Name _________Solution___________

Email ___________________@mit.edu

Computer System Architecture
6.823 Quiz #4
May 13th, 2015
Professors Daniel Sanchez and Joel Emer

This is a closed book, closed notes exam.

80 Minutes
15 Pages

Notes:
- Not all questions are of equal difficulty, so look over the entire exam and budget your time carefully.
- Please carefully state any assumptions you make.
- Show your work to receive full credit.
- Please write your name on every page in the quiz.
- You must not discuss a quiz's contents with other students who have not yet taken the quiz.

Part A	_________	34 Points
Part B	_________	26 Points
Part C	_________	40 Points
TOTAL	_________	**100 Points**

Name ____________________________

Part A: Reliability (34 Points)

Ben Bitdiddle has a two-core processor with the following characteristics:

- Each core has a single-set, two-way set-associative private cache.
- Caches use the LRU replacement policy.
- Caches use a snoopy, bus-based MSI coherence protocol.
- Each cache line has the following fields: tag, data, and coherence state.
- The coherence state field is two bits (M state = 11, S state = 10, I state = 0X). The high-order bit represents whether the line is valid, and the low-order bit represents whether the line is dirty.
- Both cache lines share a single LRU bit. If the LRU bit is 1, block 1 will be replaced first. Otherwise, block 2 will be replaced.
- All instructions complete in a single cycle, including cache misses and bus transfers.

The structure of the private caches in the processor is shown below.

Core P1

	Cache block 1			Cache block 2		
LRU bit	Coherence state (2 bits)	Tag (3 bits)	Data (8 bits)	Coherence state (2 bits)	Tag (3 bits)	Data (8 bits)

Core P2

	Cache block 1			Cache block 2		
LRU bit	Coherence state (2 bits)	Tag (3 bits)	Data (8 bits)	Coherence state (2 bits)	Tag (3 bits)	Data (8 bits)

Suppose the private caches start with all their bits set to 0. Ben's target program for this multi-core processor is the following code sequence:

```
Time :      Operation :         ACE Instruction? :
Cycle  0    P1: read   0x0A          Yes
Cycle  1    P2: read   0x0A          No
Cycle  2    P2: write  0x0A          Yes
Cycle  3    P1: read   0x0A          Yes
Cycle  4    P1: read   0x0B          Yes
Cycle  5    P1: read   0x0B          No
Cycle  6    P1: read   0x0A          Yes
Cycle  7    P1: read   0x0C          No
Cycle  8    P1: halt                 No
Cycle  9    P2: halt                 No
```

Question 1 (10 points)

Suppose the first load in core P1 brings A to cache block 1. During which cycles are the following bits ACE? Use Y to indicate the bit is ACE, or N to indicate it is un-ACE.

Cycle	0	1	2	3	4	5	6	7	8	9
LRU bit in P1	N	N	N	N	N	N	N	N	N	N

LRU bit is not ACE for the sequence.

P1 Cache block 1

Cycle	0	1	2	3	4	5	6	7	8	9
High-order bit of coherence state	N	N	Y	Y	N	N	N	N	N	N
Low-order bit of coherence state	N	N	N	N	N	N	N	N	N	N

The high-order bit is ace during 2~3, when it is invalidated by P2. If the bit flip from 0 to 1 after P2 writes but before P1 reads, it will read wrong data and tag will match.

Since P1 only reads, the low-order bit does not matter at all.

P1 Cache block 2

Cycle	0	1	2	3	4	5	6	7	8	9
High-order bit of coherence state	N	N	N	N	N	N	N	N	N	N
Low-order bit of coherence state	N	N	N	N	N	N	N	N	N	N

The high-order bit is never ACE since if 1 goes to 0, just read from memory, and if 0 goes to 1, the tag will not match.

Again, since P1 only reads, the low-order bit does not matter at all.

However, since in this question the time line is not defined very precisely, it’s okay to have a shift.

Question 2 (6 points)

What is the **AVF** of the coherence state fields in the private cache of core P1 over cycles 0-9? (Consider the coherence state fields only, not other fields)

AVF = # cycle of bits is ACE / total # cycles = 2 / 40 = 5%

Name ____________________________

Question 3 (12 points)

Ben wants to protect his processor from cosmic rays by adding a protection mechanism. He wants to know the AVF of tag, data, and the LRU bit in the private cache first before adding his mechanism. Help Ben classify these three fields by their AVF into the following three categories: high (AVF near 100%), low (AVF near 0%), and medium (in between). Use one or two sentences to explain your answer for each case.

