



01076114

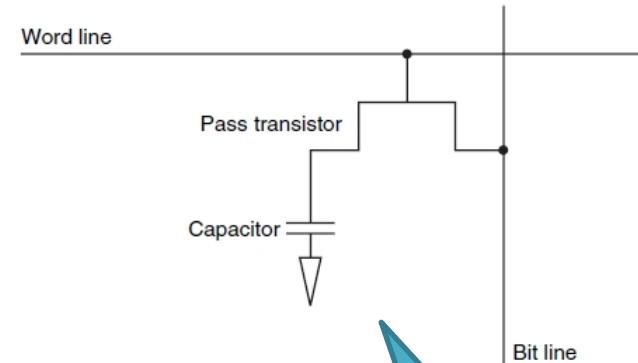
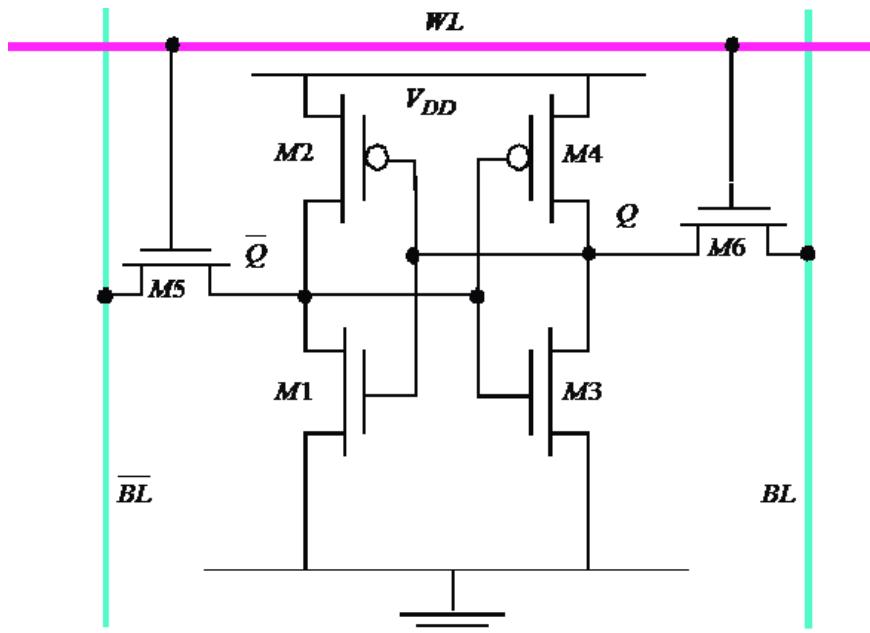
องค์ประกอบและสถาปัตยกรรมคอมพิวเตอร์
Computer Organization and Architecture

Memory Hierarchy



Dynamic RAM vs Static RAM

- โครงสร้างของ DRAM จะง่ายกว่า SRAM มาก
- DRAM จะต้องมีการ refresh เพื่อคงข้อมูลไว้



