การจัดการระบบหน่วยความจำเป็นลำดับชั้น
Memory Hierarchy

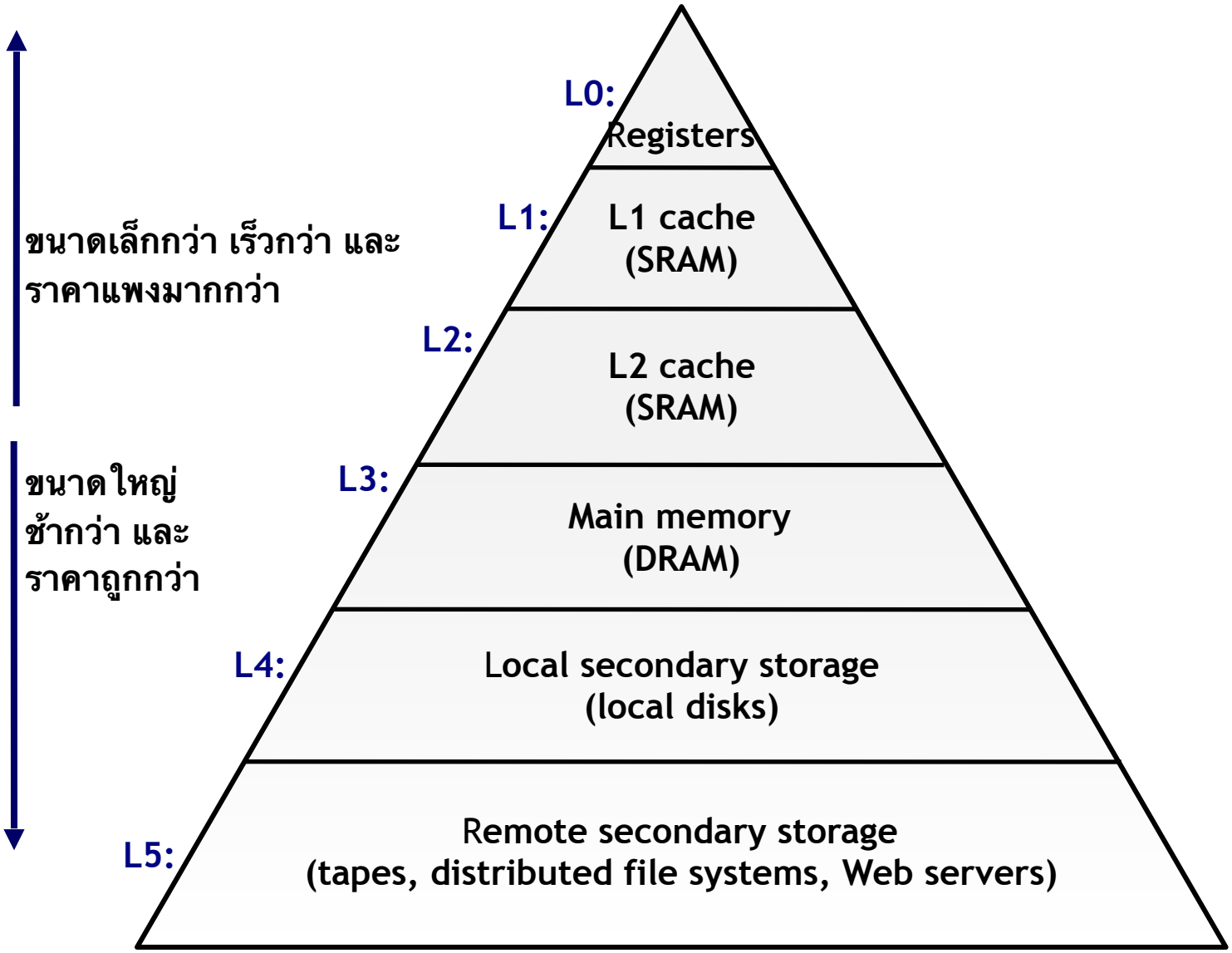
Principle of Locality

- ธรรมชาติของโปรแกรม
  - วนซ้ำชุดคำสั่งเดิมๆ
  - ประมวลผลข้อมูลที่ address เดิมและข้างเคียงซ้ำไปซ้ำมา
- Temporal locality
  - ข้อมูลที่ถูกอ้างถึง ณ ขณะนี้ จะถูกอ้างถึงอีกในเวลามาถัดไปไม่นาน
- Spatial locality
  - ถ้าขณะนี้อ้างถึงข้อมูล ณ ตำแหน่งหนึ่ง ในเวลาถัดไปไม่นานจะอ้างถึงข้อมูลในตำแหน่ง address ที่อยู่ติดๆกัน

การใช้ Locality ให้เกิดประโยชน์สูงสุด

- จัดระบบหน่วยความจำให้เป็นลำดับชั้น (memory hierarchy)
  - ทุกอย่างเก็บลง disk
  - ทำก๊อปปี้ของข้อมูลใน disk ที่ถูกอ้างถึงมากๆและข้อมูลที่ใกล้เคียงกันมาเก็บไว้ในหน่วยความจำหลัก (main memory)
  - ทำก๊อปปี้ของข้อมูลใน main memory ที่ถูกอ้างถึงมากๆและข้อมูลที่ใกล้เคียงกันมาเก็บไว้ในแคช (cache)

