



成功大學
National Cheng Kung University

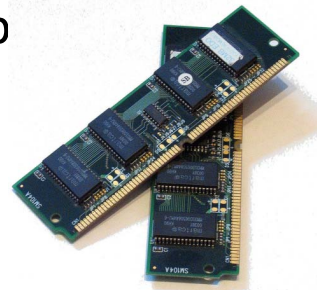
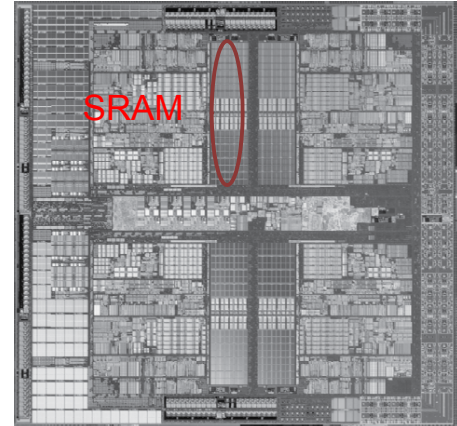
Chapter 5

Large and Fast: Exploiting Memory Hierarchy



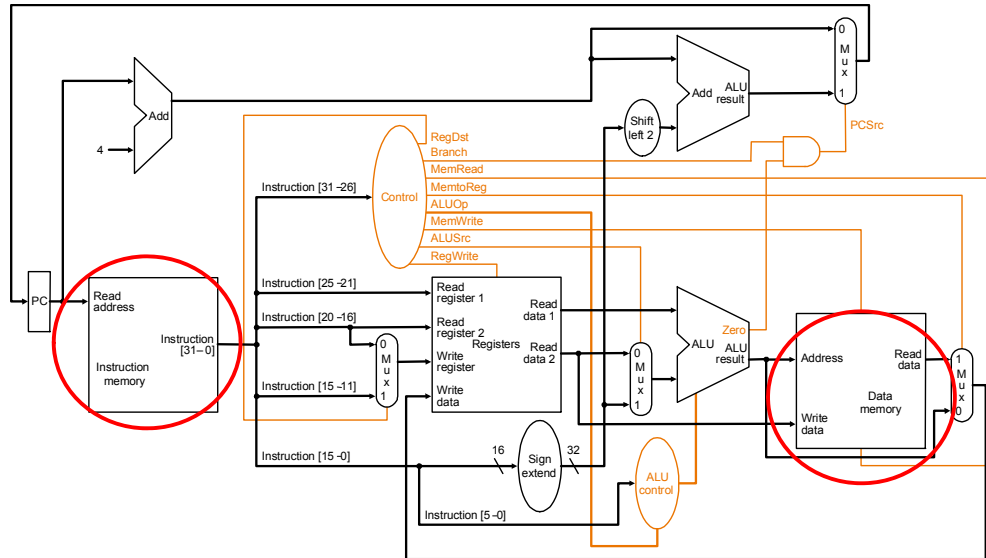
Memory Technology

- Static RAM (**SRAM**)
 - 0.5ns – 2.5ns, \$2000 – \$5000 per GB
- Dynamic RAM (**DRAM**)
 - 50ns – 70ns, \$20 – \$75 per GB
- Magnetic disk
 - 5ms – 20ms, \$0.20 – \$2 per GB
- Ideal memory (fast and cheap)
 - Access time of **SRAM**
 - Capacity and cost/GB of **disk**



wiseGEEK

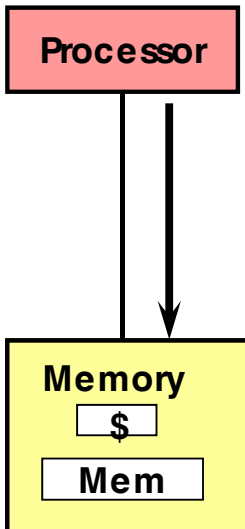
Recap: Single-Cycle MIPS



Two memory are used: Instruction and data memory

Memory References Workload

- programs



Memory reference stream are instructions that will access memory

<op,addr>,
<op,addr>,
<op,addr>,
<op,addr>,
...

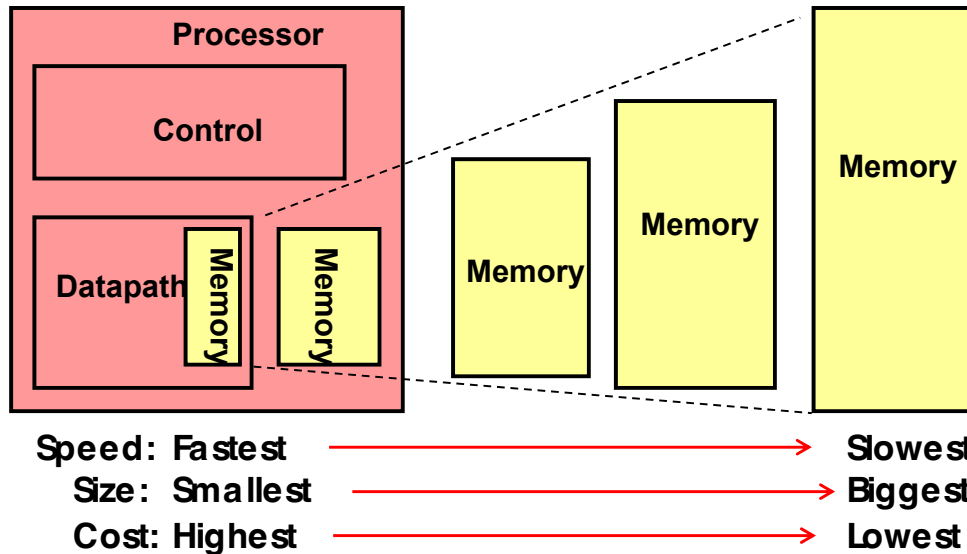
op: **instruction fetch**, read (**lw**), and write (**sw**)

Goal:

Optimize the memory system organization to **minimize** the average memory **access time** for typical workloads



Memory Hierarchy: Concept



Why Hierarchy Works?

