Recent Memory Problems

Question 4 (Retake Spring 2016) Cache Modeling

a) A word-addressable computer has a 2-way set associative 1KB cache with a block size of 64 bits. Assume that the main memory is a 256KB memory. Explain the address format used to access this cache in a 16-bit processor.

256KB = 2^18bytes, 2^18bytes/2^2bytes; addresses are 16 bits
Block size = 64/8 = 8 bytes/4bytes = 2=2^1
1KB/2 = 2^9 bytes for each way, 2^9/2^3 = 2^6 blocks in each way, therefore

Tag: 9 bits
Index/Set: 6 bits
Offset: 1 bit
Byte Offset 0 bits, since blocks are 4 bytes
Block Offset 1 bit, since there are 2 words in each 64-bit block

b) Consider a 2-way set associative 1KB cache with a block size of 16 bytes, where main memory is a byte-addressable 256KB memory. Consider the following read pattern (addresses) repeats for 1000 times assuming that cache replacement policy is LRU:

1. 0x12AB0
2. 0x2A4B1
3. 0x0038F
4. 0x3FEB8
5. 0x2A4BF
6. 0x3FEB0

What is the hit rate? Explain in detail.

256KB = 2^18, addresses are 18 bits
Block size = 16/4 = 2^2 words/block
1KB/2 = 2^9 bytes for each way, 2^9/2^4 = 2^5 blocks in each way, therefore

Tag: 9 bits
Index/Set: 5 bits
Offset: 4 bits
Byte Offset 2 bits, since there are 4 bytes per word
Block Offset 2 bit, since there are 4 words in each 16-byte block

Expanding the memory references into binary, and grouping by Tag, Index, and Offset:

1. *0x12AB0 → 01 0010 1010 1011 0000*
2. *0x2A4B1→ 20 1010 0100 1011 0001*
3. *0x0038F→ 00 0000 0011 1000 1111*
4. *0x3FEB8→ 11 1111 1110 1011 1000*
5. *0x2A4BF→ 20 1010 0100 1011 1111*

6. 0x3FEB0→ 11 1111 1110 1011 0000

From this we see that references 1, 2, 4, and 5 are in the same index/set. Since this is 2-way set associative, it will tolerate some conflicts. 2 and 5 are to the same block which makes it hit after the first compulsory miss. However, 1 and 4 will keep kicking one out causing conflict misses. On the other hand, 6 will always enjoy a hit due to spatial locality of 4. So here are the misses in each iteration:

1st iteration -> M M M M H H
2nd iteration -> M H H M H H
3rd iteration -> M H H M H H
.....
So the result will be ~1/3 misses, ~ 2/3 hits.

*To be exact: Miss Rate = (2 * 999 + 4) / (6*1000) = 2002/6000, so Hit Rate = 3998/6000*

Question 3 b c (Final Spring 2016): Memory

<u>The following 3 parts are independent of each other, you should answer each as if it is a separate question.</u>

b) (Assume you have two CPUs: X and Y, each with one level of cache. The cache size is 128 bytes, the cache block size is 32 bytes, and the cache uses LRU replacement. The only difference between the CPUs is the associativity of the cache: X uses a direct mapped cache, whereas Y uses a fully associative cache. A benchmark is executed assuming that the cache is empty before execution. The cache accesses generated by the program are as follows, in order of access from left to right:

A, B, A, H, B, G, H, H, A, E, H, D, H, G, C, C, G, C, A, B, H, D, E, C, C, B, A, D, E, F

Each letter represents a unique cache block. All 8 cache blocks are contiguous in memory. However, the ordering of the letters does not necessarily correspond to the ordering of the cache blocks in memory.

1) For X, you observe the following cache **misses** in order of generation:

A, B, A, H, B, G, A, E, D, H, C, G, C, B, D, A, F

By using the above trace, identify which cache blocks are in the same set for X processor.

A and B
C and G
H and D
E and F

2) Write down the sequence of cache misses for the Y processor in their order of generation. (Hint: You might want to write down the cache state after each request).

By simulating the cache and using the requests:

Req:	*A*	*B*	*A*	*H*	*B*	*G*	*H*	*H*	*A*	*E*	*H*	*D*	*H*	*G*	*C*	*C*	*G*	*C*	*A*	*B*	*H*	*D*	*E*	*C*	*C*	*B*	*A D*		*E*	*F*
Miss?	*x*	*x*		*x*		*x*				*x*		*x*		*x*	*x*				*x*	*x*	*x*	*x*	*x*	*x*		*x*	*x x*		*x*	*x*

MRU :	*A*	*B*	*A*	*H*	*B*	*G*	*H*	*H*	*A*	*E*	*H*	*D*	*H*	*G*	*C*	*C*	*G*	*C*	*A*	*B*	*H*	*D*	*E*	*C*	*C*	*B*	*A D*		*E*	*F*
:	-	*A*	*B*	*A*	*H*	*B*	*G*	*G*	*H*	*A*	*E*	*H*	*D*	*H*	*G*	*G*	*C*	*G*	*C*	*A*	*B*	*H*	*D*	*E*	*E*	*C*	*B A*		*D*	*E*
:	-	-	-	*B*	*A*	*H*	*B*	*B*	*G*	*H*	*A*	*E*	*E*	*D*	*H*	*H*	*H*	*H*	*G*	*C*	*A*	*B*	*H*	*D*	*D*	*E*	*C B*		*A*	*D*
LRU:	-	-	-	-	-	*A*	*A*	*A*	*B*	*G*	*G*	*A*	*A*	*E*	*D*	*D*	*D*	*D*	*H*	*G*	*C*	*A*	*B*	*H*	*H*	*D*	*E C*		*B*	*A*

The misses are: A B H G E D G C A B H D E C B A D E F

c) In a 64-bit processor, a 16-way set-associative cache is designed where data words are 64 bits long and are addressed to the half-word. Cache holds 2 Mbytes of data with each block holding 16 data words. Each cache line has two additional bits used to indicate whether the cache line is "valid" and/or "dirty".

1) How many bits of tag, index, and offset are needed to support references to this cache?
2) What is the actual size of the cache in hardware?

1) We can calculate the number of bits for the offset first:

- There are 16 data words per block which implies that at least 4 bits are needed
- Because data is addressable to the ½ word, an additional bit of offset is needed
- Thus, the offset is 5 bits

To calculate the index, we need to use the information given regarding the total capacity of the cache:
- 2 MB is equal to 2^21 total bytes.
- We can use this information to determine the total number of blocks in the cache…
2^21 bytes x (1 block / 16 words) x (1 word / 64 bits) x (8 bits / 1 byte) = 2^14 blocks

- Now, there are 16 (or 2^4) blocks / set
Therefore there are 2^14 blocks x (1 set / 2^4 blocks) = 2^10 or 1024 sets
- Thus, 10 bits of index are needed
Finally, the remaining bits form the tag:
- 64 – 5 – 10 = 49
- Thus, there are 49 bits of tag
To summarize: Tag: 49 bits; Index: 10 bits; Offset: 5 bits

*2) Each cache line has 16 words * 64 bits/word = 1024 bits for data + 49 bits for tag + 2 bits = 1075 bits per cache line. There are 2^10 cache lines, therefore 2^10 * 1075 bits in one way. There are 2^4 ways. Total: 2^4 * 2^10 * 1075 bits ~ 2.1MB*

Question 5 (MT #2 Spring 2016) Virtual Memory

You are given the following 4-entry, fully-associative TLB that has a least-recently-used (LRU) replacement policy. Note that '1' means "Most Recently Used and '4' means "Least Recently Used". In addition to the TLB, for a process, the following page table is also given. Virtual memory addresses are given in 16 bits.

TLB

Valid	LRU	Virtual Page	Physical Page

Page Table

1	3	1100	100
1	4	0011	010
1	2	1001	110
1	1	0100	101

Virtual Page Number	**Valid**	**Physical Page Number**
0000	0	001
0001	1	001
0010	1	000
0011	1	010
0100	1	101
0101	0	010
0110	1	011
0111	0	010
1000	0	100
1001	1	110
1010	1	111
1011	0	101
1100	1	100
1101	0	110
1110	0	100
1111	0	001

(a) (3 points) What is the page size?

16 – 4 = 12 bits for page offset.
2^{12} = 4KB pages.

(b) How big is the physical memory?

*2^3 = 8 physical pages exist. 8 * 4KB = 32KB physical memory space is available.*

(c) For the following virtual addresses, what physical address is calculated in **HEX**? In this system, disk access time is 50000 cycles, whereas, memory access time is 100 cycles, and TLB access time is 4 clock cycles. What will be the time required to retrieve the data?

1) 1100 0010 0010 0100
2) 0000 0101 0011 0110
3) 1001 1110 0110 1110
4) 1010 0000 1000 0011

1) *VPN: 1100 → 100 TLB Hit. Physical address: **100** 0010 0010 0100 – 0x4224*
   Time: Access TLB – 4 + Bring data – 100 = 104
2) *VPN: 0000 → TLB Miss. Page Table : **Page Fault** (No physical address)*
   Time: Access TLB – 4 + Page Fault – 100 + Disk access – 50000 = 50104
3) *VPN: 1001 → 110 TLB Hit. Physical address: **110** 1110 0110 1110 – 0x6E6E*
   Time: Access TLB – 4 + Bring data – 100 = 104
4) *VPN: 1010 → TLB Miss. Page Table : Hit -*
   *Physical address: **111** 0000 1000 0011 – 0x7083*
   Time: Access TLB – 4 + Page Table – 100 + Bring data – 100 = 204

(d) Consider a different virtual memory system with 32 bit virtual addresses and 65,536 (2^16) byte pages. Moreover, each page table entry has: 1 valid bit, 1 dirty bit, and the physical page number. If the physical memory is 1GB, how much memory would we need to simultaneously hold the page tables for two different processes?

VPN is 16 bits long – so each page table will have 2^{16} entries
Page offset = 32 – 16 = 16 bits
Physical page number is 30 – 16 = 14 bits

Each page table entry will hold the above 14 bit PPN + 1 valid bit + 1 dirty bit
- Thus, each page table entry is 2 bytes.
- There are 2^{16} 2-byte entries per process
- Thus, each page table requires 2^{17} bytes or ~128 Kbytes
- Because each process has its own page table, a total of 2^{18} bytes or ~256 Kbytes are needed

Question 4 (MT #2 Spring 2016) Cache modeling and tracing

Part 1 An embedded 16-bit processor has 32 Mbytes of main memory, 32 Kbytes of cache, and no hard drive. Memory is byte-addressable. The cache is 8-way set associative, with block size of 8 words.

a) Draw the address format, naming all fields and give the widths in bits.

