Name ______________________________

# Computer Architecture
# EE 4720
# Final Examination

7 May 2009, 17:30–19:30 CDT

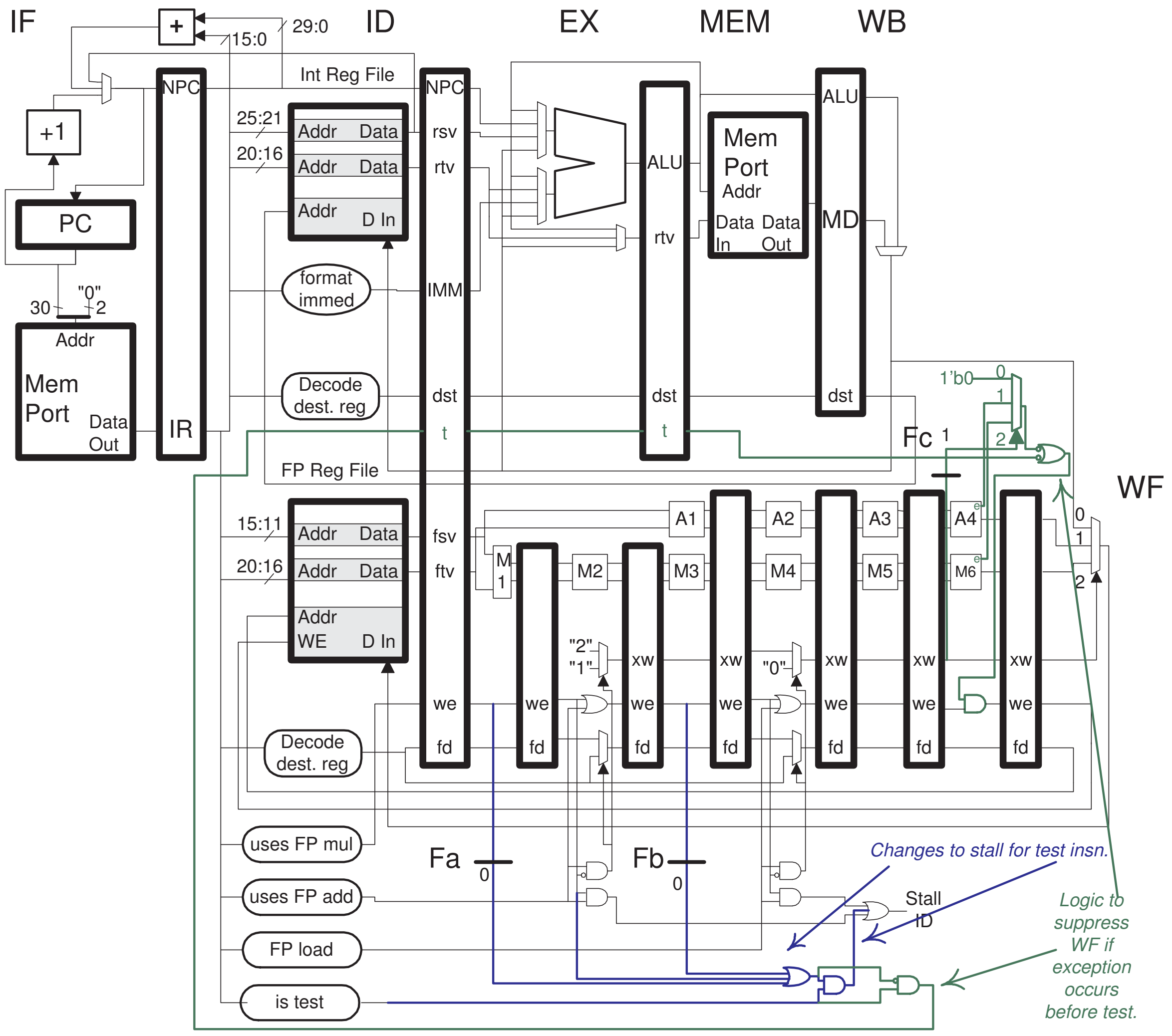
| | | |
|---|---|---|
| Problem 1 | ________ | (20 pts) |
| Problem 2 | ________ | (20 pts) |
| Problem 3 | ________ | (20 pts) |
| Problem 4 | ________ | (20 pts) |
| Problem 5 | ________ | (20 pts) |
| Exam Total | ________ | (100 pts) |

