Homework 4

5.2Caches 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.0x03, 0xb4, 0x2b, 0x02, 0xbf, 0x58, 0xbe, 0x0e, 0xb5, 0x2c, 0xba, 0xfd

5.2.1[10] <§5.3> 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.

Solution:

Tag: 28 bits (Only last 4 are shown)

Index: 4 bits

Word address	tag	index	hit/miss
0x03	0	3	M
0xb4	b	4	M
0x2b	2	b	M
0x02	0	2	M
0xbf	b	f	M
0x58	5	8	M
0xbe	b	e	M
0x0e	0	e	M
0xb5	b	5	M
0x2c	2	c	M
0xba	b	a	M
0xfd	f	d	M

5.2.2[10] <§5.3> 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.

Solution:

Tag 4 bits

Index 3 bits

Offset 1 bit

Word address	tag	index	Offset	hit/miss
0x03	0	1	1	M
0xb4	b	2	0	M
0x2b	2	5	1	M
0x02	0	1	0	H
0xbf	b	7	1	M
0x58	5	4	0	M
0xbe	b	7	0	H
0x0e	0	7	0	M
0xb5	b	2	1	H
0x2c	2	6	0	M
0xba	b	5	0	M
0xfd	f	6	1	M

5.2.3[20] <§§5.3, 5.4> 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.

Solution:

C1

Tag 5 bits

Index 3 bits

Word address	binary	tag(hex)	index	hit/miss
0x03	0000_0011	0x00	3	M
0xb4	1011_0100	0x16	4	M
0x2b	0010_1011	0x05	3	M
0x02	0000_0010	0x00	2	M
0xbf	1011_1111	0x17	7	M
0x58	0101_1000	0x0b	0	M
0xbe	1011_1110	0x17	6	M
0x0e	0000_1110	0x01	6	M
0xb5	1011_0101	0x16	5	M
0x2c	0010_1100	0x05	4	M
0xba	1011_1010	0x17	2	M
0xfd	1111_1101	0x1f	5	M

C2

Tag 5 bits

Index 2 bits

Word address	binary	tag(hex)	index	hit/miss
0x03	0000_0011	0x00	1	M
0xb4	1011_0100	0x16	2	M
0x2b	0010_1011	0x05	1	M
0x02	0000_0010	0x00	1	M
0xbf	1011_1111	0x17	3	M
0x58	0101_1000	0x0b	0	M
0xbe	1011_1110	0x17	3	H
0x0e	0000_1110	0x01	3	M
0xb5	1011_0101	0x16	2	H
0x2c	0010_1100	0x05	2	M
0xba	1011_1010	0x17	1	M
0xfd	1111_1101	0x1f	2	M

C3
Tag 5 bits

Index 1 bit

Word address	binary	tag(hex)	index	hit/miss
0x03	0000_0011	0x00	0	M
0xb4	1011_0100	0x16	1	M
0x2b	0010_1011	0x05	0	M
0x02	0000_0010	0x00	0	M
0xbf	1011_1111	0x17	1	M
0x58	0101_1000	0x0b	0	M
0xbe	1011_1110	0x17	1	H
0x0e	0000_1110	0x01	1	M
0xb5	1011_0101	0x16	1	M
0x2c	0010_1100	0x05	1	M
0xba	1011_1010	0x17	0	M
0xfd	1111_1101	0x1f	1	M

C1 miss rate = 100%

C2 miss rate = 10/12 = 83%

C3 miss rate = 11/12 = 92%

5.3 By convention, a cache is named according to the amount of data it contains (i.e., a 4 KiB cache can hold 4 KiB of data); however, caches also require SRAM to store metadata such as tags and valid bits. For this exercise, you will examine how a cache's configuration affects the total amount of SRAM needed to implement it as well as the performance of the cache. For all parts, assume that the caches are byte addressable, and that addresses and words are 64 bits.

5.3.1[10] <§5.3> Calculate the total number of bits required to implement a 32 KiB cache with two-word blocks.

Solution:

Given memory address as 64 [bits] and cache size as 32 [KiB].

