CSE 471 Winter 2000

Computer Design and Organization

Final

Monday March 13th

NAME : ______________________________

Do all your work on these pages. Do not add any pages. Use back pages if necessary. Show your work to get partial credit.

This exam is worth 100 points. After each question, you will find the number of points it is worth. You should spend approximately x minutes on a question worth x points. That will leave you with 15 minutes to read the statement of the problem and to look over your work.

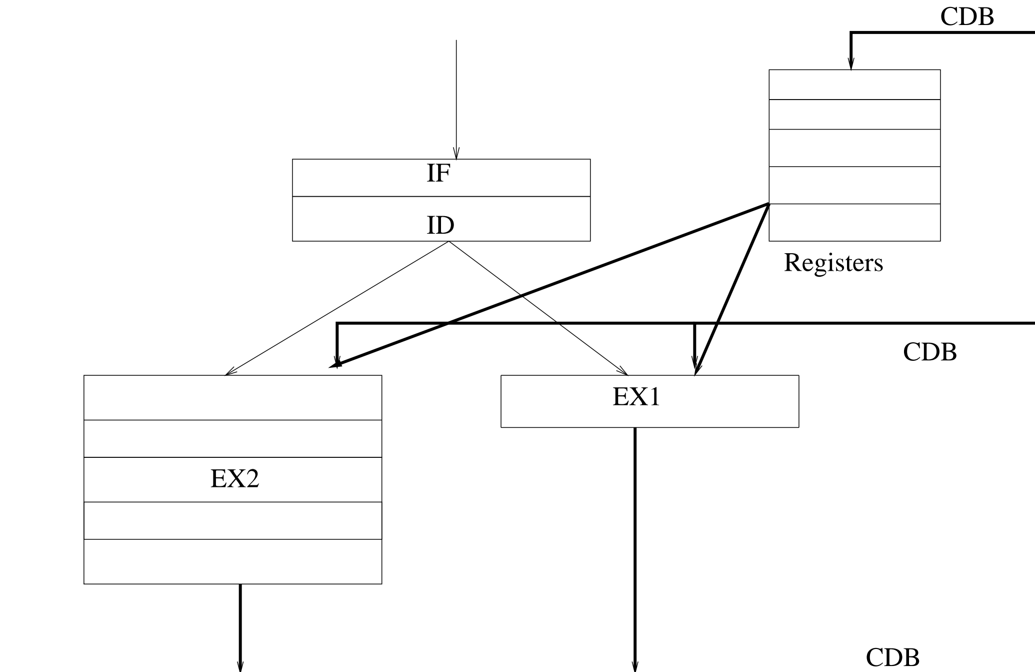
1. 50 points
   - (a) 4 points ______________
   - (b) 10 points ______________
   - (c) 12 points ______________
   - (d) 16 points ______________
   - (e) 8 points ______________
2. 16 points
   - (a) 8 points ______________
   - (b) 8 points ______________
3. 24 points
   - (a) 12 points ______________
   - (b) 12 points ______________
4. 10 points

1. (50 points)

Consider the special-purpose processor below that computes only multiplications and additions. It is pipelined with the following stages:

- IF Instruction fetch: takes 1 cycle
- ID Instruction decode and read registers: takes 1 cycle
- EX1 Execution of additions: takes 1 cycle
- EX2 (with 5 stages) Execution of multiplications: takes 5 cycles for a single multiplication but a new multiplication can be started every cycle

There is no write stage (WB). Instead the results are broadcast on the common data bus (CDB) to the register file and to the input stages of the multiplier (EX2) and adder (EX1). EX1 and EX2 can use the result as input during the cycle where it is broadcast. In case both EX1 and EX2 want to broadcast in the same cycle, EX1 is stalled. If the instruction in the ID stage at that time is an addition, it is stalled in the ID stage and the next instruction is stalled in the IF stage, and no new instruction is fetched.



Questions (b) through (d) are related but are independent of each other. Doing them in order might be preferable.

(a) (4 points)

If one wants to reduce RAW dependencies, *is there a need to create additional paths to forward* (justify your answer if you think such a path is needed):

- The result of EX1 to the input of EX1
- The result of EX2 to the input of EX2
- The result of EX2 to the input of EX1
- The result of EX1 to the input of EX2

In the remaining of this question, we are going to use the following sequence of instructions:

```
R1  =  R2  *  R3    \\multiplication
R2  =  R4  +  R5    \\addition
R3  =  R1  +  R2    \\addition
R1  =  R6  +  R7    \\addition
```

You can assume that the contents of all registers are available before the first instruction is fetched.

(b) (10 points) (This question continues on the next page)

In a first (naive) implementation the instructions are required to issue and to complete in order (e.g., the result of the multiplication in the example sequence must be stored in R1 before the result of the first addition is stored in R2).

What control mechanism is needed to implement this requirement? (Hint: You can answer with something like: "don't issue instruction i to EXj if ...".)

How many cycles will it take to execute the sequence of 4 instructions above? The following table might be of help to you (the numbers that are entered correspond to the cycles where the corresponding actions are performed):

Instruction	IF	ID	EX1 or EX2 start	Write in register
R1 = R2 * R3	1	2	3	8
R2 = R4 + R5	2	3		
R3 = R1 + R2				
R1 = R6 + R7				

For that particular sequence would it help if you could issue 2 instructions per cycle with the obvious restrictions due to structural hazards?

