Name ______________________________

Computer Architecture
EE 4720
Final Examination

7 May 2008, 10:00–12:00 CDT

Problem 1	______	(10 pts)
Problem 2	______	(30 pts)
Problem 3	______	(20 pts)
Problem 4	______	(15 pts)
Problem 5	______	(15 pts)
Problem 6	______	(10 pts)
Exam Total	______	(100 pts)

Alias ______________________________

Good Luck!

Problem 1: (10 pts) The tables below show the state of a dynamically scheduled system using method 3 during the execution of a code fragment.

- The code is preceded by many nop instructions.
- Instructions use either a 4-stage FP multiply unit (M1-M4) or a 2-stage FP add unit (A1, A2).
- Assume that there are unlimited RR, WF and execute (A,M) units.

(*a*) Show a program and pipeline execution diagram consistent with these tables.

☐ Show program and all stages used.

☐ On the physical register file show where physical registers are removed from the free list, using a [, and where they are put back in the free list, using a].

☐ All destination registers can be determined exactly.

☐ The time for all stages can be determined exactly.

# Cycle	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
	IF	ID													
		IF	ID												
			IF	ID											
				IF	ID										
# 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
f15	9			39												
f20	16		43		82	93										

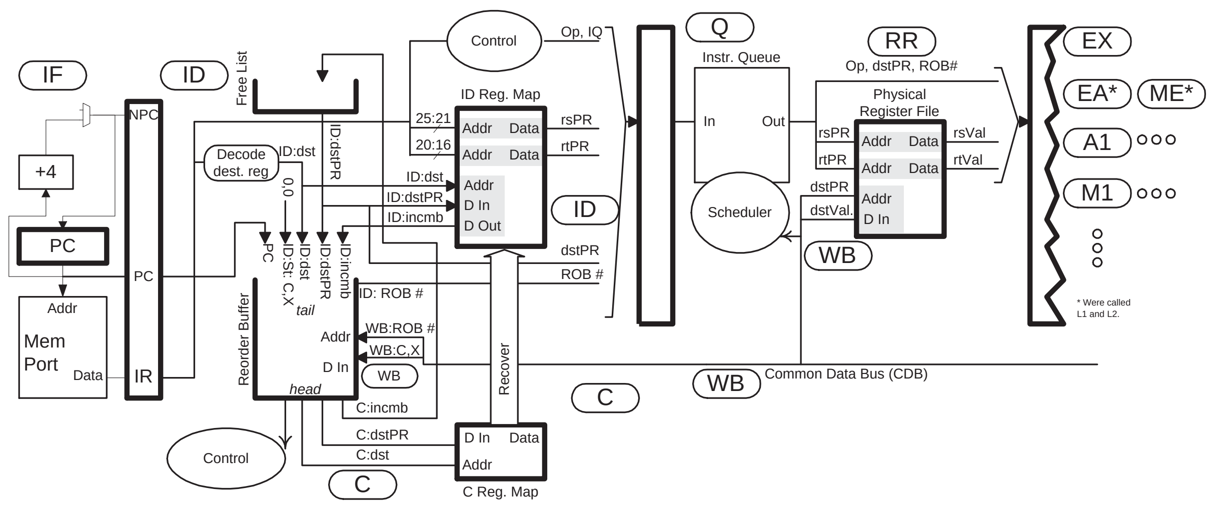
Commit Register Map

# Cycle		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
f15	9												39			
f20	16								43					82	93	

Physical Register File

# Cycle		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
9	11															
16	22															
39												33				
43								44								
82										55						
93												66				
# Cycle		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Problem 1, continued: The implementation diagram is provided below for references. Don't forget to answer the question below.



(*b*) In the code fragment on the previous page only two true dependencies are possible.

☐ Show them (by connecting the destination and source with an arrow).

☐ Briefly explain why other dependencies are not possible.

Problem 2: (30 pts) Answer the following branch predictor questions.

(a) The code below runs on two branch predictors, a bimodal predictor with a 2^{10} entry BHT and a local predictor with a 2^{10} entry BHT and a 9-outcome history.

The outcomes of branch B1 form a repeating pattern as shown below.

The outcome pattern for branch B2 can be constructed by tossing a fair coin, adding a `N N N` for heads or a `T T T` for tails, then repeating. More precisely, outcome $3i$ is a Bernoulli random variable with $p = .5$ and outcomes $3i + 1$ and $3i + 2$ match outcome $3i$, for $i = 0, 1, 2, \ldots$. For example, the following is a possible pattern of outcomes for B2: `N N N T T T T T T`. The following is **not possible** for B2: `N N N T T N T T T`, it's not possible because the number of consecutive N's or T's must be a multiple of 3.

```
LOOP:

 B1:      N  N  N  N  T  T  T  N  N  N  N  T  T  T  N  N  N  N  T  T  T...

 B2:       N  N  N  T  T  T  T  T  T  N  N  N  T  T  T  N  N  N  (See text.)

 j LOOP
 nop
```

For the questions below assume all tables are initially zero. All accuracies are after warmup.

