Name _____

Computer Architecture
EE 4720
Final Examination

8 May 2006, 10:00–12:00 CDT

Problem 1	_____	(25 pts)
Problem 2	_____	(25 pts)
Problem 3	_____	(20 pts)
Problem 4	_____	(30 pts)
Exam Total	_____	(100 pts)

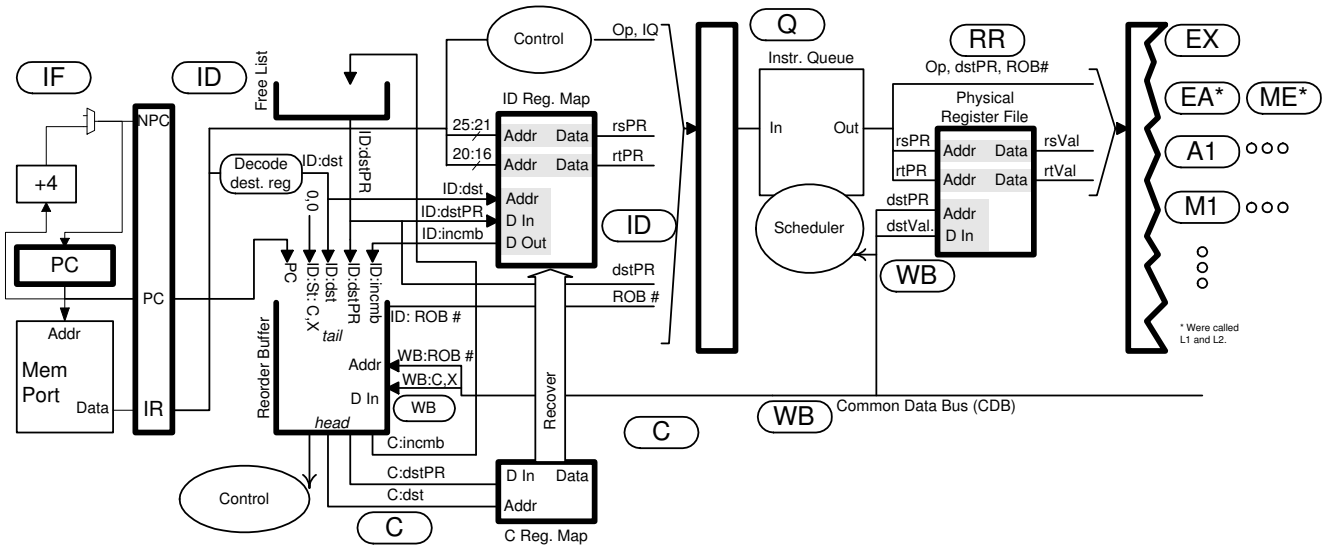
Alias _____

Good Luck!

Problem 1: The execution of code on a dynamically scheduled scalar MIPS implementation using Method 3 (the only one covered in class) is shown below. Beneath the diagram are signal labels, for example, `ID:dst`. (25 pts)

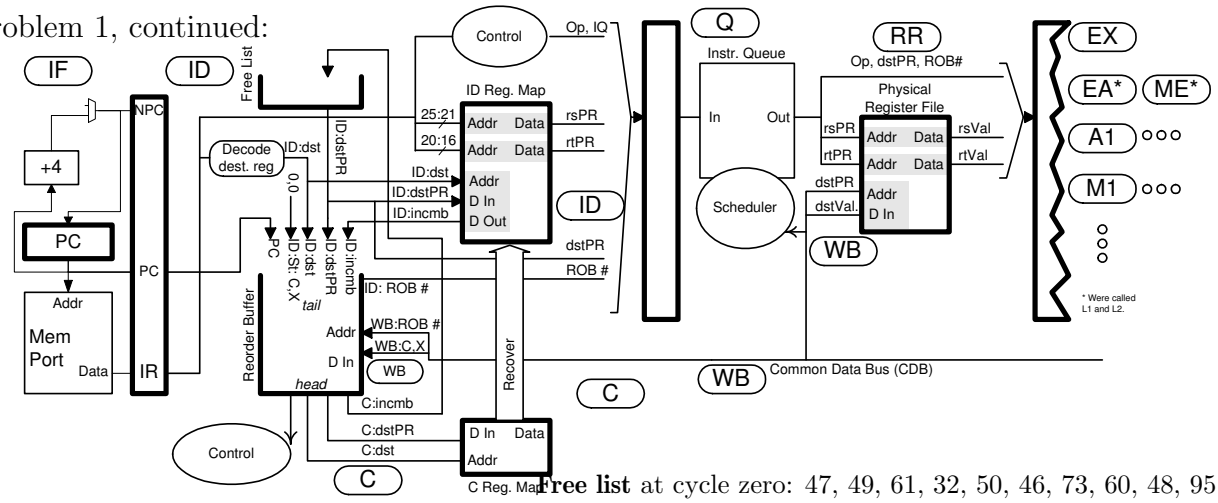
(*a*) On the next page show the values of the signals at the cycles they are used.

