Memory sample problems

Assume we have 8GB of word addressable memory with a word size of 64 bits and each refill line of memory stores 16 words. We have a direct-mapped cache with 1024 refill lines. Answer the following questions.

a. What is the address format for memory addresses?
b. What is the address format for the cache?
c. If we changed the cache to a 2-way set associative cache, how does the format change?
d. If we changed the cache to a 8-way set associative cache, how does the format change?
e. If we changed the cache to a fully associative cache, how does the format change?

a. 8GB = 1GW (64 bit word = 8 bytes/word), which requires 30 bits ($\log_2 1G = 30$). The memory address is divided into line number and word number. With 16 words per line, we need 4 bits for the line number leaving 26 bits of the line number: 26 – 4.
b. The direct-mapped cache has 1024 refill lines so requires 10 bits for the line number, leaving 26 – 10 = 16 bits for the tag: 16 – 10 – 4.
c. If we changed the cache to a 2-way set associative cache, we would divide the refill lines in half for each of the two sets, so we would have 512 lines instead requiring 9 bits, the extra bit slides to the tag: 17 – 9 – 4.
d. The 8-way set associative cache would have 128 lines in each of 8 sets, so we need 7 bits for the line number leaving 19 bits for the tag: 19 – 7 – 4.
e. For a fully associative cache, we would have 1024 sets each with 1 line, so there are 0 bits for the line number leaving all 26 extra bits for the tag: 26 – 0 – 4.

Do problem 11 on page 393.

The computer has 8 bit addresses divided up in a cache format of 4 – 2 – 2. The 8 bit addresses are denoted in hexadecimal such as 6E (01101110). The first 4 bits is the tag which is also the first hex digit. The line number is the first 2 bits of the second hex digit. To solve this problem, convert each hexadecimal address to binary and then look at the line number and compare the tag to the tag in the cache. For each miss, replace the line with the new line and update the tag. We will assume that the cache is initially empty so that the first four accesses are misses.

a.

Address	Block 0	Block 1	Block 2	Block3	Result
6E (block 3)				load tag 6	miss
B9 (block 2)			load tag B		miss
17 (block 1)		load tag 1			miss
E0 (block 0)	load tag E				miss
4E (block 3)				load tag 4	miss
4F (block 3)				keep tag 4	hit
50 (block 0)	load tag 5				miss
91 (block 0)	load tag 9				miss
A8 (block 2)			load tag A		miss
A9 (block 2)			keep tag A		hit
AB (block 2)			keep tag A		hit
AD (block 3)				load tag A	miss

93 (block 0)	keep tag 9		hit
94 (block 1)		load tag 9	miss

Out of 14 cache accesses, we have 4 hits, or a hit rate of 4 / 14 = 0.286 (awful!)
b. The final blocks in cache are 9, 9, A and A

Do problems 3 and 5 from page 392.

3: 2^{32} = 4GB, byte addressable, so 32 bit addresses. Lines contain 64 bytes and there is a direct-mapped cache of 512 bytes.

a. How many blocks of main memory are there? 4GB / 64 bytes/block = 64M (64 million)
b. What is the address format? There are 64 bytes per line requiring 6 bits. With 64 bytes per block and a cache of 512 bytes, there are 512 / 64 = 8 blocks requiring 3 bits. This leaves 32 – 3 – 6 = 23 bits for the tag: 23 – 3 – 6.
c. Address 13A4498A = 0001 0011 1010 0100 0100 100**1 10**00 1010. We want the bits denoted in bold as the block number, so this particular address would be mapped to line 110 or line 6 in the cache.

5: 2^{24} = 16MB, byte address, 24 bit addresses, lines of 64 bytes and a fully associative cache of 128 blocks.

a. How many block of main memory? 16MB / 64 bytes = 256K (a quarter million)
b. What is the address format? There are no line numbers, so we just need to determine the number of bits for the word (byte) which is 6, so the tag is 24 – 6 = 18, or 18 – 0 – 6.
c. Where does address 01D872 map? Anywhere (line 0 in any of the 128 sets).