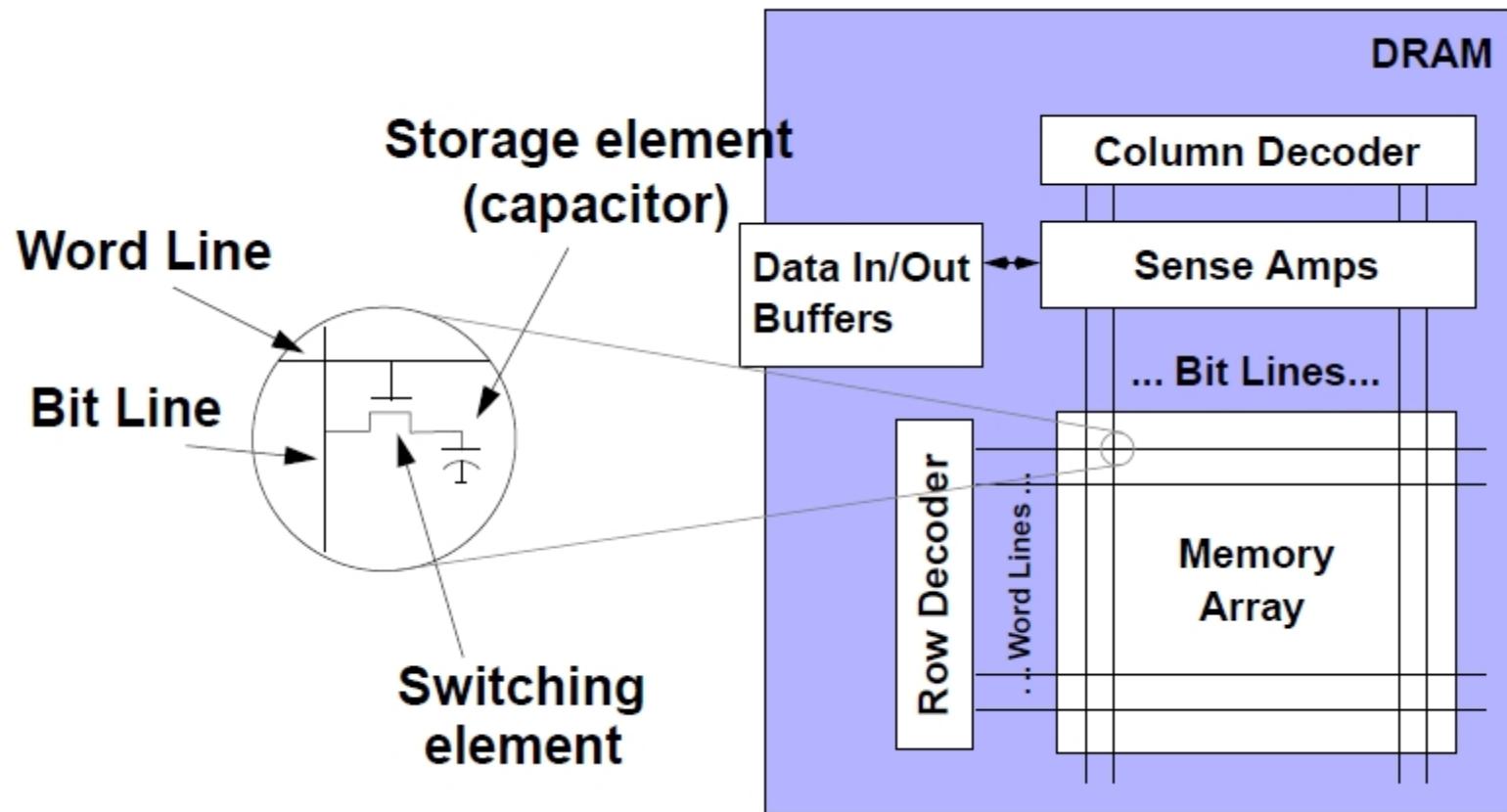
SRAM

DRAM

DRAM Organization



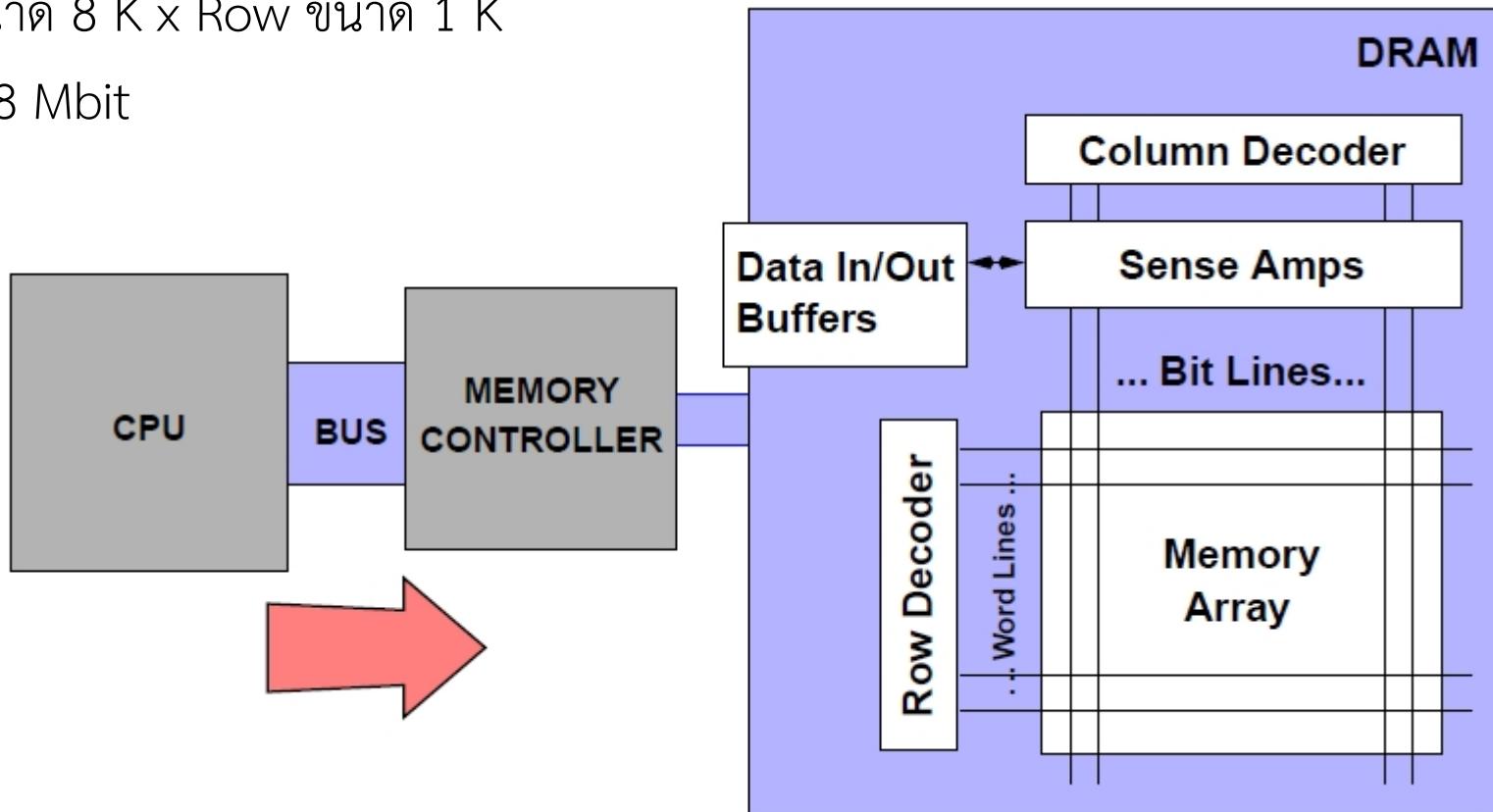
- DRAM แต่ละบิตจะเป็นเพียง Capacitor ที่เก็บประจุมาก-น้อยแทน 0 หรือ 1
- มีiranซิสเตอร์ทำหน้าที่เป็นสวิตซ์ให้ข้อมูลเข้าหรือออก





DRAM Organization

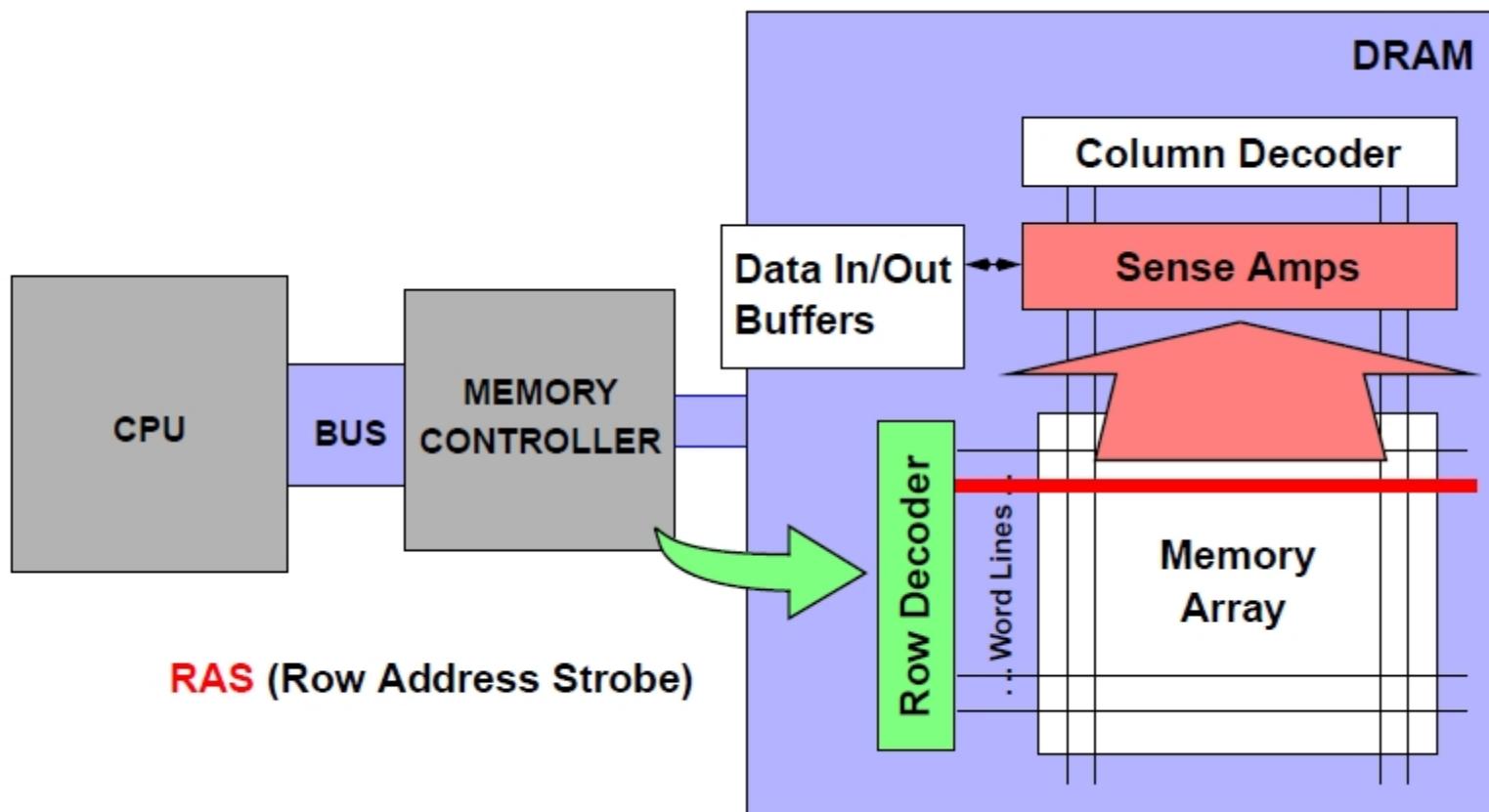
- DRAM จะติดต่อกับ CPU ผ่าน Memory Controller เนื่องจาก DRAM มีการทำงานที่ซับซ้อนกว่า Static RAM โดย DRAM จะจัดโครงสร้างเป็น array of bits เช่น col. ขนาด $8\text{ K} \times \text{Row}$ ขนาด 1 K
 $= 8\text{ Mbit}$





