HW5

5.1

5.1.1 $\frac{16\text{byte}}{64\text{bit}} = \frac{16\times 8\text{bit}}{64\text{bit}} = 2$

5.1.2 I, J, B[I][0] exhibit temporal locality

5.1.3 A[I][J] exhibit spatial locality

5.1.4 I, J, B[][0] exhibit temporal locality

5.1.5 A(J,1) exhibit spatial locality

5.1.6 16 byte block holds 2 words

for matrix A, whether using C or Matlab to store,

the size is 8000×8000, thus the block num $= \frac{8000\times 8000}{2} = 3.2\times 10^7$

for matrix B,

using Matlab's matrix storage: B(I,0) is contiguous in memory

for every two I, B(I,0) holds a new block

thus the block num = 4

using C's matrix storage: B(I,0) is not contiguous in memory

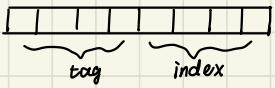
for each I, B(I,0) holds a new block

thus the block num = 8

5.2

5.2.1 cache performance: direct-mapped, 16 blocks → 4 bit index

block size = 1 word → block offset = 0 bit



hex addr	binary addr	tag	index	hit/miss
0x03	00000011	0000	0011	miss
0xb4	1011 0100	1011	0100	miss
0x2b	0010 1011	0010	1011	miss
0x02	0000 0010	0000	0010	miss
0xbf	1011 1111	1011	1111	miss
0x58	0101 1000	0101	1000	miss
0xbe	1011 1110	1011	1110	miss
0x0e	0000 1110	0000	1110	miss
0xb5	1011 0101	1011	0101	miss
0x2c	0010 1100	0010	1100	miss
0xba	1011 1010	1011	1010	miss
0xfd	1111 1101	1111	1101	miss

5.2.2 cache performance: direct-mapped, 8 blocks → 3 bit index

block size = 2 word → block offset = 1 bit

hex addr	binary addr	tag	index	offset	hit/miss
0x03	00000011	0000	001	1	miss
0xb4	1011 0100	1011	010	0	miss
0x2b	0010 1011	0010	101	1	miss
0x02	0000 0010	0000	001	0	hit
0xbf	1011 1111	1011	111	1	miss
0x58	0101 1000	0101	100	0	miss
0xbe	1011 1110	1011	111	0	hit
0x0e	0000 1110	0000	111	0	miss
0xb5	1011 0101	1011	010	1	hit
0x2c	0010 1100	0010	110	0	miss
0xba	1011 1010	1011	101	0	miss
0xfd	1111 1101	1111	110	1	miss

5.2.3

C_1 has the highest miss rate with 1-word block size.
Because every word address is different from each other,
one-word block size loses spatial locality badly.

consider C_2 and C_3

C_2 block size = 2 words → block offset = 1 bit
block num = 8/2 = 4 → index = 2 bit
tag = 8 − 3 = 5 bit

C_3 block size = 4 words → offset = 2 bit
block num = 8/4 = 2 → index = 1 bit
tag = 5 bit

			C_2			C_3		
hex addr	binary addr	tag	index	offset	hit/miss	index	offset	hit/miss
0x03	00000011	00000	01	1	miss	0	11	miss
0xb4	1011 0100	10110	10	0	miss	1	00	miss
0x2b	0010 1011	00101	01	1	miss	0	11	miss
0x02	0000 0010	00000	01	0	miss	0	10	miss
0xbf	1011 1111	10111	11	1	miss	1	11	miss
0x58	0101 1000	01011	00	0	miss	0	00	miss
0xbe	10111110	10111	11	0	hit	1	10	hit
0x0e	00001110	00001	11	0	miss	1	10	miss
0xb5	1011 0101	10110	10	1	hit	1	01	miss
0x2c	0010 1100	00101	10	0	miss	1	00	miss
0xba	1011 1010	10111	01	0	miss	0	10	miss
0xfd	1111 1101	11111	10	1	miss	1	01	miss

C_2 hit rate > C_3 hit rate
∴ C_2 is the optimized design for the given references.