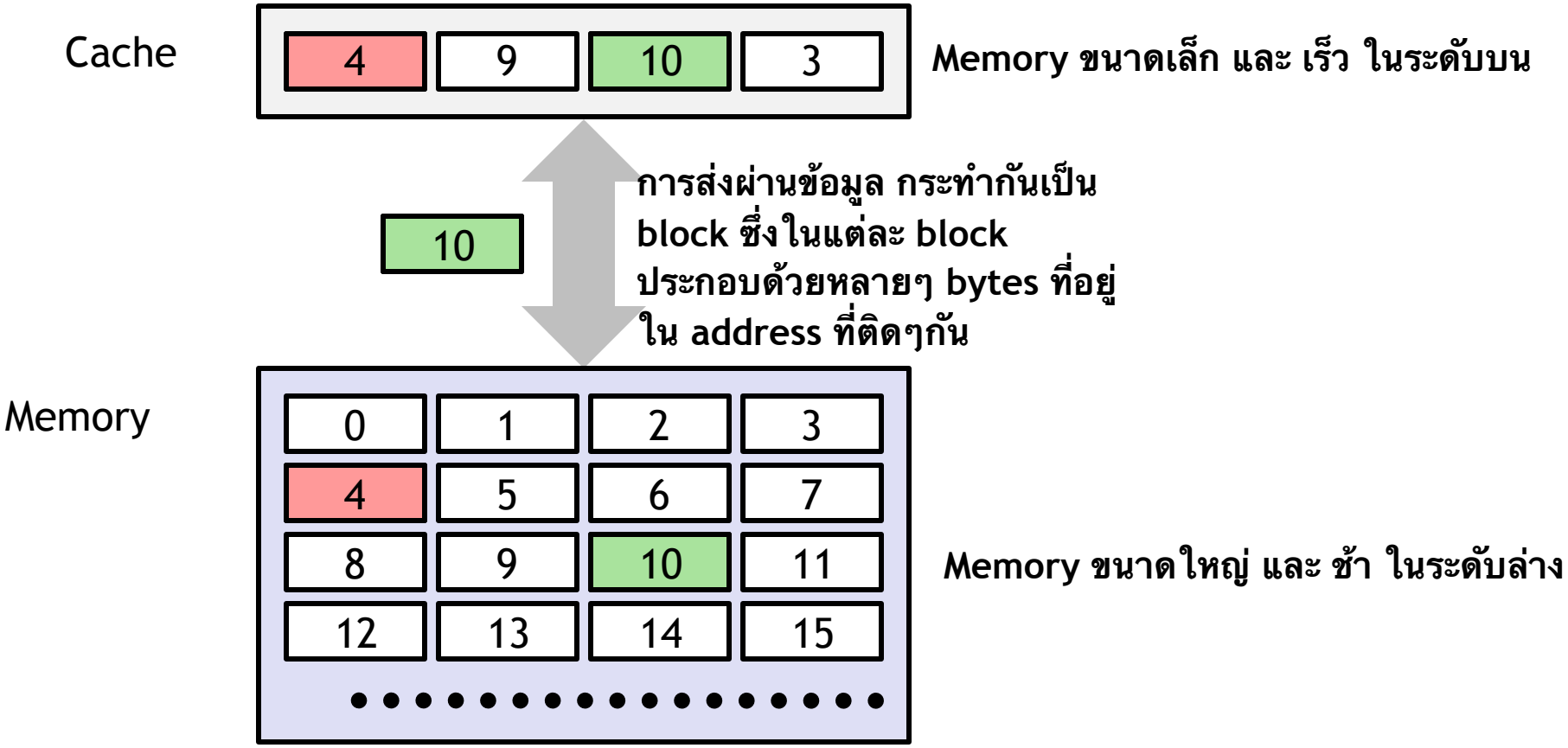
ลำดับชั้นต่างๆ ใน Memory Hierarchy



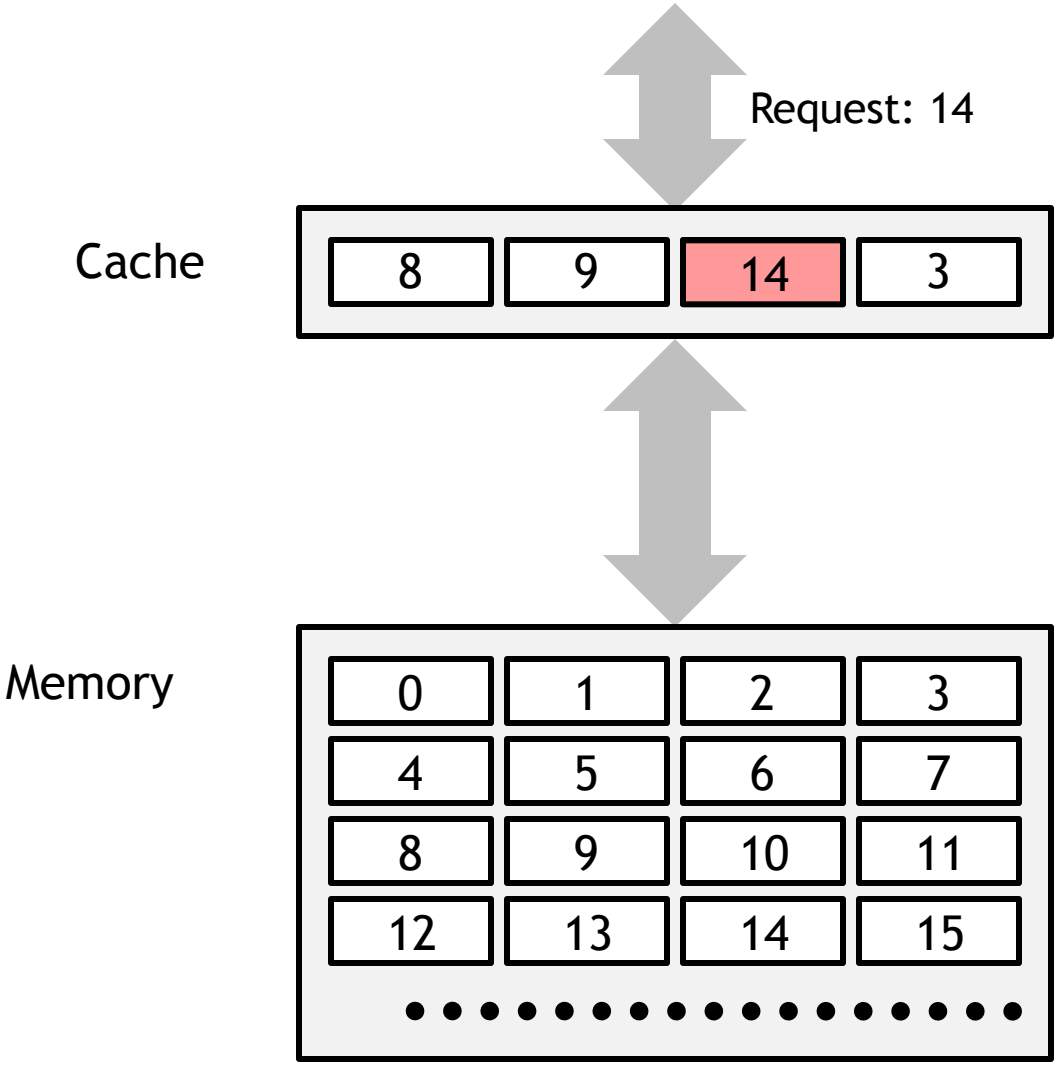
Caches

- *Cache:* คือหน่วยความจำขนาดเล็กแต่เร็ว ทำหน้าที่เก็บข้อมูลเฉพาะบางส่วน จากข้อมูลทั้งหมดที่มีอยู่ในหน่วยความจำ (ที่มีขนาดใหญ่ แต่ช้ากว่ามาก)
- สมรรถนะที่เพิ่มขึ้นจากการมี cache เป็นไปได้เพราะโปรแกรมมี locality สูง
- การจัดการ memory เป็นลำดับชั้น ให้ความรู้สึกว่ามี memory ขนาดใหญ่ และราคาถูก แต่มีความเร็วมาก
  - ณ เวลาใดๆ ดึงเฉพาะข้อมูลบางส่วนที่ใช้งานเก็บไว้ใน memory ระดับสูง

