Name _____

Computer Architecture
EE 4720
Final Examination

13 December 2005, 12:30–14:30 CST

Problem 1	_____	(15 pts)
Problem 2	_____	(20 pts)
Problem 3	_____	(17 pts)
Problem 4	_____	(15 pts)
Problem 5	_____	(33 pts)
Exam Total	_____	(100 pts)

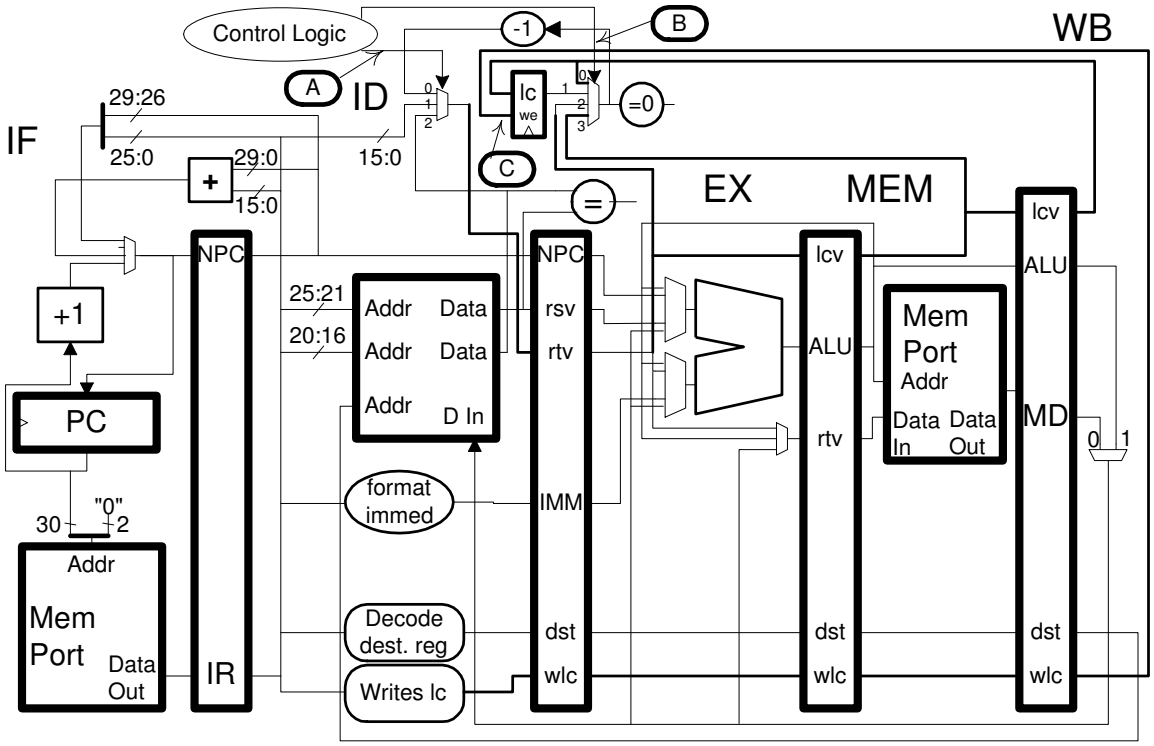
Alias _____

Good Luck!

Problem 1: The implementation of a version of MIPS with loop count instructions is shown below, these are the same instructions used in Homework 5. Instruction `mtlc rt` moves the contents of the `rt` register into a special loop count register, `lc`, and instruction `mtlci immed` moves a 16-bit immediate, `immed`, into `lc`. The loop-counted branch `bclz` is taken unless the `lc` register is zero, it also decrements `lc`. (In the diagram the upper input to the `lc` register is the data input and the lower input, `we`, is a write enable.)

Three wires in the implementation are labeled (A , B , and C). Show the values present on those wires for each cycle of the illustrated execution of the program in the space provided. Leave a value blank if the value has no effect, assume that instructions before and after the illustrated code are nops. Note that two iterations of the loop are shown. (15 pts)

```
 # Cycle            0    1    2    3    4    5    6    7    8
 mtlci 100          IF   ID   EX   ME   WB
LOOP:
 bclz LOOP               IF   ID   EX   ME   WB
 add r2, r2, r3               IF   ID   EX   ME   WB
# (2nd iteration)
 bclz LOOP                    IF   ID   EX   ME   WB
 add r2, r2, r3                    IF   ID   EX   ME   WB
 # Cycle            0    1    2    3    4    5    6    7    8
 A:                                                      <- ANSWER HERE

 B:                                                      <- AND HERE

 C:                                                      <- AND HERE TOO
 # Cycle            0    1    2    3    4    5    6    7    8
```



