Name ______________________________

Computer Architecture
EE 4720
Final Examination

10 May 2005,   15:00–17:00 CDT

Problem 1	________	(25 pts)
Problem 2	________	(15 pts)
Problem 3	________	(20 pts)
Problem 4	________	(20 pts)
Problem 5	________	(20 pts)
Exam Total	________	(100 pts)

Alias ______________________________

Good Luck!

Problem 1: (25 pts) The `madd.s fd, fr, fs, ft` instruction writes floating-point register `fd` with (`fr` × `fs`) + `ft`. The `fr` field is in bits 25:21, the other fields are in their usual places. The instruction is used in the code below:

```
# With ordinary instructions:
 mul.s f1, f2, f3
 add.s f1, f1, f4

# With a madd.s instruction:
 madd.s f1, f2, f3, f4
```

(*a*) Add datapath connections (including any connections to the register file) to the implementation on the next page (also shown below) to implement the `madd.s` instruction. The code below should execute as shown (pay attention to `f4`).

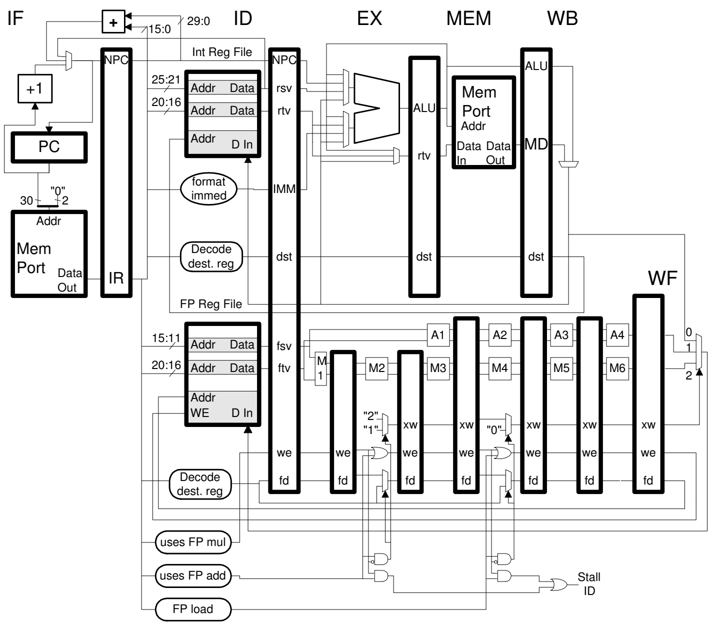
```
 madd.s f1, f2, f3, f4 IF ID M1 M2 M3 M4 M5 M6 A1 A2 A3 A4 WF
 lwc1 f4, 0(r2)           IF ID EX ME WF
 add.s  f5, f6, f7           IF ID A1 A2 A3 A4 WF
```

- Use the existing multiplier and adder.
- It should still be possible to execute ordinary floating point multiply and add instructions.

(*b*) Modify the logic so that `xw`, `we`, and `fd` work correctly for the `madd.s` instruction (and continue to work correctly for existing instructions).

(*c*) Without a `madd.s` instruction there is no possible structural hazard on `WF` in the implementation below for multiply because multiply is the longest latency instruction implemented. With `madd.s` that's no longer true, add control logic to detect this new structural hazard and generate the Stall ID signal.

SOLVE PROBLEM ON THE NEXT PAGE.



SOLVE PROBLEM ON THE NEXT PAGE.