- Principle of Locality:

- Program access a relatively small portion of the address space at any instant of time
- 90/10 rule or 80/20 rule: 10% (20%) of code executes 90% (80%) of time

- Types of locality:

- **Temporal locality**: if an item is referenced, it will tend to be referenced again soon

- E.g., some counter variable in a loop

```
for (i=0; i<1000; i++) {  
    A[i+1]=A[i]+1  
}
```

- **Spatial locality**: if an item is referenced, items whose addresses are close by tend to be referenced soon

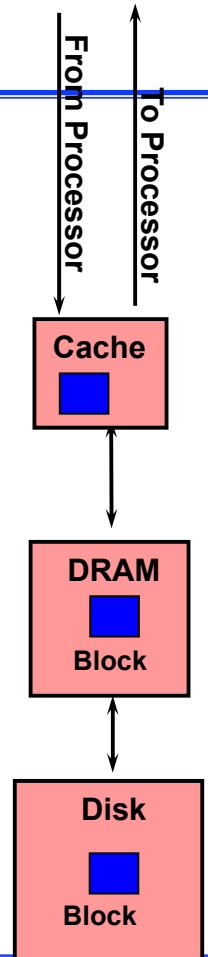
- E.g., nearby array data $A[i]$ and $A[i+1]$

- i is referenced frequently
- $A[i+1]$, $A[i]$ are nearby



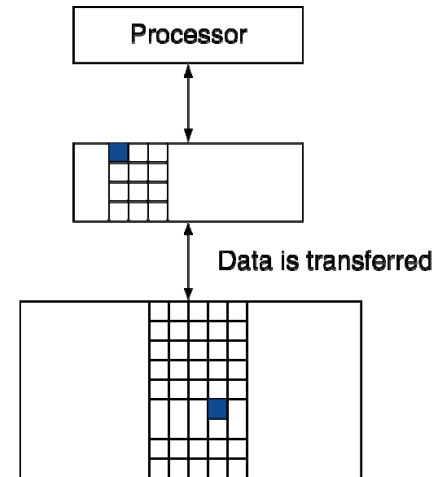
Taking Advantage of Locality

- **Memory hierarchy**
- Store everything on **disk**
- Copy recently accessed (and nearby) items from disk to smaller **DRAM** memory
 - Main memory
- Copy more recently accessed (and nearby) items from **DRAM** to smaller **SRAM** memory
 - Cache memory attached to CPU



Memory Hierarchy Levels

- **Block** (aka **line**): unit of copying, may be multiple words
- **Hit**: data is **present** in upper level
 - Access satisfied by upper level
- **Miss**: data is **absent** in upper level
 - data needs to be retrieved from a block in the lower level (Block Y)
 - Then accessed data are supplied from upper level

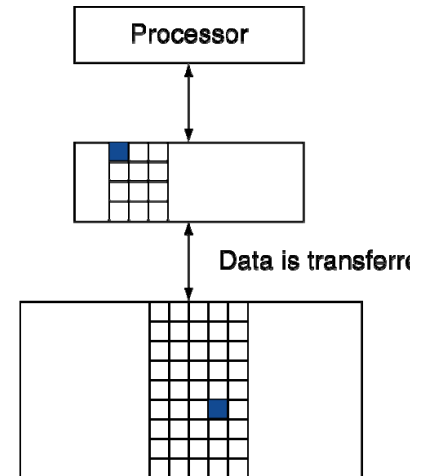


Memory Hierarchy Levels: Terminology

- **Hit rate (ratio)**: fraction of memory access found in the upper level

Hit Rate = hits / accesses

- **Hit time**: time to access the upper level
 - Time to determine hit/miss + Mem access time
- **Miss rate** = $1 - (\text{Hit rate})$
- **Miss penalty**:
 - Time to place (replace) a block (with a block) in the upper level + time to **deliver** the block to the processor



Hit Time << Miss Penalty \Rightarrow It is important to reduce miss rate



Exercise

- Suppose 20 misses per 1000 access. The total number of accesses is 10,000, miss rate is 2%, and miss penalty is 100 cycles

1. What is the miss rate?

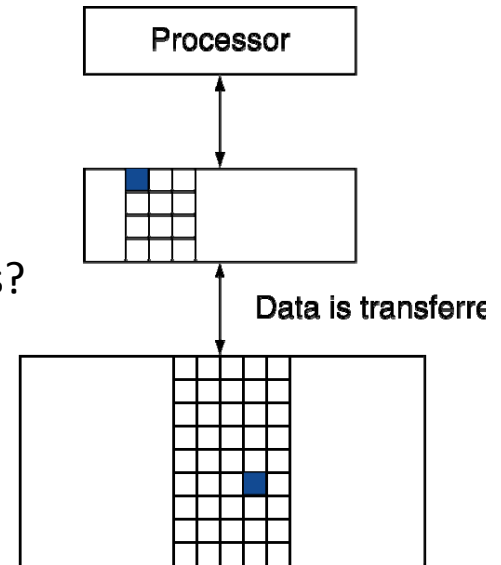
$$20/1000=2\%$$

2. What are the extra cycles caused by miss?

$$10000*2\%*100=20000$$

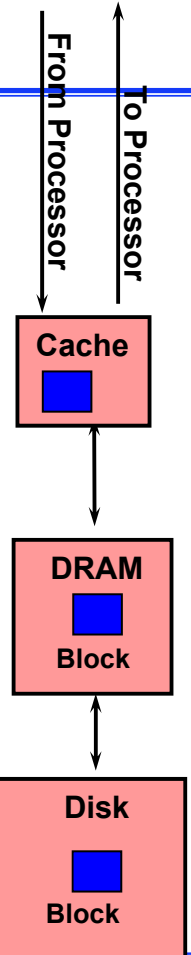
3. What are the **total cycles** for the 10,000 access?

$$10000+10000*2\%*100=30000$$



Cache Memory

- Cache memory
 - The level of the memory hierarchy **closest** to the CPU
- Given accesses X_1, \dots, X_{n-1}, X_n , how do we **know** if the data is **present**? **Where** do we look?
 - First technique: **directed mapped**