Alias ______________________________

*Good Luck!*

Problem 1: (20 pts) The MIPS implementation below, taken from the solution to last semester's final, includes hardware to implement an exception test instruction.

Several wires on the diagram are broken with heavy solid lines and marked with `Fx` and a value (mostly `0`). These indicate *potential* fault locations. If there is no fault the wire acts normally, if there is a fault the wire is broken at the heavy line and the free half takes on the indicated value. For example, if fault `Fa` is present the bottom input to the OR gate is always zero however the `ID/EX.we` signal on the other side is unaffected.



The problem here is to detect which fault, if any, is present by running test programs. One test program, and a pipeline diagram, appears below. A handler has been set up that will set a `goodException` variable to 1 if register and memory values are as expected, otherwise it is set to -1. The `goodException` variable is initialized to 0 before each test.

Problem 1, continued:

```
# Test 1: The mul should raise a precise exception.
#         Initial: f0 = 1;  r1 = 20;  Mem[r2] = 50;   f2 * f4 = NaN
# Cycle            0  1  2  3  4  5  6  7  8  9
Many nops.
mul.s f0, f2, f4   IF ID M1 M2 M3 M4 M5 M6x
test                  IF ID -------> EX MEx
sw r1,0(r2)              IF -------> ID EXx
```

(*a*) When a test is run the exception handler is called (because the tests intentionally raise an exception). A handler is shown below, written in C, but the handler does not set `goodException` correctly (not even close). Modify the code so that `goodException` is set correctly based on the information provided by the Test 1 code and comments. That is, `goodException` is set to -1 if the exception could not be precise.

```
int handler_fp(Regs *regs){
 // Code below wrong, but shows how to read registers and memory.
 if ( regs->f10 == regs->f12 && is_nan(f14) && MemW(regs->r31) == 0x1234 )
  goodException = 1; else goodException = -1;
```

(*b*) Suppose Test 1 is run and `goodException` is set to -1 (it would be 1 in the no-fault case). Which of the faults (Fa, Fb, or Fc) could have been responsible for the `goodException` value?

☐ Faults that are definitely present. Explain.

☐ Faults that could be present. Explain.

## Problem 1, continued:

(*c*) Develop tests to determine for certain whether each of the faults is present. Test 1 and each of your tests will be run, and based on the `goodException` values from each test one can say for certain which faults are present.

Assume that at most one of the faults is present. Your tests should look similar to Test 1.

☐ Tests (Code like Test 1)

☐ Which combination of `goodException` values conclude `Fa` for certain.

☐ Which combination of `goodException` values conclude `Fb` for certain.

☐ Which combination of `goodException` values conclude `Fc` for certain.

Problem 2: (20 pts) Answer each question below. **Be sure to check each code fragment carefully for dependencies.**

(*a*) The loop below runs on a statically scheduled 4-way superscalar MIPS implementation.

☐ Show a pipeline execution diagram.

```
LOOP:
addi r2, r2, 8

lw r1, 0(r2)

add r3, r1, r4

bneq r2, r5 LOOP

sw r3, 4(r2)
```

☐ Determine the IPC for a large number of iterations and assuming no cache misses.

☐ Comment on the difference between the IPC and the potential IPC of the processor.

☐ Schedule the code to improve execution time.

Problem 2, continued:

(*b*) The code below (same as the previous problem) executes on a 4-way superscalar dynamically scheduled machine. Assume that branch prediction on this machine is perfect. Load instructions use the `EA` and `ME` stages, branch instructions use the `B` stage, store instruction only use `EA` (they write memory when they commit).

