Name ______________________________

Computer Architecture

EE 4720

Final Examination

9 December 2003,   10:00–12:00 CST

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 execution of a MIPS code fragment on a dynamically scheduled machine is shown on the next page. The diagram shows the values on certain wires at certain cycles. For example, $\boxed{4{:}65}$ means that at cycle 4 the labeled wire holds value 65. The physical register file table is completed, ID- and commit-map tables are blank.

- The FP add unit has 3 stages, the FP multiply unit has 5, and the EA and ME are used for loads and stores.
- All destination registers are floating point.
- WB and commit can be done in the same cycle (indicate with a WC).
- To keep things simple the result of every instruction is zero and there are no cache misses.

(*a*) The ID and commit register map tables are blank . . .

☐ . . . complete them (the ID and commit register map tables.)

☐ Show the correct architected register numbers, or for partial credit make them up. (Two are easy, the rest are interesting.)

☐ Show the initial values (just before cycle 0) in the ID and commit map tables.

(*b*) Complete the pipeline execution diagram.

☐ Be sure to show Q, RR, WB, C (or WC), and a possible functional unit.

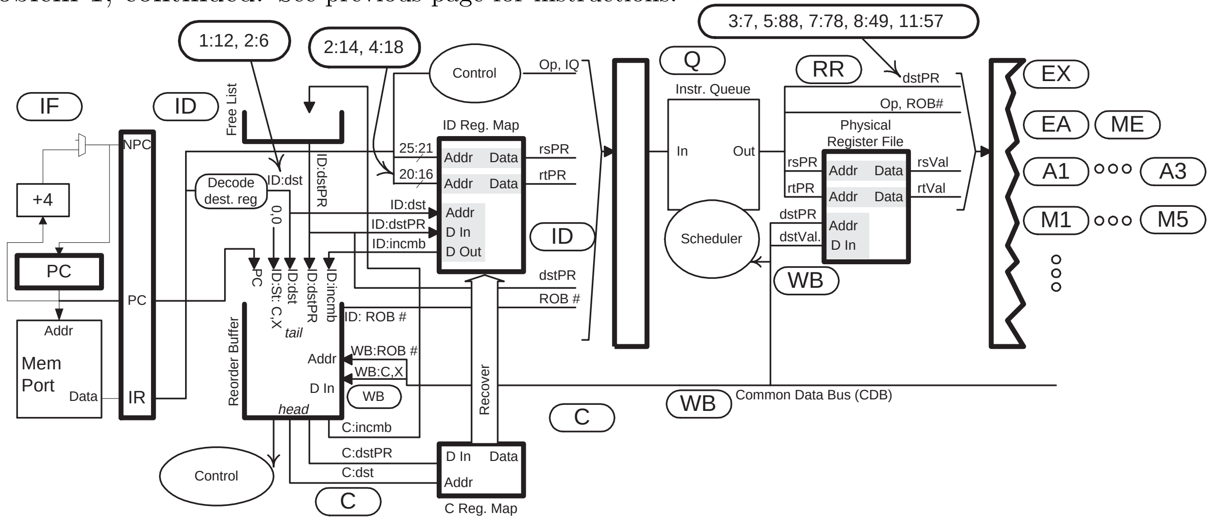
(*c*) Write a program consistent with these tables and labels.

☐ Choose consistent instructions.

☐ Choose consistent registers. If a register number cannot be determined, use a question mark.

- *Hint 1: In the physical register file table, put a "1" next to the first (earliest) register removed from the free list, put a "2" next to the second register removed from the free list, and so on. Similarly, put a "1" next to the first register put back in the free list, etc. To figure out which physical register belongs to which instruction destination (easy) use the fact that certain events occur in program order.*
- *Hint 2: To figure out which architected register an instruction is writing (interesting) remember what causes a register to be put back in the free list.*

Problem 1, continued: See previous page for instructions.



# Cycle	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	IF	ID															
		IF	ID														
			IF	ID													
				IF	ID												
					IF	ID											

# ID Map	Cycle	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

# Commit Map	Cycle	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

# Phys. Reg. File Cy	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
7		[								0.				]			
33										]							
49			[										0.			]	
57						[										0.	
78					[							0.					
88				[					0.						]		
95													]				
# Phys. Reg. File Cy	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Problem 2: (20 pts) In all of the problems below please check the code samples carefully for dependencies. All implementations below are fully bypassed. Please check the code samples carefully for dependencies.