- Some values are shown, they are needed to determine other values.
- All values can be determined.
- Ignore the distinction between integer and FP registers.
- The free list contents at cycle zero is shown on the lower right-hand side of the figure on the next page.



Tables for answer on next page.

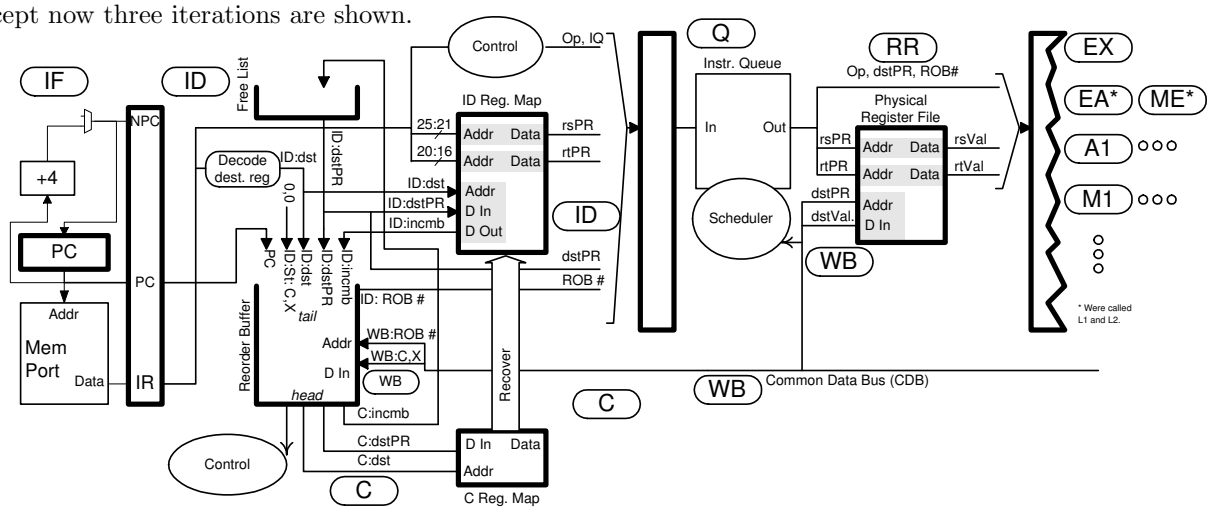
Problem 1, continued:



Free list at cycle zero: 47, 49, 61, 32, 50, 46, 73, 60, 48, 95

```
LOOP:              # First Iteration
 add.s  f2, f2, f4  IF ID Q  RR A1 A2 A3 A4 WB C
 bgtz r5  LOOP         IF ID Q  RR B  WB          C
 sub r5, r5, r6           IF ID Q  RR EX WB          C
# Cycle:            0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15
LOOP:              # Second Iteration
 add.s  f2, f2, f4           IF ID Q     RR A1 A2 A3 A4 WB C
 bgtz r5  LOOP                  IF ID Q     RR B  WB          C
 sub r5, r5, r6                    IF ID Q     RR EX WB          C
# Cycle:            0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15

ID:rsPR               56 30 30

ID:rtPR               63    54

ID:dst

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

ID:dstPR

ID:incmb

RR:rsPR

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

RR:rtPR

WB:dstPR

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

C:incmb

C:dstPR

C:dst

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

Problem 1, continued: The diagram below is for the same code on the same system as the previous part, except now three iterations are shown.