Though the machine is 4-way, assume an unlimited number of `WB`, `EX`, and `RR` stages.

☐ Show a pipeline execution diagram for two iterations.

```
LOOP:
addi r2, r2, 8

lw r1, 0(r2)

add r3, r1, r4

bneq r2, r5 LOOP

sw r3, 4(r2)
```

☐ Determine the IPC for a large number of iterations.

Problem 3: (20 pts) Code producing the branch patterns shown below is to run on three systems, each with a different branch predictor. All systems use a $2^{14}$-entry BHT. One system has a bimodal predictor, one system uses a local history predictor with a 10-outcome local history, and one system uses a global predictor with a 10-outcome global history.

The code has three branches, `B1`, `B2`, and `B3`. The outcome of `B1` is random, described by a Bernoulli random variable with $p = .5$ and is independent of everything. Branch `B2` has a simple $j$-iteration loop pattern ($j-1$ T's and an N) and branch `B3` is a repeating pattern of $k$ N's followed by $k$ T's (see diagram below). Also from the diagram notice that `B1` occurs just before `B2` but that a set of $j$ `B1` and `B2s` occur between each `B3`, (similar to a branch in the homework and last semester's final).

```
              A                      A                      A
     ------------------    ------------------    ------------------   ...
B1:  R  R  R  ...  R        R  R  R  ...  R        R  R  R  ...  R
B2:   T  T  T ...T   N       T  T  T ...T   N       T  T  T ...T   N
      !<---  j    --->!

B1&2:   A   A   A       A
B3:       N   N   N  ...  N   T   T   T  ...  T   N   N   N ...  N   T...
          !<---  k Ns  --->!  !<--- k Ts  ---->!  !<--- k Ns  --->!  ...
```

For the questions below accuracy is after warmup.

☐ What is the accuracy of the bimodal on `B1`?

☐ What is the accuracy of the bimodal on `B2` in terms of $j$?

☐ What is the accuracy of the bimodal on `B3` in terms of $k$?

☐ What is the accuracy of the local predictor on `B3` when $k \gg 10$ and $j < 5$ in terms of $j$ and $k$?

☐ What is the accuracy of the local predictor on `B3` when $k \gg 10$ and $j > 10$ in terms of $j$ and $k$?

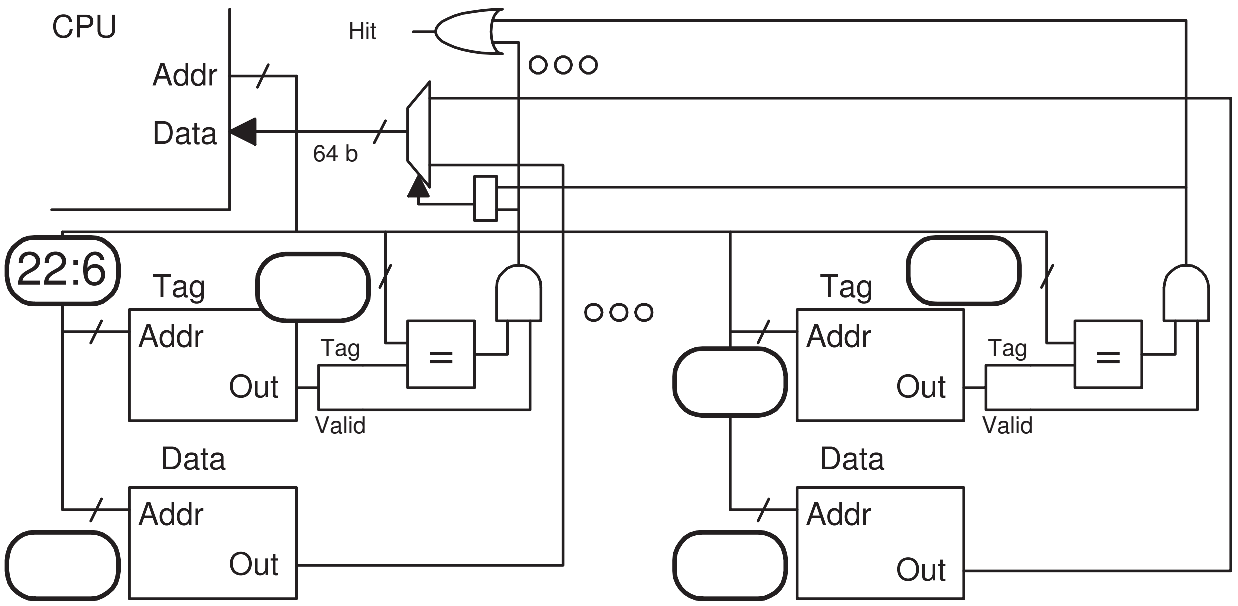
☐ What is the accuracy of the global predictor on `B3` when $j = 10$ and $k$ is very, very large, in terms of $j$ and $k$?