(*a*) The code below executes on a 1-way statically scheduled machine in which the add FP functional unit has a latency of 3 and an initiation interval of 2.

☐ Show a pipeline execution diagram for the code.

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

add.s f2, f4, f6      IF

add.s f8, f2, f12

add.s f14, f10, f16

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

(*b*) The code below executes on a 1-way statically scheduled machine in which the add FP functional unit has a latency of 3 and an initiation interval of 1.

☐ Show a pipeline execution diagram for the code.

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

add.s f2, f4, f6      IF

add.s f8, f2, f12

add.s f14, f10, f16

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

Problem 2, continued:

(*c*) The code below executes on a 2-way statically scheduled machine in which the add FP functional unit has a latency of 3 and an initiation interval of 1.

☐ Show a pipeline execution diagram for the code.

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

LINE: # LINE = 0x1000
 add.s f2, f4, f6       IF

 add.s f8, f2, f12

 add.s f14, f10, f16

 and r1, r2, r3

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

(*d*) In a correct solution to the problem above there should be at least one instruction for which a precise exception is impossible. If that describes your solution, show a pipeline execution diagram below in which all instructions could raise precise exceptions (even though they don't). It's also possible that in a correct solution to the problem above all instructions can raise precise exceptions. If so, show a pipeline execution diagram below in which some instructions cannot raise a precise exceptions. In the absence of exceptions all pipeline execution diagrams must show correct execution.

☐ Show the appropriate pipeline execution diagram, or show how the one above would be different.

☐ Identify those instructions for which precise exceptions are impossible (above or below) and explain why.

```
 # Cycle                0   1   2   3   4   5   6   7   8   9   10  11  12  13  14  15
LINE: # LINE = 0x1000   (Either put solution here or show changes to previous solution.)

 add.s f2, f4, f6       IF

 add.s f8, f2, f12

 add.s f14, f10, f16

 and r1, r2, r3

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

Problem 2, continued:

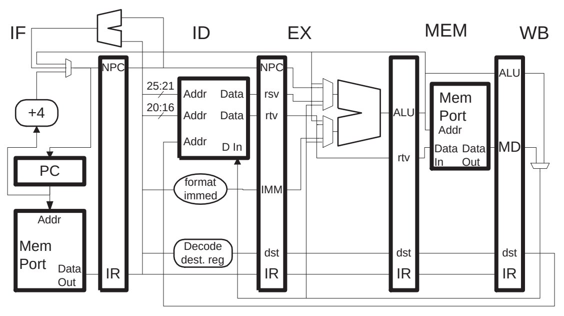
(*e*) The code below executes on a 2-way superscalar dynamically scheduled machine using method 3 (the only one covered this semester), the same one used in Problem 1. The FP add unit has a latency of 3 and an initiation interval of 1.

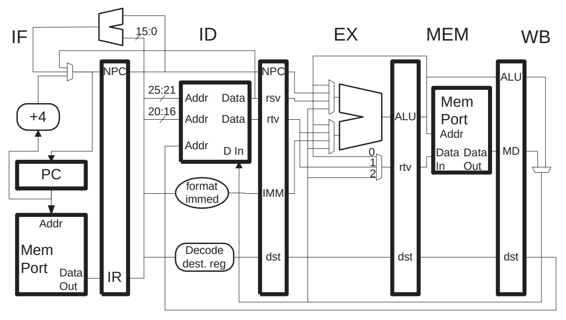
☐ Show a pipeline execution diagram. Don't forget the commit stage.

Assume an unlimited number of reorder buffer entries and physical registers.

```
 # Cycle               0   1   2   3   4   5   6   7   8   9   10  11  12  13  14  15
LINE: # LINE = 0x1000

 add.s f2, f4, f6      IF

 add.s f8, f2, f12

 add.s f14, f10, f16

 and r1, r2, r3

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

Problem 3: (20 pts) Two MIPS implementations are illustrated below, the right one has a multiplexor at the input to the `EX/MEM.rtv` pipeline latch, the left one does not.