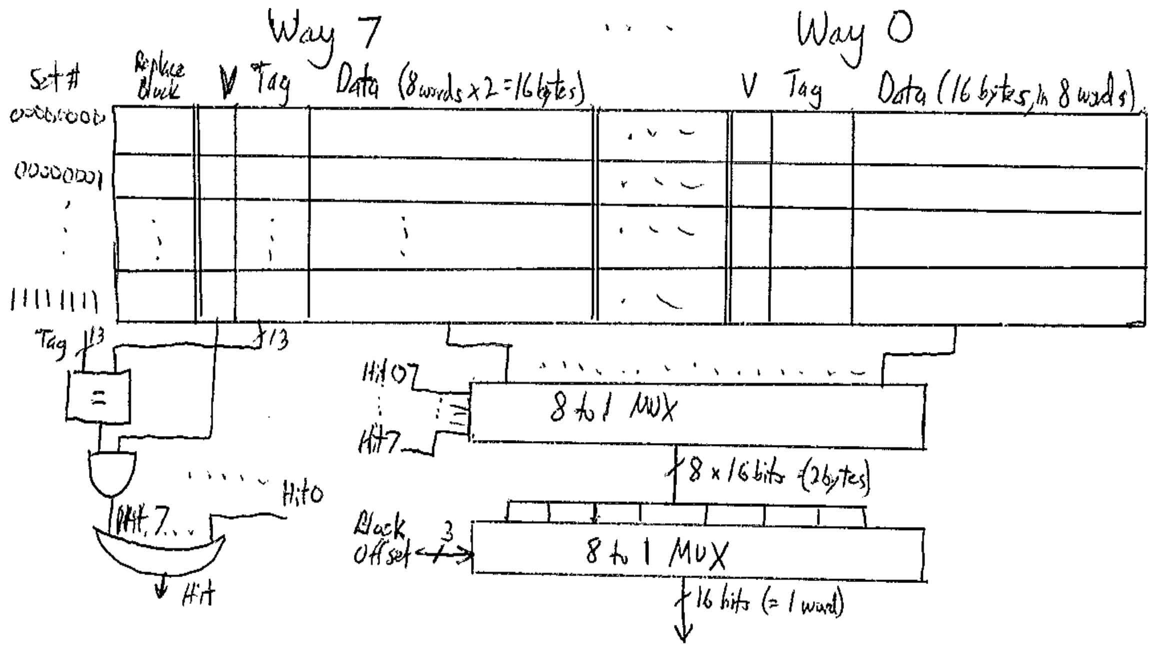
Solution:

25-bit address is needed for 2^{25} = 32 Mbytes memory
16-bit words have 2-bytes per word, so 1 Byte Offset bit is needed
8-word blocks require 3 Block Offset bits, to choose which word in the block
Blocks are 8 x 2 = 16 bytes. There are 8 blocks per set, so each set contains 8 x 16 = 128 bytes.
# of sets in the cache = (2^{15} Bytes/cache) / (2^{7} Bytes/set) = 2^{8} sets/cache
So 8 Set bits are needed, and 25 – (8+3+1) = 13 Tag bits are needed

Tag	***Set***	***Bl.Off.***	***By.Off.***
13 bits	***8 bits***	***3 bits***	***1 bit***

b) For the cache described above, draw the cache diagram, including the storage part and the logic part. Use clever abbreviations and efficient ways to indicate large-size things without drawing all parts of them.

Solution:



Part 2 [Note: this part is independent from Part 1 above] For the cache given below, each block holds 8 bytes of data. Trace the address reference string and detemine for each address access if it is a Hit or Miss in cache, and if it Evicts something (Y/N), giving the results in the table below. For each Miss, identify the type (Cumpulsory, Conflict or Capacity). In the cache diagram, show the final contents of the cache including LRU, V, Tag and Data for each location that contains a block. The LRU bit shows which way is least recently used. Use the right-hand way (Way0) for first placement of a block into an empty set. The memory reference string is 0x0027, 0x0028, 0x0044, 0x004C, 0x01FF, 0x31F3, 0x31FE, 0xFFFF, 0xFFE0, 0x002F, 0x0144, 0x01F8, 0x01F0, 0x0021. [Note: Assume that memory is byte-addressable, and that cache uses LRU replacement.]

Solution: *Since each block holds 8 bytes of data, the block offset + byte offset together must be 3 bits. Since there are 8 sets, the Set field is 3 bits so the Tag must be 10 bits. Therefore the references, in binary, are:*

Tag	*Set*	*Bl/By off*	
0000000000	*100*	*111*	*# 0x0027*
0000000000	*101*	*000*	*# 0x0028*
0000000001	*000*	*100*	*# 0x0044*
0000000001	*001*	*100*	*# 0x004C*
0000000111	*111*	*111*	*# 0x01FF*
0011000111	*110*	*011*	*# 0x31F3*
0011000111	*111*	*110*	*# 0x31FE*
1111111111	*111*	*111*	*# 0xFFFF*
1111111111	*100*	*000*	*# 0xFFE0*
0000000000	*101*	*111*	*# 0x002F*
0000000101	*000*	*100*	*# 0x0144*
0000000111	*111*	*000*	*# 0x01F8*
0000000111	*110*	*000*	*# 0x01F0*
0000000000	*100*	*001*	*# 0x0021*

Tracing these references, using LRU, gives the results in the table and cache below:

Hex addr	**Binary Addr**	**Hit or Miss**	**Evict? Y/N**
0x0027	0000 0000 0010 0111	***Miss-Compulsory***	***N***
0x0028	0000 0000 0010 1000	***Miss-Compulsory***	***N***
0x0044	0000 0000 0100 0100	***Miss-Compulsory***	***N***
0x004C	0000 0000 0100 1100	***Miss-Compulsory***	***N***
0x01FF	0000 0001 1111 1111	***Miss-Compulsory***	***N***
0x31F3	0011 0001 1111 0011	***Miss-Compulsory***	***N***
0x31FE	0011 0001 1111 1110	***Miss-Compulsory***	***N***
0xFFFF	1111 1111 1111 1111	***Miss-Compulsory***	***Y***
0xFFE0	1111 1111 1110 0000	***Miss-Compulsory***	***N***
0x002F	0000 0000 0010 1111	***Hit***	***N***
0x0144	0000 0001 0100 0100	***Miss-Compulsory***	***N***
0x01F8	0000 0001 1111 1000	***Miss-Conflict***	***Y***
0x01F0	0000 0001 1111 0000	***Miss-Compulsory***	***N***
0x0021	0000 0000 0010 0001	***Hit***	***N***

		Way 1			Way 0		
Set	**LRU**	**V**	**Tag**	**Data**	**V**	**Tag**	**Data**
000	*0*	*1*	*0x005*	*M[0x0140-0147]*	*1*	*0x001*	*M[0x0040-0047]*
001	*1*				*1*	*0x001*	*M[0x0048-004F]*
010							
011							
100	*0*	*1*	*0x3FF*	*M[0xFFE0-FFE7]*	*1*	*0x000*	*M[0x0020-0027]*
101	*1*				*1*	*0x000*	*M[0x0028-002F]*
110	*0*	*1*	*0x007*	*M[0x01F0-01F7]*	*1*	*0x0C7*	*M[0x31F0-31F7]*
111	*0*	*1*	*0x007*	*M[0x01F8-01FF]*	*1*	*0x3FF*	*M[0xFFF8-FFFF]*

Question b (Quiz #5 Sec1 Spring 2016) Cache modeling and tracing

Assume a 256-word word-addressable main memory and a four-block direct-mapped cache with two words per block. The cache is initially empty. For the word address reference stream given below, indicate which of the references are hits. Also, show the final contents of the cache. (The word addresses are in decimal.)

2, 10, 3, 26, 12, 4, 12, 10, 5, 12, 6, 10

The cache has 4 lines, or sets, each containing one block, of 2 words. It looks like:

Index	
00	
01	
10	
11	

From the problem statement, addresses are 8 bits, with no byte offset, a 1-bit block offset and 2 index bits, and 5 tag bits. Convert the decimal addresses to binary:

2 = 00000$\underline{01}$0 3 = 00000$\underline{01}$1 4 = 00000$\underline{10}$0 5 = 00000$\underline{10}$1
6 = 00000$\underline{11}$0 10 = 00001$\underline{01}$0 12 = 00001$\underline{10}$0 26 = 00011$\underline{01}$0

For each memory address, the index bits have been underlined, to make cache tracing easier. Tracing the 12 memory references give the following results, in the 2 table below:

Access:

access	*2-M*	*10-M*	*3-M*	*26-M*	*12-M*	*4-M*	*12-M*	*10-M*	*5-M*	*12-M*	*6-M*	*10-H*

Cache contents:

start 2 10 3 26 12 4 12 10 5 12 6 10

empty												
empty	~~*2,3*~~	~~*10,11*~~	~~*2,3*~~	*26,27*	*26,27*	*26,27*	~~*26,27*~~	*10,11*	*10,11*	*10,11*	*10,11*	*10,11*
empty					~~*12,13*~~	~~*4,5*~~	*12,13*	~~*12,13*~~	~~*4,5*~~	*12,13*	*12,13*	*12,13*
empty											*6,7*	*6,7*

Question 1 (Quiz #5 Sec2/3 Spring 2016) Virtual memory

Consider a virtual memory system that can address a total of 2^{50} bytes. You have unlimited hard drive space, but are limited to 2 GB of semiconductor (physical memory). Assume that virtual and physical pages are each 4 KB in size.

a) How many bits is the physical address?

Since 2 GB = 2^{31} bytes, the physical address is 31 bits

b) What is the maximum number of virtual pages in the system?

The virtual memory is 2^{50} bytes, max. Virtual page size is 4 KB = 2^{12} bytes.
Therefore there are $2^{50}/2^{12} = 2^{38}$ = 256 Giga (or ~ 256 billion) virtual pages, max

c) How many physical pages are in the system?

Using the same approach as in b), $2^{31} / 2^{12} = 2^{31-12} = 2^{19}$ = 512 Kilo physical pages

d) How many bits are the virtual and physical page numbers?

The virtual page number is 38 bits. The physical page number is 19 bits.

e) How many page table entries will the page table contain?

