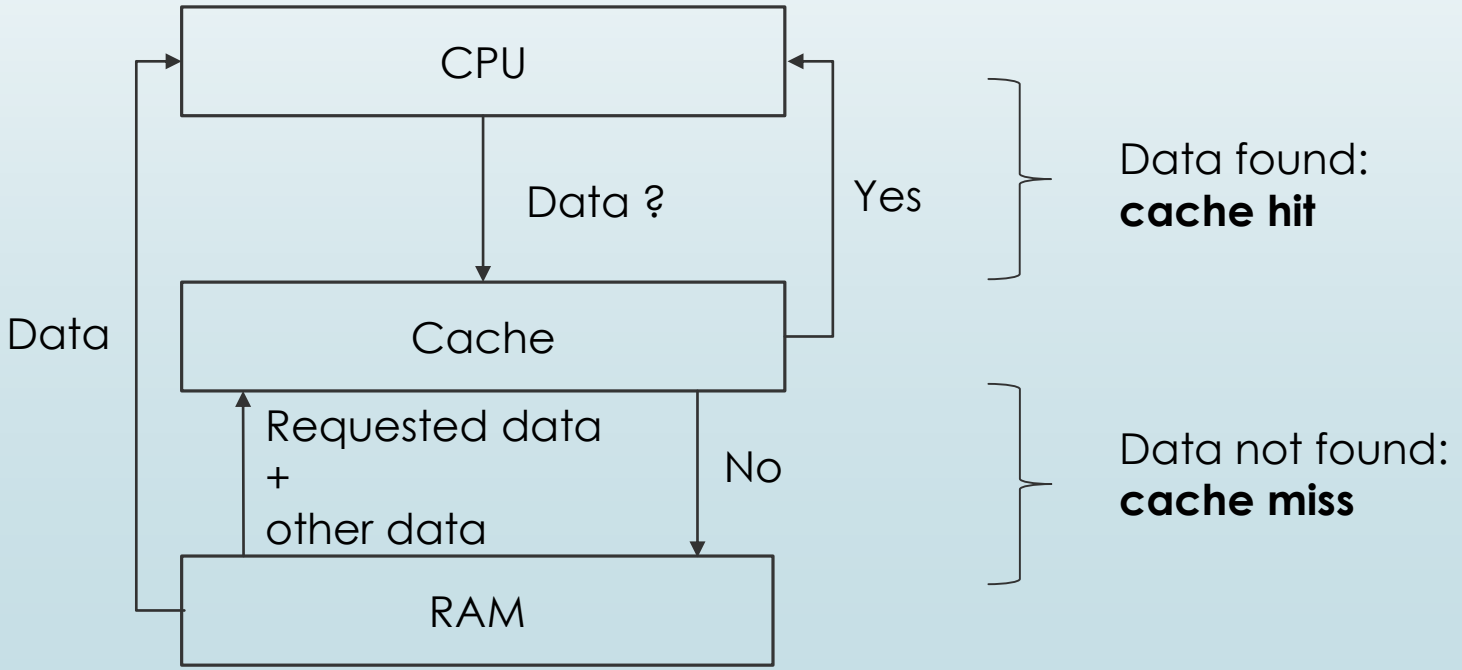
Cache memory

To fill the performance gap between the CPU and the RAM.

When the CPU needs to acccess memory, the cache is examined.



Mapping process

The transformation of data between from RAM to Cache.

RAM and Cache are decomposed into cells of one byte each.

Transformation between RAM and Cache is done line by line.

RAM and cache are divided into lines of same length.

Line length (usually 64 bytes): the number of bytes brought from RAM to Cache in response to a cache miss.

Three strategies:

1. Direct mapping
2. Fully associative
3. Set associative

Direct mapping

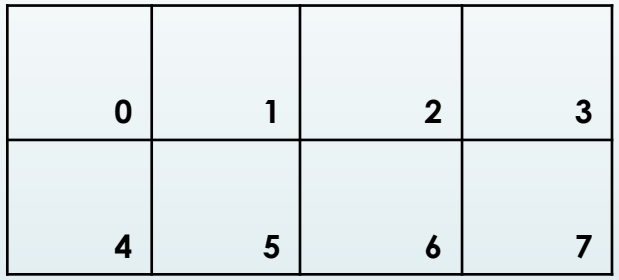
The data of a memory address is stored in only one location in the cache.

