Name _____

Computer Architecture
EE 4720
Final Examination

14 May 2003, 15:00–17:00 CDT

Problem 1	_____	(20 pts)
Problem 2	_____	(15 pts)
Problem 3	_____	(15 pts)
Problem 4	_____	(20 pts)
Problem 5	_____	(30 pts)
Exam Total	_____	(100 pts)

Alias _____

Good Luck!

Problem 1: The execution of a MIPS code fragment on a dynamically scheduled machine is shown in the tables and in the labels on the diagram, both on the next page. The tables show the contents of the ID Register Map, Commit Register Map, and the Physical Register File at each cycle. The diagram shows the values on certain wires at certain cycles. For example, 4:65 means that at cycle 4 the labeled wire holds value 65.

The following are functional unit segment labels: Load/store, L1 L2; floating-point add, A1 A2 A3 A4; floating-point multiply, M1 M2 M3 M4 M5 M6; integer, EX. The register maps handle both integer and floating-point registers.

(*a*) Write a program consistent with these tables and labels.(12 pts)

☐ Show a pipeline execution diagram, be sure to show where each instruction commits.

☐ Choose consistent instructions.

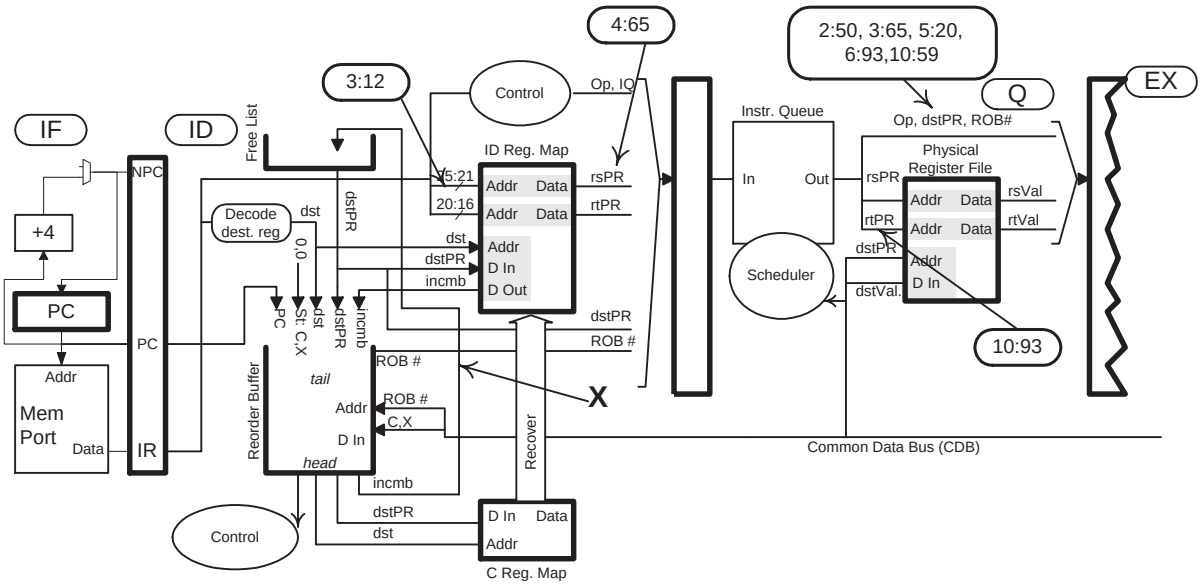
☐ Choose consistent registers. If a register number cannot be determined, use a question mark.

(*b*) Complete the tables on the next page as follows:(8 pts)

☐ Show where registers are added to, "]", and removed from, "[", the free list.

☐ Show the values on the line marked `X` in the illustration.

Problem 1, continued: See previous page for instructions.




```
# Cycle              0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16

                     IF ID

                        IF ID

                           IF ID

                              IF ID

                                 IF ID

                                    IF ID

# Cycle              0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16

X (See Diagram)
# ID Map       Cycle  0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16
f12    7                50    93    59
r5     3                   65
f10   10                         20
# Commit Map   Cycle  0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16
f12    7                                  50          93          59
r5     3                                     65
f10   10                                                 20
# Phys. Reg. File  Cy 0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16
3              0
7              0.
10             0.
20                                           0.
50                                        0.
59                                                                  0.
65                                  100
93                                                       0.
# Cycle              0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16
```

Problem 2: The diagram below shows the branch outcome patterns for a branch. (15 pts)

```
BIGLOOP:

B1: 0x1000 beq $t1, $t2, SKIP1    N  N  N  T  T  N  N  N  T  T  N  N  N  T  T
...
B2: 0x1020 beq $v0, $v1, SKIP2
...
    0x2010 j BIGLOOP
```

☐ How accurately would branch B1 be predicted by a bimodal (one-level) branch predictor with a 2^{14}-entry branch history table?

