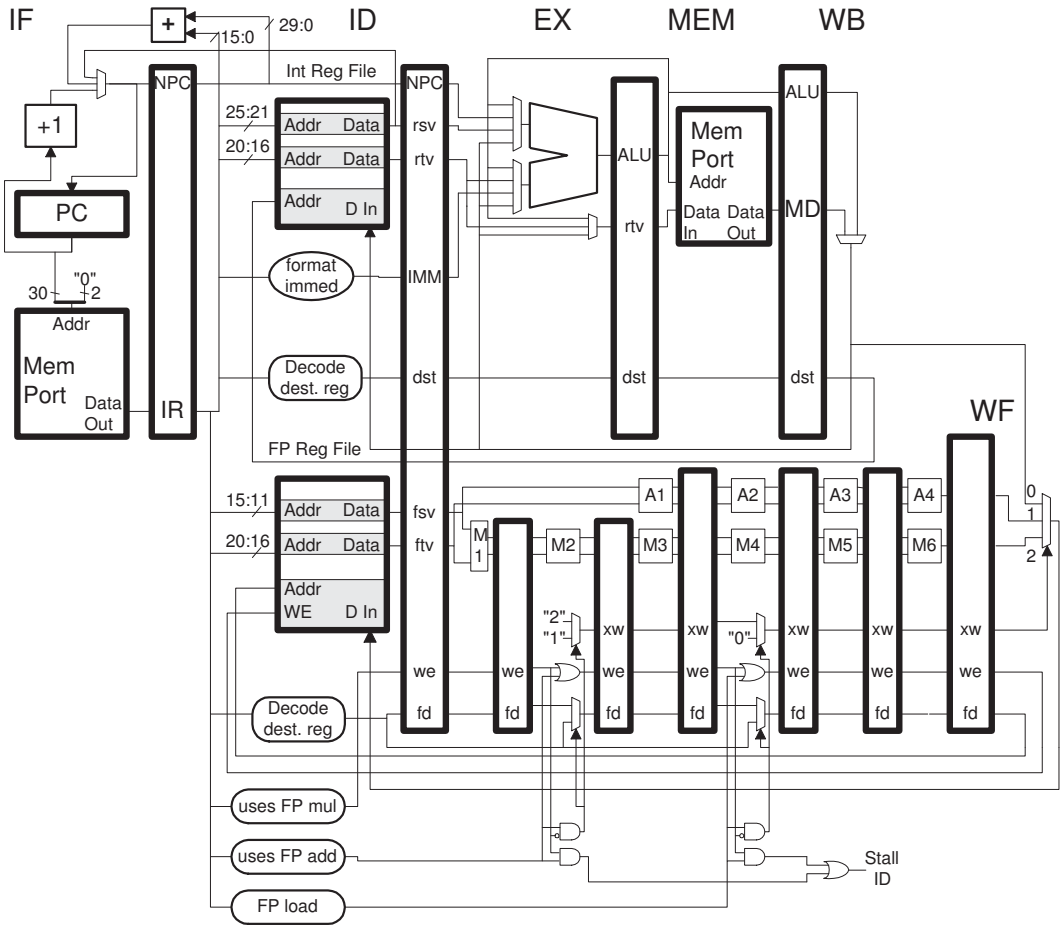
(*a*) [20 pts] The code below executes for many iterations. Show a pipeline execution diagram for the execution of the code on the implementation above for enough iterations to determine the CPI, and determine the CPI.

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

 lw r3, 0(r1)

 bne r3, r4, LOOP

 lw r2, 8(r1)
```

☐ Pipeline diagram of execution. Don't forget to check for dependencies!

☐ CPI for a large number of iterations.

Problem 1, continued:



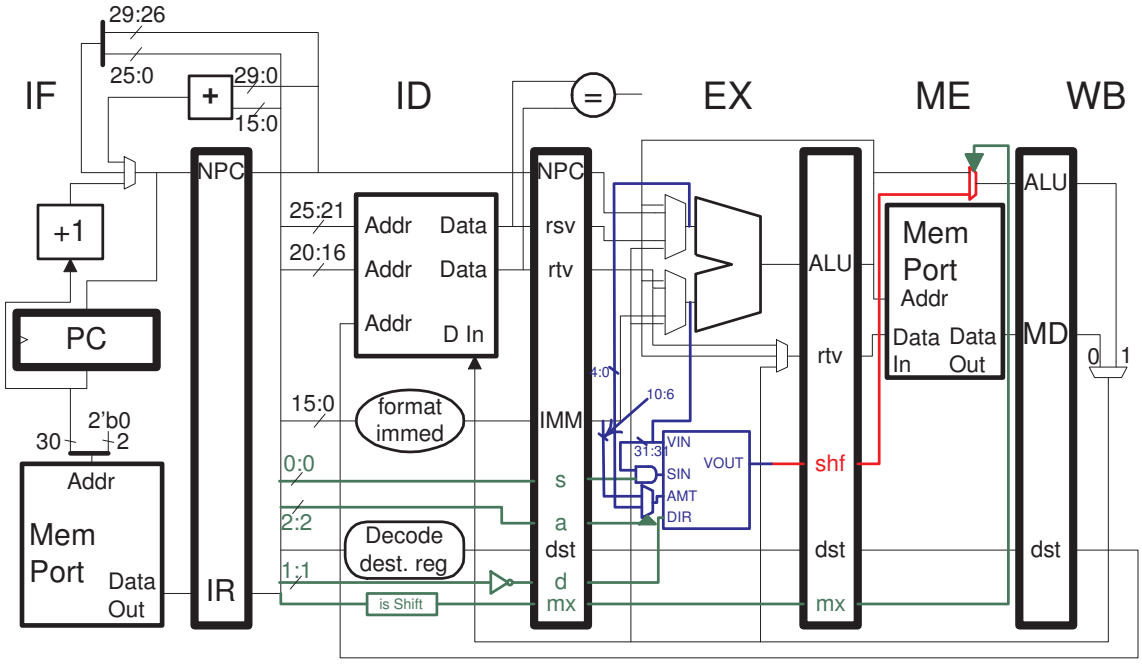
(*b*) [10 pts] Show the execution of the code below on the implementation above.

