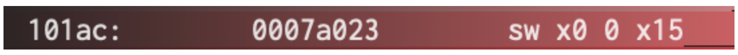
108062213 顏浩昀
Computer Architecture
2022.06.12

1.

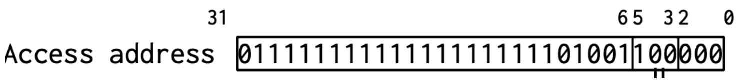
a).

(i) load/store instruction



```
101ac:          0007a023          sw x0 0 x15
```

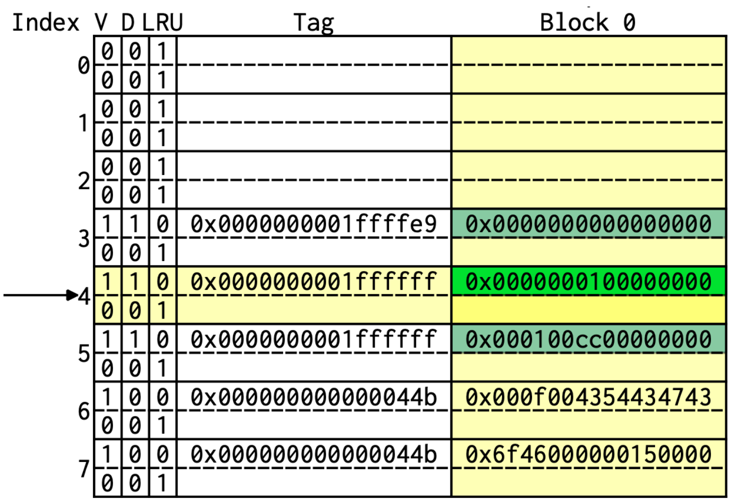
(ii) Access address



```
                31                              65  32   0
Access address  01111111111111111111101001|100|000
```

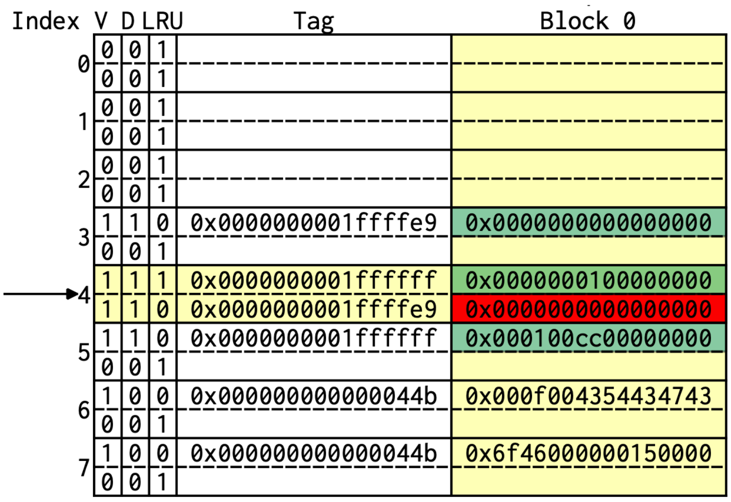
(iii)

Cache states before



Index	V	D	LRU	Tag	Block 0
0	0	0	1		
	0	0	1		
1	0	0	1		
	0	0	1		
2	0	0	1		
	0	0	1		
3	1	1	0	0x0000000001ffffe9	0x0000000000000000
	0	0	1		
→4	1	1	0	0x0000000001ffffff	0x0000000100000000
	0	0	1		
5	1	1	0	0x0000000001ffffff	0x000100cc00000000
	0	0	1		
6	1	0	0	0x000000000000044b	0x000f004354434743
	0	0	1		
7	1	0	0	0x000000000000044b	0x6f46000000150000
	0	0	1		

Cache state after

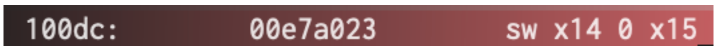


Index	V	D	LRU	Tag	Block 0
0	0	0	1		
	0	0	1		
1	0	0	1		
	0	0	1		
2	0	0	1		
	0	0	1		
3	1	1	0	0x0000000001ffffe9	0x0000000000000000
	0	0	1		
→4	1	1	1	0x0000000001ffffff	0x0000000100000000
	1	1	0	0x0000000001ffffe9	0x0000000000000000
5	1	1	0	0x0000000001ffffff	0x000100cc00000000
	0	0	1		
6	1	0	0	0x000000000000044b	0x000f004354434743
	0	0	1		
7	1	0	0	0x000000000000044b	0x6f46000000150000
	0	0	1		

Because of miss, we need to fetch a new block into cache. Using the block address to calculate index is 0b100 = 4 and there is a empty(invalid) way in 4th set. Hence we map the block into it after a write miss and let valid bit and dirty bit become 1 LRU bit become 0.