☐ What is the accuracy of the bimodal predictor on branch B1?

☐ What is the accuracy of the local predictor on B1?

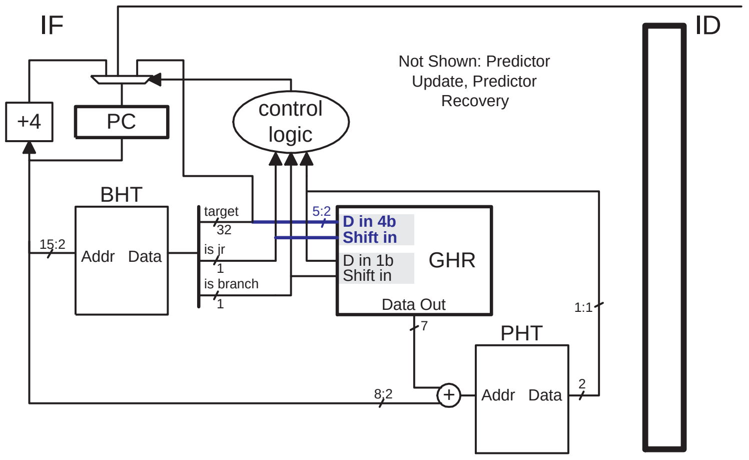
☐ What is the minimum local history size needed to predict B1 with 100% accuracy ignoring branch B2. Briefly explain.

☐ What is the approximate accuracy of the bimodal predictor on branch B2? Explain.

☐ What is the approximate accuracy of the local predictor on branch B2 ignoring B1? Explain.

☐ What is the approximate warmup time for the local predictor on B2? *Note: This problem was alot more difficult to get perfectly correct than originally intended.*

(*b*) The predictor to the right is from the solution to last semester's final exam (and this semester's Homework 4). It is similar to gshare except in how the GHR is updated. Like global and gshare, when a branch is predicted the predicted outcome is shifted into the GHR, but when a `jr` is predicted four bits of the target are shifted into the GHR.



The code fragment below is the one used to justify the predictor, except that now the code is in a four-iteration loop. Important assembler code is shown in the comments.

```
for ( iter=0; iter<4; iter++ ) {
  int c = getchar(); // Unpredictable
  switch (c) {                    // jr $t0  # Jump to correct case.
  case 'a': x = 3; j++; break;    // addi $t1, $0, 3;  j ENDSWITCH;  addi $t2, $t2, 1
  case 'b': x = 7; break;
    ...
  case 'z': x = 1; i++; break;
  }                               // ENDSWITCH:
  if ( x < 5 ) foo(); else bar();
}
```

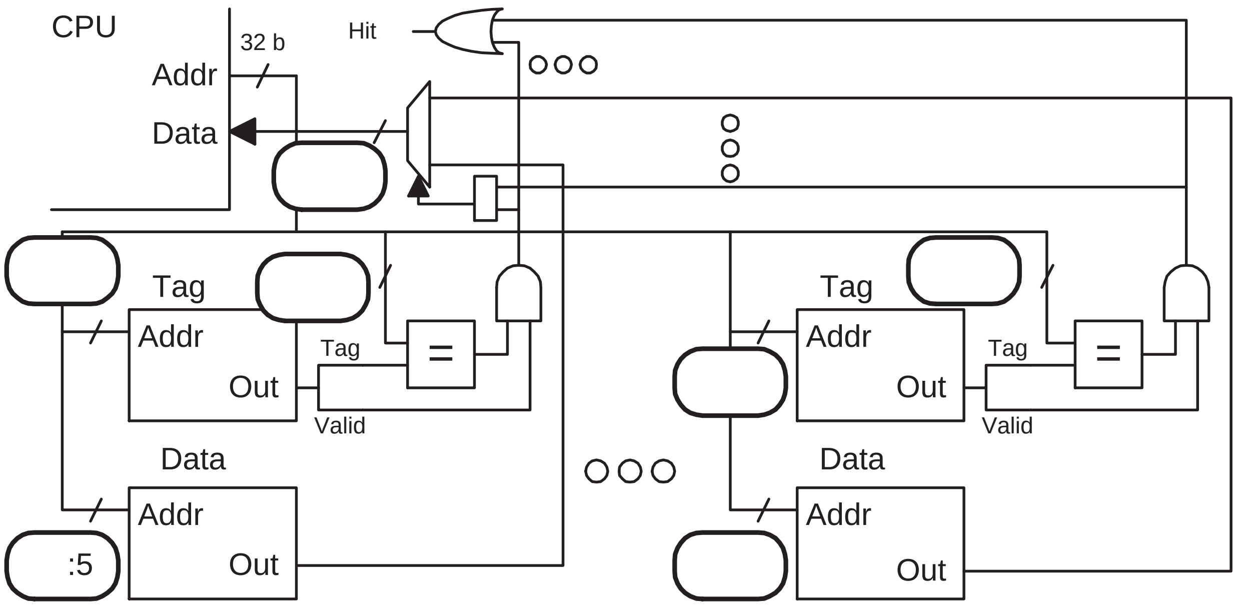
☐ Why might the prediction accuracy of the `for` loop branch be worse with the predictor above than an ordinary gshare?

