Computer Architecture and System Laboratory

處理器設計與實作

實習講義

編撰者
助教：黃冠霖、蘇郁翔、鄭基漢、曾微中、金育涵
暨成大電通所計算機架構與系統研究室CASLAB

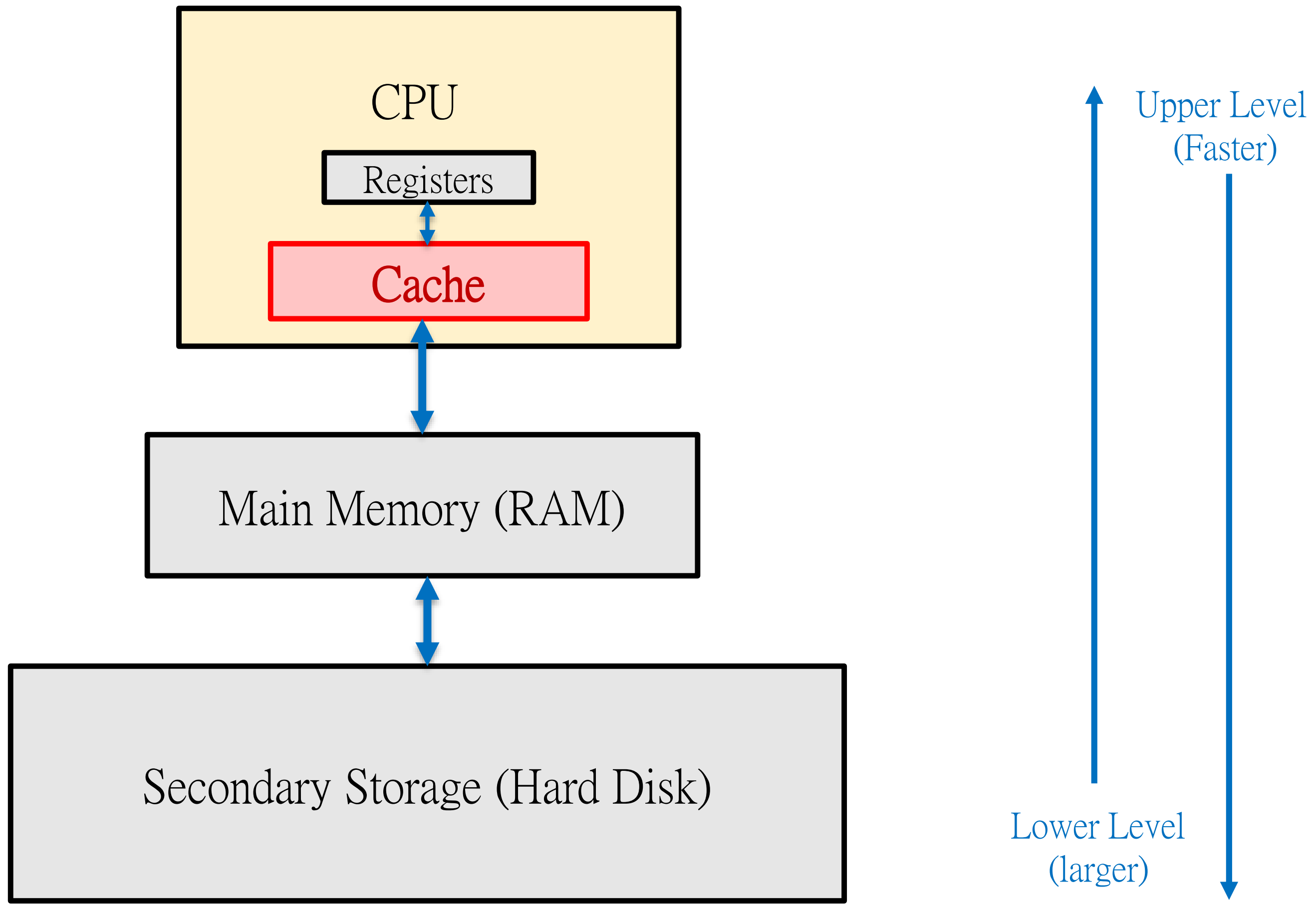
國立成功大學電機系與電腦與通信工程研究所

LAB 9
Simulation Of I-Cache
&
Set-Associative D-cache

實驗目的

1. 認識CPU的Memory Hierarchy
2. 了解cache的運作原理、特性與架構
3. 了解cache之write policies
4. 練習ARMV7A CPU Simulator中icache的實作與應用
5. 實作簡易Set-Associative Cache修改操作

Levels of Memory Hierarchy



The Principle of Locality

- **Temporal Locality** (Locality in Time):

  If an item is referenced, it will tend to be referenced again soon.

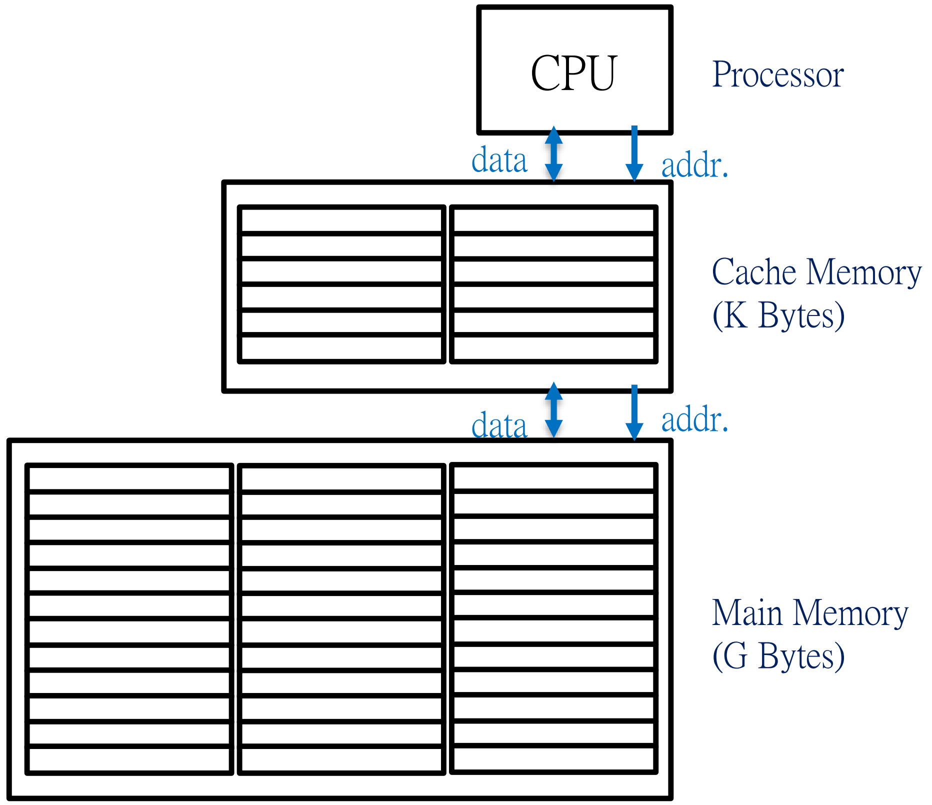
  (Example: loop, reuse)
- **Spatial Locality** (Locality in space):

  If an item is referenced, items whose addresses are close by tend to be referenced soon.

  (Example: array access, straight line code)

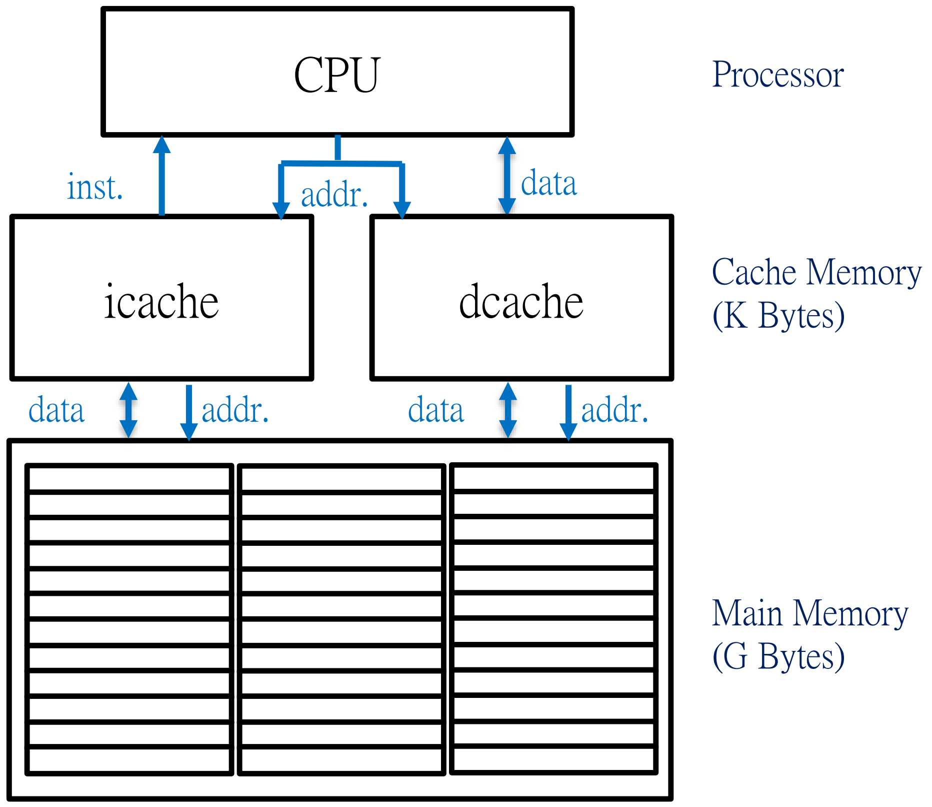
Introduction to Cache(1/2)

- A smaller, faster memory which stores copies of the data from frequently used main memory locations. (**Temporal Locality**)



Introduction to Cache(2/2)

- In a Harvard architecture of caches, instruction and data are stored separately.
  - **Instructions (I-Cache)**
  - **Data (D-Cache)**



Cache Performance

- **Hit Rate**: Fraction of hits in Cache
- Miss Rate : 1 – (Hit Rate)
- **Hit Time**: Time to access Cache
- **Miss Penalty**: Time to replace a block from lower level
- **Average memory-access time** (AMAT)

  = Hit Time + Miss rate x Miss penalty

Hit vs. Misses

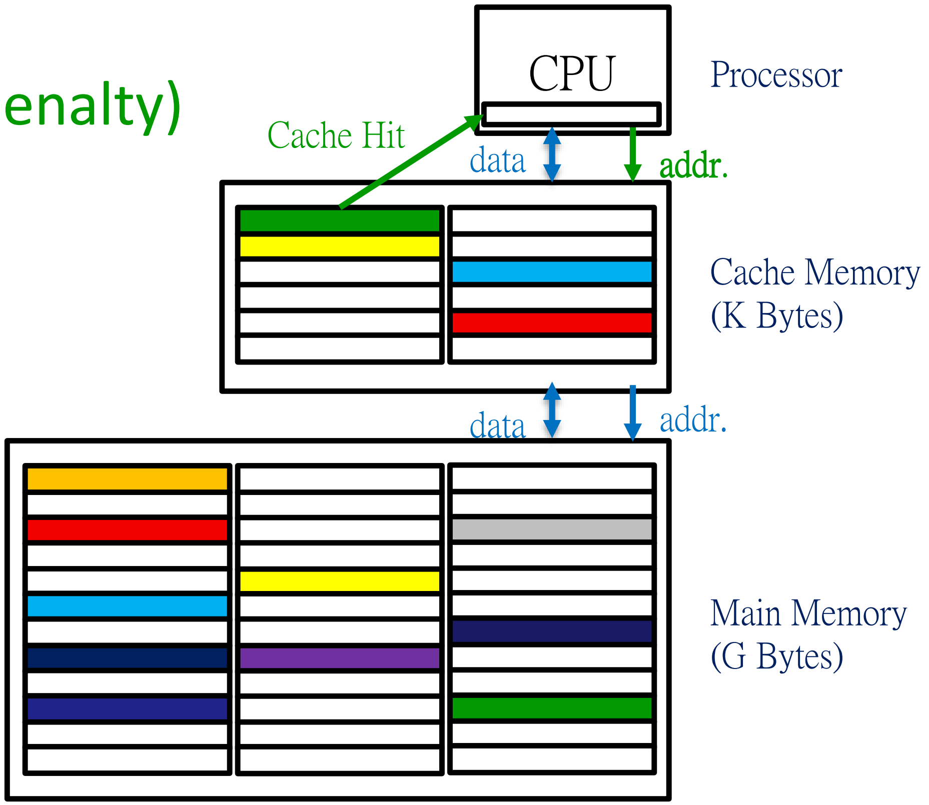
- Read hits
  - this is what we want!
- Read misses
  - stall the CPU, fetch block from memory, deliver to cache, restart
- Write hits
  - can replace data in cache and memory (write-through)
  - write the data only into the cache (write-back)
- Write misses
  - data at the missed-write location is loaded to cache, followed by a write-hit operation.(write-allocate)
  - data at the missed-write location is not loaded to cache, and is written directly to the backing store.(write-around)

D-cache vs. I-cache

- CPU has the demand for writing and reading data memory, while it has no requirement for writing instruction memory. (Modifying instruction memory is prohibited)
- As a result , d-cache will be writed and read by CPU and i-cache will be only read.

