PRIMARY MEMORY

Cache Memory

Imbalance Between CPU and Memory

- Historically , CPU is faster than memories
- The capacity of memories are increased, not speed
- The slower the memory, the more cycles the CPU will have to wait

Two solutions:

1. Hardware, continue executing and stall CPU if an instruction tries to use the memory word before it has arrived;
2. Software, the compiler is forced to insert NOP (no operation)

Technology problem?

1. No, economics problem. Engineers can build memories as fast as CPUs
2. Has to be located on the CPU chip and then make CPU chip more expensive and bigger

PRIMARY MEMORY

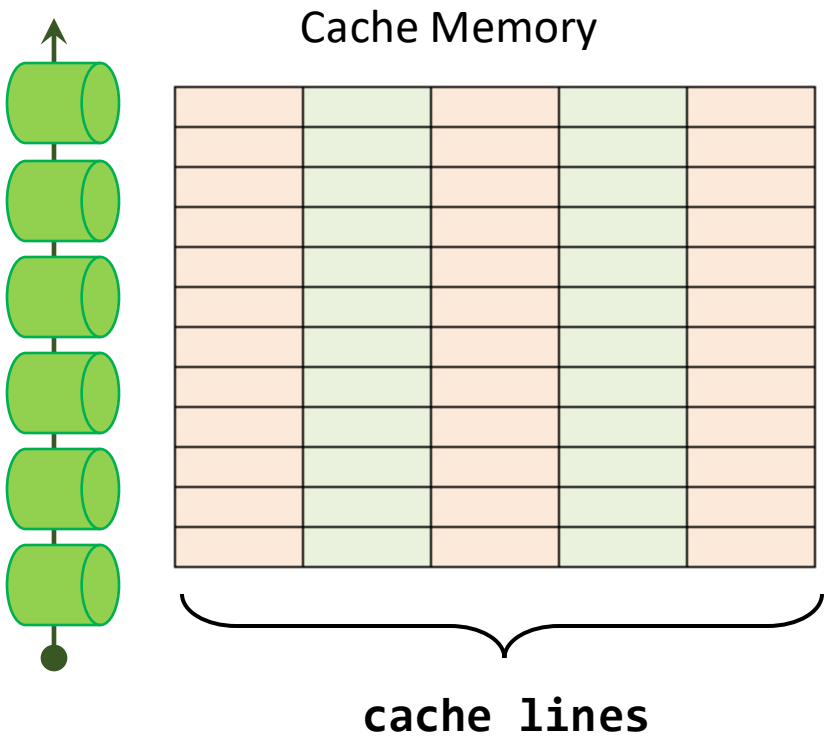
Cache Memory

Practical Solution

1. Combining a small amount of fast memory (**cash**) with a large amount of slow memory to get the speed of the fast memory (almost) and the capacity of the large memory at a moderate price.
2. Most Heavily used memory words are kept in the cache.
3. The general idea is that when a word is referenced, it and some of its neighbors are brought from the large slow memory into the cache, so that the next time it is used, it can be accessed quickly
4. **Locality Principle**: when a word is referenced, it and some of its neighbors are brought from large slow memory to the cache
5. **Temporal Principle**: if a memory location is accessed once, it is likely to be accessed again soon (keep recently accessed data in cache memory)

6. When the CPU needs a word, it first looks in the cache

PRIMARY MEMORY

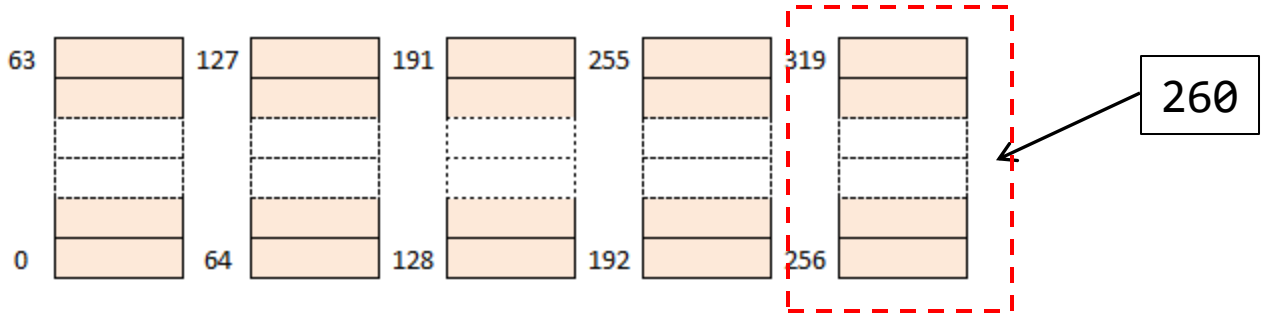


1. Main memories and caches are divided up into fixed-size blocks (**cache lines**).
2. Line is loaded from the main memory into the cache, not just the word needed.

PRIMARY MEMORY

Cache Memory

Example : a 64-byte line size, a reference to memory address 260, what line is pulled into one cash line ?

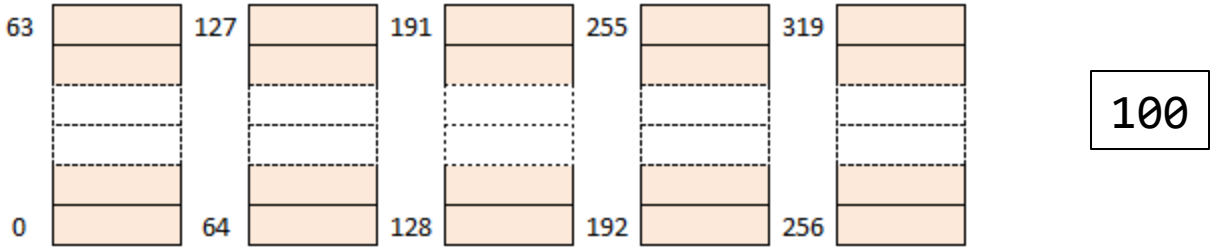


Answer: line consisting of bytes 256 to 319 is pulled into one cache line.

PRIMARY MEMORY

Cache Memory

Example : a 64-byte line size, a reference to memory address 100, what line is pulled into one cash line ?



PRIMARY MEMORY

Cache mapping

Main memory is divided up into fixed size blocks called **cache lines** (4 to 64 consecutive bytes)

Lines are numbered consecutively starting at 0, so with a 32-byte line size, line 0 is bytes 0 to 31, line 1 is bytes 32 to 63, and so on

When memory is referenced, the cache controller circuit checks to see if the word referenced is currently in the cache (**cache hit**). If so, the value there can be used, saving a trip to main memory. If the word is not there (**cache miss**), some line entry is removed from the cache and the line needed is fetched from memory.

Cache memory

cache hit and miss

$$\text{Miss rate} = \frac{\text{Number of misses}}{\text{Number of total memory accesses}} = 1 - \text{Hit rate}$$

$$\text{Hit rate} = \frac{\text{Number of hits}}{\text{Number of total memory accesses}} = 1 - \text{Miss rate}$$

Number of total memory accesses = number of hits + number of misses

Example:

Suppose that a CPU has a level 1 cache and a level 2 cache, with access times of 1 nsec and 3 nsec, respectively. The main memory access time is 15 nsec. If 30% of the accesses are level 1 cache hits and 50% are level 2 cache hits, what is the average access time?

