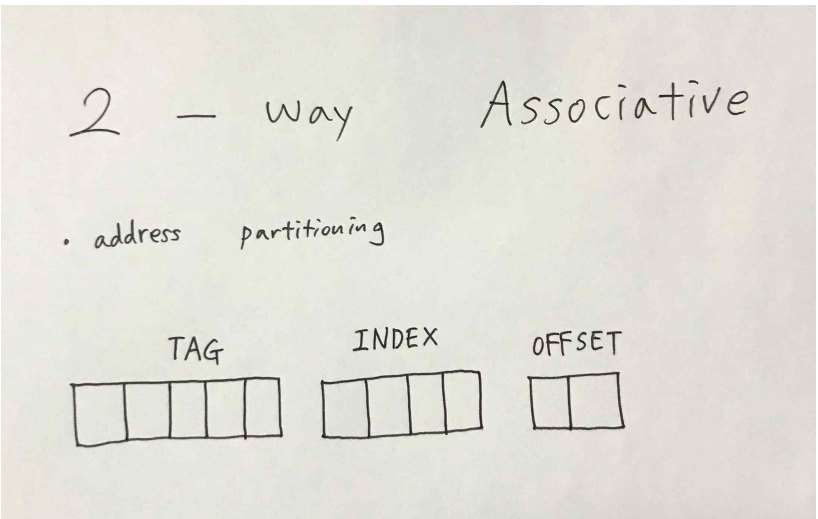
Computer Architecture

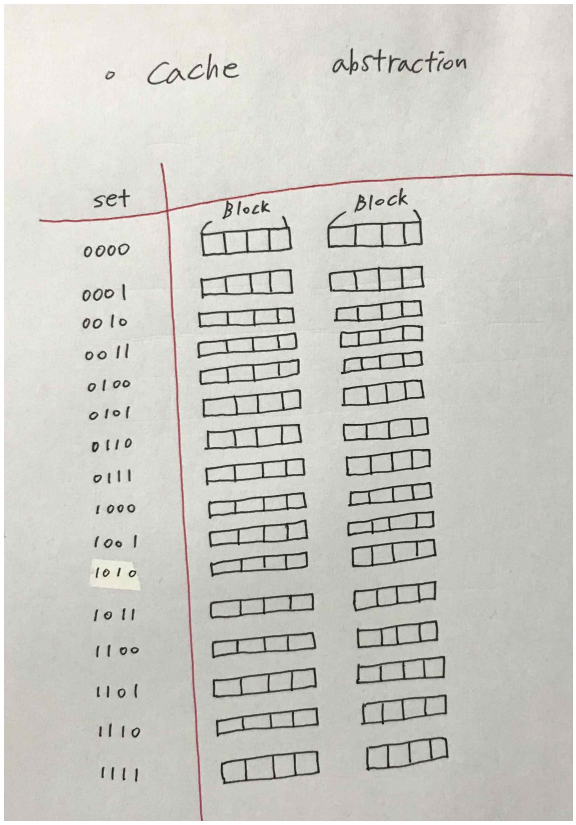
Lab 3

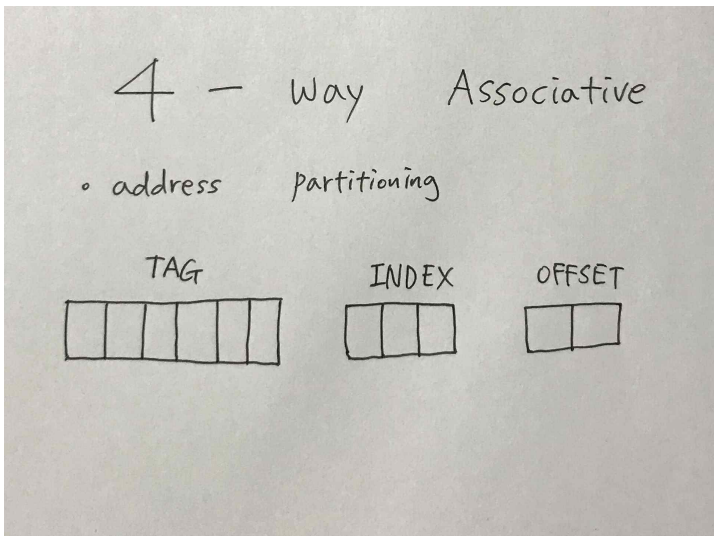
Lab3 : Cache Simulation

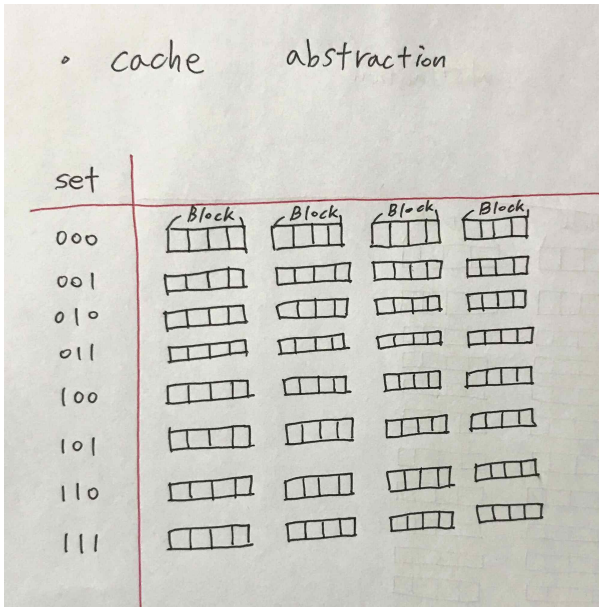
학번 : 2016707079 이름 : 하상천

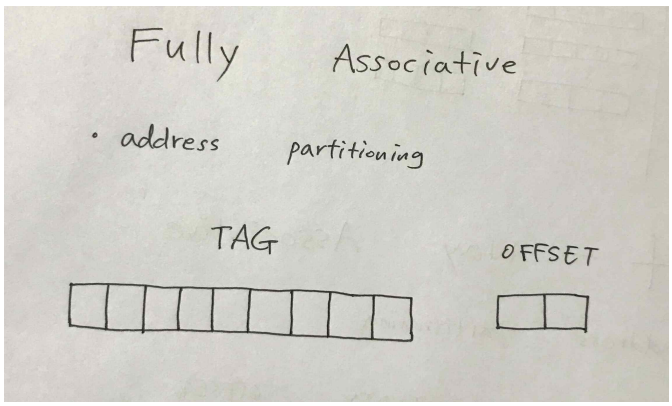
A. Draw the visualization of address partitioning and cache abstraction.

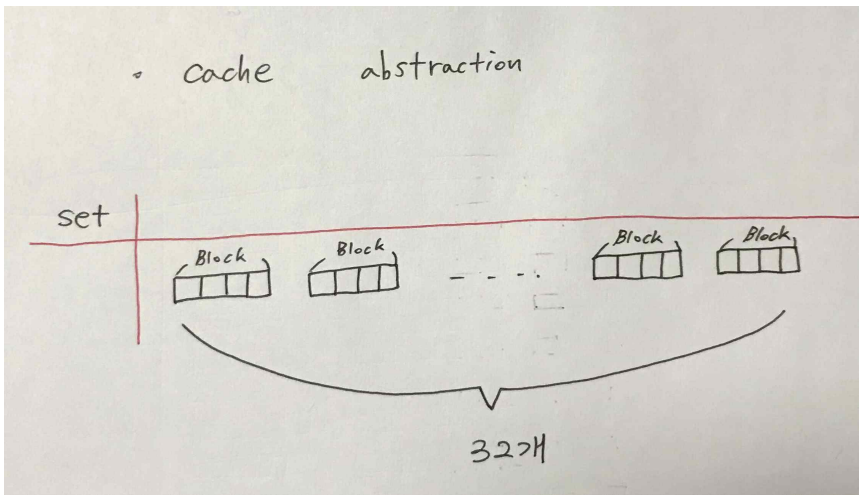



Cache abstraction
set
Block
Block
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111


4 - way Associative
address partitioning
TAG
INDEX
OFFSET


• cache abstraction
set
Block
Block
Block
Block
000
001
010
011
100
101
110
111


Fully Associative
• address partitioning
TAG
OFFSET


• cache abstraction
set
Block
Block
Block
Block
32개

B. Show the simulation command and also the output.

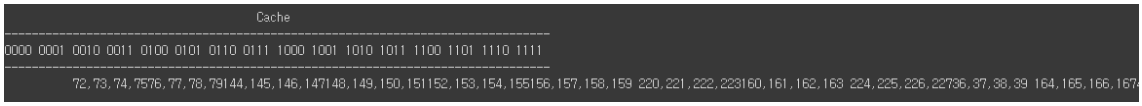
- 2-Way Associative

Simulation command :