Block size = 2 words = 2 * 8 B = 16 B

Number of blocks = 32 KiB / 16 B = 2 Ki = 2^11 blocks

Index bits = log2(number of blocks) = log2(2^11) = 11 bits

Offset bits = 1 word select (log2(2 word/block))+ 3 byte select (log2(8 B/word)) = 4 bits

Tag bits = 64 bit address - 11 index bits - 4 offset bits = 49 bits

Total bits/block = tag bits + data bits + 1 valid bit = 49 + 128 +1 = 178 bits

Total bits in cache = 178 bits/block * 2^11 blocks = 364544 bits

5.3.2[10] <§5.3> Calculate the total number of bits required to implement a 64 KiB cache with 16-word blocks. How much bigger is this cache than the 32 KiB cache described in Exercise 5.3.1? (Notice that, by changing the block size, we doubled the amount of data without doubling the total size of the cache.)

Solution:

Given memory address as 64 [bits] and cache size as 64 [KiB].

Block size = 16 words = 16 * 8 B = 128 B

Number of blocks = 64 KiB / 128 B = 512 = 2^9 blocks

Index bits = log2(number of blocks) = log2(2^9) = 9 bits

Offset bits = 4 word select (log2(16 word/block)) + 3 byte select (log2(8 B/word)) = 7 bits

Tag bits = 64 bit address - 9 index bits - 7 offset bits = 48 bits

Total bits/block = tag bits + data bits + 1 valid bit = 48 + 1024 +1 = 1073 bits

Total bits in cache = 1073 bits/block * 2^9 blocks = 549357

549357/364544 = 1.507x larger

5.3.3[5] <§5.3> Explain why this 64 KiB cache, despite its larger data size, might provide slower performance than the first cache.

Solution:

Typically increased size comes with increased hit latency and larger miss penalties. Because the block size is larger, it takes longer to select the correct word/byte on a hit. Likewise more data needs to be loaded in on a miss.

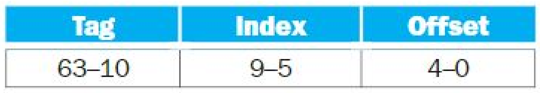
5.3.4[10] <§§5.3, 5.4> Generate a series of read requests that have a lower miss rate on a 32 KiB two-way set associative cache than on the cache described in Exercise 5.3.1.

Solution:

The following shows that the lower miss rate on a 32 KiB two-way set associative cache than on the 32 KiB.

Address	32 KiB Block	Hit or Miss	32 KiB Set	Hit or Miss
0x00000	Block 0	Miss	Set 0 - Way 0	Miss
0x10000	Block 0	Miss	Set 0 - Way 1	Miss
0x00000	Block 0	Miss	Set 0 - Way 0	Hit

5.5 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

5.5.1[5] <§5.3> What is the cache block size (in words)?

Solution:

Offset = byte select + word select

8 B/word (64 bit word) = 3 byte select bits

Words select bits = 5 offset bits - 3 byte select bits = 2 bits

Words /block = 2^2 = 4 words/block

5.5.2[5] <§5.3> How many blocks does the cache have?

Solution:

Since the cache size is determined by the index bits, then with 5 index bits we have 2^5 or 32 blocks.

5.5.3[5] <§5.3> What is the ratio between total bits required for such a cache implementation over the data storage bits?

Solution:

Cache size = 32 blocks * 4 words/block * 64 bits/word = 8192 bits

Determine tag information storage=54*2^5=1728 [bits].

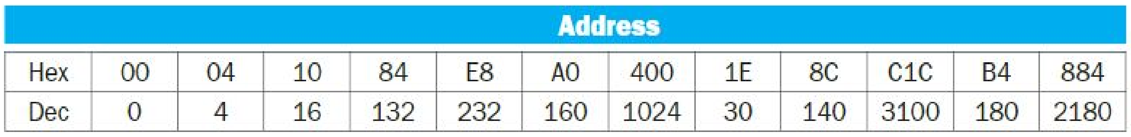
Total tag bits = 32 blocks * 54 tags bits/block = 1728 bits

Valid bits = 32 blocks * 1 bit/block = 32 bits

Determine total bits required for cache = 8192+1728 + 32 = 9952 [bits].

Total bits:data bits = 9952 total /8192 data = 1.215

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

5.5.4[20] <§5.3> 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).

