Name _____

Computer Architecture
EE 4720
Final Examination

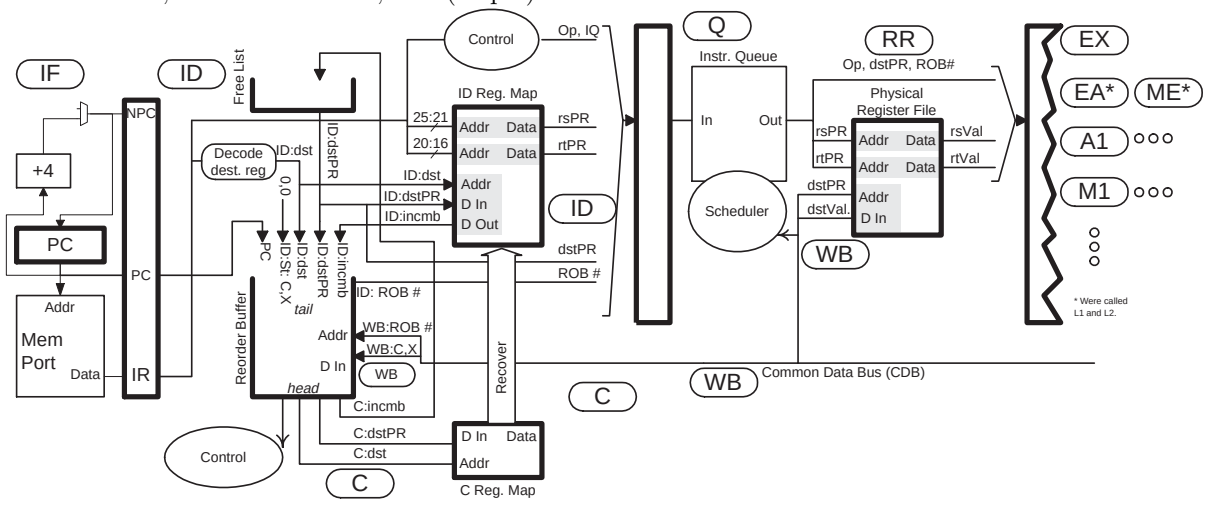
6 December 2004, 7:30–9:30 CST

Problem 1	_____	(15 pts)
Problem 2	_____	(25 pts)
Problem 3	_____	(25 pts)
Problem 4	_____	(10 pts)
Problem 5	_____	(25 pts)
Exam Total	_____	(100 pts)

Alias _____

Good Luck!

Problem 1: The MIPS code below executes as shown on the illustrated dynamically scheduled scalar implementation. There are no exceptions or recoveries; the result (value to be written in `r2`) of the first instruction is 101, the second is 102, etc. (15 pts)



- ☐ Show where each instruction commits.
- ☐ The initial value of `r1` is 111 and the initial value of `r2` is 222. Fill in the tables to show these values.
- ☐ Complete the tables for the execution of the code. Take into account the results (see first paragraph).
- ☐ Show where registers are removed from and put back in the free list.

```
# Cycle              0   1   2   3   4   5   6   7   8   9   10  11  12
lw  r2, 0(r4)        IF  ID  Q   RR  EA  ME  WB
add r1, r2, r3           IF  ID  Q       RR  EX  WB
sw r1, 0(r4)                 IF  ID  Q   RR  EA      ME  WB
xor r2, r5, r3               IF  ID  Q   RR  EX  WB
and r2, r2, r6                   IF  ID  Q       RR  EX  WB
# Cycle              0   1   2   3   4   5   6   7   8   9   10  11  12
# ID Map
```

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

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

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

Problem 2: An extended version of MIPS, called *MMMIPS*, includes memory-to-memory (MM) arithmetic instructions that can read the first source operand from memory and write a result to memory; the second source operand is always an immediate and the MM's are encoded in format I. Their mnemonics end with `.mm`, `.mr`, or `.rm` and they operate as described in the comments below. (25 pts)

```
lw     r1, 2(r3)        #  r1 = Mem[r3+2]  (Existing instruction, for your reference.)
add.mm (r4),(r5), 3     #  Mem[r4] = Mem[r5] + 3
sub.rm  r6, (r7), 3     #  r6 = Mem[r7] + 3
or.mr  (r8), r9,  3     #  Mem[r8] = r9 + 3
```