แนวคิดหลักของ Cache



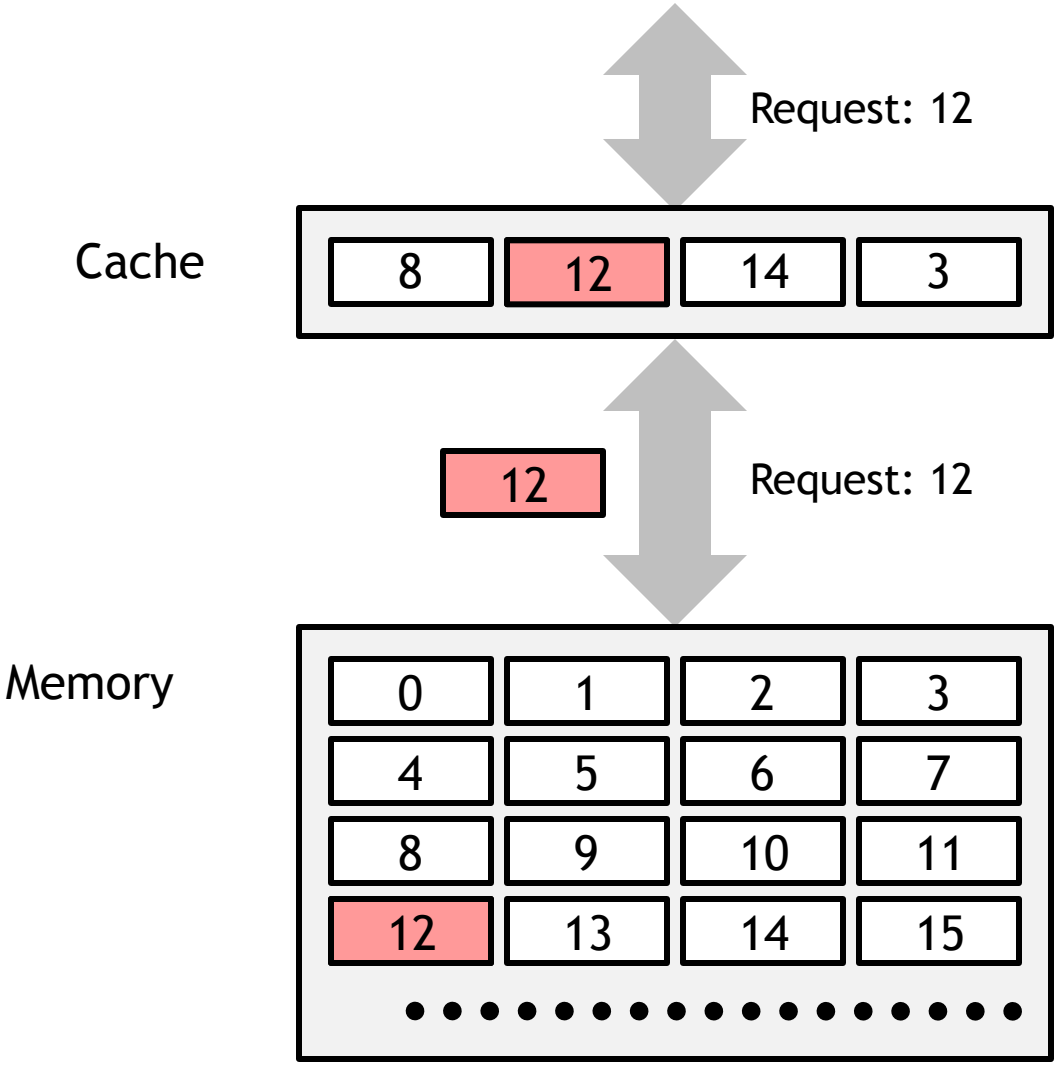
Cache Hit



ต้องการข้อมูลใน *block b*

***Block b* อยู่ใน *cache*:**
Hit!

Cache Miss



ต้องการข้อมูลที่อยู่ใน *block b*

ไม่พบ *b* ใน *cache:*
Miss!

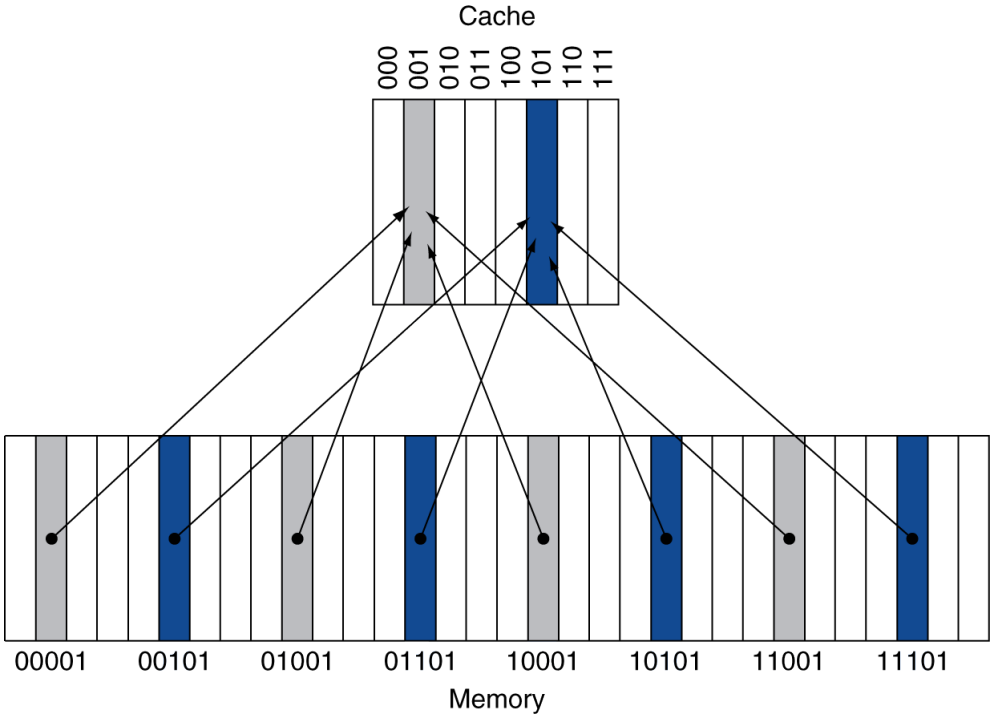
ต้อง *transfer b* จาก
หน่วยความจำในระดับล่าง

ชนิดของ Cache Miss

- Cold (compulsory) miss
  - เกิดในตอนเริ่มต้นที่ cache ยังว่างเปล่า
- Conflict miss
  - เกิดเมื่อ address ต่างกัน map เข้าที่ block เดียวกันใน cache
- Capacity miss
  - เกิดเมื่อ locality reference ของโปรแกรมมีขนาดมากกว่า cache

Direct Mapped Cache

- การคำนวณหา block ที่ address ที่อ้างถึง map ไปหา
  - (Block address) modulo (จำนวน block ใน cache)
  - ทำไมจำนวน block ใน cache จึงเป็น power of two (จำนวนที่อยู่ในรูป 2^n โดย n เป็นจำนวนเต็ม >= 1



Tags และ Valid Bits

- จะรู้ได้อย่างไรว่า address ใดกันแน่ที่อยู่ใน block ของ cache เมื่อ address หลายๆตัว map เข้าได้ใน block เดียวกัน
  - ใช้ Tag เป็นตัวบอกความแตกต่างของ address ที่ map เข้าหา block เดียวกัน
- Valid bit ใช้บอกว่า block ของ cache ยังว่างเปล่า หรือ มีข้อมูลที่ใช้ได้อยู่