Cache Read Operation(1/5)

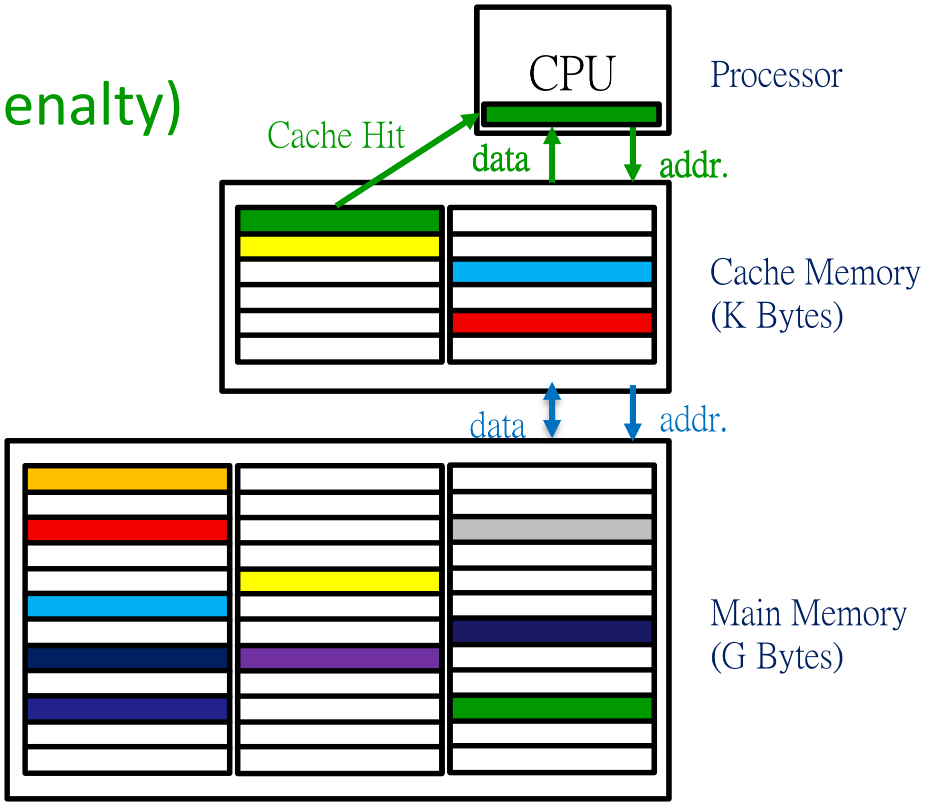
- CPU sends an address to Cache
- **Hit** : Data in Cache (no penalty)
- Miss: Data not in Cache
  (miss penalty)



Cache Read Operation(2/5)

- CPU sends an address to Cache
- **Hit** : Data in Cache (no penalty)
- Miss: Data not in Cache

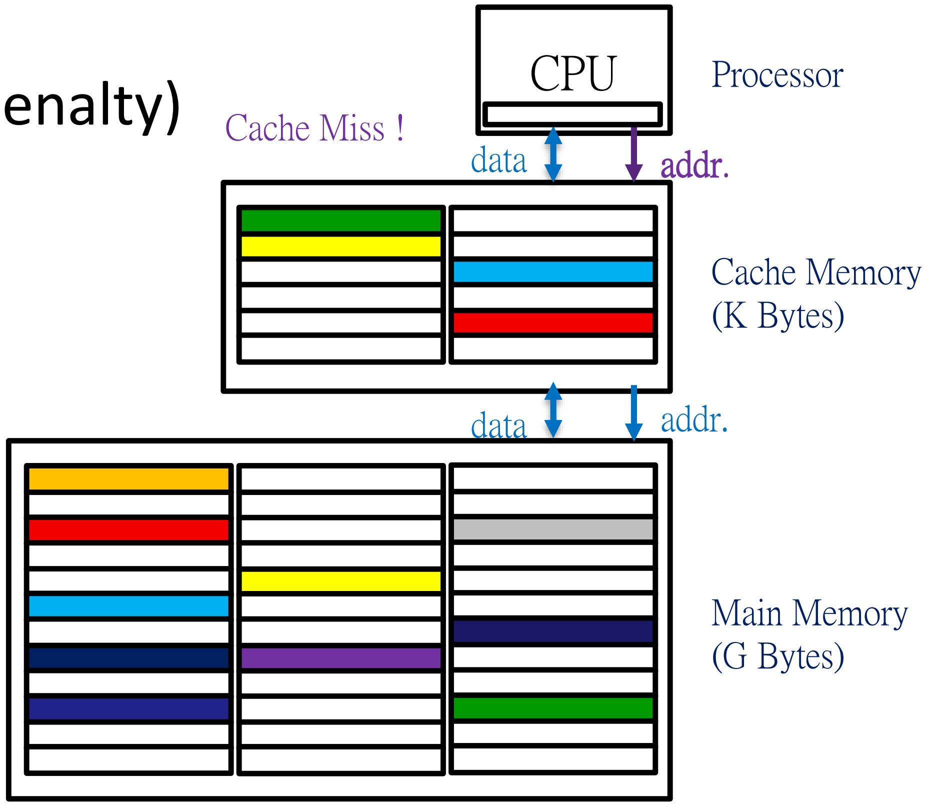
  (miss penalty)



Cache Read Operation(3/5)

- CPU sends an address to Cache
- Hit : Data in Cache (no penalty)
- **Miss**: Data not in Cache

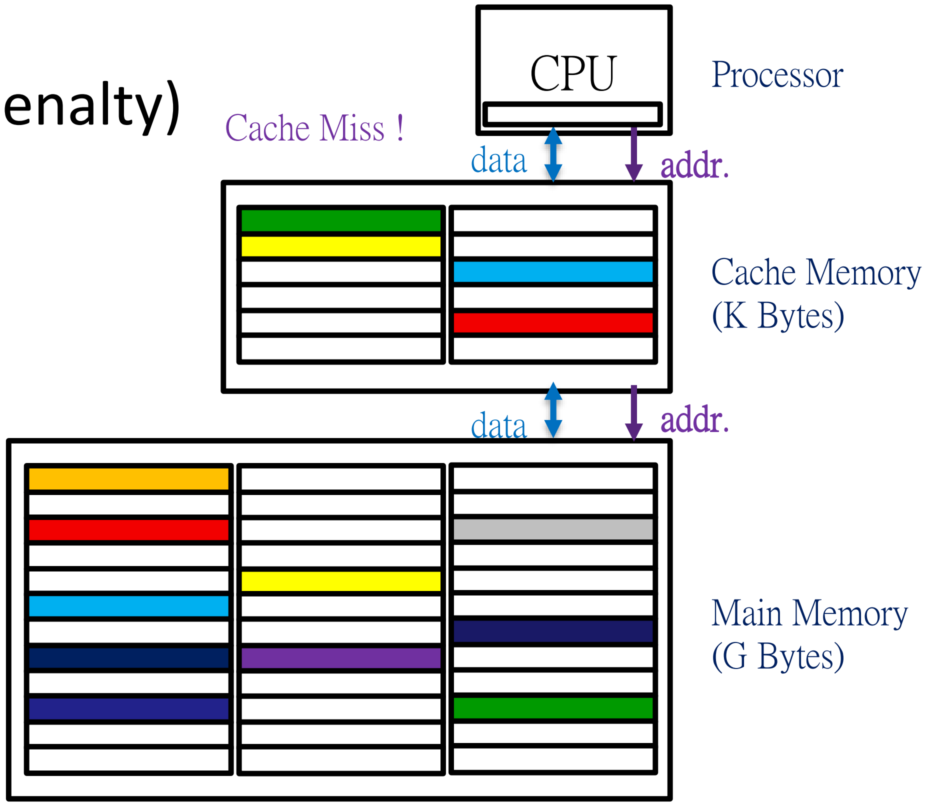
  (miss penalty)



Cache Read Operation(4/5)

- CPU sends an address to Cache
- Hit : Data in Cache (no penalty)
- **Miss**: Data not in Cache

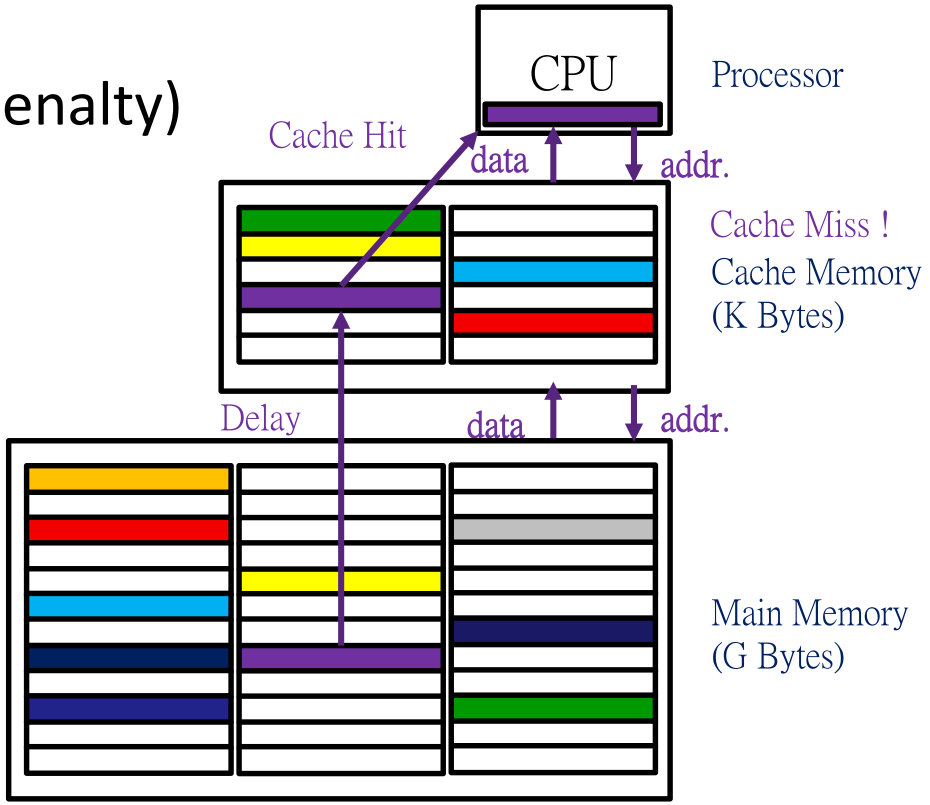
  (miss penalty)



Cache Read Operation(5/5)

- CPU sends an address to Cache
- Hit : Data in Cache (no penalty)
- **Miss**: Data not in Cache

  (miss penalty)



Cache Line

- **Cache Line**: unit of memory transfer between two levels in a memory hierarchy. Also called a block.
- Rather than reading a single word or byte from main memory at a time ,each cache entry is holds a certain number of words.(**Spatial Locality**)
- For example:
  - Line size = 1 word: one entry, one word.
  - Line size = N words: one entry, N words.

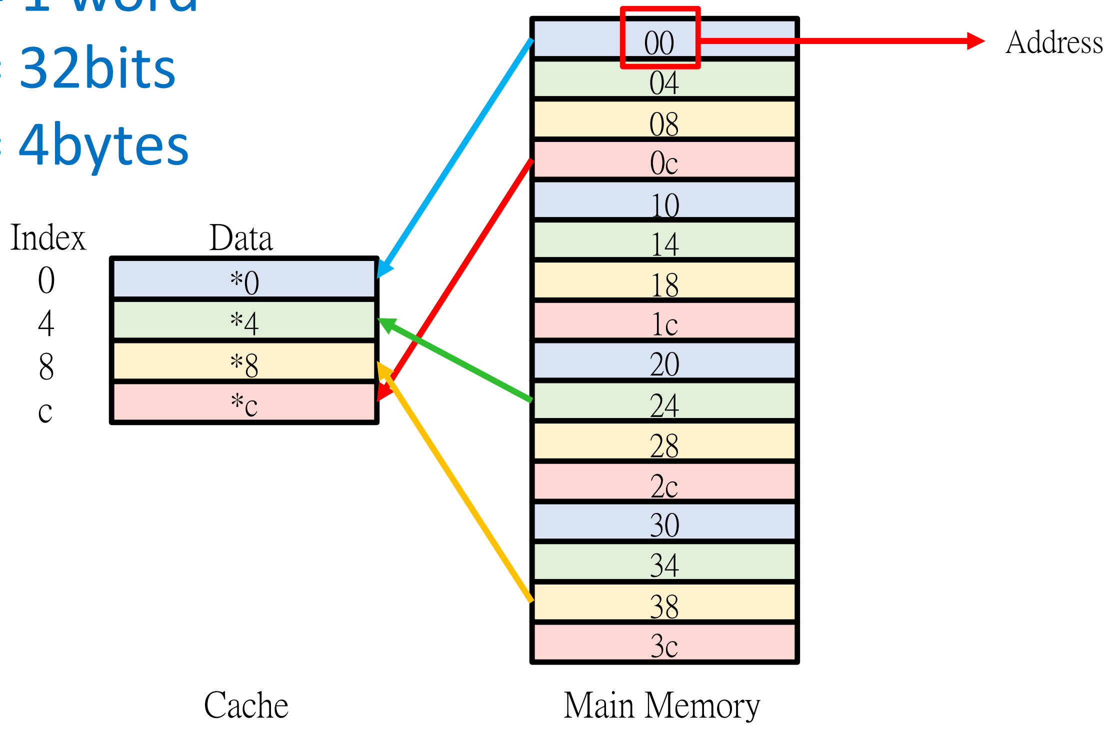
Associativity

- **Associativity**: The replacement policy decides where a copy of a particular entry of main memory will go.
- For example:
  - Direct mapped: each cache line has only one way to go.
  - 4-way set associative: each cache line has 4 ways to go.

