NYU Tandon School of Engineering

Fall 2022, ECE 6913

Homework Assignment 7

Instructor: Azeez Bhavnagarwala, email: ajb20@nyu.edu

Course Assistants

Varadraj Kakodkar (vns2008), Kartikay Kaushik (kk4332), Siddhanth Iyer (si2152), Swarnashri Chandrashekar (sc8781), Karan Sheth (kk4332), Haotian Zheng (hz2687), Haoren Zhang (kk4332), Varun Kumar (vs2411)

Homework Assignment 6 [released Friday November 11th 2022] [due Monday November 28th by 11:59PM]

You *are allowed* to discuss HW assignments with anyone. You are *not allowed* to share your solutions with other colleagues in the class. Please feel free to reach out to the Course Assistants or the Instructor during office hours or by appointment if you need any help with the HW. Please enter your responses in this Word document after you download it from NYU Classes. *Please use the Brightspace portal to upload your completed HW.*

.

1. Caches are important to providing a high-performance memory hierarchy to processors. Below is a list of 64-bit memory address references, given as word addresses.

`0x02, 0xb3, 0x2a, 0x01, 0xbe, 0x57, 0xbf, 0x0d, 0xb6, 0x2b, 0xbc, 0xfd`

1.1 For each of these references, identify the binary word address, the tag, and the index given a direct-mapped cache with 16 one-word blocks. Also list whether each reference is a hit or a miss, assuming the cache is initially empty.

Address references	Binary Reference	Tag	Index	Hit or Miss
0x02	0000 0010	0	2	Miss
0xb3	1011 0011	b	3	Miss
0x2a	0010 1010	2	a	Miss
0x01	0000 0001	0	1	Miss
0xbe	1011 1110	b	e	Miss
0x57	0101 0111	5	7	Miss
0xbf	1011 1111	b	f	Miss
0x0d	0000 1101	0	d	Miss
0xb6	1011 0110	b	6	Miss
0x2b	0010 1011	2	b	Miss
0xbc	1011 1100	b	c	Miss
0xfd	1111 1101	f	d	Miss

1.2 For each of these references, identify the binary word address, the tag, the index, and the offset given a direct-mapped cache with two-word blocks and a total size of eight blocks. Also list if each reference is a hit or a miss, assuming the cache is initially empty.

Address references	Binary Reference	Tag	Index	Offset	Hit or Miss
0x02	0000 0010	0	1	0	Miss
0xb3	1011 0011	b	1	1	Miss
0x2a	0010 1010	2	5	0	Miss
0x01	0000 0001	0	0	1	Miss
0xbe	1011 1110	b	7	0	Miss
0x57	0101 0111	5	3	1	Miss
0xbf	1011 1111	b	7	1	Hit
0x0d	0000 1101	0	6	1	Miss
0xb6	1011 0110	b	3	0	Miss
0x2b	0010 1011	2	5	1	Hit
0xbc	1011 1100	b	6	0	Miss
0xfd	1111 1101	f	6	1	Miss

1.3 You are asked to optimize a cache design for the given references. There are three direct-mapped cache designs possible, all with a total of eight words of data:
C1 has 1-word blocks,
C2 has 2-word blocks, and
C3 has 4-word blocks.

For C1, total number of blocks are 8. Thus, the index bits are 3 bits.

Address references	Binary Reference	Tag (first 5 bits)	C1	
			Index	Hit or Miss
0x02	0000 0010	0x00	2	Miss
0xb3	1011 0011	0x16	3	Miss
0x2a	0010 1010	0x05	2	Miss
0x01	0000 0001	0x00	1	Miss
0xbe	1011 1110	0x17	6	Miss
0x57	0101 0111	0x0a	7	Miss
0xbf	1011 1111	0x17	7	Miss
0x0d	0000 1101	0x01	5	Miss
0xb6	1011 0110	0x16	6	Miss
0x2b	0010 1011	0x05	3	Miss
0xbc	1011 1100	0x17	4	Miss
0xfd	1111 1101	0x1f	5	Miss

For C2, total number of blocks are 4. Thus, the index bits are 2 bits.

Address references	Binary Reference	Tag (first 5 bits)	C2	
			Index	Hit or Miss
0x02	0000 0010	0x00	1	Miss
0xb3	1011 0011	0x16	1	Miss
0x2a	0010 1010	0x05	1	Miss
0x01	0000 0001	0x00	0	Miss
0xbe	1011 1110	0x17	3	Miss
0x57	0101 0111	0x0a	3	Miss
0xbf	1011 1111	0x17	3	Miss
0x0d	0000 1101	0x01	2	Miss

0xb6	1011 0110	0x16	3	Miss
0x2b	0010 1011	0x05	1	Hit
0xbc	1011 1100	0x17	2	Miss
0xfd	1111 1101	0x1f	2	Miss

Address references	Binary Reference	Tag (first 5 bits)	C3	
			Index	Hit or Miss
0x02	0000 0010	0x00	1	Miss
0xb3	1011 0011	0x16	1	Miss
0x2a	0010 1010	0x05	1	Miss
0x01	0000 0001	0x00	0	Miss
0xbe	1011 1110	0x17	1	Miss
0x57	0101 0111	0x0a	1	Miss
0xbf	1011 1111	0x17	1	Miss
0x0d	0000 1101	0x01	0	Miss
0xb6	1011 0110	0x16	1	Miss
0x2b	0010 1011	0x05	1	Miss
0xbc	1011 1100	0x17	0	Miss
0xfd	1111 1101	0x1f	0	Miss

Thus, we can notice that C2 is the best design because only C2 has 1 hits.