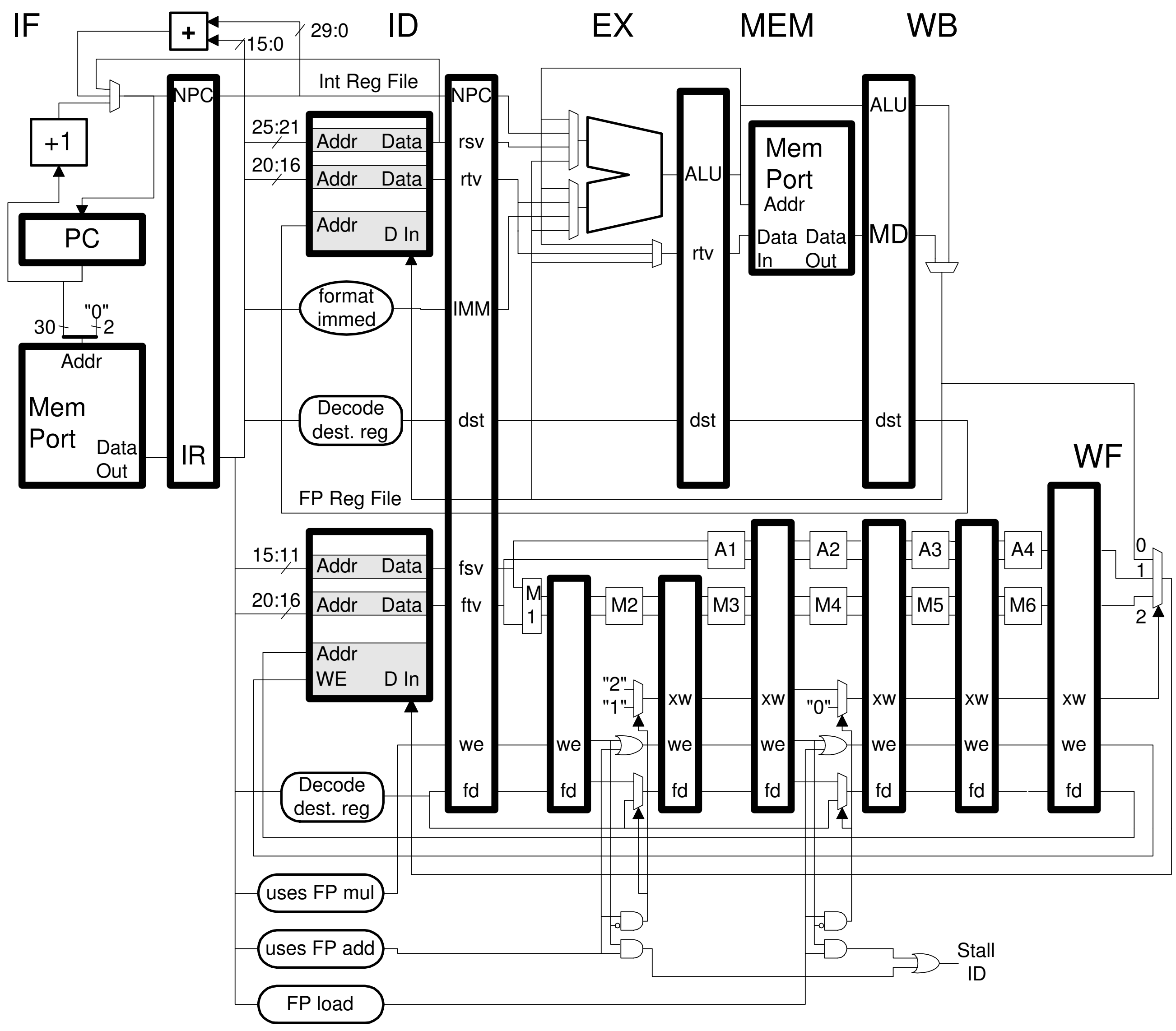
Problem 1, continued:

☐ Datapath for `madd.s`, don't break `add.s` or `mul.s`.

☐ Code on previous page must run as shown.

☐ Modify `xw`, `we`, and `fd` for `madd.s`, don't break other instructions.

☐ Detect and handle new structural hazard when `mul.s` in ID.



Problem 2: (15 pts) The execution of a MIPS program on a dynamically scheduled system using method 3 appears below. Complete the ID Register Map, Commit Register Map, and Physical Register File tables for registers `f2` and `f4`.

- The initial value of `f2` is 2.0, the mul writes 2.1, ldc1 writes 2.2, and sub.d writes 2.3. The initial value of `f4` is 4.0, the add writes 4.1. Make up physical register numbers as needed.

☐ As always, show table contents.