Direct Mapped Example 1

Example 1:

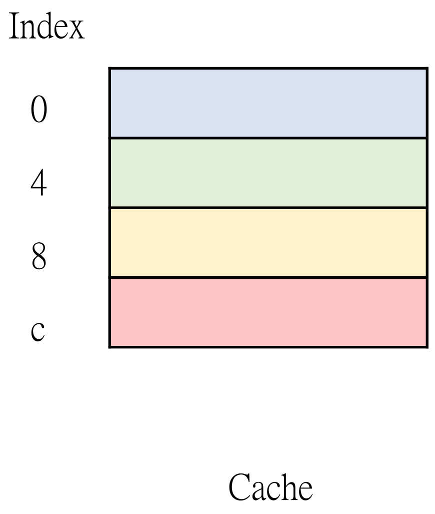
Line size = 1 word
= 32bits
= 4bytes



Direct Mapped Example 1(1/8)

The sequence of memory access: 0<u>0</u>, 0<u>4</u>, 0<u>8</u>, 0<u>c</u>, 1<u>0</u>

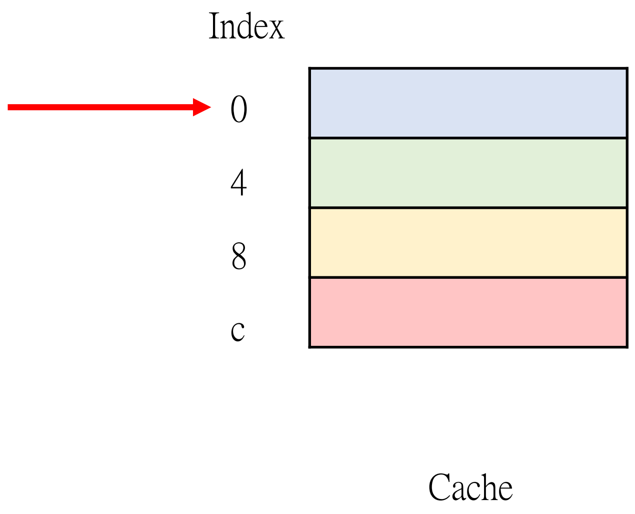
Memory Block	Hit/Miss



Direct Mapped Example 1(2/8)

The sequence of memory access: 00, 04, 08, 0c, 10

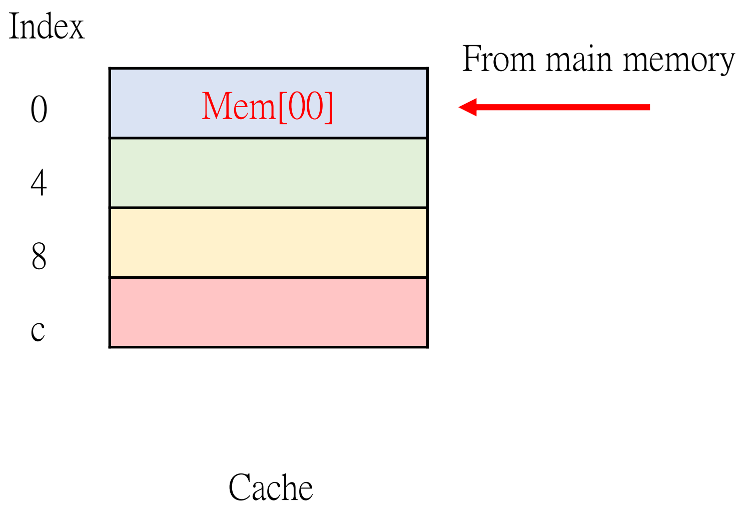
Memory Block	Hit/Miss
00	Miss



Direct Mapped Example 1(3/8)

The sequence of memory access: 00, 04, 08, 0c, 10

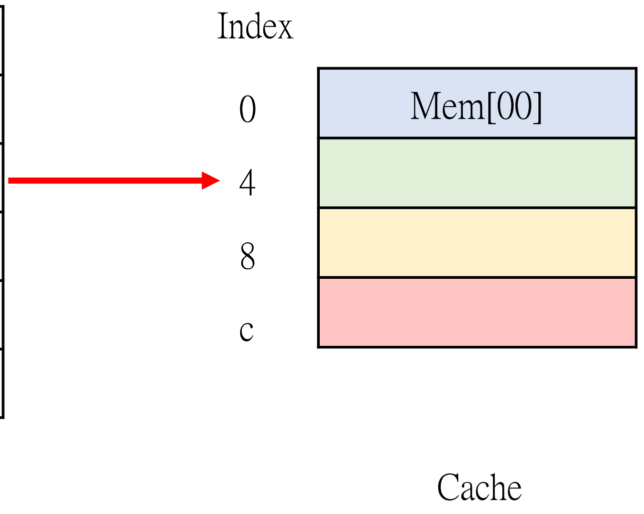
Memory Block	Hit/Miss
00	Miss



Direct Mapped Example 1(4/8)

The sequence of memory access: 00, 04, 08, 0c, 10

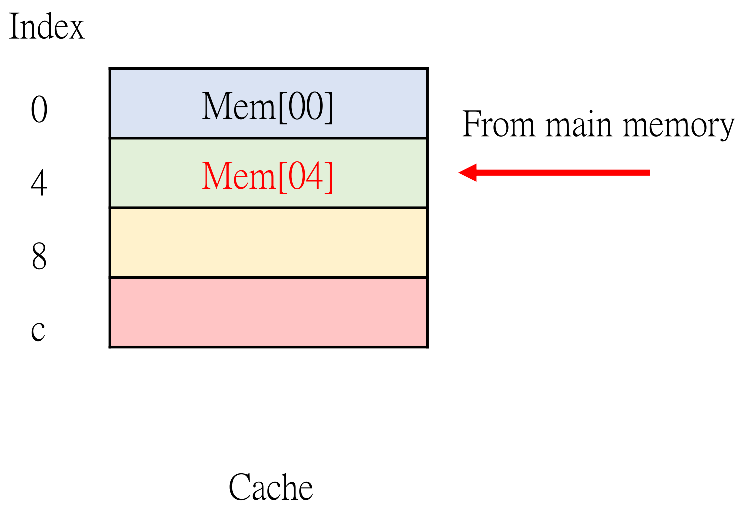
Memory Block	Hit/Miss
00	Miss
04	Miss



Direct Mapped Example 1(5/8)

The sequence of memory access: 00, 04, 08, 0c, 10

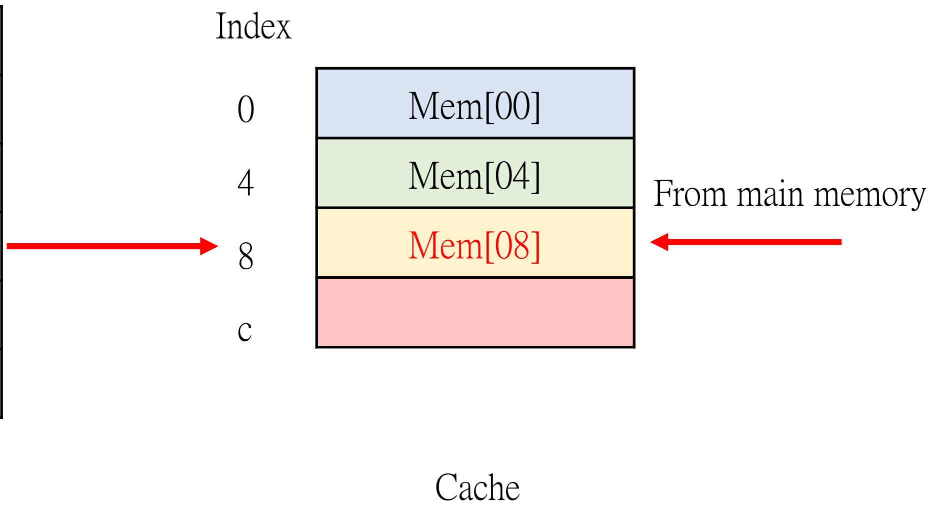
Memory Block	Hit/Miss
00	Miss
04	Miss



Direct Mapped Example 1(6/8)

The sequence of memory access: 00, 04, 08, 0c, 10

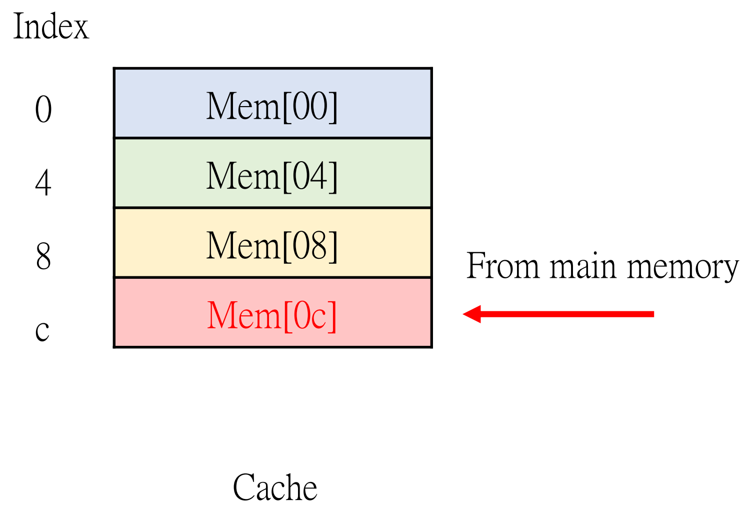
Memory Block	Hit/Miss
00	Miss
04	Miss
08	Miss



Direct Mapped Example 1(7/8)

The sequence of memory access: 00, 04, 08, 0c, 10

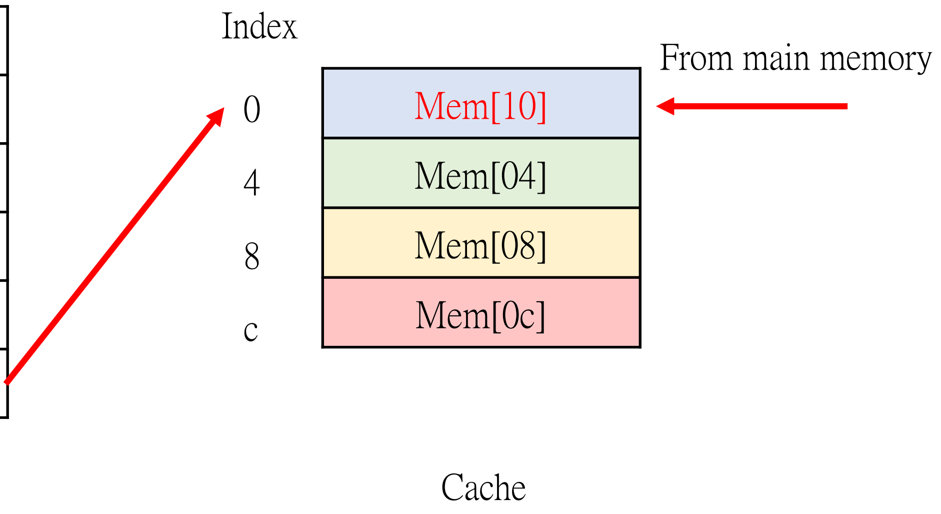
Memory Block	Hit/Miss
00	Miss
04	Miss
08	Miss
0c	Miss



Direct Mapped Example 1(8/8)

The sequence of memory access: 00, 04, 08, 0c, 10

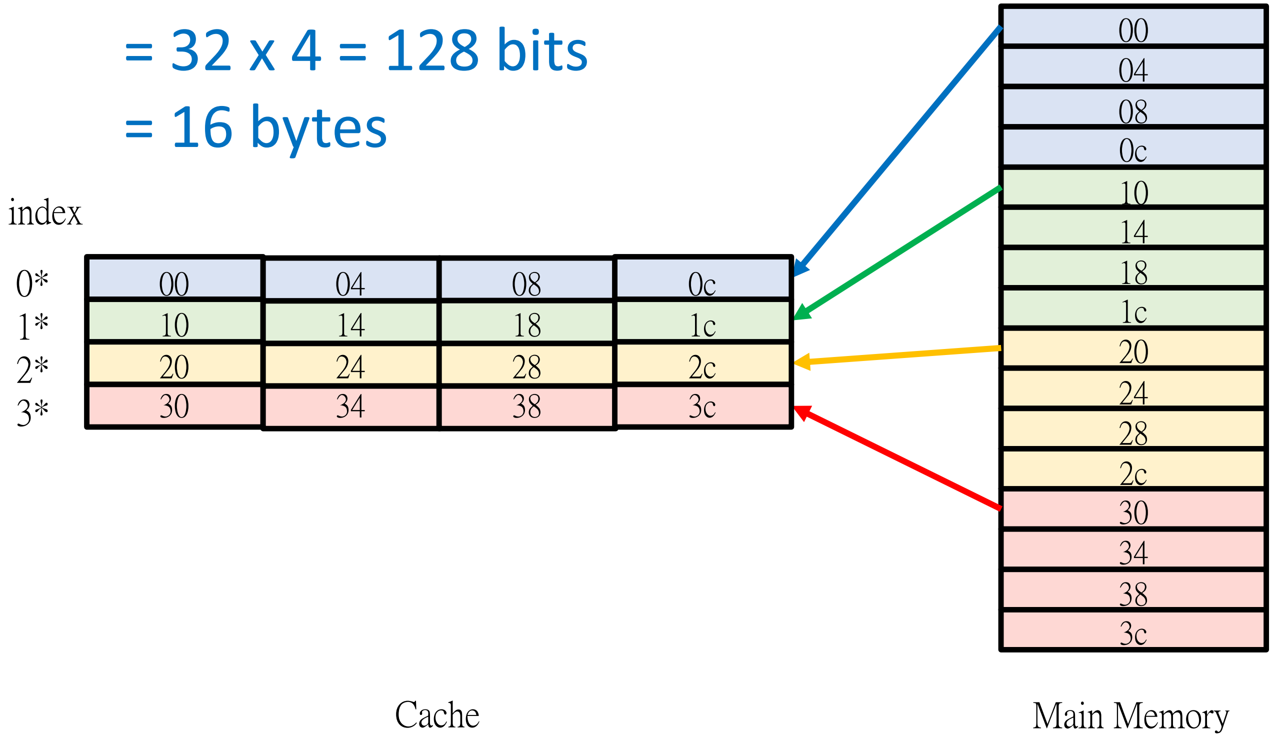
Memory Block	Hit/Miss
00	Miss
04	Miss
08	Miss
0c	Miss
10	Miss



Direct Mapped Example 2

Example 2:

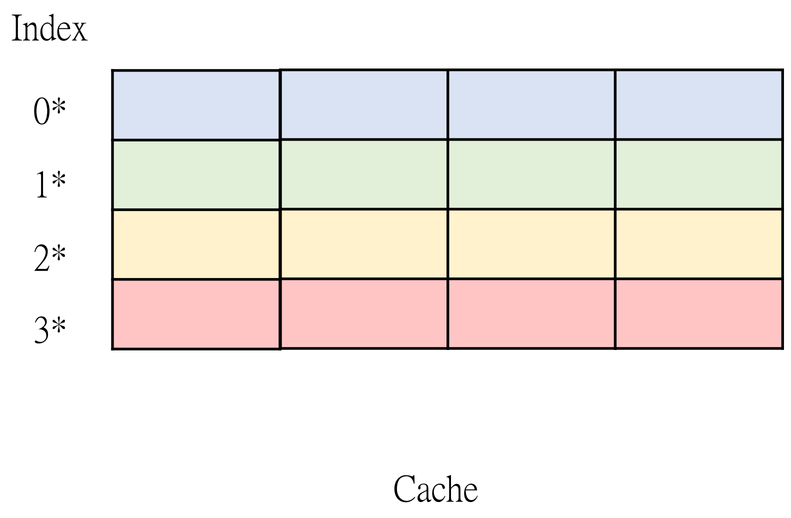
Line size = 4 words
= 32 x 4 = 128 bits
= 16 bytes



