ECS 154A **Homework #5 (74 points)** **Fall 2014**

Due Friday, December 12th, Written homework: 4:00pm. Programs: 11:59pm in p5 of cs154a using handin.

Written Assignment (44 points)

Assume all memory is byte addressable unless stated otherwise.

1. (5 points) Here is a string of hex address references given as byte addresses:
   1, 2, 3, 1A, A, 1B, 16, 14, 3, 12, 9, 23, 3A, 5, 19, 1, 9
   a) Assuming a direct mapped cache with 16 one-byte blocks that is initially empty, label each reference in the list as a hit or miss and show the final contents of the cache tag array. Compute the hit rate for this example.

   b) Show the hits and misses and final cache contents for a direct mapped cache with four-byte blocks (lines) and a *total* size of 16 bytes. Compute the hit rate for this example.

   c) Show the hits and misses and final cache contents for a two-way set associative cache with one-byte blocks (lines) and a *total* size of 16 bytes. Assume LRU replacement. Compute the hit rate for this example.

   d) Show the hits and misses and final cache contents for a fully associative cache with one-byte blocks (lines) and a *total* size of 16 bytes. Assume LRU replacement. Compute the hit rate for this example.

   e) Show the hits and misses and final cache contents for a fully associative cache with four-byte blocks (lines) and a *total* size of 16 bytes. Assume LRU replacement. Compute the hit rate for this example.

2. (2 points) A set associative cache consists of 40 lines divided into 5-line sets. (In other words, it is 5-way set associative). Each line is 4 bytes long. Main memory contains 16K blocks of 64 words each, and a word consists of 4 bytes. Show the format of main memory addresses.

3. (3 points) Consider a memory system that uses a 32-bit address and is byte addressable, and a cache that uses 64 byte lines.

   a) Assume a direct-mapped cache with a tag field in the address of 17 bits. Show the address format and determine the following parameters: number of lines in the cache, the size of the cache, and the size of the tag.

   b) Assume a fully-associative cache. Now how big is the tag?

   c) Assume a 3-way set associative cache with a tag field in the address of 8 bits. Show the address format and determine the following parameters: number of lines in the cache, size of the cache, number of lines per set, number of sets in the cache, and the size of the tag.

4. (5 points) Consider a machine with a byte addressable main memory of 2^16 (65536) bytes, which has a direct-mapped cache with 32 lines. Lines are 16 bytes long.
   a) How is the 16-bit memory address partitioned by the cache? (In other words, how big is the tag field, the entry into the line, etc.)?
   b) Into what line would bytes with each of the following addresses be stored?
      0x111B
      0xC334
      0xD01D
      0xAAAA
   c) Suppose the byte with address 0x2B2B is stored in the cache. What are the addresses of the other bytes stored along with it?
   d) How many total bytes of memory can be stored in the cache?
   e) Why is the tag also stored in the cache?

5. (5 points) Design a byte-addressable memory system with a total capacity 4096 bits using SRAM chips of size 64x1 bit. Give the array configuration of the chips on the memory board showing all required input and output signals for assigning this memory to the lowest address space.

6. (4 points) Consider a memory system with the following parameters:
   Tc (cache access time) = 2ns Cc (cache cost) = .001 cents/bit
   Tm (memory access time) = 300ns Cm (memory cost) = .00005 cents/bit

   a) What is the cost of 1 Megabyte of main memory?
   b) What is the cost of 1 Megabyte of cache memory?
   c) What is the cost of a memory system with a 512 Megabyte memory and a 64Kbyte cache?
   d) If the effective access time is 13% greater than the cache access time, what is the hit ratio H?

7. (3 points) A virtual memory system for a byte-addressable processor with 8-byte words has a page size of 512 words, sixteen virtual pages, and four physical page frames. The page table is as follows:

Virtual Page Number	**Page Frame Number**
0	1
1	3
2	-
3	0
4	-
5	-
6	-
7	-

8	-
9	-
10	-
11	2
12	-
13	-
14	-
15	-

   a) What is the size of the virtual address space? (How many bits in a virtual address?)
   b) What is the size of the physical address space? (How many bits in a physical address?)
   c) What are the physical addresses corresponding to the following decimal virtual addresses (yes, you have to convert from decimal to binary): 0, 3728, 2047, 2048, 10025, 22550, 7596? (Indicate which, if any, cause page faults).

8. (2 points) Give reasons why the page size in a virtual memory system should be neither too large or too small.

9. (3 points) A computer has a cache, main memory, and a disk. If a reference to the cache is a hit, it takes 6 ns to retrieve the data. If a reference misses in the cache, it takes 89 ns to fetch the item from memory and put it in the cache, at which point the request is reissued to the cache. If the required item is not in main memory, it takes 13 ms to fetch the word from the disk, followed by 81 ns to copy the word to the cache, and then the reference is reissued to the cache. The cache hit ratio is .94 and the main memory hit ratio is .84. What is the average time in nanoseconds to access a data item on this system?

10. (3 points) Assume a task is divided into 4 equal-sized segments, and that the system builds an 8-entry page descriptor table for each segment. Thus, the system has a combination of segmentation and paging. Assume also that the page size is 4K bytes
    a) What is the maximum size of each segment?
    b) What is the maximum logical address space for the task?
    c) Assume that an element in physical location 0x1ABC is accessed by this task. What is the format of the logical address that the task generates for it?