```
LOOP:            #  First Iteration
 add.s  f2, f2, f4  IF ID Q  RR A1 A2 A3 A4 WB C
 bgtz r5  LOOP         IF ID Q  RR B  WB          C
 sub r5, r5, r6           IF ID Q  RR EX WB          C
# Cycle:            0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18
LOOP:            #  Second Iteration
 add.s  f2, f2, f4           IF ID Q     RR A1 A2 A3 A4 WB C
 bgtz r5  LOOP                  IF ID Q     RR B  WB          C
 sub r5, r5, r6                    IF ID Q     RR EX WB          C
# Cycle:            0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18
LOOP:            #  Third Iteration
 add.s  f2, f2, f4                     IF ID Q        RR A1 A2 A3 A4 WB C
 bgtz r5  LOOP                            IF ID Q  RR B  WB                C
 sub r5, r5, r6                              IF ID Q     RR EX WB             C
```

(*b*) What is the CPI for a large number of iterations? *Hint: It's not 1.*

☐ CPI?

(*c*) Suppose the ROB can hold 256 entries and there are an unlimited number of physical registers. Assuming a very large number of iterations, at about what cycle will fetch stall?

☐ Fetch will stall at cycle ≈

(*d*) Would the code above run faster on a four-way superscalar dynamically scheduled system with the same clock frequency and pipeline depth? Explain.

☐ Circle One: Faster Not Faster.

☐ Because

Problem 2: The code fragments below run on three systems which are identical except for the branch predictor: One uses a **bimodal** predictor with a 2^{14}-entry BHT, one uses a **local predictor** with an 8-outcome local history and a 2^{14}-entry BHT, and one uses a **global predictor** with an 8-outcome global history. (25 pts)

(*a*) Provide the information requested below.

- All accuracies are after warmup.
- For the warmup time show the approximate number of times the predicted branch needs to be executed before the predictor reaches its warmup accuracy. Assume the 2-bit counters start out at 0 or 3, whichever is worse.

```
BIG:addi r3, r0, 3
LP: bne r3, r0 LP  # Iterates four times.
    addi r3, r3, -1

    lw r1,0(r2)
C2: beq r1, 0 SK   # T N T T N  T N T T N  T N T T N  T N T T N  ...
    nop
    nop
SK: j BIG
    addi r1, r1, 4
```

☐ Bimodal: accuracy on `C2`:

☐ Bimodal: warmup time on `C2`:

☐ Local: accuracy of `C2`:

☐ Local: warmup time on `C2`:

☐ Local: smallest history size needed for 100% accuracy on `C2`:

☐ Global: accuracy on `C2`:

☐ Global: warmup time on `C2`:

☐ Global: smallest history size needed for 100% accuracy on `C2`:

Problem 2, continued:

(*b*) The code below is similar to the code on the previous page except the four-iteration loop has been replaced by two random branches. Each random branch will be taken with probability .5 and the outcome is independent of everything, including the other random branch.

```
BIG:lb r3, 0(r4)
    beq r3, r0  S2  # Random.
    lb r3, 1(r4)
S2: beq r3, r0  S3  # Random.
    addi r4, r4, 2
S3: nop