2. *Section 5.3* shows the typical method to index a direct-mapped cache, specifically (Block address) modulo (Number of blocks in the cache). Assuming a 64-bit address and 1024 blocks in the cache, consider a different indexing function, specifically (Block address[63:54] XOR Block address[53:44]). Is it possible to use this to index a direct-mapped cache? If so, explain why and discuss any changes that might need to be made to the cache. If it is not possible, explain why.

Yes, it is possible. Any function that generates 10-btis output can be used as the index function for this cache. However, this function may not be the best design because of potential increase of conflicts. Also it may require larger tag.

3. For a direct-mapped cache design with a 64-bit address, the following bits of the address are used to access the cache.

Tag	Index	Offset
63–10	9–5	4–0

3.1 What is the cache block size (in words)?

4 words. Each block has 64 bits/ 8 bytes of data. For the offset inside one word, we need 3 bits. The other 2 bits can be used to find the index of blocks. Thus the cache block szie is 4 words.

3.2 How many blocks does the cache have?

32 blocks. Because we have 5 bits for indexing.

3.3 What is the ratio between total bits required for such a cache implementation over the data storage bits?

Beginning from power on, the following byte-addressed cache references are recorded.

Address												
Hex	00	04	10	84	E8	A0	400	1E	8C	C1C	B4	884
Dec	0	4	16	132	232	160	1024	30	140	3100	180	2180

The total bits need for the cache is:
32 blocks * 4 word per row * 8 bytes per word = 1024 bytes = 8192 bits

To store data, each line will additionally need 54 bits for tag and 1 bit for valid bit.
So the total bits for data storage is 8192 + (54 + 1) * 32 = 9952 bits

The ratio is 9952 / 8192 = 1.215

3.4 For each reference, list (1) its tag, index, and offset, (2) whether it is a hit or a miss, and (3) which bytes were replaced (if any).

Index = 5 bits
Offset = 5 bits

Hex Address	Binary address	Tag (first 2 bits)	Index	Offset	Hit/ Miss	Bytes replaced
0x00	0000 0000 0000	0x0	0x00	0x00	Miss	
0x04	0000 0000 0100	0x0	0x00	0x04	Hit	
0x10	0000 0001 0000	0x0	0x00	0x10	Hit	

0x84	0000 1000 0100	0x0	0x04	0x04	Miss	
0xe8	0000 1110 1000	0x0	0x07	0x08	Miss	
0xa0	0000 1010 0000	0x0	0x05	0x00	Miss	
0x400	0100 0000 0000	0x1	0x00	0x00	Miss	0x00-0x1f
0x1e	0000 0001 1110	0x0	0x00	0x1e	Miss	0x400-0x41f
0x8c	0000 1000 1100	0x0	0x04	0x0c	Hit	
0xc1c	1100 0001 1100	0x3	0x00	0x1c	Miss	0x00-0x1f
0xb4	0000 1011 0100	0x0	0x05	0x14	Hit	
0x884	1000 1000 0100	0x2	0x04	0x04	Miss	0x80-0x9f

3.5 What is the hit ratio?

4/12 = 0.333

3.6 List the final state of the cache, with each valid entry represented as a record of <*index, tag, data*>. For example,

<0, 3, Mem[0xC00]-Mem[0xC1F]>

The final state of this cache is:
<0, 3, Mem[0xc00] - Mem[0xc1f]>
<4, 2, Mem[0x880] - Mem[0x89f]>
<5, 0, Mem[0x0a0] - Mem[0x0bf]>
<7, 0, Mem[0x0e0] - Mem[0x0ff]>

4. Recall that we have two write policies and two write allocation policies, and their combinations can be implemented either in L1 or L2 cache. Assume the following choices for L1 and L2 caches:

L1	**L2**
Write through, non-write allocate	Write back, write allocate

4.1 Buffers are employed between different levels of memory hierarchy to reduce access latency. For this given configuration, list the possible buffers needed between L1 and L2 caches, as well as L2 cache and memory.

L2 cache has higher write miss penalty than L1 cache. Thus the write buffer between L1 and L2 will help imporve the write miss latency of L2. The L2 cache will be benefited by the write buffers when the dirty block is replaced. That is because the new block will be read in before the dirty block is physically written to memory.

4.2 Describe the procedure of handling an L1 write-miss, considering the components involved and the possibility of replacing a dirty block.

When there is a write-miss in L1, the word needed is written directly to L2 because we are using the buffer right now. If this action causes a miss in L2, a block must be written to the L2 cache.

4.3 For a multilevel exclusive cache configuration (a block can only reside in one of the L1 and L2 caches), describe the procedures of handling an L1 write-miss and an L1 read-miss, considering the components involved and the possibility of replacing a dirty block.

After an L1 cache write miss, the block will residen in L2 rather than L1. Then there will be a read miss on the same block whcih requires the block in L2 to be written back to memory, transferred to L1.

5. Consider the following program and cache behaviors.

Data Reads per 1000 Instructions	Data Writes per 1000 Instructions	Instruction Cache Miss Rate	Data Cache Miss Rate	Block Size (bytes)
250	100	0.30%	2%	64

5.1 Suppose a CPU with a write-through, write-allocate cache achieves a CPI of 2. What are the read and write bandwidths (measured by bytes per cycle) between RAM and the cache? (Assume each miss generates a request for one block.)