Tag is medium or low because for false positive is rare, and false negative is dangerous only when data is dirty.

Data is high or medium, depending on the assumption of how is the data being reused.

LRU bit is low because it does not affect the program outcome.

Question 4 (6 points)

After finding the AVFs, Ben decides to add parity bits to all fields (coherence state, tag, data, and LRU bit). However, this causes a large number of false DUE events (detected unrecoverable errors). What structure in the private cache has the largest fraction of false DUE events, relative to their total DUE events?

It is LRU bit. LRU bit always causes false DUE, so the fraction of false DUE relative to its total DUE is 100%.

Part B: Transactional Memory (26 Points)

Ben Bitdiddle wants to implement a transactional memory system with pessimistic conflict detection in a two-core processor. This system has the following characteristics:

- When a transaction starts, it is assigned a unique global timestamp.
- The memory system tracks the set of addresses read or written by each transaction (i.e., its **read set** and **write set**).
- For every transactional load, the memory system checks whether this load reads an address in the **write set** of any other transaction, and declares a conflict if so.
- For every transactional store, the memory system checks whether this store writes an address in the **read set or write set** of any other transaction, and declares a conflict if so.
- On a conflict, the transaction with the later timestamp aborts.
- An aborted transaction restarts execution 10 cycles later.

Ben runs a program with two types of transaction: X and Y, shown below.

Cycle relative to start	Transaction X
Cycle 0	Starts
Cycle 10	Read B
Cycle 20	Read A
Cycle 30	Write A
Cycle 40	Ends

Cycle relative to start	Transaction Y
Cycle 0	Starts
Cycle 10	Read B
Cycle 20	Read A
Cycle 30	Read B
Cycle 40	Ends

Question 1 (6 points)

Suppose the system is executing two transactions: a type X transaction that starts at cycle 0 and receives timestamp 0, and a type Y transaction that starts at cycle 5 and receives timestamp 5. Is there a conflict between these two transactions? If so, at what cycle does this conflict happen?

There is a conflict at cycle 30 due the write A in transaction X.

Question 2 (12 points)

Ben implements conflict detection by extending a conventional MSI coherence protocol. Furthermore, drawing inspiration from the delay invalidation cache coherence protocol in Quiz 3, Ben wants to optimize his transactional memory system as follows:

- When a core receives an abort for its currently running transaction, it delays the abort until the next local cache miss. If the transaction finishes without additional misses, it will commit successfully.

With this optimization, assume the same scenario as in the previous question: a type X transaction that starts at cycle 0 and receives timestamp 0, and a type Y transaction that starts at cycle 5 and receives timestamp 5. Are any of these transactions aborted? If so, when do aborts happen?

No, since the optimization delays the abort for transaction Y, and it does not miss after that, transaction Y will commit. This is logically same as Y starts before X.

Does this optimization always provide correct transactional semantics? Explain your answer in one or two sentences.

No, it does not provide correct transactional semantics. Consider the following example:

Cycle relative to start	Transaction X
Cycle 0	Starts
Cycle 20	Read A
Cycle 30	Use value A to Write C
Cycle 40	Ends

Cycle relative to start	Transaction Y
Cycle 0	Starts
Cycle 10	Write A
Cycle 20	Read B
Cycle 30	Ends

If X starts at 0, and Y starts at 5, Y will abort at cycle 25 due to read miss, but X will read the data from Y since at cycle 20, it sees the write from Y. Finally, X commits will modification that should have abort.

Question 3 (8 points)

Ben believes this optimization works well and always needs fewer cycles to complete transactions. Is he correct? If so, explain why this always improves performance with one or two sentences. Otherwise, provide an example where this optimization causes a transaction to finish later.

No, Ben is incorrect. This optimization is somehow similar to optimistic conflict detection, so it's possible that it takes longer to finish transactions. For example, if a transaction should have abort at cycle 10, but delay the abort till later, it will start later and thus finish later.