Given a cache of 8 bytes divided into lines of 4 bytes each → 2 lines
A RAM of 16 bytes **must be divided** into lines of 4 bytes each → 4 lines

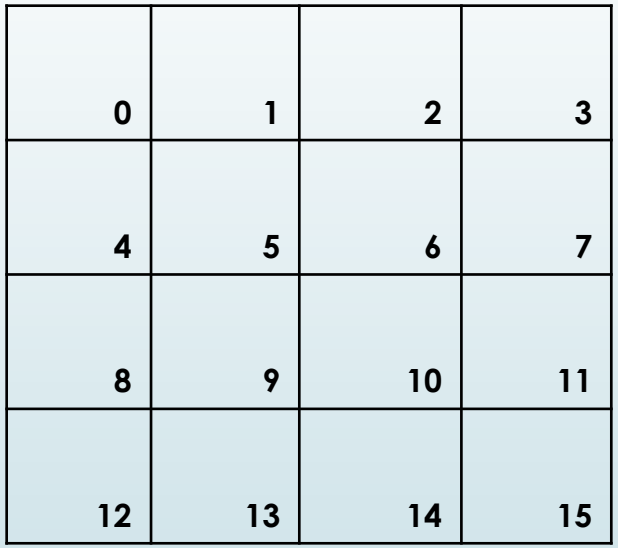
Determine the size of memory address ?

Cell size: 1 byte
Number of cells in the RAM: $\frac{16}{1} = 16$
Memory address on: $log_2(16) = 4$ bits

Direct mapping



Cache



RAM

This cache can store only two lines from RAM at the same time.

Assume that the cache contains 2 RAM lines.

Q: How many bits are needed to determine the position of a byte within a line ?

Line length: 4 bytes → Positions: 0, 1, 2, 3
→ size(offset): 2 bits

Q: How many bits are needed to determine the line in which a byte is located?

Number of lines in the cache: 2
→ size(index): 1 bit.

Q: Index and offset are not enough because memory addresses 7 and 15 have the same index and offset.

The remaining bits in the RAM address are used as a tag
→ size(tag): 4-1-2=1

4-bit addresses in RAM

0	0	0	0	RAM Line 0
0	0	0	1	
0	0	1	0	**Cache Line 0**
0	0	1	1	
0	1	0	0	RAM Line 1
0	1	0	1	
0	1	1	0	**Cache Line 1**
0	1	1	1	
1	0	0	0	RAM Line 2
1	0	0	1	
1	0	1	0	**Cache Line 0**
1	0	1	1	
1	1	0	0	RAM Line 3
1	1	0	1	
1	1	1	0	**Cache Line 1**
1	1	1	1	
Tag	**Index**	**Offset**		

Direct mapping

A memory address is decomposed into 3 fields:

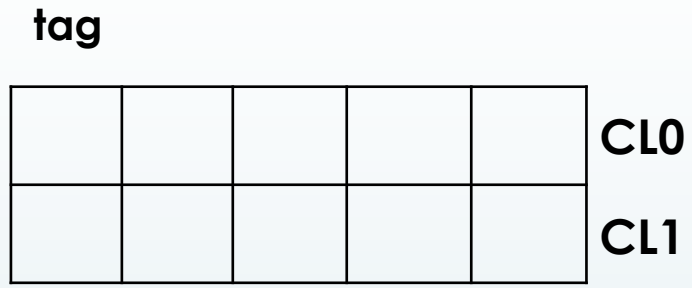
$$size(offset) = log_2(line\ length)$$

$$size(index) = log_2(Number\ of\ lines\ in\ the\ cache)$$

$$size(tag) = size(memory\ address) - size(offset) - size(index)$$

Direct mapping

All bytes in the same RAM line have the same tag → each cache line must have only one tag.



Cache

When starting the computer, the cache is empty:
bits of tags and bits of cells are set to 0.

Assume that the CPU needs to access the byte at the address $5 = (0101)_2$.

CPU examines the cache?
Index: 1
Tag:0
Offset: 1

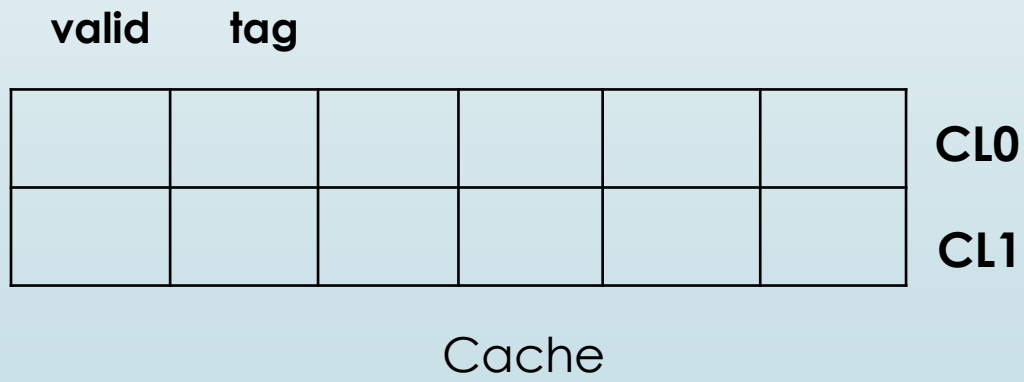
→ return 0. But the address 5 in the RAM contains 25, for example!