```
mul.s f1, f2, f3

add.s f4, f5, f6

add.s f7, f8, f9

lwc1 f10, 0(r1)
```

☐ Pipeline diagram of execution.

Problem 2: The MIPS implementation below includes the shift unit (taken from the Homework 3 solution). Notice that it is not possible to bypass a shift result to the EX stage. For that reason the `add` in the code below stalls:

```
# Cycle          0  1  2  3  4  5  6
sll r1, r2, r3   IF ID EX ME WB
add r4, r1, r5      IF ID -> EX ME WB
```



(*a*) [10 pts] Design control logic to generate a signal named `STALL` which will be logic `1` when a stall is needed due to a shift result that cannot be bypassed, as in the example above. (Generate the signal, but don't do anything with it.)

☐ Logic to generate stall signal.

(*b*) [5 pts] The stall above can be avoided by disconnecting the `ME`-to-`EX` bypass from the `EX/ME.ALU` latch and instead connecting it to the output of the `ME`-stage mux. What was the reason for **not** doing something like that in Homework 3.

☐ Problem with `ME`-to-`EX` bypass for shift results.

Problem 3: Answer the following questions about interrupts.

(*a*) [7 pts] An important part of an ISA's interrupt mechanism is a separate privileged mode (also called system or supervisor mode) and user mode.

☐ Why is it necessary to have these two modes?

☐ Explain the difference between privileged and user mode in how the CPU operates.

(*b*) [8 pts] In class our 5-stage implementations resolved exceptions only in `ME`, and by doing so all integer-pipeline exceptions were precise. *Note: In the original exam the phrase "and by doing so..." was not present.* Suppose instead we resolved exceptions in `WB`. Show a code fragment in which an exception could not be precise on such a system.

☐ Simple code fragment.

☐ What can't the handler do, and why can't it do it?

Problem 4: Answer each question below.

(*a*) [8 pts] Consider the distinction between an ISA and its implementation.

☐ What was to be achieved by separating computer design into separate ISA design and implementation design?

☐ Provide a reason not to have separate ISA and implementation design, consider the point of view of a conservative computer engineer from the 1960s.

(*b*) [8 pts] One reason to not compile a program with optimization turned on is because you plan to debug the program. Explain why stepping through a program in a debugger can be confusing when the code has been optimized.

☐ Debugging optimized code is confusing because ...

☐ Name a particular type of optimization and explain how it can cause confusing results when single stepping through a program.

(*c*) [7 pts] Do you agree or disagree with the statement below regarding the rules for building the SPECcpu benchmarks? Answer with respect to the goals for the SPECcpu benchmarks.

"Testers should not be allowed to use their own compilers because the SPECcpu benchmarks are supposed to test CPUs, not compilers. All testers of a particular ISA should use the same compiler." *Grading note: the phrase "of a particular ISA" was not in the original exam.*

☐ Agree or disagree? Explain.

Problem 5: [10 pts] CISC programs generally are smaller than RISC programs.

(*a*) Show how the instruction below is encoded in MIPS and in VAX. For MIPS the name and bit position of each field should be known, and the value of all but one field should be known. For VAX one should know the fields and their sizes, but not every field value needs to be known. *Hint: VAX has 16 general-purpose registers. The size of the two instructions should be the same.*

```
add r1, r2, r3
```

☐ Encoding in MIPS.

☐ Encoding in VAX.

(*b*) Identify unused field(s) in the MIPS instruction.

☐ Unused MIPS field.

(*c*) Since VAX instruction sizes can vary they should be able to use space more efficiently. Yet even though the MIPS instruction has unused field(s) the two instructions are the same size. Explain why the VAX instruction is larger than one might expect and explain what advantage that provides.

☐ Why is VAX larger than one might expect?

☐ What advantage does this larger size provide?

Problem 6: Answer each question below.

(*a*) [7 pts] Unlike MIPS, SPARC integer branches use condition codes.

☐ Why might this allow SPARC branch targets to be further away than MIPS branch targets?

☐ Why might this enable certain SPARC implementations to have a higher clock frequency than comparable MIPS implementations?