11. (3 points) Consider a paged logical address space (composed of 32 pages of 2K bytes each) mapped into a 0.5 MByte physical memory space.
    a) What is the format of the processor's logical address?
    b) What is the length and width of the page table (ignoring any access control bits)?
    c) What is the effect on the page table if the physical memory space is reduced by half?

12. (6 points) The following tables contain information about a segmented, paged virtual memory system and certain select memory locations. Total physical memory size is 2K bytes. All numbers in this table are in Hex unless otherwise noted. The processor is byte-addressable, and uses little-endian storage.

Segment Table		
Entry Number	**Presence Bit**	**Page Table**
0	0	0
1	1	1

Page Table 0			
Entry Number	**Presence Bit**	**Disk Address**	**Physical Page Number**
0	0	0x443BH096	0
1	1	0x08D22108	3
2	1	0xF0871A09	1
3	0	0x7BA54C21	2

Page Table 1			
Entry Number	**Presence Bit**	**Disk Address**	**Physical Page Number**
0	1	0x88B04136	2
1	0	0xEF444219	0
2	1	0x00222957	3
3	1	0x28756554	1

Physical Memory Address	Contents
0x02A4	0x7230
0x03A4	0x86a9
0x04A4	0x9723
0x05A4	0x3423
0x06A4	0x8876
0x0FA4	0x2373
0x11A4	0x1346
0x17A4	0x6792
0x1EA4	0x5292
0x37A4	0x7974
0x3BA4	0x3205
0x67A4	0x6623

a. Assuming 512-byte pages, convert the virtual address 0xFA4 into a physical address.
b. What is the value in memory stored at the physical address corresponding to the virtual address 0xFA4?
c. Repeat (a) and (b) for the virtual address 0x3A4.
d. Repeat (a) and (b) for the virtual address 0xEA4.
e. Repeat (a) and (b) for the virtual address 0xBA4.
f. What changes to this Virtual Memory structure would need to take place if the page size was increased to 1024 bytes?

Cache Design (30 points, 2 hours)

To begin this lab you are to implement a 256 byte Direct-mapped cache with a line size of 4 bytes. The cache is to be byte addressable. The cache will be implemented in C or C++. Your cache will need to support a read operation (reading a byte from the cache) and a write operation (writing a new byte of data into the cache). This cache will support a write-back write policy which will require you to use a dirty-bit. In addition, cache must support a write-allocate write miss policy, in which a write miss causes the appropriate line to be brought into the cache from memory, and the write's value to update the correct part of the line in the cache, which will then become dirty.

After implementing the Direct-mapped cache you will alter it (in a separate file) in order to implement a 160-byte, five-way set associative cache with line size of 4 bytes. The other specifications will remain the same, you must support read

and write operations, as well as a write-back policy, write-allocate policy, and a least recently used (LRU) replacement policy for blocks.

Both caches take as input a filename from the command line. The file specified by the filename will be an ASCII file with each line in the following 4-byte format:

Bytes	Function
1-2	16 bit address
3	*Read/Write
4	8 bits of Data

The read function will be designated by all 0's and the write will be designated by all 1's. Upon a read operation the data segment of the 4-byte format will be ignored. However when a write occurs the data will be written into the specified byte and the dirty bit may or may not be set. For ease of creation the input file will be in hex. For example:

Address	Read/Write	Data
002D	FF	FD
002E	FF	4E
002D	00	28

Which would appear in the input file as:
002D FF FD
002E FF 4E
002D 00 28

The first two lines signify that FD and 4E should be written to the appropriate locations in the cache, and the third line signifies that data should be read from the cache.

The direct mapped cache produces as output a file named dm-out.txt, and the set associative cache produces a file named sa-out.txt. Each line of the output file corresponds to the results of each read operation from the input file. The information on each line will be the byte of data returned by the read operation, a HIT output indicating whether the requested item was found in the cache, and the dirty-bit. These pieces of information should be separated by a space, and for each of the read inputs there should be a line in the output file Thus for the example above we will have a one line output file that would appear as follows:

FD 1 1

You will find a test input file, test-file.txt, along with the corresponding correct output files, dm-test-output.txt and sa-test-output.txt in ~ssdavis/154/hw5 in the CSIF.

An important thing to notice is that when a line gets evicted from the cache, and at some later point is brought back into the cache by a subsequent read, the read must return the correct value (not just zero), as if it was stored in "memory" when it was evicted from the cache. A specific example of this is line 9 in dm-test-output.txt. You can implement this however you like, but a perfectly acceptable way to do it is to have an array of length 2^16 to act as main memory that cache lines get evicted to. Initialize the contents of your cache and memory to all 0's.

Submission Details
You must submit two source files, along with a Makefile that will produce two executables: one named "dmcache" for the direct mapped cache, and one named "sacache" for the set associative cache. Your submission must compile and run on the CSIF machines.

Since there will be no opportunity for re-grading of this assignment, it is VERY important that the following works perfectly:
>make
>./dmcache inputfilename.txt
>./sacache inputfilename.txt
>diff dm-out.txt dmcorrectoutput.txt
>diff sa-out.txt sacorrectoutput.txt