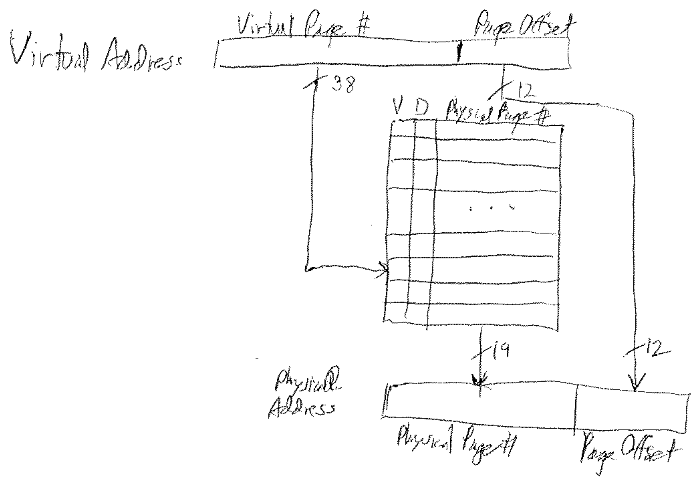
The page table contains one entry for each virtual page, so 2^{38} entries

f) Assume that, in addition to the physical page number, each page table entry also contains some status information in the form of a valid bit (V) and a dirty bit (D). How many bytes long is each page table entry? (Round up to an integer number of bytes.)

Physical page numbers are 19 bits, plus the V and D bits, equals 21 bits. So 3 bytes are needed for each page table entry.

g) Sketch the layout of the page table. What is the total size of the page table in bytes?

It will be 3 bytes/entry $\cdot\ 2^{38}$ entries = $3 \cdot 2^{38} = 1.5 \cdot 2^{39}$ bytes total size.
Nearly 2^{40} = 1 TeraByte in size.



Question 3 (Final Fall 2015) Cache modeling and tracing

Part 1 Using the address format given below and the additional fact that cache is 2-way set associative, answer the following:

Tag	Set	Bl. Off.	By. Off.
24 bits	12 bits	5 bits	3 bit

a) What is the data capacity of cache (in bytes)?

It has 2^{12} sets, and each set contains 2^1 blocks, and each block contains 2^{5+3} bytes, so the data capacity of cache is $2^{12} \cdot 2^1 \cdot 2^8 = 2^{21} = 2 \cdot 2^{20}$ = 2 MBytes

b) What is the size of memory (in bytes)?

Memory address is 24+12+5+3 = 44 bits, so its size is $2^{44} = 2^4 \cdot 2^{40}$ = 16 Terabytes

c) What is the word-size in this processor?

Each byte offset is 3 bits, so word size is 2^3 = 8 bytes = 64-bits

d) Where might this processor be used? What kind of application programs might run on it?

This processor is a 64-bit processor, with a big cache (2 MB) and very big main memory capacity (16 TB). It could be used as a server, perhaps as part of a cloud of powerful processors. It may be running very big programs, or running part of very big programs (along with other processors), where performance is important.

Part 2 (16 points) [Note: this part is independent from Part 1 above] For the cache shown on the next page, each block holds 16 words and each word is 32-bits. The cache implements FIFO (first in first out) replacement. Memory is byte addressable. Trace the address reference string and detemine for each address access where it will be located in cache. You should show the final contents of the cache, including the V bit, FIFO bits, Tag and Data contents of each location that contains a block. Assume that empty blocks are filled starting from the left-most way. The FIFO bits should indicate which block is first in (=00), 2nd in (=01), 3rd in (=10), and 4th in (=11). The memory reference string, in order from first reference to last, is 0x0088, 0x018A, 0x0208, 0x054E, 0x03CC, 0x018A, 0x00CE, 0x008C.

Tag Set Bloff Byoff

```
The references, in binary, are: 0000000  010  0010  00   # 0x0088
                                0000000  110  0010  10   # 0x018A
                                0000001  000  0010  00   # 0x0208
                                0000010  101  0011  10   # 0x054E
                                0000001  111  0011  00   # 0x03CC
                                0000000  110  0010  10   # 0x018A
                                0000000  011  0011  10   # 0x00CE
                                0000000  010  0011  00   # 0x008C
```

(Only the left-hand way of the cache contains data blocks! So to make it fit in the page, only 3 ways are shown)

Set	*V*	*FIFO*	*Tag*	*Data*	*V*	*FIFO*	*Tag*	*Data*	*V*	*FIFO*	*Tag*	*Data*
000	*1*	*00*	*0000001*	*M[0x0200 -0x023F]*								
001												
010	*1*	*00*	*0000000*	*M[0x0080 -0x00BF]*								
011	*1*	*00*	*0000000*	*M[0x00C0 -0x00FF]*								
100												
101	*1*	*00*	*0000010*	*M[0x0540 -0x057F]*								
110	*1*	*00*	*0000000*	*M[0x0180 -0x01BF]*								
111	*1*	*00*	*0000001*	*M[0x03C0 -0x03FF]*								

Question 2 (MT #3 Fall 2015) Cache modeling and tracing

Part 1 Using the address format given below, and the additional fact that cache is direct-mapped, answer the following:

Tag	Set	Bl. Off.	By. Off.
7 bits	8 bits	2 bits	1 bit

d) What is the data capacity of cache (in bytes)?

2^8 x 2^3 = 2^{11} = 2048 bytes = 2KB cache

e) What is the size of memory (in bytes)?

2^{18} = 256 KB memory

f) What is the word-size in this processor?

2 bytes/word , so word size is 16-bits

d) Where might this processor be used? What kind of application programs might run on it?

Such a 16-bit processor with small cache and small memory is likely to be embedded in a car or appliance, in a factory or transportation system, etc. It will run small control programs, process signals, do music/audio, etc

Part 2 [Note: this part is independent from Part 1 above] For the cache shown on the next page, each block holds 8 bytes of data. The cache implements LRU replacement. Memory is byte addressable. Trace the address reference string and detemine for each address access where it will be located in cache. You should show the final contents of the cache, including the V bit, LRU bits, Tag and Data contents of each location that contains a block. Assume that empty blocks are filled starting from the left-most way. The LRU bits should indicate which block is least recently used (=00), 2nd least recently used (=01), 2nd most recently used (=10), and most recently used (=11). The memory reference string, in order from first reference to last, is 0x0088, 0x018A, 0x0208, 0x054E, 0x03CC, 0x018A, 0x00CE, 0x008C.

Solution (Left half of cache)

Set	*V*	*LRU*	*Tag*	*Data*	*V*	*LRU*	*Tag*	*Data*
000								
001	*1*	*00*	*.....0001111*	*M[0x 03C8-03CF]*	*1*	*01*	*...0000110*	*M[0x 0188-018F]*
010								
011								
100								
101								
110								
111								

Way 0 **Way 1**

(Right half of cache)

Set	*V*	*LRU*	*Tag*	*Data*	*V*	*LRU*	*Tag*	*Data*
000								
001	*1*	*10*	*.....0000011*	*M[0x 00C8-00CF]*	*1*	*11*	*...0000010*	*M[0x 0088-008F]*
010								
011								
100								
101								
110								
111								

Way 2 **Way 3**

Question 4 (MT #2 Fall 2015) Cache modeling and tracing

Part 1 For a 16-bit processor with 1GB address space, connected to a 64KB 8-way set-associative cache with 4096 blocks in the cache, find the following:

g) the size of the data portion of each block (in words)

Block size = (64K bytes/cache)/(4K blocks/cache) = 16 bytes/block
Words are 16-bits = 2 bytes/word , so (16 bytes/block)/(2 bytes/word) = 8 words/block

h) the number of sets in the cache

Since the cache is 8-way set-associative, there are 8 blocks per set
# sets = (4096 blocks/cache)/(8 blocks/set) = 512 sets /cache

i) the number of tag bits in the Tag field of the memory address

1GB addresses means 2^{30} bytes addresses, so addresses are 30 bits

512 sets: 2^9 set addresses, so 9 index bits
8 words/block: 2^3 block offset addresses, so 3 block offset bits
2 bytes/word: 2^1 byte offset addresses, so 1 byte offset bit
# tag bits = 30 – (9 + 3 + 1) = 17 tag bits

Tag	*Index*	*Block offset*	*Byte offset*
17 bits	*9 bits*	*3 bits*	*1*

Part 2 For the cache shown on the next page, each block holds 16 bytes of data. Trace the address reference string and detemine for each address access if it is a Hit or a Miss in cache, giving the result in the table on the next page. For each Miss, identify the type (Cumpulsory, Conflict or Capacity). In addition, show the final contents of the cache. Use the left-hand way for first placement of a block into cache. The memory reference string is 0x0027, 0x0028, 0x0044, 0x004C, 0x01FF, 0x31F3, 0x31FE, 0xFFFF, 0xFFE0, 0x002F, 0x0144, 0x11FF, 0x01F0, 0x0021. [Note: Assume that memory is byte-addressable, and that cache uses LRU replacement.]

Hex addr	**Binary Addr**	**Hit or Miss**
0x0027	0000 0000 0010 0111	***Miss-cold start***
0x0028	0000 0000 0010 1000	***Hit***
0x0044	0000 0000 0100 0100	***Miss-cold start***
0x004C	0000 0000 0100 1100	***Hit***
0x01FF	0000 0001 1111 1111	***Miss-cold start***
0x31F3	0011 0001 1111 0011	***Miss-cold start***
0x31FE	0011 0001 1111 1110	***Hit***
0xFFFF	1111 1111 1111 1111	***Miss-cold start***
0xFFE0	1111 1111 1110 0000	***Miss-cold start***
0x002F	0000 0000 0010 1111	***Hit***
0x0144	0000 0001 0100 0100	***Miss-cold start***
0x11FF	0001 0001 1111 1111	***Miss-cold start***
0x01F0	0000 0001 1111 0000	***Miss-conflict***
0x0021	0000 0000 0010 0001	***Hit***

Block 0x01Fx was in set 1111, then was replaced by LRU, then re-accessed. So the 2nd access is a conflict miss. All other misses are compulsory (or cold-start) misses, caused the first time a block is accessed.

Cache

Set	*V*	*Tag*	*Data*	*V*	*Tag*	*Data*
0000						
0001						
0010	*1*	*00000000*	*M[0x0020-0x002F]*			
0011						
0100	*1*	*00000000*	*M[0x0040-0x004F]*	*1*	*00000001*	*M[0x0140-0x014F]*
0101						
0110						
0111						
1000						
1001						
1010						
1011						
1100						
1101						
1110	*1*	*11111111*	*M[0xFFE0-0xFFEF]*			
1111	*1*	*00000001*	*M[0x01F0-0x01FF]*	*1*	*00010001*	*M[0x11F0-0x11FF]*

Question 1 (Classwork Week #11 Fall 2015) Cache Modeling and Tracing

For this problem, an 8-word 2-way set associative cache is used, with block size of 1. The main memory has 16 words, and address references are given as word addresses. The cache uses LRU replacement policy. If both fields for an index are empty, the data should be stored in the left-most field.

a) Assuming that the cache is initially empty, for the reference string 1, 2, 3, 4, 5, 9, 1, 2, 3, 5, 1, 9, 10, 12, 2, determine which references are hits and which are misses.

