Name Solution

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 MPL phone home!!!

Good Luck!

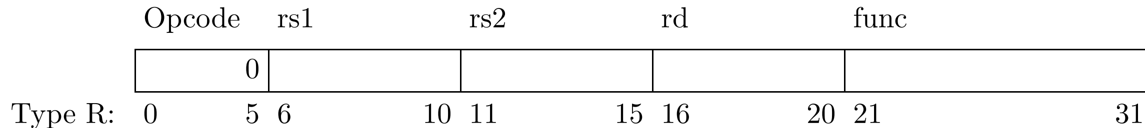
Problem 1: An extended DLX ISA, Triple-DLX [tm], includes new three-operand integer ALU instructions. (Assume that the integer ALU can perform integer multiply.) Some sample instructions appear below:

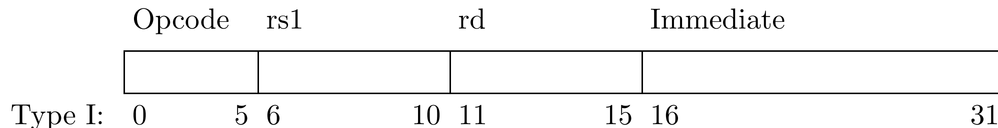
```
add_add r1, r2, r3, r4 ! r1 = r2 + r3 + r4
add_mul r5, r6, r7, r8 ! r5 = ( r6 + r7 ) * r8
mul_add r5, r6, r7, r8 ! r5 = ( r6 * r7 ) + r8
```

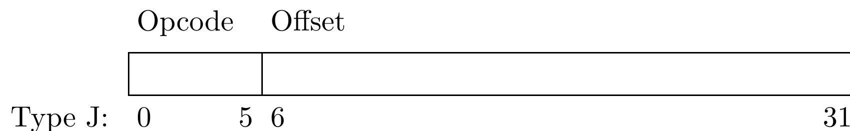
(*a*) (8 pts)A new instruction type, Type-T, will be used for these instructions. Show how the new Type-T instructions can be coded. *The coding should be chosen to ease implementation and should allow for at least 64 three-operand instructions.* Assume that there are six free opcode field values and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

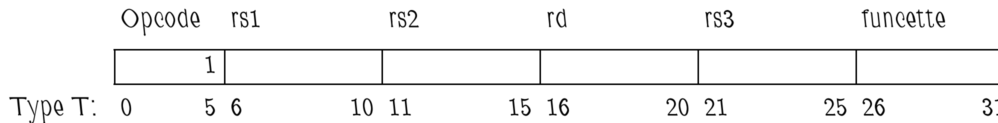
The DLX codings are given below for reference and can be used to explain your answer.