X_4
X_1
X_{n-2}
X_{n-1}
X_2
X_3

a. Before the reference to X_n

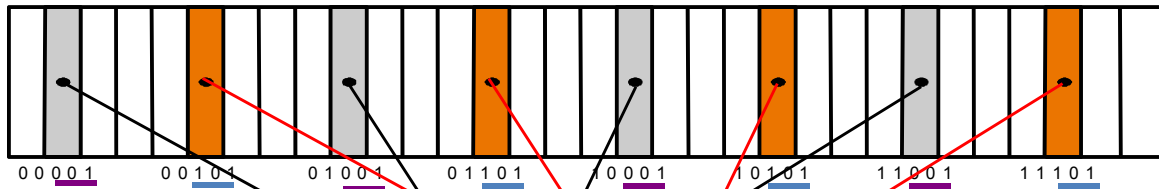
X_4
X_1
X_{n-2}
X_{n-1}
X_2
X_n
X_3

b. After the reference to X_n 

Basics of Cache – Direct-mapped cache

- Location determined by **address**
- Direct mapped: only one choice
 - (Block address) modulo (#Blocks in cache)
- 32-block main memory, block address are 00000, 00001, 00010..., mapped into a 8-block cache

常用的放入cache，
其餘放回memory



$00001 \bmod 8 = 1$, $01001 \bmod 8 = 1$, $10001 \bmod 8 = 1$

一個address固定map到1個位置
Direct mapping：多對一容易發生cache miss



Cache Example

- **Cache 8**-blocks, direct mapped
- Block Address are: 10110, 11010, 10110, **11010**,
10000, 00011, 10000, 10010, 10000
- Initial state

Valid bit: 1=present, 0 = not present

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		



Cache Example (access 10110_2)

(1)

Decimal addr	Binary addr	Hit/miss	Assigned cache block (Index)
22	10110	Miss	110

(1) Address referred 10110 (*miss*):

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		



Cache Example (access 11010_2)

	Decimal addr	Binary addr	Hit/miss	Assigned cache block (Index)
(1)	22	10110	Miss	110
(2)	26	11010	Miss	010

Index	V	Tag	Data
000	N		
001	N		
010	Y	11	Mem[11010]
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		



Cache Example

	Decimal addr	Binary addr	Hit/miss	Cache block
(3)	22	10 110	Hit	110
(4)	26	11 010	Hit	010

Nothing is changed in cache

Index	V	Tag	Data
000	N		
001	N		
010	Y	11	Mem[11010]
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		



Cache Example

	Decimal addr	Binary addr	Hit/miss	Cache block
(5)	16	10 000	Miss	000
(6)	3	00 011	Miss	011
(7)	16	10 000	Hit	000

Index	V	Tag	Data
000	Y	10	Mem[10000]
001	N		
010	Y	11	Mem[11010]
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		



Cache Example

(8)

Decimal addr	Binary addr	Hit/miss	Cache block
18	10 010	Miss	010

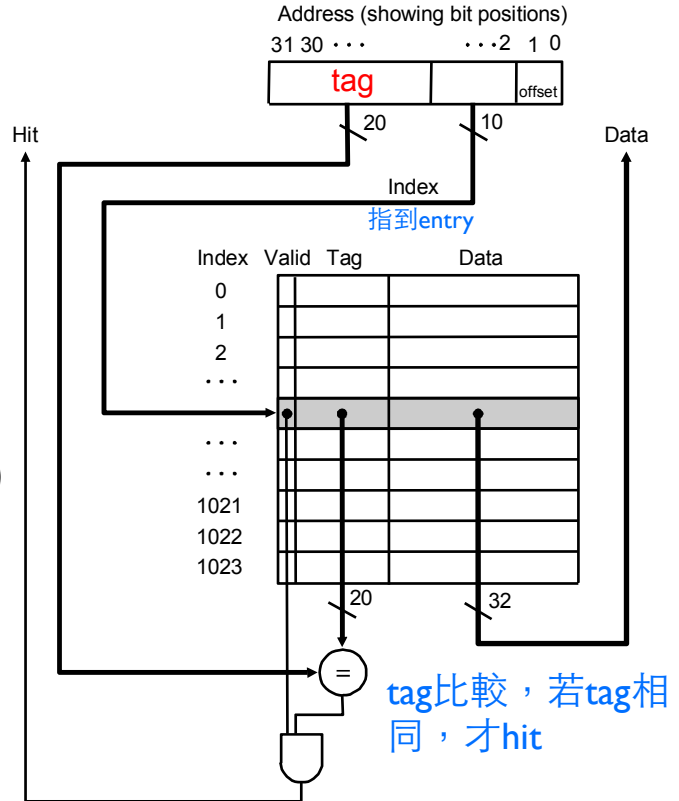
Index	V	Tag	Data
000	Y	10	Mem[10000]
001	N		
010	Y	10	Mem[10010]
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		



Direct-Mapped Cache Organization

How do we know where a block is stored in a cache?

- Use **Tag** and **valid bit**
- Example, cache has **1024** blocks, each block is 1-word (4bytes)
 - Offset (byte offset): 2 bits ($2^2=4$)
 - Cache **index**: lower **10** bits ($2^{10}=1024$)
 - Cache **tag**: upper 20 bits ($32-2-10$)
 - **Valid bit** (when start up, valid is 0)



Example: Larger Block Size

- Consider a direct-mapped cache with 64 blocks and a block size of 4 word (byte addressable) .
 - How many bits for tag, index and offset
 - What is the index of the address 1200?

$$2^4$$

Block size= 4 words= 16 bytes, therefore, offset is 4 bits (word offset is 2 bits, and byte offset is 2bits)

64 blocks in total, therefore, index is 6 bits
($2^6=64$)

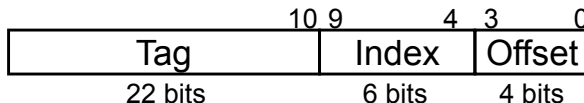
Tag has (32-4-6)= 22 bits

$$1200 = 0 \dots 10010110000_2$$

index offset

$$\text{Index} = 001011$$

$$\text{Tag} = 000000000000000000000001$$



64 blocks 16 bytes in a block



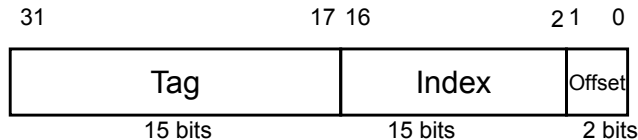
Example 2

- What is the total cache size for a direct-mapped cache with 2^{15} blocks and 1-word block size, assuming a 32-bit address? 1 block = 4 bytes