Direct mapping

How to know if a line contains data or not ?

Add one valid bit (v) to each cache line.

v_i :
- 1 if CL_i contains data
- 0 otherwise



Cache

Direct mapping

In which memory the CPU modifies data?

- **Write-through cache**: data is updated in cache and memory at the same time.
- **Write-back cache**: data is updated in cache only.

Write-back cache

We need an extra bit: dirty bit (d).

$$d_i : \begin{cases} 1 & \text{if } CL_i \text{ is modified} \\ 0 & \text{otherwise} \end{cases}$$

Direct mapping: pseudo-algorithm

Consider a cache of **n** lines and the **@**

tag	index	offset

Reset

$\forall\ i \in [0, n-1], v_i \leftarrow 0$

Check

$v_{index} = 0$ → cache miss
$t_{index} \neq tag$ → cache miss
cache hit

Get

Check?
Load the byte at the position offset

Set

Check?
$d_{index} \leftarrow 1$
Modify the byte at the position offset

Direct mapping: pseudo-algorithm

Transfer of data in response to a cache miss

1. Determine the RAM line RL_i to be moved

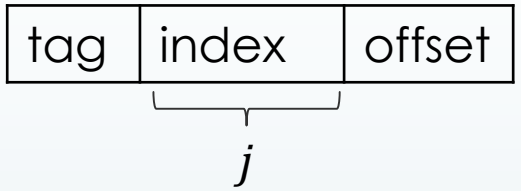
Example:
The address $(0101)_2$ belongs to RL_1 since $(01)_2$=1.

What is the address of the first byte in RL_i?

$$i \times line\ length$$

Direct mapping: pseudo-algorithm

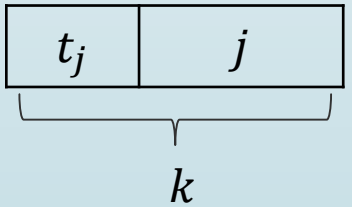
2. Determine the Cache line CL_j to be replaced



Example:
RL_1 that contains the address $(0\underbrace{1}_{\downarrow}01)_2$ must be moved to CL_1 since $(1)_2$=1.

3. Once i and j are determined perform the following steps:

Step1: if $d_j = 1$ → move CL_j to the corresponding RAM line RL_k where

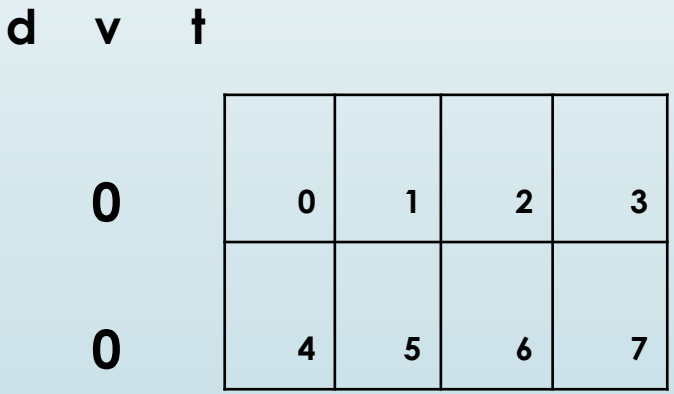


Step2:
$v_j \leftarrow 1$
$d_j \leftarrow 0$
$t_j \leftarrow$ tag

Exercise

Explain how the CPU executes this program using the direct mapping process.

I1: Load byte at @5
I2: Load byte at @9
I3: Add the loaded 2 bytes and store result at @6
I4: Load byte at @13



Cache

RAM

Address on : 4 bits

Line length: 4 bytes

Offset: 2 bits

Index: 1 bit

Tag: 1

I1:

0101

Index: 1

V1=0 → cache miss

RL1 → CL1

I2:

1001

Index: 0

V0=0 → cache miss

RL2 → CL0

I3:

0110

Index: 1

V1=1

T1=0=tagof @ → cache hit

I4:

1101

Index: 1

V1=1

T1=0 not equal to tagof @ → cache miss

CL1 → RL (01)2 =RL 1

RL3 → CL1