Part C: VLIW, Vector Machines, and GPUs (40 Points)

Consider the following C code fragment:

```
for(int i = 0; i < 301; i++)
{
    if(A[i] != B[i])
        C[i] = A[i] + 1;
    else
        C[i] = A[i] - 1;
}
```

A, B and C are arrays of 301 integers each. (Note: sizeof(int) = 4 bytes). Assume that A, B and C are stored in non-overlapping regions of memory.

The MIPS assembly for this code is shown below.

```
# R1 points to A[0]
# R2 points to B[0]
# R3 points to C[0]
# R4 contains a value of 301

loop:   LW      R5, 0(R1)
        LW      R6, 0(R2)
        BEQ     R5, R6, else
        ADDI    R5, R5, #1
        J       next
else:   ADDI    R5, R5, #-1
next:   SW      R5, 0(R3)
        ADDI    R1, R1, #4
        ADDI    R2, R2, #4
        ADDI    R3, R3, #4
        ADDI    R4, R4, #-1
        BNEZ    R4, loop
```

In the rest of the problem, assume that <u>load instructions that hit in the cache take 4 cycles</u> (i.e., if load instruction I1 starts execution at cycle N, then instructions that depend on the result of I1 can only start execution at or after cycle N+4) while <u>all other instructions take 1 cycle</u>. Assume the data cache has two read ports, two write ports, and is pipelined (i.e., it can accept a new request every cycle). Also assume perfect branch prediction and 100% hit rate in the instruction and data caches.

Name ______________________________

Question 1 (15 points)

Consider a VLIW processor. Each instruction can contain up to two integer ALU operations (including branches) and two memory operations. In addition, in this machine, any operation can be predicated with any general-purpose register. For example:

`[R3] SW R1, 0(R2)` executes the store instruction only if `R3` is not zero; similarly, `[!R3] SW R1, 0(R2)` executes the store only if `R3` is zero.

Fill in the following table by unrolling enough loop iterations to eliminate the stall cycles in the main loop. Do not use software pipelining.

Label	Mem	Mem	ALU/Branch	ALU/Branch
	LW R5, 0(R1)	LW R6, 0(R2)		
			ADDI R1, R1, #4	ADDI R2, R2, #4
			ADDI R3, R3, #4	ADDI R4, R4, -1
			SUB R7, R5, R6	
			[R7] ADDI R5, R5, #1	
			[!R7] ADDI R5, R5, #-1	
	SW R5, -4(R3)			
loop:	LW R5, 0(R1)	LW R6, 0(R2)		
	LW R8, 4(R1)	LW R9, 4(R2)		
			ADDI R4, R4, #-2	ADDI R3, R3, #8
			ADDI R1, R1, #8	ADDI R2, R2, #8
			SUB R7, R5, R6	
			[R7] ADDI R5, R5, #1	SUB R10, R8, R9
			[!R7] ADDI R5, R5, #-1	[R10] ADDI R8, R8, #1
	SW R5, -8(R3)			[!R10] ADDI R8, R8, #-1
	SW R8, -4(R3)		BNEZ R4, loop	

Name ____________________________

Question 2

Now consider a vector machine. In addition to scalar registers, the machine has 32 vector registers, each 32-elements long. Vector instructions are described in the following table.

Instruction		Meaning
MTC1	VLR, Ri	Set VLR (vector length register) to the value of register Ri.
CVM		Set all elements in vector-mask (VM) register to 1.
LV	Vi, Rj	Load vector register Vi from memory starting at address Rj (under mask vector).
SV	Vi, Rj	Store Vi to memory starting at address Rj (under mask vector).
ADDVV	Vi, Vj, Vk	Add elements of Vj and Vk and then put each result in Vi (under mask vector).
ADDVS	Vi, Vj, Rk	Add Rk to each element of Vj and then put each result in Vi (under mask vector).
SUBVV	Vi, Vj, Vk	Subtract elements of Vk from Vj and then put each result in Vi (under mask vector).
SUBVS	Vi, Vj, Rk	Subtract Rk from elements of Vj and then put each result in Vi (under mask vector).
S--VV	Vi, Rj	Compare the elements (EQ, NE, GT, LT, GE, LE) in Vi and Vj. If the condition is true, put a 1 in the mask vector (VM), otherwise put 0.

Question 2-1 (10 points)

Rewrite the code fragment for this vector machine by filling in the table on the next page. For your convenience, part of the assembly code is already written for you. You may not need all the rows.