Solution:

00 -> 0000 0000 0000 0000 -> (1) Tag=0000 00, Index=00 000, Offset=0 0000, (2) Miss, (3) None

04 -> 0000 0000 0000 0100 -> (1) Tag=0000 00, Index=00 000, Offset=0 0100, (2) Hit, (3) None

10 -> 0000 0000 0001 0000 -> (1) Tag=0000 00, Index=00 000, Offset=1 0000, (2) Hit, (3) None

84 -> 0000 0000 1000 0100 -> (1) Tag=0000 00, Index=00 100, Offset=0 0100, (2) Miss, (3) None

E8 -> 0000 0000 1110 1000 -> (1) Tag=0000 00, Index=00 111, Offset=0 1000, (2) Miss, (3) None

A0 -> 0000 0000 1010 0000 -> (1) Tag=0000 00, Index=00 101, Offset=0 0000, (2) Miss, (3) None

400 -> 0000 0100 0000 0000 -> (1) Tag=0000 01, Index=00 000, Offset=0 0000, (2) Miss, (3) Replace 0th Index

1E -> 0000 0000 0001 1110 -> (1) Tag=0000 00, Index=00 000, Offset=1 1110, (2) Miss, (3) Replace 0th Index

8C -> 0000 0000 1000 1100 -> (1) Tag=0000 00, Index=00 100, Offset=0 1100, (2) Hit, (3) None

C1C -> 0000 1100 0001 1100 -> (1) Tag=0000 11, Index=00 000, Offset=1 1100, (2) Miss, (3) Replace 0th Index

B4 -> 0000 0000 1011 0100 -> (1) Tag=0000 00, Index=00 101, Offset=1 0100, (2) Hit, (3) None

884 -> 0000 1000 1000 0100 -> (1) Tag=0000 10, Index=00 100, Offset=0 0100, (2) Miss, (3) Replace 4th Index

5.5.5[5] <§5.3> What is the hit ratio?

Solution:

Given total access as 12, total hit as 4, and total miss as 8.

Determine hit ratio=4/12=0.333.

5.5.6[5] <§5.3> 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]>

Solution:

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

<4, 2, Mem[0x880]-Mem[0x89F]>

<5, 0, Mem[0xA0]-Mem[0xBF]>

<7, 0, Mem[0xE0]-Mem[0xFF]>

5.7 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

5.7.1[10] <§§5.3, 5.8> 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.)

Solution:

One in four instructions is a data read, one in ten instructions is a data write. For a CPI of 2, there are 0.5 instruction accesses per cycle, 12.5% of cycles will require a data read, and 5% of cycles will require a data write.

The instruction bandwidth is thus (0.0030 (miss/instruction) * 64 (bytes/miss)) * 0.5 (instruction/cycle) = 0.096 bytes/cycle.

The data read bandwidth is thus 0.02 (miss/access) (0.13 (reads/cycle)+0.050 (writes/cycle)) * 64 (bytes/miss) = 0.23 bytes/cycle.

The total read bandwidth requirement is 0.33 bytes/cycle. (instruction bw + data bw)

The data write bandwidth requirement is 0.05 (writes/cycle) * 4 (bytes/write) = 0.2 bytes/cycle.

5.7.2[10] <§§5.3, 5.8> 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?

Solution:

The instruction and data read bandwidth requirement is the same as in 5.7.1. The data write bandwidth requirement becomes 0.02 (miss/access) * 0.30 (writes/miss) * (0.13+0.050) (access/cycle) * 64 (bytes/write)= 0.069 bytes/cycle.

5.9 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%

5.9.1[10] <§5.3> What is the optimal block size for a miss latency of 20 × B cycles?

Solution:

B=8 0.04*20*8=6.4

B=16 0.03*20*16=9.6

B=32 0.02*20*32=12.8

B=64 0.015*20*64=19.2

B=128 0.01*20*128=25.6

B=8 is optimal.

5.9.2[10] <§5.3> What is the optimal block size for a miss latency of 24 + B cycles?

Solution:

B=8 0.04*(24+8)=1.28

B=16 0.03*(24+16)=1.2

B=32 0.02*(24+32)=1.12

B=64 0.015*(24+64)=1.32

B=128 0.01*(24+128)=1.52

B=32 is optimal.

5.9.3[10] <§5.3> For constant miss latency, what is the optimal block size?