ตัวอย่างการใช้งาน Cache

- 8-blocks, 1 byte/block, direct mapped

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

ตัวอย่างการใช้งาน Cache

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Miss	110

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

ตัวอย่างการใช้งาน Cache

Word addr	Binary addr	Hit/miss	Cache block
26	11 010	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		

ตัวอย่างการใช้งาน Cache

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Hit	110

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		

ตัวอย่างการใช้งาน Cache

Word addr	Binary addr	Hit/miss	Cache block
26	11 010	Hit	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		

ตัวอย่างการใช้งาน Cache

Word addr	Binary addr	Hit/miss	Cache block
30	11 110	Miss	110

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		

ตัวอย่างการใช้งาน Cache

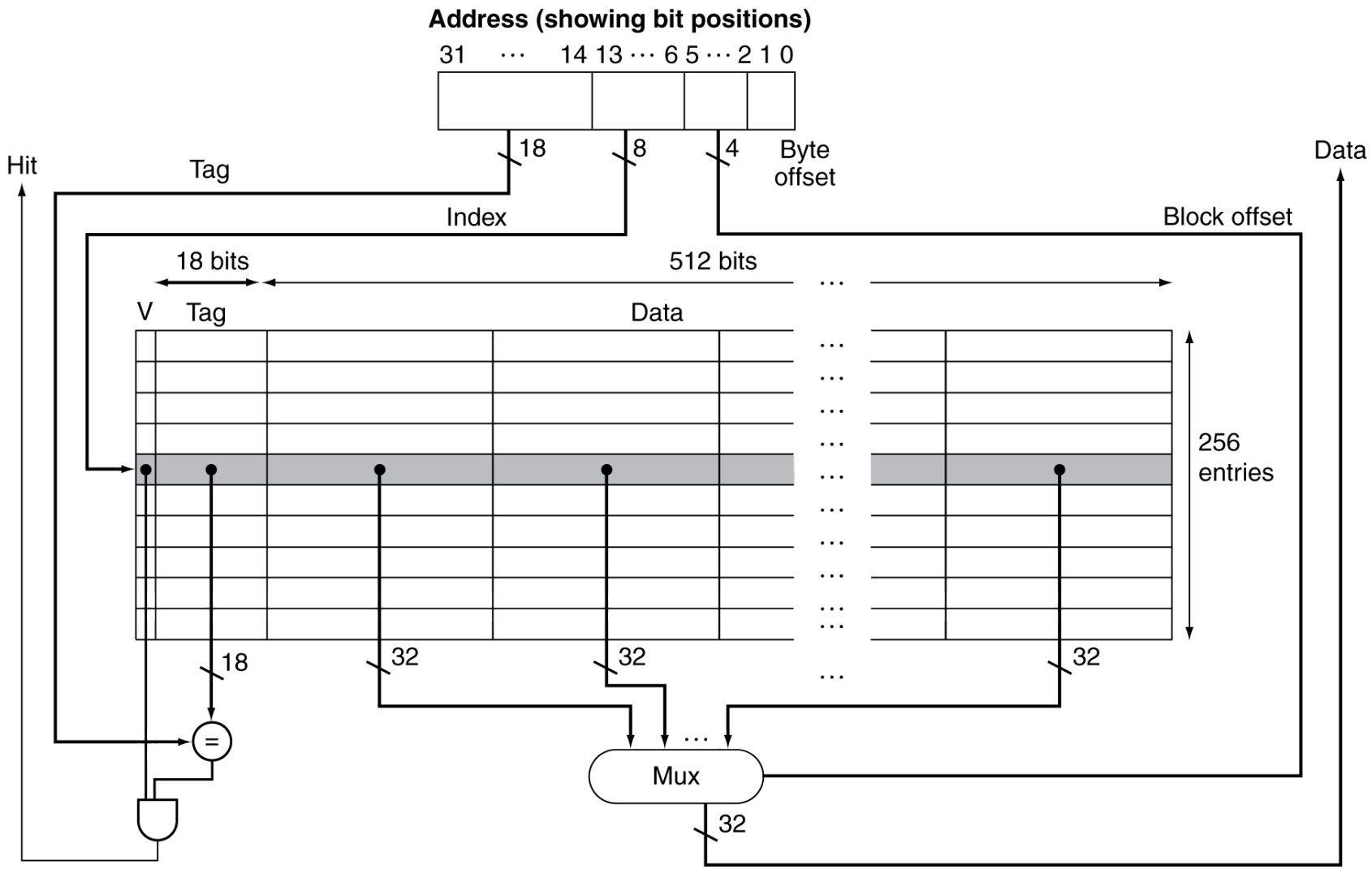
Word addr	Binary addr	Hit/miss	Cache block
30	11 110	Miss	110

Index	V	Tag	Data
000	N		
001	N		
010	Y	11	Mem[11010]
011	N		
100	N		
101	N		
110	Y	11	Mem[11110]
111	N		

ขนาดของ Block

- ทำไมเราจึงควรให้ขนาดของ block มากกว่า 1 byte
  - ลดการ miss เพราะ spatial locality ของโปรแกรม
- แต่ถ้าขนาด cache มีจำกัด การเพิ่มขนาด block อาจทำให้ miss เพิ่มขึ้นได้ (ทำไม?)