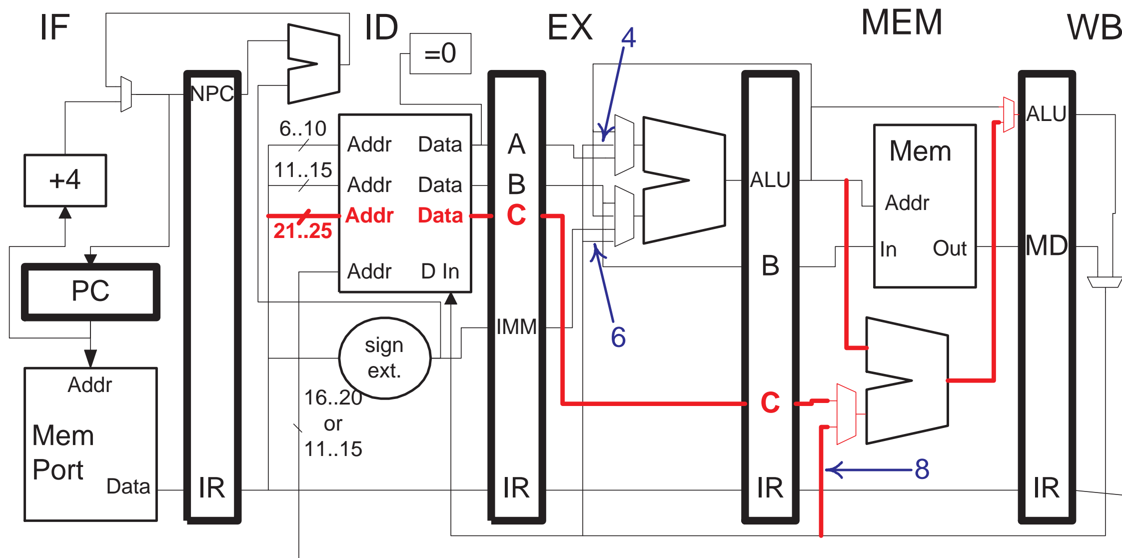


The instruction format is similar to Type R, except an rs3 field is added. It's added after rd, rather than before, so that decoding logic would not have to look for rd in a third place. Type T instruction use one of the 6 opcodes, opcode 1 is used in the example. The funcette field, at 6 bits, specifies which of 64 instructions to perform.



(*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)

Changes to the pipeline are shown in **red bold.**



(*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)

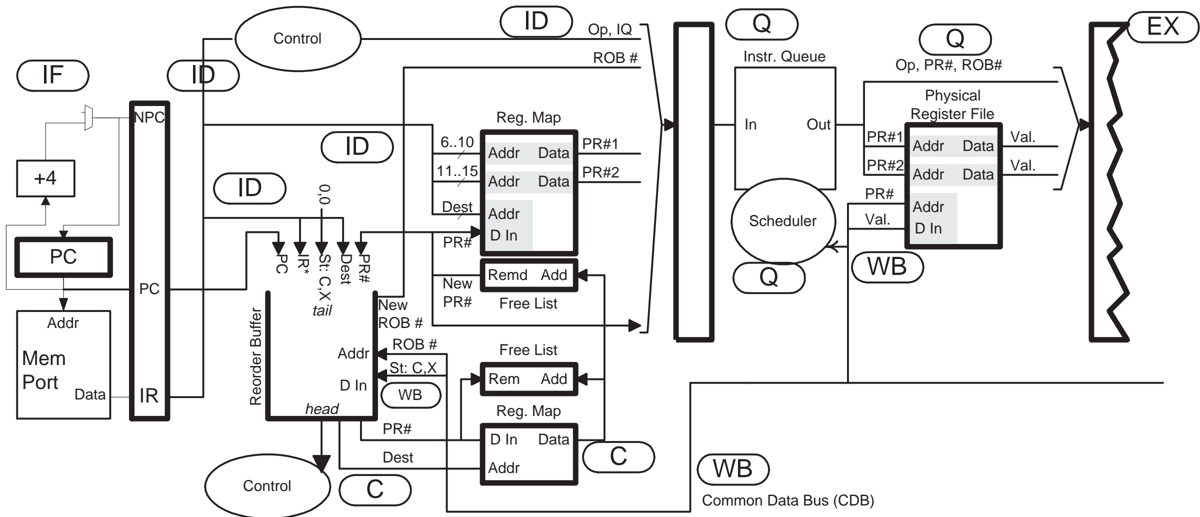
The pipeline execution diagram is shown below. Note that the MEM stage has been labeled E2 to emphasize its role. Also note that in cycle 8 r6 is bypassed to the second ALU in the E2 (MEM) stage. The bypass paths and the cycles in which they are used are shown in blue in the illustration above.

```
! Solution
! Cycle                    0  1  2  3  4  5  6  7  8  9
 add_add r1, r2, r3, r4   IF ID EX E2 WB
 add_sub r5, r1, r2, r3      IF ID -> EX E2 WB
 sub_sub r6, r1, r5, r6         IF -> ID -> EX E2 WB
 sub_add r7, r1, r5, r6               IF -> ID EX E2 WB
```

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	WC					
`add f6,f0,f4`		IF	ID	Q									A0	A1	WC			
`add f0,f4,f4`			IF	ID	Q					A0	A1	WB				C		
`add f6,f0,f4`				IF	ID	Q								A0	A1	WB	C	
`sub f0,f2,f4`					IF	ID	Q	A0	A1	WB								C