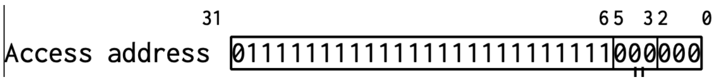
b).

(i) load/store instruction



```
100dc:        00e7a023        sw x14 0 x15
```

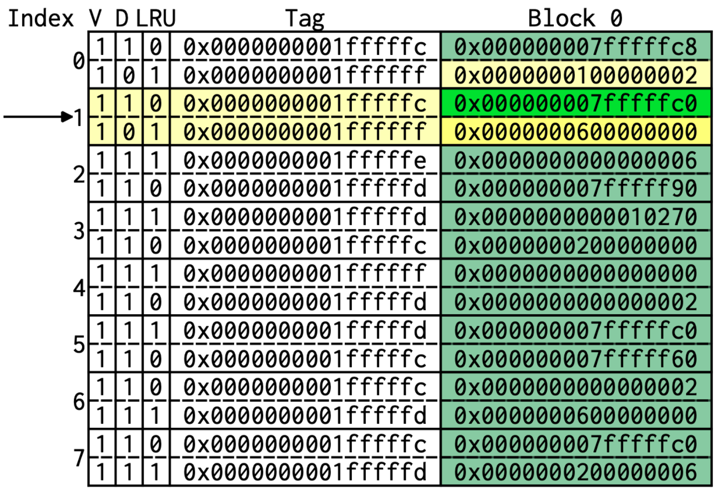
(ii) Access address



```
                 31                          65 32  0
Access address  01111111111111111111111111|000|000
```

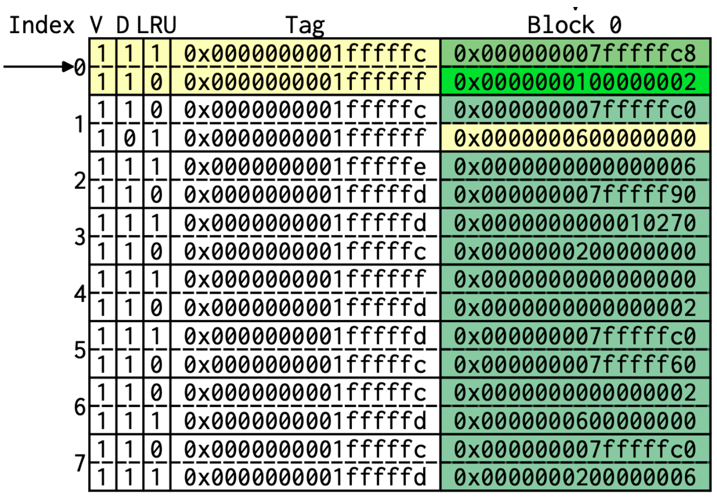
(iii)

Cache states before



Index	V	D	LRU	Tag	Block 0
0	1	1	0	0x0000000001fffffc	0x000000007fffffc8
	1	0	1	0x0000000001ffffff	0x0000000100000002
1	1	1	0	0x0000000001fffffc	0x000000007fffffc0
	1	0	1	0x0000000001ffffff	0x0000000600000000
2	1	1	1	0x0000000001fffffe	0x0000000000000006
	1	1	0	0x0000000001fffffd	0x000000007fffff90
3	1	1	1	0x0000000001fffffd	0x0000000000010270
	1	1	0	0x0000000001fffffc	0x0000000200000000
4	1	1	1	0x0000000001ffffff	0x0000000000000000
	1	1	0	0x0000000001fffffd	0x0000000000000002
5	1	1	1	0x0000000001fffffd	0x000000007fffffc0
	1	1	0	0x0000000001fffffc	0x000000007fffff60
6	1	1	0	0x0000000001fffffc	0x0000000000000002
	1	1	1	0x0000000001fffffd	0x0000000600000000
7	1	1	0	0x0000000001fffffc	0x000000007fffffc0
	1	1	1	0x0000000001fffffd	0x0000000200000006