Solution:

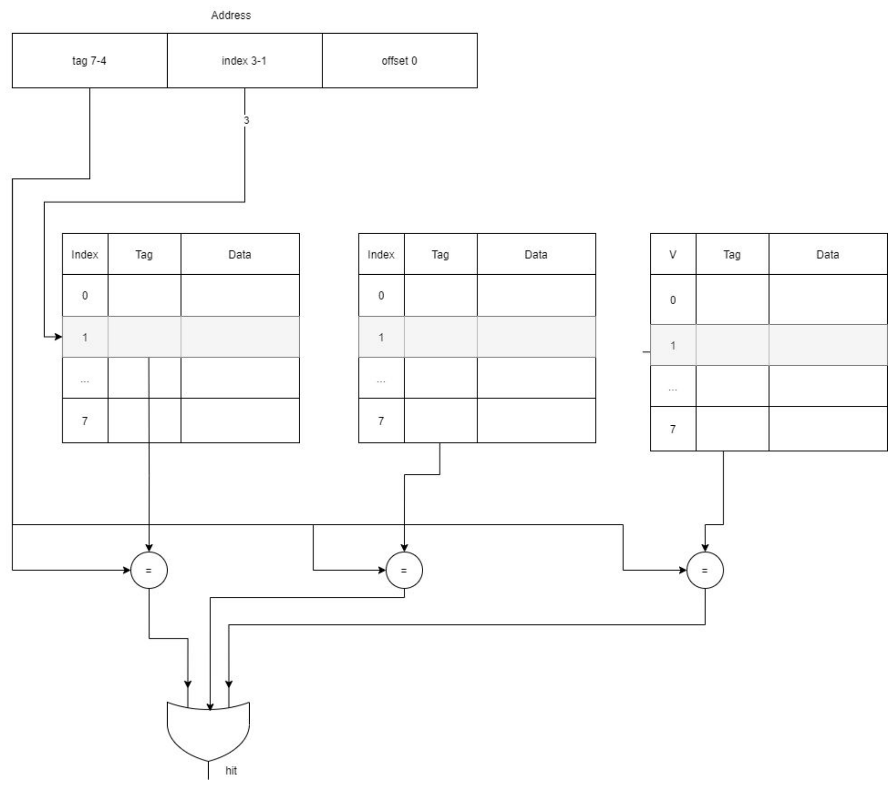
B=128

5.11This 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

5.11.1[10] <§5.4> 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.

Will have three tables instead of four as in figure 5.18. Index goes from 0 to 7 (Explanation: (1) in 3-way set associative cache, each set contains 3 blocks (i.e. cache lines), and (2) we have 2-word blocks in the problem. Thus, the number of sets is 48/3/2=8). Index width is 3 (log 8=3). Tag width is 4 (8-1-3=4)



5.11.2[10] <§5.4> Trace the behavior of the cache from Exercise 5.11.1. 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.

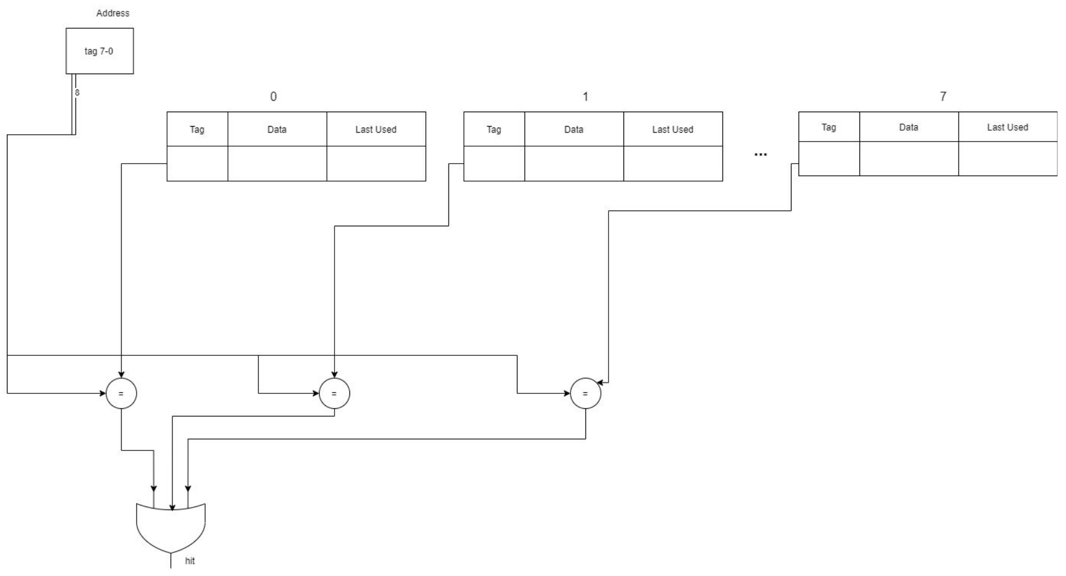
(r) means "replaced" in the table below.

