December 16, 2001

Answers to CSE378 Practice Cache Problems (v2.12)

Problem 1.
Given a 2 way set associative cache with 2 word blocks and a total size of 32 words, show the final state of the cache after the following address stream from the processor occurs.

```
00000080 M
00000084 H
00000088 M
00000144 M
00000148 M
0000014C H
00000150 M
00000160 M
00000080 H
00000084 H
00000088 H
0000008C H
00000090 M
```

Assume that all requests are reads and that the cache is initially empty. Assume LRU replacement policy.
Calculate how many cycles are required to complete this address stream, if cache hit time is 1 cycle, and miss penalty is $10 + W$ cycles, where W is the cache width. (So it takes 11 cycles to read/write the first word, and 1 cycle to read/write the next one).
NOTE: Initially all valid bits are zero, since the cache is empty. I also assumed that set 0 initially had lru bits set to 1.

	set 0					set 1				
Line #	tag	data0	data1	valid	lru	tag	data0	data1	valid	lru
0	2			1	0	5			1	1
1	2			1	0	5			1	1
2	5			1	1	2			1	0
3				0	1				0	0
4	5			1	0				0	1
5				0	1				0	0
6				0	1				0	0
7				0	1				0	0

Address breakdown: the two least significant bits select byte within the word (byte offset), the next bit selects word in the block (word offset), the next three bits select the block or line #, and the rest are used for the tag. Thus tag is $32 - 3 - 1 - 2 = 26$ bits long.
 # cycles: 7 misses at $10 + 2$ cycles each, plus 6 hits at 1 cycle each $= 7 \times 12 + 6 \times 1 = 90$.

Problem 2.
Using the series of references given in Problem 1, show the final state of of the cache for a direct-mapped cache with 8 word blocks and a total size of 32 words. Also calculate the number of cycles required for the address stream, if cache hit time is 1 cycle, and miss penalty is $10 + W$ cycles, where W is the cache width.
Answer:
Address breakdown: the two least significant bits select the byte within the word, the next three bits select the word in the block, the next two bits select the block. The rest $(32 - 2 - 3 - 2 = 25)$ goes in the tag.
Request stream with hits/misses marked:

```
00000080 M
00000084 H
00000088 H
00000144 M
00000148 H
0000014C H
00000150 H
00000160 M
00000080 H
00000084 H
00000088 H
0000008C H
00000090 H
```

Final state of the cache:

Line #	tag	data0-7	valid
0	1	X	1
1		X	0
2	2	X	1
3	2	X	1

 # cycles: 3 misses at $10 + 8$ cycles each, plus 10 hits at 1 cycle each $= 18 \times 3 + 10 \times 1 = 64$.

Problem 3.
To reduce conflict misses, you decide to add a small, 1 entry victim buffer (holds one cache line) to the direct-mapped cache from problem 2. Show the final state of the cache and the victim buffer after the address stream given in problem 1 executes.
Calculate how many cycles are required to complete this address stream, if cache hit time is 1 cycle, victim buffer hit time is 2 cycles, and miss penalty is $10 + W$ cycles, where W is the cache width.
Answer: Same as Problem 2, since there were no conflict misses there. (I meant to make the total size 16 not 32 .. oh well).

Problem 4.
Given a 2 way set associative cache with 2 word blocks and a total size of 32 words, show the final state of the cache after the following address stream from the processor occurs.

```
0FF00F70 (R)
0FF00F60 (W)
0FE0012C (R)
0FF00F5C (R)
0FE0012C (W)
0FE001E8 (R)
0F000F64 (R)
0F000144 (R)
0FE00204 (W)
0FF00F74 (W)
0F000F64 (R)
0FE00EF4 (R)
0F000BB4 (R)
```

Assume LRU replacement policy. Assume write-back, write-allocate policy.
Calculate how many cycles are required to complete this address stream, if cache hit time is 1 cycle, and miss penalty is $10 + W$ cycles, where W is the cache width.
The way this is phrased is ambiguous. What I meant is ***it takes*** $10 + W$ ***cycles to move*** W ***consecutive words between cache and memory.***
Answer.
Lets look at the requests one by one...