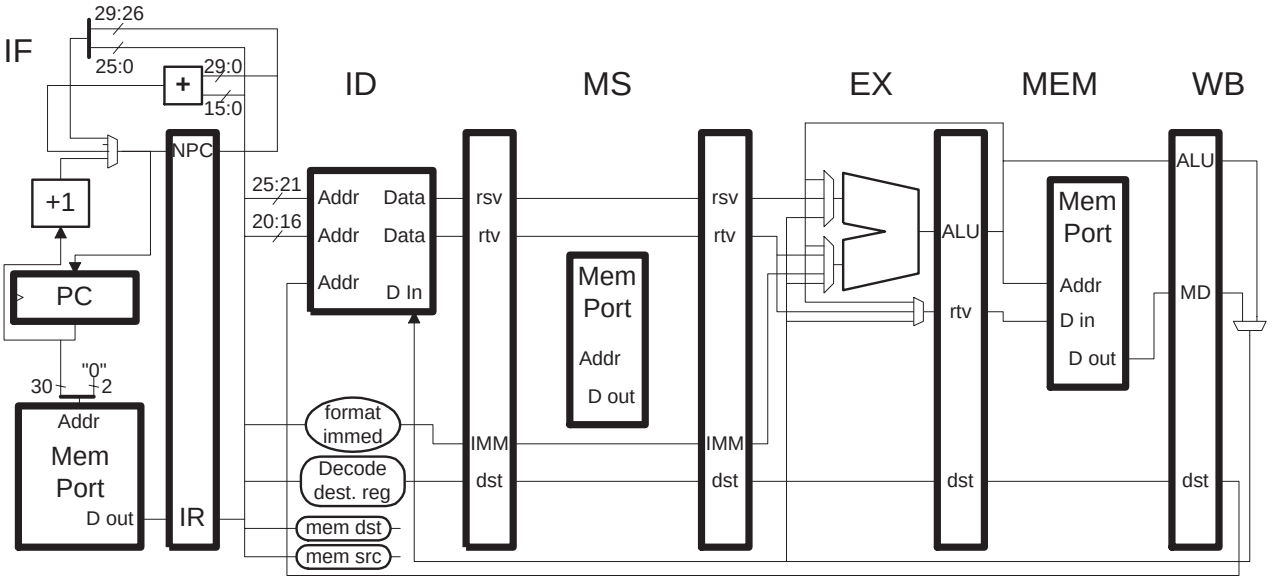
(*a*) Shown below is a partially completed implementation of MMMIPS. It includes a new stage, MS (the S is for source), that has a memory port for the source operand, shown unconnected. For use in a later part of this problem, boxes mem src and mem dst identify an instruction as having a memory source operand (output `1`) or a memory destination operand (output `1`) (an instruction can have both). An output of `0` means the respective operand is not from or to memory.

☐ Connect the MS-stage memory port for the MM instructions.

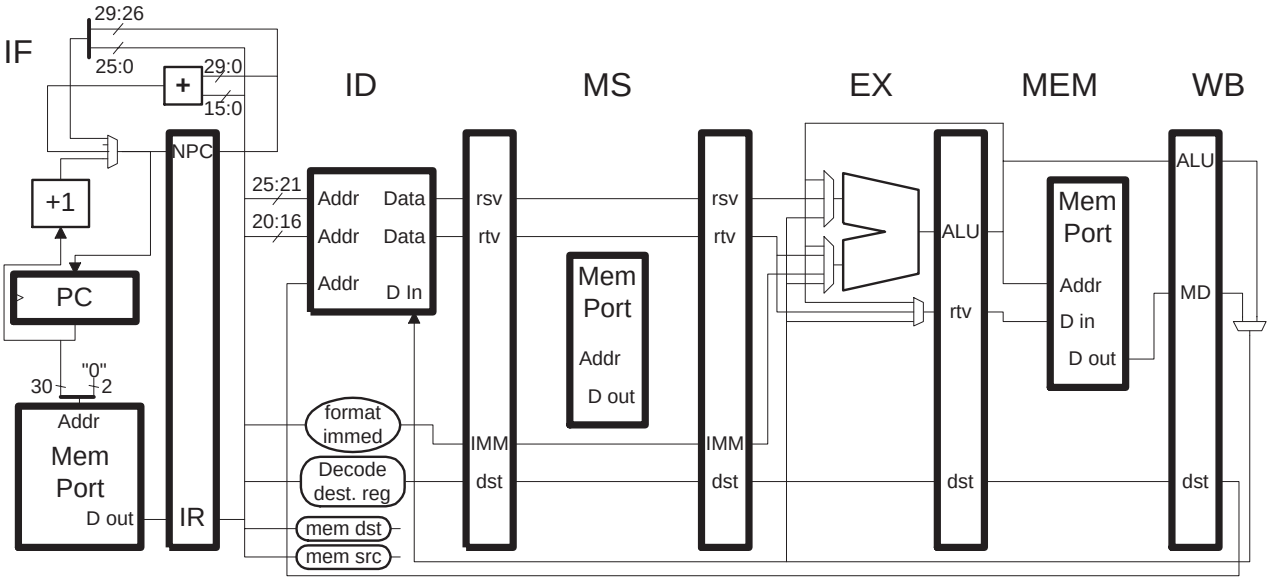
☐ Modify connections to the MEM-stage memory port for the MM instructions.

