Name ____________________

Computer Architecture
EE 4720
Midterm Examination

Friday, 22 March 2002, 13:40–14:30 CST

Problem 1	______	(35 pts)
Problem 2	______	(25 pts)
Problem 3	______	(40 pts)
Exam Total	______	(100 pts)

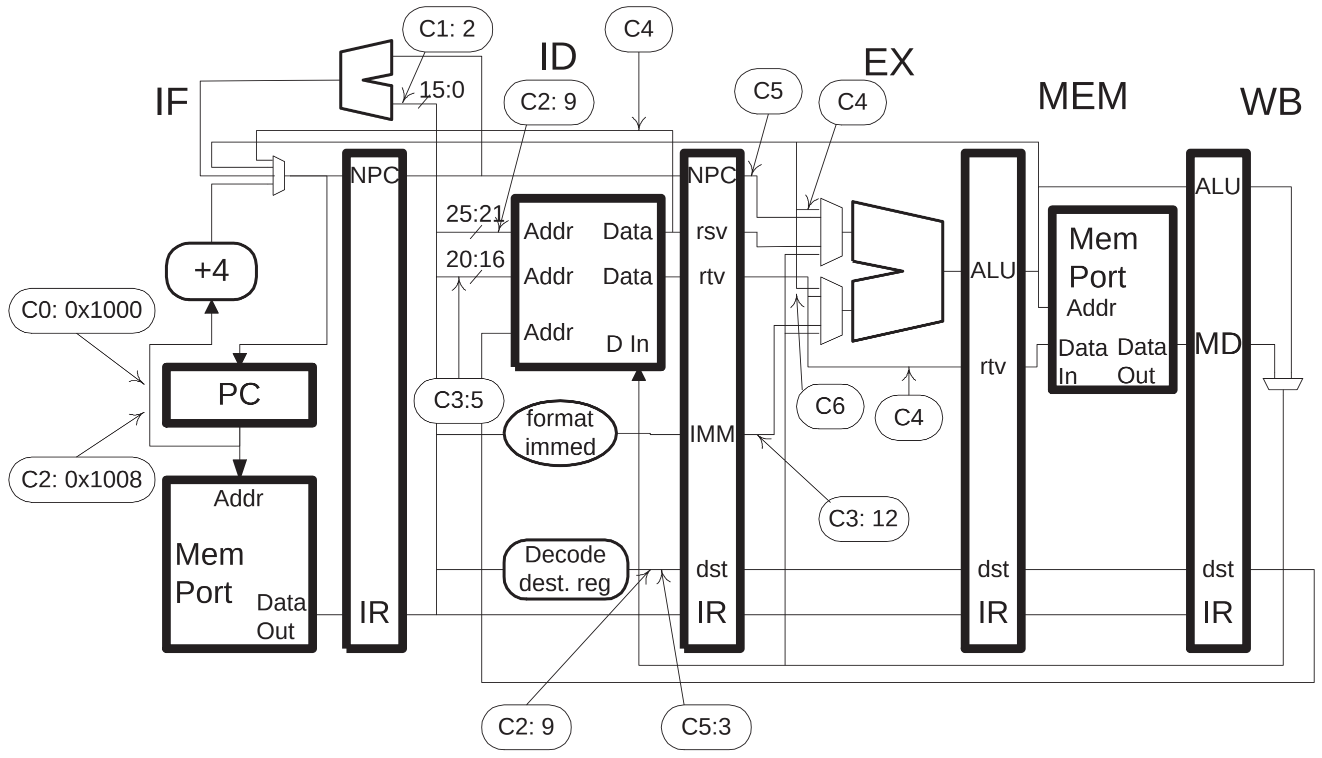
Alias ____________________

Good Luck!

