JOHNS HOPKINS UNIVERSITY | Engineering for Professionals

Module 10 Example Set 2

1. An instruction cache (I-cache) has a storage capacity of 524288 bytes and employs 128-byte cache lines. The memory with which the cache is used is 209715200 bytes in size.

   a) How many instructions can each set hold at one time if the I-cache has a 2-way set associative organization?

   Each set would contain two 128-byte lines. Each line holds 128/4 = 32 instructions, so each set can hold 2*32 = 64 instructions.

   b) How many sets are in the cache?

   There are two lines per set, so each set is 2*128 = 256 bytes in size.

   The number of set is the total cache size divided by the set size = 524288/256 = 2048.

   c) How many blocks are there in the memory if it is used with this I-cache?

   Since the memory block size is always the same as the cache line size (128 bytes in this case), the number of blocks is just the memory size divided by the block or line size: 209715200/128 = 1638400 blocks.

   d) To which set would the instruction at address 0x4008400C map if the I-cache were organized as a 4-way set associative cache?

   In this case, each set would contain 4 lines and is therefore 4*128 = 512 bytes in size. So the number of sets is 524288/512 = 1024, Therefore the set number field requires 10 bits. $\log_2(1024) = 10$. The width of the offset is 7 bits [$\log_2(128)$] and the following address format applies:

Tag	Set number	Offset
31 17	16 7	6 0

   0x4008400C = binary 010000000000100 0010000000 0001100

   This address maps to set 0010000000 = 0x080 = 128

   e) To which memory block would this address correspond?

   The memory blocks are the same size as the cache lines (128 bytes), and thus have the same size offset field (i.e. 7 bits). The bits to the left of the offset field would give the block number. This is the same number obtained by the integer division of the address/128.

   The block number is binary 0100000000000100001000000 = 0x801080 = 8392832

f) How many different memory blocks could potentially map to set 10 within the 4-way set associative cache?

Any memory block whose block number has bits 16 through 7 equal to 0000001010 would map to set 10. Since the memory is 209715200 bytes in size (=0xC800000), so only the low 28 bits of the address will ever be none-zero. That is, at most 28 bits are needed to specify any address within the memory. So the upper 28-10-7 = 11 bits would determine how many different blocks map to an individual set. The value in the upper 11 bits range from 0 to 11000111111 = decimal1599. So 1600 different blocks map to set 10 within the 4-way set associative cache.

2. A system has a unified cache with a 10ns access time, and a memory with a 120 ns access time.

a) With the cache operating in look-through mode, the effective access time for reads is 22ns. What would be the corresponding hit ratio for reads?

With look-through mode, the effective access time for reads is
T = 22 = 10 + (1-h)*120 = 130 - 120h where h is the hit ratio.
Hence h= 108/120 = 0.9 or 90 %

b) If the system has a read hit ratio of 85% with the cache operating in look-aside mode, what is the corresponding effective access time for reads?

With look-aside mode, the effective access time for reads is
T= h*10 + (1-h)120 = 0.85*10 + 0.15*120 = 26.5 ns.

3. On a system that employs multiprogramming with virtual memory, how is one process prevented from accessing or overwriting the pages that belong to another process? The OS would insure that the frames assigned to one process never match those assigned to the other. This is done by making sure that the page map table for the two processes contain distinct and different page to frame mappings.

4. What difference does it make whether a system, containing a cache and employing virtual memory, uses the 32-bit physical or 32-bit virtual address to access the cache when running two different processes concurrently?

None of the physical addresses generated by one process would match those generated by another process, so the cache would only be accessed using different and distinct physical addresses.

With virtual addresses, two different processes could reference the same virtual address even though their respective page map tables would translate the common virtual address to different physical addresses. Therefore the cache could be referenced with the same virtual address by two different processes.

5. A system implements a paged virtual address space for each process. The page size is 1024 bytes. The maximum size for the virtual address space is 128×2^{20} bytes and the physical address space size is 2×2^{20} bytes. The page table for the running process includes the following five valid entries (where the => notation indicates that a virtual page maps to a given frame (i.e., the page is located in that frame):

Virtual page 2 => frame 4	Virtual page 4 => frame 9
Virtual page 1 => frame 2	Virtual page 3 => frame 16
Virtual page 0 => frame 1	

a) What is the minimum number of bits required to cover the virtual address space? 27 bits are required to cover a 128 MB address range.
$\text{Log}_2(128\times 2^{20}) = 7+20=27$ bits.

b) What is the minimum number of bits required to cover the physical address space?
21 bits are required to cover a 2 MB address range.
$\text{Log}_2(2\times 2^{20}) = 1+20=21$ bits.

c) What is the maximum number of entries in a single-level page map table? The maximum number of PTEs would be the same as the maximum number of pages = 128×2^{20} /1024 = 128 * 1024 = 131072.

d) To which physical address will the virtual address decimal 1524 translate? Decimal 1524 = 0x5F4. Ten bits are required for the page offset within the 1024-byte pages. The leftmost 27-10 = 17 bits within the virtual address gives the page number. Hence the page number is 1. Page 1 maps to frame 2 therefore the corresponding physical address is binary 100111110100 = 0x9F4 = decimal 2548.