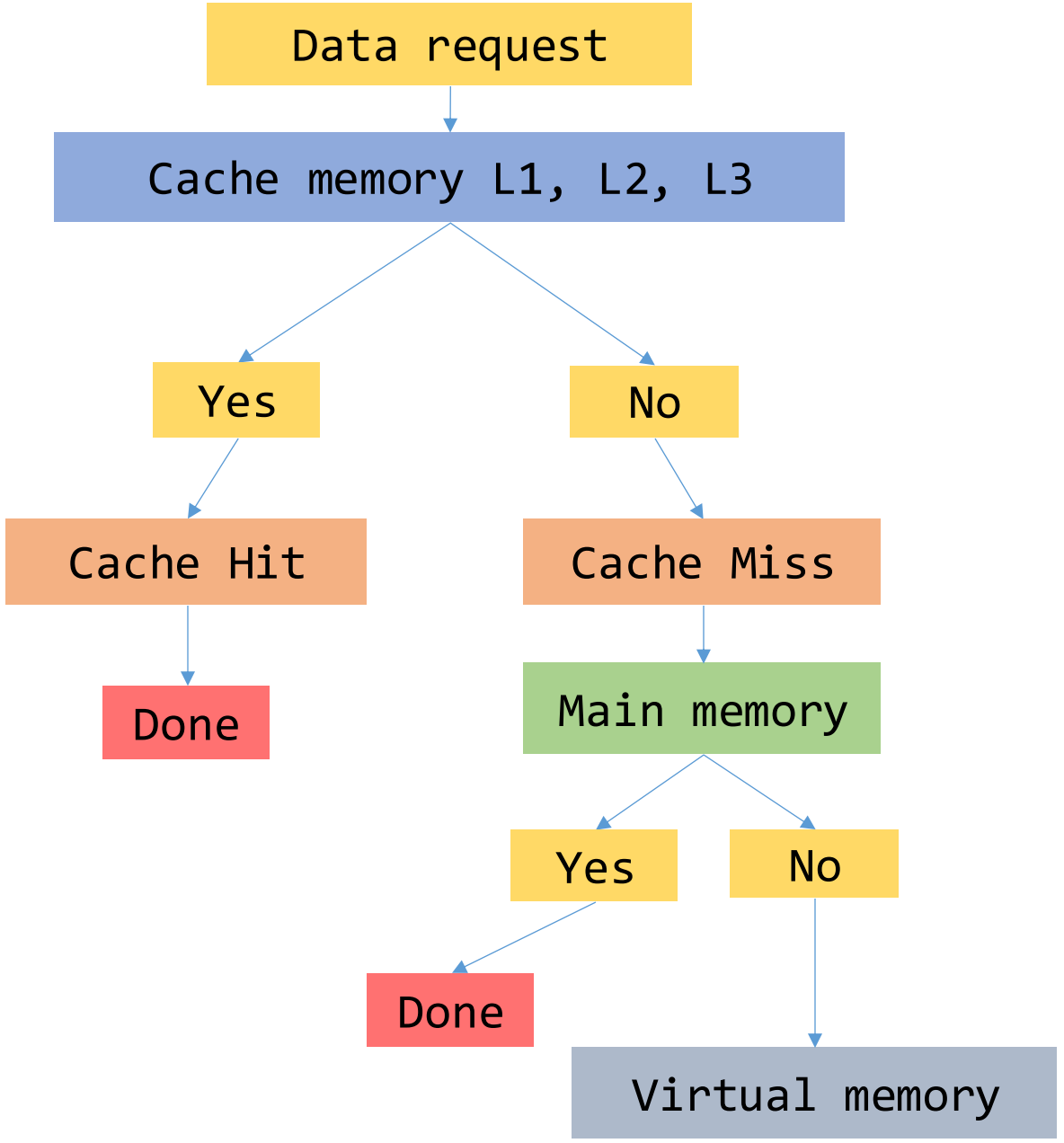
Example:

Suppose that a CPU has a level 1 cache and a level 2 cache, with access times of 1 nsec and 3 nsec, respectively. The main memory access time is 15 nsec. If 30% of the accesses are level 1 cache hits and 50% are level 2 cache hits, what is the average access time?

Main memory = 100% – (30%+50%) = 20%

The average access time = 0. 30 × 1 + 0. 50 × 3 + 0. 20 × 15 = 4.8 nsec

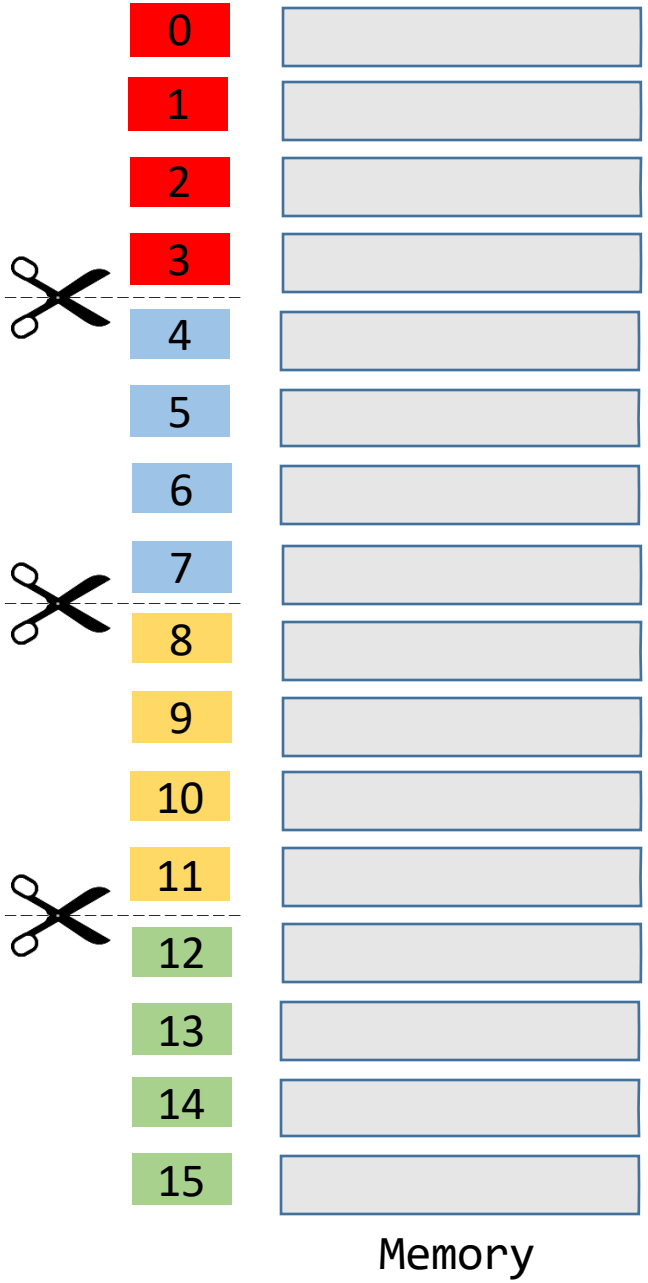
Data read by CPU



Cache Memory Mapping Techniques

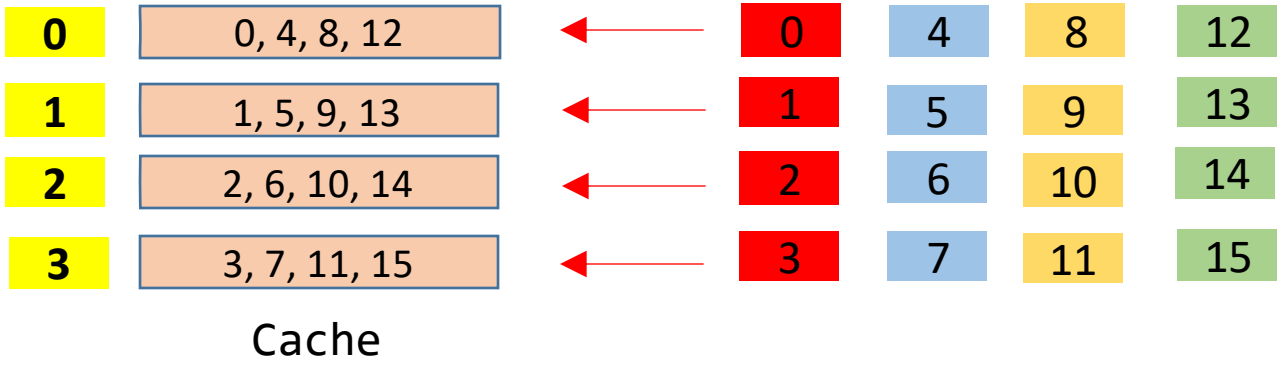
Direct mapping
Associative mapping
Set - Associative mapping