DRAM Organization

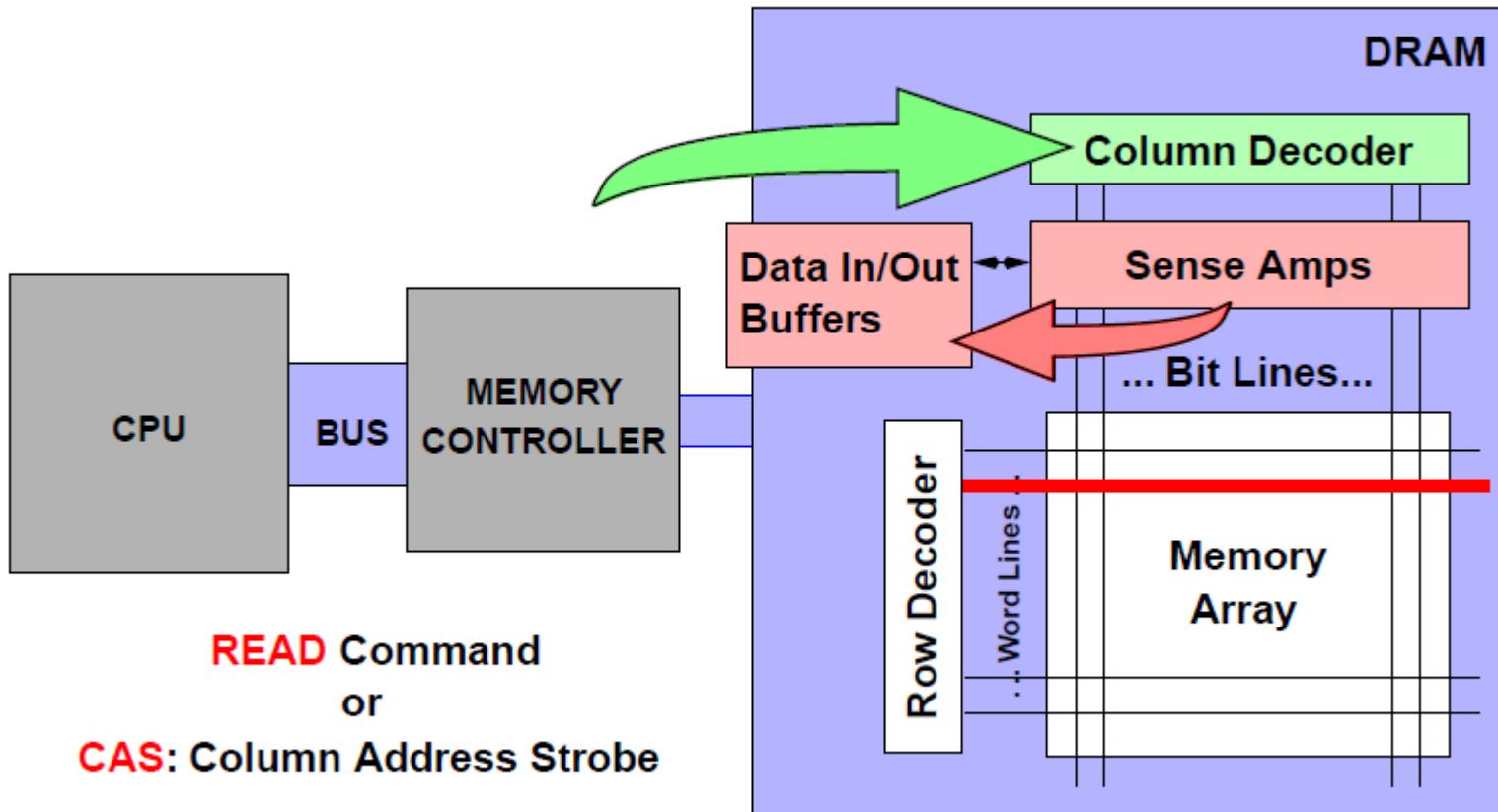
- ในการทำงานของ DRAM จะเริ่มจากการเลือกแถว โดย Mem. Ctrl. จะส่ง สัญญาณ RAS ออกไป DRAM จะส่งข้อมูลของทั้งแถวไปที่ Sense Amps





DRAM Organization

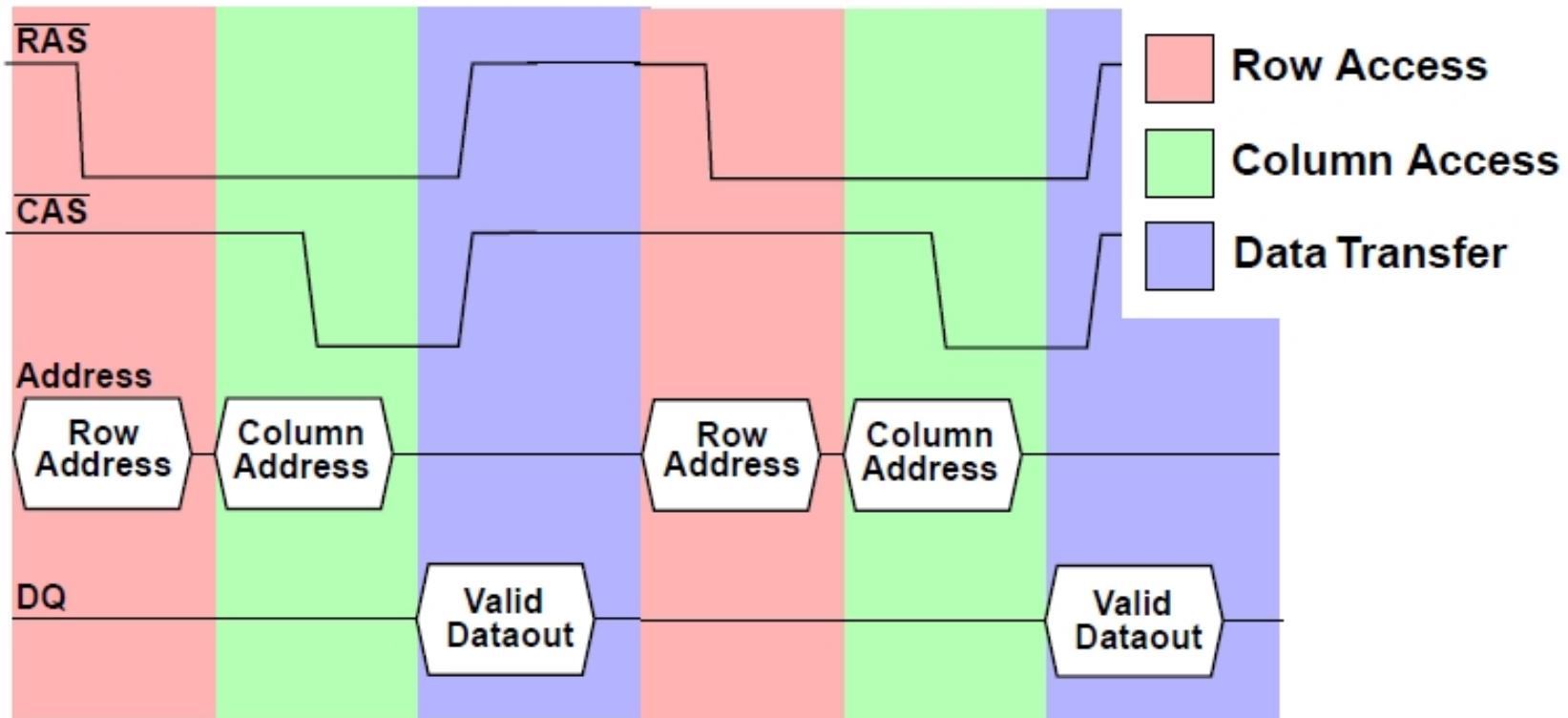
- จากนั้นจะเลือก Column โดยส่งสัญญาณ CAS จากนั้น Sense Amps จะส่ง ข้อมูล bit ที่เลือก มาไว้ที่ Data In/ Out Buffers และส่งให้ Mem. Ctrl. ต่อไป





