Name _____

Computer Architecture

EE 4720

Final Examination

18 May 2002, 12:30–14:30 CDT

Problem 1	_____	(20 pts)
Problem 2	_____	(37 pts)
Problem 3	_____	(20 pts)
Problem 4	_____	(23 pts)
Exam Total	_____	(100 pts)

Alias _____

Good Luck!

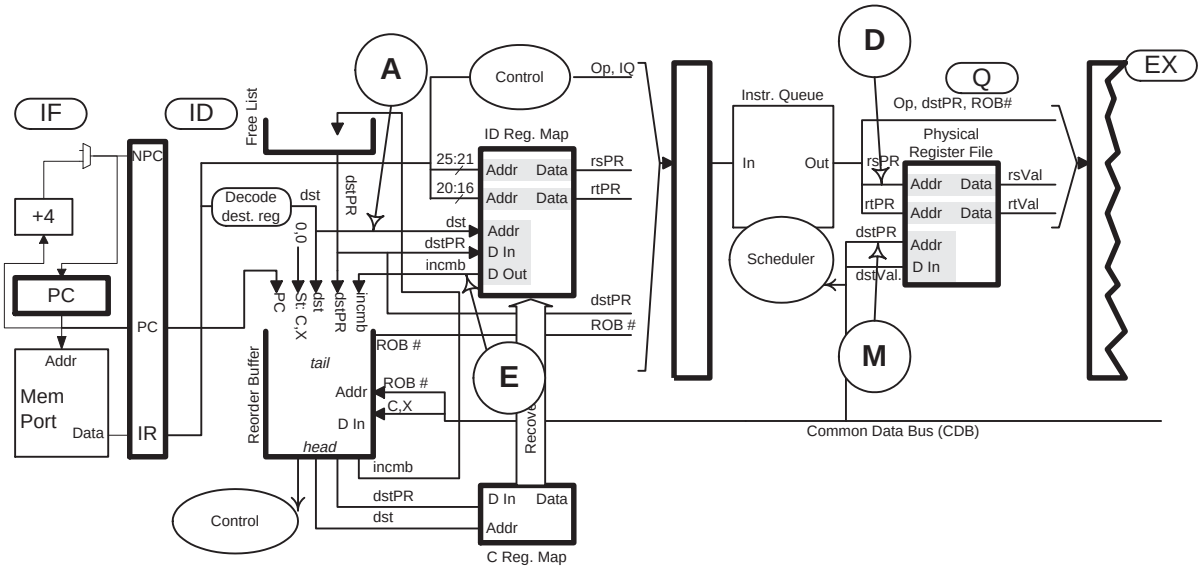
Problem 1: The (hopefully familiar) code fragment on the next page executes as shown on the dynamically scheduled system illustrated below.

(*a*) For each entry in the Physical Register File table on the next page place a "[" at the cycle(s) at which a physical register is allocated (removed from the free list) and place a "]" at the cycle(s) at which it is placed back in the free list. (This was part of the solution to Homework 5.) (10 pts)

(*b*) The diagram below includes circled letters, these letters appear in a Signals Values Table on the next page.

☐ (10 pts) Fill in the table showing the signals that will appear on the labeled wires.

- Use register names (`f0`, `$1`, etc.) for architected registers (but not for physical registers!).
- Use a question mark for a physical register number that cannot be determined from the tables.
- Since the machine is four-way superscalar each wire holds four values. Do not show values for the instructions that are not in the table, such as the two instructions following `sub`.
- Assume that values have been correctly computed, even if they depend upon preceding instructions in the same group. (This affects row E in the table.)



```
LOOP: # Cycle     0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16
 ldc1 f0, 0($1)   IF ID Q  L1 L2 WC
                           IF ID Q  L1 L2 WB              C
                                    IF ID Q  L1 L2 WB             C
                                             IF ID Q  L1 L2 WB
 mul.d f0, f0, f2 IF ID Q        M1 M2 M3 M4 M5 M6 WC
                           IF ID Q        M1 M2 M3 M4 M5 M6 WC
                                    IF ID Q        M1 M2 M3 M4 M5 M6 WC
                                             IF ID Q        M1 M2 M3 M4 M5
 sdc1 0($1), f0   IF ID Q  L1                      L2 WC
                           IF ID Q  L1                      L2 WC
                                    IF ID Q  L1                      L2 WC
 addi $1, $1, 8   IF ID Q  EX WB                     C
                           IF ID Q  EX WB                   C
                                    IF ID Q  EX WB                   C
 bne $2, $0 LOOP     IF ID Q  B  WB                   C
                              IF ID Q  B  WB                C
                                       IF ID Q     B  WB             C
 sub $2, $1, $3      IF ID Q  EX WB                   C
                              IF ID Q  EX WB                C
                                       IF ID Q  EX WB                C
 # ID Map         0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17
 f0 99              97,96    94,93    91,90
 $1 98              95       92       89
 # In cycle one first 97 is assigned to f0, then 96 (replacing 97).  The
 # same sort of replacement occurs in cycles 4 and 7.
 # Commit Map     0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17
 f0 99                          97                96    94 93    91 90
 $1 98                                               95       92       89
 # Physical Reg File    2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17
 99              1.0
 98              0x1000
 97                            10
 96                                                  11
 95                         0x1008
 94                                     20
 93                                                        2.2
 92                                  0x1010
 # Cycle          0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17

 # Signal Values  0    1    2    3    4    5    6    7    8    9    10   11   12

 A



 D



 E



 M

 # Cycle          0    1    2    3    4    5    6    7    8    9    10   11   12
```

