Sample Final 2

Problem 1. 10 Points - Caches

Consider a 8-way set-associative 32KB, VIPT, write-back, write-allocate L1 data cache with an LRU replacement policy. The system uses 32-bit virtual and 56-bit physical addresses. The bits of an address are numbered from 0 (least-significant bit) to 31 (or 55 for physical address).

1. [5 points] When accessing this cache, different parts of the virtual and physical address are used at different time.

```
First, we use the index (bits 6 through 11) from the virtual address to find which
set to look at.

Second, we use the tag (bits 12 through 55) from the physical address to check if any
block in the set has a matching tag.

Finally, we use the offset (bits 0 through 4) from the physical address to get the right
byte in the block.
```

2. [5 points] For each 32-byte block of data in the cache, some extra bits (metadata) are kept in the cache. What are these bits and how many meta-data bits altogether does each cache block have?

```
V (valid) 1 bit/line
D (dirty) 1 bit/line
LRU (LRU) 3 bits/line
tag 44 bits/line
total 49 bits/line x 1024 = 50,176 bits (2^15 / 2^5 = 2^10 = 1024)
```

Problem 2. 15 Points - More caches

A processor has a 16-way set-associative, write-back, write-allocate, physically indexed, 4kB cache with 16-byte lines. Starting with an empty cache (all lines invalid), the processor begins zeroing out the elements of an array that begins at physical address 0x40fbfca8 and is 0x4000 bytes long. Pages are 1MB in size. Each element of the array is four bytes in size and the processor writes a zero into each element from first to last, without accessing any other memory locations.

1. [3 points] By the time the zeroing-out is done, how many cache misses will have occurred?

```
The array's length is equal to 0x400 blocks exactly, but the array uses only half of
the first block (starts in the middle of the block) so it uses another block at the end.
So we access a total of 0x401 blocks, each of them for the first time so there is a
miss for each. 0x401 = 1025 blocks
```

2. [4 points] Which set in the cache will be the first one to not have any non-valid (empty) lines left?

```
The address has 4 offset bits and the number of sets is 4096/16/16 = 16, so there are
16 sets => 4 index bits. The first set that gets a line is set 0xA (0x40fbfca8), then
next set, etc. Then set A gets a second block, then the others do, etc. When A gets
the 16th block, others follow, so A is first to be full. 0xA = 10
```

3. [4 points] By the time the zeroing-out is done, how many write-backs will have occurred?

```
1025 blocks are accessed and all are written, so every replacement will result in a
write-back. At the end there are 256 dirty blocks in the cache, and all others in the
1025 have been replaced, so we had 1025-256=769 replacements.
```

4. [4 points] What is the tag of the last cache line to be written back to memory?

```
The last block that is written back is the 769th block we access.
```

```
The 1st we access is 0x40FBFCA8.
The 2nd block is accessed when zeroing out 0x40FBFCB0.
The 17th block is accessed when zeroing out 0x40FBFDA0.
The 257th block is accessed when zeroing out 0x40FBFCA0 + 0x1000 = 0x40FC0CA0.
The 513th block is accessed when zeroing out 0x40FC0CA0 + 0x1000 = 0x40FC1CA0.
The 769th block is accessed when zeroing out 0x40FC1CA0 + 0x1000 = 0x40FC2CA0.
The tag is 0x40FC2C.
```

Problem 3. 20 Points - Compiler Optimizations for ILP

We have a VLIW processor with 16 registers and the following units: 1) A branch unit (for branch and jump operations), with a one-cycle latency. 2) A load/store (LW/SW operations) unit, with a six-cycle latency. 3) Two arithmetic units for all other operations, with a two-cycle latency. Each instruction specifies what each of these four units should be doing in each cycle. The processor does not check for dependences nor does it stall to avoid violating program dependences â€“ the program must be written so that all dependences are satisfied. For example, when a load-containing instruction is executed, the next five instructions cannot use that value. If we execute the following loop (each row represents one VLIW instruction), which performs element-wise addition of two 60-element vectors (whose address are in R0 in R1) into a third (address is in R2), where the end address of the result vector is in R3, and each element is a 32-bit integer: Note how we increment the vector points while waiting for loads to complete. Also note how each operation reads its source registers in the first cycle of its execution, so we can modify source registers in subsequent cycles. Each operation also writes its result registers in the last cycle, so we can read the old value of the register until then. Overall, this code takes ten cycles per element, for a total of 600 cycles. We will unroll so three old iterations are now one iteration, and then have the compiler schedule the instructions in that (new) loop. What does the new code look like (use only as many instruction slots in the table below as you need) and how many cycles does the entire 60-iteration (20 iterations after unrolling) loop take now?