Address	1	2	3	4	5	9	1	2	3	5	1	9	10	12	2
Hit/Miss	***M***	***M***	***M***	***M***	***M***	***M***	***M***	***H***	***H***	***M***	***H***	***M***	***M***	***M***	***H***

b) Give the final contents of all the Tag and Data fields, after the string of references. (The Tag field should be given in binary, and the Data field should be given in decimal. [Assume that memory location i contains the value i.]

Set		Tag	Data	Tag	Data
00	■	***01***	***4***	***11***	***12***
01	■	***10***	***9***	***00***	***1***
10	■	***00***	***2***	***10***	***10***
11	■	***00***	***3***		

Question 2 (Classwork Week #11 Fall 2015) Cache Modeling and Tracing

For this problem, a 16-word 2-way set associative cache is used, with block size of 2 words. The main memory has 128 words, and address references are given as word addresses. The cache uses LRU replacement policy. If both fields for an index are empty, the data should be stored in the left-most field.

a) Assuming that the cache is initially empty, for the reference string 1, 2, 3, 4, 5, 9, 1, 2, 3, 5, 1, 9, 10, 12, 2, determine if the reference is a hit or a miss. Put your answer in the table below (H = hit, M = miss)

Address	1	2	3	4	5	9	1	2	3	5	1	9	10	12	5	6
H/M																

Address	1	2	3	4	5	9	1	2	3	5	1	9	10	12	5	6
H/M	***M***	***M***	***H***	***M***	***H***	***M***	***H***	***H***	***H***	***H***	***H***	***H***	***M***	***M***	***H***	***M***

b) Give the final contents of all the Tag and Data fields, after the string of references. (The Tag field should be given in binary, and the Data field values should be given in decimal. [Assume that memory location *i* contains the value *i.)]*

Set		Tag	Data	Tag	Data
00	■	***0000***	***1,0***	***0001***	***9,8***
01	■	***0000***	***3.2***	***0001***	***11,10***
10	■	***0000***	***5,4***	***0001***	***13,12***
11	■	***0000***	***7,6***		

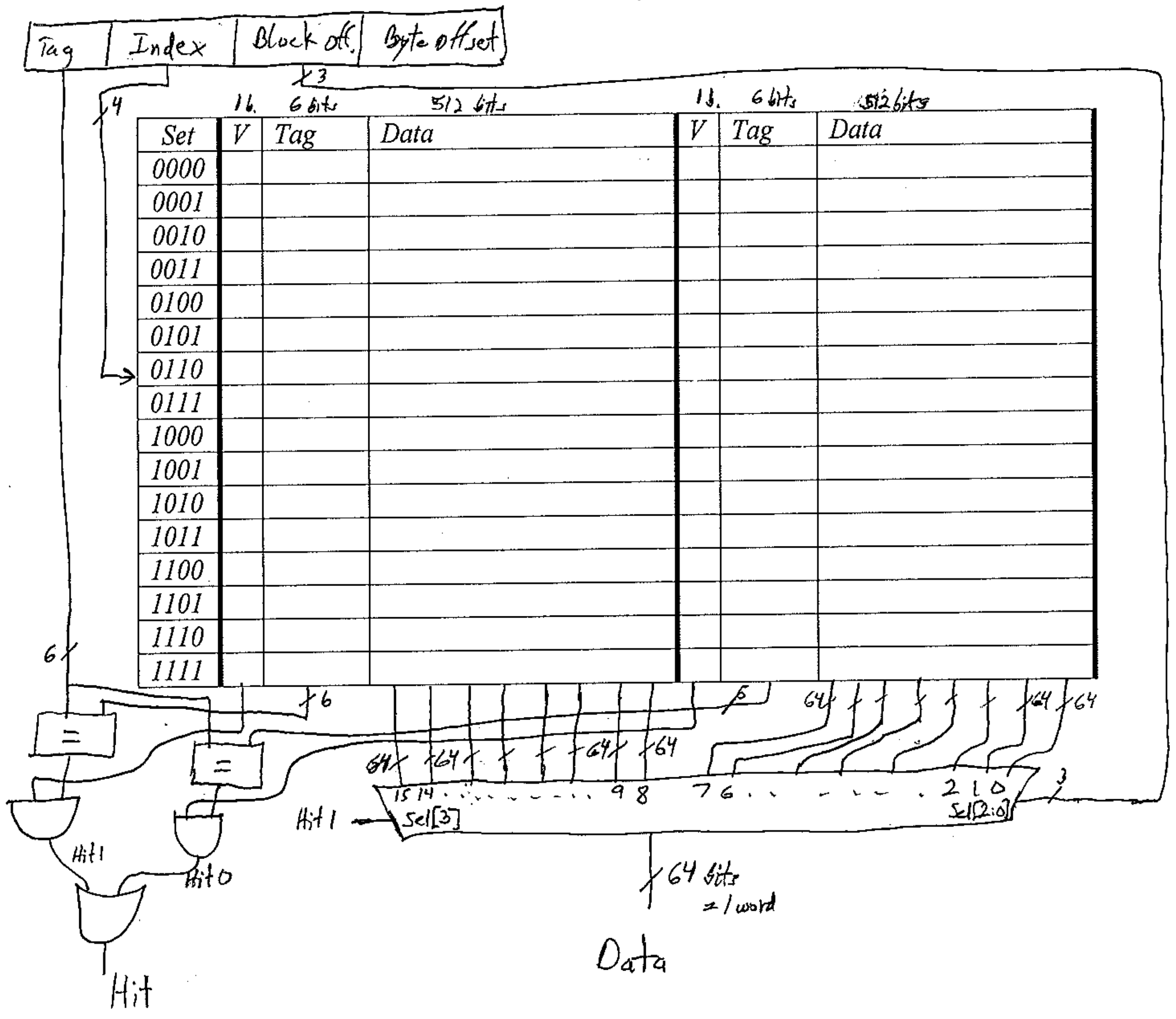
Question 1 (Quiz #5 Fall 2015) 2-way set associative cache: modeling and tracing

Consider a byte-addressed memory, of size 64K bytes. The 2-way set-associate cache size is 2K bytes. For this system, word size is 8 bytes, and block size is 8 words.

a) Draw the address format, including all fields, with names and widths.
8 bytes/word · 8 words/block = 64 bytes/block [byte offset=3 bits, block offset=3 bits]
(2048 bytes/cache) / (64 bytes/block) = 32 blocks/cache
2 blocks/set · N sets/cache = 32 blocks/cache → N = 16 sets [set index=4 bits]

6	4	3	3
Tag	***Index***	***Block offset***	***Byte offset***

b) Draw the cache structure for this system, in the general form of the figures shown in the book. Be sure to include valid and tag bits. Label all fields, with names and widths. Make clear the widths of all data and address busses. Show not just the memory part of the cache, but the logic portions as well.



c) Trace address string and detemine for each address access if it is a Hit or a Miss in cache, giving the result in the table below. In additions, show the final contents of the cache, in a cache drawing like that of part b. The memory reference string is 0x23, 0x28, 0x44, 0x45, 0x1FF, 0x31F3, 0x31FE, 0xFFFF, 0xFFE0, 0x2F, 0x144, 0x11FF, 0x12F0, 0x2F. [Note: Assume LRU replacement.]

Hex addr	Binary Addr	Hit or Miss
0x23	0000 0000 0010 0011	***Miss-cold start***
0x28	0000 0000 0010 1000	***Hit***
0x44	0000 0000 0100 0100	***Miss-cold start***
0x45	0000 0000 0100 0101	***Hit***
0x1FF	0000 0001 1111 1111	***Miss-cold start***
0x31F3	0011 0001 1111 0011	***Miss-cold start***
0x31FE	0011 0001 1111 1110	***Hit***
0xFFFF	1111 1111 1111 1111	***Miss-cold start***
0xFFE0	1111 1111 1110 0000	***Hit***
0x2F	0000 0000 0010 1111	***Hit***
0x144	0000 0001 0100 0100	***Miss-cold start***
0x11FF	0001 0001 1111 1111	***Miss-cold start***
0x12F0	0001 0010 1111 0000	***Miss-cold start***
0x2F	0000 0000 0010 1111	***Hit***

Set	*V*	*Tag*	*Data*	*V*	*Tag*	*Data*
0000	*1*	*000000*	*M[0x00-0x3F]*			
0001	*1*	*000000*	*M[0x40-0x7F]*			
0010						
0011						
0100						
0101	*1*	*000000*	*M[0x140-0x17F]*			
0110						
0111	*1*	*000100*	*M[0x11C0-0x11FF]**LRU***	*1*	*001100*	*M[0x31C0-0x31FF]*
1000						
1001						
1010						
1011	*1*	*000100*	*M[0x12C0-0x12FF]*			
1100						
1101						
1110						
1111	*1*	*111111*	*M[0xFFC0-0xFFFF]*			

d) If the above cache were fully associate, how many comparators would be needed to build it? How many of the above hits would become misses? How many of the above misses would become hits?

The above cache contains 32 blocks, so it would need 32 comparators, if fully-associative. Of the 8 misses, all were cold-start (i.e. compulsory) misses, that would happen no matter what the associativity is. So none of the misses would become hits with full associativity. And none of the hits would become misses in a fully-associative cache.

Question 2 (Quiz #5 Fall 2015) Understanding cache organization and structure

For a 32-bit processor with 4GB address space, connected to a 16KB 4-way set-associative cache with 2048 blocks in the cache, with each block containing Valid, and 2 LRU bits, plus Tag and Data, find the following:

a) the size of each data block (in words)

Block size = (16K bytes/cache)/(2K blocks/cache) = 8 bytes/block
Since words are 32-bits = 4 bytes/word, (8 bytes/block)/(4 bytes/word) = 2 words/block

