Name _____

Computer Architecture
EE 4720
Final Examination

8 May 2000, 10:00–12:00 CDT

Problem 1	_____	(20 pts)
Problem 2	_____	(10 pts)
Problem 3	_____	(10 pts)
Problem 4	_____	(21 pts)
Problem 5	_____	(39 pts)
Exam Total	_____	(100 pts)

Alias _____

Good Luck!

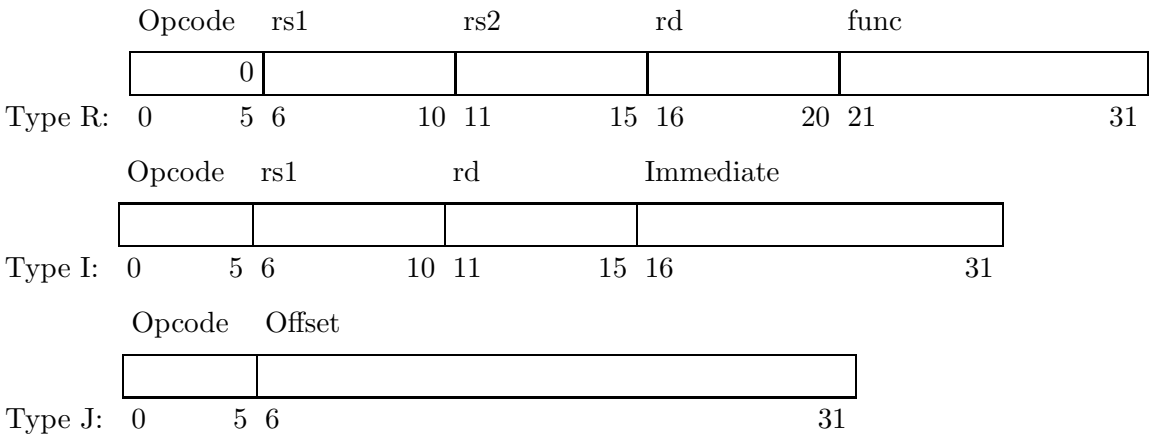
Problem 1: An extended DLX ISA, Triple-DLX [tm], includes new three-operand integer ALU instructions. (Assume that the integer ALU can perform integer multiply.) Some sample instructions appear below:

```
add_add r1, r2, r3, r4 ! r1 = r2 + r3 + r4
add_mul r5, r6, r7, r8 ! r5 = ( r6 + r7 ) * r8
mul_add r5, r6, r7, r8 ! r5 = ( r6 * r7 ) + r8
```

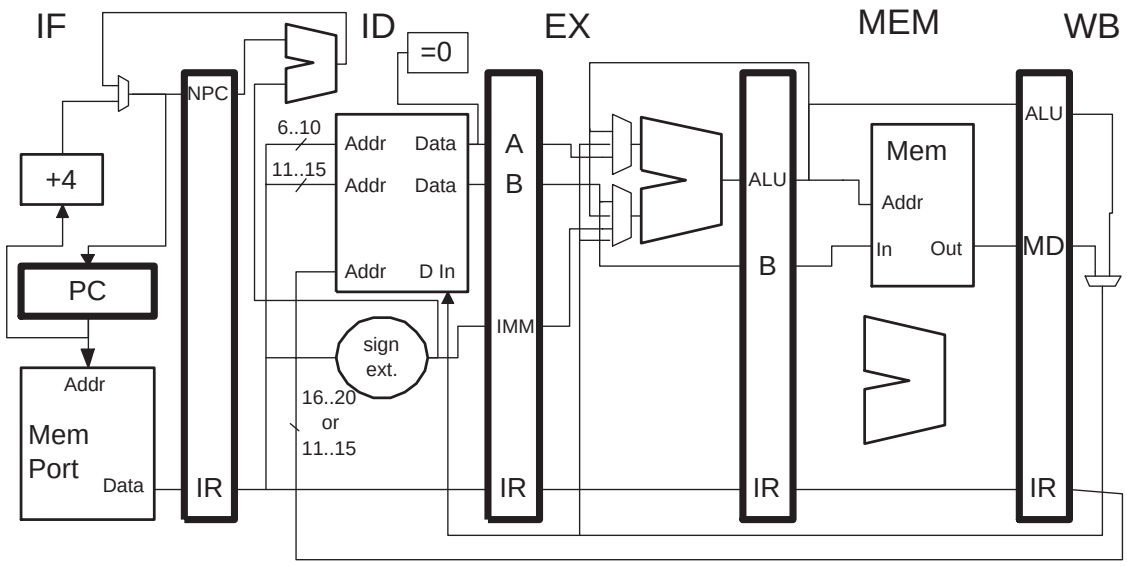
(*a*) (8 pts)A new instruction type, Type-T, will be used for these instructions. Show how the new Type-T instructions can be coded. *The coding should be chosen to ease implementation and should allow for at least 64 three-operand instructions.* Assume that there are six free opcode field values and seven free func-field values available for your use.