Because CPI is 2, it means on average there is 0.5 instrcutions per cycle.

Read traffic:
25% of the instructions are data reading and 2% of them are data cache miss. Thus, a data read missing will generate:
0.5 * 25% * 2% * 64 = 0.16 bytes per cycle

10% of the instructions are data writing and 2% of them are data cache miss. Since the cache use write-allocate policy, when there is a write miss it also requires to read data from RAM. This part of reading traffic is:
0.5 * 10% * 2% * 64 = 0.064 bytes per cycle

0.3% of instructions will cause cache miss. This part of read traffic is:
0.5 * 0.3% * 64 = 0.096 bytes per cycle

Write traffic:
Because this cache use write-through policy, there is only 1 word (8 bytes) written back to the memory. This part of write traffic is:
0.5 * 10% * 8 = 0.4 bytes per cycle

In conclusion:
There are totally 0.16 + 0.064 + 0.096 = 0.32 bytes per cycle of read bandwidths and 0.4 bytes per cycle of write bandwidths.

5.2 For a write-back, write-allocate cache, assuming 30% of replaced data cache blocks are dirty, what are the read and write bandwidths needed for a CPI of 2?

The data read bandwidth is the same as 5.1 which is 0.32 bytes per cycle.

For data write bandwidth, it is different because now the cache use write-back policy. When there is a cache miss (both read and write), the data is written to the memory. Thus the write bandwidth is :
0.5 * (25% + 10%) * 2% * 0.3% * 64 = 6.72 * 10^(-5) bytes per cycle

6. Media applications that play audio or video files are part of a class of workloads called "streaming" workloads (i.e., they bring in large amounts of data but do not reuse much of it). Consider a video streaming workload that accesses a 512 KiB working set sequentially with the following word address stream:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9 …

6.1 Assume a 64 KiB direct-mapped cache with a 32-byte block. What is the miss rate for the address stream above? How is this miss rate sensitive to the size of the cache or the working set? How would you categorize the misses this workload is experiencing, based on the 3C model?

The block size is 32 bytes so it contains 4 words. Thus on average the miss rate is 1/4.

The misses here is not possible to be avoided. It is not sensitive to the size of the cache or the size of the working set. It is sensitive to the block size and how the cache is accessed.

6.2 Re-compute the miss rate when the cache block size is 16 bytes, 64 bytes, and 128 bytes. What kind of locality is this workload exploiting?

For 16-bytes, the miss rate is: 1/2.
For 64-bytes, the miss rate is: 1/8.
For 128-bytes, the miss rate is: 1/16.

6.3 "*Prefetching*" is a technique that leverages predictable address patterns to speculatively bring in additional cache blocks when a particular cache block is accessed. One example of prefetching is a stream buffer that prefetches sequentially adjacent cache blocks into a separate buffer when a particular cache block is brought in. If the data are found in the prefetch buffer, it is considered as a hit, moved into the cache, and the next cache block is prefetched. Assume a two-entry stream buffer; and, assume that the cache latency is such that a cache block can be loaded before the computation on the previous cache block is completed. What is the miss rate for the address stream above?

Here the missrate is 0. Because the pre-fetch buffer will always prepare for the next requested data.

7. Cache block size (B) can affect both miss rate and miss latency. Assuming a machine with a base CPI of 1, and an average of 1.35 references (both instruction and data) per instruction, find the block size that minimizes the total miss latency given the following miss rates for various block sizes.

8: 4%	16: 3%	32: 2%	64: 1.5%	128: 1%

7.1 What is the optimal block size for a miss latency of 20 ×B cycles?

We can compare the AMAT for different caches to select the optimal block size.

AMAT for 8 blocks: 4% * 20 * 8 = 6.4

AMAT for 16 blocks: 3% * 20 * 16 = 9.6

AMAT for 32 blocks: 2% * 20 * 32 = 12.8

AMAT for 64 blocks: 1.5% * 20 * 64 = 19.2

AMAT for 128 blocks: 1% * 20 * 128 = 25.6

Thus, the optimal block size is 8.

7.2 What is the optimal block size for a miss latency of 24 +B cycles?
Similarly, we can have:

AMAT for 8 blocks: 4% * (24 + 8)= 1.28

AMAT for 16 blocks: 3% * (24 + 16)= 1.20

AMAT for 32 blocks: 2% * (24 + 32)= 1.12

AMAT for 64 blocks: 1.5% * (24 + 64)= 1.32

AMAT for 128 blocks: 1% * (24 + 128)= 1.52

Thus, the optimal block size is 32.

7.3 For constant miss latency, what is the optimal block size?

If the miss latency is constant, then the AMAT only depends the miss rate. Thus, the optimal block size is 128.

8. In this exercise, we will look at the different ways capacity affects overall performance. In general, cache access time is proportional to capacity. Assume that main memory accesses take 70 ns and that 36% of all instructions access data memory. The following table shows data for L1 caches attached to each of two processors, P1 and P2.

	L1 Size	L1 Miss Rate	L1 Hit Time
P1	2 KiB	8.0%	0.66 ns
P2	4 KiB	6.0%	0.90 ns

8.1 Assuming that the L1 hit time determines the cycle times for P1 and P2, what are their respective clock rates?

P1: 1 / (0.66 * 10^ -9) = 1.515 GHz
P2: 1 / (0.90 * 10^ -9) = 1.111 GHz

8.2 What is the Average Memory Access Time for P1 and P2 (in cycles)?

P1: 1 + 8% * (70 / 0.66) = 9.56 cycles
P2: 1 + 6% * (70 / 0.9) = 5.68 cycles

8.3 Assuming a base CPI of 1.0 without any memory stalls, what is the total CPI for P1 and P2? Which processor is faster? (When we say a "base CPI of 1.0", we mean that instructions complete in one cycle, unless either the instruction access or the data access causes a cache miss.)