การแบ่ง Bit เพื่ออ้างข้อมูลใน Cache



ปัญหาของ Store เมื่อมี Cache

- ถ้า write เฉพาะใน cache จะทำให้หน่วยความจำหลักไม่มี copy ของ block ที่ update ล่าสุด
- เราต้องการให้ memory hierarchy ยังคงคุณสมบัติ inclusion
- การจัดการกับการ write
  - Write-Through
  - Write-Back

Write-Through

- Update หน่วยความจำหลัก ทุกๆครั้งที่ write ไปที่ cache
- แต่ทำให้การ store ช้าลงมาก เพราะหน่วยความจำหลักช้ากว่า cache มาก
  - CPI = 1, 10% เป็น store และ memory access ใช้เวลา 100 cycles
    - Effective CPI = 1 + 0.1×100 = 11
- ผิดจุดประสงค์ของการมี cache ที่ต้องการสมรรถนะที่เพิ่มขึ้น

Write-Back

- ไม่ update หน่วยความจำหลักในทันทีที่ store ลง cache
  - จำว่ามีการ store และบ่งให้ block นั้นเป็น dirty
- ขณะที่ dirty block โดนเปลี่ยนออกจาก cache update หน่วยความจำหลักในเวลานี้
  - ทุก access ที่ CPU ขอจะต้องมาผ่าน cache ก่อน
  - ไม่จำเป็นต้อง update หน่วยความจำหลักในทันที

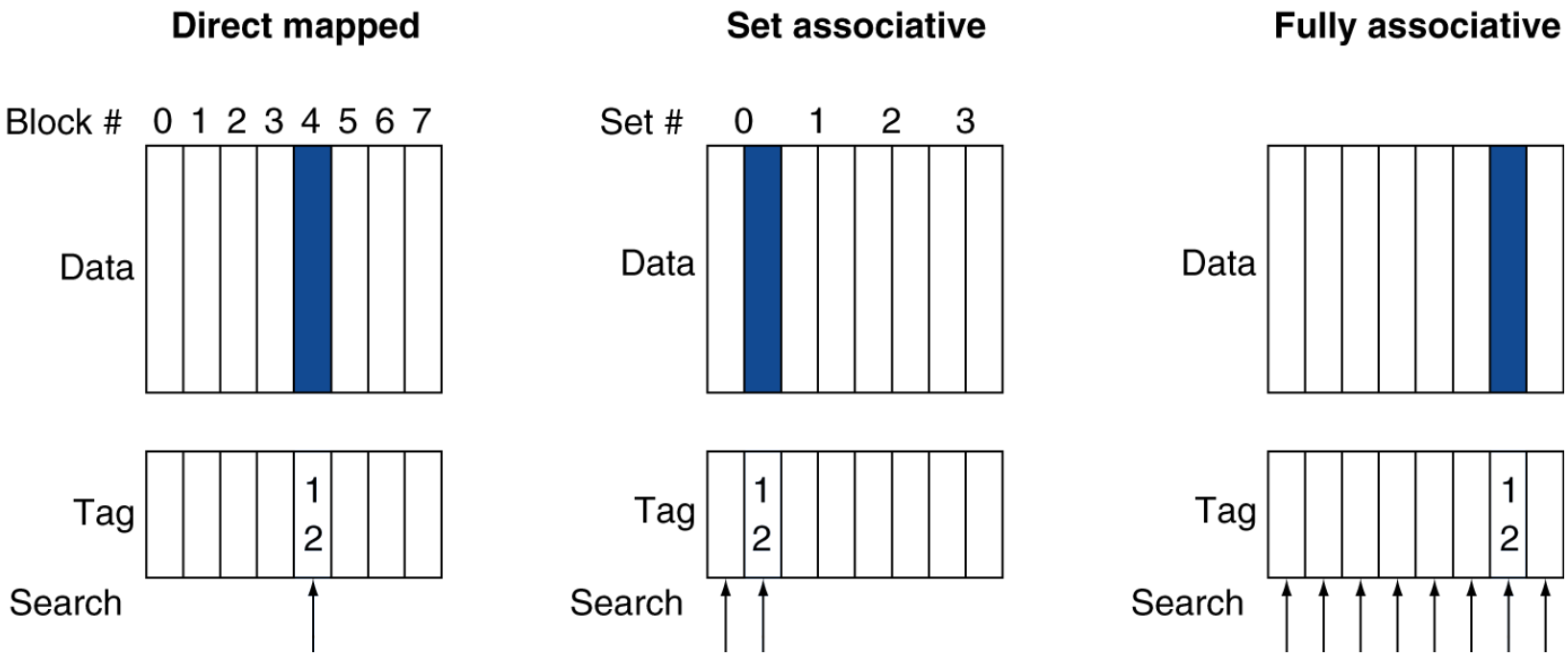
ทำอย่างไรเมื่อ cache Miss

- หยุด (stall) pipeline
- Memory controller นำ block จากหน่วยความจำหลัก เข้ามาใน cache
- เมื่อ block ใหม่นี้เข้ามาอยู่ใน cache เรียบร้อยแล้ว ดำเนินการต่อเหมือน cache hit
- Write-through cache อาจจะไม่นำ block นี้เข้ามาตอน write miss
  - Write no-allocation
- เราจะเน้นไปที่ write-back write-allocation

Associative Cache

- Fully associative
  - Block สามารถจะไปอยู่ใน entry ใดๆก็ได้ใน cache
  - การจะบอก cache hit หรือ miss ต้องค้นหาทุกๆ entry
  - สมรรถนะต่ำ
- *n*-way set associative
  - แต่ละ set มี n blocks
  - การหาว่า block อยู่ใน set ใด ใช้สมการ
    - (Block number) modulo (#Sets in cache)
  - Index ไปที่ set ก่อน แล้วค้นหา n block
  - สมรรถนะสูงขึ้นกว่า fully associative cache