Cache states after

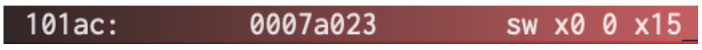


Index	V	D	LRU	Tag	Block 0
0	1	1	1	0x0000000001fffffc	0x000000007fffffc8
	1	1	0	0x0000000001ffffff	0x0000000100000002
1	1	1	0	0x0000000001fffffc	0x000000007fffffc0
	1	0	1	0x0000000001ffffff	0x0000000600000000
2	1	1	1	0x0000000001fffffe	0x0000000000000006
	1	1	0	0x0000000001fffffd	0x000000007fffff90
3	1	1	1	0x0000000001fffffd	0x0000000000010270
	1	1	0	0x0000000001fffffc	0x0000000200000000
4	1	1	1	0x0000000001ffffff	0x0000000000000000
	1	1	0	0x0000000001fffffd	0x0000000000000002
5	1	1	1	0x0000000001fffffd	0x000000007fffffc0
	1	1	0	0x0000000001fffffc	0x000000007fffff60
6	1	1	0	0x0000000001fffffc	0x0000000000000002
	1	1	1	0x0000000001fffffd	0x0000000600000000
7	1	1	0	0x0000000001fffffc	0x000000007fffffc0
	1	1	1	0x0000000001fffffd	0x0000000200000006

It's difficult to find the example in my HW5 file, so I choose to use TA's elf file and find this example. The dirty bit of second way of index 0 is 0 which means it hadn't been adapted after put into the cache. In this instruction, we get a write hit and adapt the value. After adapting the data, we change dirty bit become 1 and LRU become 0.

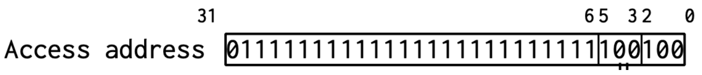
c).

(i) load/store instruction



```
101ac:          0007a023          sw x0 0 x15
```

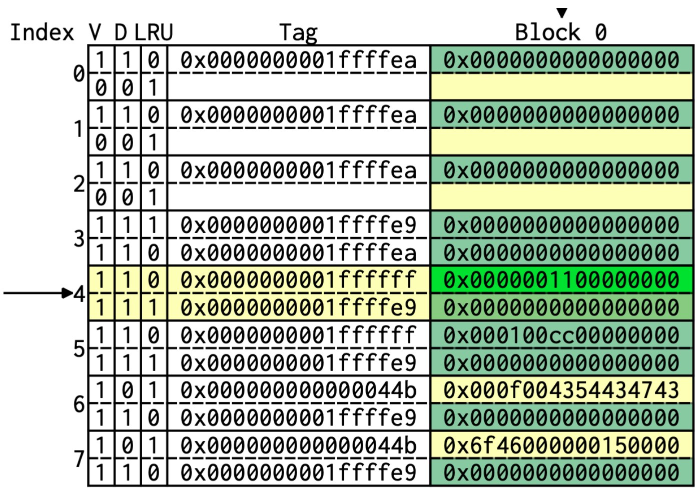
(ii) Access address



```
                 31                   6 5  3 2   0
Access address  |01111111111111111111111111|100|100|
```

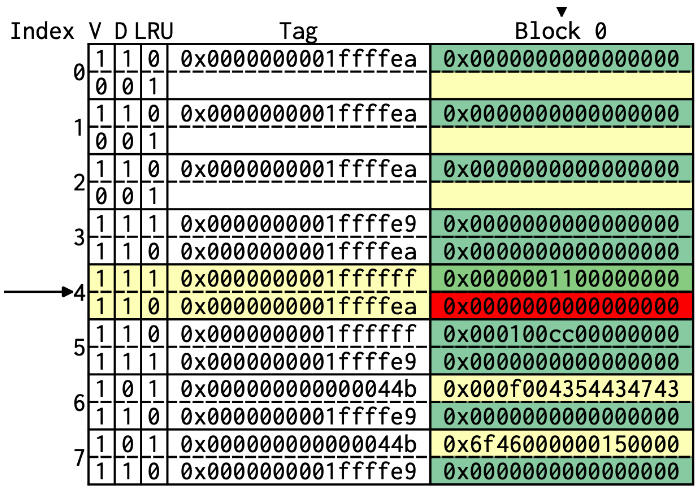
(iii)

Cache states before



Index	V	D	LRU	Tag	Block 0
0	1	1	0	0x0000000001ffffea	0x0000000000000000
	0	0	1		
1	1	1	0	0x0000000001ffffea	0x0000000000000000
	0	0	1		
2	1	1	0	0x0000000001ffffea	0x0000000000000000
	0	0	1		
3	1	1	1	0x0000000001ffffe9	0x0000000000000000
	1	1	0	0x0000000001ffffea	0x0000000000000000
→4	1	1	0	0x0000000001ffffff	0x0000001100000000
	1	1	1	0x0000000001ffffe9	0x0000000000000000
5	1	1	0	0x0000000001ffffff	0x000100cc00000000
	1	1	1	0x0000000001ffffe9	0x0000000000000000
6	1	0	1	0x000000000000044b	0x000f004354434743
	1	1	0	0x0000000001ffffe9	0x0000000000000000
7	1	0	1	0x000000000000044b	0x6f46000000150000
	1	1	0	0x0000000001ffffe9	0x0000000000000000