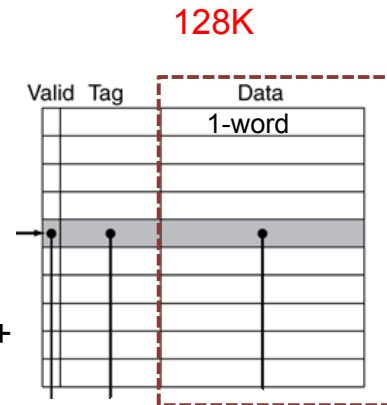
Size to store data = 2^{15} blocks = 2^{17} bytes = 128 KB

Index is 15 bit (because 2^{15} blocks)

Tag is 15 bit ($32 - 15 - 2$)



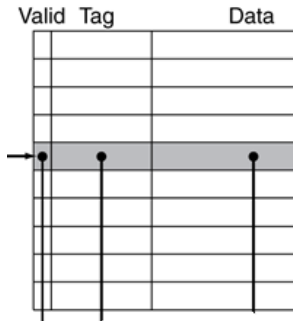
Cache entry size = block data bits + tag bits + valid bit = $32 + (32 - 15 - 2) + 1 = 48$ bits

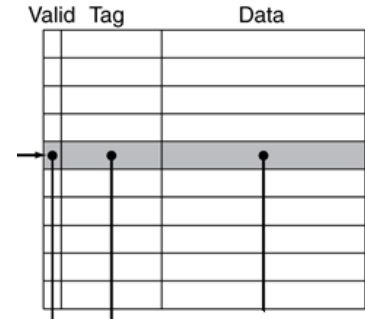


Therefore, cache size = $2^{15} \times 48$ bits = $2^{15} \times (1.5 \times 32)$ bits = 1.5×2^{20} bits = 1.5 Mbits



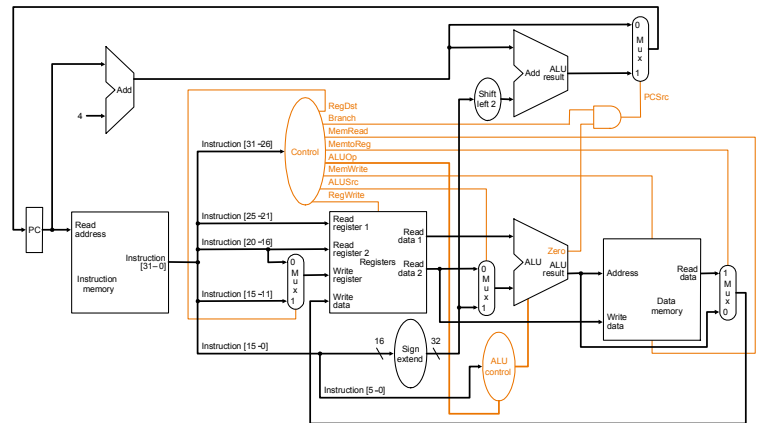
Block Size Considerations

- Larger **blocks** should reduce miss rate
 - Due to **spatial** locality
 - But in a fixed-sized cache
 - Larger blocks \Rightarrow **less number of blocks**
 - More address compete for a block
 - \Rightarrow **increased** miss rate
 - Larger blocks \Rightarrow more data are not used in a block \Rightarrow **pollution**
 - Larger blocks \Rightarrow Larger **miss penalty**
 - Take more time to move a block
 - Can override benefit of reduced miss rate
- 
- The diagram illustrates a cache structure with three columns: Valid, Tag, and Data. It shows a grid of 8 rows. The first row is highlighted in gray. An arrow points to the Valid bit in the first row, which is set to 1. The Tag and Data fields are empty in this row. The other rows have Valid bits set to 0 and empty Tag and Data fields.



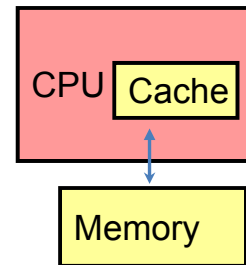
Cache Misses

- On cache hit, CPU proceeds normally
- On cache miss
 - Stall the CPU pipeline, Fetch block from next level of hierarchy
 - Instruction cache miss
 - Restart instruction fetch
 - Data cache miss
 - Complete data access



Handling Writes

- On data-write hit, if only update the block in cache CPU 從cache讀資料 (cache會將資料從memory備份)
 - But then cache and memory would be **inconsistent** cache、memory的資料不一致
- **Two policies: Write Through and Write back**
- **Write through**: also update memory
 - makes writes take longer because needs to wait for MEM operation 需等待memory寫入，速度慢



e.g. if base CPI = 1, write to memory takes **100 cycles**, miss rate is **10%**, find new CPI

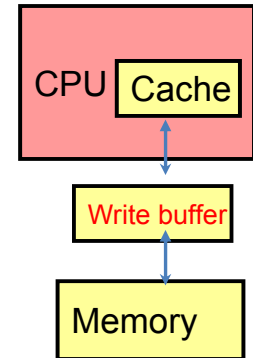
$$\text{New CPI} = 1 + 10\% \times 100 = 11$$



Handling Writes – Write through

- Improve the performance of write through using **write buffer** 改善write through效率
 - Holds data waiting to be written to memory
 - CPU continues immediately
 - Only **stalls** on write if write buffer is already **full**

Write Buffer(CPU 控制記憶體)：先將資料寫到write buffer，等到write buffer滿了，才寫回memory



- Alternative: On data-write hit, just update the block in cache 只寫cache，memory不寫

- When Dirty is 1, mean it is already modified (cache and memory data are different) 資料有被更改，cache和memory的資料會不相同