Direct mapping

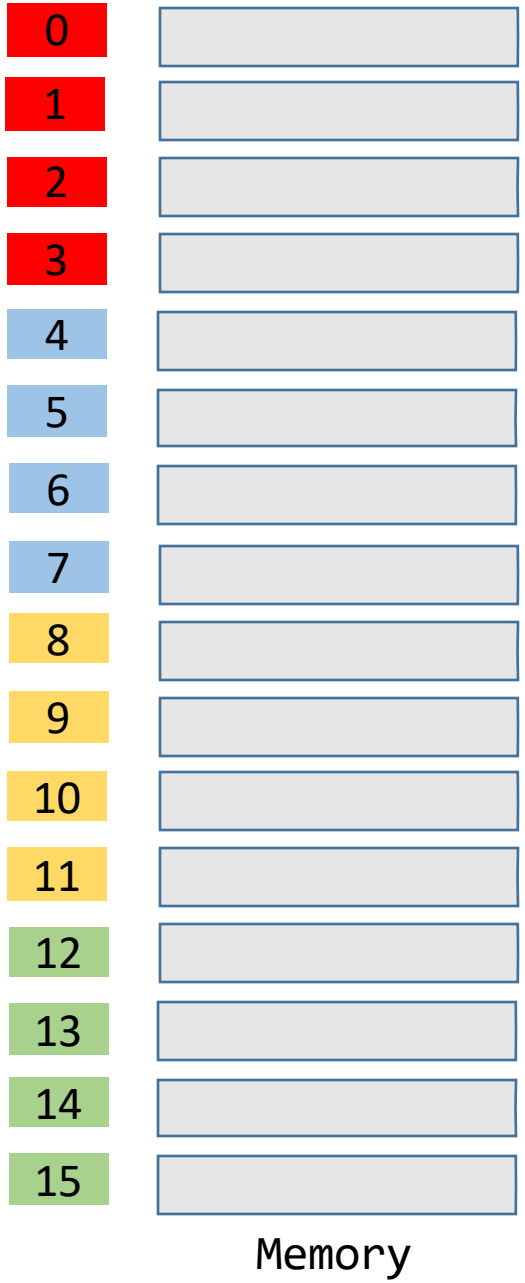


1. Main memory is divided up into fixed size blocks called cache lines
2. A given memory block can be loaded into one and only one cache line
3. Cache line number = memory address % number of cache line

e.g.
Cache line 0 = 0, 4, 8, or 12 (main memory block)
Or
Cache line 3 = 3, 7, 11, or 15 (main memory block)



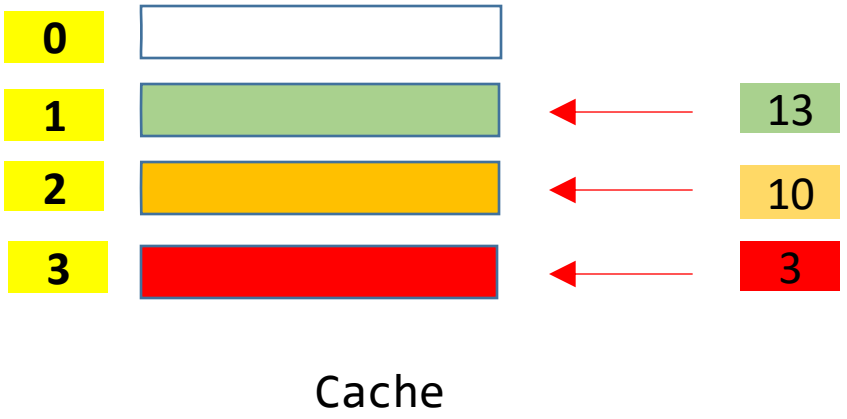
Direct mapping



Example: Obtain the cache line numbers for the following memory addresses:
3, 10,13

Cache line number = memory address % number of cache line

Cache line number for 3 = 3 % 4 = 3
Cache line number for 10 = 10 % 4 = 2
Cache line number for 13 = 13 % 4 = 1



Direct mapping

Example:

Obtain the cache line numbers for the following memory addresses 386, 255, 4000, 4095 assuming that
The number of cache lines (blocks) = 128 (from 0 to 127)
The number of memory locations   = 4096 (from 0 to 4095)

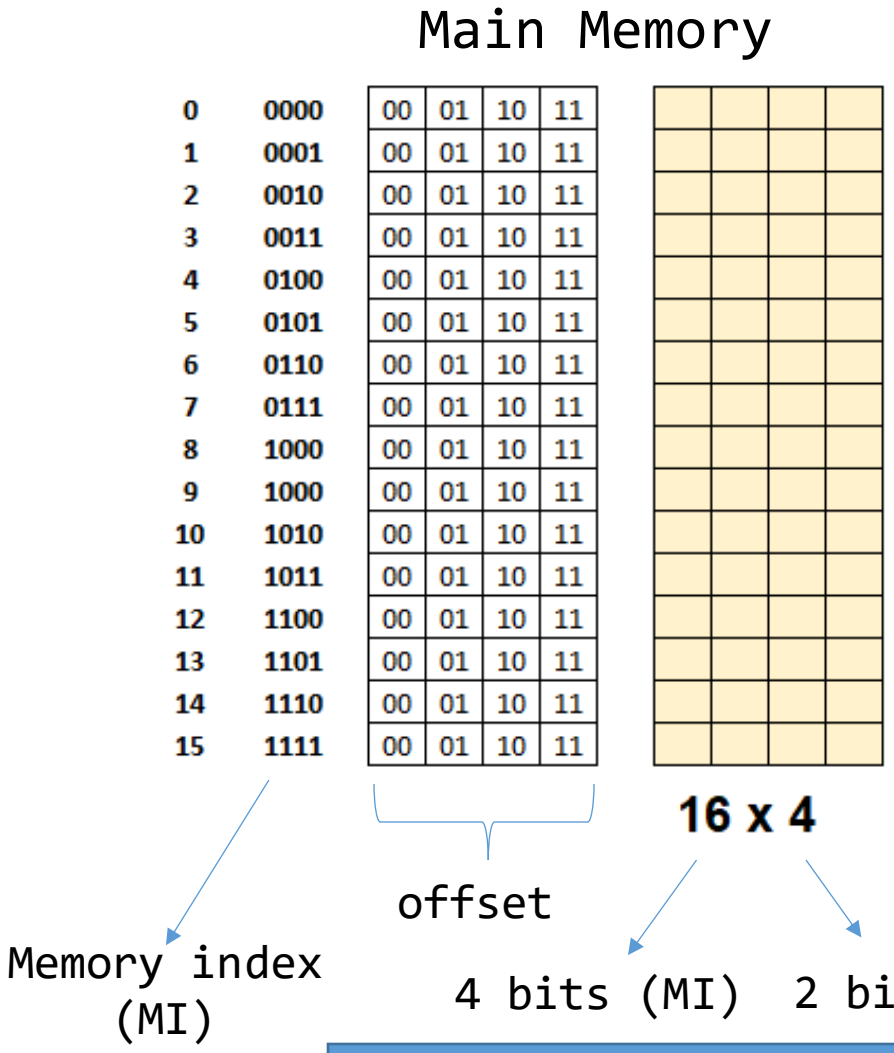
Direct mapping

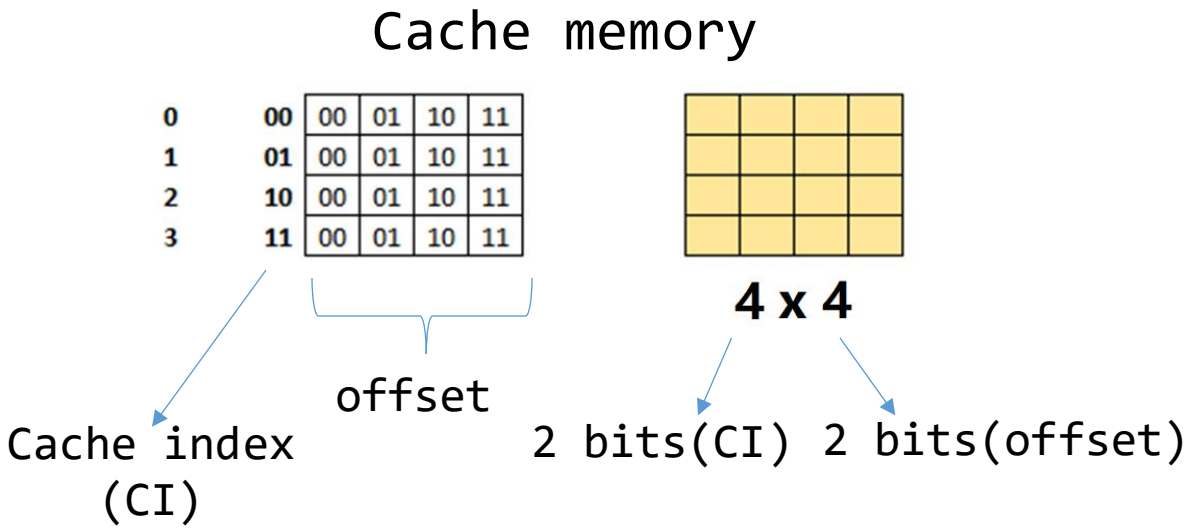
Example:

Obtain the cache line numbers for the following memory addresses 386, 255, 4000, 4095 assuming that
The number of cache lines (blocks) = 128 (from 0 to 127)
The number of memory locations   = 4096 (from 0 to 4095)

Cache line number for 386   = 386 % 128 = 2
Cache line number for 255   = 255 % 128 = 127
Cache line number for 4000 = 4000 % 128 = 32
Cache line number for 4095 = 4095 % 128 = 127

Direct mapping



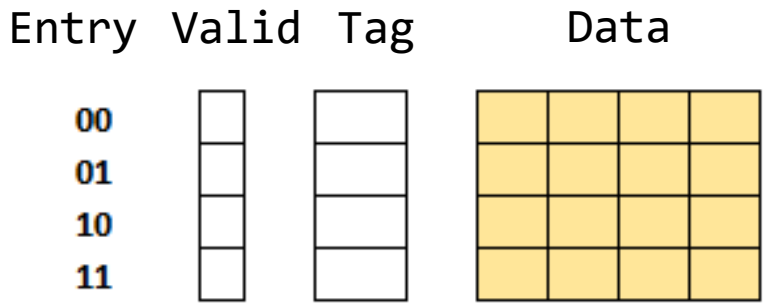


Cache address = 2(CI) + 2(offset) = 4 bits
4 cache line and each cache line contains 4 cells (e.g. 4 bytes)

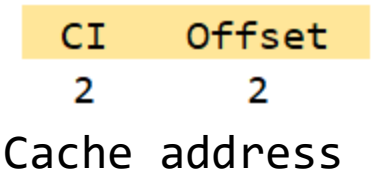
Memory address = 4(MI) + 2(offset) = 6 bits

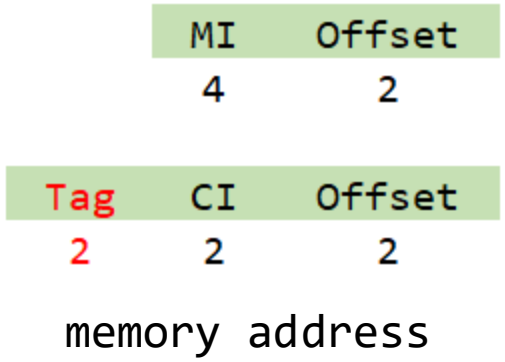
Each cache entry consists of three parts:

1. The Valid bit indicates whether there is any valid data in this entry or not. When the system is booted (started), all entries are marked as invalid.
2. The Tag field consists of a unique, 2-bit value identifying the corresponding line of memory from which the data came.
3. The Data field contains a copy of the data in memory. This field holds one cache line of 4 bytes.



For storing and retrieving data from the cache, the memory address is broken into three components as follows:





Tag = MI - CI

Example: A list of memory requests by CPU is given below. Use this list to find hit and miss rate.

0	0000	00	01	10	11	M0
1	0001	00	01	10	11	M1
2	0010	00	01	10	11	M2
3	0011	00	01	10	11	M3
4	0100	00	01	10	11	M4
5	0101	00	01	10	11	M5
6	0110	00	01	10	11	M6
7	0111	00	01	10	11	M7
8	1000	00	01	10	11	M8
9	1001	00	01	10	11	M9
10	1010	00	01	10	11	M10
11	1011	00	01	10	11	M11
12	1100	00	01	10	11	M12
13	1101	00	01	10	11	M13
14	1110	00	01	10	11	M14
15	1111	00	01	10	11	M15

Entry (CI)	Valid	Tag	Data
00	1	00	M0
01	1	01	M5
10	0		
11	1	11	M15

Entry (CI)	Valid	Tag	Data
00	1	01	M4
01	1	01	M5
10	1	10	M10
11	1	11	M15

Memory Requests by CPU

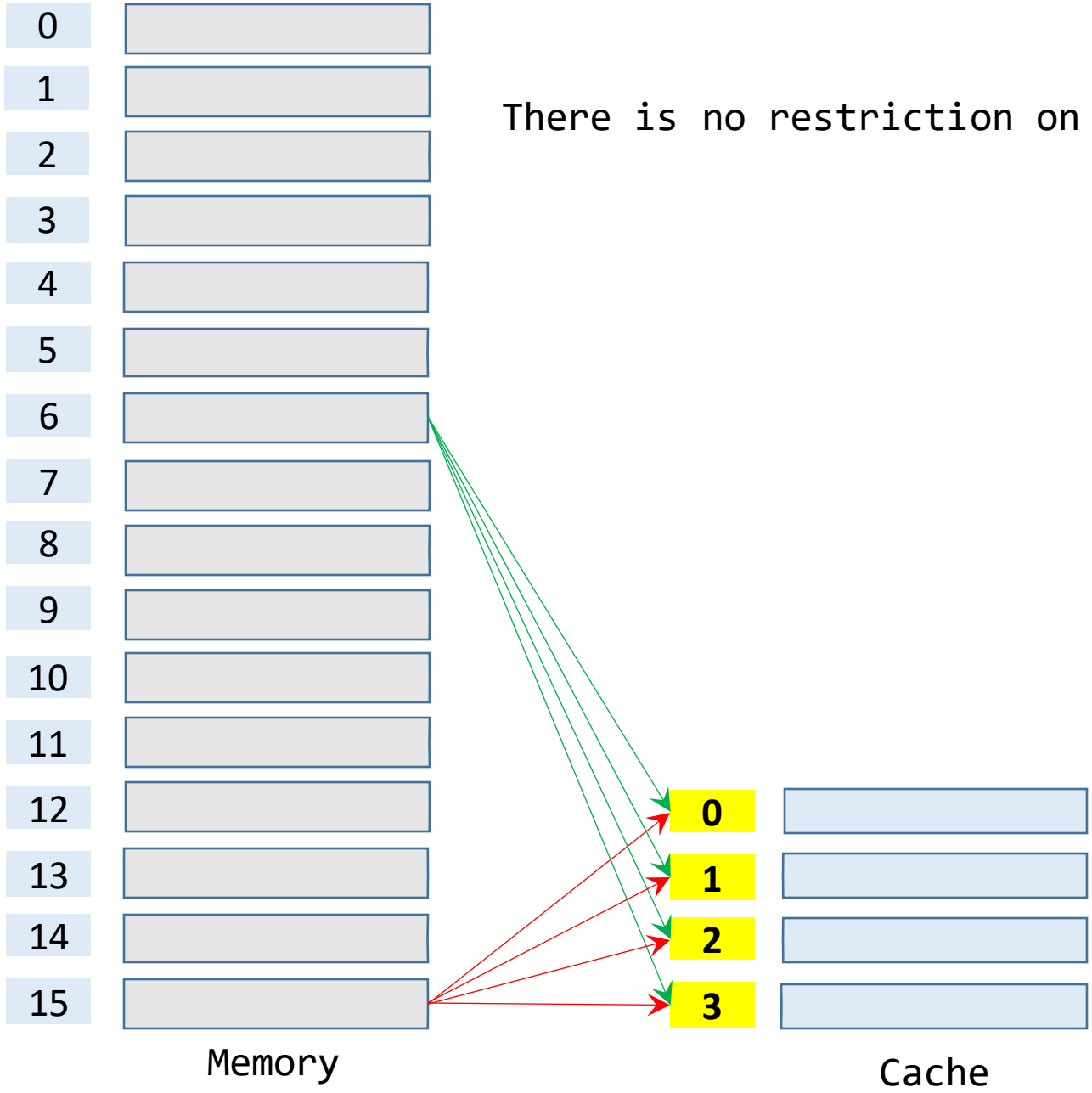
Tag	CI	Offset		
0 0 0 0	0	0	Hit	M0[00]=M0[0]
0 1 0 1	1	0	Hit	M5[10]=M5[2]
0 1 0 0	0	1	Miss	M4[01]=M4[1]
0 1 0 1	1	1	Hit	M5[11]=M5[3]
1 1 1 1	0	0	Hit	M15[00]=M15[0]
1 0 1 0	0	0	Miss	M10[00]=M10[0]

$$\text{Hit rate} = \frac{4}{6}$$

$$\text{Miss rate} = \frac{2}{6}$$

Associative mapping

There is no restriction on mapping from memory to cache



Associative mapping

For storing and retrieving data from the cache, the memory address is broken into two components as follows:

CI	Offset
2	2

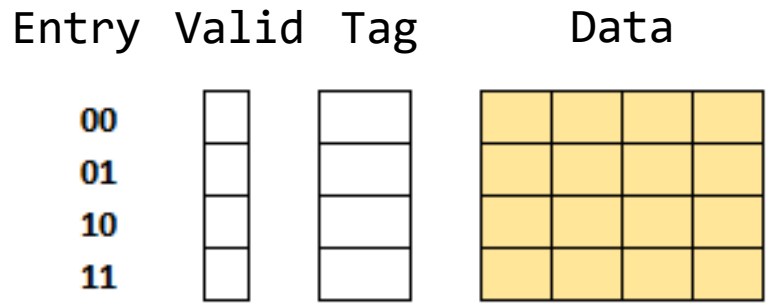
Cache address

MI	Offset
4	2

Tag	Offset
4	2

memory address

Tag = MI



Example: A list of memory requests by CPU is given below. Determine cache memory hit or miss for each case.