Cache states after

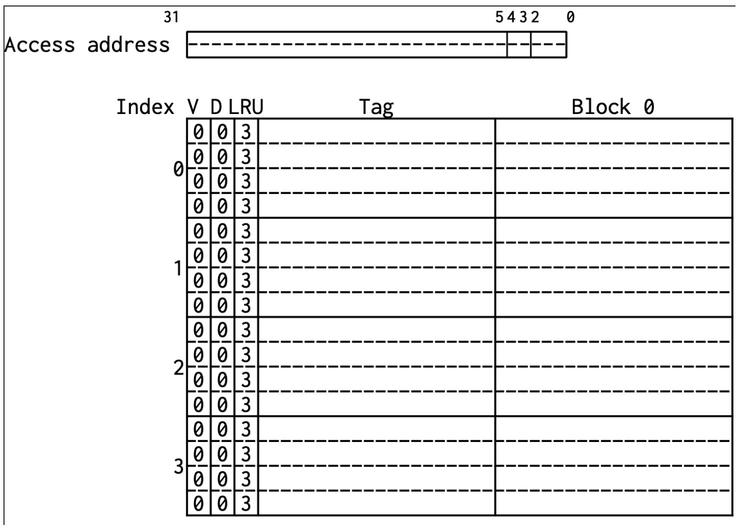


Index	V	D	LRU	Tag	Block 0
0	1	1	0	0x0000000001ffffea	0x0000000000000000
	0	0	1		
1	1	1	0	0x0000000001ffffea	0x0000000000000000
	0	0	1		
2	1	1	0	0x0000000001ffffea	0x0000000000000000
	0	0	1		
3	1	1	1	0x0000000001ffffe9	0x0000000000000000
	1	1	0	0x0000000001ffffea	0x0000000000000000
→4	1	1	1	0x0000000001ffffff	0x0000001100000000
	1	1	0	0x0000000001ffffea	0x0000000000000000
5	1	1	0	0x0000000001ffffff	0x000100cc00000000
	1	1	1	0x0000000001ffffe9	0x0000000000000000
6	1	0	1	0x000000000000044b	0x000f004354434743
	1	1	0	0x0000000001ffffe9	0x0000000000000000
7	1	0	1	0x000000000000044b	0x6f46000000150000
	1	1	0	0x0000000001ffffe9	0x0000000000000000

This is a miss example. The address should be mapped to second way of index 4. However, the space is filled with a dirty data, it need to be wrote back first. After writing back and mapping, valid and dirty bit is set as 1, and LRU is set as 0.

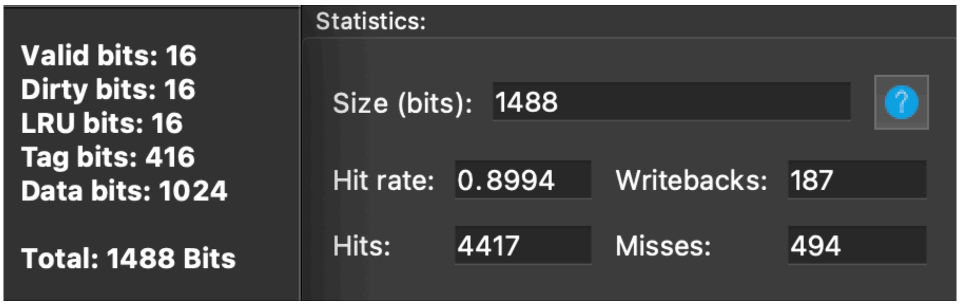
2.