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	12		8		3												
f6	6		7		4													
	ID Free List																	
	12 7 8 4 3	7 8 4 3	8 4 3	4 3	3								17		17 6	17 6 12	17 6 12 7	17 6 12 7 8

Cycle:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
	Commit Register Map																	
f0	17												12			8		3
f6	6														7		4	
	Commit Free List																	
	12 7 8 4 3												7 8 4 3 17		8 4 3 17 6	4 3 17 6 12	3 17 6 12 7	17 6 12 7 8

Cycle:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Phys Reg.	**Physical Register File**																	
3										-20								
4																120		
6	60																	
7															100			
8												80						
12													60					
17	5																	

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*.

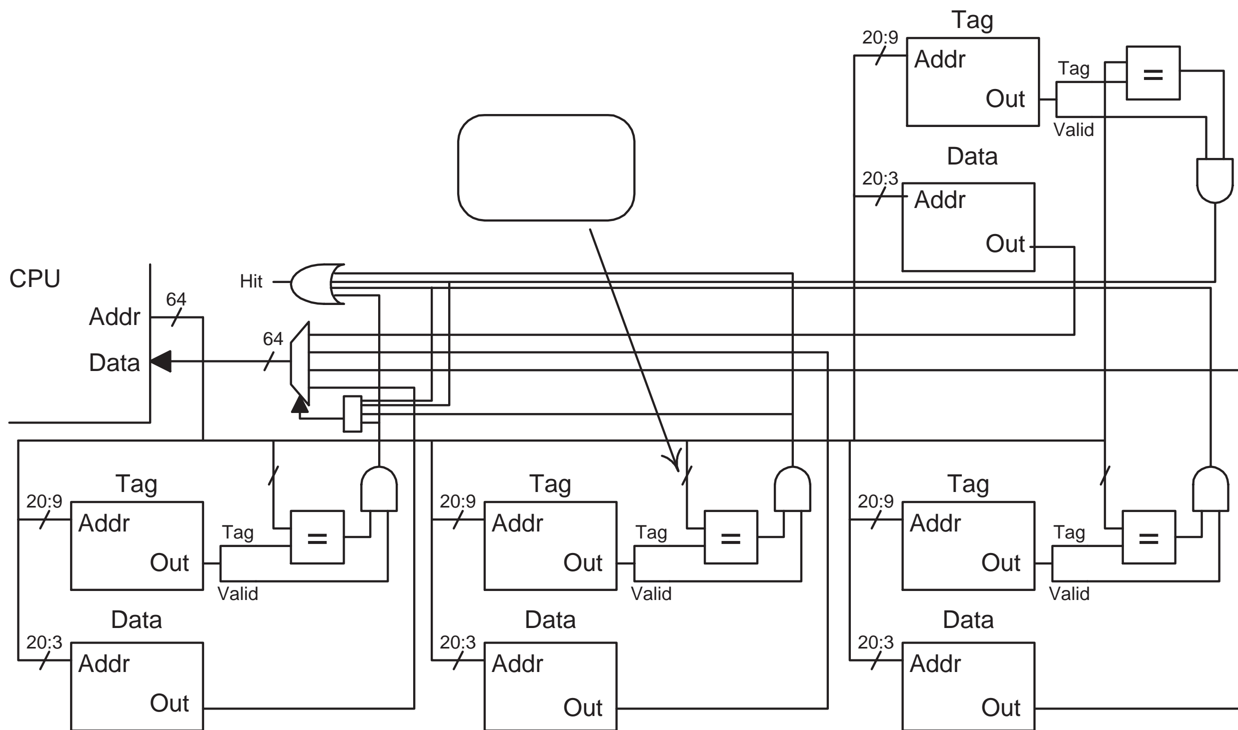
The first `lw` misses the cache, the data arrives 6 cycles after it first enters the L1 stage; the second `lw` also misses the cache, its data arrives 4 cycles after it first enters the L1 stage. (10 pts)