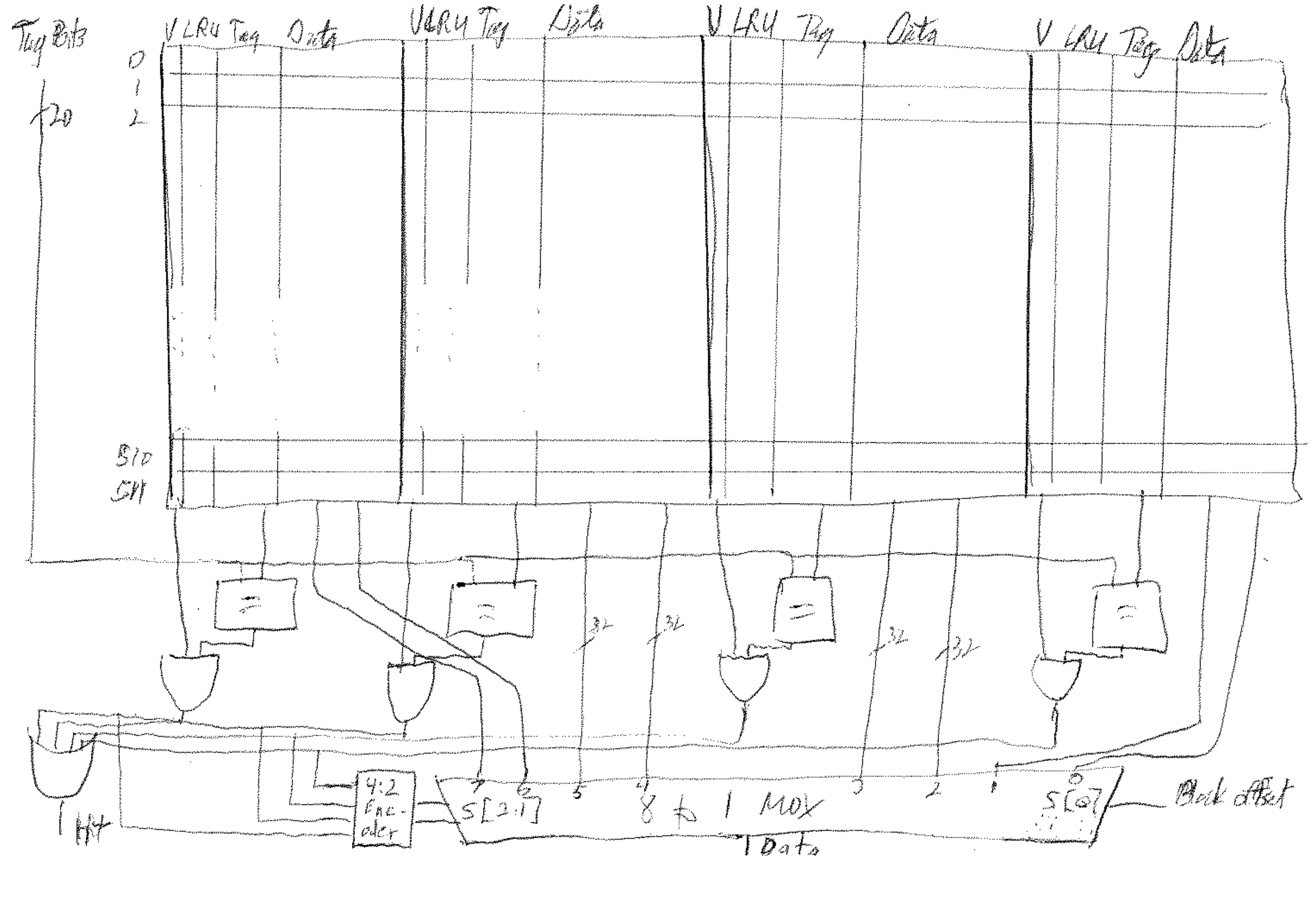
b) the number of sets in the cache

Since the cache is 4-way set-associative, there are 4 blocks per set
# bytes per set = 4 blocks/set · 8 bytes/block = 32 bytes/set
# sets = (16K bytes/cache)/(32 bytes/set) = 0.5K sets/cache = 512 sets in the cache

c) the number of tag bits in the Tag field of the memory address

4GB addresses means 2^{32} bytes addresses, so addresses are 32 bits
512 sets: 2^9 set addresses, so 9 index bits
2 words/block: 2^1 block offset addresses, so 1 block offset bit
4 bytes/word: 2^2 byte offset addresses, so 2 byte offset bits
# tag bits = 32 – (9 + 1 + 2) = 20 tag bits

d) Draw the block diagram of the cache, including the MUXs, gates, and comparators. You don't need to show the hardware associated with the LRU bits.



Question 4 (MT #2 Spring 2015): Cache Modeling and Tracing

a) Consider a 4 GB byte-addressable main memory with a four-way set-associative cache of 2 MB and 32 bytes per block.
   1. How many total blocks are there in cache? Draw the structure of the cache to explain it.
   2. Show how the main memory address is partitioned into fields for the cache access and give the bit lengths of those fields.
   3. How many tag comparators are there in the cache?

1. *2MB / (32 bytes/block) = 2^21 / 2^5 blocks = 2^16 blocks*
   Each set contains 4 blocks (since it is 4-way set-associative), so there are 2^14 sets

2. *offset bits: log2(32) = 5 [2 are for byte offset (since 4 bytes/word), and 3 for block offset]*
   index bits: log2(2^16 / 4) = log2(2^14) = 14
   tag bits: log2(4G) – (14 + 5) = 32 - 19 = 13

13 bits	*14 bits*	*3 bits*	*2 bits*
Tag	*Index*	*Block Offset*	*Byte Offset*

3. *There is one per way, and there are 4 ways (or banks) in the cache.*

b) Assume a 256-word word-addressable main memory and a four-block direct-mapped cache with two words per block. The cache is initially empty. For the word address reference stream given below, indicate which of the references are hits. Also, show the final contents of the cache. (The word addresses are in decimal.)

2, 10, 3, 26, 12, 4, 12, 10, 5, 12, 6, 10

From the problem statement, addresses are 8 bits, with no byte offset, a 1-bit block offset and 2 index bits, and 5 tag bits. Convert the decimal addresses to binary:

2 = 00000$\underline{01}$0   3 = 00000$\underline{01}$1   4 = 00000$\underline{10}$0   5 = 00000$\underline{10}$1
6 = 00000$\underline{11}$0   10 = 00001$\underline{01}$0   12 = 00001$\underline{10}$0   26= 00011$\underline{01}$0

For each memory address, the index bits have been underlined, to make cache tracing easier.

Index	
00	
01	
10	
11	

Access

access	*2-M*	*10-M*	*3-M*	*26-M*	*12-M*	*4-M*	*12-M*	*10-M*	*5-M*	*12-M*	*6-M*	*10-H*

Contents

empty												
empty	~~*2,3*~~	~~*10,11*~~	~~*2,3*~~	*26,27*	*26,27*	*26,27*	~~*26,27*~~	*10,11*	*10,11*	*10,11*	*10,11*	*10,11*
empty					~~*12,13*~~	~~*4,5*~~	*12,13*	~~*12,13*~~	~~*4,5*~~	*12,13*	*12,13*	*12,13*
empty											*6,7*	*6,7*

Question 3a (Final Exam Spring 2015): Cache Modeling and Performance

a) A computer has a cache of 64KB with a block size of 64 Bytes. Assume that the main memory is a byte-addressable 16MB memory.

i) If the cache is directed mapped, what is the address format used to access this cache? Explain.
ii) If the cache is 2-way associative, what is the address format? Explain.
iii) How would the results to parts i and ii change if block size is 32 Bytes?

Answer:

i) *16MB = 2^24, 24 bits for memory addresses.*
64bytes = 2^6, 6 bits for block addresses (including byte offset)
64KB/64 bytes = 1K=2^10, 10 bits for index

8 bits Tag	*10 bits Index*	*4 bits Block Offset*	*2 bits Byte Offset*

ii) *2-way set associative will reduce the index bit by 1 and increase the tag by 1, therefore*

9 bits Tag	*9 bits Index*	*4 bits Block Offset*	*2 bits Byte Offset*

iii) *32 bytes = 2^5 bytes. Block size is smaller, therefore there will be more blocks in the cache increasing the index bits for both cases by 1.*

8 bits Tag	*11 bits Index*	*3 bits Block Offset*	*2 bits Byte Offset*

9 bits Tag	*10 bits Index*	*3 bits Block Offset*	*2 bits Byte Offset*

Question 3 (Final Retake Spring 2015): Cache Modeling

c) A computer has a 4-way set associative 1KB cache with a block size of 64 bits. Assume that the main memory is a byte-addressable 64KB memory. Explain the address format used to access this cache in a MIPS-like 32-bit processor.

Answer:

64KB = 2^16, addresses are 16 bits
Block size = 64/8 = 8 bytes = 2^3. So Offset is 3 bits
Byte Offset 2 bits, since there are 4 bytes per word (32-bit word size)
Block Offset 1 bit, since there are 2 words in each 64-bit block
1KB/4 = 2^8 bytes for each way, 2^8/2^3 = 2^5 blocks in each way, therefore Index/Set is 5 bits
Address has 16 bits, 3 are Offset and 5 are Index/Set, so Tag is 8 bits

Tag	*Set*	*Off.*
8 bits	*5 bits*	*3 bits*

d) Consider a direct mapped 1KB cache with a block size of 64 bits, where main memory is a byte-addressable 64KB memory. Consider the following read pattern (addresses) repeats for 100 times assuming that cache replacement policy is LRU:

7. 0x0000
8. 0x0100
9. 0x0200
10. 0x0300
11. 0x1000

What is the hit rate? Explain in detail.

Answer:

Cache is similar to part a, but direct mapped.

64KB = 2^16, addresses are 16 bits
Block size = 64/8 = 8 bytes = 2^3
1KB = 2^10 bytes, 2^10/2^3 = 2^7 blocks in the cache, therefore

Tag: 6 bits
Index/Set: 7 bits
Offset: 3 bits

Expanding the memory references into binary, and grouping by Tag, Index, and Offset:

7. 0x0000 => 000000 0000000 000
8. 0x0100 => 000000 0100000 000
9. 0x0200 => 000000 1000000 000
10. 0x0300 => 000000 1100000 000
11. 0x1000 => 000100 0000000 000

*From this we see that references 1 and 5 are in the same index/set. They will be in conflict and every reference will be a miss. However, references 2, 3, 4 are in different sets. So they will only encounter a compulsory miss the first time they are requested. Subsequent accesses will all be cache hits. So the hit rate is (99*3)/100*5= 297/500 = 59.4%*

Question 1 (Quiz #6 Spring 2015) Virtual Memory

Consider a virtual memory system that can address a total of 2^{50} bytes. You have unlimited hard drive space, but are limited to 2 GB of semiconductor (physical memory). Assume that virtual and physical pages are each 4 KB in size.

a) How many bits is the physical address?
Since 2 GB = 2^{31} bytes, the physical address is 31 bits

b) What is the maximum number of virtual pages in the system?
The virtual memory is 2^{50} bytes, max. Virtual page size is 4 KB = 2^{12} bytes.
Therefore there are $2^{50-12} = 2^{38}$ = 256 Giga (or ~ 256 billion) virtual pages, max

c) How many physical pages are in the system?

Using the same approach as in b), $2^{31} / 2^{12} = 2^{31-12} = 2^{19} = 512$ Kilo physical pages

d) How many bits are the virtual and physical page numbers?

The virtual page number is 38 bits. The physical page number is 19 bits.

e) How many page table entries will the page table contain?