Associative Cache



ตัวอย่าง Associativity

- สำหรับ cache ที่มี 8 entries

One-way set associative
(direct mapped)

Block	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

Two-way set associative

Set	Tag	Data	Tag	Data
0				
1				
2				
3				

Four-way set associative

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

Eight-way set associative (fully associative)

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

ตัวอย่าง Associativity

- เปรียบเทียบ cache ขนาด 4-block
  - Direct mapped, 2-way set associative, fully associative
  - ลำดับการ access block: 0, 8, 0, 6, 8
- Direct mapped

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

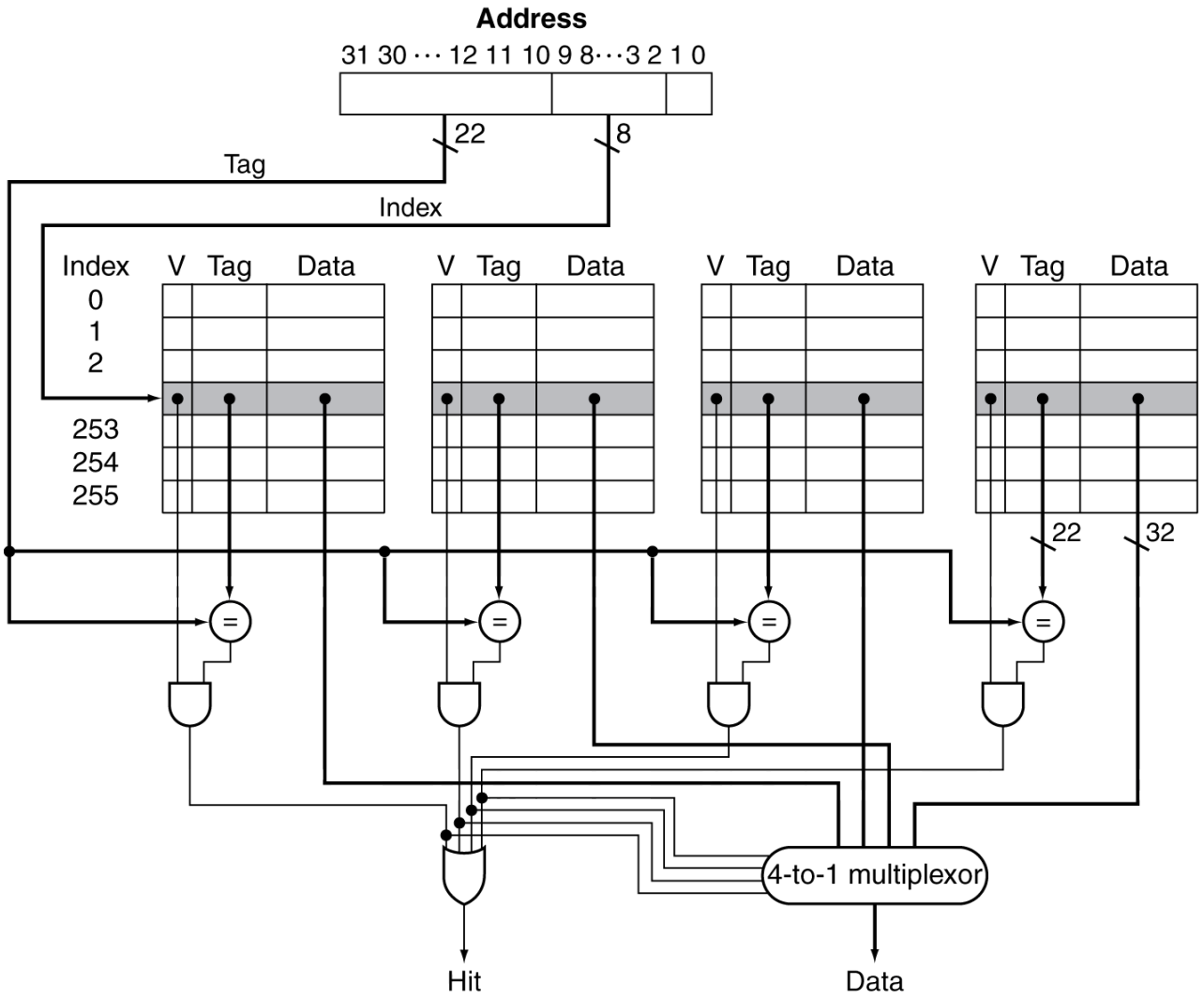
- 2-way set associative

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

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]	