Direct Mapped Example 2(1/7)

The sequence of memory access: 00, 04, 08, 0c, 10

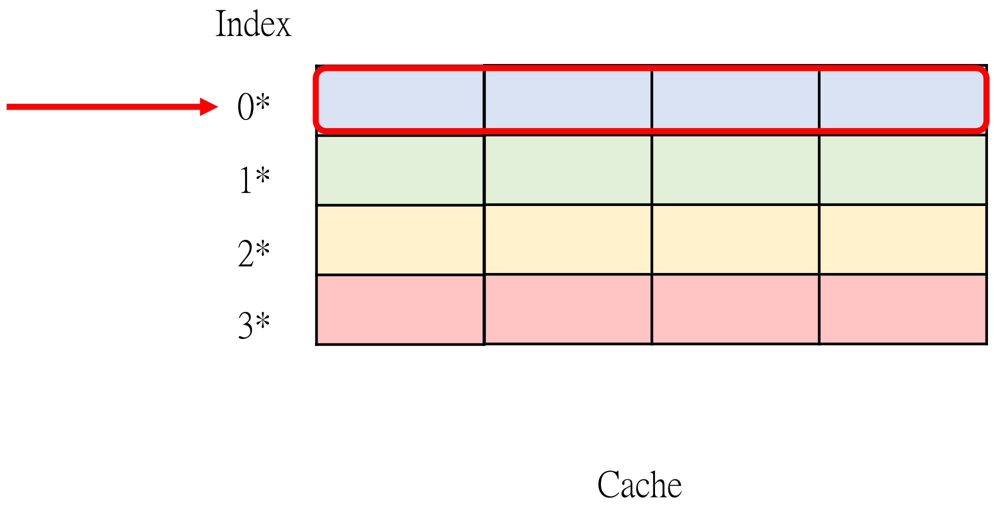
Mem Block	Hit/Miss



Direct Mapped Example 2(2/7)

The sequence of memory access: 00, 04, 08, 0c, 10

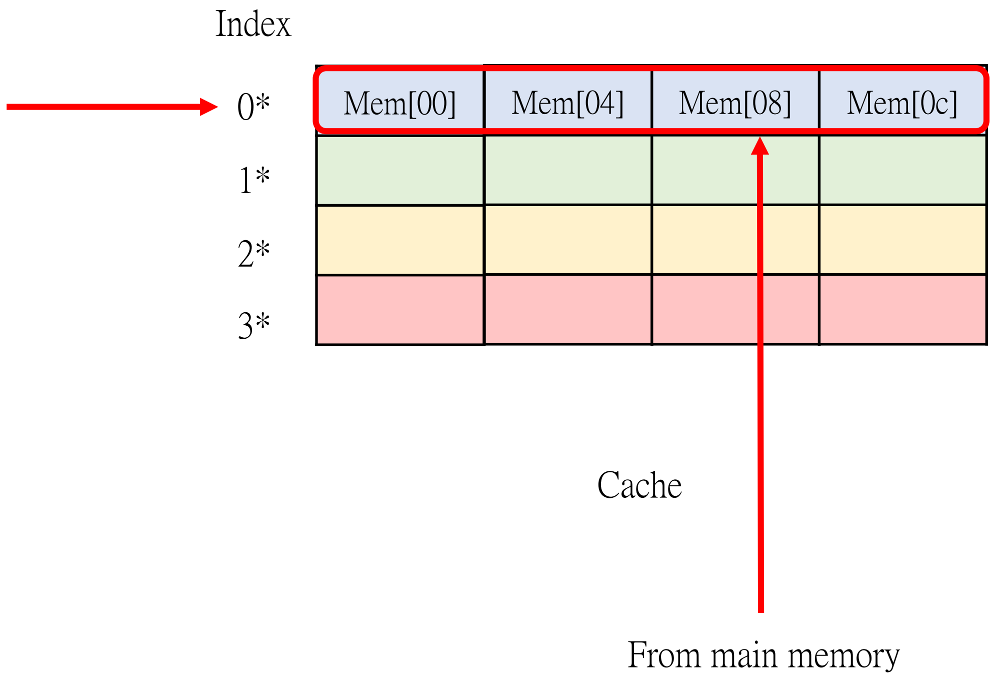
Mem Block	Hit/Miss
00	Miss



Direct Mapped Example 2(3/7)

The sequence of memory access: 00, 04, 08, 0c, 10

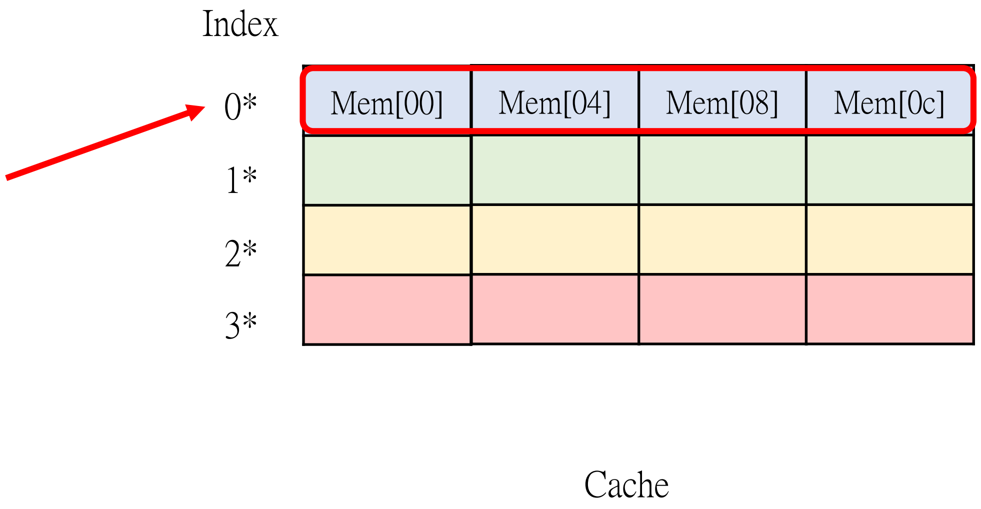
Mem Block	Hit/Miss
00	Miss



Direct Mapped Example 2(4/7)

The sequence of memory access: 00, 04, 08, 0c, 10

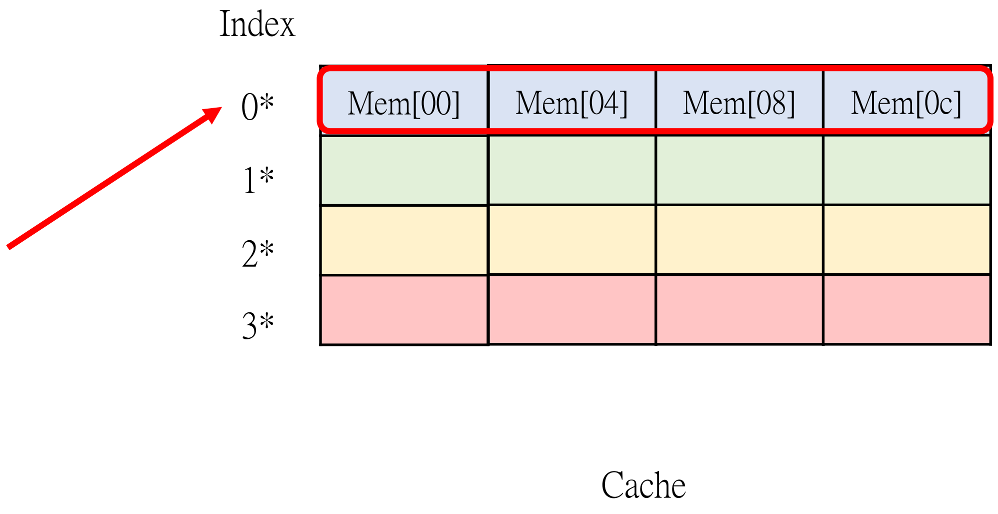
Mem Block	Hit/Miss
00	Miss
04	Hit



Direct Mapped Example 2(5/7)

The sequence of memory access: 00, 04, 08, 0c, 10

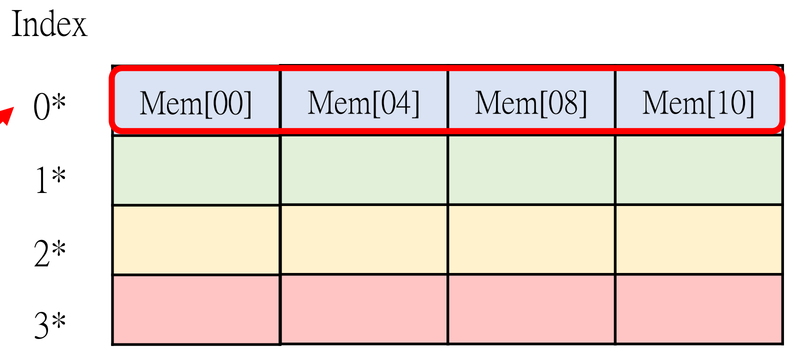
Mem Block	Hit/Miss
00	Miss
04	Hit
08	Hit



Direct Mapped Example 2(6/7)

The sequence of memory access: 00, 04, 08, 0c, 10

Mem Block	Hit/Miss
00	Miss
04	Hit
08	Hit
0c	Hit

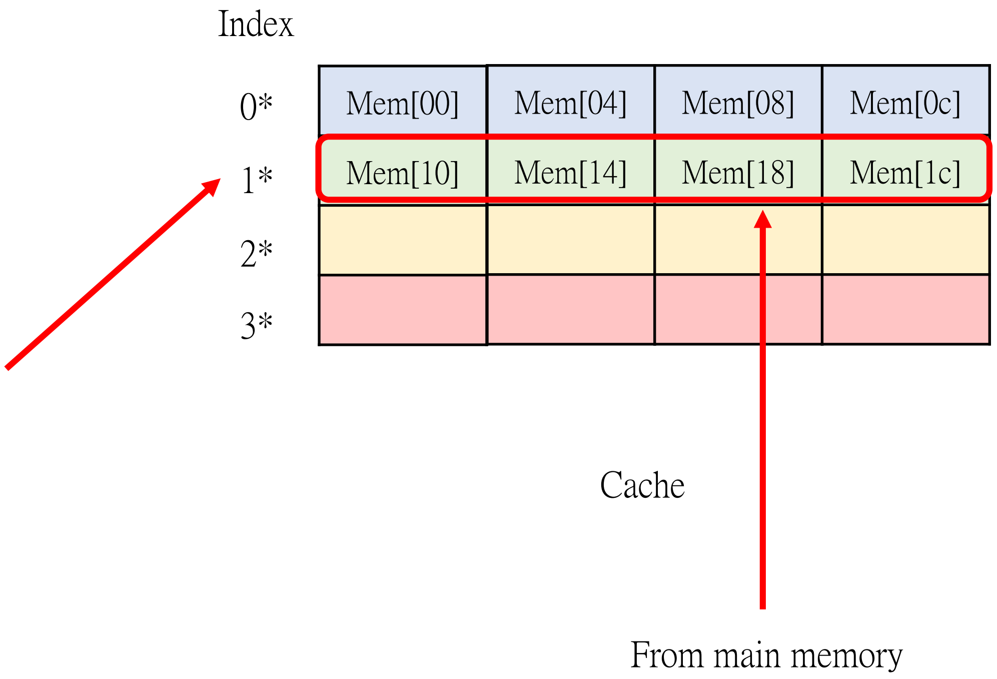


Cache

Direct Mapped Example 2(7/7)

The sequence of memory access: 00, 04, 08, 0c, 10

Mem Block	Hit/Miss
00	Miss
04	Hit
08	Hit
0c	Hit
10	Miss



Cache Architecture(1/4)

Cache Addressing mode (aligned to a word for instruction):

Block Address (32-bit)			
Tag	Index	word	00

Index : Decide which entry of cache should be accessed

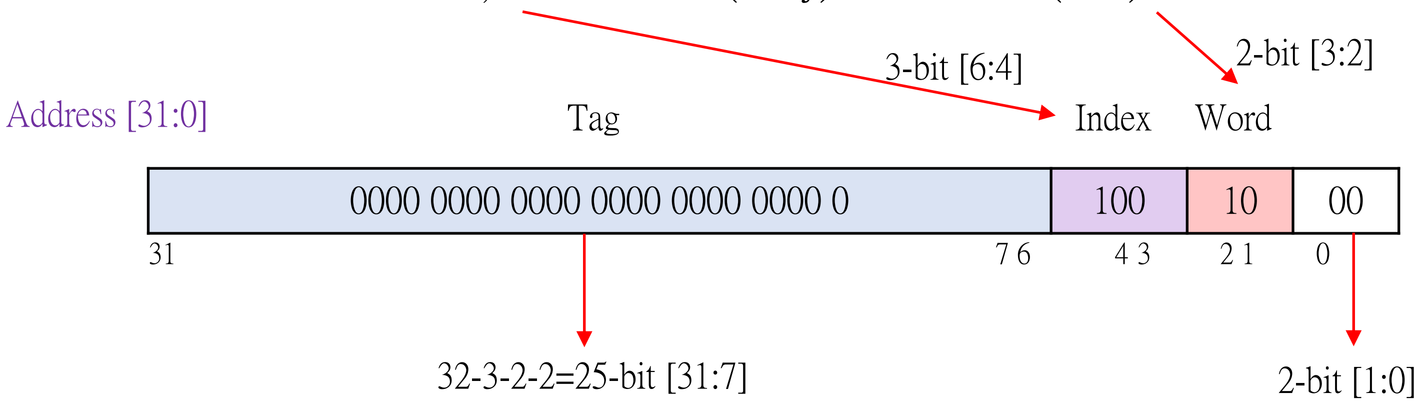
Tag : Check if the cache access is a hit or not

Word(offset) : Decide which data of entry to output

How do we decide number of bits in each part?