Instruction	IF	ID	EX1 or EX2 start	Write in register
R1 = R2 * R3	1	2	3	
R2 = R4 + R5	1	2		
R3 = R1 + R2				
R1 = R6 + R7				

(c) (12 points) (This question continues on the next page)

In a second implementation, instructions are still required to issue in order but they can now execute and complete out-of-order subject of course to the semantics of the program (i.e., the WAW and WAR dependencies must be taken care of). The issuing is controlled during the ID stage.

Assuming no register renaming, no reservation stations and no reorder buffer, *what does the control unit need to know to allow/prevent instructions to issue? Suggest a possible implementation.*

Under these conditions, *how many cycles will it take to execute the sequence of 4 instructions above?*

Instruction	IF	ID	EX1 or EX2 start	Write in register
R1 = R2 * R3	1	2	3	
R2 = R4 + R5	2	3		
R3 = R1 + R2				
R1 = R6 + R7				

For that particular sequence would it help if you could issue 2 instructions per cycle with the obvious restrictions due to structural hazards?

Instruction	IF	ID	EX1 or EX2 start	Write in register
R1 = R2 * R3	1	2	3	
R2 = R4 + R5	1	2		
R3 = R1 + R2				
R1 = R6 + R7				

(d) (16 points) (This question continues on the next page)

In a third implementation, instructions are required to issue and complete in order but they can execute out-of-order. This is accomplished via a combination of register renaming, and/or introduction of reservation stations, and/or introduction of a reorder buffer. More specifically:

- During the ID stage, register renaming takes place, a reservation station is filled for the relevant execution unit with either values or pointer (tags) to values, and an entry is created at the tail of the reorder buffer (You can assume that there won't be any structural hazards so all these steps can be taken).
- During any cycle, the instruction at the head of the reorder buffer can be committed (i.e., the adequate value can be written in the logical register) if it is ready to be committed. In particular, if the result being broadcast on the common data bus (CDB) is the value expected by the head of the reorder buffer, the value is forwarded during the same cycle to the logical register.

With the above mechanisms, the latency of a single instruction in isolation is the same as in the case of in-order issue, execution, and completion.

Fill out the tables below showing the time where various events occur during execution of the sample sequence. The first table corresponds to a single issue machine and only one instruction can be committed during a given cycle while the second is for a machine issuing two instructions at a time and two instructions can be committed during the same cycle.

Instruction	IF	ID	EX1 or EX2 start	Write in reorder buffer/or renamed register	Write in logical register
R1 = R2 * R3	1	2	3		
R2 = R4 + R5	2	3			
R3 = R1 + R2					
R1 = R6 + R7					

Instruction	IF	ID	EX1 or EX2 start	Write in reorder buffer/or renamed register	Write in logical register
R1 = R2 * R3	1	2	3		
R2 = R4 + R5	1	2			
R3 = R1 + R2					
R1 = R6 + R7					

(e) (8 points) (You can use the next page left blank for that purpose)

Describe one possible implementation of register renaming. (you can use the above sequence to explain the implementation.)

2 (16 points)

Array A contains 256 elements of 4 bytes each. Its first element is stored at physical address 4096 (4 * 1 K).

Array B contains 512 elements of 4 bytes each. Its first element is stored at physical address 8192 (8 * 1 K).

Assume that only arrays A and B can be cached in an initially empty physically addressed, physically tagged (i.e., the usual type of cache), direct-mapped, 2 KB cache with an 8 byte block size.

The following loop is then executed:

```
for (i = 0; i < 256; i++)
    A[i] = A[i] + B[2*i];         // Load A[i]; Load B[2i];
                                  // Add; Store A[i]
```

(a) (8 points)

Assume a write-through write-around policy.

What are the contents of the cache at the end of the loop?

How many bytes will be written to memory?

(b) (8 points)

Assume a write-back write-allocate policy.

What are the contents of the cache at the end of the loop?

How many bytes will be written to memory?

3 (24 points) (This question continues on the next page)

Consider a two-level cache hierarchy. The data cache at the first level (L1) is 16 KB, 2-way set-associative, with a block size of 32 bytes. The cache at the second level L2 is 512 KB, 4-way set-associative, and has a block size of 64 bytes.

(a) (12 points)

Show the lengths of the various subfields (tag,index,displacement) of a 32-bit address as seen:

- by the L1 cache

- by the L2 cache

If this cache hierarchy is to be used in conjunction with a paged virtual memory system, *what should be the page size so that the L1 cache can be virtually addressed and physically tagged?*

Would the choice of that page size make it possible to have L2 also virtually addressed and physically tagged? If so, explain why. If not, state why you think it is less important to have this capability for L2.

(b) (12 points) (This question continues on the next page)

We now introduce a victim cache "behind" L1 and "before" L2. The victim cache is used only on read misses. The victim cache has a capacity of 4 blocks (32 bytes each) and is fully-associative.

What is the size of the tag associated with each victim cache entry?

Assuming that the page size has been selected so that L1 can be virtually addressed and physically tagged, *can the victim cache be also virtually addressed and physically tagged?* Justify your answer.

Under these conditions *explain why read hits would take 1 cycle for L1 and 2 cycles for the victim cache.*

When there is a miss in L1 and a miss in the victim cache, the missing block is brought in from L2 (or main memory) and stored in L1. The block replaced in L1 replaces the LRU entry in the victim cache.

How is the tag for the new entry in the victim cache deduced from that of the replaced block in L1?

What happens to the contents of the replaced (LRU) entry of the victim cache if:

- L1 is write-through

- L1 is write-back (Hint: There are several possible answers; choose whichever you wish.)

4 (10 points)

What are the three types of cache misses (the three C's)?

For each type name a hardware or software enhancement that can decrease the number of misses of that type.

Name and describe very briefly 2 techniques to reduce or tolerate memory latency caused by cache misses.