Problem 2: The `add` instruction below is a predicated version of a MIPS instruction (for this exam). If the contents of register `$s5` (for the example) is non-zero the `add` instruction executes normally. If it's zero then it's as though the `add` never executed. Any GPR can be used to hold the predicate.

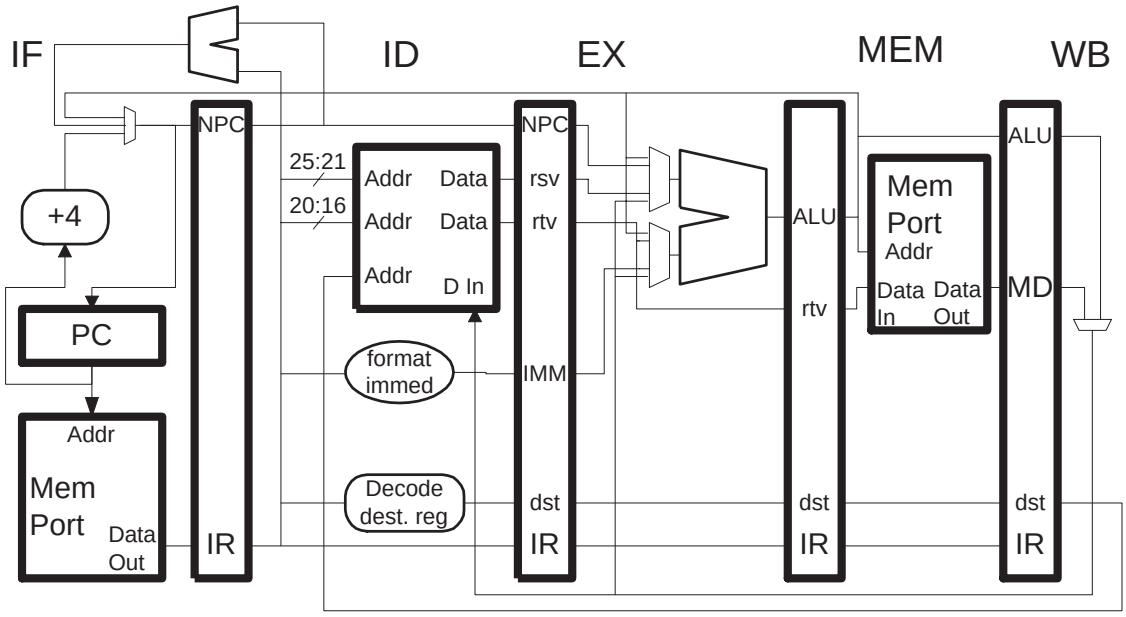
```
 # Cycle                          0  1  2  3  4  5  6
           xor $v1, $a0, $a1      IF ID EX ME WB
           lw $s5, 0($t2)            IF ID EX ME WB
     ($s5) add $s1, $s2, $s3            IF ID EX ME WB
```

(*a*) Suppose there are predicated versions of other two-source, one-destination instructions. How might they be coded in MIPS? The coding should be chosen to ease implementation. (5 pts)

(*b*) Modify the pipeline below to implement predication; it should run the code above correctly and without stalls (as illustrated). The added hardware must detect whether the predicate is true or false and take appropriate action. It should include bypassing, for example, to handle the dependency carried by `$s5` or from `xor` if the predicate were `$v1`.