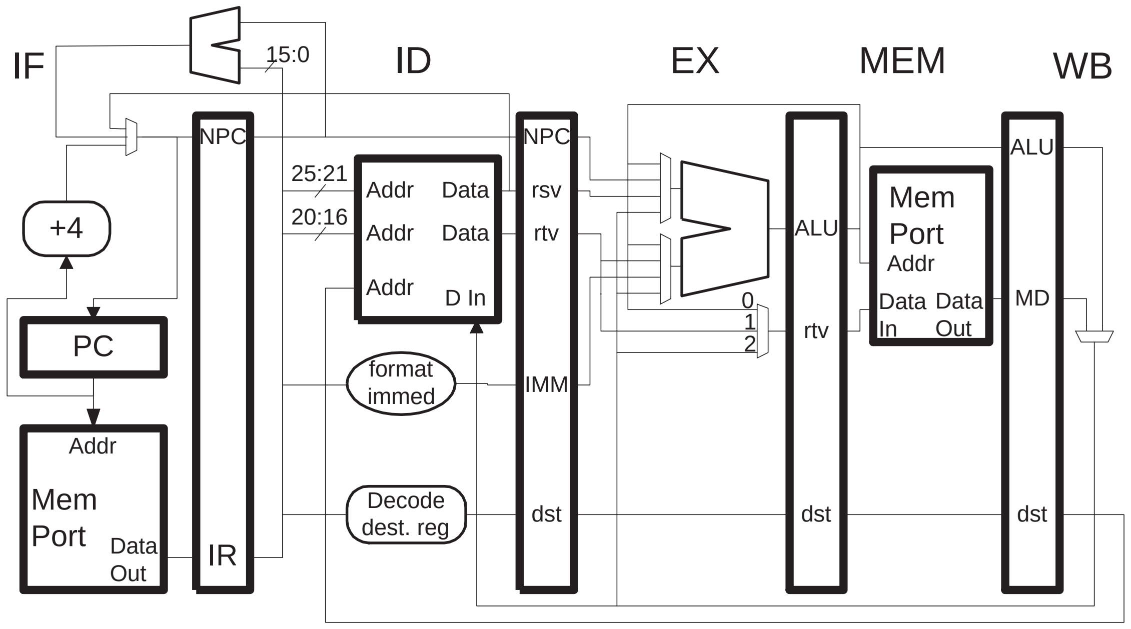


(*a*) Provide two code samples, one in which the multiplexor is useful and one in which it is not. Briefly explain.

(*b*) Suppose version 5.11 of a compiler was written for the implementation on the left and is in the hands of customers. Version 5.99 of the compiler also includes the right implementation and is being released soon. Which two compilation options would you have to use to take advantage of the changes made for the right implementation? (The exact names of the compiler options is unimportant, but it should be obvious what they do.) Briefly explain why each option is necessary and how it would affect the code.

Problem 3, continued:

(*c*) Design the control logic for the `rtv` multiplexor. Unlike Homework 4, **the logic must be in the EX stage**. Where appropriate, show which bits are being used, e.g., `12:5`.