DRAM Timing

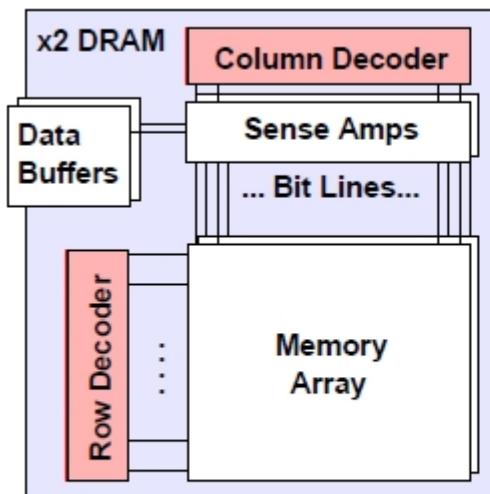
- สามารถเขียนเป็น Timing Diagram ได้ดังนี้



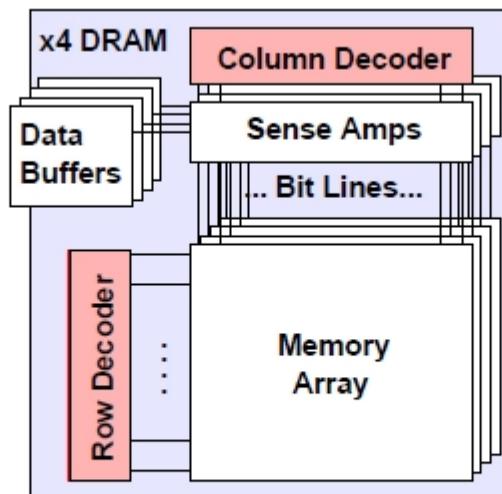


DRAM Physical Organization

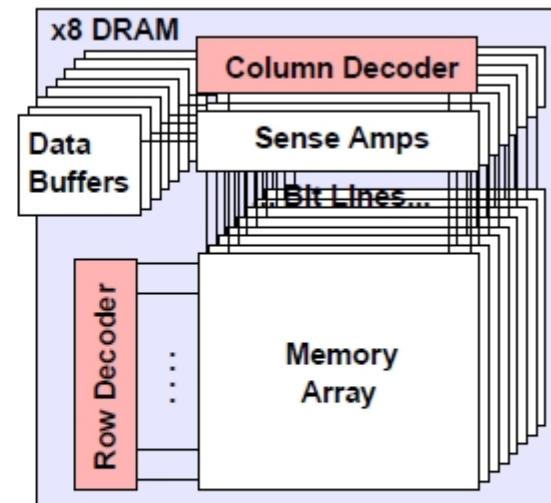
- เนื่องจาก DRAM เป็น array of bits ดังนั้นการเชื่อมต่อทาง physical จึงมักใช้ DRAM Chip หลายๆ ตัวมาต่อเข้าด้วยกัน (แต่ละชิป อาจจะมีหลายบิต)