Hint: A correct solution uses multiplexors, an $\boxed{is\ P}$ *(box that recognizes a predicated instruction), assorted gates, and some modifications to existing components.*

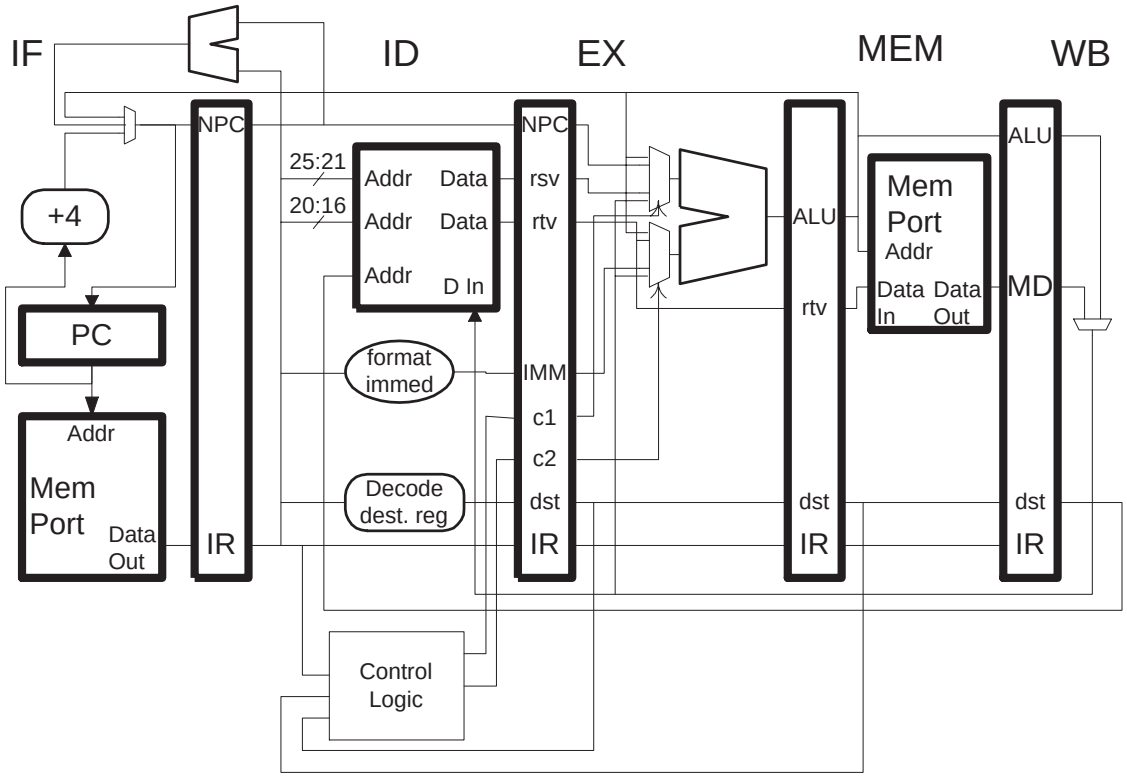
(9 pts)



Problem 2, continued:

(*c*) Assume predicates were implemented as requested in the previous part. Explain why bypassing of the dependency carried by `$s1` in the program below won't work for the hardware below. Explain how the problem might be fixed. (8 pts)

```
# Cycle                       0  1  2  3  4  5  6
          xor $s5, $a0, $a1   IF ID EX ME WB
          lw $s5, 0($t2)         IF ID EX ME WB
     ($s5) add $s1, $s2, $s3        IF ID EX ME WB
          or  $s4, $s1, $v0            IF ID EX ME WB
```



Problem 2, continued: Consider such predicated instructions in a dynamically scheduled implementation using method 3. The implementation without predicate prediction (the next two parts) should realize the benefit of predicated instructions: avoiding the need for branches.

(*d*) How might the ID Register Map be updated for predicated instructions? (For partial credit: Why is this a good question?)(5 pts)