    lw r1,0(r2)
C2: beq r1, 0  SK   # T N T T N  T N T T N  T N T T N  T N T T N  ...
    nop
    nop
SK: j BIG
    addi r1, r1, 4
```

☐ Global: accuracy on `C2`:

☐ Explain using GHR values.

☐ Global: warmup time on `C2`:

☐ Explain using GHR values.

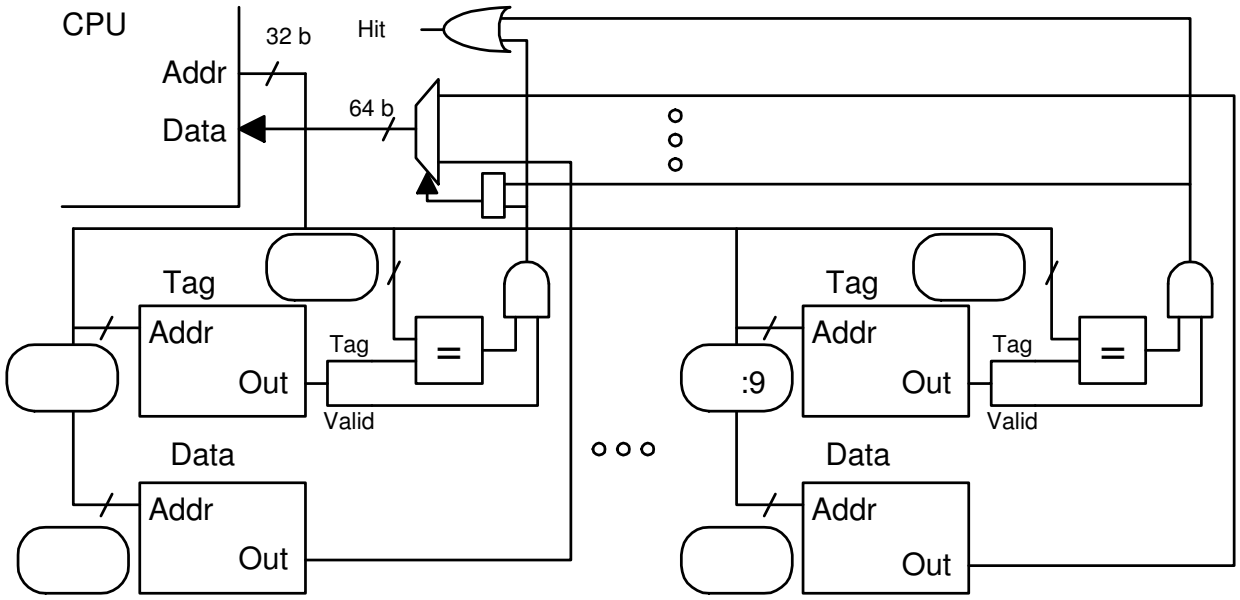
☐ Global: smallest history size needed for 100% accuracy on `C2`:

☐ Explain using GHR values.

Problem 3: The diagram below is for a 64-MiB (2^{26}-character) 16-way set-associative cache on a system with the usual 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.



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

Address:

☐ Memory Needed to Implement (Indicate Unit!!):

☐ Line Size (Indicate Unit!!):

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

Address:

Problem 3, continued: For the problems on this page use the cache from the previous page.

(*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;
char *a = 0x2000000;    // sizeof(char) = 1 character.
int i;
int ILIMIT = 1 << 27;  // = 2^27

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

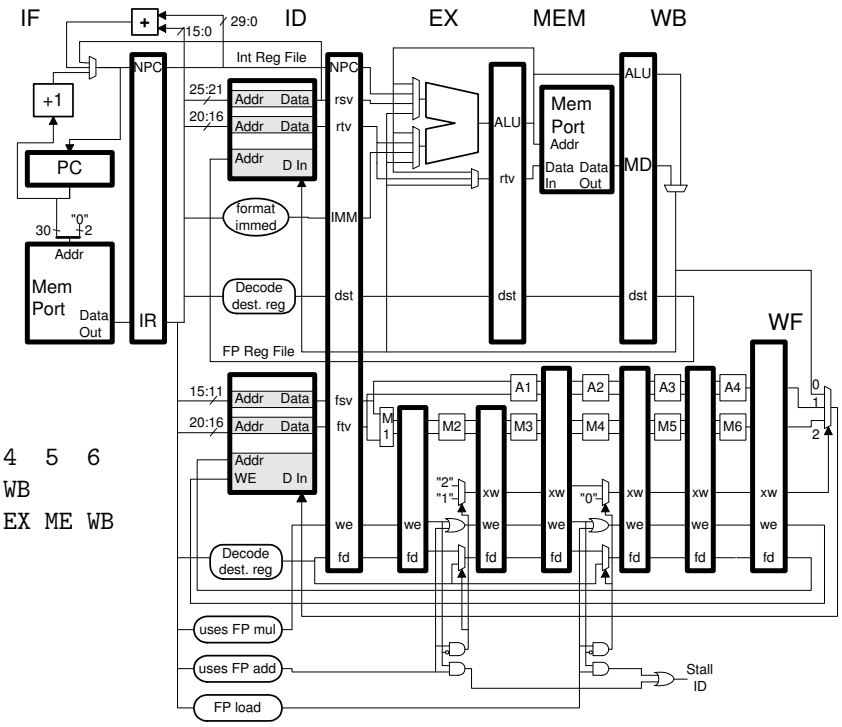
(*c*) The slightly different code below runs on the same cache as the first part of this problem. Initially the cache is empty; consider only accesses to the array.

☐ Find the hit ratio. Very briefly explain.