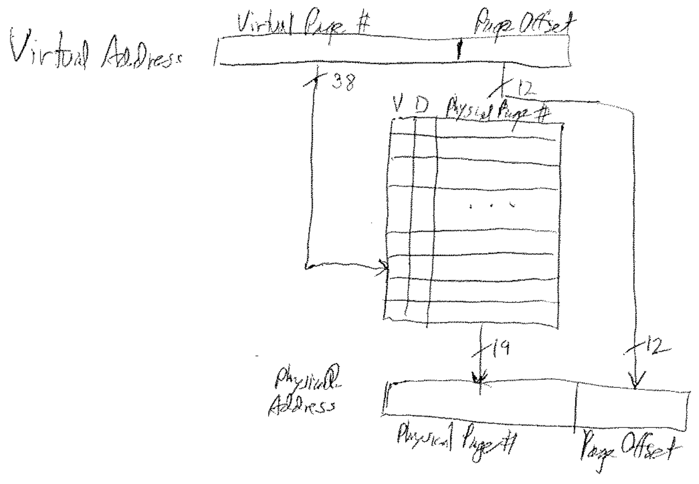
The page table contains one entry for each virtual page, so 2^{38} entries

f) Assume that, in addition to the shysical page number, each page table entry also contains some status information in the form of a valid bit (V) and a dirty bit (D). How many bytes long is each page table entry? (Round up to an integer number of bytes.)

Physical page numbers are 19 bits, and with the V and D bits, 21 bits, or 3 bytes, are needed for each page table entry.

g) Sketch the layout of the page table. What is the total size of the page table in bytes?

It will be 3 bytes/entry $\cdot$ 2^{38} entries = $3 \cdot 2^{38} = 1.5 \cdot 2^{39}$ bytes total size.
Nearly 2^{40} = 1 TeraByte in size.



Question 4 (MT #2 Spring 2014): Cache Memory: Modeling and Tracing

In this question, memory is word-addressable (not byte-addressable, like MIPS). Memory size is 256 MB. Cache size is 256 Bytes. Word size is 64 bits. Block size is 4 words. Cache is direct mapped.

a. For this memory system, give the memory address format. Identify each field and give its width in bits.

Since 1 byte = 8 bits, each word of 64 bits is 64/8 = 8 bytes
Since 256 MB/(8 bytes/word) is 32 Mwords = 2^{25}, the word address will be 25 bits. There is no byte offset field in the address, since bytes are not addressable. (Only words are addressable in word-addressable memory.)

Since each block is 4 words, the block offset field in the address will therefore be 2 bits, to address any of the 4 words in the block.

Block size is 4 words x 8 bytes/word = 32 bytes.
So the cache contains 256 / 32 = 8 blocks. It is direct mapped, which means each set contains 1 block. So there are 8 sets in the cache. The set field in the address will therefore be 3 bits, to address any of the 8 sets.

20 bits	*3 bits*	*2 bits*
Tag	*Set #*	*Block offset*

b. Draw the cache block diagram for this system. Be sure to include any hardware needed for hit determination.

Set #	*V*	*Tag*	*Data3*	*Data2*	*Data1*	*Data0*
000						
001						
010						
011						
100						
101						
110						
111						

The additional hardware needed is:
-- four copies (one per way) of the equality detector for Tag matching, followed by the AND gate to check Valid bit, producing Hit0, Hit1, Hit2 and Hit3 signals. The OR of these is the overall Hit signal.
-- a 16-to-1 MUX, to select which of the 16 data words in the set should be sent to the processor when a Hit occurs. 2 of the select bits come from Block Offset, and the other 2 are determined from the 4 HitX bits, which tell which way had the hit.

c. Consider the following memory references given in the left column.

Referenced Memory Location hex: binary (only LSBs are given)	Hit/Miss (miss type)
1B: 0001 1011	***Miss (CS)***
58: 0101 1000	***Miss (CS)***
5C: 0101 1100	***Miss (CS)***
D7: 1101 0111	***Miss (CS)***
58: 0101 1000	***Hit***
D7: 1101 0111	***Hit***
58: 0101 1000	***Hit***
D7: 1101 0111	***Hit***
D8: 1101 1000	***Miss (CS)***
1C: 0001 1100	***Miss (CS)***

In the left column, underline the index (set) field in all of the cases.

In the right column, indicate if the referenced memory location generates a hit or miss. If it is a miss, state what type of miss: Cold Start (CS), Conflict (CO), or Capacity (CA)

In the cache block diagram you drew in part b), show the final contents of the cache, with the value of the Valid bits, the Tag bits, and the range of Data given for each non-empty cache block. In the Data portion of each block, indicate the current contents of the cache memory in terms of address ranges (i.e., from which memory location to which memory location). For example M[7-0] means the memory values from address 0 up to address 7. Express the addresses in decimal.

Set #	*V*	*Tag*	*Data3*	*Data2*	*Data1*	*Data0*
000						
001						
010						
011						
100						
101	*1*	*...110*	*M[215]*	*M[214]*	*M[213]*	*M[212]*
110	*1*	*...110*	*M[219]*	*M[218]*	*M[217]*	*M[216]*
111	*1*	*...000*	*M[31]*	*M[30]*	*M[29]*	*M[28]*

Question 4 (Final Exam Spring 2014): Cache Memory: Modeling and Tracing

In this question, memory is byte-addressable. Memory size is 1 GB. Cache size is 4096 Bytes. Word size is 16 bits. Block size is 8 words. Cache is 4-way set associative.

a) For this memory system, give the memory address format. Identify each field and give its width in bits.

Word size is 16-bits = 2 bytes. So the byte offset is 1 bit.
Block size is 8 words, so the block offset is 3 bits.
The block size is 16 bytes, so there are 4096/16 = 256 blocks in the cache.
There are 4 blocks per set, since it is 4-way set associative.
Since there are 256/4 = 64 sets in the cache, the index (or set#) is 6 bits
The whole memory is 1 GByte (2^{30}) so it takes 30 bits to address any byte.

The address format is therefore:

20 bits	*6 bits*	*3 bits*	*1 bit*
Tag	*Set #*	*Block offset*	*Byte o.s.*

b) Draw the cache block diagram for this system. Be sure to include any hardware needed for hit determination.

The cache drawing will have:
--4 ways in 4 vertical strips in parallel, each a direct mapped cache
--each way contains 1 block with Valid bit, Tag bits and Data field.

--the labels for V, Tag, and Data are given for each way
--there are 64 sets in the cache, numbered from 0 up to 63
--each way has its own hit determination hardware, consisting of a 20-bit equality detector (one input is the 20-bit Tag field of the address, the other the Tag from the selected set) and an AND gate, which takes the V bit from the selected set and the output of the = HW
--the 4 hit signals are ORed together to form the overall Hit determination of the cache

c) Consider the following string of byte-addressable memory references, applied to a 512 Byte two-way set-associative cache using LRU replacement policy, where words are 32-bits and block size is 4 words.

Ref #	Referenced Memory Location hex: binary (only LSBs are given)	Hit/Miss (miss type)
1	0DB: 0000 <u>1101</u> 1011	*Miss (CS)*
2	1D8: 0001 <u>1101</u> 1000	*Miss (CS)*
3	25C: 0010 <u>0101</u> 1100	*Miss (CS)*
4	0D2: 0000 <u>1101</u> 0010	*Hit*
5	2D8: 0010 <u>1101</u> 1000	*Miss (CS)*
6	250: 0010 <u>0101</u> 0000	*Hit*
7	1DF: 0001 <u>1101</u> 1111	*Miss (CO)*
8	0D5: 0000 <u>1101</u> 0101	*Miss (CO)*

In the left column, underline the bits of the index (set) field in all of the cases.

With 4 bytes per word, the address format will have a 2-bit Byte offset field
With 4 words per block, there will be a 2-bit Block offset field.
With 16 bytes per block, there are 512/16 = 32 blocks in the cache. But since it is 2-way set associative, this means there are 16 sets, each containing 2 blocks.
So the <u>middle 4 bits</u> are the set # (or index) bits, the upper 4 are Tag, and the lower 4 are for block and byte offsets.

In the right column, indicate if the referenced memory location generates a hit or miss. If it is a miss, state what type of miss: Cold Start (CS), Conflict (CO), or Capacity (CA)

Note: <u>each wrong answer takes away the value of a correct answer!</u>

When the Tag bits or the Set# bits are different, the reference is to a different block. When they are the same, and only the last 4 bits are different, then the reference is to a different word or byte in the same block.

The first 3 references are all to different blocks not yet in cache (0D, 1D, 25), so they are Cold Start misses, since they are the first reference to these blocks. These 3 blocks are brought in. The 4th reference is to 0D, which is now in the cache, so it is a hit.

The 5th reference (to block 2D) is another Cold Start miss, and when it is brought in, set D is full, so one of the current blocks (0D or 1D) must be evicted. Using LRU replacement policy, 1D is selected to be evicted, it is replaced with block 2D.

The 6th reference is to 25, which is now in the cache, so it is a hit.

The 7th reference is to 1D, which was recently evicted. This is a Conflict miss. Block 1D is brought in, but since set D is full, one of 0D or 2D must be evicted. Using LRU replacement policy, 0D is selected to be evicted, it is replaced with block 1D.

The 8th reference is to 0D, which was just evicted. This is another Conflict miss. Block 0D is brought in, but since set D is full, one of 1D or 2D must be evicted. Using LRU replacement policy, 2D is selected to be evicted, it is replaced with block 0D

Question 4 (Final Retake Spring 2014): Cache Memory and Virtual Memory

a) In this question, memory is byte-addressable. Word size is 32-bits. Block size is 4-words. Replacement policy is LRU. The cache organization is 4-way set-associative, with 64 sets.

For the cache described above, trace the following string of memory references, and determine if each one is a Hit or a Miss, and if it is a miss, determine what type of miss: Cold Start (CS), Conflict (CO), or Capacity (CA): 00FFA, 053FC, 197F4, 00FF6, 063FC, FFFF8, FF0F4, 053F5, 063FB, 197FC

Referenced Memory Location hex: binary ()	Hit/Miss - miss type
00FFA: 0000 0000 11$\underline{\text{11 1111}}$ 1010	*Miss-CS*
053FC: 0000 0101 00$\underline{\text{11 1111}}$ 1100	*Miss-CS*
197F4: 0001 1001 01$\underline{\text{11 1111}}$ 0100	*Miss-CS*
00FF6: 0000 0000 11$\underline{\text{11 1111}}$ 0110	*Hit*
063FC: 0000 0110 00$\underline{\text{11 1111}}$ 1100	*Miss-CS*
FFFF8: 1111 1111 11$\underline{\text{11 1111}}$ 1000	*Miss-CS*
FF0F4: 1111 1111 00$\underline{\text{00 1111}}$ 0100	*Miss-CS*
053F5: 0000 0101 00$\underline{\text{11 1111}}$ 0101	*Miss-CO*
063FB: 0000 0110 00$\underline{\text{11 1111}}$ 1011	*Hit*
197FC: 0001 1001 01$\underline{\text{11 1111}}$ 1100	*Miss-CO*

In the left column, underline the index (set) field in all of the cases. In the right column, indicate if the referenced memory location generates a hit or miss. If it is a miss, state what type of miss: Cold Start (CS), Conflict (CO), or Capacity (CA) Note: each wrong answer takes away the value of a correct answer!

b) A virtual memory system can address a total of 4 GB of memory. There are 8 MB of DRAM, and a hard drive much larger than virtual memory. Assume that virtual and physical pages are both 8KB in size.

i) How many bits are in the physical address? In the virtual address?
ii) How many bits are the physical page numbers? The virtual page numbers?
iii) How many physical pages are in the system? How many virtual pages?
iv) How many page table entries will the page table contain?
v) Now consider using a TLB to speed up the virtual memory system. For the values given in the table below, what is the average memory access time of the virtual memory system before and after adding the TLB? Assume that the page table is always resident in physical memory and that it is not cached.