x2 DRAM



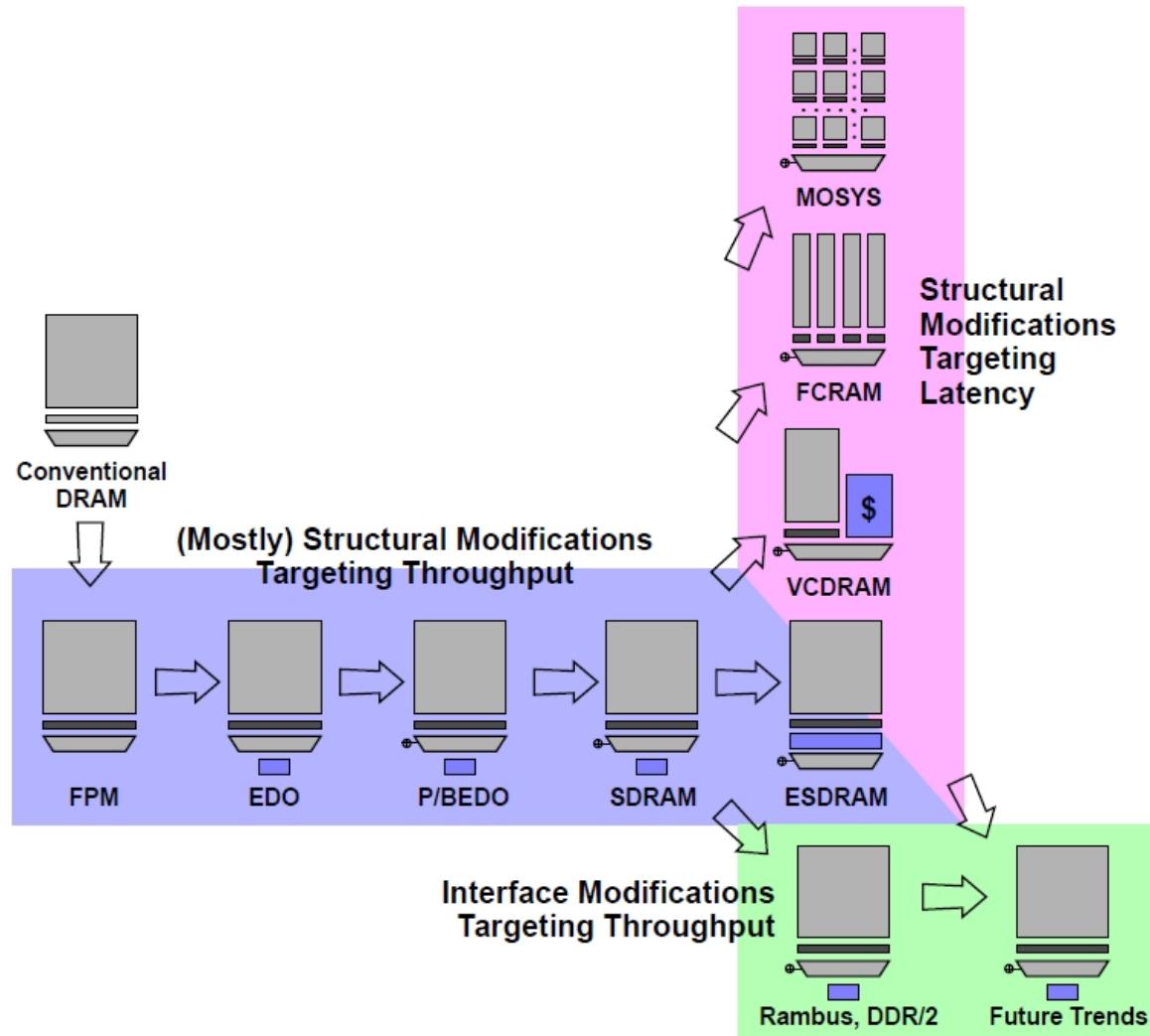
x4 DRAM



x8 DRAM



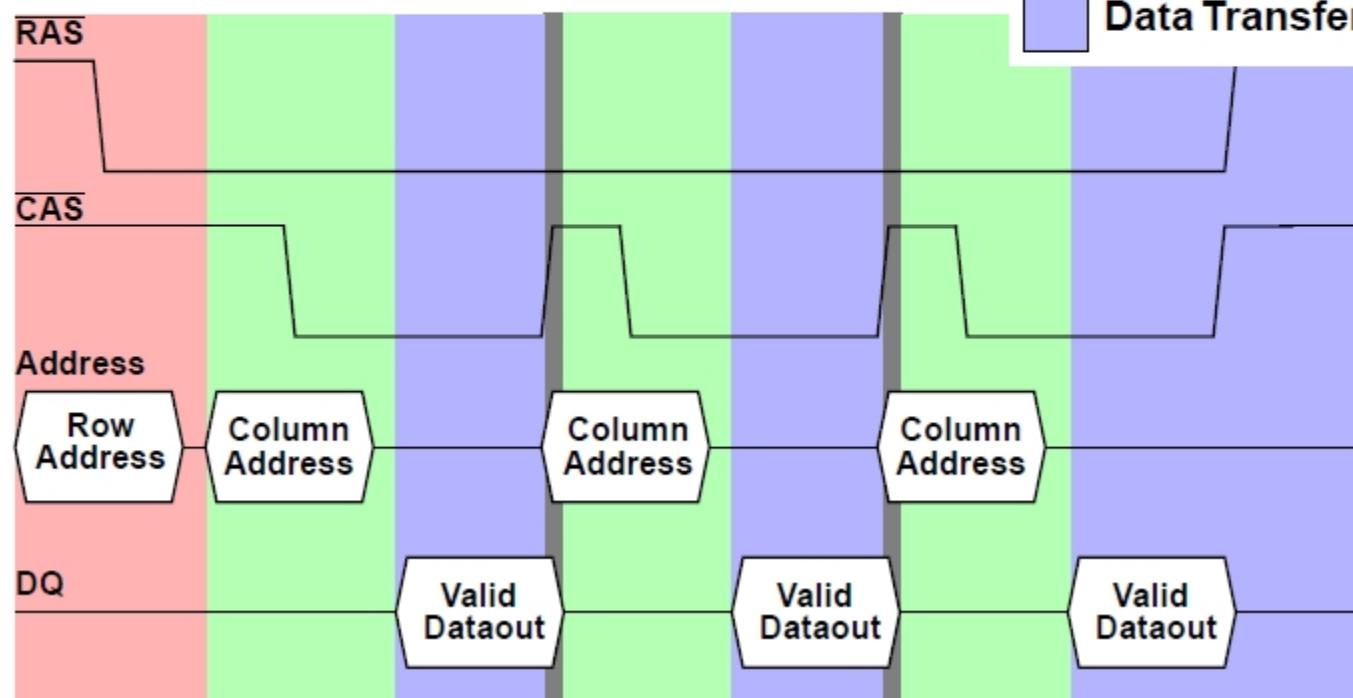
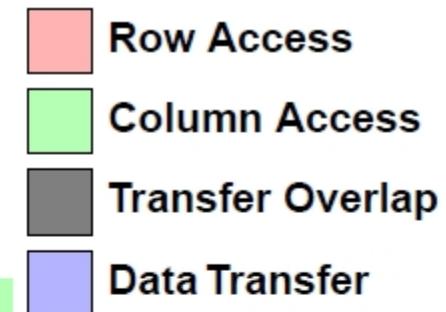
DRAM Evolutionary





DRAM Evolution

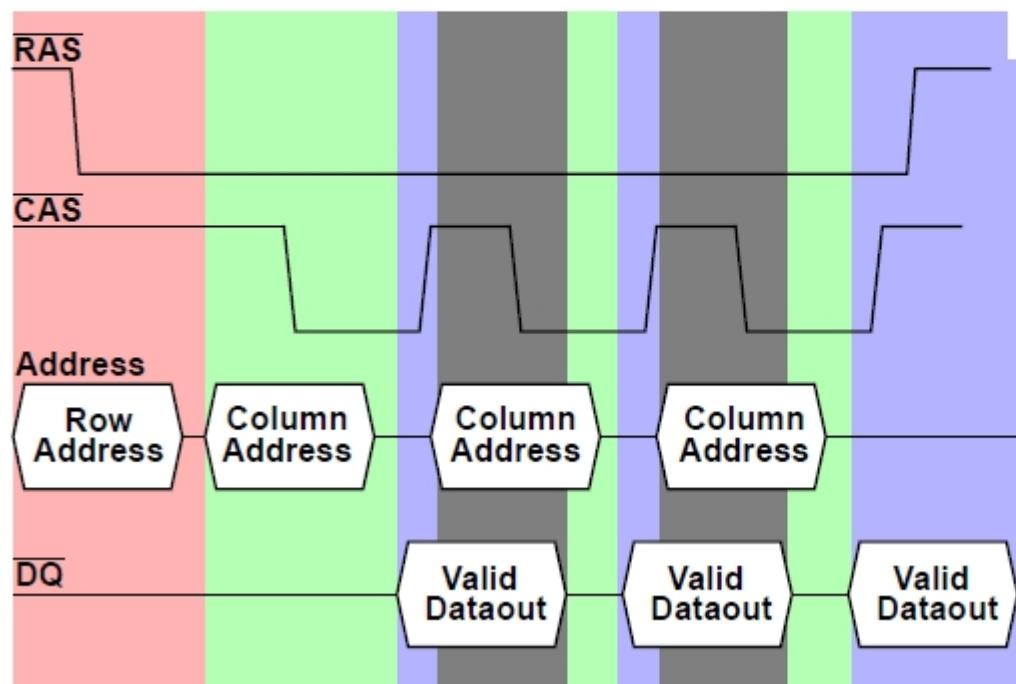
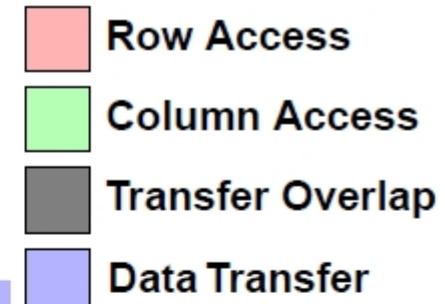
- Fast Page Mode เป็นเทคนิคในการคงค่า RAS เอาไว้ ดังนั้นสามารถจะส่ง CAS ไปเรื่อยๆ ตราบได้ทีบังคงเรียก ข้อมูลจาก Row เดิม ทำให้ลดเวลาได้





DRAM Evolution

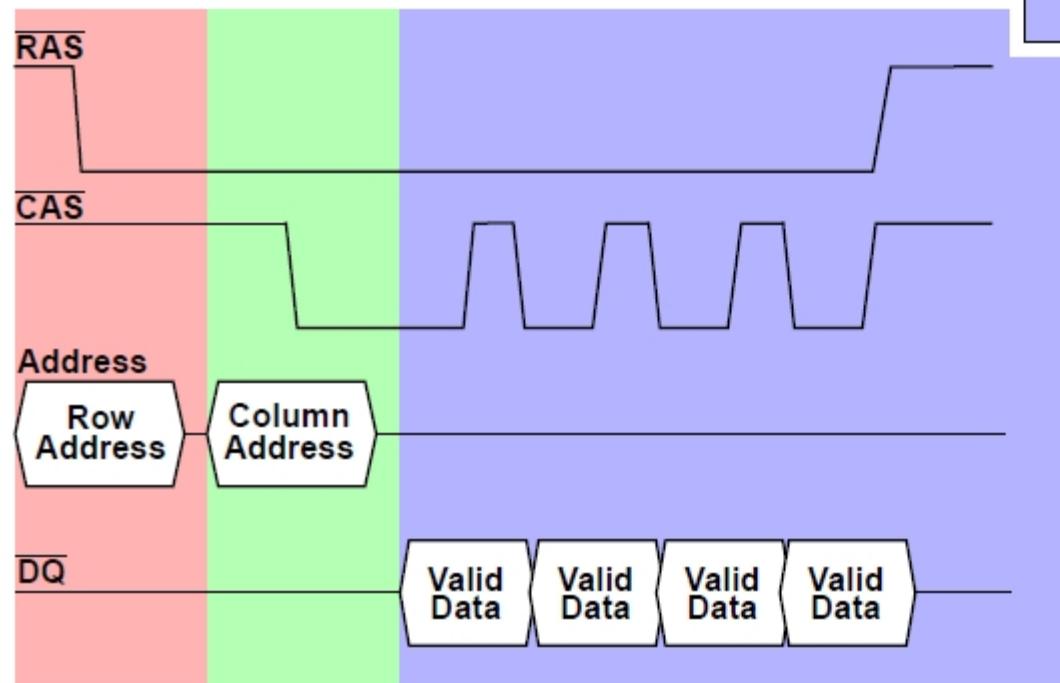
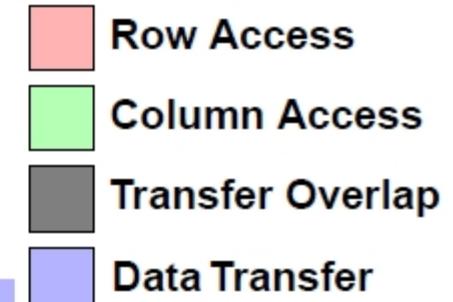
- Extended Data Out (EDO) เป็นเทคนิคในการส่งค่า CAS ในจังหวะที่เหลือมเวลา กับ Data Out โดยจะคงค่า Data Out ไว้ช่วงเวลาหนึ่ง แม้ CAS จะเปลี่ยนแล้ว