Cache Architecture(2/4)

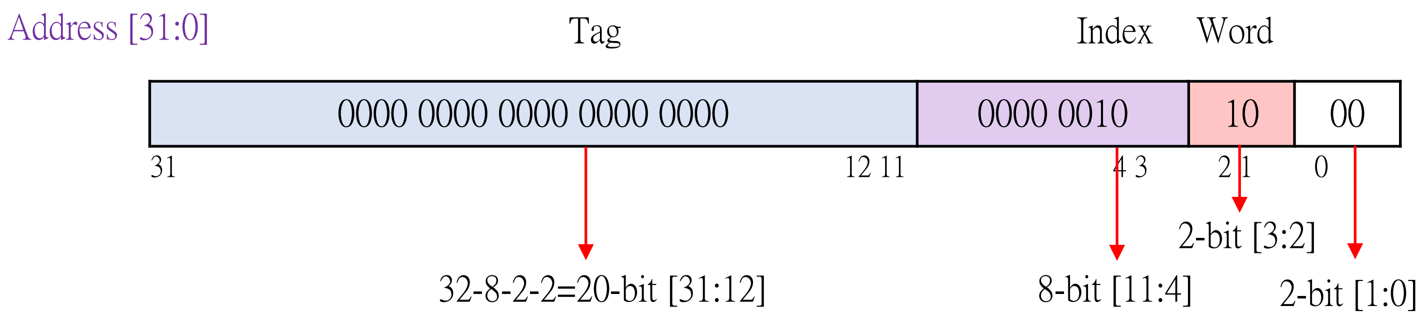
This is a Cache with 8 entries, and each line (entry) has 4 words (data).



Index	Valid	Tag	Word=0	Word=1	Word=2	Word=3
000	1	0008···	0x00001c0	0x00001c4	0x00001c8	0x00001cc
001	0					
010	0					
011	1	00c0···	0x0024000	0x0024004	0x0024008	0x002400c
100	1	0000···	0x1c00000	0x1x00004	0x1c00008	0x1c0000c
101	0					
110	0					
111	0					

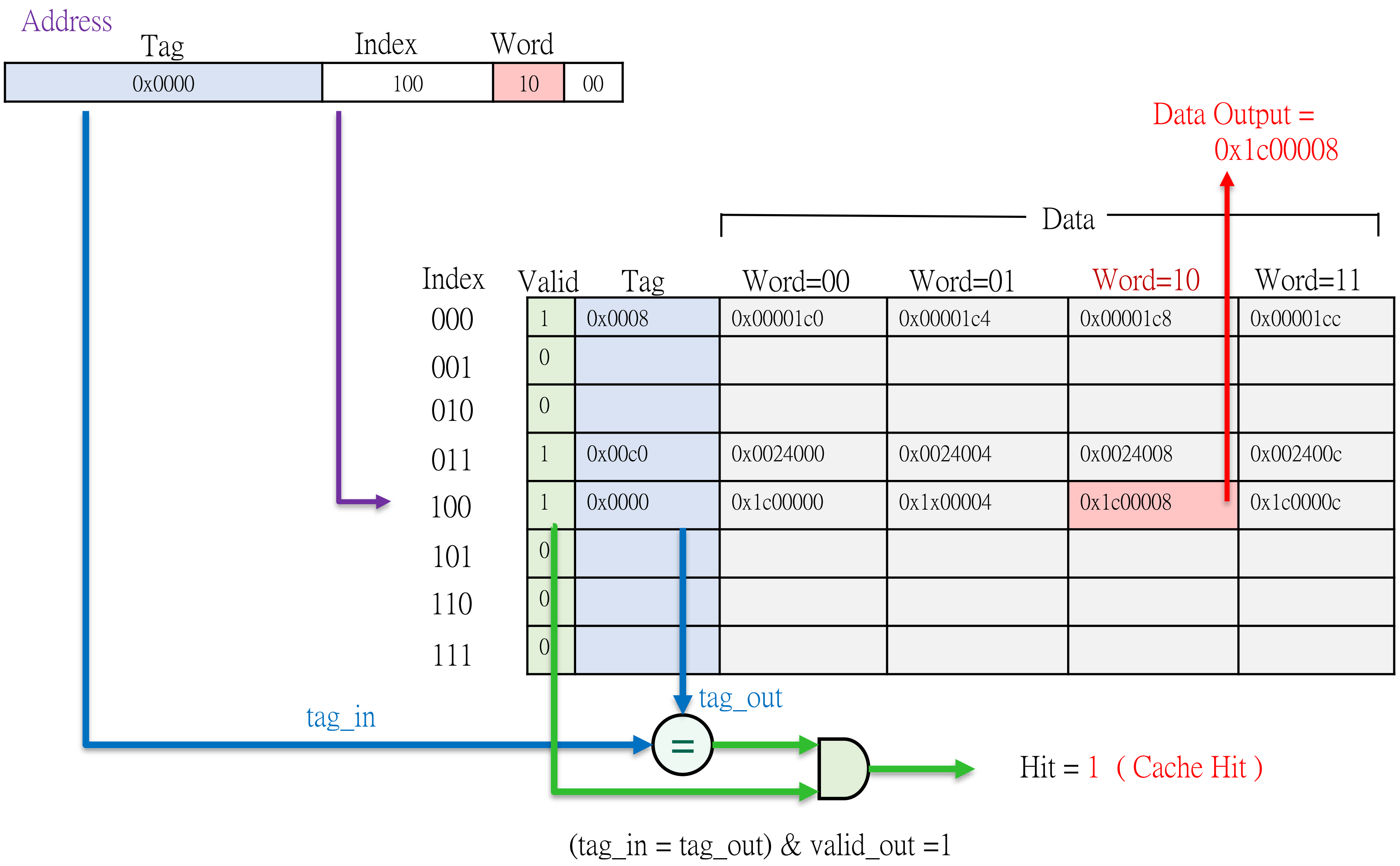
Cache Architecture(3/4)

If a Cache with 256 entries, and each line (entry) has 4 words (data) ⋯



Index	Valid	Tag	Word=0	Word=1	Word=2	Word=3
0000 0000	1	0008⋯	0x00001c0	0x00001c4	0x00001c8	0x00001cc
0000 0001	0					
.	0					
.	1	00c0⋯	0x0024000	0x0024004	0x0024008	0x002400c
.	1	0000⋯	0x1c00000	0x1x00004	0x1c00008	0x1c0000c
.	0					
	0					
1111 1111	0					

Cache Architecture(4/4)



Index	Valid	Tag	Word=00	Word=01	Word=10	Word=11
000	1	0x0008	0x00001c0	0x00001c4	0x00001c8	0x00001cc
001	0					
010	0					
011	1	0x00c0	0x0024000	0x0024004	0x0024008	0x002400c
100	1	0x0000	0x1c00000	0x1x00004	0x1c00008	0x1c0000c
101	0					
110	0					
111	0					

Write Policy

- Write hit
  - Write-through (WT)
  - Write –back (WB)
- Write miss
  - Write-allocate (or write allocation)
  - Write around

Write Policy

- Write-hit policies
  - Write-through (also called store-through)
    - Write to main memory whenever a write is performed to the cache.
  - Write –back (also called store-in or copy-back)
    - Write to the main memory when the modified data in cache is evicted.

Write Policy

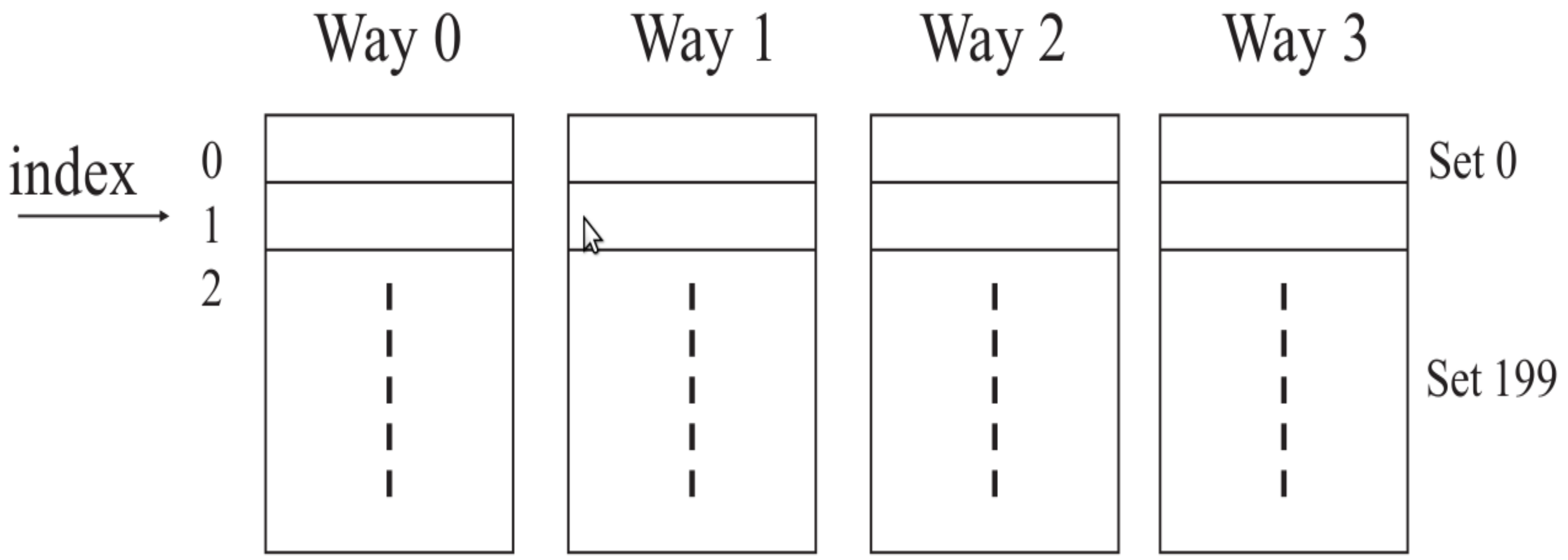
	Write-Through	Write-Back
Policy	Data written to cache block also written to lower-level memory	Write data to the cache only, copy back when replacing a dirty copy
Debug	Easy	Hard
Do read misses produce writes?	No	Yes
Do repeated writes make it to lower level?	Yes	No

Write Policy

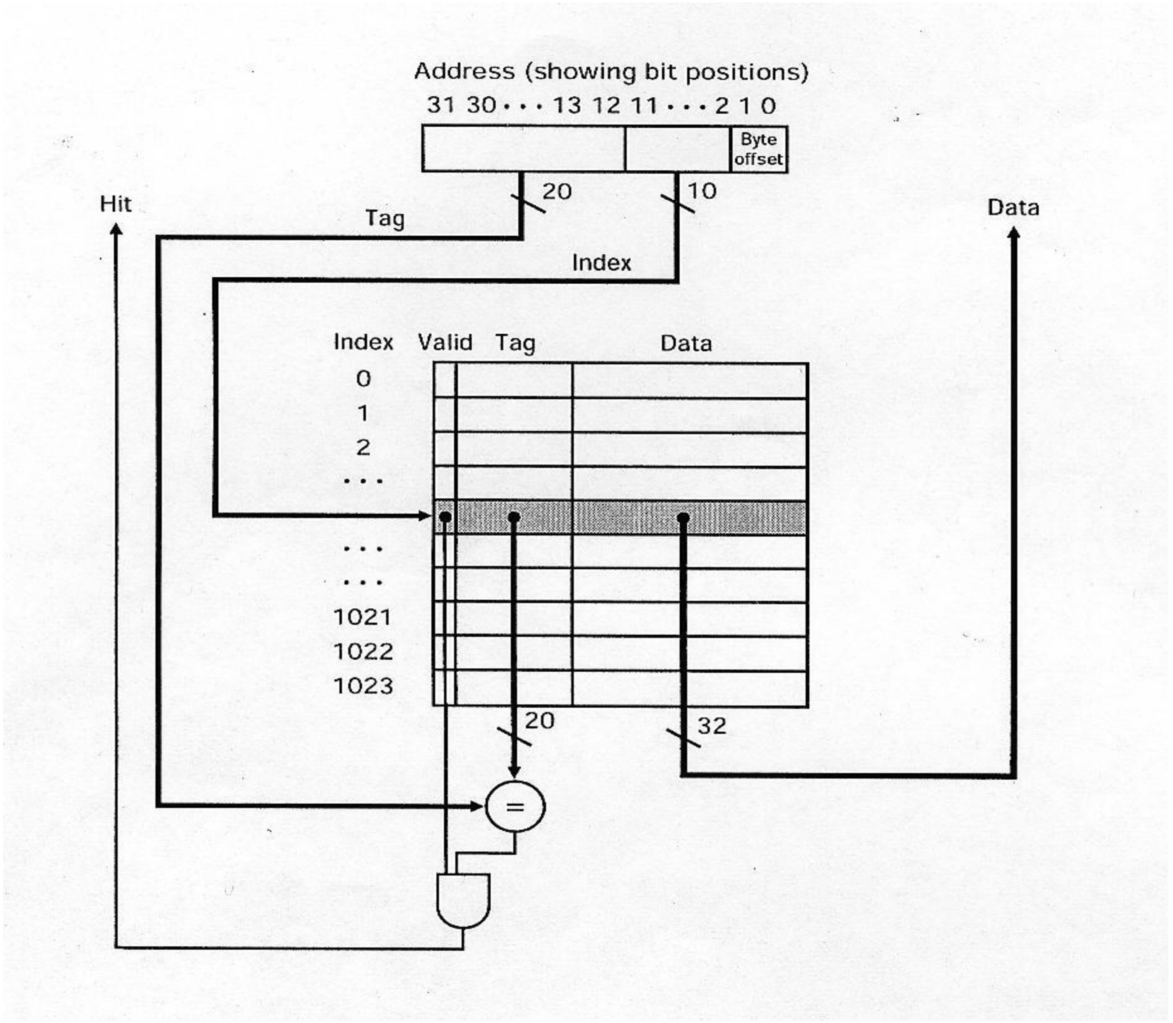
- Write-miss policies
  - Write-allocate (or write allocation)
    - Read the missing block from the lower level memory into cache, and then work as write hit (WT or WB).
  - Write-around
    - Write the data into the next level memory.