Name ______________________________

```
# R1 points to A[0]
# R2 points to B[0]
# R3 points to C[0]
# R4 contains a value of 301
```

Label	Instruction	Comment (Optional)
	ADDI R7, R0, #1	Set R7 to 1
	ANDI R5, R4, #31	Set R5 to R4%32
	MTC1 VLR, R5	Set VLR to R5
	SLL R6, R5, #2	Set R6 to R5*4
loop:	CVM	Set all elements in mask to 1
	LV V1, R1	
	LV V2, R2	
	SNEVV V1, V2	
	ADDVS V3, V1, R7	
	SEQVV V1, V2	
	SUBVS V3, V1, R7	
	CVM	
	SV V3, R3	
	ADD R1, R1, R6	
	ADD R2, R2, R6	
	ADD R3, R3, R6	
	SUB R4, R4, R5	
	ADDI R5, R0, #32	Set R5 to 32
	MTC1 VLR, R5	Set VLR to R5
	SLL R6, R5, #2	Set R6 to R5*4
	BGTZ R4, loop	

Name ______________________________

Question 2-2 (5 points)

Suppose this vector machine has four lanes. Each lane has one ALU for adds, one ALU for comparisons, and a load-store unit with one read port and one write port. Both ALUs take a single cycle, and memory takes 4 cycles. Assume we use vector chaining to reduce stalls due to data dependences. The machine can chain a load to an ALU instruction, or an add ALU instruction to a compare ALU instruction. Also assume that the mask register is updated at the end of the cycle when an entire S—VV instruction is finished.

In this question, assume each vector register has at least N elements. If we run the same program but with N iterations (instead of 301) on this vector machine, what is the average number of cycles per element for this loop in steady state for a very large value of N?

The answer to this question is based on the answer of Question2-1. We give you full grades if your calculation is correct based on the program you wrote.

Since the program has N iterations and each vector register has N elements, there is only one iteration.

(1) If we assume that the machine cannot chain a compare ALU instruction to an add ALU instruction:

```
LV       V1, R1         ->   N/4
LV       V2, R2         -> + N/4
SNEVV    V1, V2         -> + 4  (chaining: start after first 4 elements in V2 finish loading)
ADDVS    V3, V1, R7     -> + N/4 (no chaining: start after SNEVV is done)
SEQVV    V1, V2         -> +1 (start a cycle after ADDVS to avoid overwriting mask)
SUBVS    V3, V1, R7     -> + N/4 (no chaining: start after SEQVV is done)
CVM                     -> +1
SV       V3, R3         -> + N/4
```

Since N is very large, the average number of cycles per element is (N*5/4)/N = 5/4

(2) If we assume that the machine can chain a compare ALU instruction to an add ALU instruction:

```
LV       V1, R1         ->   N/4
LV       V2, R2         -> + N/4
SNEVV    V1, V2         -> + 4  (chaining: start after first 4 elements in V2 finish loading)
ADDVS    V3, V1, R7     -> + 1  (chaining with SNEVV)
SEQVV    V1, V2         -> + (N/4 -1) (start after SNEVV is done)
SUBVS    V3, V1, R7     -> + 1  (chaining with SEQVV)
CVM                     -> + 1
SV       V3, R3         -> + N/4
```

Since N is very large, the average number of cycles per element is (N*4/4)/N = 1

Question 3 (10 points)

Suppose we code this program to run on a GPU with N warps. Each warp has 32 threads sharing the same PC and thus executing the same instruction. Assume each operation takes 16 cycles to execute. At most one instruction can be issued per cycle. In this GPU, each lane has one ALU and one load-store unit.

(1) If the machine has 32 lanes, what is the minimum value of N to achieve the highest pipeline utilization?

With 32 lanes, issuing 32 threads in a warp takes 1 cycle (1=32/32). To achieve the highest pipeline utilization, we need at least 16 warps (16 warps = 16 cycle / 1 cycle per warp).

(2) If the machine has 16 lanes, what is the minimum value of N to achieve the highest pipeline utilization?

With 16 lanes, issuing 32 threads in a warp takes 2 cycles (2=32/16). To achieve the highest pipeline utilization, we need at least 8 warps (8 warps = 16 cycle / 2 cycle per warp).