0ff00f70 – Read Miss (12 cycles), goes to set 0 line 6.

0ff00f60 – Write Miss. Since we allocate on write, we allocate a cache block for this address, write the value at address 0ff00f60. We also bring in the value stored at 0ff00f64 from memory. Since we're doing write back, we don't write to memory until this block gets kicked out. Turns out caches always read the whole line from the DRAM, so this still takes 12 cycles.

0fe0012c – Read Miss (12 cycles), goes to set 0 line 5.

0ff00f5c – Read Miss (12 cycles), goes to set 0 line 3.

0fe0012c – Write Hit (1 cycle). Just update in the cache.

0fe001e8 – Read Miss (12 cycles), goes to set 1 line 5.

0f000f64 – Read Miss (12 cycles), goes to set 1 line 4.

0f000144 – Read Miss (12 cycles), goes to set 0 line 0.

0fe00204 – Write Miss (12 cycles), goes to set 1 line 0. The write value goes into data1, the value from memory address 0fe00200 goes to data0.

0ff00f74 – Write Hit (1 cycle). Just update in the cache (set 0 line 6).

0f000f64 – Read Hit (1 cycle) – in set 1 line 4.

0fe00ef4 – Read Miss (12 cycles) – goes to set 1 line 6.

0f000bb4 – Read Miss. Based on LRU bits this goes to set 0 line 6, replacing a dirty cache line (0ff00f74 was a write). Assuming no write buffers, this takes 24 cycles – 12 cycles to write back the dirty cache line and another 12 to bring in the new one.

 # cycles $= 12 + 12 + 12 + 12 + 1 + 12 + 12 + 12 + 12 + 1 + 1 + 12 + 24 = 135$

Problem 5.
Now assume the cache from problem 4 is write-through, and write-around (no allocate on write). Calculate the final state of the cache and number of cycles for the address stream.
Answer.
Again, lets look at the requests one by one...

0ff00f70 – Read Miss (12 cycles), goes to set 0 line 6.

0ff00f60 – Write Miss. Since we don't allocate on write, this doesn't go to cache, only to memory. 11 cycles assuming no write buffers.

0fe0012c – Read Miss (12 cycles), goes to set 0 line 5.

0ff00f5c – Read Miss (12 cycles), goes to set 0 line 3.

0fe0012c – Write Hit (11 cycles). Update in the cache and write back to memory.

0fe001e8 – Read Miss (12 cycles), goes to set 1 line 5.

0f000f64 – Read Miss (12 cycles), goes to set 0 line 4.

0f000144 – Read Miss (12 cycles), goes to set 0 line 0.

0fe00204 – Write Miss (11 cycles), goes to memory only.

0ff00f74 – Write Hit (11 cycles). Update in the cache (set 0 line 6) and in memory.

0f000f64 – Read Hit (1 cycle).

0fe00ef4 – Read Miss (12 cycles) – goes to set 1 line 6.

0f000bb4 – Read Miss. Based on LRU bits this goes to set 0 line 6. 12 cycles.

 # cycles $= 12 + 11 + 12 + 12 + 11 + 12 + 12 + 12 + 11 + 11 + 1 + 12 + 12 = 141$

Problem 6.
Repeat problems 4 and 5 for a direct-mapped cache with 4 word blocks and a total cache size of 32 words.
Ask me in office hours :)

Problem 7.
You have a machine configured as follows:
42-bit virtual address
32-bit physical address
32Kb, 2-way set associative cache with 16-word blocks
32-entry TLB (fully associative)
3-level page table (top level has 256 entries, next levels have 1024 entries).
Assume that, as you've previously seen, the VPN in this machine is the same as the cache tag and that the lower bits of the address are concatenated with the PFN to form the physical address.

How big is the Virtual Page Number (VPN)?