Memory Unit	Access Time (cycles)	Miss Rate
TLB	1	0.05%
Cache	1	2%
Main memory	100	0.0003%
Hard drive	1,000,000	0%

i) Physical 23 Virtual 32
ii) Physical 10 Virtual 19
iii) Physical 1024 Virtual 524288 (512K)
iv) 524288
v) AMAT = Virt-to-Phys translation time + MemAccessTime
Before TLB: [100] + [1+ 0.02(100 + 0.000003(1000000))] = 100 + 1 + 2.06 = 103.06
After TLB: [1 + 0.0005(100)] +[3.06] = 4.11 cycles

Question 4 (Final Exam Fall 2013): Memory

a) Fill in the empty cells of the following table. Assume that main memory size is 4GB, a word is 32 bits. **Index Size**: # of bits needed to express the set number in an address, **Block Offset**: # of bits needed to indicate the word offset in a block, **Byte Offset**: # of bits needed to indicate the byte offset in a word. **Block Replacement Policy Needed**: Indicate if a block replacement policy such as FIFO, LRU etc. is needed (yes) or not (no).

#	**Cache Size KB**	**N-way cache**	**Block size (# of words)**	**No. of Sets**	**Tag Size in bits**	**Index Size (Set No.) in bits**	**Block Offset Size in bits**	**Byte Offset Size in bits**	**Block Replacement Policy Needed (Yes/No)**
1	128	1	2	*2^{14}*	*15*	*14*	*1*	*2*	*No*
2	512	4	8	*2^{12}*	*15*	*12*	*3*	*2*	*Yes*
3	1024	Full	8	*1*	*27*	*0*	*3*	*2*	*Yes*

#1: Number of sets= (2^7 x 2^{10}) / (1 block/set x 2 words/block x 4 bytes/word) = $2^{17}/2^3$= 2^{14}
#2: Number of sets= (2^9 x 2^{10}) / (4 blocks/set x 8 words/block x 4 bytes/word) = $2^{19}/2^7$= 2^{12}
Memory size is 4GB= 4 x 2^{30} bytes
Tag size= 32 - (index size in bits + block offset size in bits + byte offset size in bits)

b) For # 1, explain your decision for the block replacement policy (i.e., why yes/no)?

In a 1-way cache, it is direct-mapped. There is only one choice to replace a block and therefore no replacement policy is needed.

c) Now consider a fully- associative cache memory that has room for three blocks. In this environment references to various main memory blocks are as follows (the blocks referenced are shown by letters).

A, B, A, C, A, D, D, E, C, A

Show the contents of the cache memory during each memory reference and indicate if there is a hit (H) or miss (M) and calculate the final miss rate. Do this for FIFO and LRU.

For showing the contents of the cache memory and if there is a hit or miss for each reference you are given tables below for FIFO and LRU.

Using FIFO: *top-most item is the newest one and bottom-most item is the oldest one, to kick out from the cache.*

	A	B	A	C	A	D	D	E	C	A
Cache Contents ==>	*A*	*B* *A*	*B* *A*	*C* *B* *A*	*C* *B* *A*	*D* *C* *B*	*D* *A* *C*	*E* *D* *A*	*C* *E* *D*	*A* *C* *E*
H/M==>	*M*	*M*	*H*	*M*	*H*	*M*	*H*	*M*	*M*	*M*

Final miss rate: *7/ 10= 0.7*

Using LRU: top-most item is the most recently used one, and the bottom-most one is least recently used, to kick out from the cache.

	A	B	A	C	A	D	D	E	C	A
Cache Contents ==>	*A*	*B* *A*	*A* *B*	*C* *A* *B*	*A* *C* *B*	*D* *A* *C*	*D* *A* *C*	*E* *D* *A*	*C* *E* *D*	*A* *C* *E*
H/M==>	*M*	*M*	*H*	*M*	*H*	*M*	*H*	*M*	*M*	*M*

Final miss rate: *7/ 10= 0.7*

Question 4 (Final Retake Fall 2013): Cache Memory: Modeling and Tracing

In a memory system like MIPS (word size is 32 bits, and using 32 bits for addressing memory) assume a direct-mapped cache memory of size 128 bytes. Block size is given as 4-words. A word is 32 bits.

a. Give the memory address format in terms of its subfields, giving the size of each subfield (tag, etc.).

Block size: 4 words ==> 4 x 4= 16 bytes
No. of blocks in cache memory 128/16= 8
Direct-mapped means 1 block per set ==> 8 sets ==> 3 bits for index field
4 words/block ==> 2 bits for block offset
4 bytes/word ==> 2 bits for byte offset
tag size= 32 - (3 + 2 + 2)= 25 bits

```
          tag              index       block       byte
                                       offset      offset
<----------------------><-----------><----------><----------->
           25                3          2           2
```

b. Assuming a valid bit stored with each cache block, how many total bits of storage are needed to implement the cache memory?

8 x (1 + 25 + 4 x 32)= 1232 bits

c. Consider the following memory references given in the table below, left column.

In the left column, underline the index field in all of the cases.

In the middle column, indicate the current contents of the cache memory in terms of address range. Indicate the current contents of the entire cache memory in terms of the address ranges of the blocks that it contains. For example M[0-7] means from address 0 up to address 7. Express the address ranges in decimal.

In the right column, indicate if the referenced memory location generates a hit or miss.

Referenced Memory Location hex: binary	Contents of Cache Memory *Index: M[from, to] See * below to understand how it works*	Hit/Miss
1B: 0001 1011	*001: M[16-31],*	*M*
58: 0101 1000	*001: M[16-31], 101: M[80-95]*	*M*
5C: 0101 1100	*001: M[16-31], 101: M[80-95]*	*H*
D7: 1101 0111	*001: M[16-31], 101: M[208-223]*	*M*
58: 0101 1000	*001: M[16-31], 101: M[80-95]*	*M*
D7: 1101 0111	*001: M[16-31], 101: M[208-223]*	*M*
58: 0101 1000	*001: M[16-31], 101: M[80-95]*	*M*
D7: 1101 0111	*001: M[16-31], 101: M[208-223]*	*M*
D8: 1101 1000	*001: M[16-31], 101: M[208-223]*	*H*
1C: 0001 1100	*001: M[16-31], 101: M[208-223]*	*H*
1B: 0001 1011	*001: M[16-31], 101: M[208-223]*	*H*
DC: 1101 1100	*001: M[16-31], 101: M[208-223]*	*H*
59: 0101 1001	*001: M[16-31], 101: M[80-95]*	*M*
D4: 1101 0100	*001: M[16-31], 101: M[208-223]*	*M*
1C: 0001 1100	*001: M[16-31], 101: M[208-223]*	*H*

(*) hex beginning location: memory range in decimal16 bytes M[from-to] ***00**: M[0-15],* ***10**: M[16-31],* ***20**: M[32-47],* ***30**: M[48-63],* ***40**: M[64-79],* ***50**: M[80-95],* ***60**: M[96-111],* ***70**: M[112-127],* ***80**: M[128-143],* ***90**: M[144-159],* ***A0**: M[160-175],* ***B0**: M[176-191],* ***C0**: M[192-207],* ***D0**: M[208-223],* ***E0**: M[224-239],* ***F0**: M[240-255]*

Question 3 (Final Exam Spring 2013): Cache tracing

In this problem, the cache will be used in the memory system of a 32-bit processor like MIPS. Its block size is 8 words, and its overall size is 2K Bytes. It is 2-way set associative, as shown below. Replacement policy is LRU. All cache lines are not shown, only the first few and last few lines. The middle lines of this cache are represented with . . . .

Line #	V	Tag	Data	V	Tag	Data
0						

1	*1*	*...0001*	*M[1087-1056]*	*1*	*...0010*	*M[2111-2080]*
2						
. . . .	.	. . . . . .	. . . .	.	. . . . . .	. . . .
. . . .	.	. . . . . .	. . . .	.	. . . . . .	. . . .
29	*1*	*...0001*	*M[1983-1952]*	*1*	*...0000*	*M[959-928]*
30						
31	*1*	*...0011*	*M[4095-4064]*			

a) Determine the memory address format, and draw a diagram indicating all the fields in the format, and the width of each field.

The number of byte offset bits must be 2, since there are 4 bytes in a 32-bit data word. The number of block offset bits must be 3, since there are 8 words in a block. Therefore there are 32 bytes in each block in cache. Since there are 2 KB total data in cache, there must be (2K bytes/cache)/(32 bytes/block) = 64 blocks/cache. For this cache, with 2 blocks per line, there must be 32 lines in the cache. This requires 5 index bits. All the remaining bits are tag bits. With no other information given about the size of memory, or the size of memory addresses, we can use the fact that the memory system is "like MIPS" to infer a 32-bit addrsess. This means tag field will be 22 bits.

Tag: 22 bits	*Index: 5 bits*	*Block. Off.: 3 bits*	*Byte Off: 2 bits*

b) Number the lines in the cache above, under "Line #".

Since there are 32 lines in the cache, the addresses will be from 0 to 31

c) Trace the following string of memory address references, and show the final contents of the cache in the diagram above.

For tracing the memory references, it is necessary to find the index bits. Those five bits are the address of the cache line that will be accessed. The 22 tag bits can be abbreviated to the last few least significant bits, since all leading bits are 0s. Since block size is 32 bytes, the data field in cache will contain values like M[X+31 to X], where X is the address of the lowest byte in the block, the byte whose 5 offset bits are 00000.

For tracing purposes, the table below contains the hits and misses, and the block that is replaced if the set is full (i.e if both locations in the line are full), according to LRU policy.