DRAM Evolution

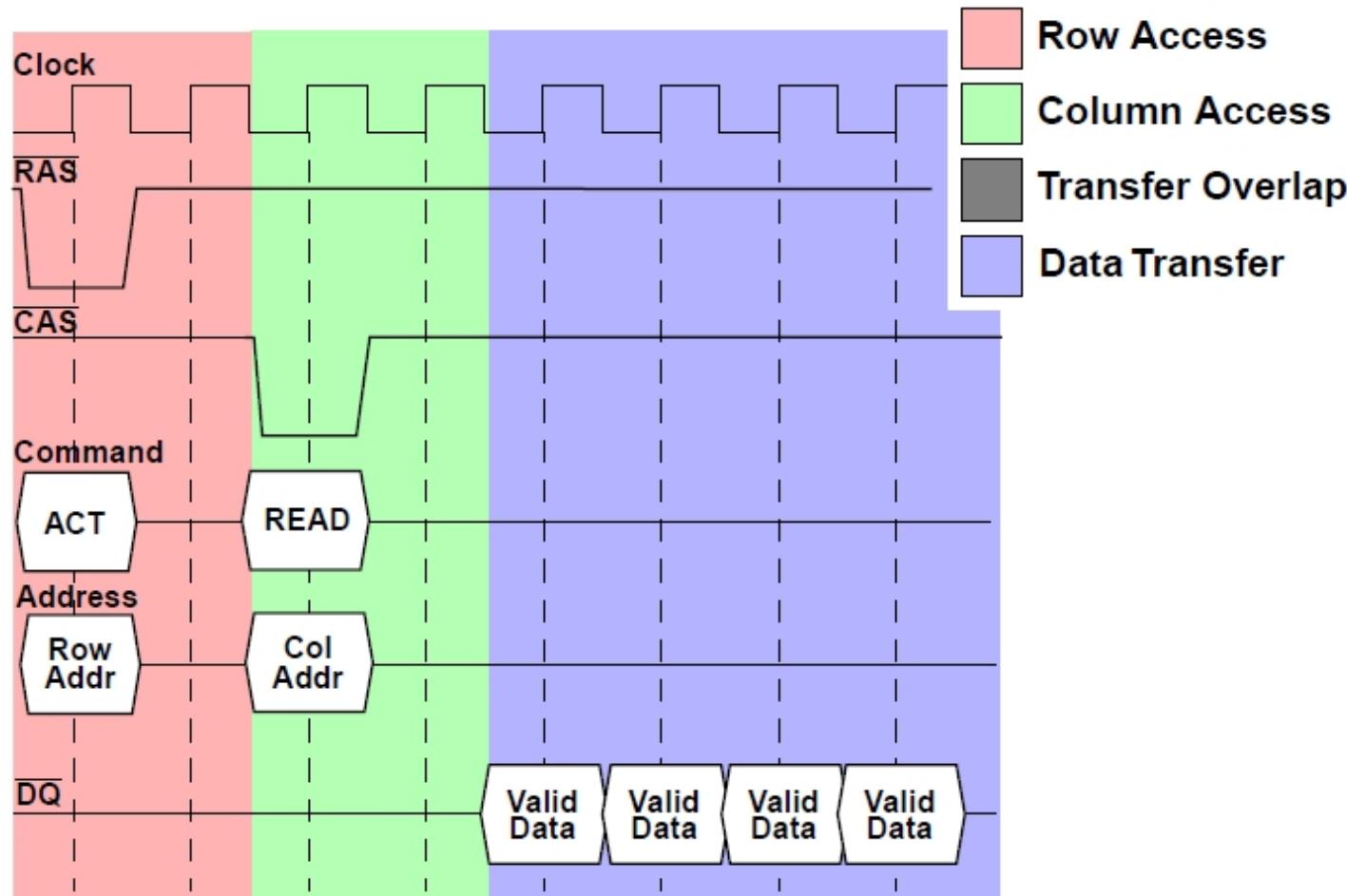
- Burst EDO ใช้เทคนิคเดียวกับ EDO แต่สำหรับข้อมูลใน address ที่ต่อเนื่องกัน สามารถใช้ Burst Mode โดยจะส่ง ข้อมูลในแอดдресที่ติดๆ กันออกมากได้ต่อเนื่องกัน





DRAM Evolution

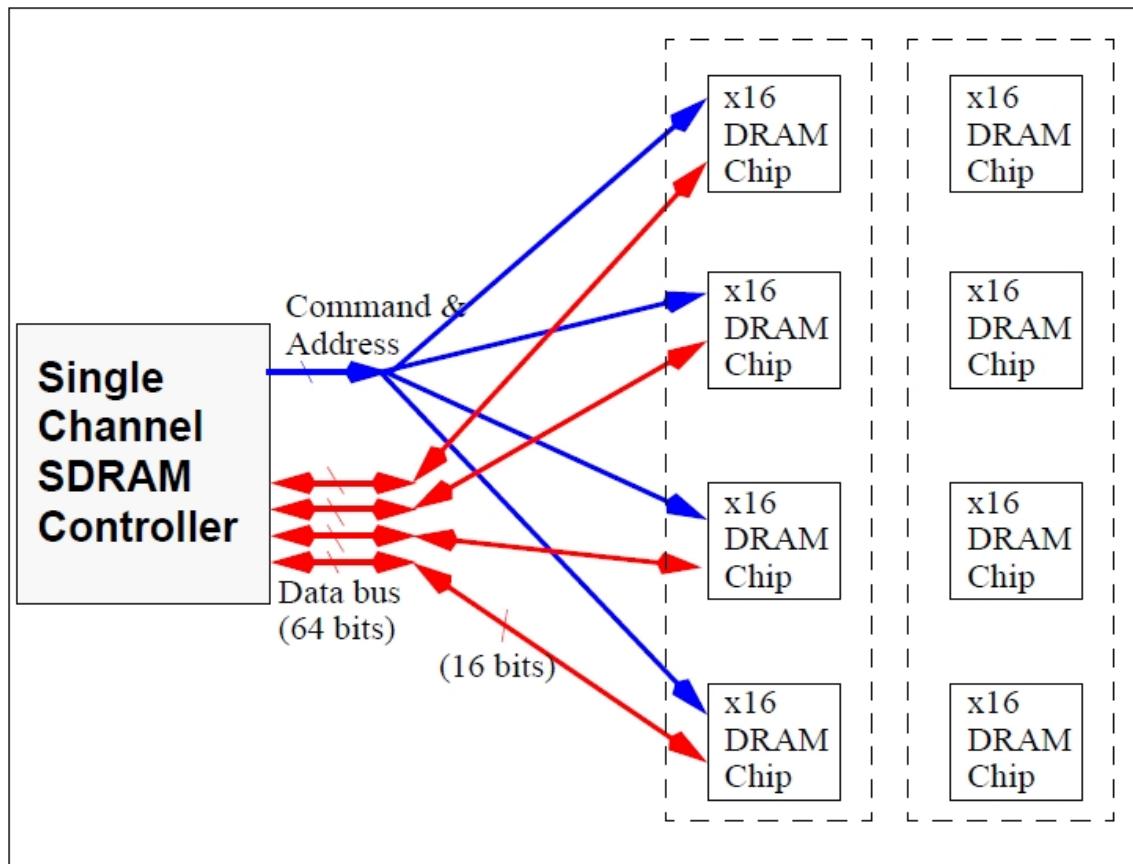
- Synchronous DRAM (SDRAM) เพิ่มสัญญาณ Clock เข้ามา กำกับการทำงาน ทำให้มีต้องใช้ วิธีการรอ เนื่องจาก DRAM จะให้ Output ออกมายield ที่กำหนด