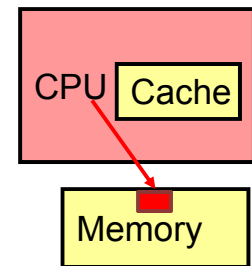
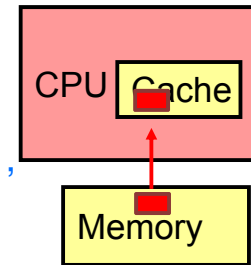


- 讀取時直接讀取write back buffer
(不用從memory讀取)



Write Allocation

- What should happen on a **write miss** (write a block, but the block is not in cache)?
- For **write through**
 - Allocate the block in cache, and write **cache** and **memory** Write allocate : 先將block從memory搬回cache, cache、memory都要寫
 - don't allocate the block (**write no-allocate**)
 - Aka. Write around 不寫入cache, 直接寫到memory
 - Useful for **initialization** (initialize data before use it)
- For **write-back** 只寫cache
 - Allocate the block on cache, and write **the cache** only

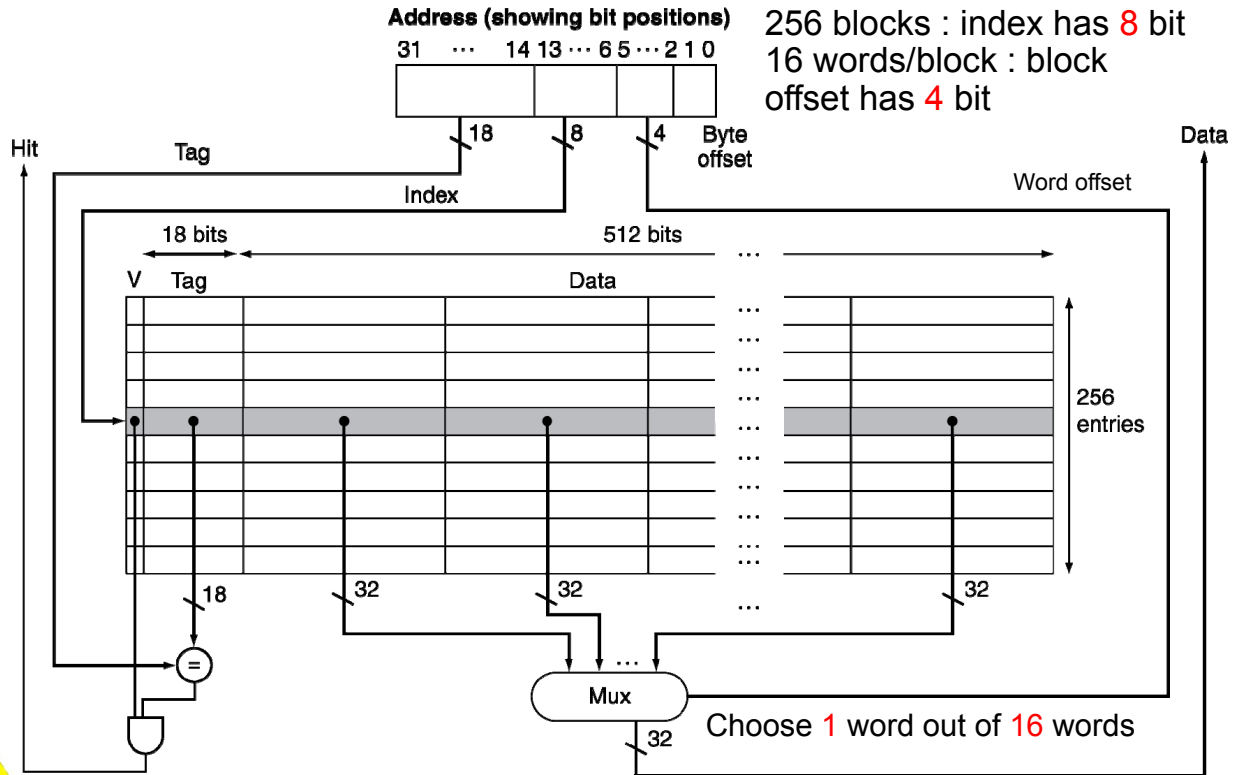


Example: Intrinsity FastMATH

- Embedded MIPS processor (See **next slides**)
 - 12-stage pipeline
 - Instruction and data access on each cycle
- Split cache: separate I-cache and D-cache
 - Each 16KB: 256 blocks \times 16 words/block
 - D-cache: write-through or write-back
- SPEC2000 miss rates
 - I-cache: 0.4%
 - D-cache: 11.4%
 - Weighted average: 3.2%



Example: Intrinsity FastMATH



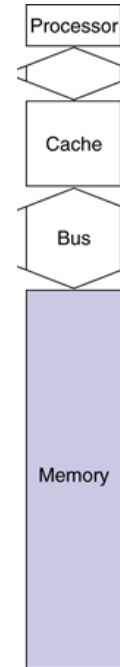
Main Memory Supporting Caches

- Use **DRAMs** for main memory
 - Fixed width (e.g., **1** word)
 - Connected by fixed-width clocked **bus**
 - **Bus clock** is typically **slower** than CPU clock
- e.g. cache block read
 - **1** bus cycle for address transfer
 - **15** bus cycles per DRAM access(read)
 - **1** bus cycle per data transfer
- For **4**-word block, **1**-word-wide DRAM

Miss penalty = $1 + 4 \times 15 + 4 \times 1 = 65$ bus cycles

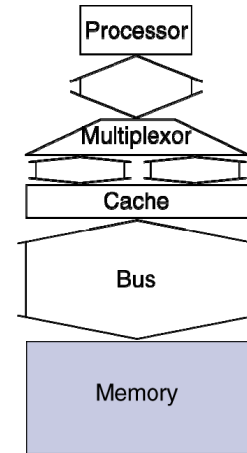
Note: address transfer is 1 cycle because of sequential access

Bandwidth = $\frac{16 \text{ bytes}}{65 \text{ cycles}} = 0.25 \text{ Bytes/cycle}$



Increasing Memory Bandwidth

- Method1: **Wider memory and bus width**
 - Access 4-word and transfer 4-word at a one time
- Miss penalty and Bandwidth for 4-word wide memory



b. Wider memory organization

- Miss penalty = $1 + 15 + 1 = 17$ bus cycles
- Bandwidth = $16 \text{ bytes} / 17 \text{ cycles} = 0.94 \text{ B/cycle}$