Hex	Binary	Tag	Index	Offset	hit/miss	Tag 1	Tag 2	Tag 3
0x03	0000_0011	0000	001	1	Miss	0000		
0xb4	1011_0100	1011	010	0	Miss	1011		
0x2b	0010_1011	0010	101	1	Miss	0010		
0x02	0000_0010	0000	001	0	Hit	0000		
0xbe	1011_1110	1011	111	0	Miss	1011		
0x58	0101_1000	0101	100	0	Miss	0101		
0xbf	1011_1111	1011	111	1	Hit	1011		
0x0e	0000_1110	0000	111	0	Miss	1011	0000	
0x1f	0001_1111	0001	111	1	Miss	1011	0000	0001
0xb5	1011_0101	1011	010	1	Hit	1011		
0xbf	1011_1111	1011	111	1	Hit	1011	0000	0001
0xba	1011_101	101	101	0	Miss	0010	1011	

	0	1						
0x2e	0010_1110	0010	111	0	Miss	1011	0010 (r)	0001
0xce	1100_1110	1100	111	0	Miss	1011	0010 (r)	1100 (r)

5.11.3[5] <§5.4> 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.

Will have 8 sets (tables) instead of four as in figure 5.18. Index goes from 0 to 0 (no index is needed for fully associative). Index width is 0. Tag width is 8 (the whole address)



5.11.4[10] <§5.4> Trace the behavior of the cache from Exercise 5.11.3. 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, and
- ■the contents of the cache after each reference has been handled

(r) means "replaced" in the table below.

Hex	Binary	Tag	Index	Offset	hit/miss	Cache Content
0x03	0000_0011	0000_0111	N/A	N/A	Miss	0000_0011 xxxx_xxxx xxxx_xxxx xxxx_xxxx xxxx_xxxx xxxx_xxxx xxxx_xxxx xxxx_xxxx
0xb4	1011_0100	1011_0100	N/A	N/A	Miss	0000_0011 1011_0100 xxxx_xxxx xxxx_xxxx xxxx_xxxx xxxx_xxxx xxxx_xxxx xxxx_xxxx
0x2b	0010_1011	0010_1011	N/A	N/A	Miss	0000_0011 1011_0100 0010_1011 xxxx_xxxx xxxx_xxxx xxxx_xxxx xxxx_xxxx xxxx_xxxx
0x02	0000_0010	0000_0010	N/A	N/A	Miss	0000_0011 1011_0100 0010_1011 0000_0010 xxxx_xxxx xxxx_xxxx xxxx_xxxx xxxx_xxxx
0xbe	1011_1110	1011_1110	N/A	N/A	Miss	0000_0011 1011_0100 0010_1011 0000_0010 1011_1110 xxxx_xxxx xxxx_xxxx

						xxxx_xxxx
0x58	0101_1000	0101_1000	N/A	N/A	Miss	0000_0011 1011_0100 0010_1011 0000_0010 1011_1110 0101_1000 xxxx_xxxx xxxx_xxxx
0xbf	1011_1111	1011_1111	N/A	N/A	Miss	0000_0011 1011_0100 0010_1011 0000_0010 1011_1110 0101_1000 1011_1111 xxxx_xxxx
0x0e	0000_1110	0000_1110	N/A	N/A	Miss	0000_0011 1011_0100 0010_1011 0000_0010 1011_1110 0101_1000 1011_1111 0000_1110
0x1f	0001_1111	0001_1111	N/A	N/A	Miss	0001_1111 (r) 1011_0100 0010_1011 0000_0010 1011_1110 0101_1000 1011_1111 0000_1110
0xb5	1011_0101	1011_0101	N/A	N/A	Miss	0001_1111 1011_0101 (r) 0010_1011 0000_0010 1011_1110 0101_1000 1011_1111 0000_1110
0xbf	1011_1111	1011_1111	N/A	N/A	Hit	0001_1111

						1011_0101 0010_1011 0000_0010 1011_1110 0101_1000 1011_1111 0000_1110
0xba	1011_1010	1011_1010	N/A	N/A	Miss	0001_1111 1011_0101 1011_1010 (r) 0000_0010 1011_1110 0101_1000 1011_1111 0000_1110
0x2e	0010_1110	0010_1110	N/A	N/A	Miss	0001_1111 1011_0101 1011_1010 0010_1110 (r) 1011_1110 0101_1000 1011_1111 0000_1110
0xce	1100_1110	1100_1110	N/A	N/A	Miss	0001_1111 1011_0101 1011_1010 0010_1110 1100_1110 (r) 0101_1000 1011_1111 0000_1110