!python cachesimulator --cache-size 128 --num-blocks-per-set 2 --num-words-per-block 4 --replacement-policy lru --num-addr-bits 11 --word-addrs 52 56 36 40 44 48 72 76 144 148 152 156 160 164 168 172 176 180 184 220 224 228 184 188 144 148 152 156 160 164 168 172 52 56 36 40 44 48 72 76

Output :

WordAddr	BinAddr	Tag	Index	Offset	Hit/Miss
5200000	110 100	00000	1101	00	miss
5600000	111 000	00000	1110	00	miss
3600000	100 100	00000	1001	00	miss
4000000	101 000	00000	1010	00	miss
4400000	101 100	00000	1011	00	miss
4800000	110 000	00000	1100	00	miss
7200001	001 000	00001	0010	00	miss
7600001	001 100	00001	0011	00	miss
14400010	010 000	00010	0100	00	miss
14800010	010 100	00010	0101	00	miss
15200010	011 000	00010	0110	00	miss
15600010	011 100	00010	0111	00	miss
16000010	100 000	00010	1000	00	miss
16400010	100 100	00010	1001	00	miss
16800010	101 000	00010	1010	00	miss
17200010	101 100	00010	1011	00	miss
17600010	110 000	00010	1100	00	miss
18000010	110 100	00010	1101	00	miss
18400010	111 000	00010	1110	00	miss
22000011	011 100	00011	0111	00	miss
22400011	100 000	00011	1000	00	miss
22800011	100 100	00011	1001	00	miss
18400010	111 000	00010	1110	00	HIT
18800010	111 100	00010	1111	00	miss
14400010	010 000	00010	0100	00	HIT
14800010	010 100	00010	0101	00	HIT
15200010	011 000	00010	0110	00	HIT
15600010	011 100	00010	0111	00	HIT
16000010	100 000	00010	1000	00	HIT
16400010	100 100	00010	1001	00	HIT
16800010	101 000	00010	1010	00	HIT
17200010	101 100	00010	1011	00	HIT
5200000	110 100	00000	1101	00	HIT
5600000	111 000	00000	1110	00	HIT
3600000	100 100	00000	1001	00	miss
4000000	101 000	00000	1010	00	HIT
4400000	101 100	00000	1011	00	HIT
4800000	110 000	00000	1100	00	HIT
7200001	001 000	00001	0010	00	HIT
7600001	001 100	00001	0011	00	HIT



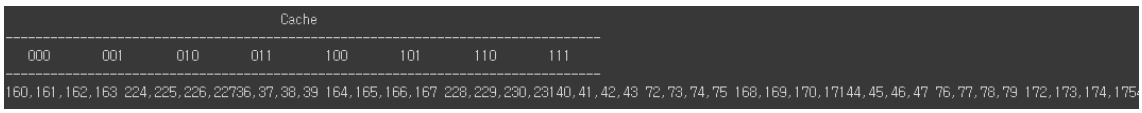
(잘 안보여서 나눠서 캡쳐 했습니다.)

- 4-Way Associative

Simulation command :
!python cachesimulator --cache-size 128 --num-blocks-per-set 4 --num-words-per-block 4 --replacement-policy lru --num-addr-bits 11 --word-addrs 52 56 36 40 44 48 72 76 144 148 152 156 160 164 168 172 176 180 184 220 224 228 184 188 144 148 152 156 160 164 168 172 52 56 36 40 44 48 72 76

Output :