- Explain how to distinguish Type-T instructions from Type-R, Type-I, and Type-J instructions.
- How many Type-T instructions can be provided using your coding?

The DLX codings are given below for reference and can be used to explain your answer.



(*b*) Modify the pipeline below so that it can execute the three-operand instructions. A second ALU has been placed in the MEM stage; it should be used to help implement the instructions. The register file is among the parts that need to be modified. (6 pts)



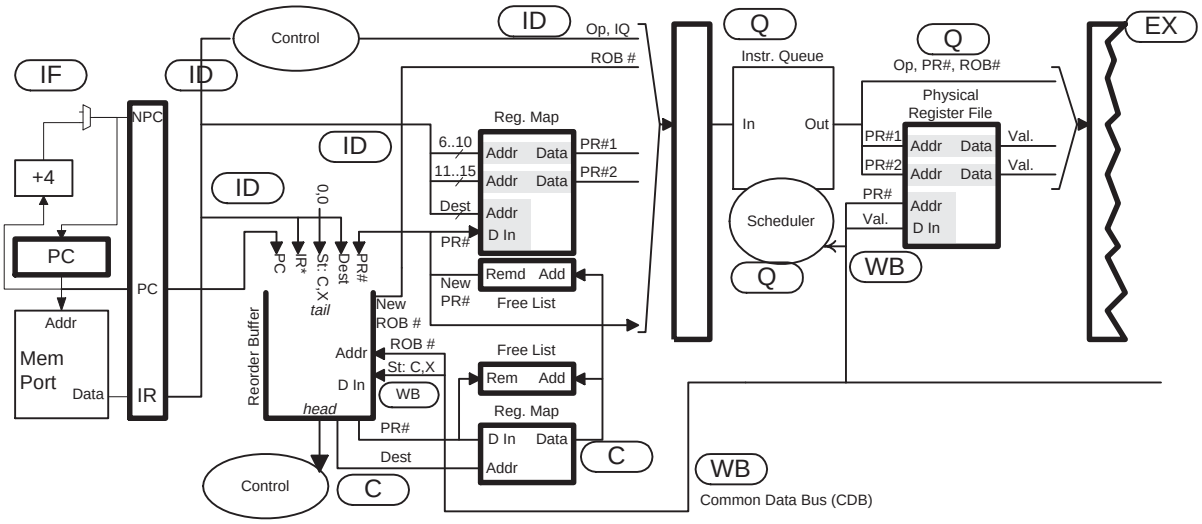
(*c*) Show a pipeline execution diagram for the code below assuming that all needed bypass paths are available. The code should execute as fast as possible. **Add** any needed bypass paths to the diagram above. Do not add any bypass paths that are not needed by the code below. Label each bypass path (added or already present) with the cycle in which it is used. Please be sure not to miss any true dependencies. (6 pts)

```
add_add r1, r2, r3, r4

add_sub r5, r1, r2, r3

sub_sub r6, r1, r5, r6

sub_add r7, r1, r5, r6
```

Problem 2: The code appearing on the next page executes on a dynamically scheduled machine using physical registers to name destination registers.