Label	Branch Unit	LD/ST unit	Arith. Unit 1	Arith. Unit 2
Loop:	NOP	LW R4,0(R0)	ADDI R0,R0,4	NOP
	NOP	LW R5,0(R1)	ADDI R1,R1,4	NOP
	NOP	NOP	ADDI R2,R2,4	NOP
	NOP	NOP	NOP	NOP
	NOP	NOP	NOP	NOP
	NOP	NOP	NOP	NOP
	NOP	NOP	NOP	NOP
	NOP	NOP	ADD R6,R4,R5	NOP
	NOP	NOP	NOP	NOP
	BNE R2,R3,Loop	SW R6,-4(R2)	NOP	NOP

Program

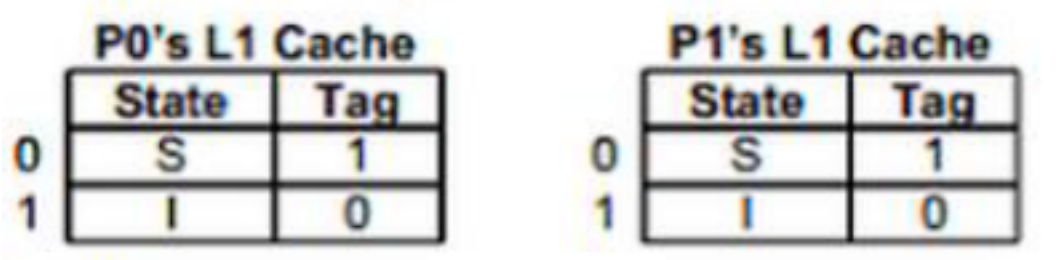
Label	Branch Unit	LD/ST Unit	Arith. Unit 1	Arith. Unit 2
Loop:		LW R4,0(R0)	ADDI R2,R2,12	
		LW R5,0(R1)		
		LW R7,4(R0)		
		LW R8,4(R1)		
		LW R10,8(R1)	ADDI R0,R0,12	
		LW R11,8(R1)	ADDI R1,R1,12	
			ADD R6,R4,R5	
		SW R6,-12(R2)	ADD R9,R7,R8	
		SW R9,-8(R2)	ADD R12,R10,R11	
	BNE R2,R3,Loop	SW R12,-4(R2)		

```
14 cycles * 20 iterations = 280 cycles
```

Problem 4. 25 Points - Cache Coherence

A shared-memory bus-based multiprocessor has two processors. Each processor has a small direct-mapped L1 cache, with only two 8-byte blocks in each cache. All addresses in this problem are 12-bit physical addresses, and all caches are physically indexed and tagged. The MESI coherence protocol is used in this multiprocessor. The initial state for the two caches is with tags of 0 and in I state.

1. [3 points] What is the shortest sequence of accesses that will, starting with the initial state, result in the following cache state: Note: The minimum number of accesses is fewer than five. Space for five is provided just in case you need it. Similar extra space is provided in B), C), etc.



Final State

Access #	Processor	Load or Store	Address
1	0	L	0x10
2	1	L	0x10

2. [4 points] What is the shortest sequence of accesses that will, starting with the initial state, result in the following cache state:

P0's L1 Cache

	State	Tag
0	I	1
1	I	0

P1's L1 Cache

	State	Tag
0	M	1
1	I	0

Final State

Access #	Processor	Load or Store	Address
1	0	L	0x10
2	1	S	0x10

3. [4 points] What is the shortest sequence of accesses that will, starting with the initial state, result in the following cache state:

P0's L1 Cache

	State	Tag
0	I	1
1	I	0

P1's L1 Cache

	State	Tag
0	E	1
1	I	0

Final State

Access #	Processor	Load or Store	Address
1	0	L	0x10
2	1	S	0x10
3	1	L	0x20
4	1	L	0x10

4. [4 points] What is the shortest sequence of accesses that will, starting with the initial state, result in the following cache state:

P0's L1 Cache

	State	Tag
0	S	1
1	I	0

P1's L1 Cache

	State	Tag
0	E	2
1	I	0

Final State

Access #	Processor	Load or Store	Address
1	0	L	0x10
2	1	L	0x10
2	1	L	0x20

5. [5 points] The following state cannot happen:

P0's L1 Cache

	State	Tag
0	S	1
1	I	0

P1's L1 Cache

	State	Tag
0	I	1
1	I	0

Final State

To have the block in I state in P1's cache, P0 must write it after P1 accessed the block for the last time. But then P0 would get it in M or E state -> the only way to get S for P0 is if P1 has the block in a non-I state when P0 accesses it.