Problem 2: Complete the pipeline execution diagrams for the different MIPS implementations below. Pay attention to the type of implementations in each part, they're not all the same. Don't forget to **check for dependencies.**

☐ (5 pts) Scalar (not superscalar), statically scheduled, as illustrated in the previous problem.

```
# Cycle          0    1    2    3    4    5    6    7    8    9    10   11   12   13   14
add r1, r2, r3

lw r4, 6(r1)

xor r7, r4, r8
# Cycle          0    1    2    3    4    5    6    7    8    9    10   11   12   13   14
```

In all of the parts below the floating-point multiply unit has four stages and is fully pipelined, the stage labels are *M1-M4*. *The floating-point add unit is two stages and fully pipelined, the stage labels are* *A1 and A2*.

☐ (5 pts) Scalar, statically scheduled, full set of bypass paths.

```
# Cycle          0    1    2    3    4    5    6    7    8    9    10   11   12   13   14
mul.d f2, f4, f6

add.d f8, f2, f10

mul.d f2, f4, f14

sub.d f12, f2, f10
# Cycle          0    1    2    3    4    5    6    7    8    9    10   11   12   13   14
```

☐ (5 pts) Two-way superscalar, statically scheduled, full set of bypass paths. No alignment restriction on instruction fetches.

```
# Cycle          0    1    2    3    4    5    6    7    8    9    10   11   12   13   14
mul.d f2, f4, f6

add.d f8, f2, f10

mul.d f2, f4, f14

sub.d f12, f2, f10
# Cycle          0    1    2    3    4    5    6    7    8    9    10   11   12   13   14
```

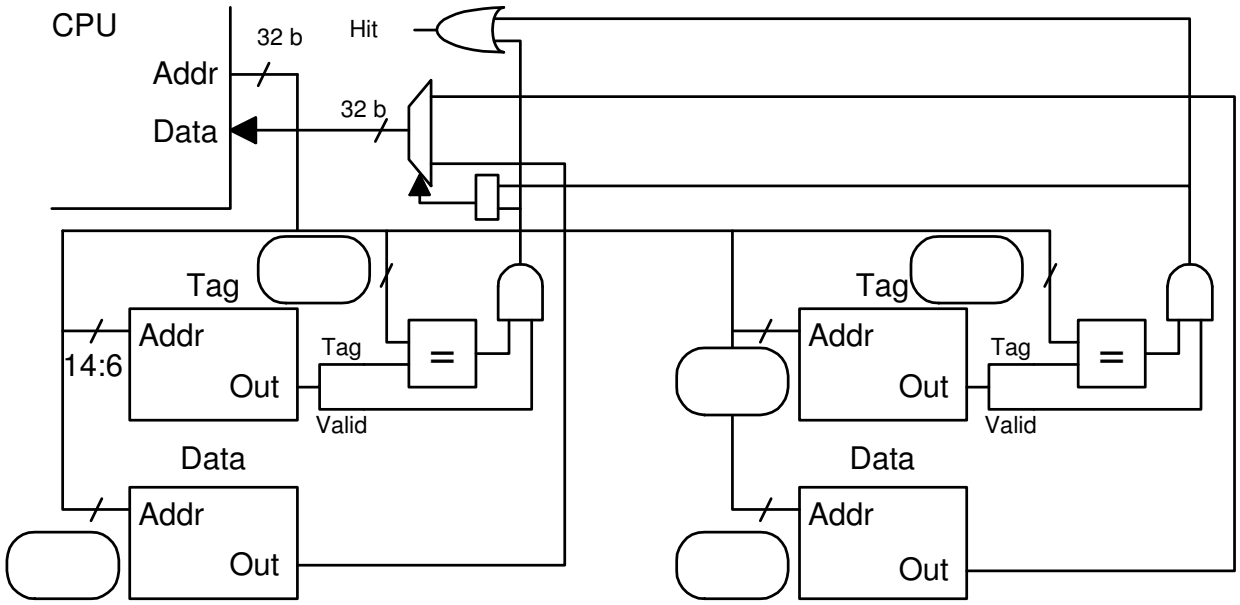
☐ (5 pts) Two-way superscalar, dynamically scheduled, with a full set of bypass paths. No alignment restriction on instruction fetches.

```
# Cycle          0    1    2    3    4    5    6    7    8    9    10   11   12   13   14
mul.d f2, f4, f6

add.d f8, f2, f10

mul.d f2, f4, f14

sub.d f12, f2, f10
# Cycle          0    1    2    3    4    5    6    7    8    9    10   11   12   13   14
```

Problem 3: The diagram below is for a cache on a system with the usual 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:

☐ Cache Capacity (Indicate Unit!!):

☐ 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.