- Increase bandwidth, but memory and bus cost is high

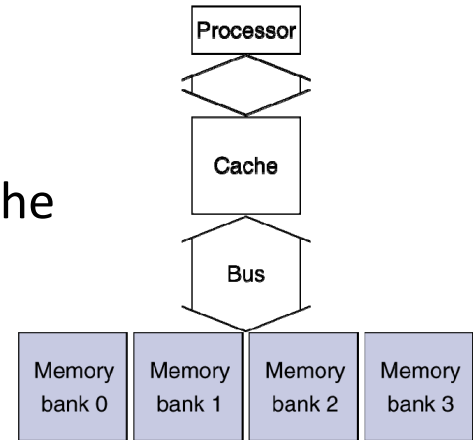


Increasing Memory Bandwidth

- Method 2: **Interleaved** memory organize
 - Bus **width** is the **same**
 - Sending **address** to 4 banks at the time

=> Four memory bank can be accessed at the same time

=> **overlap** the access time



c. Interleaved memory organization

- Miss penalty = 1 + **15** + 4×1 = **20 bus** cycles
- Bandwidth = 16 bytes / 20 cycles = 0.8 B/cycle

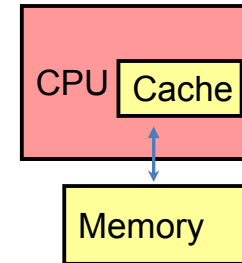
只有一條bus可回傳，4個

4個同時讀 memory bank輪流傳



Measuring Cache Performance

- Components of CPU time
 - Program execution cycles
 - Includes cache hit time
 - Memory stall cycles
 - Mainly from cache misses



$$\text{CPU Time} = (\text{CPU Clock cycle} + \text{Memory stall cycle}) \times \text{Cycle Time}$$

Memory stall cycles

$$= \frac{\text{Memory accesses}}{\text{Program}} \times \text{Miss rate} \times \text{Miss penalty}$$

$$= \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Misses}}{\text{Instruction}} \times \text{Miss penalty}$$



Cache Performance Example

- Given
 - ^{Instruction cache} I-cache miss rate = 2%, ^{Data cache} D-cache miss rate = 4%
 - Miss penalty = 100 cycles
 - Base CPI (ideal cache) = 2
 - Load & stores are 36% of instructions
- 1. Find new CPI
- 2. Compare performance of ideal and actual CPU performance (hint: assume instruction number is I)

Miss cycle per instruction

- I-cache: $0.02 \times 100 = 2$
- D-cache: $0.36 \times 0.04 \times 100 = 1.44$

^{Base CPI}
Actual CPI = $2 + 2 + 1.44 = 5.44$

Speed up = $5.44 / 2 = 2.72$

Ideal CPU is 2.72 faster than actual CPU



Average Memory Access Time (AMAT)

- Hit time is also important for performance
- Use AMAT to estimate **average** time to access data for **both** hit and miss
- **Average memory access time (AMAT)**
 - $\text{AMAT} = \text{Hit time} + \text{Miss rate} \times \text{Miss penalty}$
- Example
 - CPU with 1ns clock, hit time = **1** cycle, miss penalty = **20** cycles, I-cache miss rate = **5%**, find AMAT

$$\text{AMAT} = 1 + 0.05 \times 20 = 2\text{ns}$$

AMAT is 2 cycles per instruction



Different Cache Architectures

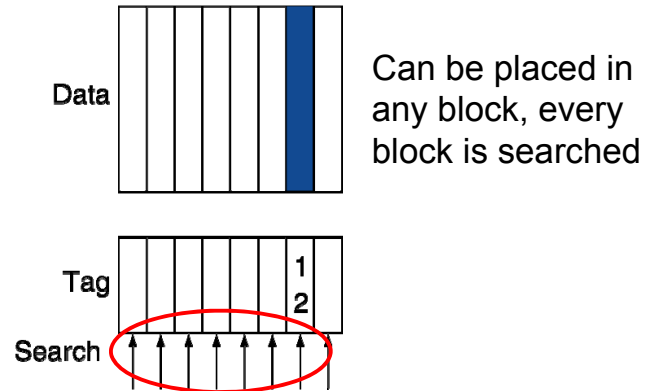
- **Direct mapped** (have discussed) (= One way set associative)
- **Fully associative**: each memory block can locate anywhere in cache
全關聯性 所有block皆在同一個set
 - Allow a given block to go in any cache entry 彈性較大，但成本較高
 - All cache entries are searched (in parallel) to locate block
 - Comparator per entry (**expensive**)

Location of a memory block with address 12 in a cache with 8 entries

Direct mapped



Fully associative



Different Cache Architectures

- **N-way Set associative:**

- Each memory block can place in a **set of** cache locations (any location in the set , each set contain **N** entries)
- cache **set number** = (Block number) modulo (number of Sets in cache)
取餘數
- all cache entries in the set are searched (in parallel) to locate block 資料可隨意放入每一個set

- **N** comparators (less expensive)

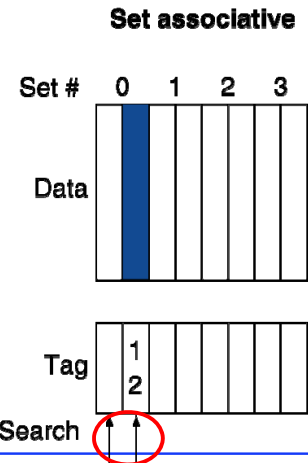
What is the location of a memory block with address 12 in a 2-way cache with 8 entries?

each set has 2 blocks

Total # entry= 8, each set has 2 blocks => 4 sets

$$\text{Set address} = 12 \bmod 4 = 0$$

block number 12 is put into set 0



Different Associativity

- For a cache with **8 entries**, **four different** cache organization

Block	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

One way set-associative (a set has one block, directed mapped)

Two-way set associative

Set	Tag	Data	Tag	Data
0				
1				
2				
3				

2-way set-associative (a set has two blocks)