N-way set associative

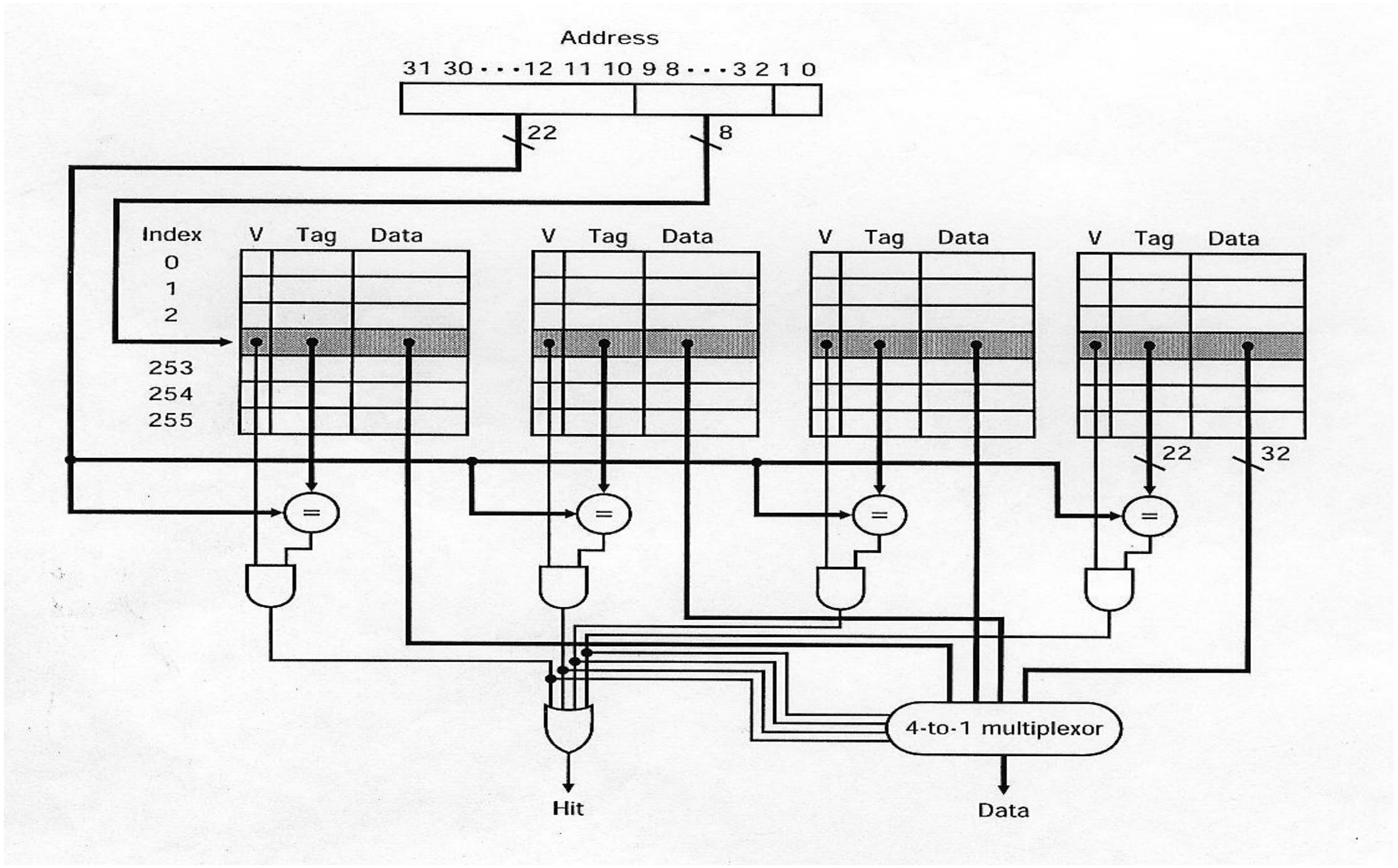
- N direct mapped caches in parallel
- An index gets N blocks



Direct Mapped Cache



Set-Associative Cache



Associativity

- Associativity is a trade-off.
- Cache operations with more associativity takes more power, chip area, and potentially time.
- However, caches with more associativity suffer fewer misses, so that CPU wastes less time reading from main memory.

2-Way Associative Example

The sequence of memory access: 00, 20, 00, 1c, 00

Mem Block	Hit/Miss

Main Memory

Index		
0		
1		
2		
3		

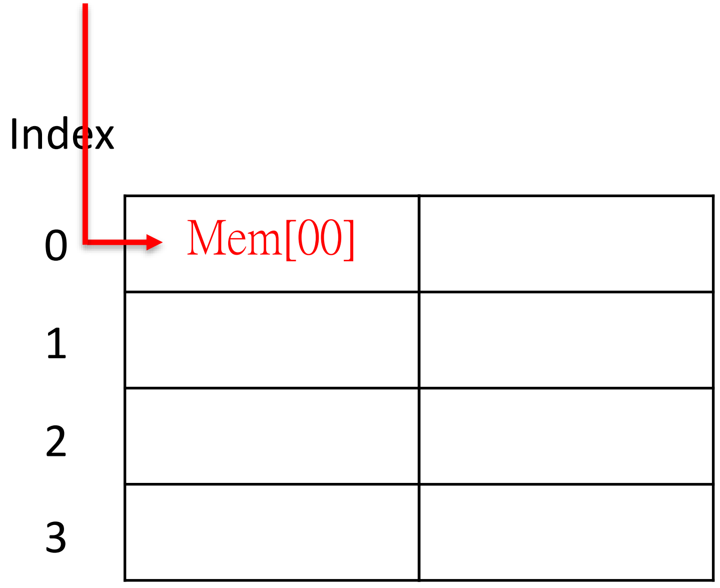
Cache

2-Way Associative Example

The sequence of memory access: 00, 20, 00, 1c, 00

Mem Block	Hit/Miss
00	Miss

Main Memory



Cache

2-Way Associative Example

The sequence of memory access: 00, 20, 00, 1c, 00

Mem Block	Hit/Miss
00	Miss
20	Miss

Main Memory

Index		
0	Mem[00]	Mem[20]
1		
2		
3		

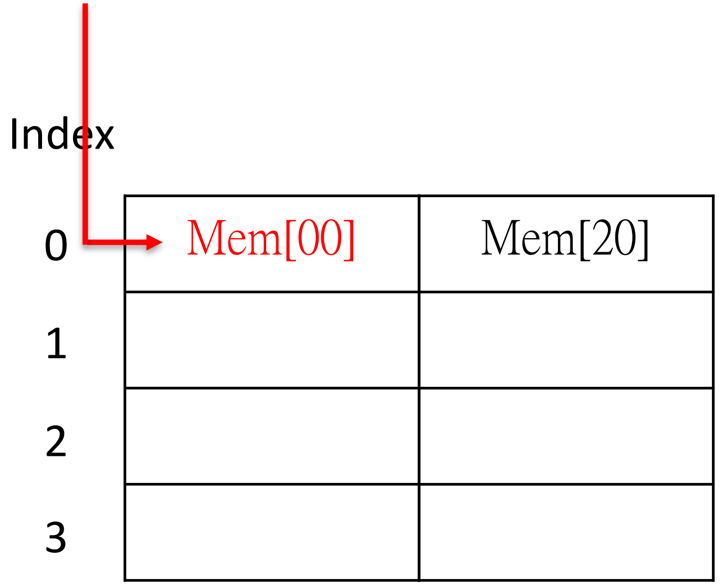
Cache

2-Way Associative Example

The sequence of memory access: 00, 20, 00, 1c, 00

Mem Block	Hit/Miss
00	Miss
20	Miss
00	Hit

Main Memory



Cache

2-Way Associative Example

The sequence of memory access: 00, 20, 00, 1c, 00

Mem Block	Hit/Miss
00	Miss
20	Miss
00	Hit
1c	Miss

Main Memory

Index		
0	Mem[00]	Mem[20]
1		
2		
3	Mem[1c]	

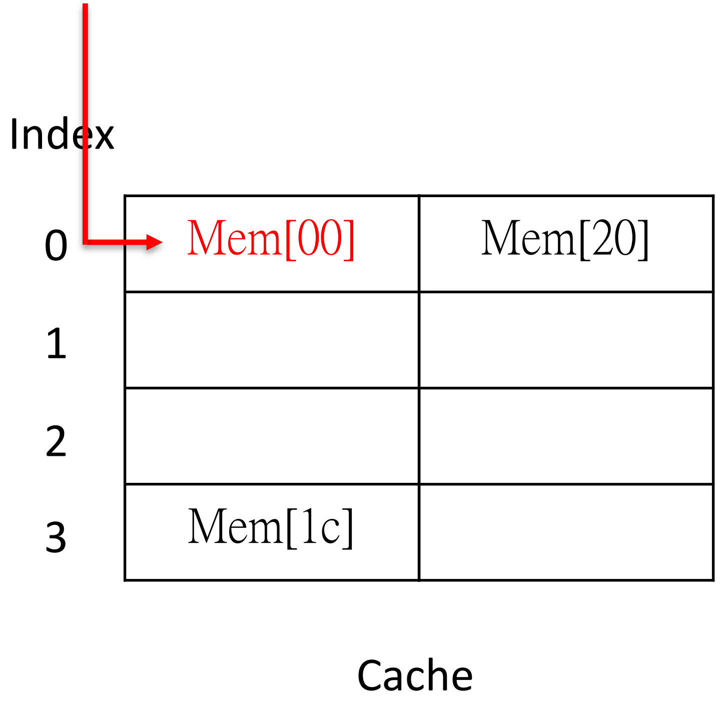
Cache

2-Way Associative Example

The sequence of memory access: 00, 20, 00, 1c, 00

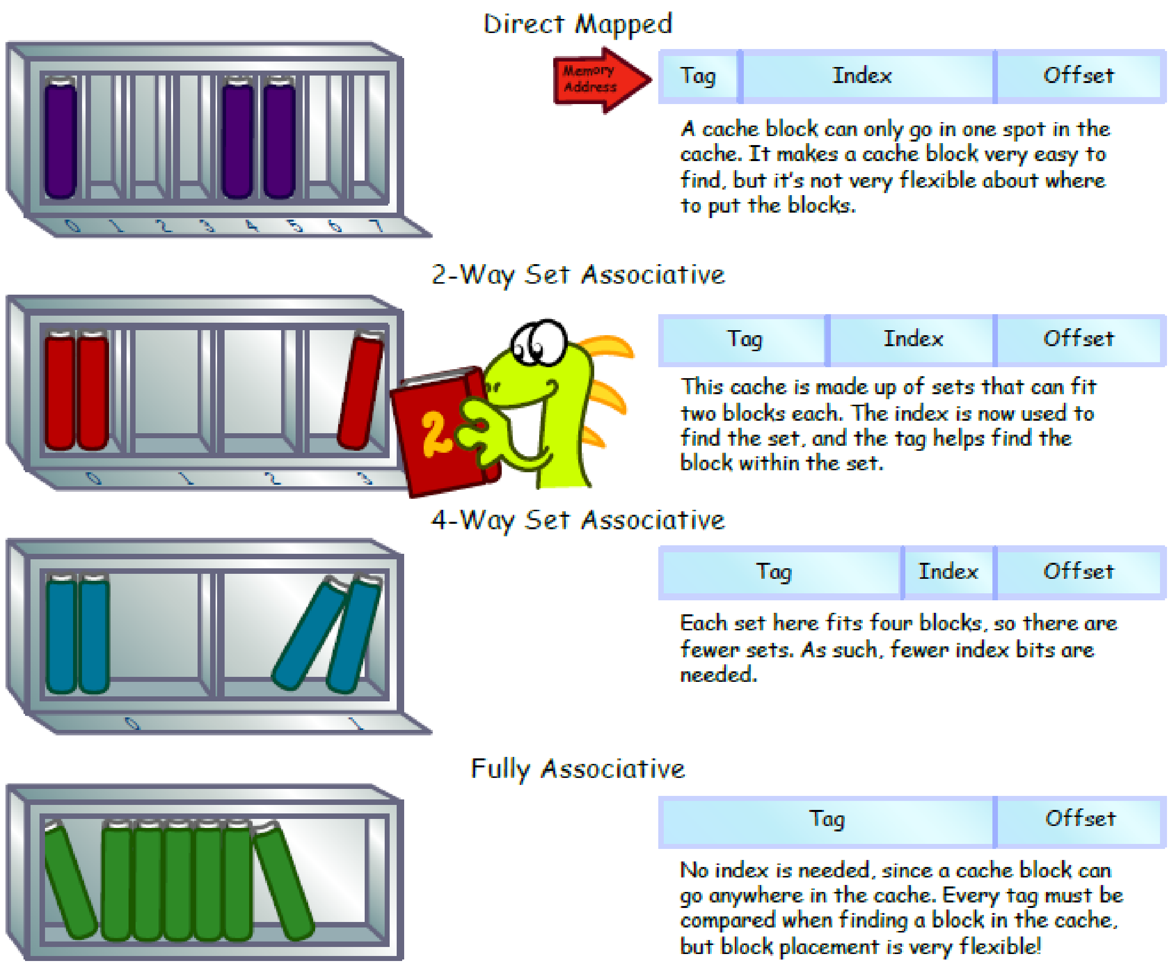
Mem Block	Hit/Miss
00	Miss
20	Miss
00	Hit
1c	Miss
00	Hit

Main Memory



Hit Rate = 40%

Set-Associative Cache



A cache block can only go in one spot in the cache. It makes a cache block very easy to find, but it's not very flexible about where to put the blocks.

This cache is made up of sets that can fit two blocks each. The index is now used to find the set, and the tag helps find the block within the set.

Each set here fits four blocks, so there are fewer sets. As such, fewer index bits are needed.

No index is needed, since a cache block can go anywhere in the cache. Every tag must be compared when finding a block in the cache, but block placement is very flexible!

Type of cache misses

- Compulsory misses:

  The block must be brought into the cache on the first access to a block; also called code start misses
- Capacity misses:

  Blocks are being discarded from cache because cache can't contain all block needed for program execution
- Conflict misses:

  Conflict misses occur when multiple blocks are mapped to the same set, and it could not happen in case of fully set associative cache

Type of cache misses

- Cache optimization

Design change	Effect on miss rate	Possible negative performance effect
increase cache size	reduce capacity misses	possibly increase access time
increase associativity	reduce conflict misses	possibly increase access time
increase block size	reduce compulsory misses	increase miss penalty

Replacement policy

- When a line must be evicted from a cache to make room for incoming data, the replacement policy determines which line is evicted.
- The general goal of the replacement policy is to minimize future cache misses by evicting a line that will not be referenced often in the future.

Replacement policy

- Least recently used (LRU)
  - The cache ranks each the lines in a set according to how recently they have been accessed.
  - Evicts the least-recently used line from a set when an eviction is necessary.
- Random
  - A randomly selected line from the appropriate set is evited to make room for incoming data.
  - Studies have shown that LRU replacement generally gives slightly higher hit rates than random replacement, but that the differences are very small for caches of reasonable size.

