CS3350B Computer Organization
Chapter 1: CPU and Memory
A Cache-Filling Example

Iqra Batool

Department of Computer Science
University of Western Ontario, Canada

January 24, 2024

Fill a cache

Consider a 2-way set associative cache with 4 byte cache lines, 4 sets, and an LRU replacement policy. Consider also the following sequence of memory addresses:

4, 6, 26, 12, 0, 2, 21, 31

Question: What are the cache's contents after this sequence of memory addresses are accessed? Which accesses are cache hits? Which are misses?

3 Steps:

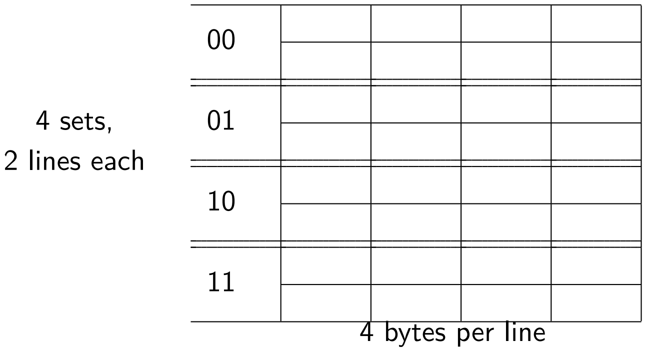
1. Draw a table with the correct shape based on the cache's specifications.
2. (Optional) determine each address's set index and block offset.
3. Fill the cache one access after another, determine hit/misses along the way

Step 1: Draw the table

Consider a 2-way set associative cache with 4 byte cache lines, 4 sets, and an LRU replacement policy.

↳ This tells us, 4 sets, 2 lines per set, 4 bytes per line.

↳ It's an 8x4 table.



Step 2: Convert Addresses to set and block offset

4 sets $\implies$ 2 bits for set index.
4 bytes per line $\implies$ 2 bits of block offset.

$$\boxed{4, \quad 6, \quad 26, \quad 12, \quad 0, \quad 2, \quad 21, \quad 31}$$

$(4)_{10} = (0...00100)_2 \implies$ set index is 01, block offset is 00
$(6)_{10} = (0...00110)_2 \implies$ set index is 01, block offset is 10
$(26)_{10} = (0...11010)_2 \implies$ set index is 10, block offset is 10
$\vdots$

Notice actual size m of address not important.

Step 3: Fill the cache one by one

4,	6,	26,	12,	0,	2,	21,	31

$(4)_{10} = (0...00100)_2 \implies$ set index is 01, block offset is 00
Cache initially empty, so access to 4 misses.
Its cache line gets installed in the cache.
It gets installed in set with index 01, and 4 goes in block offset 0.
Fill the rest of the cache line in: 4,5,6,7.

00				
01	4	5	6	7
10				
11				

Step 3: Fill the cache one by one

4,	6,	26,	12,	0,	2,	21,	31

Repeat:

$(6)_{10} = (0...00110)_2 \implies$ set index is 01, block offset is 10

$(26)_{10} = (0...11010)_2 \implies$ set index is 10, block offset is 10

$\vdots$

00				
01	4	5	6	7
10				
11				

Step 3: Fill the cache one by one

4,	6,	26,	12,	0,	2,	21,	31

Repeat:

$(6)_{10} = (0...00110)_2 \implies$ set index is 01, block offset is 10

$(26)_{10} = (0...11010)_2 \implies$ set index is 10, block offset is 10

$\vdots$

6 hits!

00				
01	4	5	**6**	7
10				
11				

Step 3: Fill the cache one by one

4,	6,	26,	12,	0,	2,	21,	31

Repeat:

$(6)_{10} = (0...00110)_2 \implies$ set index is 01, block offset is 10

$(26)_{10} = (0...11010)_2 \implies$ set index is 10, block offset is 10

$\vdots$

00				
01	4	5	**6**	7
10	24	25	26	27
11				

Step 3: After a while...