Ref #	Address$_{10}$	Address$_2$	*Hit?*
1	2999	….0010 11101 10111	*Miss: cold-start*
2	1073	….0001 00001 10001	*Miss: cold-start*
3	2082	….0010 00001 00010	*Miss: cold-start*
4	1952	….0001 11101 00000	*Miss: cold-start*
5	62	….0000 00001 11110	*Miss: cold-start*
6	2101	….0010 00001 10101	*Hit*
7	2983	….0010 11101 00111	*Hit*
8	4070	….0011 11111 00110	*Miss: cold-start*
9	942	….0000 11101 01110	*Miss: cold-start,1952 goes out*
10	1976	….0001 11101 11000	*Miss: conflict, 2983 goes out*
11	1076	….0001 00001 10100	*Miss: conflict, 62 goes out*

Question 4 (Final Retake Spring 2013): Cache modeling and cache tracing

In this problem part a) and part b) are completely independent from each other.

a) The address format of given computer system with a direct-mapped cache is shown below.

Tag: 11 bits	Index: 8 bits	Block Offset: 4 bits	Byte Offset 1 bit

Starting from the given memory address format:

i) (6 pts) Draw a diagram of the cache, numbering the beginning and ending lines.
ii) (2 pts) What is the size of the cache (bytes of data)?
iii) (2 pts) What is the size of the main memory (maximum)?

The # of byte offset bits is 1, so there are 2 bytes in a 16-bit data word. The # of block offset bits is 4, so there are 16 words in a block. So there are 2·16 = 32 bytes in each cache block. There are 8 index bits, so there are 2^8 = 256 cache lines. This is a direct-mapped cache, with 1 block per line, so 256 blocks in the cache. 256 blocks/cache x 32 bytes/block = 8 KB total data in cache. Address size of 24 bits means there is 2^{24} = 16 Mbytes of memory in this system, maximum.

Line #	*V*	*Tag*	*Data*
0			
1			
2			
. . . .	.	. . . . . .	. . . .
. . . .	.	. . . . . .	. . . .
253			
254			
255			

b) (10 points) The 2-way set associative cache shown below is used. It has block size of 16 bytes/block, and uses LRU replacement policy. Trace the string of memory address references given in the table on the next page, and show the final contents in the cache diagram below.

Line # V Tag Data V Tag Data

0						
1						
2						
. . . .	.	. . . . . .	. . . .	.	. . . . . .	. . . .
. . . .	.	. . . . . .	. . . .	.	. . . . . .	. . . .
61	*1*	*...0011*	*M[4063-4048]*	*1*	*...0010*	*M[3039-3024]*
62	*1*	*...0001*	*M[2031-2016]*			
63	*1*	*...0011*	*M[4095-4080]*	*1*	*...0010*	*M[3071-3056]*

Ref #	Address_{10}	Address_2	*Hit/Miss? What type of miss?*
1	3031	….0010 111101 0111	*Miss: cold-start*
2	1009	….0000 111111 0001	*Miss: cold-start*
3	978	….0000 111101 0010	*Miss: cold-start*
4	2016	….0001 111110 0000	*Miss: cold-start*
5	990	….0000 111101 1110	*Hit*
6	2037	….0001 111111 0101	*Miss: cold-start*
7	4055	….0011 111101 0111	*Miss: cold-start, 3031 goes out*
8	1014	….0000 111111 0110	*Hit*
9	3070	….0010 111111 1110	*Miss: cold-start,2037 goes out*
10	3032	….0010 111101 1000	*Miss: conflict, 990 goes out*
11	4052	….0011 111101 0100	*Hit*
12	4084	….0011 111111 0100	*Miss: cold-start, 1014 goes out*

For tracing the memory references, it is necessary to find the index bits. Since there are 16 bytes/block, the last 4 bits are the offset bits. The 6 bits in the middle are the index bits, the address of the particular cache line (from 0 to 63) that will be accessed. The remaining bits to the left are the tag bits. Since block size is 16 bytes, the data field in cache will contain values like M[X+15 to X], where X is the address of the lowest byte in the block, the byte whose 4 offset bits are 0000.

For tracing purposes, the table above contains the hits and misses, and if the set is full (i.e if both locations in the line are full) when a 3rd block is referenced, it shows the block that is replaced (by giving its most-recently accessed address) according to LRU policy.

Question 3 (Prob Set #6 Spring 2013) Cache modeling, and tracing memory reference strings

For a system in which memory references are word, not byte references, and which has a 3-way set-associative cache of size 24 words, where block size is 2 words:

a) Draw the cache structure diagram

Index	*V*	*Tag*	*Data*	*V*	*Tag*	*Data*	*V*	*Tag*	*Data*
00									
01									
10									
11									

b) For each of the memory references given in the string below, convert into binary and determine the tag bits, the index bits and the block offset bits.
1 134 212 1 135 213 162 2 44 41 221

First, the memory address format must be determined. 0 byte offset, 1 block offset, and 2 index bits are needed, and all the rest are tag bits (but we don't know how many).

Tag (? bits)	*Index (2 bits)*	*Bl. Off. (1 bit)*

Now all the addresses can be broken into their 3 fields:

1 =..0000000001	*..000000*	*00*	*1*
134=..0010000110	*..010000*	*11*	*0*
212=..0011000000	*..011000*	*00*	*0*
1 =..0000000001	*..000000*	*00*	*1*
135= ..0010000111	*..010000*	*11*	*1*
213=..0011000001	*..011000*	*00*	*1*
162=..0010100010	*..010100*	*01*	*0*
2 =..0000000010	*..000000*	*01*	*0*
44=..0000101100	*..000101*	*10*	*0*
41=..0000101001	*..000101*	*00*	*1*
221=..0011001001	*..011001*	*00*	*1*

c) Assuming LRU replacement policy (least recently used), trace the string of references, determine if each reference is a hit or a miss, apply the LRU policy, and show the final cache contents.

1= ..0000000001	*..000000*	*00*	*1*	*Miss*
134=..0010000110	*..010000*	*11*	*0*	*Miss*
212=..0011000000	*..011000*	*00*	*0*	*Miss*
1=..0000000001	*..000000*	*00*	*1*	*Hit*
135=..0010000111	*..010000*	*11*	*1*	*Hit*
213=..0011000001	*..011000*	*00*	*1*	*Hit*
162=..0010100010	*..010100*	*01*	*0*	*Miss*
2=..0000000010	*..000000*	*01*	*0*	*Miss*
44=..0000101100	*..000101*	*10*	*0*	*Miss*
41=..0000101001	*..000101*	*00*	*1*	*Miss*
221=..0011001001	*..011001*	*00*	*1*	*Miss*

Note that all the misses are <u>cold-start misses</u>, due to first reference to a block, except for the final miss. The final reference, to 221, causes a <u>conflict miss</u> in line #00, so M[1,0] is evicted (it is least recently used) and M[221,220] is placed in the cache.

Index	*V*	*Tag*	*Data*	*V*	*Tag*	*Data*	*V*	*Tag*	*Data*
00	*1*	*..011001*	*M[221,220]*	*1*	*..011000*	*M[213,212]*	*1*	*..000101*	*M[41,40]*
01	*1*	*..010100*	*M[163,162]*	*1*	*..000000*	*M[3,2]*			
10	*1*	*..000101*	*M[45,44]*						
11	*1*	*..010000*	*M[135,134]*						

Question 1 (Prob Set #7 Spring 2013) Modeling cache and tracing memory reference string

A parallel multiprocessor has many 64-bit processors, and the maximum amount of byte-addressable memory is 4 TBytes. As part of the memory hierarchy, each processor has its own local L1 cache for data and for instructions. Each L1 cache is 32 KB, 2-way set associative, with LRU replacement policy, and block size of 2 words.

a) Give the memory address format, showing the name of each field and the number of bits in that field.

Tag—28 bits	*Index—10 bits*	*Block offset-- 1 bit*	*Byte offset -- 3 bits*

The 4 TByte (= 2^{42} bytes) address space means addresses must be 42 bits. The 64-bit data words contain 8 bytes, so 3 bits are needed for byte offset. There are 2 words in a block in the L1 cache, so 1 bit of address is needed for block offset. There are 8 x 2 = 16 bytes of data in each block, therefore each 32Kbyte L1 cache contains 32K/16 = 2K blocks. Since it is a 2-way set associative cache, with 2 blocks per cache line (or set) this means there are 1K = 1024 = 2^{10} lines in the cache. So the index must be 10 bits, which leaves 42 – (10 + 1 + 3) = 28 bits for the tag field.

b) Draw the cache structure diagram for one of the L1 caches, with each set shown on a single line. Of course you don't have room to draw all the sets, so draw the first few and the last few and use … in between. For each set show the Valid Bits, the Tags, the Data fields, and of course the address of each set.

Address	*V*	*Tag*	*DataWord1-- DataWord0*	*V*	*Tag*	*DataWord1-- DataWord0*
0000000000						
0000000001						
0000000010						

. . .

. . .

1111111101						
1111111110						
1111111111						

c) A memory reference string for one of the L1 caches is given below.

i) Translate the decimal addresses into binary, and for each reference, determine if it is a Hit or a Miss. [HINT: divide the binary address bits into the correct fields, to make tracing the string in cache easier.]

Decimal Address	**Binary Address**	**Hit or Miss**
16411	*..0001 0000000001 1011*	*Miss--coldstart*
48	*..0000 0000000011 0000*	*Miss--coldstart*
16407	*..0001 0000000001 0111*	*Hit*
32788	*..0010 0000000001 0100*	*Miss--coldstart*
16447	*..0001 0000000011 1111*	*Miss--coldstart*
60	*..0000 0000000011 1100*	*Hit*
49206	*..0011 0000000011 0110*	*Miss--coldstart*
16400	*..0001 0000000001 0000*	*Hit*
31	*..0000 0000000001 1111*	*Miss--coldstart*
88	*..0000 0000000101 1000*	*Miss--coldstart*

Note that all the misses are compulsory misses (aka "coldstart" misses) which must happen the first time a block is referenced. But also note that 2-way associative is not enough to hold the 3rd block which was referenced in cache line 1 (M[32799 – 32784]) *and the 3rd block referenced in cache line 3 (M[16447 – 16432]). If these blocks are referenced again, they will miss and be brought back in, and another block evicted due to not enough space at a given cache address. This is called a "conflict" miss.*

ii) Show the final contents of the cache (either below, or using the cache you have already drawn in part b). Be sure to indicate the contents of each field in each block, if it is non-empty.

Address	*V*	*Tag*	*Data*	*V*	*Tag*	*Data*
0000000000						
0000000001	*1*	*..0001*	*M [16415 – 16400]*	*1*	*..0000*	*M[31 – 16]*
0000000010						
0000000011	*1*	*..0000*	*M[63 – 48]*	*1*	*..0011*	*M[49215 – 49200]*
0000000100						
0000000101	*1*	*..0000*	*M[95 – 80]*			

. . .
. . .