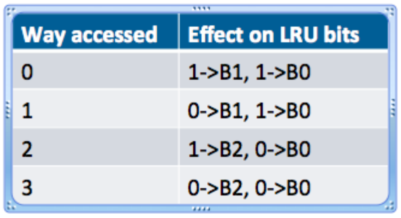
e) Which virtual address will translate to physical address decimal 17880? Decimal 17880 = 0x45D8, so the frame number = binary 010001 = 17 (the leftmost 11 bits of the physical address). None of the valid PTEs correspond to frame 17, so no virtual address would translate to this physical address until some page has been brought into memory and assigned to frame 17.

6. A system with $4 * 2^{30}$ bytes (4 GB) of memory employs a unified fully associative cache with a line size of 256 bytes and a total of 8192 cache lines. Currently line 768 within the cache is valid and has a tag = 0x23E450. Which memory block currently occupies line 768 within the cache?
The memory block size is always the same as the cache line size (256 bytes in this case). The offset field requires 8 bits to cover the range 0 through 255. Hence the tag is made up of the bits to the left of this 8-bit offset field. Given any memory address for this system, the block number would be the bits to the left of the offset field. Hence the block number and the tag are the same. So line 768 contains memory block 0x23E450.

7. a) How many blocks would a system that contains $4 * 2^{30}$ bytes (4 GB) of memory contain if the block size is 4096 bytes?
This would just be the total memory size / blocksize = $2^{32} / 2^{12} = 2^{20}$.
b) To which memory block would the address 0x209788CE correspond?
Block number = (address / block size) = 0x209788CE / 4096 = 0x20978.
Note that the same result can be obtained by truncating the low 12 bits of the address.

8. Set 10 within a 4-way set associative cache is currently full and the pseudo-LRU bits for set 10 were most recently updated from 110 to 101.

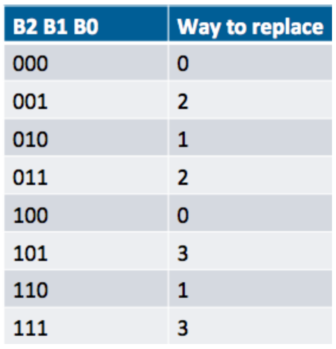
a) What line within set 10 was most recently accessed?
If the pseudo-LRU bits go from 110 to 101, then B1 went from 1 to 0, and B0 went from 0 to 1. These are the updates that would be made if way 1 within the set is accessed as shown in the table below from module 10:



Way accessed	Effect on LRU bits
0	1->B1, 1->B0
1	0->B1, 1->B0
2	1->B2, 0->B0
3	0->B2, 0->B0

b) If the very next memory access corresponds to a hit within way 3 of set 10, to what 3-bit pattern should the pseudo-LRU bits be changed?
Since way3 is accessed, B2 should go to 0 and B0 should go to 0. Hence the pseudo-LRU bits change from 101 to 000.

c) If the next memory access results in a miss in set 10, what action should be taken?
In response to the miss, a line within the full set 10 will have to be replaced. Since the pseudo-LRU bits are 000, way0 should be replaced as shown in the table below:



B2 B1 B0	Way to replace
000	0
001	2
010	1
011	2
100	0
101	3
110	1
111	3

Replacing way0 is also a reference, so the pseudo-LRU bits should be updated from 000 to 011 based on the table in part a).

9. A matrix of 32-bit integers, x[512][32] resides at memory address 3EA4000 on a machine with a virtual memory system that employs 8192-byte pages. The matrix contains 512 rows and 32 columns. Assume that the amount of physical memory available to contain pages from the matrix is 32768 bytes. Note that in the C language, matrices are stored in row major order (i.e. one row after another). Recall that memory is byte addressable and its size is measured in units of bytes not bits.

a) How many frames are required to hold the complete matrix?

There are 512 rows. Each row contains 32 elements. Each element is 4 bytes in size. Hence the total size of the matrix is 512*32*4 = 65536 bytes. Frames and pages have the same size (8192 bytes in this case). Hence the matrix corresponds to 65536/8192 = 8 frames.

b) How many different pages are referenced by the following code sequence:

```
 for (j = 0; j < 3; j++) {
            for (k = 0; k < 35; k++)    x[k][j]  = x[k][j] + 1;
        }
```

One row of the matrix corresponds to 32 four-byte elements or 128 bytes. Hence each page can hold 8192/128 = 64 rows. The code references columns 0, 1 and 2 within each of the first 35 rows. The first 64 rows of the matrix are contained in a single page, so the code only references a single page.