Problem 1: In the diagram below certain wires are labeled with cycle numbers and values that will then be present, for example, $\boxed{\text{C2:9}}$ indicates that at cycle 2 the wire will hold a 9. Other wires are labeled just with cycle numbers, indicating that the wire is used at that cycle. Write a program consistent with these labels. There are no stalls during the execution of the code. The code should use five instructions. *Note: The diagram in the original exam had an error that resulted in contradictory information about the third instruction (fetched in cycle 2). The error was a* $\boxed{\textit{C3:5}}$ *pointing to the input to the* `ID/EX.dst` *latch; that now points to the rt address input to the register file.* [35 pts]

☐ Write a program consistent with these labels.

☐ Use labels for branch targets (if any) and label the target line.



```
 # Cycle              0    1    2    3    4    5    6    7    8    9

                      IF   ID   EX   ME   WB

                           IF   ID   EX   ME   WB

                                IF   ID   EX   ME   WB

                                     IF   ID   EX   ME   WB

                                          IF   ID   EX   ME   WB

                                               IF   ID   EX   ME   WB
```

```
# Cycle              0    1    2    3    4    5    6    7    8    9
```

Problem 2: Consider the CISCy instruction below: [25 pts]

(*a*)

```
 add 0x123456(r1), 8(r2), r3   # 0x123456(r1) is the destination.
```

☐ What makes it CISCy? (CISCy means characteristic of CISC ISAs.)

☐ Ignoring instruction size, why would it be difficult to implement such an instruction on the kind of five-stage pipeline used in class for MIPS?

Problem 2, continued:

```
add 0x123456(r1), 8(r2), r3   # 0x123456(r1) is the destination.
```

(*b*) How could the single-issue (not superscalar) pipeline used in class be modified so that instructions like the one above could be executed with a potential for a CPI of 1 without impacting clock frequency? (Instructions like the one above have one destination and two sources, the destination and first source can either use a register value or memory value specified with displacement addressing, the second source can be either a register value or an immediate.) Feel free to throw hardware at the problem, but do not assume that the hardware is any faster than what we've been using.

☐ Show a sketch of the pipeline, briefly explaining what should be done in each stage.

☐ Show a pipeline execution diagram for the following code on your new pipeline, assuming no dependencies.

```
add 0x1234(r1), 8(r2), r3

or r4, 7(r5), r6

and 0x1234(r7), r8, r9
```

☐ In the part above, why was it necessary to *assume* no dependencies?

Problem 3: Answer each question below.

(*a*) What'll it be? One FP adder with an initiation interval of 2 and a latency of 3 (four cycles of computation) or two FP adders each with an initiation interval of 4 and a latency of 3? [10 pts]

☐ What is the maximum number of FP adds per second with each alternative on a 1 GHz system?

☐ Show a code fragment in which one of the alternatives is slightly faster. For the alternative with two adders, use label "A" for one adder and "B" for the other.

☐ Ignoring the cost of the adders themselves, which alternative would be more costly and why?

(*b*) SPEC benchmark numbers are provided in "base" and "peak" forms. [10 pts]

☐ How are they different?

☐ When should an intelligent (passed EE 4720) computer buyer use the base numbers, and when should the buyer use the peak numbers?

(*c*) Stack ISAs had burrowed their way in to our past and percolated to the fore in the past decade. [10 pts]

☐ Why are programs compiled for stack ISAs smaller than the same programs compiled for other ISAs? (Assume all compilers are of high quality.)

☐ Why would superscalar implementations of stack ISAs be less efficient (further from the ideal CPI) than superscalar implementations of RISC and some other "conventional" ISAs? Ignore instruction fetch problems. Consider the simple stack programs presented in class.

(*d*) At last superscalar implementations and VLIW ISAs will wage their epic [tm] battle now at the dawn of the twenty-first century.[10 pts]

☐ What distinguishes a superscalar implementation from a nonsuperscalar implementation? (VLIW isn't part of this question).

☐ Explain two ways in which VLIW machines overcome problems encountered in superscalar implementations of conventional (say RISC) ISAs?