☐ How accurately would branch B1 be predicted by a local history predictor with a 10-bit local history and a 2^{14}-entry branch history table?

☐ What is the minimum local history size needed to predict B1 with 100% accuracy (after warmup). No partial credit without an explanation.

☐ Find a pattern for branch B2 that will reduce the accuracy of the local predictor on branch B1. The branch history table (not to be confused with the pattern history table) remains 2^{14} entries and the history length remains 10 bits.

Problem 3: Answer the following load/store unit questions.

(*a*) Why don't store instructions write to the cache until they commit? (5 pts)

For the two problems below consider the four instructions (repeated) which are in the reorder buffer of a dynamically scheduled 1-way system. On a cache miss data will arrive in four cycles or more. The effective address for the second load is `0x1000`. (10 pts)

(*b*) Describe a scenario in which the data for address `0x1000` is not cached but the second load does not wait for its data.

☐ Show a pipeline execution diagram.

☐ Explain the involvement of the load/store queue and why the second load does not wait more than a cycle or two.

```
First:  lw $t1, 0($t2)

        addi $t0, $0, 4720

        sw 0($t1), $t0

Second: lw $t3, 0($t4)
        Eff. Addr is 0x1000
```

(*c*) Describe a scenario in which the data for address `0x1000` is cached but the second load waits for its data at least four cycles.

☐ Show a pipeline execution diagram.

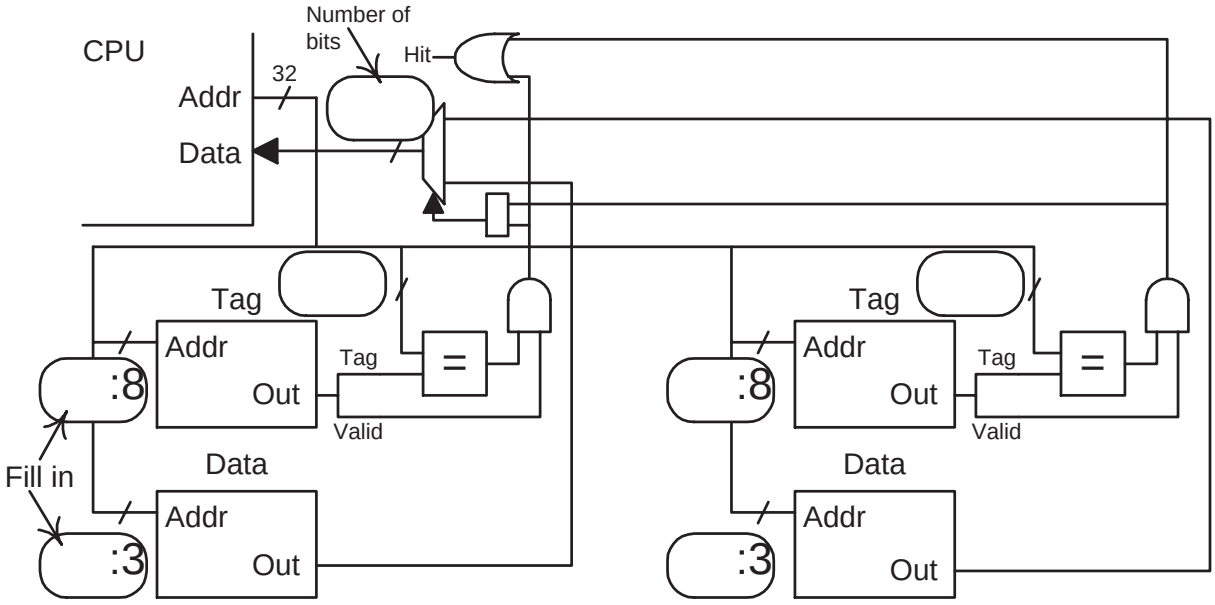
☐ Explain the involvement of the load/store queue and why the second load waits.

```
First:  lw $t1, 0($t2)

        addi $t0, $0, 4720

        sw 0($t1), $t0

Second: lw $t3, 0($t4)
        Eff. Addr is 0x1000
```

Problem 4: The diagram below is for a 2-MiB (2^{21} bytes) cache on a system with 8-bit characters.