0	0000	00	01	10	11	M0
1	0001	00	01	10	11	M1
2	0010	00	01	10	11	M2
3	0011	00	01	10	11	M3
4	0100	00	01	10	11	M4
5	0101	00	01	10	11	M5
6	0110	00	01	10	11	M6
7	0111	00	01	10	11	M7
8	1000	00	01	10	11	M8
9	1001	00	01	10	11	M9
10	1010	00	01	10	11	M10
11	1011	00	01	10	11	M11
12	1100	00	01	10	11	M12
13	1101	00	01	10	11	M13
14	1110	00	01	10	11	M14
15	1111	00	01	10	11	M15

Entry (CI)	Valid	Tag	Data
00	1	0010	M2
01	1	0100	M4
10	1	0001	M1
11	1	1111	M15

Entry (CI)	Valid	Tag	Data
00	1	0010	M2
01	1	0100	M4
10	1	0001	M1
11	1	0011	M3

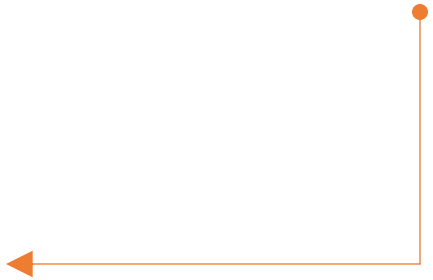
Memory Requests by CPU

Tag Offset

000101 Hit M1[01]

001111 Miss M3[11]

Replacement Algorithms
e.g. RANDOM



Set - Associative mapping

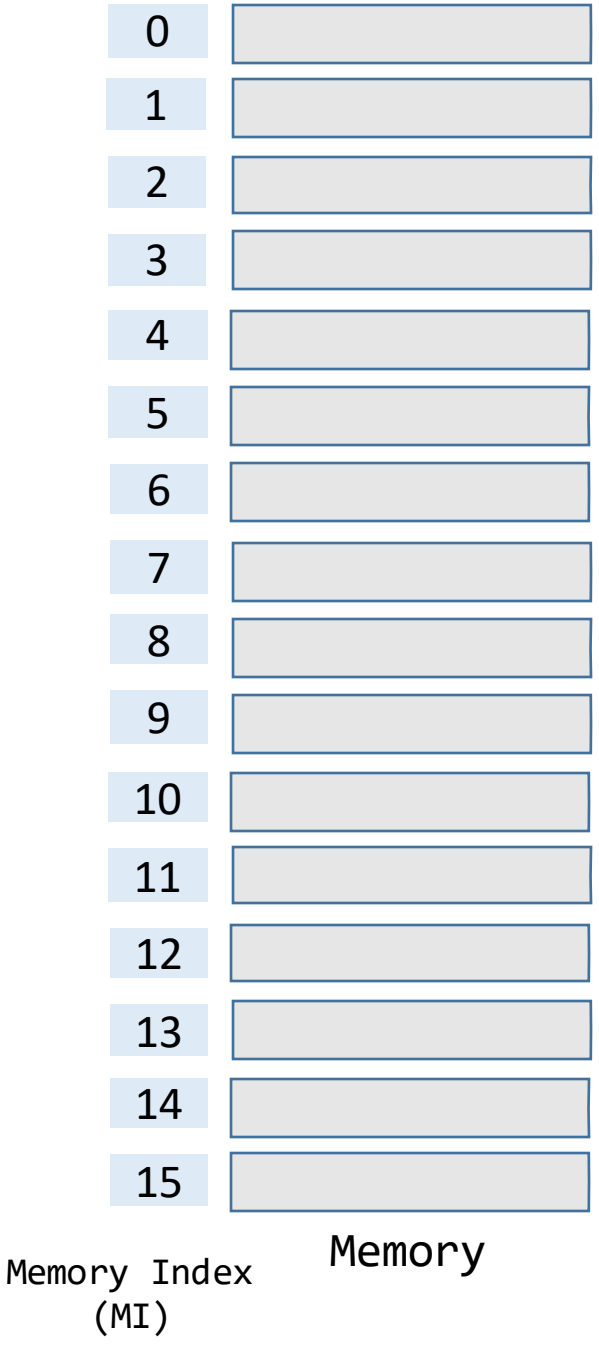
e.g. two way set-associative mapping
four way set-associative mapping
eight way set-associative mapping

- Cache is divided into a number of sets
- Each set contains a number of cache lines (or blocks)

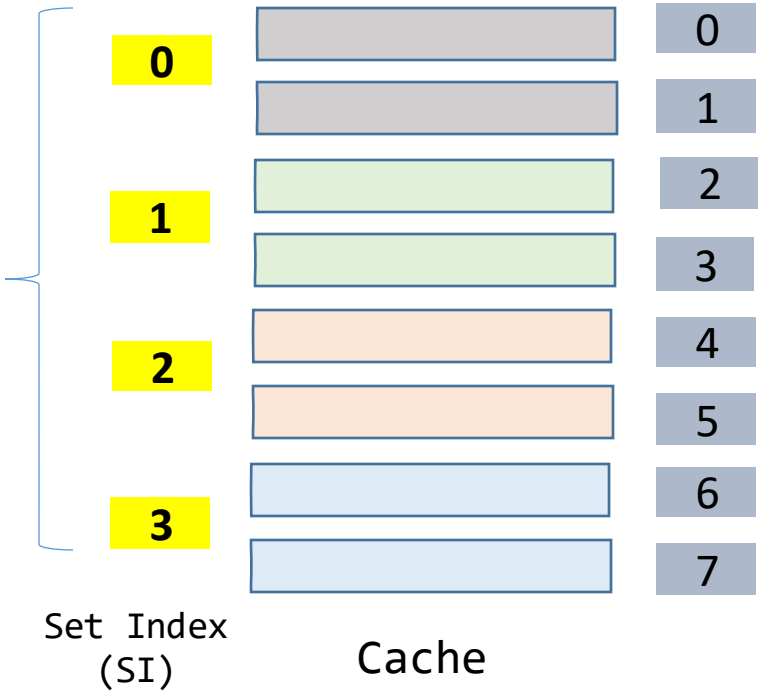
cache set number (k) = main memory block number (m) % number of sets (s)

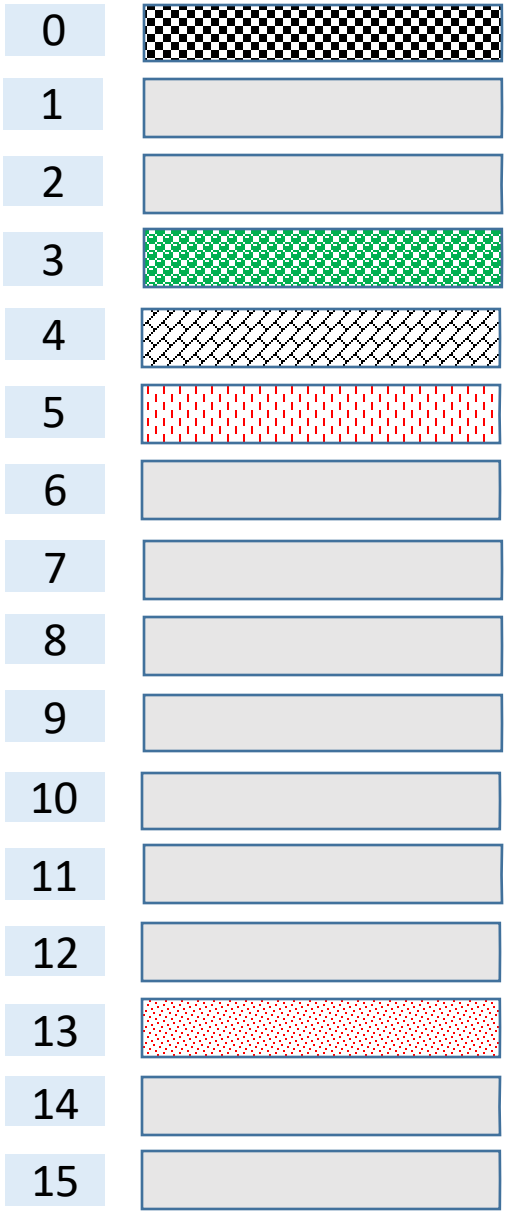
Set - Associative mapping

Two way set-associative mapping



Number of set (s)
s = 4





Example: (two way set-associative mapping)