```
double sum = 0.0;
char *a = 0x2000000;  // sizeof(char) = 1 character.
int i;
int ILIMIT = 1 << 27;

for(i=0;        i<ILIMIT; i++) sum += a[ i ];
for(i=ILIMIT-1; i>=0;     i--) sum += a[ i ];
```

Problem 4: Answer each question below.

(*a*) The execution of some code fragments on the illustrated implementation appear below. Explain why each execution is impossible and show a corrected pipeline execution diagram. (7 pts)




```
# Cycle         0  1  2  3  4  5  6
lw r2, 0(r4)    IF ID EX ME WB
add r1, r2, r3     IF -> ID EX ME WB
```

☐ Fix.

☐ Was impossible because:

```
# Cycle         0  1  2  3  4  5  6  7
lw r2, 0(r4)    IF ID EX ME WB
add r1, r2, r3     IF ID -> EX ME WB
xor r5, r6, r7        IF ID -> EX ME WB
```

☐ Fix.

☐ Was impossible because:

```
add.d f0, f2, f4  IF ID A1 A2 A3 A4 ME WB
```

☐ Fix.

☐ Was impossible because:

(*b*) For each feature below indicate whether it is usually a feature of the ISA or the implementation, and explain why it's not a feature of the other. (7 pts)

Feature: *The opcode can be found in bits 31:26.*

☐ ISA or Implementation?

☐ Why not the other?

Feature: *The* `add` *in the code below will stall for one cycle.*

```
lw  r1, 0(r2)
add r3, r1, r4
```

☐ ISA or Implementation?

☐ Why not the other?

Feature: *Two consecutive delayed (as in MIPS) branches will yield unpredictable results.*

```
bneq r1, r2 DEST1
beq r3, r4 DEST2
addi r5, r6, r7
```

☐ ISA or Implementation?

☐ Why not the other?

Feature: *Integer addition overflow raises exception.*

☐ ISA or Implementation?

☐ Why not the other?

(*c*) Consider the use of a packed-operand 8-bit add (of the type described in class) to speed up the code fragments below. For each fragment indicate whether it's certainly feasible, feasible with certain assumptions, or not feasible. (6 pts)

```
extern char *a, *b, *c;
for(i=0; i<1024; i++) a[i] = b[i] + c[i];
```

☐ Circle One: No. Yes! Yes, assuming ...

☐ Explain.

```
extern int *a, *b, *c;
for(i=0; i<1024; i++) a[i] = b[i] + c[i];
```

☐ Circle One: No. Yes! Yes, assuming ...

☐ Explain.

```
extern char *a, *b, *c;
for(i=1; i<1024; i++) a[i] = a[i] + a[i-1];
```

☐ Circle One: No. Yes! Yes, assuming ...

☐ Explain.

(*d*) MIPS and other RISC ISAs typically have instructions using displacement addressing but lack register deferred addressing. (See the examples below.)

```
lw r1, 4(r2)   # Displacement addressing.
lw r1, (r2)    # Register deferred addressing.
```

Given that RISC and CISC ISAs have displacement addressing:

☐ (5 pts) Why is register deferred addressing helpful for CISC but not helpful for RISC?

(*e*) Dead-code elimination (DCE) is a common compiler optimization.

☐ (5 pts) What is it? Illustrate using an example.