Complete the tables, showing only changes. Show where instructions commit. Show only entries associated with registers `f0` and `f6` in the physical register file.

At cycle zero, register `f0` contains a 5, `f2` contains a 20, `f4` contains a 40, and `f6` contains a 60. None of the instructions raise exceptions. The diagram below is provided for convenience; it is the same one used in class. (10 pts)



Cycle	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
add f0,f2,f4	IF	ID	Q								A0	A1	WB					
add f6,f0,f4		IF	ID	Q									A0	A1	WB			
add f0,f4,f4			IF	ID	Q					A0	A1	WB						
add f6,f0,f4				IF	ID	Q								A0	A1	WB		
sub f0,f2,f4					IF	ID	Q	A0	A1	WB								

Cycle:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

Arch. Reg.	ID Register Map
f0	17
f6	6

	ID Free List
	12 7 8 4 3

Cycle:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

	Commit Register Map
f0	17
f6	6

	Commit Free List
	12 7 8 4 3

Cycle:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

Phys Reg.	Physical Register File

Problem 3: Provide a pipeline execution diagram for the code below running on a dynamically scheduled *two-way superscalar* machine using reorder buffer entry numbers to name destination operands. Be sure to show when instructions complete.

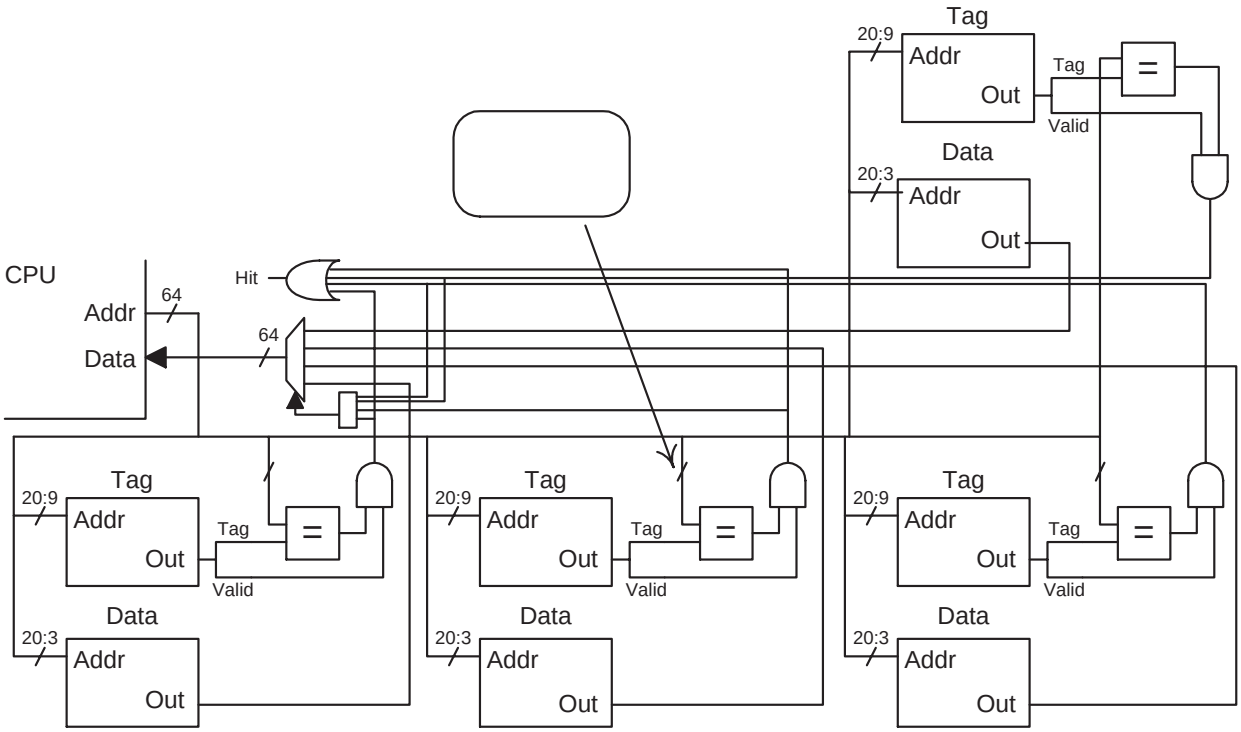
There are more than enough reservation stations and reorder buffer entries. Do not show reservation station numbers in the diagram. There are two load-store units and the *cache is nonblocking*.

