CSE 471 Autumn 2001

Computer Design and Organization

Final

Friday December 14th

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. - 8 points ______________
2. - (a) 12 points ______________
   - (b) 18 points ______________
3. - (a) 4 points ______________
   - (b) 8 points ______________
   - (c) 12 points ______________
   - (d) 10 points ______________
4. - (a) 10 points ______________
   - (b) 10 points ______________
5. - 8 points ______________

1. (8 points; 1 each for correct true or false)

Answer by **True** or **False**

- The accuracy of branch prediction does not impact the performance of an in-order superscalar processor.

- The accuracy of branch prediction does not impact the performance of an out-of order processor.

- Multiple issue of instructions provides the opportunity for more performance benefits in out-of-order processors than in in-order processors

- Victim caches are more efficient when used in conjunction with direct-mapped caches rather than with 4-way set-associative caches.

- With lock-up free caches, there is no need to use synchronization primitives.

- A TLB is still necessary for performance/protection reasons when using a virtually addressed, virtually tagged L1 cache.

- In shared-bus multiprocessor systems, it is feasible to use a directory protocol rather than a snoopy protocol.

- In NUMA multiprocessors, it is feasible to use a snoopy protocol rather than a directory protocol.

2. (12 points for part (a) and 18 points for part (b). The two parts are independent of each other)

In the next two pages, you'll find tables showing the state of the Reservation stations, Reorder buffer, and FP register status for an out-of-order execution and in-order completion processor implementing Tomasulo's algorithm at a given stage of execution of a code sequence (instruction formats are similar to those you worked with in Assignment #3). Filling of the tables has been done assuming the following:

- Only *one instruction* can *issue* per cycle
- The *reorder buffer* has *depth 8* (results of loads are forwarded via the common data bus -CDB– to the reorder buffer)
- All *functional units* are *pipelined*.
- There are *2 FP multiply* reservation stations
- There are *3 FP add* reservation stations
- There are *3 integer* reservation stations
- Memory requests occur and complete in one cycle, i.e., barring structural hazards, loads issue in one cycle - say t -, "execute" in the next (t+1), "write" in the next (t+2), and a dependent instruction can start execution one cycle later, i.e., at t+3
- No exceptions occur during the execution of the code
- All *integer* operations (including loads and stores) require *1 cycle* in their execute stage
- All FP multiplication operations require *4 cycles* in their execute stage
- All FP addition operations require *2 cycles* in their execute stage
- There is a single CDB and, *on a CDB write conflict, the instruction issued the earliest gets priority*
- Execution for a dependent instruction can begin on the cycle after one of its operand is broadcast on the CDB if all other conditions for its execution have been met
- All reservation stations, reorder buffer, and functional units were empty and not busy when the code started execution
- Integer registers are not shown in the tables and you don't have to show their state
- When filling up the tables, you may overwrite some entries, e.g., from "not busy" to "busy" or any other one if some state/value change makes it necessary.
- The "Value" column gets updated when the value is broadcast on the CDB
- **If you need to make any other assumption, state it clearly on this or the next page**

(a) The tables below show the state at the end of the cycle in which the second SUBI (the one that is commented) from the code below issued. Show the state after the next cycle.

```
LD          F0,  0(R1)
LD          F2,  0(R2)
MULTD       F4, F0, F2
ADDD        F6, F0, F0
SUBI        R1, R1, 8
SUBI        R2, R2, 8          /*this one*/
ADDI        R3, R3, 1
```

Name	Busy	Op	V_j	V_k	Q_j	Q_k	Dest
Add1	Y	ADDD	F0	F0			#4
Add2							
Add3							
Mult1	Y	MULTD	F0	F2			#3
Mult2							
Int1	Y	SUBI	R2	8			#6
Int2	N	LD	R2				#2
Int3	Y	SUBI	R1	8			#5

Table 1: **Reservation Stations**. “F0” in column V_j refers to the value stored in F0. We show a reservation station to be busy even when the associated operation is being executed in the functional unit.