☐ Make sure existing MIPS load and store instructions continue to work correctly.

☐ For this part don't add bypassing or control logic.



For the problems below use either the diagram below or the one on the previous page.



(*b*) Add a bypass connection so that the code below can execute without a stall or show which existing bypass connection can be used.

☐ Add or identify the connection.

☐ Show which cycle the bypass connection is used.

```
# Cycle              0  1  2  3  4  5  6
add     r1, r2, r3   IF ID MS EX ME WB
sub.mr (r1), r4, 5      IF ID MS EX ME WB
```

(*c*) Can a bypass connection be added for the code below? If yes, should it be added? Explain.

☐ Show how and/or explain why not.

```
# Cycle              0  1  2  3  4  5  6
add    r1, r2, r3    IF ID MS EX ME WB
sub.rm r6, (r1), 5      IF ID MS EX ME WB
```

(*d*) Show the bypass connections (if any) and control logic so that the code below executes without a stall (and without causing other instructions to execute incorrectly, of course). The control logic should deliver a `BYPASS` signal to the EX stage at the right time, it should be `1` if a bypass of the type needed below is necessary. Do not connect it to anything.

☐ Show the control logic generating `BYPASS`.

☐ Add the bypass connection or show which existing one would be used.

```
# r1 = 0x1000, r5 = 0x1000
# Cycle              0  1  2  3  4  5  6
add.mr (r1), r2, 3   IF ID MS EX ME WB
sub.rm  r4, (r5), 6     IF ID MS EX ME WB
```

Problem 3: Answer each question below. Be sure to check each code fragment for dependencies. (25 pts)

(*a*) Show a pipeline execution diagram for the code fragment below running on the usual statically scheduled and bypassed scalar MIPS implementation. *Note: Bypassing and static scheduling not mentioned in the original exam.*

```
# Cycle         0    1    2    3    4    5   6   7   8

lw  r1, 0(r2)

add r3, r1, r4
```

☐ Solve this easy problem.

(*b*) The `sub.s` instruction below stalls in IF.

```
# Cycle           0  1  2  3  4  5  6  7  8  9  10 11
add.s f4, f5, f6  IF ID A1 A2 A3 A4 WF
sub.s f7, f4, f8     IF -------> ID A1 A2 A3 A4 WF
add.s f9, f10, f11                IF ID A1 A2 A3 A4 WF
```

☐ What's wrong about the stall?

☐ Show correct execution.

Problem 3, continued:

(*c*) The code below executes on a dynamically scheduled machine of the type used in the first problem and in class. The `mul.s` stalls in ID.

```
# ROB initially empty.
# Cycle           0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16
add.s f4, f5, f6  IF ID Q  RR A1 A2 A3 A4 WF C
sub.s f7, f4, f8     IF ID Q           RR A1 A2 A3 A4 WF C
mul.s f9, f10, f11      IF ID -------> Q  RR M1 M2 M3 M4 M5 M6 WF C
```

☐ What's wrong about the stall?

☐ Show correct execution.

☐ If the "`ROB initially empty.`" comment was not there then the execution above would be possible, albeit misleading. How would the stall be possible?