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

4-way set-associative (a set has 4 blocks)

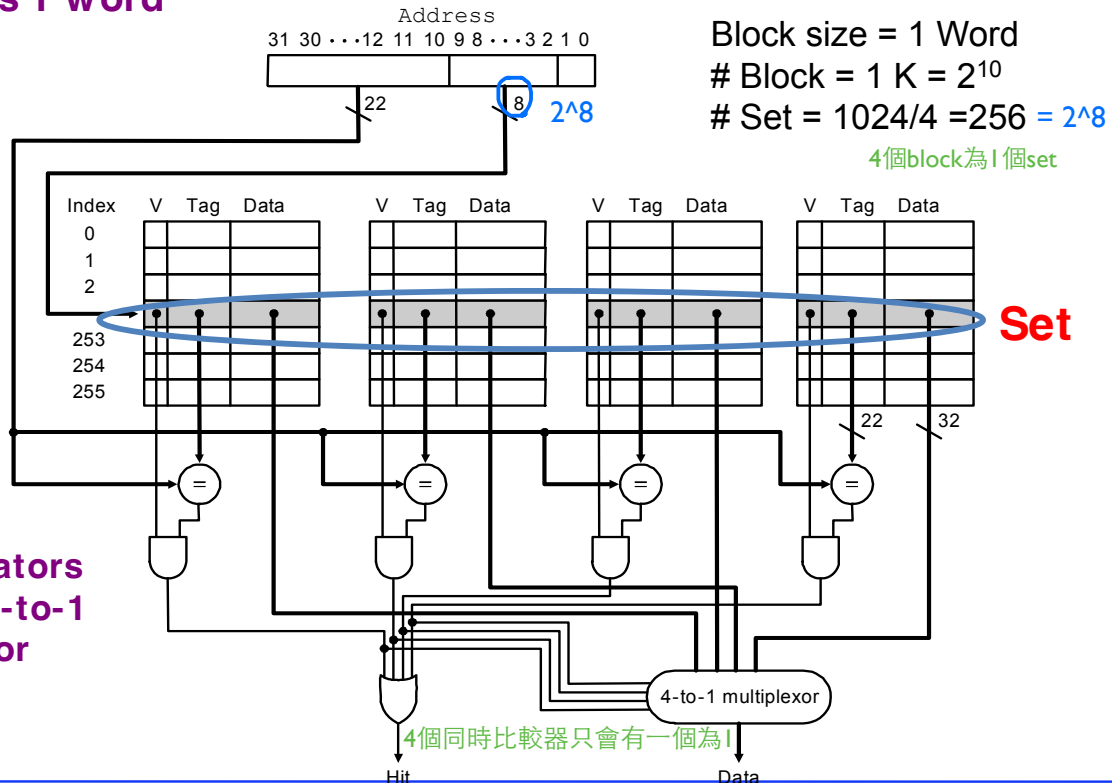
Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data

8-way set-associative (a set has 8 blocks), only one set, all blocks are search in a cache access



A 4-way set-associative Cache example

4-way set-associative cache size of cache is 1 K blocks, each block is 1 word



4-to-1 multiplexer

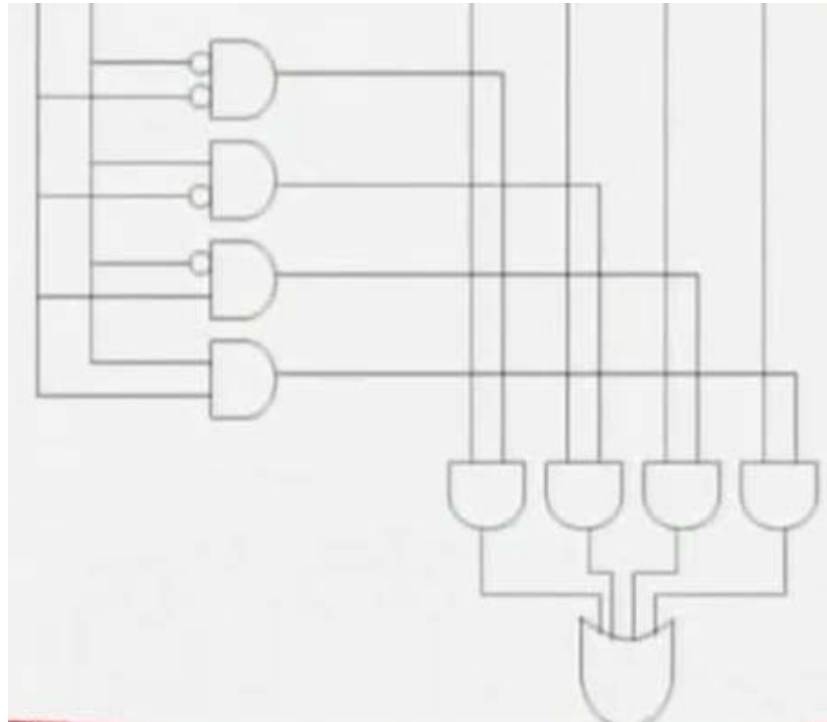
S1 S0

D0

D1

D2

D3



Example Problem

- Find the number of **misses** for a 4-block cache with (each block is **1-word**) given the following sequence of **word addresses** of memory block accesses: **0, 8, 0, 6 and 8**, for each of the following cache configurations
 - direct mapped
 - 2-way set associative (use LRU replacement policy)
 - fully associative



Associativity Example

- Direct mapped for **four 1-word blocks**
- Block addresses : **0, 8, 0, 6, 8**

Block address	Cache block
0	0(=0 mod 4)
6	2(=6 mod 4)
8	0(=8 mod 4)

Block address	Cache index	Hit/miss	Cache content after access			
			0	1	2	3
0	0	miss	Mem[0]			
8	0	miss	Mem[8]			
0	0	miss	Mem[0]			
6	2	miss	Mem[0]		Mem[6]	
8	0	miss	Mem[8]		Mem[6]	



Associativity Example

- 2-way set associative (4 misses, 1 hit)

Block address	Cache index	Hit/miss	Cache content after access			
			Set 0		Set 1	
0	0	miss	Mem[0]			
8	0	miss	Mem[0]	Mem[8]		
0	0	hit	Mem[0]	Mem[8]		
6	0	miss	Mem[0]	Mem[6]		
8	0	miss	Mem[8]	Mem[6]		

- Fully associative

Use LRU (least recently used), so Mem[8] is replaced

Block address		Hit/miss	Cache content after access			
0		miss	Mem[0]			
8		miss	Mem[0]	Mem[8]		
0		hit	Mem[0]	Mem[8]		
6		miss	Mem[0]	Mem[8]	Mem[6]	
8		hit	Mem[0]	Mem[8]	Mem[6]	

(3 misses, 2 hit)



Why Set Associate Cache?