Obtain the cache set numbers for the following memory addresses 0, 3, 4, 5, and 13

Cache set number for 0 = 0 % 4 = 0

Cache set number for 3 = 3 % 4 = 3

Cache set number for 4 = 4 % 4 = 0

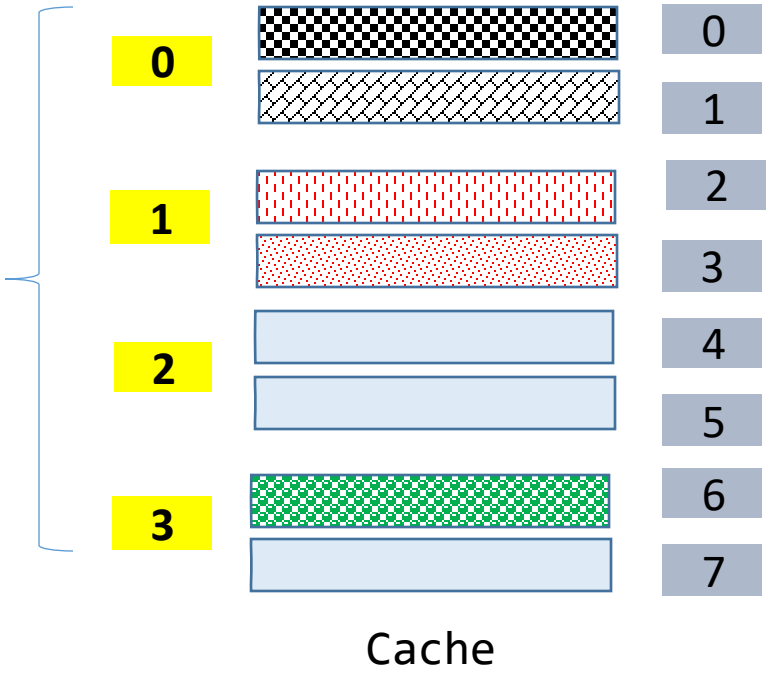
Cache set number for 5 = 5 % 4 = 1

Cache set number for 13 = 13 % 4 = 1

Number of set (s)

s = 8/2 = 4

two way set-associative mapping



Associative mapping

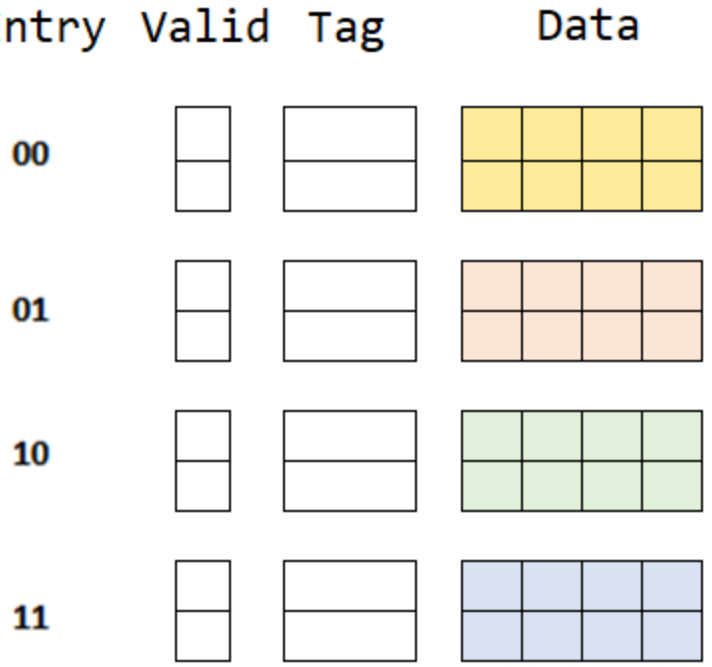
For storing and retrieving data from the cache, the memory address is broken into two components as follows:

MI	Offset
4	2

Tag	SI	Offset
2	2	2

memory address

Tag = MI - SI



Example: A list of memory requests by CPU is given below. Determine cache memory hit or miss for each case.

		00	01	10	11	
0	0000					M0
1	0001					M1
2	0010					M2
3	0011					M3
4	0100					M4
5	0101					M5
6	0110					M6
7	0111					M7
8	1000					M8
9	1001					M9
10	1010					M10
11	1011					M11
12	1100					M12
13	1101					M13
14	1110					M14
15	1111					M15

(SI)

Entry	Valid	Tag	Data
00	1	11	M12
	1	01	M4
01	1	10	M9
	1	11	M13
10	0		
	0		
11	1	00	M3
	1	11	M15

Memory Requests by CPU

Tag	SI	Offset		
1 1	0 0	0 0	Hit	M12[00]
0 1	0 0	1 0	Hit	M4[10]
1 1	1 0	0 1	Miss	M14[01]
0 1	0 1	1 1	Miss	M5[11]
1 1	1 1	0 0	Hit	M15[00]
1 0	1 0	0 0	Miss	M10[00]

Example: A list of memory requests by CPU is given below. Determine cache memory hit or miss for each case. Obtain the updated version of the cache memory. Calculate hit and miss rate.

0	0000	00	01	10	11	M0
1	0001	00	01	10	11	M1
2	0010	00	01	10	11	M2
3	0011	00	01	10	11	M3
4	0100	00	01	10	11	M4
5	0101	00	01	10	11	M5
6	0110	00	01	10	11	M6
7	0111	00	01	10	11	M7
8	1000	00	01	10	11	M8
9	1001	00	01	10	11	M9
10	1010	00	01	10	11	M10
11	1011	00	01	10	11	M11
12	1100	00	01	10	11	M12
13	1101	00	01	10	11	M13
14	1110	00	01	10	11	M14
15	1111	00	01	10	11	M15

(SI)

Entry	Valid	Tag	Data
00	1	11	M12
	1	01	M4
01	1	01	M5
	1	11	M13
10	1	11	M14
	1	10	M10
11	1	00	M3
	1	11	M15

Memory Requests by CPU

Tag	SI	Offset		
1 1	0 0	0 0	Hit	M12[00]
0 1	0 0	1 0	Hit	M4[10]
1 1	1 0	0 1	Miss	M14[01]
0 1	0 1	1 1	Miss	M5[11]
1 1	1 1	0 0	Hit	M15[00]
1 0	1 0	0 0	Miss	M10[00]

$$\text{Hit rate} = \frac{3}{6} = \frac{1}{2}$$

$$\text{Miss rate} = 1 - \text{Hit rate} = \frac{1}{2}$$

Cache Replacement Algorithms

There are some algorithms (given below) which are used to replace the existing block in the cache memory with the new fetched data.