WordAddr	BinAddr	Tag	Index	Offset	Hit/Miss
5200000	110 100	000 001	101	00	miss
5600000	111 000	000 001	110	00	miss
3600000	100 100	000 001	001	00	miss
4000000	101 000	000 001	010	00	miss
4400000	101 100	000 001	011	00	miss
4800000	110 000	000 001	100	00	miss
7200001	001 000	000 010	010	00	miss
7600001	001 100	000 010	011	00	miss
14400010	010 000	000 100	100	00	miss
14800010	010 100	000 100	101	00	miss
15200010	011 000	000 100	110	00	miss
15600010	011 100	000 100	111	00	miss
16000010	100 000	000 101	000	00	miss
16400010	100 100	000 101	001	00	miss
16800010	101 000	000 101	010	00	miss
17200010	101 100	000 101	011	00	miss
17600010	110 000	000 101	100	00	miss
18000010	110 100	000 101	101	00	miss
18400010	111 000	000 101	110	00	miss
22000011	011 100	000 110	111	00	miss
22400011	100 000	000 111	000	00	miss
22800011	100 100	000 111	001	00	miss
18400010	111 000	000 101	110	00	HIT
18800010	111 100	000 101	111	00	miss
14400010	010 000	000 100	100	00	HIT
14800010	010 100	000 100	101	00	HIT
15200010	011 000	000 100	110	00	HIT
15600010	011 100	000 100	111	00	HIT
16000010	100 000	000 101	000	00	HIT
16400010	100 100	000 101	001	00	HIT
16800010	101 000	000 101	010	00	HIT
17200010	101 100	000 101	011	00	HIT
5200000	110 100	000 001	101	00	HIT
5600000	111 000	000 001	110	00	HIT
3600000	100 100	000 001	001	00	HIT
4000000	101 000	000 001	010	00	HIT
4400000	101 100	000 001	011	00	HIT
4800000	110 000	000 001	100	00	HIT
7200001	001 000	000 010	010	00	HIT
7600001	001 100	000 010	011	00	HIT



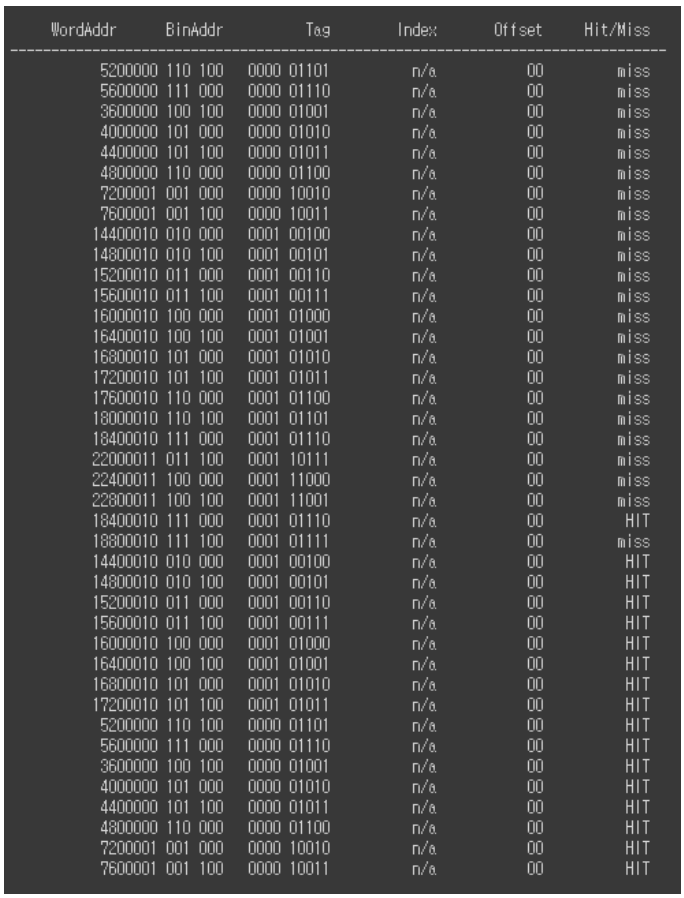
(잘 안보여서 나눠서 캡쳐 했습니다.)

- Fully Associative

Simulation command :

!python cachesimulator --cache-size 128 --num-blocks-per-set 32 --num-words-per-block 4 --replacement-policy lru --num-addr-bits 11 --word-addrs 52 56 36 40 44 48 72 76 144 148 152 156 160 164 168 172 176 180 184 220 224 228 184 188 144 148 152 156 160 164 168 172 52 56 36 40 44 48 72 76

Output :