P1: 1 + (8% + 8% * 36%) * (70 / 0.66) = 12.64 CPI
12.64 * 0.66 = 8.34 ns/ instruction

P2: 1 + (6% + 6% * 36%) * (70 / 0.90) = 7.36 CPI
7.36 * 0.90 = 6.63 ns/ instruction

Thus, P2 is faster.

we will now consider the addition of an L2 cache to P1 (to presumably make up for its limited L1 cache capacity). Use the L1 cache capacities and hit times from the previous table when solving these problems. The L2 miss rate indicated is its local miss rate.

L2 Size	L2 Miss Rate	L2 Hit Time
1 MiB	95%	5.62 ns

8.4 What is the AMAT for P1 with the addition of an L2 cache? Is the AMAT better or worse with the L2 cache?

AMAT of P1 with L2 cache:

$1 + 8\% * ((5.62 / 0.66) + 95\% * (70 / 0.66)) = 9.85$ cycles

Thus AMAT of P1 with L2 cache is slightly slower than without L2 cache.

8.5 Assuming a base CPI of 1.0 without any memory stalls, what is the total CPI for P1 with the addition of an L2 cache?

Total CPI $= 9.85 + 36\% * (9.85 - 1) = 13.04$ cycles

8.6 What would the L2 miss rate need to be in order for P1 with an L2 cache to be faster than P1 without an L2 cache?

We want to reach that:

AMAT with L2 < AMAT with only L1

$1 + 8\% * ((5.62 / 0.66) + \text{miss rate} * (70 / 0.66)) < 9.56$ cycles

miss rate $< 91.59\%$

8.7 What would the L2 miss rate need to be in order for P1 with an L2 cache to be faster than P2 without an L2 cache?

For this question we want is

CPI of P1 * 0.66ns < 6.63ns

(AMAT of P1 + 36% * (AMAT of P1 - 1)) * 0.66ns < 6.63ns

AMAT of P1 < 7.65

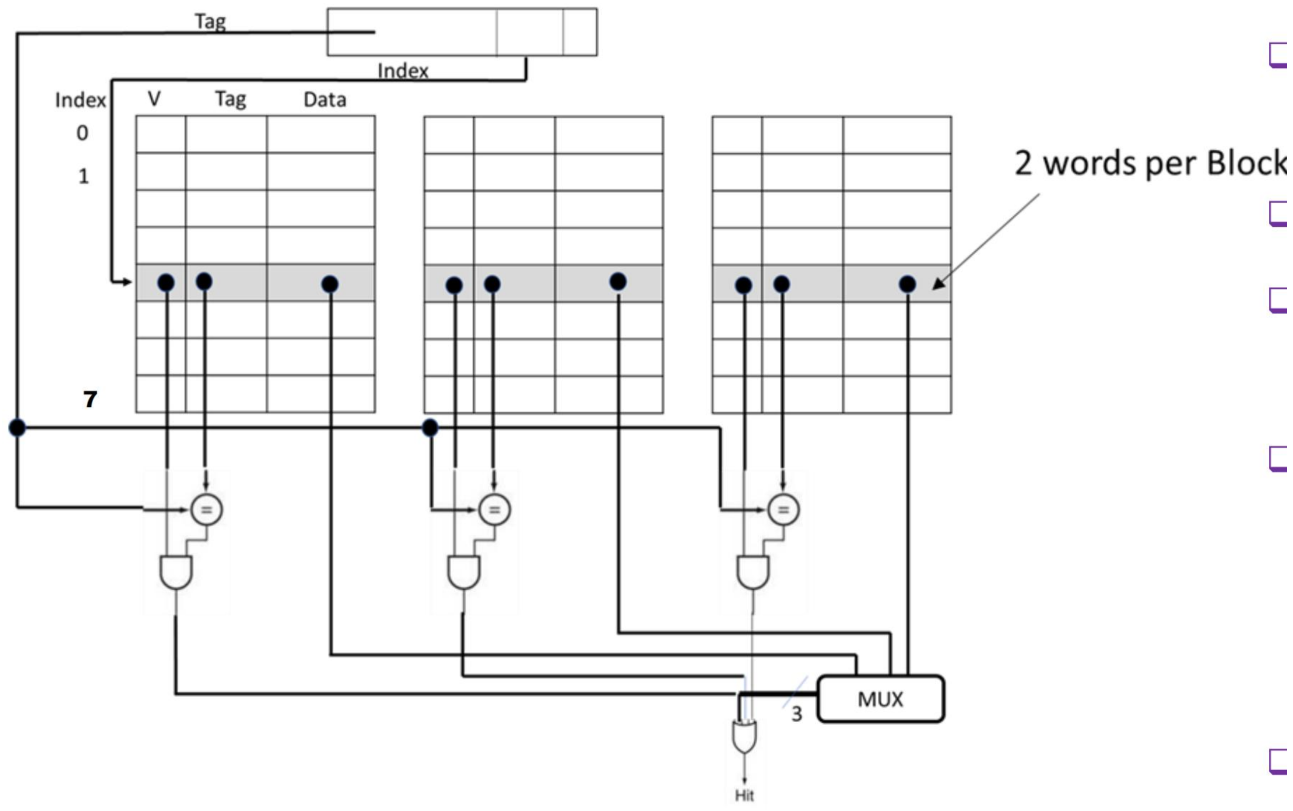
$1 + 8\% * ((5.62 / 0.66) + \text{miss rate} * (70 / 0.66)) < 7.65$

miss rate $< 69.28\%$

9. This exercise examines the effect of different cache designs, specifically comparing associative caches to the direct-mapped caches from *Section 5.4*. For these exercises, refer to the sequence of word address shown below.