The first `lw` misses the cache, the data arrives 6 cycles after it first enters the L1 stage; the second `lw` also misses the cache, its data arrives 4 cycles after it first enters the L1 stage. (10 pts)

Carefully check the code for dependencies before working on a solution.

```
LINE: LINE = 0x1000


 lw r3, 0(r1)

 lw r5, 0(r2)

 sw 0(r4), r5

 lw r7, 0(r9)

 sw 0(r3), r1

 lw r8, 0(r10)
```

Problem 4: A system uses the following cache:



(*a*) Determine the value of the following parameters for the cache illustrated above. Be sure to specify units (bits, bytes, etc.). Answers can be in the form of mathematical expressions. (8 pts)

Associativity:

Number of Sets:

Address Space Size:

Block (Line) Size:

Cache Capacity:

Amount of Memory to Implement Cache:

Character Size:

Tag Bits: (Can show on diagram.)

(*b*) Write a program that will fill the cache using the minimum number of accesses. (Ignore instruction accesses.) (8 pts)

(*c*) Determine the parameters for a direct-mapped cache with the same block size and the same capacity designed for the same system. (5 pts)

Associativity:

Number of sets:

Address Space Size:

Block (Line) Size: (Same, don't answer.)

Cache Capacity: (Same, don't answer.)

Character Size:

Tag Bits:

Problem 5: Answer each question below.

(*a*) The program below runs on a system using a gselect branch predictor. What is the minimum global history size needed so that there is a good chance that the last branch will be correctly predicted at the last iteration (after warmup)? Assume that there are no collisions. Explain your answer. (7 pts)

```
 addi r1, r0, #10
 add  r5, r0, r0
LOOP:
 lw r2, 0(r3)
 add r5, r5, r2
 subi r1, r1, #1
 addi r3, r3, #4
 bneq r1, LOOP
```

(*b*) What is the advantage of backing up (checkpointing) the register map when a branch is encountered in a system using branch prediction and dynamic scheduling? Explain how execution would be different if the register map were not backed up. (7 pts)

(*c*) Why are three-way superscalar machines difficult to build but three-instruction-bundle VLIW ISAs are common? (5 pts)

(*d*) An ISA is almost like DLX except, oops, the `lbu` (load byte unsigned) instruction was omitted, and so the program below won't run on an implementation of this ISA. Modify the program so that it will run, and run as though an `lbu` instruction was used. (In other words, replace `lbu r1, 1(r2)` by instructions that do the same thing.) (5 pts)

```
lbu r1, 1(r2)
```

(*e*) The table below shows virtual and physical addresses in use in a virtual memory system having a 32-bit address space. What is the largest possible page size this system can have? Using this page size (or some other one, but show what page size is being used) show the possible contents of a two-level page table storing this virtual-to-physical mapping. State any assumptions made. (For partial credit solve the problem for a one-level page table.) (10 pts)

Virtual	Physical
0xfea34b62	0x74b4b62
0xfeb92b90	0x14b2b90
0xeaa31f16	0x77b1f16

(*f*) Describe two ways that loop unrolling improves performance. (5 pts)