WordAddr	BinAddr	Tag	Index	Offset	Hit/Miss
5200000	110 100	0000 01101	n/a	00	miss
5600000	111 000	0000 01110	n/a	00	miss
3600000	100 100	0000 01001	n/a	00	miss
4000000	101 000	0000 01010	n/a	00	miss
4400000	101 100	0000 01011	n/a	00	miss
4800000	110 000	0000 01100	n/a	00	miss
7200001	001 000	0000 10010	n/a	00	miss
7600001	001 100	0000 10011	n/a	00	miss
14400010	010 000	0001 00100	n/a	00	miss
14800010	010 100	0001 00101	n/a	00	miss
15200010	011 000	0001 00110	n/a	00	miss
15600010	011 100	0001 00111	n/a	00	miss
16000010	100 000	0001 01000	n/a	00	miss
16400010	100 100	0001 01001	n/a	00	miss
16800010	101 000	0001 01010	n/a	00	miss
17200010	101 100	0001 01011	n/a	00	miss
17600010	110 000	0001 01100	n/a	00	miss
18000010	110 100	0001 01101	n/a	00	miss
18400010	111 000	0001 01110	n/a	00	miss
22000011	011 100	0001 10111	n/a	00	miss
22400011	100 000	0001 11000	n/a	00	miss
22800011	100 100	0001 11001	n/a	00	miss
18400010	111 000	0001 01110	n/a	00	HIT
18800010	111 100	0001 01111	n/a	00	miss
14400010	010 000	0001 00100	n/a	00	HIT
14800010	010 100	0001 00101	n/a	00	HIT
15200010	011 000	0001 00110	n/a	00	HIT
15600010	011 100	0001 00111	n/a	00	HIT
16000010	100 000	0001 01000	n/a	00	HIT
16400010	100 100	0001 01001	n/a	00	HIT
16800010	101 000	0001 01010	n/a	00	HIT
17200010	101 100	0001 01011	n/a	00	HIT
5200000	110 100	0000 01101	n/a	00	HIT
5600000	111 000	0000 01110	n/a	00	HIT
3600000	100 100	0000 01001	n/a	00	HIT
4000000	101 000	0000 01010	n/a	00	HIT
4400000	101 100	0000 01011	n/a	00	HIT
4800000	110 000	0000 01100	n/a	00	HIT
7200001	001 000	0000 10010	n/a	00	HIT
7600001	001 100	0000 10011	n/a	00	HIT

(잘 안보여서 나눠서 캡쳐 했습니다.)

C. Compute the Hit Ratio and compare it to the direct mapping hit ratio shown in section B-8.

- 2-Way Associative
Hit 16개, Miss 24개로
Hit Ratio = 16 / 40 = 0.4 이다.

- 4-Way Associative
Hit 17개, Miss 23개로
Hit Ratio = 17 / 40 = 0.425 이다.

- Fully Associative
Hit 17개, Miss 23개로
Hit Ratio = 17 / 40 = 0.425 이다.

Direct Mapping hit ratio는 Hit 11개, Miss 29개로 11 / 40 = 0.275이다. 2-Way Associative, 4-Way Associative, Fully Associative와 비교 했을 때 hit ratio가 제일 작은 것을 확인 할 수 있다. 그 다음으로 2-Way Associative의 hit ratio가 작은 이유는 뒤에서 6번째에 miss가 발생하기 때문이다. cache 1001에 36, 37, 38, 39 가 들어오고 그 다음으로 164, 165, 166, 167이 들어오는데 set이 꽉 찼기 때문에 가장 긴 시간동안 사용되지 않은 36, 37, 38, 39가 지워지고 228, 229, 230, 231이 들어온다. 다음으로 164는 있기 때문에 hit 이고 36은 없기 때문에 가장 긴 시간동안 사용되지 않은 228, 229, 230, 231이 지워지고 36, 37, 38, 39가 들어온다. 따라서 뒤에서 6번째에 miss가 발생하고, hit ratio는 작아진다. Direct Mapping도 마찬가지 방법으로 index가 겹쳐서 지워지고 다시 또 들어오는 방식을 반복해서 miss가 더 많아진 것이다. 다른 방법보다 index가 더 많이 겹치는 이유는 blocks per set이 1로 제일 작기 때문이다.