`0x03, 0xb4, 0x2b, 0x02, 0xbe, 0x58, 0xbf, 0x0e, 0x1f, 0xb5,`
`0xbf, 0xba, 0x2e, 0xce`

9.1 Sketch the organization of a three-way set associative cache with two-word blocks and a total size of 48 words. Your sketch should have a style similar to *Figure 5.18*, but clearly show the width of the tag and data fields.



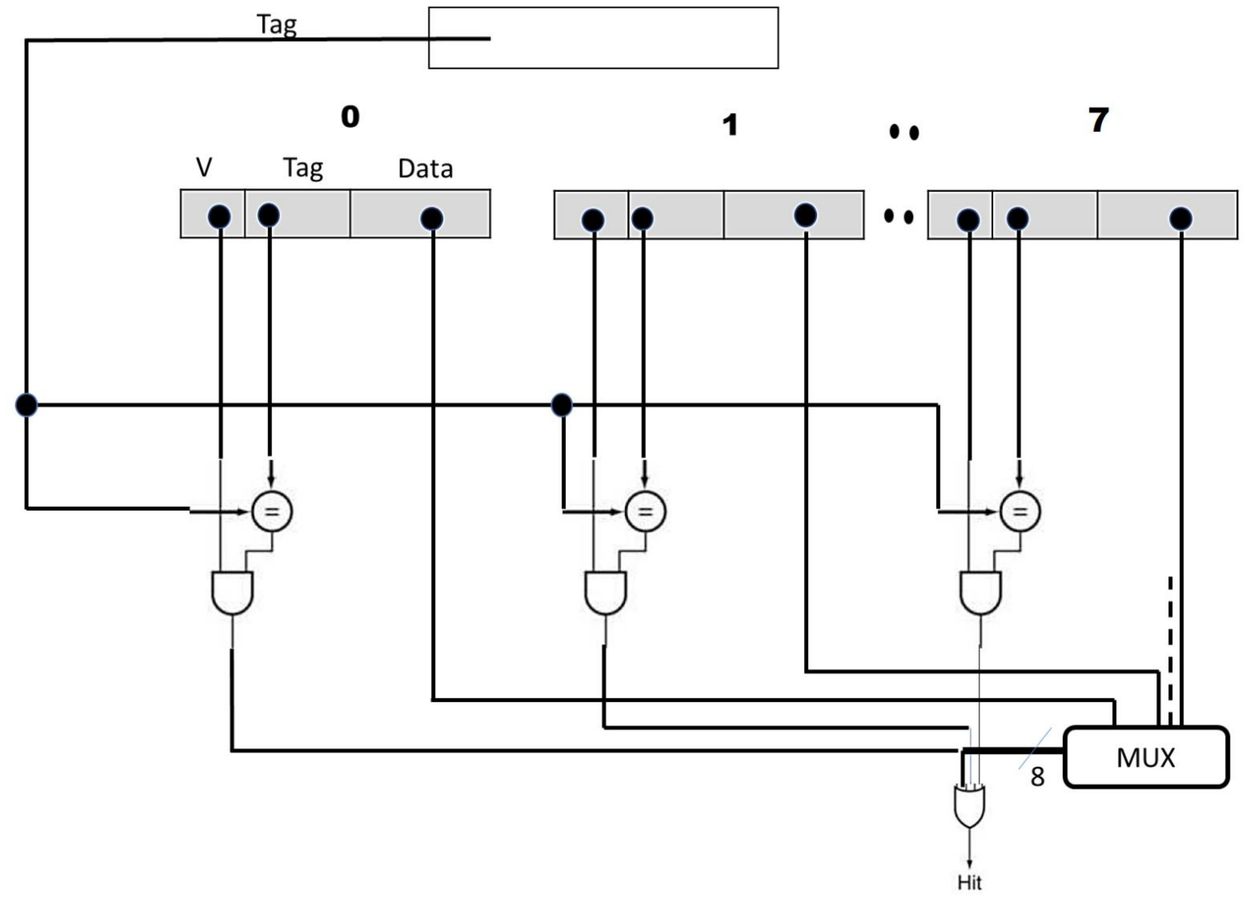
9.2 Trace the behavior of the cache from Exercise 9.1 Assume a true LRU replacement policy. For each reference, identify

- *the binary word address,*
- *the tag,*
- *the index,*
- *the offset*
- *whether the reference is a hit or a miss, and*
- *which tags are in each way of the cache after the reference has been handled.*