6. [5 points] The following state cannot happen:

P0's L1 Cache

	State	Tag
0	I	1
1	I	0

P1's L1 Cache

	State	Tag
0	I	1
1	I	0

Final State

If both have the block in the I state, some other core had to invalidate them, but there is no other core.

Problem 5. 15 Points - Barriers

1. Two shared-memory systems use different implementations of the sense-reversal barrier: Variable locSens is local in each thread, while cntlock, cnt, tot, and rel are global variables shared by all threads. All these variables start out with zero values, except of course cntlock (which is correctly initialized using pthread_mutex_init. Whereas Implementation 1 is correct, Implementation 2 does not. Show a (sequentially consistent) execution order for two threads (tot is 2) in Implementation 2 that leads to incorrect behavior. An execution order (interleaving) can be represented as a sequence of [Core #: Line #] events, using "Work" instead of a line number for actual work between calls to the barrier, e.g. [C1: L1], [C2: L1], [C1: L2,L3,L4], [C2: L2,L3,L4], [C1: L5], [C2: L5], [C1: L6], [C2: L6], [C1: L7], [C2: L7], [C1: Work], [C2: Work], [C1: L1], [C2: L1], etc. leads to correct behavior.

```
[C1, L1] [C1, L2] [C1, L3] [C1, L4] [C1, L5] [C1, L9] (C1 arrives and begins to wait)
[C2, L1] [C2, L2] [C2, L3] [C2, L4] [C2, L5] [C2, L6] (C2 arrives and sets the release var)
[C1, L9] [C1, work] [C1, L1] [C1, L2] [C1, L3] [C1, L4] (C1 arrives again, cnt is now 3)
[C1, L5] [C1, L9] (cnt is 3, not == 2, so C1 waits for rel)
[C2, L7] [C2, work] [C2, L1] [C2, L2] [C2, L3] [C2, L4] [C2, L5] [C2, L9]
Now both waiting at L9 (forever)
```

Problem 6. 15 Points - Storage

A 12,000RPM disk drive has 2 platters, with 100,000 tracks per surface, 1,000 sectors per tracks, and 512 data bytes per sector. Each sector also has a 128-bit error detection code that can detect all errors in its sector but cannot correct any errors. The head takes ten microseconds (a 100,000th cylinder, and multi-cylinder movements are done at the same speed (100,000 cylinders per second). The disk controller is very fast (assume zero latency for everything it does) and the I/O bus has very large (assume infinite) bandwidth.

1. [2 point] How many heads does the disk drive have?

```
2 platters x 2 surfaces / platter (one head / surface)
```

2. [3 points] Assuming that the disk controller and the drive itself are not servicing any other requests, what is the worst-case time needed to read a sector from the disk?

```
99,999 tracks to traverse => 999,990 microseconds
Full rotation to get to start of sector => 1/200 s = 5,000 microseconds
Read sector - 1/1000 of a rotation => 5 microseconds
Total 1,004,995 microseconds
```

3. [3 points] If we use two of these disk drives in a RAID0 configuration, and if the array is full (it contains as much data as its capacity allows), what is the best-case time needed to read all of the data stored in the array? Read both disks simultaneously, so on each we need to read cylinder (all heads read at the same time), move to next, ...

```
99,999 tracks traversed overall => 999,990 microseconds
100,000 track read rotations => 500,000,000 microseconds
Best case for rotational latency is 0 (we arrive just when a sector begins)
If each head reads alone, 4x rotations (to read 4 tracks on each cylinder)
So we would get 2,000,999,990 microseconds instead
```

4. [3 points] If we use five of these disk drives in a RAID5 configuration, and if the array is full (it contains as much data as its capacity allows), what is the best-case time needed to read all of the data stored in the array?

```
Best-case scenario is when a stripe is a full cylinder, so we don't need to visit any
cylinder that contains parity. We still need to skip such cylinders when moving heads,
so 999,990 microseconds total seek line still.
```

We read all 5 disks at the same time, but only read data (4/5 of all cylinders), so 400,000,000 microseconds read time 80,000 cylinders (or 1,600,000,000 if each head reads alone).

Total: 400,999,990 microseconds

5. [4 points] If we use three of these disk drives in a RAID5 configuration and we are extremely lucky, what is the maximum number of sectors that can be damaged in this array that still allow us to recover all of the data? One entire disk

4 x 100,000 x 1,000 = 400,000,000