Assume we have a memory hierarchy with 2 levels of cache, an L1 cache with a hit time of .2 ns and a hit rate of 94%, an L2 cache with a hit time of 0.5 ns and a hit rate of 97.7%, and main memory with a hit time of 25 ns and a hit rate of 100%.

a. What is this hierarchy's effective access time?
b. We add an L3 cache with a hit time of 1.2 ns and a hit rate of 99.92%. What is the new effective access time?

a. EAT = .2 ns + .06 * (.5 ns + .023 * 25 ns) = .265 ns
b. EAT = .2 ns + .06 * (.5 ns + .023 * (1.2 ns + .0008 * 25 ns)) = .232 ns

We have a word addressable computer with 8GB and 64 bit words where the page size is 1024 words. We have a program of 91KB. Answer the following questions.

a. What is the virtual memory address format?
b. What is the physical memory address format?
c. How many pages is the program?
d. How many frames are there in memory?

Program is 91KB which is 11648 words which requires 14 bits. A page size is 1024 words which requires 10 of the 14 bits for the offset leaving 4 bits for the page number. Memory is

8GB which is 1GW which requires 30 bits. The frame offset is the same as the page offset so requires 10 bits leaving 20 bits for the frame number.

a. 4 – 10
b. 20 – 10
c. Number of pages = 11648 / 1024 = 11.375, so 12 pages (note: 16 pages is not accurate although an understandable answer if you just did 2^4 since there are 4 bits for the page number)
d. Number of frames = 2^{20} = 1M frames

A word addressable computer has 4GB and a 64 bit word size. The page/frame size is 4096 words. A program is running and has the following page table. Answer the following questions.

Page #	Frame #	Valid
0	301	T
1	985	T
2	4531	T
3	----	F
4	----	F
5	----	F
6	16	T
7	----	F
8	2268	T
9	----	F
10	87	T
11	669	T
12	0	T
13	----	F

a. Where would virtual address 0110111111110000 map to?
b. Where would virtual address 1101000000000101 map to?
c. Where would virtual address 1000100010001000 map to?
d. How large is the program in bytes (approximately)?
e. How much (percentage) of the program is currently in memory?

We can see that this program has 14 pages, so the page number is the first 4 bits of the address and the page offset is the last 12 bits ($\log_2 4096$). To determine the frame size, we need to figure out how many frames are in memory. With 4GB, we have 512MW or a 29 bit address. Since the frame offset is 12 bits, we have 17 bits for the frame. Therefore we have to convert the frame number into 17 bits and take on the offset.

a. 0110 111111110000 – page 6 = frame 16 = 00000000000010000 111111110000
b. 1101 000000000101 – page 13, page fault
c. 1000 100010001000 – page 8 = frame 2268 = 00000100011011100 100010001000
d. The program consists of 14 pages and each page is 4096 words = 14 * 4096 = 57344 words = 458,752 bytes. This is the upper limit. There could be as few as 1 word on page 14 meaning that it would instead be 425,992 bytes.
e. 8 of the program's 14 pages are resident in memory, so 57.1% of the program is in memory.

A computer has 8GBs of word addressable memory with a word size of 32 bits with 1024 byte page/frame sizes. A program is 812 MBs. The page table will store for each page of this program the frame number (if in memory) and a valid bit. Determine how many pages this program consists of and then the size of the table page.

> Memory is 2GWs with each frame being 256 words or 8M frames. The program consists of 812 MB / 4 bytes/word = 203M words. Each page is 256 words, so there are 203M / 256 = 812K pages (831488 pages). Thus, the page table will consist of 831,488 pages. Each entry in the page table will consist of a frame number and a valid bit. With 8M frames, we need 23 bits for the frame number. With the valid bit, we need 24 bits per entry or 3 bytes. The size of this page table then is 831,488 * 3 bytes = 2,494,464 bytes (more than 2M bytes!)

Assume that the memory access process is as follows:

> CPU generates the effective address, use TLB and/or page table to obtain the physical address (page faults are not determined in this phase). Given the physical address, go down the memory hierarchy until the address is found: L1 cache, L2 cache, main memory, swap space. Assume the following hit times and hit rates:
>
> > TLB: .1 ns, 93.5%
> > L1: .1 ns, 96.8%
> > L2: .8 ns, 99.1%
> > Main memory: 20 ns, 100% (page table), 99.99971% (data/instruction access)
> > Swap space: 1,000,000 ns, 100%
>
> Compute the effective access time. What hit time or hit rate has the greatest impact on this value?
>
> EAT = TLB/Page access + Memory hierarchy access =
>
> > .1 ns + .065 * 20 ns +
> > .1 ns + .032 * (.8 ns + .009 * (20 ns + .0000029 * 1,000,000 ns))
> > = 1.4 ns + .132 ns = 1.53 ns
>
> The greatest impact is the TLB miss rate. If the miss rate were improved to say 96%, the EAT would reduce to 1.03 ns, a speedup of nearly 50%.