(i) cache design



(ii) hit rate improvement

Origin design: hit rate=0.8994



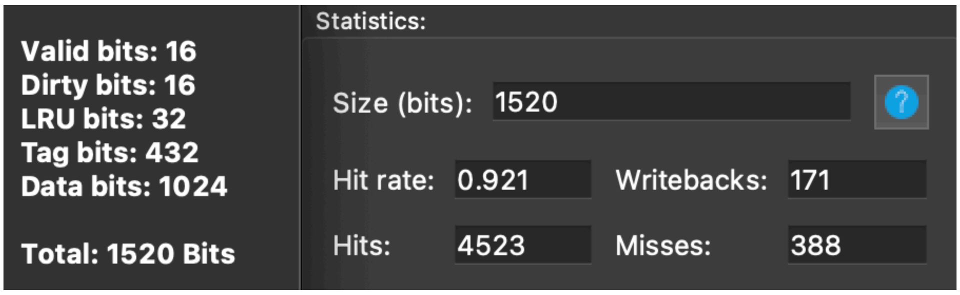
Valid bits: 16
Dirty bits: 16
LRU bits: 16
Tag bits: 416
Data bits: 1024

Total: 1488 Bits

Statistics:

Size (bits):	1488		
Hit rate:	0.8994	Writebacks:	187
Hits:	4417	Misses:	494

New design: hit rate=0.921



Valid bits: 16
Dirty bits: 16
LRU bits: 32
Tag bits: 432
Data bits: 1024

Total: 1520 Bits

Statistics:

Size (bits):	1520		
Hit rate:	0.921	Writebacks:	171
Hits:	4523	Misses:	388

After some experiment, I find 4 ways 4 sets cache have the best hit rate performance in this case. I keep cache storing only 1024 data bits because I consider different size of cache can't clearly show the different performance of different associativities. Compare between new cache design and origin one, the new one have more way for each set, which can store 2 more data with same index. I have also tried 8 ways 2 sets cache an 16 ways 1 set cache design, but the hit rate is approximate 0.90. Although there are more ways to store data, less number of sets let more memory address map to same index, the replacement increasing instead.

3.

(a) Direct-mapped with 1-word blocks and total size of 8 words

Word Address (hex)	Word Address (binary)	Tag	Index	Hit/Miss
0x74	01110100	011	101	miss
0x3c	00111100	001	111	miss
0xa4	10100100	101	001	miss
0x70	01110000	011	100	miss
0x34	00110100	001	101	miss
0x08	00001000	000	010	miss
0xc4	11000100	110	001	miss
0xb8	10111000	101	110	miss
0x94	10010100	100	101	miss
0xb8	10111000	101	110	hit

(b) 2-way set associative cache with 2-word blocks and a total size of 16 words.

Word Address (hex)	Word Address (binary)	Tag	Index	Hit/Miss
0x74	01110100	011	10	miss
0x3c	00111100	001	11	miss
0xa4	10100100	101	00	miss
0x70	01110000	011	10	hit
0x34	00110100	001	10	miss
0x08	00001000	000	01	miss
0xc4	11000100	110	00	miss
0xb8	10111000	101	11	miss
0x94	10010100	100	10	miss
0xb8	10111000	101	11	hit

108062213 顏浩昀
Computer Architecture
2022.06.12

4.

(a)

A=10011100, p1=1, p2=1, p3=1, p4 =0

B=111100101100

(b)

C=111110101100

H = h8 h4 h2 h1= 0101 -> detect error is occurred at 4+1 = 5th bit from the left

B=11100101100

(c)

2^p >= p+128+1 -> p>=9

The minimum number of parity bits is 9.

5.

(a)

P1: 50/0.5=100 (cycle), I-cache: 0.05*100=5, D-cache: 0.4*0.08*100=3.2

Total CPI = 1+5+3.2=9.2

P2: 50/1=50 (cycle), I-cache: 0.04*50=2, D-cache: 0.4*0.06*50=1.2

Total CPI = 1+2+1.2=4.2

(b)

P1: 9.2*0.5=4.6ns/instruction

P2: 4.2*1=4.2ns/instruction

P2 is faster

108062213 顏浩昀
Computer Architecture
2022.06.12

6.

4KB = 2^12 bytes -> 12bits offset

Address	Virtual Page Number	TLB miss & Page Fault	TLB			
			valid	tag	Physical page number	used
0x4a90	4	Miss & no page fault	1 1 1 1	0x5 0x0 0xa 0x4	11 5 3 9	1 0 0 1
0x210c	2	Miss & page fault	1 1 1 1	0x5 0x2 0xa 0x4	11 1 3 9	0 1 0 1
0xa3b5	a	Hit & No page fault	1 1 1 1	0x5 0x2 0xa 0x4	11 1 3 9	0 1 1 1
0x0e18	0	Miss & no page fault	1 1 1 1	0x0 0x2 0xa 0x4	5 1 3 9	1 1 0 0
0x904f	9	Miss & page fault	1 1 1 1	0x0 0x2 0x9 0x4	5 1 2 9	1 0 1 0