Hex word address	Binary address	Tag	Index	Offset	Hit/ Miss	Way 0	Way 1	Way 2
0x03	0000 0011	0x0	1	1	Miss	Tag(1)=0		
0xb4	1011 0100	0xb	2	0	Miss	Tag(1)=0 Tag(2)=b		
0x2b	0010 1011	0x2	5	1	Miss	Tag(1)=0 Tag(2)=b Tag(5)=2		
0x02	0000 0010	0x0	1	0	Hit	Tag(1)=0 Tag(2)=b Tag(5)=2		
0xbe	1011 1110	0xb	7	0	Miss	Tag(1)=0 Tag(2)=b Tag(5)=2 Tag(7)=b		
0x58	0101 1000	0x5	4	0	Miss	Tag(1)=0 Tag(2)=b Tag(5)=2 Tag(7)=b Tag(4)=5		
0xbf	1011 1111	0xb	7	1	Hit	Tag(1)=0 Tag(2)=b Tag(5)=2 Tag(7)=b Tag(4)=5		
0x0e	0000 1110	0x0	7	0	Miss	Tag(1)=0 Tag(2)=b Tag(5)=2 Tag(7)=b Tag(4)=5	Tag(7)=0	
0x1f	0001 1111	0x1	7	1	Miss	Tag(1)=0 Tag(2)=b Tag(5)=2 Tag(7)=b Tag(4)=5	Tag(7)=0	Tag(7)=1
0xb5	1011 0101	0xb	2	1	Hit	Tag(1)=0 Tag(2)=b Tag(5)=2 Tag(7)=b Tag(4)=5	Tag(7)=0	Tag(7)=1

0xbf	1011 1111	0xb	7	1	Hit	Tag(1)=0 Tag(2)=b Tag(5)=2 Tag(7)=b Tag(4)=5	Tag(7)=0	Tag(7)=1
0xba	1011 1010	0xb	5	0	Miss	Tag(1)=0 Tag(2)=b Tag(5)=2 Tag(7)=b Tag(4)=5	Tag(7)=2 Tag(5)=b	Tag(7)=1
0x2e	0010 1110	0x2	7	0	Miss	Tag(1)=0 Tag(2)=b Tag(5)=2 Tag(7)=b Tag(4)=5	Tag(7)=2 Tag(5)=b	Tag(7)=1
0xce	1100 1110	0xc	7	0	Miss	Tag(1)=0 Tag(2)=b Tag(5)=2 Tag(7)=b Tag(4)=5	Tag(7)=2 Tag(5)=b	Tag(7)=c

9.3 Sketch the organization of a fully associative cache with one-word blocks and a total size of eight words. Your sketch should have a style similar to Figure 5.18, but clearly show the width of the tag and data fields.



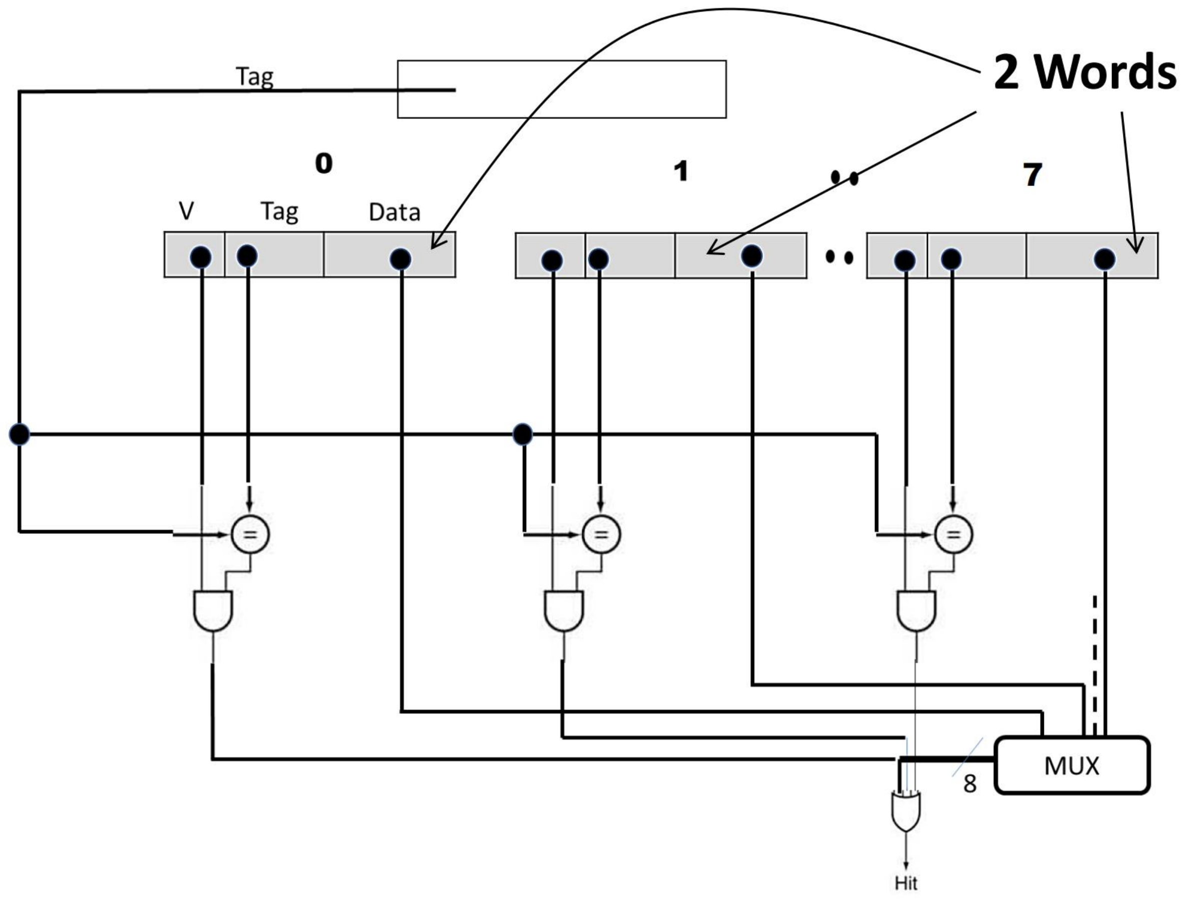
9.4 Trace the behavior of the cache from Exercise 9.3. Assume a true LRU replacement policy. For each reference, identify