- Least Recently Used (LRU)
- First In First Out (FIFO)
- Least Frequently Used (LFU)
- Random
- Clock algorithm

PRIMARY MEMORY

Cache Memory

Important for high-performance CPUs (Several issues)

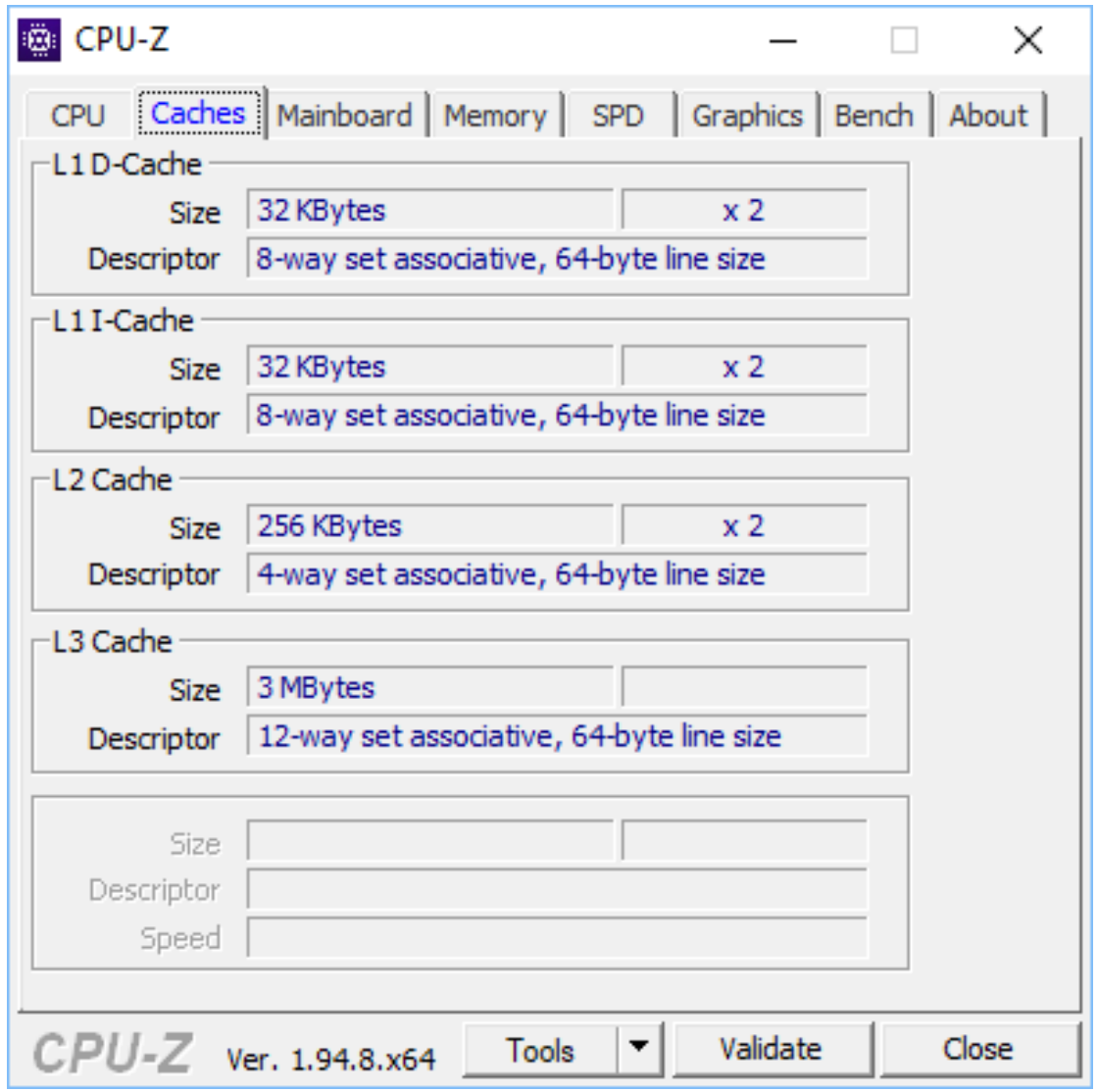
1) Cache size - the bigger the cache, the better it performs, but also more costs.

2) Size of the cache line - A 16-KB cache can be divided up into 1024 lines of 16 bytes, or 2048 lines of 8 bytes, and other combinations.

3) Cache organization - how does the cache keep track of which memory words are currently being held?

4) Whether instructions and data are kept in the same cache or different ones? **Unified cache** (instructions and data use the same cache) is a simpler design and automatically balances instruction fetches against data fetches.
Split cache (with instructions in one cache and data in the other) allows parallel accesses; a unified one does not. Also, as instructions are not modified during execution, the contents of the instruction cache never has to be written back into memory.

5) Number of caches - Chips with a primary cache on chip (L1), a secondary cache off chip but in the same package as the CPU chip (L2), and a third cache still further away (L3).


CPU-Z
CPU
Caches
Mainboard
Memory
SPD
Graphics
Bench
About
L1 D-Cache
Size
32 KBytes
x 2
Descriptor
8-way set associative, 64-byte line size
L1 I-Cache
Size
32 KBytes
x 2
Descriptor
8-way set associative, 64-byte line size
L2 Cache
Size
256 KBytes
x 2
Descriptor
4-way set associative, 64-byte line size
L3 Cache
Size
3 MBytes
Descriptor
12-way set associative, 64-byte line size
Size
Descriptor
Speed
CPU-Z
Ver. 1.94.8.x64
Tools
Validate
Close

PRIMARY MEMORY

Depending on whether it has a row of connectors on one side or both sides of the board :

Single Inline Memory Module (SIMM)

One edge connector with 72 contacts and transfer 32 bits per clock cycle

Dual Inline Memory Module(DIMM)

120 contacts on each side of the board, for a total of 240 contacts, and transfer 64 bits per clock cycle

DDR3 DIMMS

Third version of the double data-rate memories

Small Outline DIMM (SO-DIMM)

Used in notebook computers. DIMMS can have a parity bit or error correction added, but since the average error rate of a module is one error every 10 years for most garden-variety computers, error detection and correction are omitted.

https://www.youtube.com/watch?v=sLMte0Qu-0I

SECONDARY MEMORY

Memory Hierarchies

Capacity and Access time gets bigger

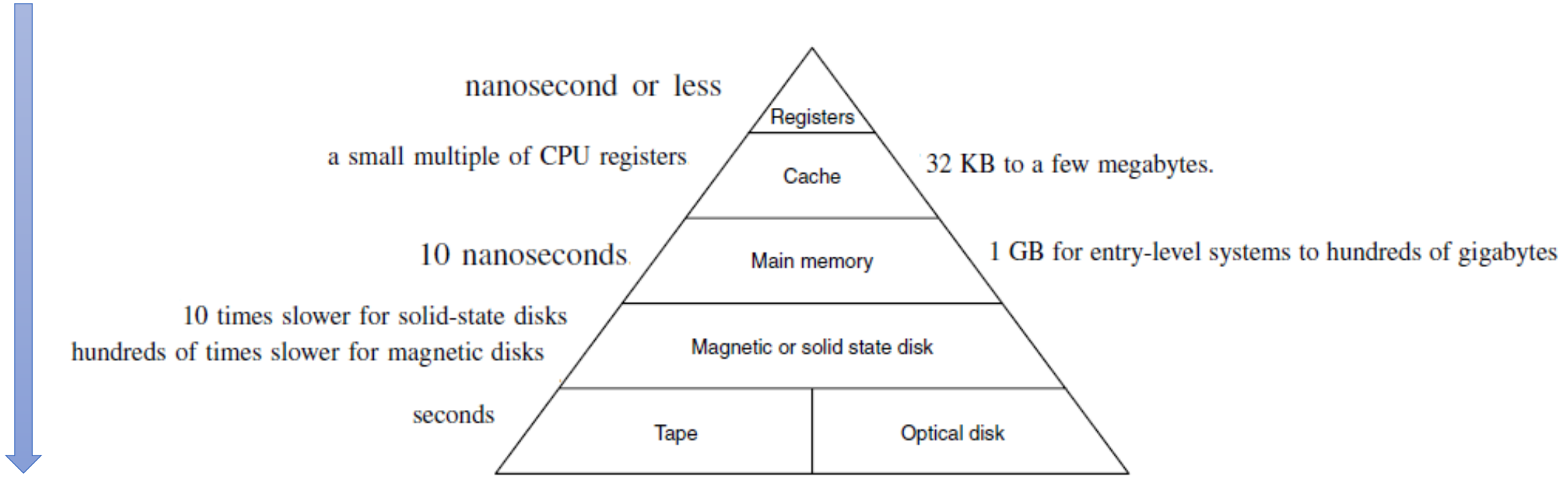


Figure 2-18. A five-level memory hierarchy.