SDRAM Topology

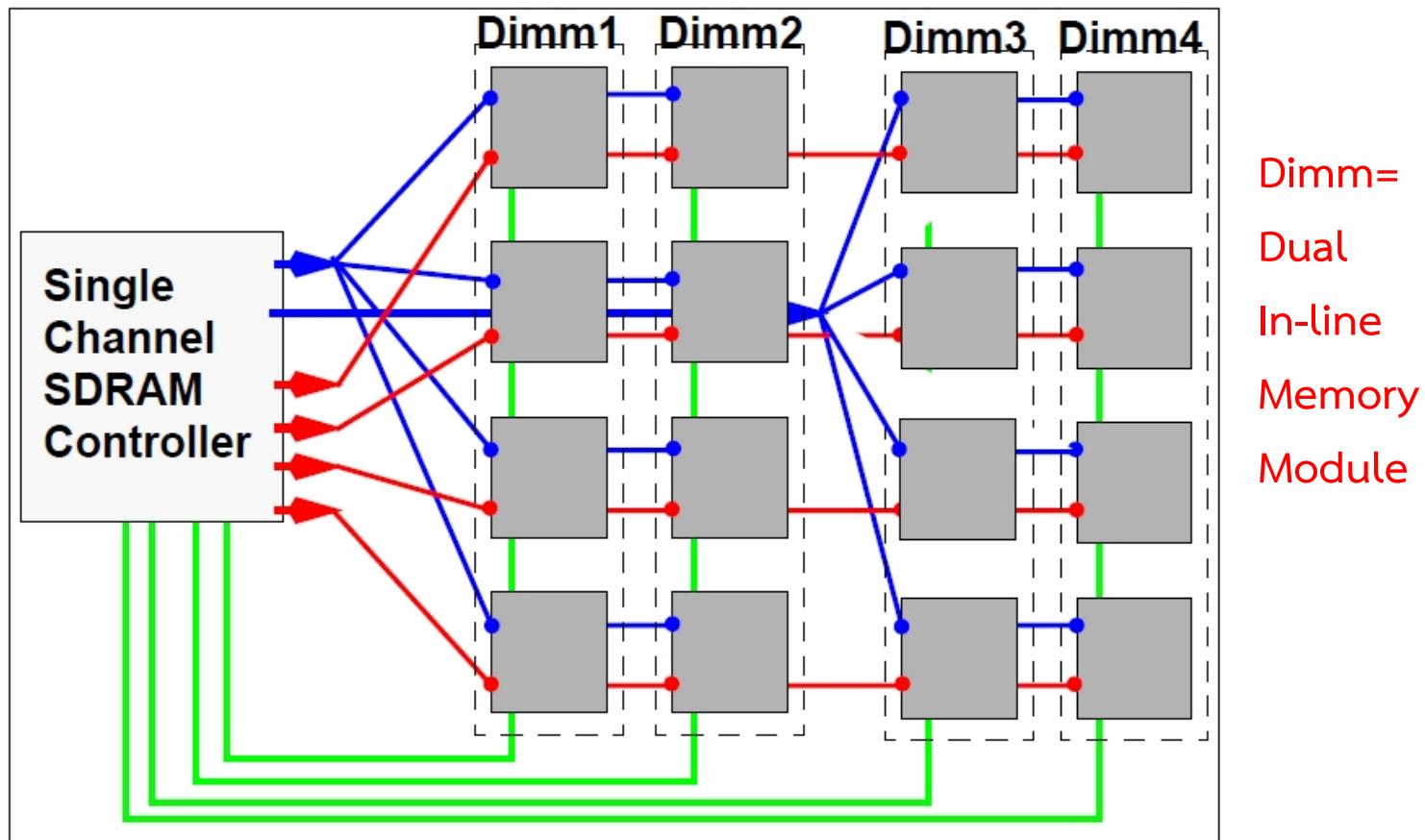
- RAM แต่ละແຜຈະເປັນ 64 ປີຕ ຄື່ອສັງຂໍ້ມູລໄດ້ສູງສຸດຄົງລະ 64 ປີຕ ໂດຍໃນການທຳກຳ ຈະສັງ ຄຳສັ່ງແລະແວດເດຣສໄປທຸກຊີປ ຈາກນັ້ນ Data ຈາກແຕ່ລະຊີປຈະມາຮັມກັນ 64 ປີຕ





SDRAM Topology

- SDRAM สามารถทำงานแบบ Dual Channel ได้ โดยจะทำงานครั้งละ 2 Dimm พร้อมกัน ทำให้สามารถได้ข้อมูลอุ่นมาได้ 2 Dimm ในเวลาใกล้เคียงกัน