Entry	Busy	Instruction	State	Destination	Value
1	N	LD F0, 0(R1)	Commit	F0	Mem[0([R1])]
2	N	LD F2, 0(R2)	Commit	F2	Mem[0([R2])]
3	Y	MULTD F4,F0,F2	Exec-2	F4	
4	Y	ADDD F6,F0,F0	Exec-2	F6	
5	Y	SUBI R1,R1,8	Execute	R1	
6	Y	SUBI R2,R2,8	Issue	R2	
7					
8					

Table 2: **Reorder buffer**. The notation Exec-i means the ith stage of execution.

Field	F0	F2	F4	F6	F8	F10	...
Reorder #			3	4			
Busy	N	N	Y	Y			

Table 3: **FP Register status**

(b) The tables below show the state at the end of the cycle in which the second MULTD (the one that is commented) from the code below issued. Show the state after ***two*** cycles.

```
MULTD           F0, F2, F4
ADDD            F6, F6, F0
ADDD            F2, F2, F8
LD              F4,  0(R2)
ADDI            R1, R1, 8
MULTD           F8, F10, F12          /*this one*/
ADDD            F4, F4, F10
ADDI            R2, R2, 1
```

Name	Busy	Op	V_j	V_k	Q_j	Q_k	Dest
Add1	Y	ADDD	F6	F0			#2
Add2	Y	ADDD	F2	F8			#3
Add3							
Mult1	N	MULTD	F2	F4			#1
Mult2	Y	MULTD	F10	F12			#6
Int1	Y	LD F4,0(R2)	R2				#4
Int2	Y	ADDI R1,R1,8	R1	8			#5
Int3							

Table 4: **Reservation Stations**.

Entry	Busy	Instruction	State	Destination	Value
1	Y	MULTD F0,F2,F4	Write	F0	F2*F4
2	Y	ADDD F6,F6,F0	Issue	F6	
3	Y	ADDD F2,F6,F8	Exec-2	F2	
4	Y	LD F4,0(R2)	Execute	F4	
5	Y	ADDI R1,R1,8	Execute	R1	
6	Y	MULTD F8,F10,F12	Issue	F8	
7					
8					

Table 5: **Reorder buffer**.

Field	F0	F2	F4	F6	F8	F10	F12	...
Reorder #	1	3	4	2	6			
Busy	Y	Y	Y	Y	Y			

Table 6: **FP Register status**.

3. (This question continues on the next 3 pages. Each subquestion can be answered independently of the others.)

A 32-bit processor has the memory hierarchy for data (we don't consider instruction caches in this question) described below. It is running under an Operating System with a virtual memory paging system. The page size is 8 KB.

- A first level cache L1, 16 KB, 2-way set-associative with a line size of 32 bytes.
- A second level cache L2, 256 KB, 4-way set-associative with a line size of 32 bytes.
- A TLB with 128 entries, 2-way set associative.

Both L1 and L2 are write-back, write-allocate caches.

(a) (4 points)
Give the format of a TLB entry.

(b) (8 points)
Is it possible to have L1 be:

1. Virtually addressed and virtually tagged
2. Virtually addressed and physically tagged
3. Physically addressed and physically tagged

Justify your answers for each of the 3 options above. Are all of these options the same? If not, what differentiates them and what are the pros and cons of each?

Would it be worthwhile to have L2 virtually addressed and physically tagged? Justify your answer.

(c) (12 points)

Assume a memory access time of 100 ns. and no contention in the memory hierarchy. Below is a series of measurements that have been taken for the data fetch portion (i.e., the “execute” part) of a load instruction in the processor with the memory hierarchy described at the beginning of this question.

Give what you think is the most likely explanation of why the data fetch took that long for each of the timings below:

- 3 ns
- 18 ns
- 100 ns
- 200 ns
- 10 ms

What would you guess to be the cycle time of this processor? Justify your answer in one or two sentences.

(d) (10 points)

The processor with the memory hierarchy described at the beginning of this question is part of 4-processor cluster, i.e., the 4 processors form a shared-memory multiprocessor communicating via a single shared bus. Cache coherence at the L2 level is accomplished via a MESI (4 states) snoopy protocol. Multi-level inclusion is enforced, i.e., a block resident in L1 has a corresponding block resident in L2.