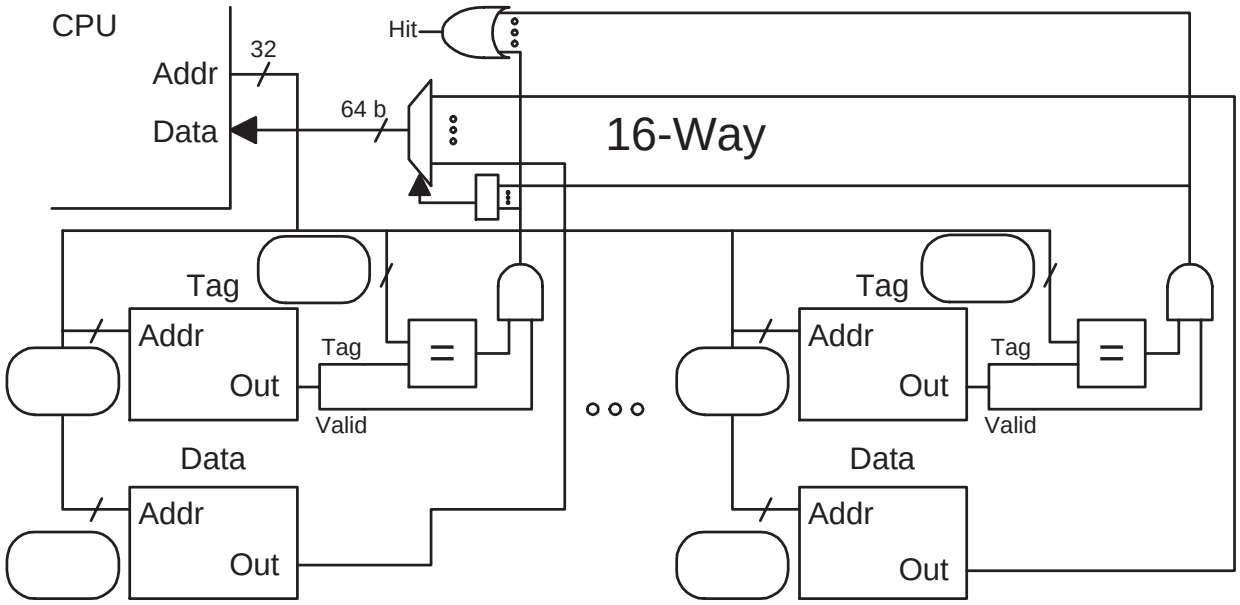
(*e*) Based on your answer above, how is execution affected if the predicate turns out to be false? (5 pts)

(*f*) Consider a dynamically scheduled system using predicate prediction. What should be done when a predicate is mispredicted? Under what circumstances (properties of program being run) would it be faster than a system without predicate prediction? Under what circumstances would it be slower? (5 pts)

Problem 3: The diagram below is for an 2 MiB (2^{21} byte) 16-way set-associative cache, with a line size of 128 characters, for a system with 8-bit (how ordinary) characters.

(*a*) Answer the following, formulæ are fine as long as they consist of literals and grade-time constants. (10 pts)

☐ Fill in the blanks in the diagram.



☐ Show the address bit categorization. Label the sections appropriately. (Alignment, Index, Offset, Tag.)

Address:

☐ Memory Needed to Implement (Indicate Unit!):

☐ Show the bit categorization for a direct mapped cache with the same capacity and line size.

Address:

Problem 3, continued: Continue to use the set-associative cache from the previous part.

When the code below starts running the cache is empty. Consider only accesses to the array.

```
half *h = 0x100000;  // sizeof(half) = 2 characters
int sum;
int i,j;
int ISTRIDE = 1;
int ILIMIT = 1024;

for(j=0; j<4; j++)
  for(i=0; i<ILIMIT; i++)
    sum += h[ ISTRIDE * i ];
```

(*b*) Answer the following. (10 pts)

☐ What is the hit ratio for the program above?

☐ Find the *minimum* value of `ISTRIDE` needed to maximize the miss ratio. (That is, *minimize* the hit ratio, make things really slow.) Don't forget that the cache is set associative. Do not modify `ILIMIT`.

Problem 4: Answer each question below.

(*a*) In the pipeline execution diagram below the multiply is squashed to avoid the WAW hazard. How does this make a precise exception impossible? (5 pts)

```
mul.d  f0, f2, f4  IF ID M1x
add.d  f0, f6, f8     IF ID A1 A2 A3 A4 WB
```

(*b*) How do processes share memory in a virtual memory system? Provide an example in which two processes share address `0x12000` but address `0x34000`, used by each process, is not shared. The addresses must be used in the example. (5 pts)

(*c*) Show how the branch history table and pattern history tables are connected in a local history branch predictor. Show how the table is indexed and how its contents are used to make a prediction. (5 pts)(For partial credit show a one-level [bimodal] predictor.)

(*d*) One way to improve performance is to divide the pipeline into more stages, as in the Pentium 4 compared to the Pentium III. This does not reduce the amount of time it takes to compute things, such as sums, though. Given that: (8 pts)

☐ How are dependencies potential performance limiters when dividing pipeline stages?

☐ How does the Pentium 4 Fast ALU get around that?

☐ Are dependencies still a problem or can we now use zillion-stage pipelines (at least as far as dependencies are concerned)? Explain.

☐ Why is branch prediction accuracy more important when there are more stages?