☐ (5 pts) What is the hit ratio running the code below?

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

for(j=0; j<3; j++)
  for(i=0; i<32; 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.

☐ (5 pts) Set `ILIMIT` to the smallest value that will fill the cache.

```
double sum = 0.0, *a = 0x2000000;  // sizeof(double) = 8 characters
int i;
int ILIMIT =                      <- ANSWER HERE

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

Problem 4: Answer each question below.

(*a*) An n-way superscalar processor need not have n copies of everything that an ordinary scalar implementation has.

☐ (5 pts) Which of the following is it most important that an n-way superscalar processor **does** have n of (explain):

- Mem-stage memory ports.
- Write ports to the register file.
- Floating-point adders.

(*b*) In the dynamically scheduled MIPS implementation covered in class:(5 pts)

☐ When is a physical register allocated (removed from the free list and assigned to an instruction)?

☐ When is the physical register put back in the free list. Show an example that includes register numbers.

(*c*) Describe the contents of an Itanium (IA-64) bundle or a bundle in a typical VLIW ISA.

☐ (5 pts) Bundle Contents

Problem 5: Answer each question below.

(*a*) When a company wants to benchmark a new computer using SPECcpu it gets the benchmark suite from SPEC, compiles the code, runs the benchmarks, and proudly publishes the results. (The last step is optional.)

Below are two variations on this procedure, for each explain how the results might differ **from those obtained using the usual SPEC procedure**, and explain how useful the results would be to the typical user of SPECcpu results. (7 pts)

(1) SPEC gives a compiler (in addition to the usual material) to the company and allows the company to compile and run the suite following the usual SPECcpu rules (but using SPEC's compiler).

☐ Compared to actual SPEC rules and to (2), how might the results differ? Explain.

☐ Compared to actual SPEC rules and to (2), how useful would results be? Explain.

(2) The company gives the new computer and a compiler to SPEC and SPEC compiles and runs the benchmark suite using the usual SPECcpu rules.

☐ Compared to actual SPEC rules and to (1), how might the results differ? Explain.

☐ Compared to actual SPEC rules and to (1), how useful would results be? Explain.

(*b*) There is much less advantage in unrolling one of the loops below when the code is to run on a two-way statically scheduled superscalar MIPS implementation. *Note: The original problem did not mention the code was to run on a superscalar implementation.*(6 pts)

☐ Unroll the loop below or explain why it shouldn't be unrolled.

```
 addi r9, r0, 1024
LOOP_A:
 lw r1, 0(r2)
 lw r1, 0(r1)
 add r3, r1, r3
 addi r2, r2, 4
 bneq r9, r0 LOOP_A
 addi r9, r9, -1
```

☐ Unroll the loop below or explain why it shouldn't be unrolled.

```
 addi r9, r0, 1024
LOOP_B:
 lw r1, 0(r2)
 add r3, r3, r1
 lw r2, 4(r2)
 bneq r9, r0  LOOP_B
 addi r9, r9, -1
```

(*c*) Which are smaller, RISC programs or CISC programs. Provide some instruction examples to illustrate your answer. (5 pts)

☐ Example and explanation.

(*d*) The cost of implementing a rarely used instruction is deemed too high and so is omitted from an implementation. How can an operating system enable the implementation to run code that uses the rare instruction without modifying the code or examining it in advance. (5 pts)

(*e*) BCD data types were once popular ISA features. (5 pts)

☐ Were BCD data types absolutely necessary?

☐ What were the reasons for including BCD data types in an ISA?

☐ Why might a BCD data type be considered old fashioned and so not worthy of adding to an ISA now?

(*f*) MIPS-I is big endian but more recent MIPS versions can run in either big-endian or little-endian modes.

☐ (5 pts) Complete (possibly modifying) the MIPS program below so that it writes `v0` with a 0 if it is running on a system using big endian byte order or a 1 if running on a system using little-endian byte order.

```
# At finish: $v0: 0, if big endian;  1, if little endian.
lui $t1, 0x1122
ori $t1, $t1, 0x3344
sw $t1, 0($t2)
```