Cache in CPU system

- In most systems, caches are meant to be transparent.
- CPU must stalls while cache fetches blocks from memory, and CPU leaves the stall state when cache finished fetching.

LAB 9
Simulation Of I-Cache
&
Set-Associative D-cache

Tool used

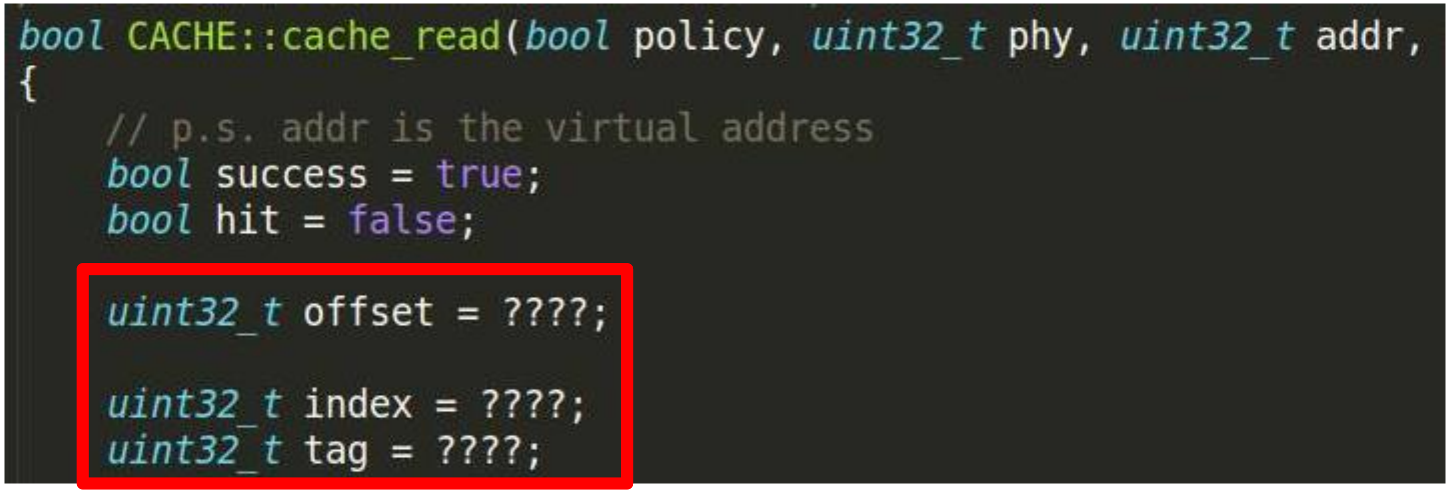
1. Linux
2. CASLab ARM ISS (Instruction Set Simulator)
   - The virtual platform support ARMv5 ISA
3. ARM Toolchain
   - 利用arm-none-eabi- 去cross-compile成 ARM code
4. SystemC
   - 在此模擬平台使用 SystemC 版本為 2.2.0
5. C
   - 主要測試程式以C為主

實作

- 本次實驗須完成 mvp-NCKU/armv5/cache.cpp 包含:
  - 取得 offset, index, tag 之值
  - (參考 mvp-NCKU/armv5/include/cache_defs.h)
  - 實作 read_miss 之 policy
  - 實作 write_miss 之 policy
- 完成cache.cpp後，於mvp-NCKU資料夾下make編譯環境
- 需在mvp-NCKU/prog/ 目錄下make 編譯 test.c
- 最後下 ./run vp 驗證是否正確

Finish the following code

- 請參考 cache_defs.h ，將正確的值填入
- P.S. : offset, index, tag 在其他function中也有，務必填入

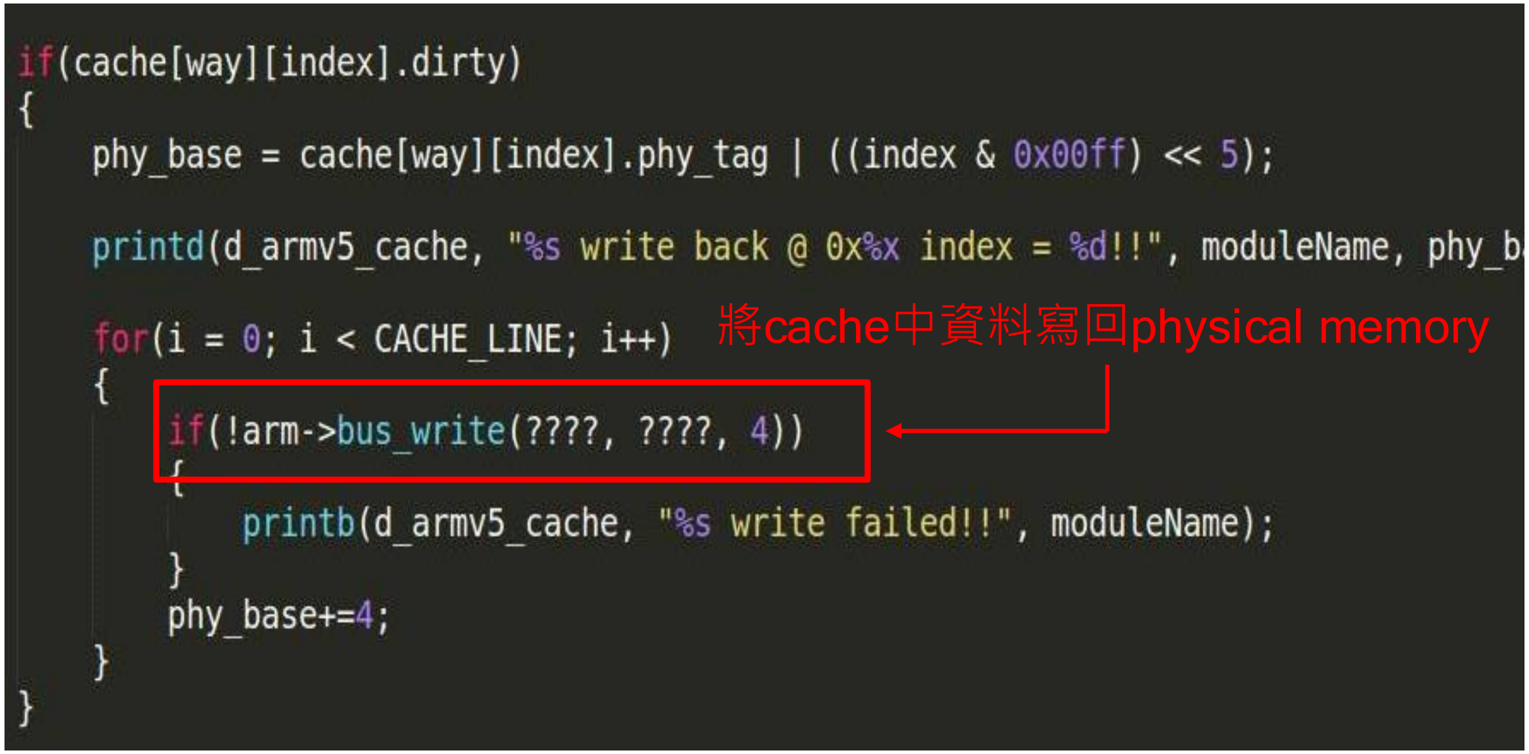


```
bool CACHE::cache_read(bool policy, uint32_t phy, uint32_t addr,
{
    // p.s. addr is the virtual address
    bool success = true;
    bool hit = false;

    uint32_t offset = ????;

    uint32_t index = ????;
    uint32_t tag = ????;
```

Read Miss Policy

1. Read-miss 發生
2. 從cache寫回physical memory
3. 從physical memory 將資料寫入cache中
4. 資料寫回cache後，調整dirty bit 及 valid狀態
5. CPU將 data 從 cache 中讀回

Read Miss Policy

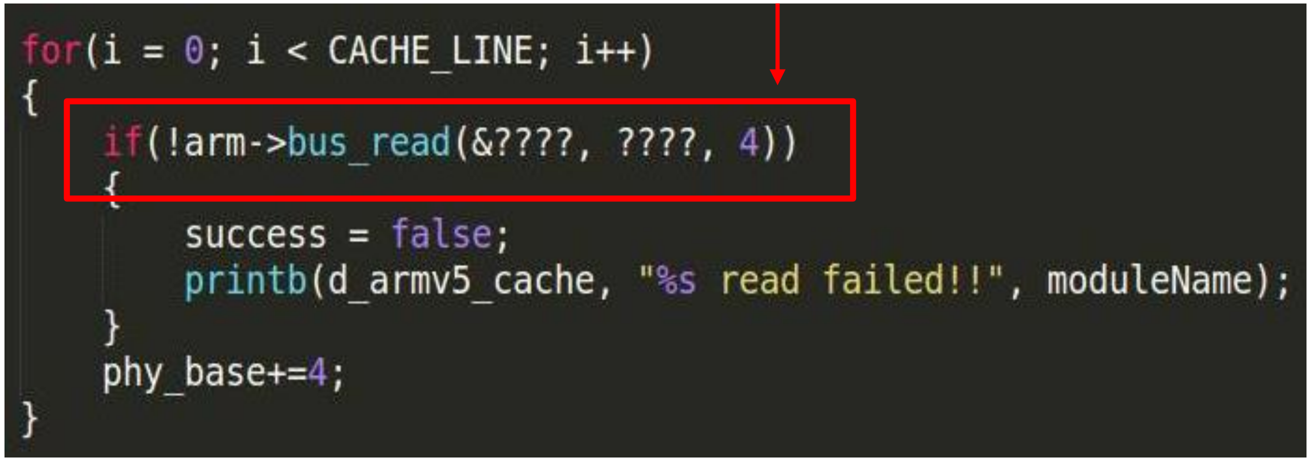


```
if(cache[way][index].dirty)
{
    phy_base = cache[way][index].phy_tag | ((index & 0x00ff) << 5);

    printd(d_armv5_cache, "%s write back @ 0x%x index = %d!!", moduleName, phy_b

    for(i = 0; i < CACHE_LINE; i++)
    {
        if(!arm->bus_write(????, ????, 4))
        {
            printb(d_armv5_cache, "%s write failed!!", moduleName);
        }
        phy_base+=4;
    }
}
```

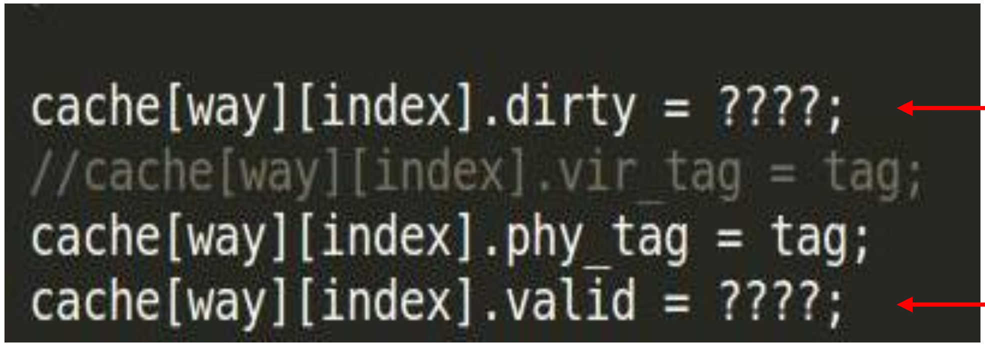
將cache中資料寫回physical memory

Read Miss Policy

將資料從physical memory讀入buffer(inst[])中



```
for(i = 0; i < CACHE_LINE; i++)
{
    if(!arm->bus_read(&????, ????, 4))
    {
        success = false;
        printb(d_armv5_cache, "%s read failed!!", moduleName);
    }
    phy_base+=4;
}
```

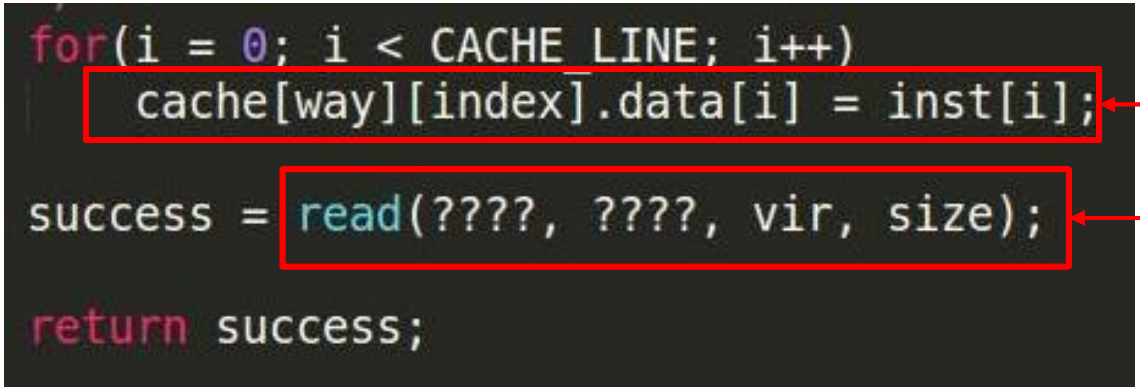


```
cache[way][index].dirty = ????;
//cache[way][index].vir_tag = tag;
cache[way][index].phy_tag = tag;
cache[way][index].valid = ????;
```