In addition to the tags, what other "metadata" is needed for each block of the L1 cache and the L2 cache? You should include those bits that are needed for both single and multiple processor environments. An example of such a bit is the LRU bit used for replacement in the L1 cache.

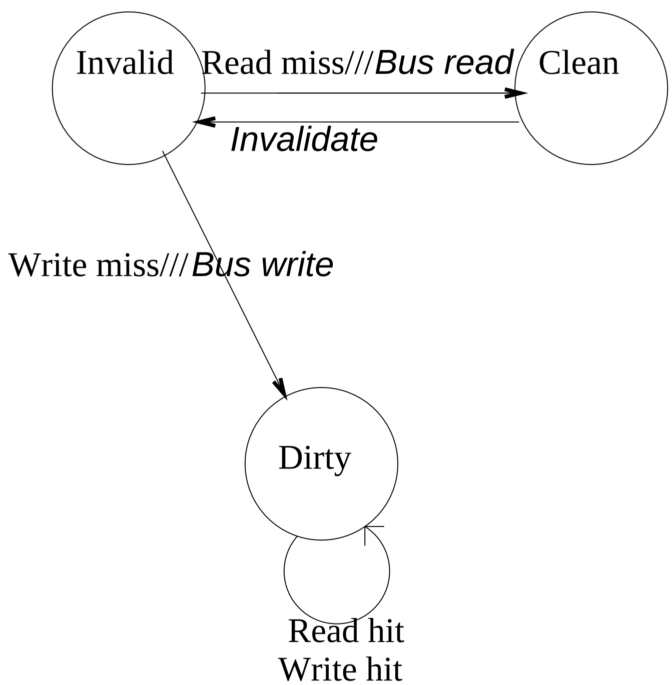
What needs to be changed if L2 had a 64 bytes line instead of a 32 bytes line.

4. (20 points) (This question continues on the next page)

The (incomplete) state diagram below shows the three necessary states for a write-invalidate cache coherence protocol and some of the state transitions.

Transitions written such as:

- "Read hit" corresponds to a CPU event that does not generate a bus transaction.
- "Read miss///*Bus read*" corresponds to a CPU event that also generates a bus transaction.
- "*Invalidate*" corresponds to a transaction initiated by snooping on the bus.



(a) (10 points)

Complete the state diagram above using he same notation. In addition, indicate when a write-back of a block to memory is needed both in the protocol and in case of block replacement.

(b) (10 points)

Assume that 3 processors with caches C1, C2, and C3 are attached to a single shared bus and that they follow the cache coherence protocol of the previous page.

Addresses A and B map to the same blocks in the direct-mapped caches but they do not belong to the same block (i.e, their contents can't both reside in the same cache at the same time). Initially, the entries in the 3 caches, say X1, X2 and X3 corresponding to A (and therefore B) are in state Invalid. Assume that it takes much less than 1000 cycles to complete a miss and associated transactions on the bus.

Show the events that occur (read miss etc.), transactions on the bus (bus read miss etc.), the states of and the values stored in X1, X2 and X3 when the processors execute the following instructions:

```
At time 0,     C1 reads A
At time 1000,  C2 reads B
At time 2000,  C3 reads B
At time 3000   C3 writes B
At time 4000,  C2 writes A
At time 5000,  C3 reads A
```

5. (8 points) (This question continues on the next page)
WARNING: THIS IS A DIFFICULT QUESTION

Improvement in frequency (MHz) does not always improve performance of computer systems. With higher frequencies fewer operations can be performed in a pipeline stage and pipeline depths have to increase. The figure below, taken from a recent article in the Proceedings of the IEEE, measures the impact of increasing the number of pipeline stages on performance using a synthetic model of an in-order superscalar machine.

SORRY THE FIGURE WAS PASTED THERE AND IS NOT AVAILABLE ON-LINE

Give two reasons why performance improve less than linearly with increased frequency.

There are 2 singular points in the graph plotting performance vs. pipeline depth, one at pipeline depth of 13 stages and the other at pipeline depth of 23 stages. Why do you think that these two singular points arise at these pipeline depths (and corresponding frequencies)?