D. Explain the result of each simulation.

- 2-Way Associative

Memory size = 2^11 이고, Block size = 4W = 2^2 이다. Cache 안에 있는 set의 수는 cache size를 (set size * block size)로 나눈 것 이므로 2^7 / (2^1 * 2^2) = 2^4 이다.
Tag 비트 수는 전체 비트 수에서 index 비트와 offset 비트를 뺀 것이므로
11 - 4 - 2 = 5 비트 이다. 따라서 순서대로 tag 5비트, index 4비트, offset 2비트이다.
16개의 set은 index 4비트 0000부터 1111로 구분한다. Block offset이 2비트인 이유는

block당 word 수가 4이기 때문에 4개씩 가져오기 때문이다. 4개씩 가져오면 전체 비트의 lsb와 그 앞 비트가 00, 01, 10, 11 이어도 index가 같기 때문에 block offset이 2비트인 것이다. 만약에 block 당 word 수가 2이면 전체 비트에서 lsb가 0, 1 이어도 index가 같기 때문에 offset은 1비트가 된다. 같은 이유로 block 당 word 수가 8이면 offset은 3비트, block 당 word 수가 16이면 offset은 4비트가 된다. 또한 문제에서 word addresses로 주어졌기 때문에 byte offset은 없다. 하지만 byte addresses로 주어졌다면 전체 비트의 lsb쪽부터 byte offset이고 그 다음 block offset, index, tag가 될 것이라고 생각한다.
처음에는 cache에 data가 없기 때문에 compulsory misses가 발생한다. 이러한 문제를 줄이려면 block size를 늘리면 된다. 또한 conflict misses를 줄이려면 하나의 set 당 block 수를 늘리면 된다. set 당 block 수가 2개이기 때문에 index가 겹쳐도 지우지 않고 2개의 block에 나눠서 위치할 수 있다. 하지만 세 번째로 index가 같다면 가장 긴 시간동안 사용 되지 않은 word address를 지우고, main memory에서 가져와서 저장해야한다. 왜냐하면 replacement policy가 lru(least-recently used) 이기 때문이다. 일반적으로 2-way는 하나의 set당 2개의 같은 index를 가지는 block이 위치 할 수 있고, 이들은 tag를 통해 구분된다. 따라서 일반적으로 Direct Mapping 보다는 hit ratio가 더 크고, 4-Way Associative와 fully Associative 보다는 hit ratio가 더 작다.

- 4-Way Associative

Memory size = 2^11 이고, Block size = 4W = 2^2 이다. Cache 안에 있는 set의 수는 cache size를 (set size * block size)로 나눈 것 이므로 2^7 / (2^2 * 2^2) = 2^3 이다.
Tag 비트 수는 전체 비트 수에서 index 비트와 offset 비트를 뺀 것이므로
11 - 3 - 2 = 6 비트 이다. 따라서 순서대로 tag 6비트, index 3비트, offset 2비트이다.
8개의 set은 index 3비트 000부터 111로 구분한다. set 당 block 수가 4개이기 때문에 index가 겹쳐도 지우지 않고 4개의 block에 나눠서 위치할 수 있다. 하지만 다섯 번째로 index가 같다면 가장 긴 시간동안 사용 되지 않은 word address를 지우고, main memory에서 가져와서 저장해야한다. 왜냐하면 replacement policy가 lru(least-recently used) 이기 때문이다. 일반적으로 4-way는 하나의 set당 4개의 같은 index를 가지는 block이 위치 할 수 있고, 이들은 tag를 통해 구분된다. 따라서 일반적으로 Direct Mapping, 2-Way Associative 보다는 hit ratio가 더 크고, fully Associative 보다는 hit ratio가 더 작다.

- Fully Associative

Memory size = 2^11 이고, Block size = 4W = 2^2 이다. Cache 안에 있는 set의 수는 cache size를 (set size * block size)로 나눈 것 이므로 2^7 / (2^5 * 2^2) = 2^0 이다.
Tag 비트 수는 전체 비트 수에서 index 비트와 offset 비트를 뺀 것이므로
11 - 0 - 2 = 9 비트 이다. 따라서 순서대로 tag 9비트, index 0비트, offset 2비트이다.
set 당 block 수가 32개이기 때문에 32개의 block에 나눠서 위치할 수 있다. 하지만 33번째가 되면 가장 긴 시간동안 사용 되지 않은 address를 지우고, main memory에서 가져와서 저장해야한다. 왜냐하면 replacement policy가 lru(least-recently used) 이기 때문이다. 일반적으로 fully는 하나의 set에 모든 block이 위치하고, 이들은 tag를 통해 구분된다. 따라서 일반적으로 Direct Mapping, 2-Way Associative, 4-Way Associative

보다는 hit ratio가 더 크다. 하지만 하나의 set에 모든 block이 위치하고 있기 때문에 cpu가 찾는 시간이 오래 걸린다.