☐ What is the warmup time for `B3` on the global predictor. (This warmup occurs whenever `B3` switches from Ns to Ts or Ts to Ns.) in terms of $j$ and $k$?

Problem 4: (20 pts) The diagram below is for a 32-MiB ($2^{25}$-character) set-associative cache with the usual 8-bit characters and a 64-bit address space.

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

☐ 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 direct mapped cache with the same capacity and line size.

Address:

Problem 4, continued:

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

☐ What is the hit ratio running the code below? Explain

```
  double sum = 0.0;
  char *a = 0x2000000;
  int i;
  int ILIMIT = 1 << 11;    // = 2^11

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

(*c*) The code below runs on a direct-mapped cache unrelated to the one above. When the code starts running the cache is cold, for the solution only count accesses to `array`.

```
struct My_Struct {
  double val;
  double something;
  double relative;
  double more_data[29];
};  // Total size: 32 * sizeof(double) = 256 bytes

const int SIZE = 1 << 12;
My_Struct array[SIZE];  // &array[0] = 0x1000000

void tri()
{
  double sum = 0;
  for ( int i=0; i<SIZE; i++ ) sum += array[i].val;
  const double avg = sum / SIZE;
  for ( int i=0; i<SIZE; i++ ) array[i].relative += array[i].val - avg;
}
```

☐ Determine the minimum cache and line size needed so that there are no misses in the second `for` loop.

☐ Explain your answer.

Problem 5: (20 pts) Answer each question below.

(*a*) In a dynamically scheduled machine one would like to be able to have a large number of instructions in flight to, say, find something to do while waiting for data from memory. Which part of the dynamically scheduled machine is most difficult to scale up to support a large number of in-flight instructions?

☐ Part that's difficult to scale, and reason why.

(*b*) How might profiling improve the performance of the following C code:

```
if ( a > b ) { x = d / e; } else { y = q / f; }
```

☐ Explain how profiling is used.

☐ Explain why this profiled code might be faster than code compiled without profile feedback.

(*c*) In the first implementation of a company's ISA the integer multiply instruction was slow. An Engineer working on the second implementation is deciding whether to speed up the multiply instruction. To decide he plans to analyze some benchmark runs, but he can't decide whether the code should be optimized. The compiler was designed for the first implementation.

☐ What is the disadvantage of counting multiply usage in code compiled with optimization?

☐ What is the disadvantage of counting multiply usage in code compiled without optimization?

(*d*) A memory system that can fetch $2^w$ chars of data is less expensive and faster if the data address is a multiple of $2^w$. (That's one reason for the alignment restriction.)

☐ How do VLIW ISAs take advantage of that?

(*e*) Compared to a RISC implementation, say the 5-stage MIPS, what additional logic does a CISC implementation require between the IF-stage memory port output and decode?

☐ Additional logic for CISC.