- *the binary word address,*
- *the tag,*
- *the index,*
- *the offset,*
- *whether the reference is a hit or a miss*
- *the contents of the cache after each reference has been handled.*

Hex word address	Binary address	Tag	Hit/ Miss	Contents
0x03	0000 0011	0x03	Miss	3
0xb4	1011 0100	0xb4	Miss	3,b4
0x2b	0010 1011	0x2b	Miss	3,b4,2b
0x02	0000 0010	0x02	Miss	3,b4,2b,2
0xbe	1011 1110	0xbe	Miss	3,b4,2b,2,be
0x58	0101	0x58	Miss	3,b4,2b,2,be,58

	1000			
0xbf	1011 1111	0xbf	Miss	3,b4,2b,2,be,58,bf
0x0e	0000 1110	0x0e	Miss	3,b4,2b,2,be,58,bf,e
0x1f	0001 1111	0x1f	Miss	b4,2b,2,be,58,bf,e,1f
0xb5	1011 0101	0xb5	Miss	2b,2,be,58,bf,e,1f,b5
0xbf	1011 1111	0xbf	Hit	2b,2,be,58,e,1f,b5,bf
0xba	1011 1010	0xba	Miss	2,be,58,e,1f,b5,bf,ba
0x2e	0010 1110	0x2e	Miss	be,58,e,1f,b5,bf,ba,2e
0xce	1100 1110	0xce	Miss	58,e,1f,b5,bf,ba,2e,ce

9.5 Sketch the organization of a fully associative cache with two-word blocks and a total size of eight words. Your sketch should have a style similar to Figure 5.18, but clearly show the width of the tag and data fields.



9.6 Trace the behavior of the cache from Exercise 9.5. Assume an LRU replacement policy.

For each reference, identify

- *the binary word address,*
- *the tag,*
- *the index,*
- *the offset,*
- *whether the reference is a hit or a miss,*
- *the contents of the cache after each reference has been handled.*

Hex word address	Binary address	Tag	Offset	Hit/ Miss	Contents
0x03	0000 0011	0x01	1	Miss	[2,3]
0xb4	1011 0100	0x5a	0	Miss	[2,3],[b4,b5]
0x2b	0010 1011	0x15	1	Miss	[2,3],[b4,b5],[2a,2b]
0x02	0000 0010	0x01	0	Hit	[b4,b5],[2a,2b],[2,3]
0xbe	1011 1110	0x5f	0	Miss	[b4,b5],[2a,2b],[2,3],[be,bf]
0x58	0101 1000	0x2c	0	Miss	[2a,2b],[2,3],[be,bf],[58,59]
0xbf	1011 1111	0x5f	1	Hit	[2a,2b],[2,3],[58,59],[be,bf]
0x0e	0000 1110	0x07	0	Miss	[2,3],[58,59],[be,bf],[e,f]
0x1f	0001 1111	0x0f	1	Miss	[58,59],[be,bf],[e,f],[1e,1f]
0xb5	1011 0101	0x5a	1	Miss	[be,bf],[e,f],[1e,1f],[b4,b5]
0xbf	1011 1111	0x5f	1	Hit	[e,f],[1e,1f],[b4,b5],[be,bf]
0xba	1011 1010	0x5d	0	Miss	[1e,1f],[b4,b5],[be,bf],[ba,bb]
0x2e	0010 1110	0x17	0	Miss	[b4,b5],[be,bf],[ba,bb],[2e,2f]
0xce	1100 1110	0x67	0	Miss	[be,bf],[ba,bb],[2e,2f],[ce,cf]

9.7 Repeat Exercise 9.6 using MRU (most recently used) replacement.

Hex word address	Binary address	Tag	Offset	Hit/ Miss	Contents
0x03	0000 0011	0x01	1	Miss	[2,3]
0xb4	1011 0100	0x5a	0	Miss	[2,3],[b4,b5]
0x2b	0010 1011	0x15	1	Miss	[2,3],[b4,b5],[2a,2b]
0x02	0000 0010	0x01	0	Hit	[b4,b5],[2a,2b],[2,3]
0xbe	1011 1110	0x5f	0	Miss	[b4,b5],[2a,2b],[2,3],[be,bf]
0x58	0101 1000	0x2c	0	Miss	[b4,b5],[2a,2b],[2,3],[58,59]
0xbf	1011 1111	0x5f	1	Hit	[b4,b5],[2a,2b],[2,3],[be,bf]
0x0e	0000 1110	0x07	0	Miss	[b4,b5],[2a,2b],[2,3],[e,f]
0x1f	0001 1111	0x0f	1	Miss	[b4,b5],[2a,2b],[2,3],[1e,1f]
0xb5	1011 0101	0x5a	1	Miss	[2a,2b],[2,3],[1e,1f],[b4,b5]
0xbf	1011 1111	0x5f	1	Hit	[2a,2b],[2,3],[1e,1f],[be,bf]
0xba	1011 1010	0x5d	0	Miss	[2a,2b],[2,3],[1e,1f],[ba,bb]
0x2e	0010 1110	0x17	0	Miss	[2a,2b],[2,3],[1e,1f],[2e,2f]
0xce	1100 1110	0x67	0	Miss	[2a,2b],[2,3],[1e,1f],[ce,cf]