(*a*) Answer the following, formulæ are fine as long as they consist of grade-time constants. (7 pts)

☐ Fill in the blanks in the diagram.



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

Address:

☐ Associativity:

☐ Memory Needed to Implement (Indicate Unit!):

☐ Show the bit categorization for a four-way set-associative cache with the same capacity and line size.

Address:

Problem 4, continued:

(*b*) The code below runs on the cache from the previous part. When the code below starts running the cache is empty. Consider only accesses to the array. (7 pts)

```
char *a = 0x1000000;  // sizeof(char) = 1 character
int sum, i, j;
int ILIMIT = 1024;

for(j=0; j<2; j++)
  for(i=0; i<ILIMIT; i++)
    sum += a[ i * 2 ];
```

☐ What is the hit ratio for the program above?

☐ What is the minimum value of ILIMIT needed to fill the cache?

(*c*) The code below runs on the same cache as parts above. Initially the cache is empty; consider only accesses to the arrays. (6 pts)

☐ Choose values for `KLIMIT`, `MLIMIT`, `KSHIFT`, and `MSHIFT` so that array `matrix` is completely removed from the cache with the **minimum** number of accesses (to `b`). That is, each `j` iteration must begin with `matrix` reloaded. Don't forget that the cache is set associative.

```
int  *matrix = 0x1000000;    // sizeof(int) = 4 characters
char *b      = 0x2000000;
int sum, dummy, i, j, k, m;

int KLIMIT =                                    int MLIMIT =

int KSHIFT =                                    int MSHIFT =

for(j=0; j<3; j++)
 {
    for(i=0; i<1024; i++)
      sum += matrix[ i ];

    for(k=0; k<KLIMIT; k++)
      for(m=0; m<MLIMIT; m++)
        dummy += b[ ( k << KSHIFT ) + ( m << MSHIFT ) ];
 }
```

Problem 5: Answer each question below.

(*a*) Suppose the instructions below are being considered for the latest extension of the MIPS ISA. For each new instruction explain why it should be added or why it should not be added. Consider a *variety* of factors related to the ISA and implementation. (10 pts)

☐ A mask instruction. Based on analysis of benchmarks.

```
New Instruction
 mask $t1, $t2, 5

Equivalent Code Using Existing Instructions
 srl $t1, $t2, 5
 sll $t1, $t1, 5
```

☐ An integer negate instruction. Based on analysis of existing code. With this extension `sub` does not have to be used for negation!!!

```
New Instruction
 neg $t1, $t2

Equivalent Code Using Existing Instructions
 sub $t1, $0, $t2
```

☐ An indirect load. Useful based on most existing benchmarks.

```
New Instruction
 lwi $t1, 0($t2)

Equivalent Code Using Existing Instructions
 lw $t1, 0($t2)
 lw $t1, 0($t1)
```

☐ Added functionality for the `sllv` instruction. The existing `sllv` looks at the low 5 bits of the `rt` register, ignoring the other bits. It only shifts left. The improved instruction also shifts right if the `rt` register holds a negative value and left if it's positive.

```
Improved Instruction
  sllv $t1, $t2, $t3   # Shifts right if $t3 negative.

Equivalent Code Using Existing Instructions
  bltz $t3, SHIFTRIGHT
  nop
  sllv $t1, $t2, $t3
  j DONE
SHIFTRIGHT:
  sub $at, $0, $t3
  srlv $t1, $t2, $at
DONE:
```

(*b*) Answer the following questions about exceptions. (5 pts)

☐ Why are precise exceptions necessary for instructions like `lw` but optional for instructions like `div`?

☐ What can handlers for instructions that raise precise exceptions do that handlers for other instructions cannot? Explain how that capability is used for `lw`.

(*c*) Delayed branches are common in RISC ISAs. (5 pts)

☐ Explain why delayed branches were useful in early RISC processors, such as the 1-way statically scheduled MIPS implementation covered in class.

☐ Why are delayed branches of less benefit with more recent implementations?

(*d*) Why might a gshare branch predictor with a longer global history register (GHR) have a significantly longer warm up time? (5 pts)

(*e*) Why might a functional unit with an initiation interval of 4 and latency of 3 be less costly (as in dollars) than a functional unit with an initiation interval of 1 and a latency of 3?(5 pts)

☐ Include an illustration with your answer.