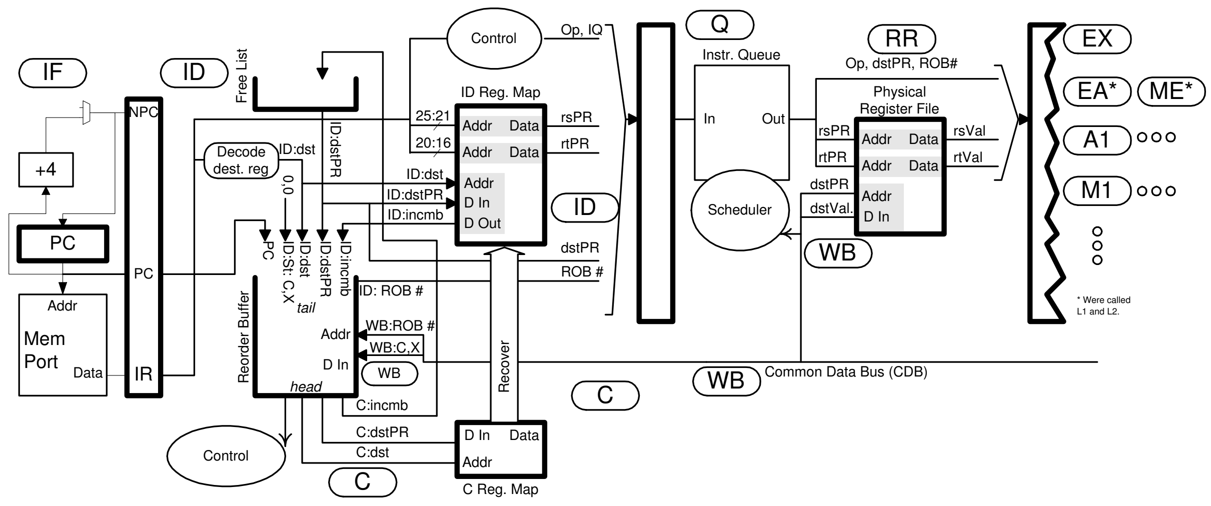
☐ Show where registers are removed from and placed back in the free list.

☐ Show initial values.

```
 # Cycle            0    1    2    3    4    5    6    7    8    9    10   11   12   13   14
 mul.d f2, f4, f6   IF   ID   Q    RR   M1   M2   M3   M4   WF   C
 add.d f4, f2, f10       IF   ID   Q                   RR   A1   A2   A3   WF   C
 ldc1 f2,0(r1)                IF   ID   Q    EA   ME   WF                            C
 sub.d f2, f2, f8                  IF   ID   Q    RR   A1   A2   A3   WF                  C

 # Cycle            0    1    2    3    4    5    6    7    8    9    10   11   12   13   14
ID Register Map











 # Cycle            0    1    2    3    4    5    6    7    8    9    10   11   12   13   14
Commit Register Map











 # Cycle            0    1    2    3    4    5    6    7    8    9    10   11   12   13   14
Physical Register File











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

HARDWARE SHOWN ON NEXT PAGE.

Problem 2, continued: Dynamically scheduled processor shown for reference.



Problem 3: (20 pts) The code below runs on three systems, one using a bimodal predictor (B) with a 2^{10}-entry BHT, a local predictor (L) with a 2^{10}-entry BHT and a 16-bit local history, and a global predictor (G) with a 16-bit GHR. The outcomes of B2 are shown, pay attention to where B1's outcomes would be located.

```
LOOP:
SHORT:
# B1 Iterates 10 times
B1: bne r3,r4 SHORT
addi r4, r4,1
...
B2: beq r1, r2 SKIP     N    N    T    N    T    N    N    T    N    T ...
...
SKIP:
j LOOP
nop
```

(*a*) Find the prediction accuracy of each predictor on each branch. Briefly explain.

☐ Accuracy predicting B1 on B:

☐ Accuracy predicting B1 on L:

☐ Accuracy predicting B1 on G:

☐ Accuracy predicting B2 on B:

☐ Accuracy predicting B2 on L:

☐ Accuracy predicting B2 on G:

(*b*) Determine the number of table entries used by branch B2 on each predictor:

☐ Entries for B2 on B:

☐ Entries for B2 on L:

☐ Entries for B2 on G:

(*c*) Suppose the history sizes were reduced.

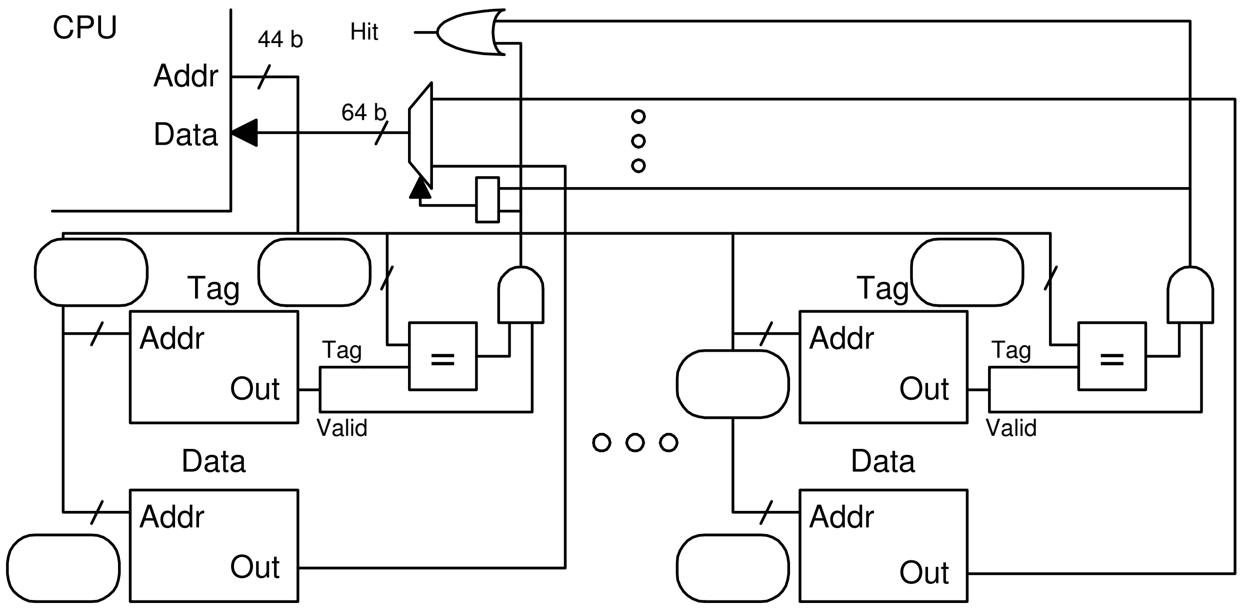
☐ What is the smallest local history size for which the accuracy on B2 will be unchanged (from the previous answer) when using the local predictor. Briefly explain.

☐ What is the smallest global history size for which the accuracy on B2 will be unchanged when using the global predictor. Briefly explain.

Problem 4: The diagram below is for a 256-kiB (2^{18} byte) 4-way set-associative cache with a line size of 16 characters on a system with 8-bit characters. (20 pts)

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

☐ Fill in the blanks in the diagram. *Pay attention to the width of the CPU address port.*



☐ 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 4, continued:

(*b*) The code below runs on the same cache as the first part of this problem. Initially the cache is empty; consider only accesses to the array.

☐ What is the hit ratio?