9.8 Repeat Exercise 9.6 using the optimal replacement policy (i.e., the one that gives the lowest miss rate).

Hex word address	Binary address	Tag	Offset	Hit/ Miss	Contents
0x03	0000 0011	0x01	1	Miss	[2,3]
0xb4	1011 0100	0x5a	0	Miss	[2,3],[b4,b5]
0x2b	0010 1011	0x15	1	Miss	[2,3],[b4,b5],[2a,2b]
0x02	0000 0010	0x01	0	Hit	[2,3],[b4,b5],[2a,2b]

0xbe	1011 1110	0x5f	0	Miss	[2,3],[b4,b5],[2a,2b],[be,bf]
0x58	0101 1000	0x2c	0	Miss	[58,59],[b4,b5],[2a,2b],[be,bf]
0xbf	1011 1111	0x5f	1	Hit	[58,59],[b4,b5],[2a,2b],[be,bf]
0x0e	0000 1110	0x07	0	Miss	[e,f],[b4,b5],[2a,2b],[be,bf]
0x1f	0001 1111	0x0f	1	Miss	[1e,1f],[b4,b5],[2a,2b],[be,bf]
0xb5	1011 0101	0x5a	1	Miss	[1e,1f],[b4,b5],[2a,2b],[be,bf]
0xbf	1011 1111	0x5f	1	Hit	[1e,1f],[b4,b5],[2a,2b],[be,bf]
0xba	1011 1010	0x5d	0	Miss	[1e,1f],[b4,b5],[ba,bb],[be,bf]
0x2e	0010 1110	0x17	0	Miss	[1e,1f],[b4,b5],[2e,2f],[be,bf]
0xce	1100 1110	0x67	0	Miss	[1e,1f],[b4,b5],[ce,cf],[be,bf]

10. Multilevel caching is an important technique to overcome the limited amount of space that a first-level cache can provide while still maintaining its speed. Consider a processor with the following parameters:

Base CPI, No Memory Stalls	Processor Speed	Main Memory Access Time	First-Level Cache Miss Rate per Instruction**	Second-Level Cache, Direct-Mapped Speed	Miss Rate with Second-Level Cache, Direct-Mapped	Second-Level Cache, Eight-Way Set Associative Speed	Miss Rate with Second-Level Cache, Eight-Way Set Associative
1.5	2 GHz	100 ns	7%	12 cycles	3.5%	28 cycles	1.5%

***First Level Cache miss rate is per instruction. Assume the total number of L1 cache misses*
(instruction and data combined) is equal to 7% of the number of instructions.

10.1 Calculate the CPI for the processor in the table using: 1) only a first-level cache, 2) a second-level direct mapped cache, and 3) a second-level eight-way set associative cache. How do these numbers change if main memory access time doubles? (Give each change as both an absolute CPI and a percent change.) Notice the extent to which an L2 cache can hide the effects of a slow memory.

The original main memory access time is 100 ns.
Since the processor speed is 2 GHz, the cycle time is 1 / 2GHz = 0.5 ns
Thus a main memory access requires 100 / 0.5 = 200 cycles.

The CPI with original main memory access time:
L1 only: 1.5 + 7% * 200 = 15.5
A L2 direct mapped cache: 1.5 + 7% * (12 + 3.5% * 200) = 2.83
L2 8-way associative cache: 1.5 + 7% * (28 + 1.5% * 200) = 3.67

If the main memory access time is doubled:
L1 only: 1.5 + 7% * 400 = 29.5, it is increased by (29.5 / 15.5) - 1 = 90.32%
A L2 direct mapped cache: 1.5 + 7% * (12 + 3.5% * 200) = 3.32 , it is increased by (3.32 / 2.83) - 1 = 17.31%
L2 8-way associative cache: 1.5 + 7% * (28 + 1.5% * 200) = 3.88, it is increased by (3.88 / 3.67) - 1 = 5.72%

10.2 It is possible to have an even greater cache hierarchy than two levels? Given the processor above with a second-level, direct-mapped cache, a designer wants to add a third-level cache that takes 50 cycles to access and will have a 13% miss rate. Would this provide better performance? In general, what are the advantages and disadvantages of adding a third-level cache?

With a L3 cache, the CPI is:
1.5 + 7% * (12 + 3.5% * (50 + 13% * 100)) = 2.49435
So it does reduce the CPI.

The main advantage is reducing the overall memory access time.
The disadvantage is adding a L3 cache will need more physical spaces.

10.3 In older processors, such as the Intel Pentium or Alpha 21264, the second level of cache was external (located on a different chip) from the main processor and the first-level cache. While this allowed for large second-level caches, the latency to access the cache was much higher, and the bandwidth was typically lower because the second-level cache ran at a lower frequency. Assume a 512 KiB off-chip second level cache has a miss rate of 4%. If each additional 512 KiB of cache lowered miss rates by 0.7%, and the cache had a total access time of 50 cycles, how big would the cache have to be to match the performance of the second-level direct-mapped cache listed above?

It is impossible to match the performance of the second level direct mapped cache.

The reason is if we want to match the performance, assuming the k is miss rate, the CPI needs to be:

$1.5 + 7\% * (50 + k * 200) < 2.83$

Solving this inequality we can have:

$k < -0.155$

which means we need a negative miss rate. Thus it's impossible to match the performance.