- **Directed** mapped may introduce too many **conflict** misses
 - Conflict miss: ≥ 1 memory blocks contend a cache line
- **Fully** associative is **too expensive**
 - Minimize the number of conflict misses
- **Set associative** intends to come up with a **balance**



How Much Associativity

- Increased associativity decreases **miss rate**
 - But with diminishing returns
- Simulation of a system with 64KB D-cache, 16-word blocks, SPEC2000
 - 1-way: 10.3%
 - 2-way: 8.6%
 - 4-way: 8.3%
 - 8-way: 8.1%



Replacement Policy

- **Direct mapped:** no choice
- **Set associative**
 - Prefer non-valid entry, if there is one
 - Otherwise, choose among entries in the set using **Least-recently used (LRU)**
 - Choose the one unused for the longest time
 - Simple for 2-way, manageable for 4-way, too hard beyond that
- **Full Associative: Random**
 - Gives approximately the same performance as LRU for high associativity

Block	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

Two-way set associative

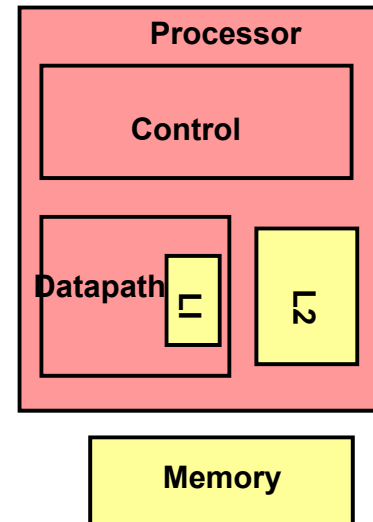
Set	Tag	Data	Tag	Data
0				
1				
2				
3				

Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data



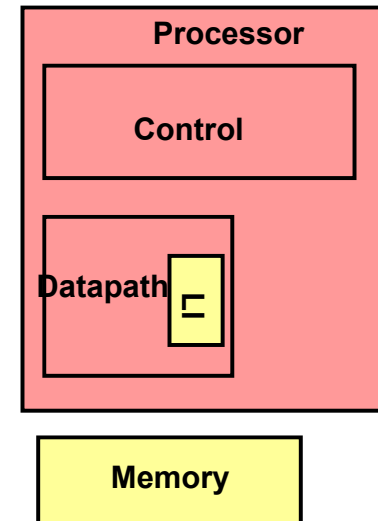
Multilevel Caches

- Use Multilevel Caches to reduce miss penalty
- **Primary cache (Level-1)** attached to CPU
 - Small, but fast
- **Level-2** cache services misses from primary cache
 - Larger, slower, but still faster than main memory
- **L-2** cache reduces the write on mem
=> reduce miss **penalty**
- Some high-end systems include L-3 cache



Multilevel Cache Example

- Given
 - CPU base CPI = **1**, clock rate = **4GHz**
 - Miss rate/instruction = 2%
 - Main memory access time = **100ns**
- With just primary cache
 - Miss penalty = $100\text{ns}/0.25\text{ns} = \mathbf{400}$ cycles
 - Effective CPI = $\mathbf{1} + \mathbf{0.02} \times \mathbf{400} = \mathbf{9}$

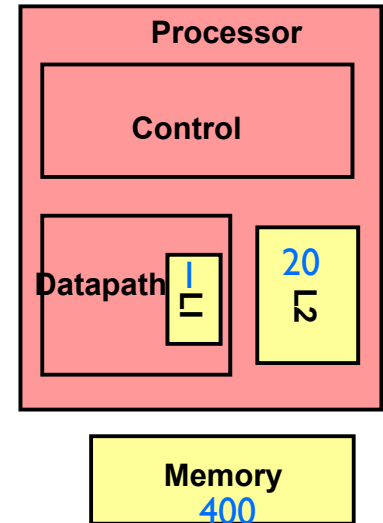


每個instruction需個cycle執行



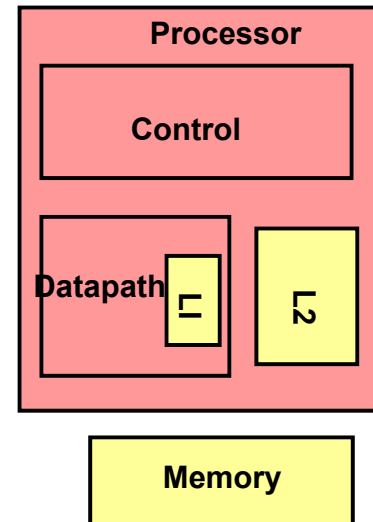
Example (cont.)

- Now CPI with L-2 Cache
- L-2 cache
 - Access time = 5ns
 - Global miss rate to main memory = 0.5%
- Primary miss with L-2 hit
 - Penalty = $5\text{ns}/0.25\text{ns} = 20 \text{ cycles}$
- Primary miss with L-2 miss
 - Extra penalty = 400 cycles
- $\text{CPI} = 1 + 0.02 \times 20 + 0.005 \times 400 = 3.4$
- $\text{Performance ratio} = 9/3.4 = 2.6$



Multilevel Cache Considerations

- Primary cache
 - Focus on minimal **hit time**
- L-2 cache
 - Focus on **low miss rate** to avoid main memory access
 - Hit time has less overall impact
- Results
 - L-1 cache usually smaller than a single cache
 - L-1 block size smaller than L-2 block size





成功大學
National Cheng Kung University

Backup Slides