☐ Considering the assembler code above, why might it make more sense to shift in the PC of the jump (`j`) instructions rather than the target of the `jr`? *Hint: This would only be useful when the case statements had branches.*

Problem 3: (20 pts) The diagram below is for a 32-MiB (2^{25}-character) 4-way set-associative cache with a line size of 4096 (2^{12}) characters, with the usual 8-bit characters.

(*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:

☐ 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:

(*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 running the code below? Explain

```
double sum = 0.0;
long *a = 0x2000000;    // sizeof(long) = 8 characters.
int i,j;
int ILIMIT = 1 << 10;    // = 2^10

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

(*c*) The code below runs on a fully associative cache with 2^7-byte lines, **not the same as the previous cache**. Let h denote the hit ratio of the code below for a cache size of 8 MiB (2^{23} bytes). As always, assume the cache is empty before the code starts.

```
void p4(double *a, double *b) {
  // sizeof double = 8 characters.
  double sum = 0;
  int size = 1 << 8;

  for ( int d=0; d<size; d++ )
    for ( int col=0; col<size; col++ )
      sum +=  b[ col * size + d ]  *  a[ d * size + col ];
}
```

☐ What is the minimum cache size for which the hit ratio is h? Explain.

Problem 4: (15 pts) Answer each question below.

(*a*) Describe what's wrong with each execution below.

☐ What's wrong with this execution on our usual bypassed 5-stage MIPS implementation.

```
# Cycle        0  1  2  3  4  5
lw r1, 0(r2)   IF ID EX ME WB
add r3, r1, r4    IF ID EX ME WB
```

☐ What are the two problems below with this execution on our usual scalar statically scheduled MIPS implementation?

```
add.d f0, f1, f2  IF ID A1 A2 A3 A4 ME WF
```

☐ What's wrong with this execution on a 2-way superscalar dynamically scheduled machine?

```
# Cycle       0  1  2  3  4  5  6  7  8  9
add r1,r2,r3  IF ID Q  RR EX WB C
add r4,r1,r6  IF ID Q     RR EX WB C
add r7,r8,r9     IF ID Q  RR EX WB    C
add r8,r2,r1     IF ID Q     RR EX WB    C
# Cycle       0  1  2  3  4  5  6  7  8  9
```

(*b*) Alas, it is unlikely that there will soon be 16-way, statically scheduled, superscalar implementations that anyone would want to buy. Three reasons are started below, complete them.

☐ It would be too expensive because . . .

☐ The clock frequency would be too low because . . .

☐ A few programs might realize the full potential of the processor, but most won't because . . .

Problem 5: (15 pts) Answer each question below.

(*a*) In ARM the PC is a general purpose register, `r15`. It could have been defined in ways consistent with ISA design principles, but it wasn't.

☐ Describe how the use of `r15` appears contrary to the usual goals of an ISA.

(*b*) The two code fragments below are supposed to do the same thing, the first is MIPS the second is SPARC. In both a branch is taken if a less-than comparison is true. The code would work correctly if the called subroutine returned immediately.

☐ Explain why the SPARC code may not execute as intended.

☐ Suggest a fix.

```
# MIPS Code
slt $s1, $s2, $s3       # Registers %s0-%s7 preserved.
jal some_subroutine
nop
bneq $s1, $0 SKIP1
...

! SPARC code
subcc %l2, %l3, %l1     ! Registers %l0-%l7 preserved.
call some_subroutine
nop
blt SKIP1
...
```

Hint: The following two questions are really asking about exceptions.

(*c*) A compiler writer is wondering whether dead-code elimination should remove the first assignment to x in the code fragment below (which sure looks dead, right?).

```
x = a / b;
x = a + c;
```

The guy down the hall wrote a program that included these two lines and that program would run differently if dead-code elimination removed the first assignment (the one with the division). The difference has nothing to do with timing or the size of the program.

☐ Why might the program have run differently?

☐ The compiler writer wants to DTRT (do the right thing), how does he or she find out what the right thing is?

(*d*) The code fragment below may have come from the code in the previous problem.

```
div.d f0, f2, f4
add.d f0, f2, f6
```

Designers of an implementation are considering whether to avoid WAW hazards like the one above by converting the `div.d` to a nop when the `add.d` reaches `ID`. This will work correctly under ordinary circumstances.

☐ Show a pipeline execution diagram illustrating the WAW hazard.

☐ Under what circumstances will this produce the wrong answer?

Problem 6: (10 pts) SPECcpu provides a separate set of training inputs to be used for feedback-directed optimization (FDO) techniques, such as profiling.

☐ Why is a separate training input set needed?

☐ Why might an honest and good tester want to substitute a different training set?

The run and reporting rules don't allow such a substitution.

☐ Why do you think the run and reporting rules don't allow a training set substitution when they allow so much flexibility on compilers, optimizations, and libraries?