(*d*) Show a pipeline execution diagram for the code below executing on a 2-way statically scheduled superscalar MIPS implementation. *Note: In the original exam "statically scheduled" was omitted so an answer for either type of system would be correct.*

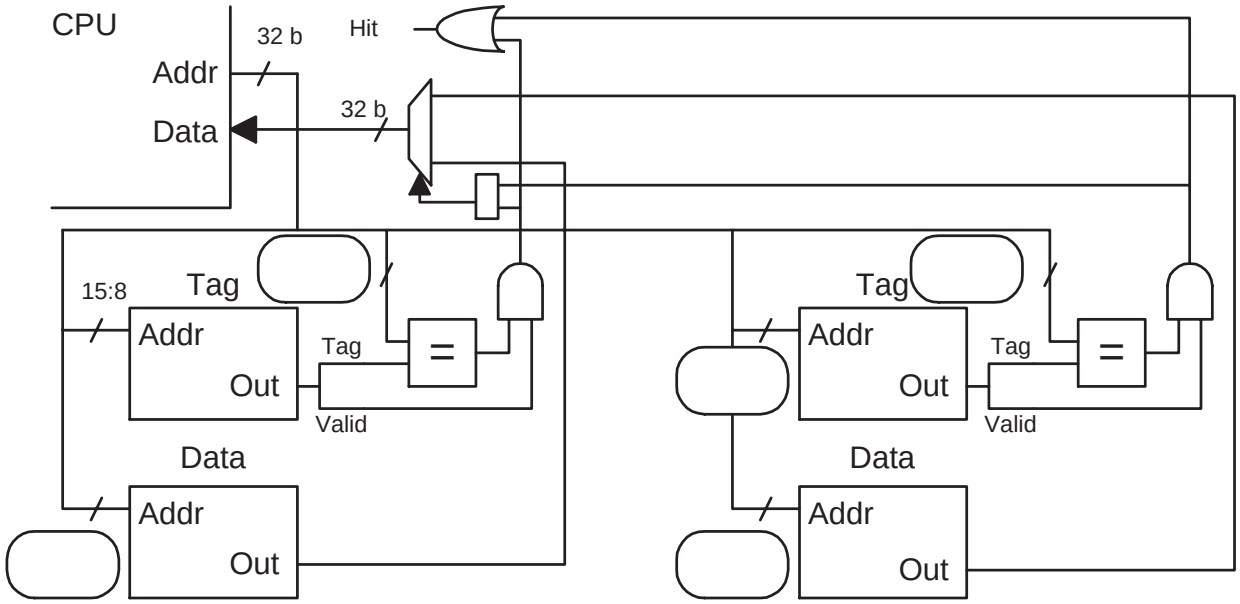
```
 # Cycle           0   1   2   3   4   5   6   7   8   9   10  11
0x1000:
 lw  r4, 0(r5)

 add r1, r2, r3

 lh  r6, 0(r4)

 xor r7, r1, r9

 sll r10, r11, 12

 srl r14, r15, 16

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

☐ Execution must be for decode logic of ordinary complexity. (See next item.)

☐ Execution must allow precise exceptions.

Problem 4: (10 pts) The diagram below is for a two-way set-associative cache on a system with 8-bit characters. 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:

☐ Cache Capacity (Indicate Unit!!):

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

☐ Line Size (Indicate Unit!!):

☐ Show the bit categorization for an eight-way cache with the same capacity and line size.

Address:

Problem 5: Answer each question below.

(*a*) The register renaming technique used in the dynamically scheduled processor (what the register maps do) solves a problem encountered when instructions execute out of order.(5 pts)

☐ What is the problem and how does it solve it?

☐ Provide an example showing what would go wrong without renaming.

(*b*) When it comes to caches one line size does not fit all.(5 pts)

☐ When is it better to have larger line sizes? Explain how the larger lines help.

☐ When is it better to have smaller line sizes? Explain how larger lines would hurt.

(*c*) Why would the SPEC CPU benchmark suite be less useful if it contained one integer program and one floating-point program? (5 pts)

☐ Why not just one of each?

(*d*) How would you answer a critic who said that the SPEC CPU benchmarks were rigged to make rich and powerful company *I*'s products look good? (5 pts)

☐ They are not rigged because ...

(*e*) What is the difference between a hardware interrupt, an exception, and a trap (as defined in class)? (5 pts)

☐ A hw interrupt's unique feature.

☐ An exception's unique feature.

☐ A trap's unique feature.