ECS 154A **Homework #5 (66 points)** **Fall 2015**

Due Friday, December 4th, Written homework: 4:00pm. Programs: 11:59pm in p5 of cs154a using handin.
Filenames: dmcache.cpp, sacache.cpp, vm.cpp, authors.csv

Format of authors.csv: author1_email,author1_last_name,author1_first_name
author2_email,author2_last_name,author2_first_name
For example: simpson@ucdavis.edu,Simpson,Homer
potter@ucdavis.edu,Potter,Harry

Written Assignment (35 points)

Assume all memory is byte addressable unless stated otherwise.

1. (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.

2. (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?

3. (9 points) A virtual memory system for a byte-addressable processor with 4-byte words has a page size of 512 words, 16 virtual pages, and eight physical page frames. The page table is as follows:

Virtual Page Number	Page Frame Number
0	1
1	5
2	-
3	4
4	-
5	-
6	-
7	-
8	-
9	-
10	-
11	7
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 hexadecimal virtual addresses: 0, E90, 7FF, 800, 2729, 5816, 1DAC? (Indicate which, if any, cause page faults).

4. (2 points) Give reasons why the page size in a virtual memory system should be neither too large or too small.

5. (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?

6. (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?

7. (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?

8. (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, 1.5 hours)

You are to implement two caches for a simulated RAM that has 16-bit addresses, and is byte addressable. Though addresses are byte addressable, blocks are always read from and written to RAM with addresses that are multiples of eight, e.g., 8, 56, and 416. Both caches will have block sizes of 8 bytes, and a total capacity of 512 bytes.

Your caches 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). Both caches 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.

The first cache, implemented in dmcache.cpp, should rely on direct mapping. After you have dmcache.cpp working properly, you can implement a four-way set associative cache in sacache.cpp that uses 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. For ease of creation the input file will be in hex. 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 address specified in the input line, the 8 bytes of data displayed with the lowest addressed byte on the right, a HIT output indicating whether the requested item was found in the cache (0 for miss, 1 for hit), and the dirty-bit. Note that the dirty bit will displayed as 1 if the original dirty bit was set, even if there is a read miss that will cause the dirty bit to be cleared. These four pieces of information should be separated by a space. Here is an example with some comments and **bold** added by me.

```
[ssdavis@lect1 p5]$ cat dmtest-8-64-40-1.txt  // put into 4 columns to save space
     3F69 FF E2           3F6F FF 23           3F6F FF D9           3F6A FF B1
     856D FF 9A           856E FF 1A           8568 00 00           8568 FF C2
     3F68 FF F9           3F6E FF 66           3F6D FF 51           3F6E FF 4C
     856D FF 0C           3F6A FF 90           3F68 FF E1           856E 00 1A
     856B FF F8           856B FF A1           8568 FF 97           3F6A FF 23
     3F6F FF 87           3F6F FF C8           856C FF 2E           3F6E FF 73
     856F FF DA           3F69 FF 33           856C 00 2E           3F6F FF C2
     3F6A FF F1           856C 00 00           3F6A FF 4C           3F6E FF 8F
     3F6E FF C1           3F69 00 33           856C FF 9A           3F6C 00 00
     856B FF 69           8569 00 00           3F6B FF DA           856F FF F8
[ssdavis@lect1 p5]$ dmcache.out dmtest-8-64-40-1.txt
[ssdavis@lect1 p5]$ cat dm-out.txt
856C DA1A0C00A1000000 0 1       // 856C 00 00
3F69 C8660000009033F9 0 0       // 3F69 00 33
8569 DA1A0C00A1000000 0 0       // 8569 00 00
8568 DA1A0C00A1000000 0 1       // 8568 00 00
856C DA1A0C2EA1000097 1 1       // 856C 00 2E
856E DA1A0C9AA10000C2 0 1       // 856E 00 1A
3F6C C28F5100DA2333E1 1 1       // 3F6C 00 00
[ssdavis@lect1 p5]$
```

You will find a test input files, their corresponding correct output files, my executables, dmcacheTest.out, and sacacheTest.out in ~ssdavis/154/p5. dmcacheTest.out and sacacheTest.out can be used to create input files. The corresponding output files are created by supplying the input files to my own versions of dmcache.out and sacache.out.

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 RAM 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 chars to act as RAM that cache lines get evicted to. Initialize the contents of your cache and memory to all 0's.

Virtual Memory Design (10 points, 30 minutes)

Filename: vm.cpp

Implement a virtual memory simulator that uses a 16 line page table, with 4KB pages, a physical address space of 1GB, and a virtual address space of 4GB. Only four of the pages can be in RAM at a time. You must implement the clock replacement algorithm. Your simulator will read a file that contains virtual addresses that are needed. The first 16 lines in the file are the addresses of the first bytes of each page in the page table. The rest of the lines are addresses that are currently needed by the executable. Note that the addresses are not necessarily that of the first byte of a page. After the initial 16, for each address read, your program will write to a file named vm-out.txt a line that lists the virtual addresses (in hex) of the first bytes of the four pages currently in RAM. The list should always start from the first entry in the four-page entry table. Your output file will be diffed with mine. If diff prints any differences, then you will receive a zero. My executable, test files, and a program to create test files are in ~ssdavis/154/p5.

You will find Stallings' virtual memory appendix available on the course website under homework assignments.

```
[ssdavis@lect1 p5]$ cat test20.txt
7464a000
14be9000
22161000
e1ad6000
561f9000
75ac9000
a0fcf000
ad08a000
71919000
29fbb000
8a15d000
2ee9c000
c3ad8000
83c18000
cb6b5000
7be1b000
561f9d96  // start of page accesses
14be9015
cb6b5469
ad08a836
29fbbd57
7464a46c
29fbb83a
7be1b411
cb6b5d07
29fbb6dd
29fbbb49
ad08aa0d
7be1b836
e1ad6777
7191913f
71919bac
14be9289
e1ad6216
14be9494
a0fcfaf3
[ssdavis@lect1 p5]$
```

```
[ssdavis@lect1 p5]$ vm.out test20.txt
[ssdavis@lect1 p5]$ cat vm-out.txt
561f9000
561f9000 14be9000
561f9000 14be9000 cb6b5000
561f9000 14be9000 cb6b5000 ad08a000
29fbb000 14be9000 cb6b5000 ad08a000
29fbb000 7464a000 cb6b5000 ad08a000
29fbb000 7464a000 cb6b5000 ad08a000
29fbb000 7464a000 7be1b000 ad08a000
29fbb000 7464a000 7be1b000 cb6b5000
29fbb000 7464a000 7be1b000 cb6b5000
29fbb000 7464a000 7be1b000 cb6b5000
ad08a000 7464a000 7be1b000 cb6b5000
ad08a000 7464a000 7be1b000 cb6b5000
ad08a000 e1ad6000 7be1b000 cb6b5000
ad08a000 e1ad6000 7be1b000 71919000
ad08a000 e1ad6000 7be1b000 71919000
ad08a000 e1ad6000 14be9000 71919000
ad08a000 e1ad6000 14be9000 71919000
ad08a000 e1ad6000 14be9000 71919000
a0fcf000 e1ad6000 14be9000 71919000
[ssdavis@lect1 p5]$
```