5.3

5.3.1 cache data size = 32KB = $32 \times 2^{10} \times 2^3$ bit

block size = 2 words = 2 × 8 bytes → block offset = 4 bit

block num = $\frac{32 \times 2^{10} \times 2^3 \text{ bit}}{2 \times 64 \text{ bit}} = 2^{11}$ → index = 11 bit

tag = 64 − 11 − 4 = 49 bit

valid = 1 bit

∴ total size = (valid + tag + block size) × block num

= (1 + 49 + 128) × 2^{11} bit

= 364544 bit

cache data size = 64KB = 2^{19} bit

5.3.2 block size = 16 words = $2^4 \times 2^3$ bytes → block offset = 7 bit

block num = $\frac{2^{19} \text{ bit}}{2^{10} \text{ bit}} = 2^9$ bit → index = 9 bit

tag = 64 − 7 − 9 = 48 bit

valid = 1 bit

total size = (valid + tag + block size) × block num

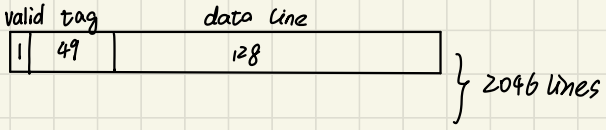
= (1 + 48 + 2^{10}) × 2^9 bit

= 549376 bit

5.3.3 ① block size is larger, which means a larger multiplexer is needed to select a specified byte from one block.

② number of blocks is smaller, which means the replacement of blocks may be more frequent, increasing miss rate.

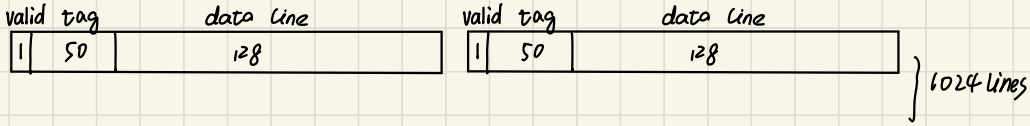
5.3.4 in 5.3.1, the cache is direct-mapped, cache data size = 32KB, block size = 2 words



in 5.3.4, the cache is 2-way set associated, cache data size = 32KB, block size = 2 words

block num = 2^{11}, index = $\frac{2^{11}}{2} = 2^{10}$ bit

tag = 64 − 4 − 10 = 50 bit



Read requests series example:

always keep the index the same, but change tag

address:
tag = 0, index = 0
tag = 1, index = 0
tag = 0, index = 0
tag = 1, index = 0
...

we will see the direct-mapped cache always misses
but the 2-way set associated cache always hit (except first two)

thus the 2-way set associated cache has a lower miss rate

5.5 assume 1 word = 64 bit

5.5.1 cache block size = 2^5 bit = 0.5 word

5.5.2 block number = $2^5 = 32$

5.5.3 cache data size = block size x block number = $2^5 \times 2^5 = 2^{10}$ bit

total cache size = (valid + tag + block size) x block number
= $(1 + 54 + 32) \times 2^5$ bit

$$\text{ratio} = \frac{87 \times 2^5 \text{ bit}}{32 \times 2^5 \text{ bit}} = 2.72$$

5.5.4

Hex Addr	Binary Addr	tag	index	offset	hit/miss	replaced bytes
00	00000000	0	00000	00000	m	
04	00000100	0	00000	00100	h	
10	00010000	0	00000	10000	h	
84	10000100	0	00100	00100	m	
E8	11101000	0	00111	01000	m	
A0	10100000	0	00101	00000	m	
400	010000000000	1	00000	00000	m	Mem[x0] - Mem[x1F]
1E	00011110	0	00000	11110	m	Mem[x400] - Mem[x41F]
8C	10001100	0	00100	01100	h	
C1C	110000011100	11	00000	11100	m	Mem[x0] - Mem[x1F]
B4	10110100	0	00101	10100	h	
884	100010000100	10	00100	00100	m	Mem[x80] - Mem[x9F]

5.5.5 hit ratio $= \frac{4}{12} = 33.3\%$

5.5.6 <0, 3, Mem[xC00] - Mem[xC1F]>

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

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

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

5.6

5.6.1 between L1 and L2: need a small buffer, such as a few words

between L2 and memory: need a bigger buffer, at least one block.

5.6.2 handling an L1 write-miss:

if the target block is in L2,

write around the data in L2

else if L2 is full, write back a dirty block to memory

read the target block from memory to L2

write the data in L2

5.6.3 handling an L1 write-miss:

if the target block is in L2,

write around the data in L2

else if L2 is full, write back a dirty block to memory

read the target block from memory to L2

write the data in L2

handling an L1 read-miss:

if the target block is in L2

if L1 is full, use replace strategy to replace one block

read block to L1 from L2

else

if L2 is full, write back one dirty block to memory

read block to L2 from memory

if L1 is full, use replace strategy to replace one block

read block to L1 from L2