4,	6,	26,	12,	0,	2,	21,	31

00	0	1	**2**	3
01	4	5	**6**	7
	20	21	22	23
10	24	25	26	27
11	12	13	14	15
	28	29	30	31

The Better Way

Recall that Step 2 was “optional”.

Rather than computing the set index and block offset, we can simply *enumerate*, or count, up from 0 to determine where an address should go.

Recall, address 0 *always* maps to set 0. Count up from there.

Step 3: Fill the cache one by one

4,	6,	26,	12,	0,	2,	21,	31

Count up from 0 until you find 4: 0, 1, 2, 3, **4**.

- If a miss, install that cache block.
- If a hit, done!

00	0	1	2	3
01	4	5	6	7
10				
11				

Step 3: Fill the cache one by one

4, 6, 26, 12, 0, 2, 21, 31

Count up from 0 until you find 6: 0, 1, 2, 3, 4, 5, **6**.

- If a miss, install that cache block.
- If a hit, done!

00	0	1	2	3
01	4	5	**6**	7
10				
11				

Step 3: Fill the cache one by one

4, 6, 26, 12, 0, 2, 21, 31

Count up from 0 until you find 26. For large numbers, count upward in multiples of the cache block size: 0, 4, 8, 12, 16, 20, 24, 25, **26**.

- If a miss, install that cache block.
- If a hit, done!

00	~~0~~ 16			
01	4	5	**6**	7
	20			
10	~~8~~ 24	25	26	27
11	12			

Step 3: Fill the cache one by one

4, 6, 26, 12, 0, 2, 21, 31

Count up from 0 until you find 12. For large numbers, count upward in multiples of the cache block size: 0, 4, 8, **12**.

- If a miss, install that cache block.
- If a hit, done!

00	0			
01	4	5	**6**	7
10	24	25	26	27
	8			
11	12	13	14	15

Step 3: Fill the cache one by one

4, 6, 26, 12, 0, 2, 21, 31

Count up from 0 until you find 0: **0**.

- If a miss, install that cache block.
- If a hit, done!

00	0	1	2	3
01	4	5	**6**	7
10	24	25	26	27
11	12	13	14	15

Step 3: Fill the cache one by one

4,	6,	26,	12,	0,	2,	21,	31

Count up from 0 until you find 2: 0, 1, **2**.

- If a miss, install that cache block.
- If a hit, done!

00	0	1	**2**	3
01	4	5	**6**	7
10	24	25	26	27
11	12	13	14	15

Step 3: Fill the cache one by one

4, 6, 26, 12, 0, 2, 21, 31

Count up from 0 until you find 21: 0, 4, 8, 12, 16, 20, **21**.

- If a miss, install that cache block. Be wary of multiple ways!
- If a hit, done!

00	0	1	**2**	3
	16			
01	4	5	**6**	7
	20	21	22	23
10	24	25	26	27
	8			
11	12	13	14	15

Step 3: Fill the cache one by one

$\boxed{4, \quad 6, \quad 26, \quad 12, \quad 0, \quad 2, \quad 21, \quad 31}$

Count up from 0 until you find 31: 0, 4, 8, 12, 16, 20, 24, 28, 29, 30, **31**.

- If a miss, install that cache block. Be wary of multiple ways!
- If a hit, done!

00	0	1	**2**	3
	16			
01	4	5	**6**	7
	20	21	22	23
10	24	25	26	27
	8			
11	12	13	14	15
	28	29	30	31

Some Final Notes

- Even though we write a cache line in the table using, e.g., 4, 5, 6, 7, we must understand that this is simply shorthand for saying “the data stored in memory located at addresses 4, 5, 6, 7”. **The cache stores data** (and tags), **not memory addresses.**
- When filling a cache line, it can be filled in other ascending or descending order. In any case, the same data will always exist in the same cache line. For example, the below two tables are equivalent.

00	0	1	**2**	3
01	4	5	**6**	7
	20	21	22	23
10	24	25	26	27
11	12	13	14	15
	28	29	30	31

00	3	**2**	**1**	0
01	7	**6**	5	4
	23	22	21	20
10	27	26	25	24
11	15	14	13	12
	31	30	29	28