This problem requires a bit of calculation.
You always need two bits for the "byte offset" of the word you are loading (in a byte-addressed memory). It is two bits because a word has 4 bytes.
You then need to calculate the number of bits for the word offset. There are 16-word blocks, so you'll need 4 bits for the word offset.
Next, you calculate the number of bits needed for the index. The cache is 32Kb, but each line has 32 words (16-word blocks * 2 blocks). This makes for 256 lines in the cache, requiring 8 bits.
So, the number of bits left for the VPN is 42 bits - 8 bits - 4 bits - 2 bits = **28 bits**.

How big is the Page Frame Number (PFN)?

Recall that the Page Frame Number will be tagged on to the bits indexing the cache. Since the physical address is 32 bits, we subtract the 14 bits to index the cache and we get **18 bits**.

How big is the cache tag?

The cache tag has the same number of bits as the VPN (as stated in the problem), so it has **28 bits**. If this were a physically tagged cache, it would have 18 bits for its tag.

Which bits of the VPN are used for accessing the top-level page table?

The first 8 bits. (Bits 20-27)

Which bits of the VPN are used for accessing the second-level page table?

The next 10 bits. (Bits 10-19)

Which bits of the VPN are used for accessing the third-level page table?

The next 10 bits. (Bits 0-9)

Problem 8.
Using the information in the page tables on the following page, translate the sequence of virtual addresses on the last page (labeled "R"ead or "W"rite) to physical addresses, show the final contents of the TLB and cache, calculate the TLB and cache hit ratios, and update page table bits if necessary. Which references cause page faults? Write protection errors? Note that the virtual address is only 11 bits while the physical address is 16 bits. The machine configuration is a 4-entry, fully associative TLB, a 2-way set associative cache with 4-word blocks and 32 words total, and a 2-level paging mechanism with 4 entries in the top level and 8 entries at the next level.

The TLB:

TLB Entry	VPN	PFN	valid	dirty	prot	LRU
0	0x01	0x100	1	0	1	01
1	0x00	0x110	1	0	1	00
2	0x08	0x220	1	1	0	11
3	0x10	0x200	1	0	1	10

The cache:

	set 0					set 1				
Line #	tag	data0-3	valid	lru	dirty	tag	data0-3	valid	lru	dirty
0	0x01	X	1	1	0	0x00	X	1	0	0
1	0x01	X	1	0	0	0x10	X	1	1	1
2	0x00	X	1	0	0		X		1	
3		X					X			

Table 0x0:

PFN	V	D	P
0x110	**1**	0	1
0x100	**1**	0	1

Table 0x1:

PFN	V	D	P
0x220	**1**	0	0

Top levelpage table:

0x0
0x1
0x2
0x3

Table 0x2:

PFN	V	D	P
0x200	**1**	0	1

Table 0x3:

PFN	V	D	P

Virtual Addresses:

Virtual	Physical	Cache	TLB	Fault?
0x040 (R)	0x4000	M	M	Y
0x044 (R)	0x4004	H	H	N
0x048 (R)	0x4008	H	H	N
0x020 (R)	0x4420	M	M	Y
0x200 (R)	0x8800	M	M	Y
0x024 (R)	0x4424	H	H	N
0x204 (R)	0x8804	H	H	N
0x028 (R)	0x4428	H	H	N
0x208 (W)	0x8808	H	H	N
0x02c (R)	0x442c	H	H	N
0x020 (R)	0x4420	H	H	N
0x208 (R)	0x8808	H	H	N
0x024 (R)	0x4424	H	H	N
0x20c (R)	0x880c	H	H	N
0x028 (R)	0x4428	H	H	N
0x210 (W)	0x8810	M	H	N
0x02c (R)	0x442c	H	H	N
0x020 (R)	0x4420	H	H	N
0x210 (R)	0x8810	H	H	N
0x024 (R)	0x4424	H	H	N
0x214 (R)	0x8814	H	H	N
0x028 (R)	0x4428	H	H	N
0x218 (W)	0x8818	H	H	N
0x04c (R)	0x400c	H	H	N
0x418 (W)	0x8018	M	M	Y
0x050 (R)	0x4010	M	H	N
0x000 (R)	0x4400	M	H	N