SECONDARY MEMORY

Magnetic Disks

- Consists of one or more platters with a magnetizable coating
- A disk head containing an induction coil floats just over the surface, resting on a cushion of air

- When a positive or negative current passes through the head, it magnetizes the surface just beneath the head, aligning the magnetic particles face right or left, depending on the polarity of the drive current

- When the head passes over a magnetized area, a positive or negative current is induced in the head, making it possible to read back the previously stored bits

https://www.youtube.com/watch?v=9eMWG3fwiEU

SECONDARY MEMORY

Magnetic Disks

- **Track**
- The circular sequence of bits written as the disk makes a complete rotation called a track
- **Sector**
- Each track is divided into some sector with fixed length, typically, 512 bytes
- Preamble: allow the head to be synchronized before reading or writing
- Data bits
- Error Correcting Code: either a Hamming code, or Reed-Solomon code

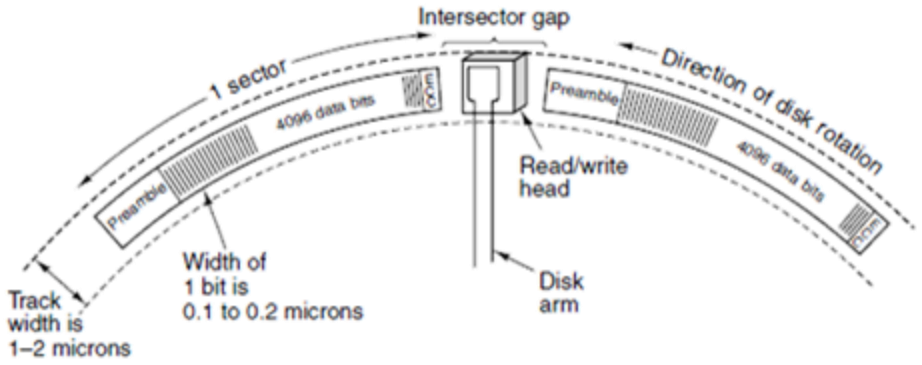


Figure 2-19. A portion of a disk track. Two sectors are illustrated.

SECONDARY MEMORY

Magnetic Disks

Most Disks

- Consist of multiple platters stacked vertically
- Each surface has its own arm and head. All arms are ganged together so they move to different radial positions all at once

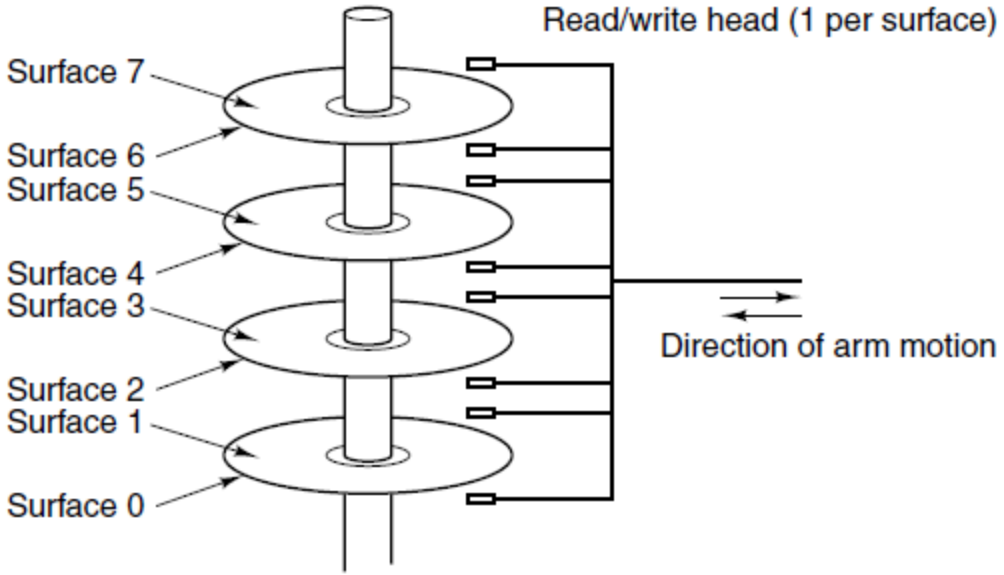


Figure 2-20. A disk with four platters.

SECONDARY MEMORY

Magnetic Disks

Disk Performance

- **Seek**

1. To read/write a sector, the arm must be moved to the right radial direction first
2. Average seek times (between random tracks) : 5 – 10 msec

- **Rotational Latency**

1. The delay until the desired sector rotates under the head once after the head is positioned radially
2. For disk rotation speed 5400RPM 7200RPM 10,800RPM, average delay is 3- 6 ms

- **Transfer Time**

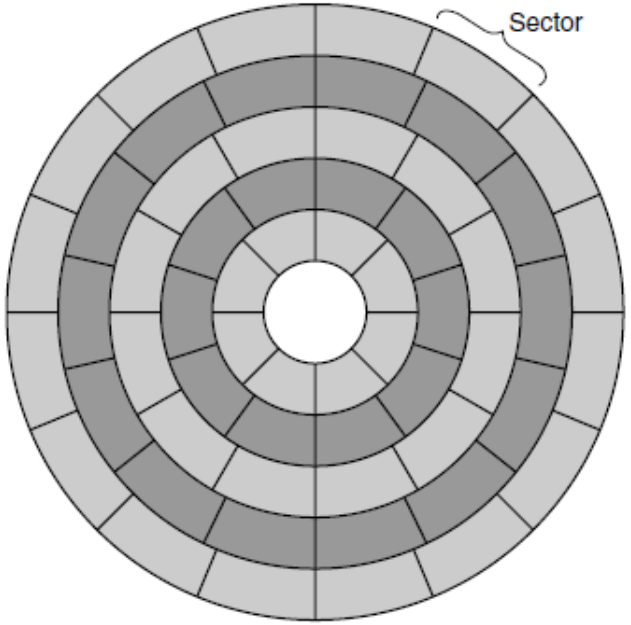
1. Typical internal transfer rate of 150 MB/sec, a 512-byte sector takes about 3.5 μ sec

- **Disk Controller**

1. Associated with each drive is a disk controller, a chip that controls the drive
2. Accepting commands from software, such as read, write, format
3. Controlling the arm motion; detecting and correcting errors
4. Converting 8-bit bytes read from memory into a serial bit stream and vice versa

SECONDARY MEMORY

Magnetic Disks



A disk with five zones. Each zone has many tracks.

- Cylinders are divided into zones (typically 10 to 30 per drive)
- The number of sectors per track is increased in each zone moving outward from the innermost track.
- All sectors are the same size

SECONDARY MEMORY

IDE/SCSI Disks

Integrated Drive Electronic (IDE)

- The controller is closely integrated with the drives
- 4 bits for head, 6 bits for sector, 10 bits for cylinder, the max drive could have 63 sector, 16 heads and 1024 cylinders
- Total 1,032,192 sectors, capacity 504 MB=1032192*512

Small Computer System Interface (SCSI)

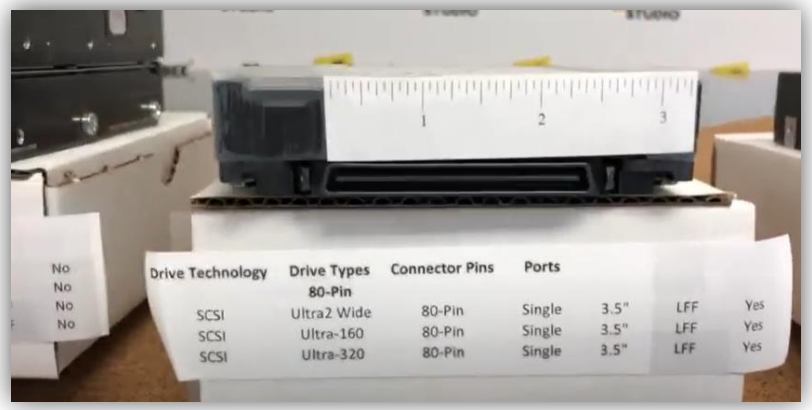
1. Same organization of cylinders, tracks and sector ;
2. Different interface and much higher transfer rates
3. More than just a hard disk interface and it is a bus to which a SCSI controller and up to 7 devices can be attached, such as SCSI disk, scanner, tape units, and others
4. SCSI allows all the devices to run at once, improving performance
5. IDE allows only one active device at a time

SECONDARY MEMORY

IDE/SCSI Disks

Name	Data bits	Bus MHz	MB/sec
SCSI-1	8	5	5
Fast SCSI	8	10	10
Wide Fast SCSI	16	10	20
Ultra SCSI	8	20	20
Wide Ultra SCSI	16	20	40
Ultra2 SCSI	8	40	40
Wide Ultra2 SCSI	16	40	80
Wide Ultra3 SCSI	16	80	160
Wide Ultra4 SCSI	16	160	320
Wide Ultra5 SCSI	16	320	640

Some of the possible SCSI parameters.



https://www.youtube.com/watch?v=A3q1x3_7tTg