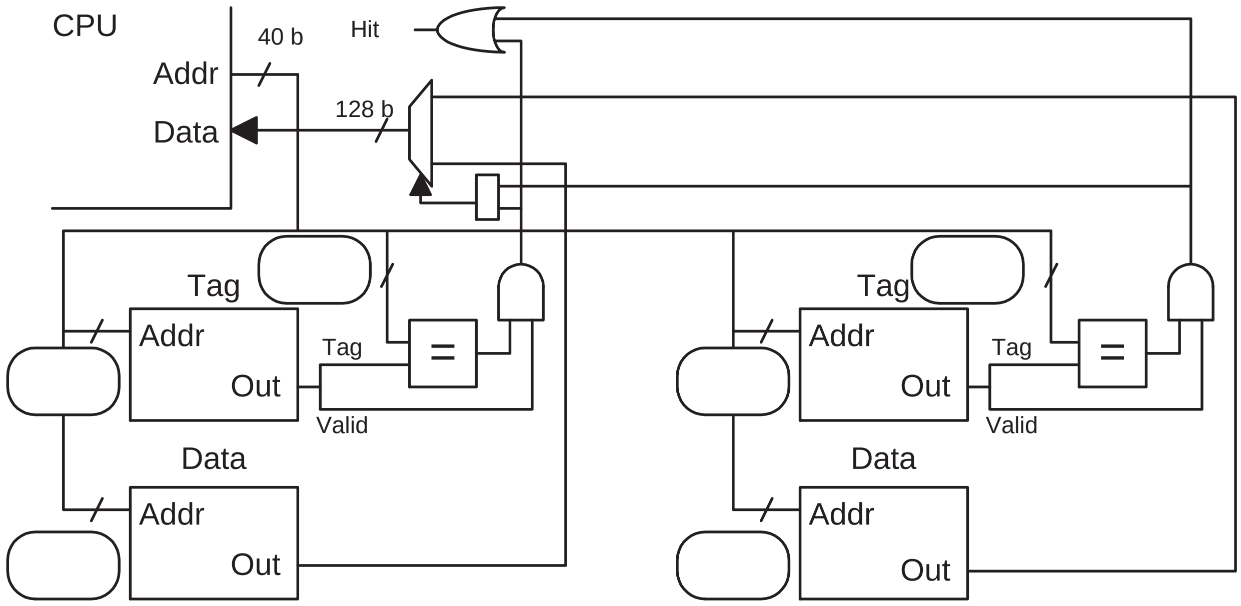


(*d*) There is a good reason why the control logic for the ALU input multiplexors should be in the ID stage that does not apply to the `rtv` multiplexor control logic. What is the reason, and why does it not apply to the `rtv` logic?

Problem 4: (20 pts) The diagram below is for a 32-MiB (2^{25} bytes) cache with 512-byte (2^9-byte) lines on a system with 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:

☐ Associativity:

☐ Memory Needed to Implement (Indicate Unit!):

☐ Show the bit categorization for a **fully associative** cache with the same capacity and line size. *Note: Emphasis not included in original exam.*

Address:

(*b*) The code below runs on the cache from the previous part. When the code below starts running the cache is empty. Consider only accesses to the array and assume the cache is cold (empty) when the code starts.

```
char *a = 0x1000000;  // sizeof(char) = 1 character
int sum, i, j;
int ILIMIT = 1024;

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

☐ What is the hit ratio for the program above?

(*c*) Find the hit ratio for the code below running on the cache from the first part. Consider only accesses to the arrays and assume the cache starts out cold. State any assumptions made.

```
char *a = 0x1000000;  // sizeof(char) = 1 character
char *b = 0x2000000;  // sizeof(char) = 1 character
char *c = 0x3000000;  // sizeof(char) = 1 character
int sum, i, j;
int ILIMIT = 1024;

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

(*d*) Modify the addresses of `a`, `b`, and `c` to maximize hit ratio. Explain how the modified addresses improve hit ratio.

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

(*a*) The diagram below shows the branch outcome patterns for two branches.

```
# Loop contains only the branches shown.
BIGLOOP:

B1: 0x1000 beq $t1, $t2, SKIP1   N  N  N  N  T  T  N  N  N  N  T  T  N  N  N  N  T  T
...
B2: 0x1200 beq $v0, $v1, SKIP2    T  T  T  T  T  T  T  T  T  T  T  T  T  T  T  T  T
...
    0x2010 j BIGLOOP
```

☐ How accurately would branch B1 be predicted by a bimodal (one-level) branch predictor with a 2^{14}-entry branch history table (BHT)?

☐ What is the minimum size of the BHT for which the accuracy in the previous part is possible? Explain.

☐ Why might it be pointless to perform branch prediction in the ID stage of the 1-way statically scheduled pipeline used in class?

(*b*) Answer the following questions about exception codes as defined for SPARC V8 and using the class terminology.

☐ What is an exception code number (or trap type)? *Note: In the original exam the question was shorter: "What is an exception code?"*

☐ How is it obtained for traps?

☐ How is it obtained for hardware interrupts?

☐ How is it obtained for exceptions?

☐ How is it used to start the handler?

(*c*) Consider two processors, one is a 6-way superscalar implementation of an ordinary ISA, say MIPS, the other is a 6-way implementation of a VLIW ISA, say Itanium (IA-64).

☐ Describe two features of Itanium (or some other VLIW ISA) that would allow it to execute faster than the superscalar implementation. Explain how the features allow faster execution.

☐ If the implementation for the VLIW ISA were faster why might the superscalar implementation still be better from a business perspective?