```
  double sum = 0.0, *a = 0x2000000;  // sizeof(double) = 8 characters
  int i, j, ILIMIT = 0x1000000;

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

The code in the problems below is to be run on a cache of unknown configuration, though it will be one of the types discussed in class (direct mapped or set-associative). In all cases the cache is empty when the program starts. Routine `get_miss_count()` *returns the number of cache misses at the time of the call (and does not cause any misses).*

(*c*) Complete the code so that `line_size` is assigned the correct line size based on the number of misses encountered and the nature of the code. Assume that all misses are due to access to `a`.

```
int ILIMIT = 0x10000;
char *a;
int miss_count_before = get_miss_count();
for(i=0; i<ILIMIT; i++) sum += a[i];
int miss_count_during = get_miss_count() - miss_count_before;

int line_size =                                         // FILL IN
```

(*d*) Complete the code so that `associativity` is aptly assigned. Assume that the cache is smaller than 256 MiB (2^{28})(but the exact size is not known), that the associativity is no larger than 64, and that there is a 64-bit address space.

```
int ISHIFT =                                            // FILL IN

int JSHIFT =                                            // FILL IN

char *a;  // Pointer to a really large array.
int miss_count_before = get_miss_count();

for(i=0; i<64; i++) sum += a[ i << ISHIFT ];

for(j=                          )                        // FILL IN
   sum += a[ j << JSHIFT ];

int miss_count_during = get_miss_count() - miss_count_before;

int associativity =                                      // FILL IN
```

Problem 5: Answer each question below.

(*a*) Suppose in a five-stage statically scheduled MIPS implementation (like the one in class) an instruction could raise an exception in the WB stage. Explain why that would make precise exceptions for that instruction impossible. Use a code example to explain what should happen for a precise exception and why its impossible (or very difficult) if the exception is raised in WB. (5 pts)

(*b*) Arrange the ISA families below in order by code (program) size, the one for which programs are smallest should be first. Starting at the second ISA in the arranged list, explain why code size is larger than the ISA above. (That is, provide three reasons why code size is larger.) (5 pts)

ISA families (in alphabetical order): CISC RISC Stack VLIW

☐ ISA families in code size order.

☐ Reasons for size differences.

(c) Deciding on a line size for a cache is a tough decision. (5 pts)

☐ Describe the behavior of programs that run better on caches with smaller line sizes.

☐ Describe the behavior of programs that run better on caches with larger line sizes.

(d) Answer the following question on cost.(5 pts)

☐ Name two parts of an n-way superscalar processor that cost about n times as much as a comparable scalar processor.

☐ Name two parts of an n-way superscalar processor that cost about n^2 times as much as a comparable scalar processor.