Set Associative Cache Organization

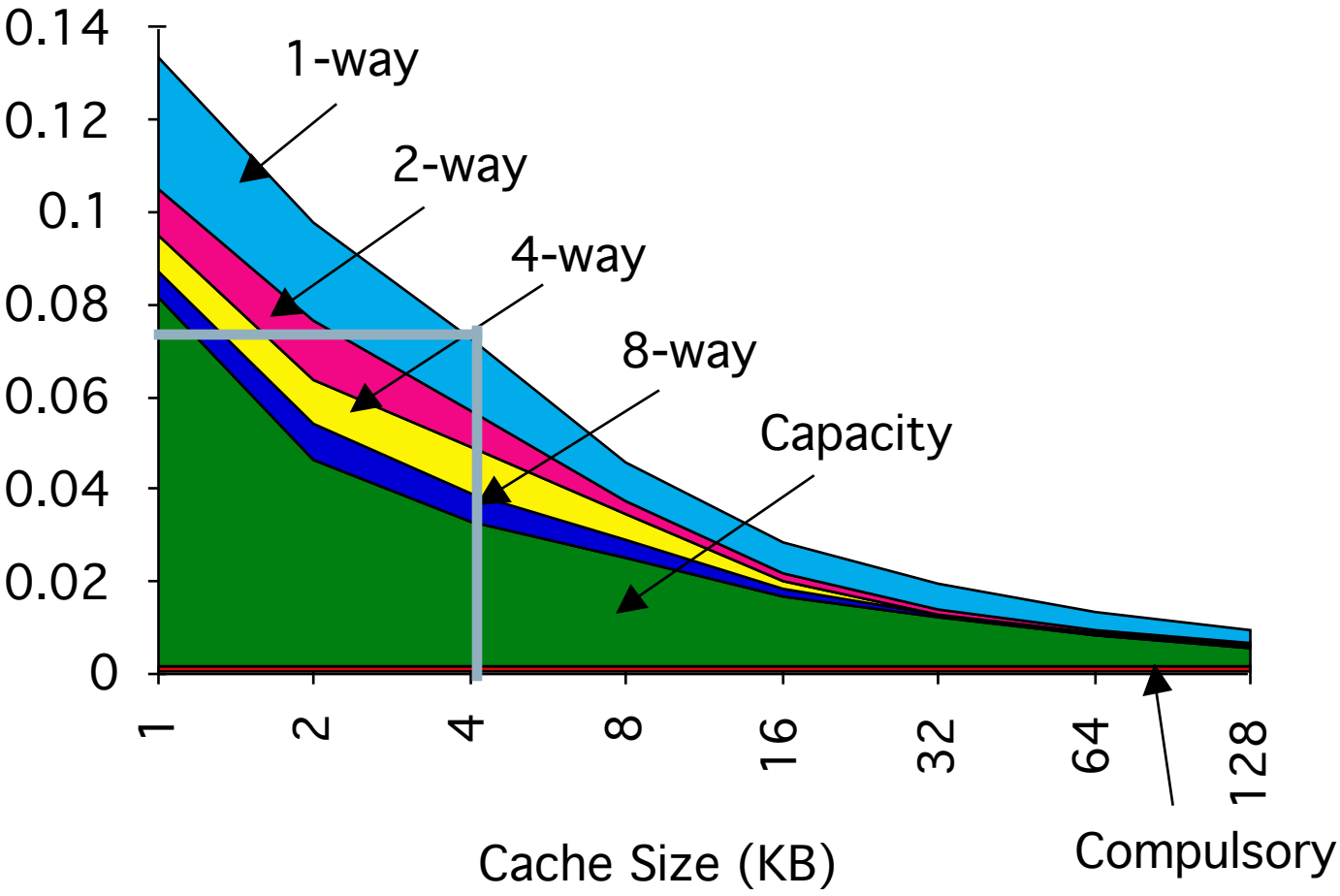


ผลที่ได้เมื่อเพิ่ม Associativity

- Miss rate ลดลง
  - แต่ยิ่งเพิ่มมากขึ้น อัตราการลดลงจะยิ่งน้อยลง (diminishing return)
- ผลการทดลองเมื่อกับ data cache ขนาด 64KB 16-word ต่อ block ด้วยโปรแกรมใน SPEC2000
  - 1-way: 10.3%
  - 2-way: 8.6%
  - 4-way: 8.3%
  - 8-way: 8.1%

2:1 Cache Rule

miss rate 1-way cache size X = miss rate 2-way cache size X/2



Replacement Policy

- Direct mapped: ไม่มีทางเลือกอื่น ต้องแทนที่ block ที่มี
- Set associative
  - non-valid
  - เลือก block ใด block หนึ่งใน set
- Least-recently used (LRU)
  - เลือก block ที่ไม่ได้ใช้มาเป็นเวลานานที่สุด
- Random
  - มีสมรรถนะใกล้เคียงกับ LRU (ไม่น่าเชื่อ!)

บทสรุป

- หน่วยความจำที่เล็กจะเร็ว หน่วยความจำที่ใหญ่จะช้า
- Cache ให้ความรู้สึกว่าเรามีหน่วยความจำที่ใหญ่แต่เร็ว
- Cache ใช้การได้ดีเพราะโปรแกรมมี locality สูง
- Memory hierarchy
  - L1 cache ↔ L2 cache ↔ … ↔ DRAM ↔ disk

แบบฝึกหัด

- กำหนดให้ระบบใช้ address ขนาด 32 บิท และมี cache ขนาด 128 KB แบบ direct-mapped โดยกำหนดให้หนึ่ง block มี 16 bytes จงคำนวณหา:
  - จำนวนบิทสำหรับ byte offset
  - จำนวนบิทสำหรับ index
  - จำนวนบิทสำหรับ tag field
  - ระบุตำแหน่งของบิทสำหรับ offset index และ tag ด้วย
- ถ้าเราเปลี่ยนจาก direct-mapped เป็น 8-way set associative จะกระทบต่อค่าบิทสำหรับ offset index และ tag อย่างไรบ้าง