調整dirty bit 及 valid狀態

Read Miss Policy

將 buffer 中的值寫入 cache



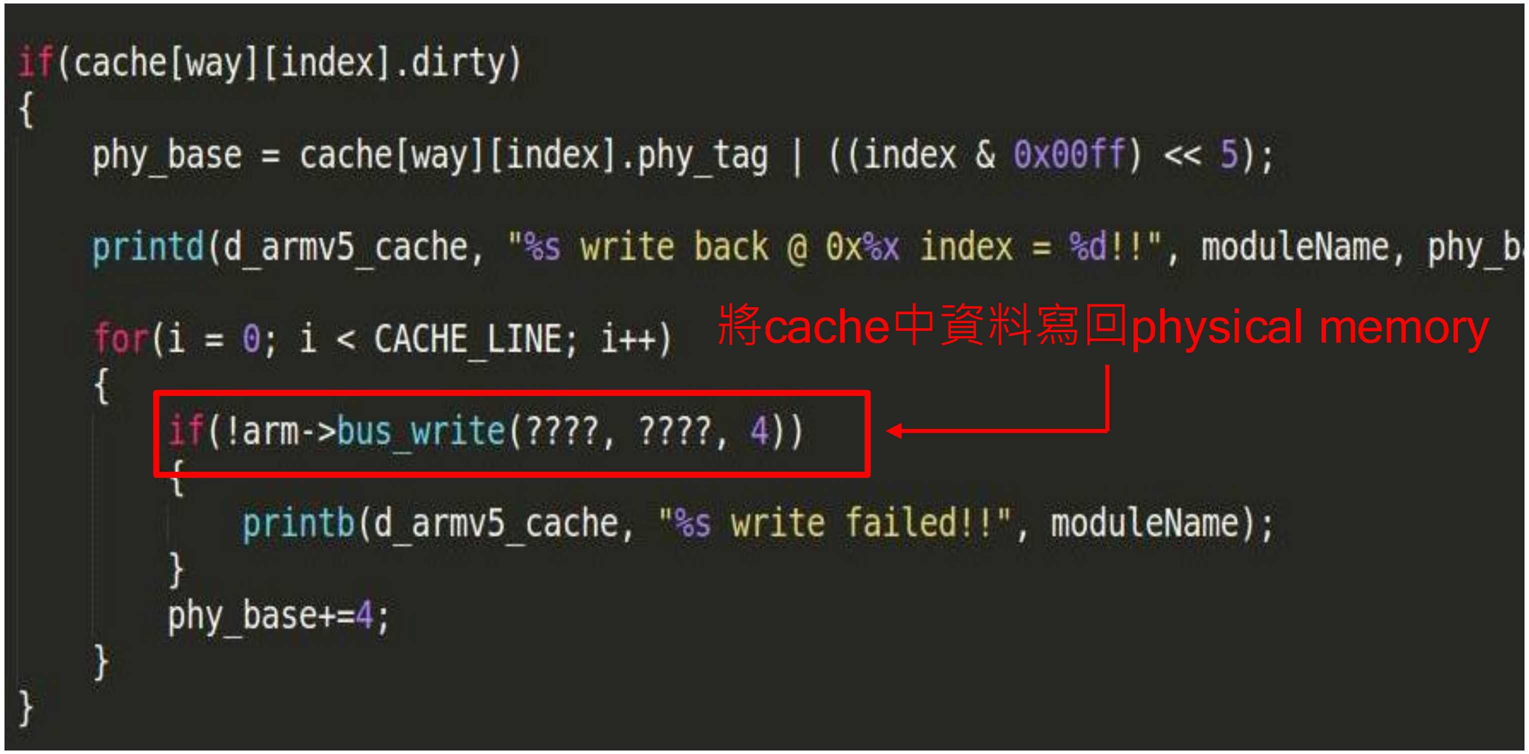
```
for(i = 0; i < CACHE_LINE; i++)
    cache[way][index].data[i] = inst[i];

success = read(????, ????, vir, size);

return success;
```

CPU將 data 從 cache 中讀回

Write Miss Policy

1. Write-miss 發生
2. 從cache寫回physical memory
3. 從physical memory 將資料寫入cache中
4. 資料寫回cache後，調整dirty bit 及 valid狀態
5. CPU再對cache寫入

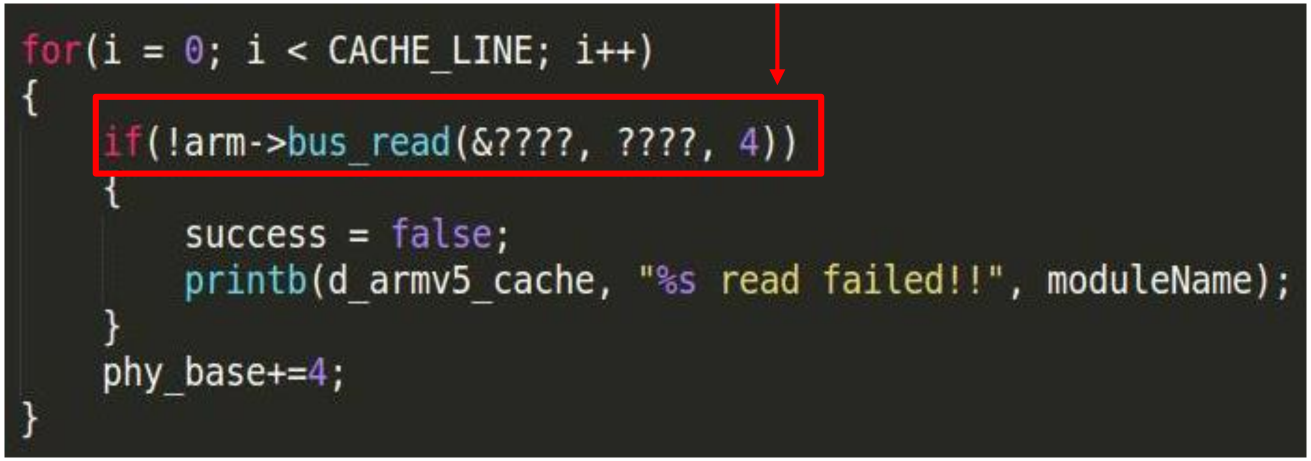
Write Miss Policy



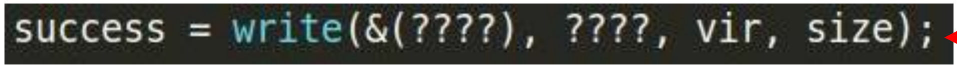
```
if(cache[way][index].dirty)
{
    phy_base = cache[way][index].phy_tag | ((index & 0x00ff) << 5);

    printd(d_armv5_cache, "%s write back @ 0x%x index = %d!!", moduleName, phy_b

    for(i = 0; i < CACHE_LINE; i++)
    {
        if(!arm->bus_write(????, ????, 4))
        {
            printb(d_armv5_cache, "%s write failed!!", moduleName);
        }
        phy_base+=4;
    }
}
```

Write Miss Policy

將資料從physical memory讀入到buffer (inst[])

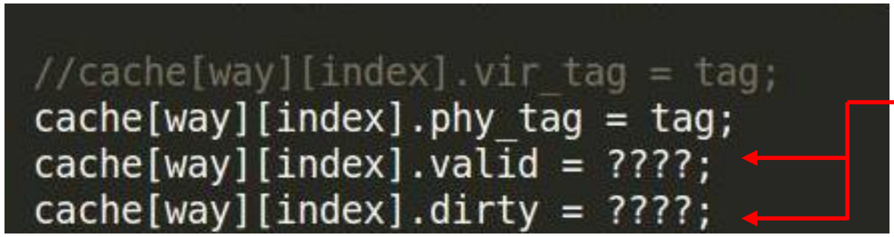


```
for(i = 0; i < CACHE_LINE; i++)
{
    if(!arm->bus_read(&????, ????, 4))
    {
        success = false;
        printb(d_armv5_cache, "%s read failed!!", moduleName);
    }
    phy_base+=4;
}
```



```
success = write(&(????), ????, vir, size);
```

CPU再對cache寫入



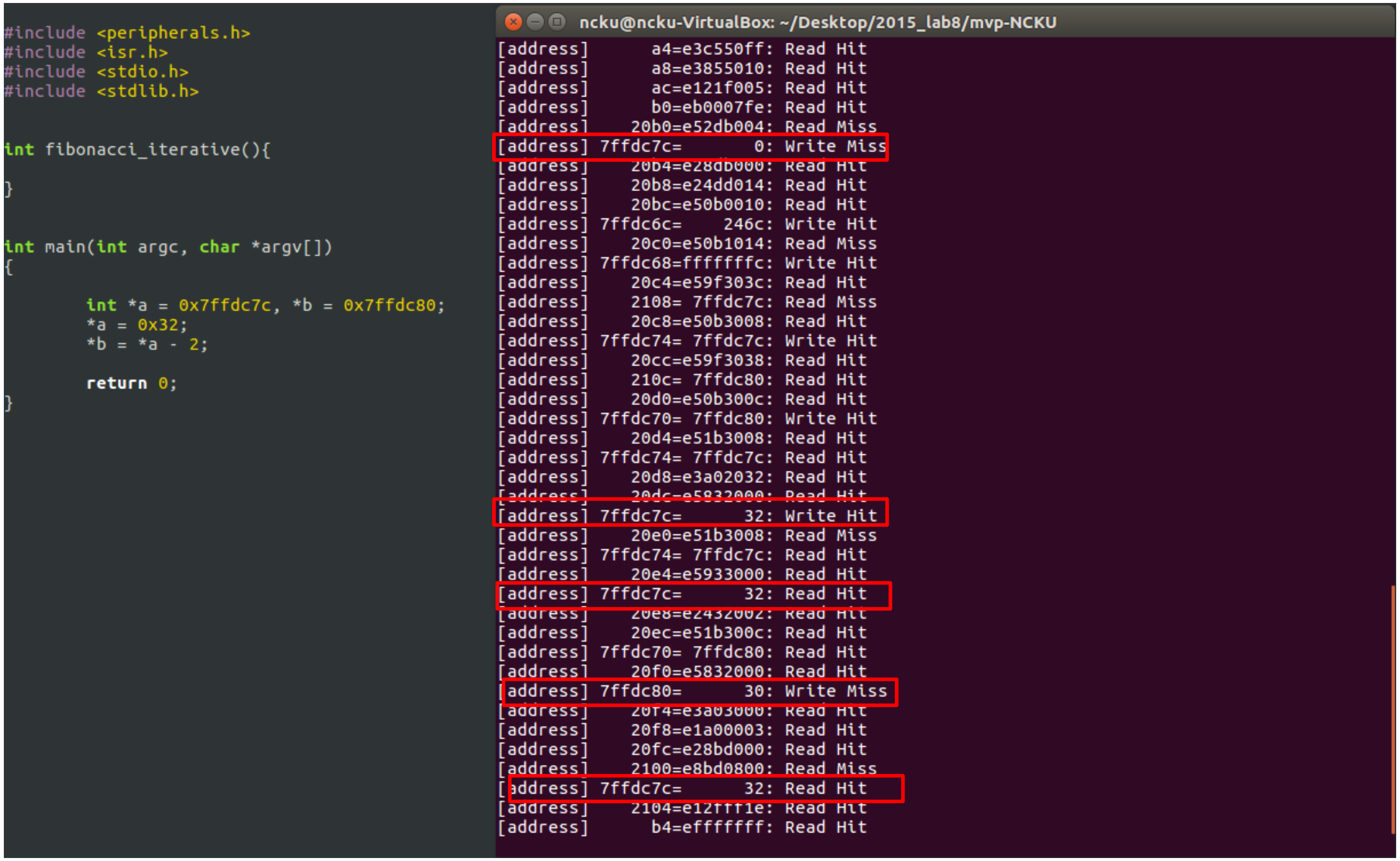
```
//cache[way][index].vir_tag = tag;
cache[way][index].phy_tag = tag;
cache[way][index].valid = ????;
cache[way][index].dirty = ????;
```

調整dirty bit 及 valid狀態

Hint!!!!

- bus_read(*data, addr, length)
  - *data : 從memory讀回來要存放之地址
  - addr : memory讀取的地址
  - length: 讀取長度(byte)
- bus_write(data, addr, length)
  - data : 須要寫入memory的資料
  - addr :寫入memory的地址
  - length:寫入長度(byte)
- Type of “valid” and “dirty” is boolean

驗證



```c
#include <peripherals.h>
#include <isr.h>
#include <stdio.h>
#include <stdlib.h>


int fibonacci_iterative(){

}


int main(int argc, char *argv[])
{

        int *a = 0x7ffdc7c, *b = 0x7ffdc80;
        *a = 0x32;
        *b = *a - 2;

        return 0;
}
```

ncku@ncku-VirtualBox: ~/Desktop/2015_lab8/mvp-NCKU

```
[address]       a4=e3c550ff: Read Hit
[address]       a8=e3855010: Read Hit
[address]       ac=e121f005: Read Hit
[address]       b0=eb0007fe: Read Hit
[address]     20b0=e52db004: Read Miss
[address] 7ffdc7c=        0: Write Miss
[address]     20b4=e28db000: Read Hit
[address]     20b8=e24dd014: Read Hit
[address]     20bc=e50b0010: Read Hit
[address] 7ffdc6c=     246c: Write Hit
[address]     20c0=e50b1014: Read Miss
[address] 7ffdc68=fffffffc: Write Hit
[address]     20c4=e59f303c: Read Hit
[address]     2108= 7ffdc7c: Read Miss
[address]     20c8=e50b3008: Read Hit
[address] 7ffdc74= 7ffdc7c: Write Hit
[address]     20cc=e59f3038: Read Hit
[address]     210c= 7ffdc80: Read Hit
[address]     20d0=e50b300c: Read Hit
[address] 7ffdc70= 7ffdc80: Write Hit
[address]     20d4=e51b3008: Read Hit
[address] 7ffdc74= 7ffdc7c: Read Hit
[address]     20d8=e3a02032: Read Hit
[address]     20dc=e5832000: Read Hit
[address] 7ffdc7c=       32: Write Hit
[address]     20e0=e51b3008: Read Miss
[address] 7ffdc74= 7ffdc7c: Read Hit
[address]     20e4=e5933000: Read Hit
[address] 7ffdc7c=       32: Read Hit
[address]     20e8=e2432002: Read Hit
[address]     20ec=e51b300c: Read Hit
[address] 7ffdc70= 7ffdc80: Read Hit
[address]     20f0=e5832000: Read Hit
[address] 7ffdc80=       30: Write Miss
[address]     20f4=e3a03000: Read Hit
[address]     20f8=e1a00003: Read Hit
[address]     20fc=e28bd000: Read Hit
[address]     2100=e8bd0800: Read Miss
[address] 7ffdc7c=       32: Read Hit
[address]     2104=e12fff1e: Read Hit
[address]       b4=efffffff: Read Hit
```

實驗結報

- 結報格式(每組一份)
  - 封面（第幾組+組員）
  - 實驗內容(程式碼註解、結果截圖)
  - 實驗心得
- 繳交位置
  - ftp : 140.116.164.**225**
  - 帳號/密碼 : coco2016 / coco2016
- TA Contact Information:
  - 助教信箱 : lance91633@gmail.com
  - Rm 92617
  - Office hour : (Wed.)09:00~12:00