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

The pipeline execution diagram appears below. Note: Because the effective address of the second store is not known in cycle 4, the last load cannot proceed (because of a possible dependency).

```
! Solution
! Cycle         0  1  2  3  4  5  6  7  8  9  10 11 12
lw r3, 0(r1)   IF ID L1 L2 RS          L2 WC
lw r5, 0(r2)   IF ID L1 L2 RS    L2 WB    C
sw 0(r4), r5      IF ID L1 RS       L2 WB    C
lw r7, 0(r9)      IF ID L1 L2 WB             C
sw 0(r3), r1         IF ID RS             L1 L2 WC
lw r8, 0(r10)        IF ID L1                L2 WC
```

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: 4

Number of Sets: Solution: $2^{20-9+1} = 2^{12}$.

Address Space Size: 64 bits

Block (Line) Size: Solution: $2^9 = 512$ characters

Cache Capacity: Solution: 4×2^{21} characters.

Amount of Memory to Implement Cache: Solution: 4×2^{21} characters plus 4×2^{12} $((63 - 21 + 1) + 1)$ bits.

Character Size: 8

Tag Bits: (Can show on diagram.) Solution: 63:21.

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

```
void fill()
{
  extern char *a;

  int line_size_lg = 9;
  int line_size = 1 << line_size_lg;
  int associativity = 4;
  int set_count_lg = 12;
  int line_count = associativity * ( 1 << set_count_lg );
  int i, dummy;

  for(i=0; i<line_count; i++) dummy += a[ i * line_size ];

}
```

(*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: One, because it's direct mapped.

Number of sets: Solution: 2^{12+2}

Address Space Size: Solution: 64, no change.

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

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

Character Size: Same, 8 bits

Tag Bits: Two less: 63:23.

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
```

If execution is at the last branch and at least one of the previous 9 branches was not taken, then execution surely has not reached the 10th iteration. It would take a global history length of 9 to hold this information. Prediction would not be perfect, it would depend on whether the last branch encountered before reaching the loop is taken.

(*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)

If the register map is backed up and a branch misprediction is discovered the register map can be restored to the state it was in when the branch was encountered without having to wait for preceding instructions to commit. With a register map backup recovery can start in the WB stage of the branch, without a backup recovery must wait for the commit stage.

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

Superscalar machines are built on existing ISAs, many of which use 32-bit instructions. If 32-bit instructions were fetched in groups of three the fetches would be unaligned and so would take more expensive hardware. In many VLIW ISAs, three instructions are placed in 128-bit bundles, so their addresses are aligned.

While its true that VLIW ISAs allow for less expensive hardware, a three-way superscalar machine is still small and so the hardware needed for dependency checking, register renaming, and scheduling would be less than difficult.

(*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)

! A correct answer:
lb r1, 1(r2)
andi r1, r1, #0xff

! A WRONG answer:
lw r1, 1(r2)       ! Error, since address may not be aligned.
andi r1, r1, #0xff
```

(*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

The number of offset bits is the log-base-2 of the page size. The offset bits of physical and virtual addresses are the same, so the largest possible page size here is 2^{17} characters (because the low 17 bits of 9xfea34b62 and 0x74b4b62 are the same, the low 17 bits of 0xfeb92b90 and 0x14b2b90 are the same, and the low 17 bits of 0xeaa31f16 and 0x77b1f16 are the same). Note that it's possible the page size is smaller.

The solution below is for a 2^{16} character page. The level-one table is index using the upper 8 bits of the address and the second-level table is index using the next 8 bits.

Level One Page Table:

Address	Level Two Base
⋮	⋮
0xea	3000
⋮	⋮
0xfe	7000
⋮	⋮

Level Two Page Table:

Address	Physical Page Number
⋮	⋮
3000	⋮
⋮	⋮
30a3	77b
⋮	⋮
7000	⋮
⋮	⋮
70a3	74b
⋮	⋮
70b9	14b
⋮	⋮

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

Fewer branches are needed, eliminating the instruction themselves plus the branch penalty. Certain operations, such as index variable increments can be eliminated. Greater scheduling freedom.