SDRAM - Speed

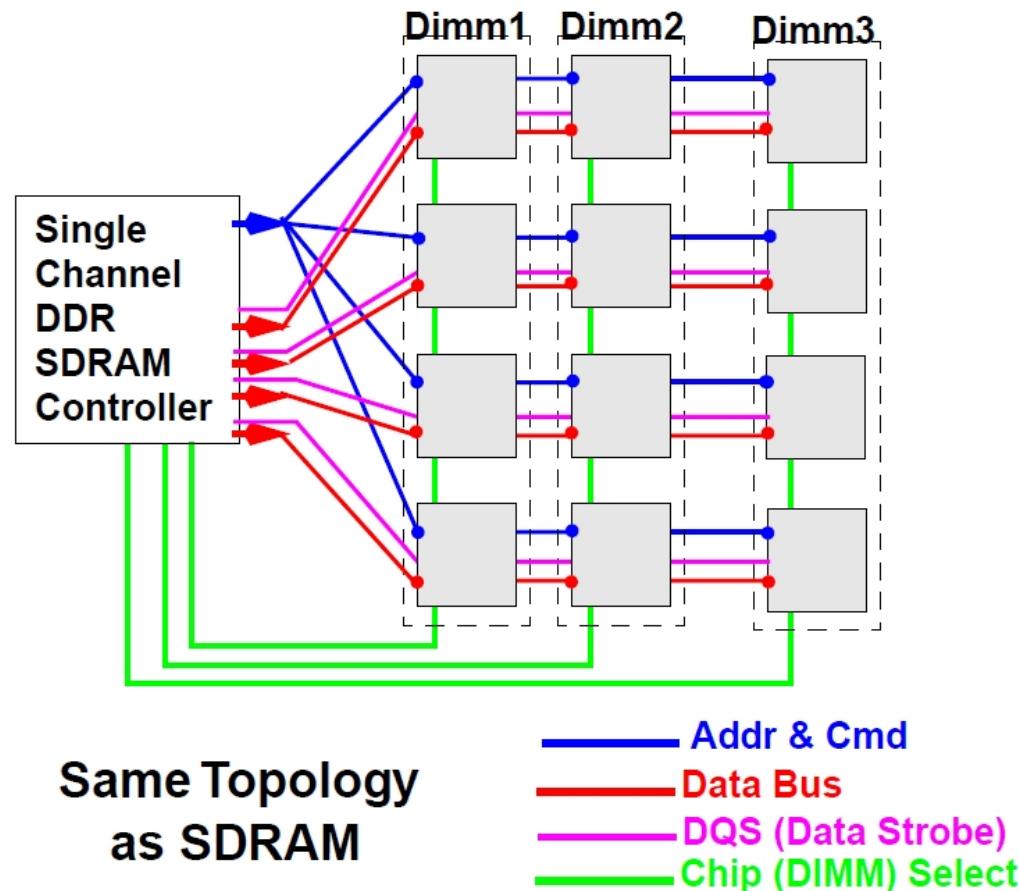
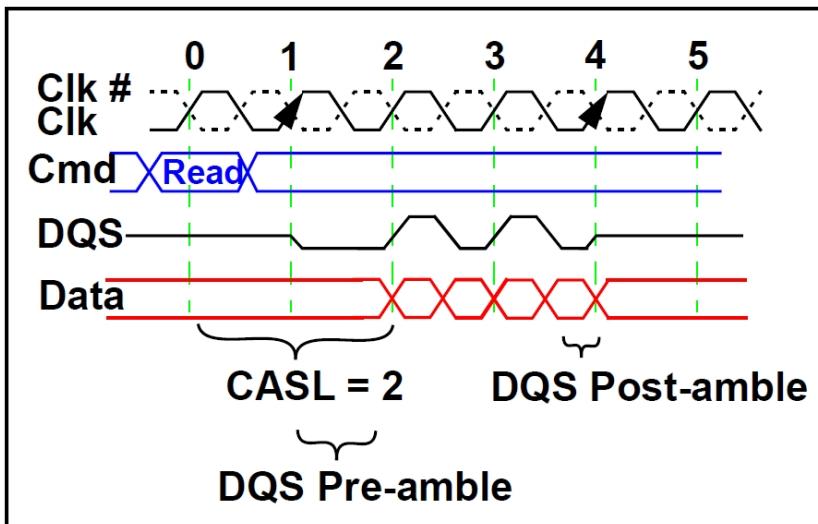
- Clock speed ของหน่วยความจำ จะน้อยกว่า Clock speed ของprocressor
- ตารางนี้เป็นความเร็วในการทำงานของ SDRAM จะเห็นว่าความเร็วสูงสุดในส่งข้อมูลประมาณ 1 GBps

Marketing Name	Chip Type	Clock Speed (MHz)	Cycles per Clock	Bus Speed (MTps)	Bus Width (Bytes)	Transfer Rate (MBps)
PC66	10ns	66	1	66	8	533
PC100	8ns	100	1	100	8	800
PC133	7ns	133	1	133	8	1,066

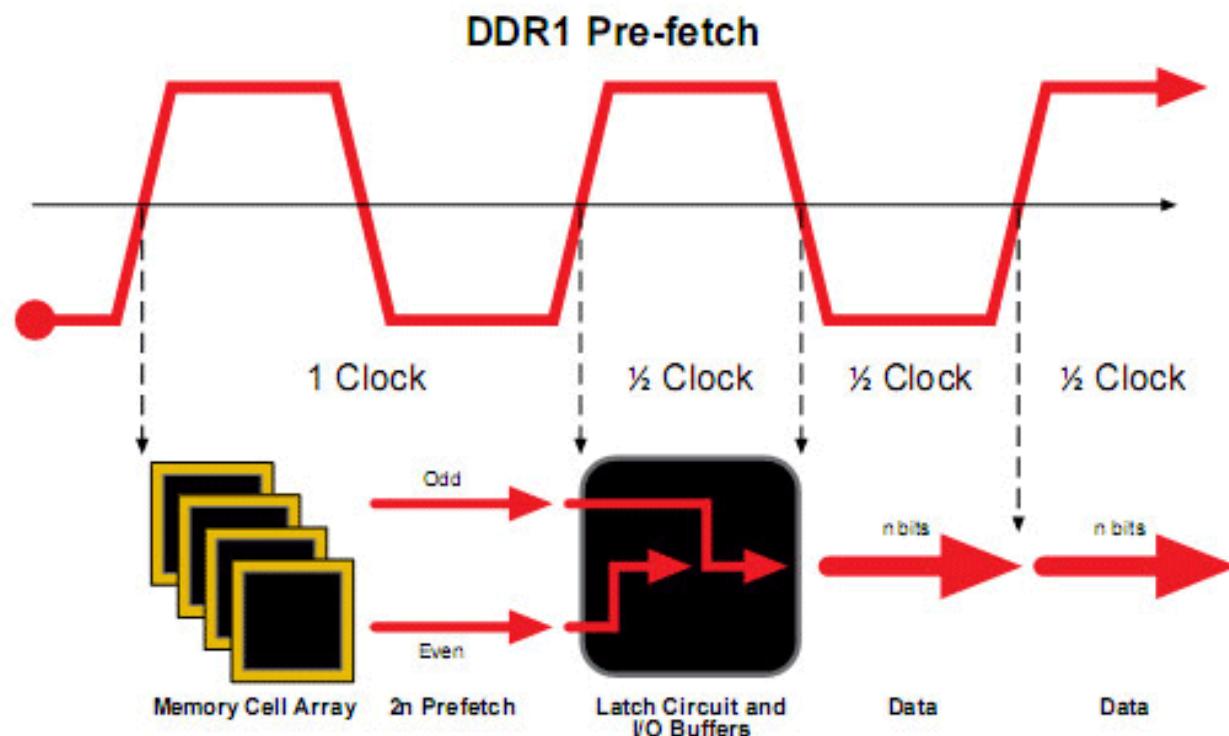


DDR SDRAM Topology

- DDR (Double Data Rate) จะเพิ่มสัญญาณ DQS ทำหน้าที่บอกให้ปล่อยจังหวะรับข้อมูล โดยจะทำงานทั้งขอบขึ้นและขอบลง



Prefetch



▲ Simplified 2n-Prefetch Architecture
Illustration: Ryan J. Leng



DDR SDRAM - Speed

- DDR

Marketing Name	Chip Type	Clock Speed (MHz)	Cycles per Clock	Bus Speed (MTps)	Bus Width (Bytes)	Transfer Rate (MBps)	Dual-Channel Rate (MBps)
PC1600	DDR200	100	2	200	8	1,600	3,200
PC2100	DDR226	133	2	266	8	2,133	4,266
PC2700	DDR333	166	2	333	8	2,667	5,333
PC3200	DDR400	200	2	400	8	3,200	6,400

- DDR2

Marketing Name	Chip Type	Clock Speed (MHz)	Cycles per Clock	Bus Speed (MTps)	Bus Width (Bytes)	Transfer Rate (MBps)	Dual-Channel Rate (MBps)
PC2-3200	DDR2-400	200	2	400	8	3,200	6,400
PC2-4200	DDR2-533	266	2	533	8	4,266	8,533
PC2-5300	DDR2-667	333	2	667	8	5,333	10,667
PC2-6400	DDR2-800	400	2	800	8	6,400	12,800
PC2-8500	DDR2-1066	533	2	1,066	8	8,533	17,066



DDR SDRAM - Speed

- DDR3

Marketing Name	Chip Type	Clock Speed (MHz)	Cycles per Clock	Bus Speed (MTps)	Bus Width (Bytes)	Transfer Rate (MBps)	Dual/Tri-Channel Rate (MBps)
PC3-6200	DDR3-800	400	2	800	8	6,400	12,800/19,200
PC3-8500	DDR3-1066	533	2	1,066	8	8,533	17,066/25,600
PC3-10600	DDR3-1333	667	2	1,333	8	10,667	21,333/32,000
PC3-12800	DDR3-1600	800	2	1,600	8	12,800	25,600/38,400

- DDR4

DDR SDRAM Standard	Internal rate (MHz)	Bus clock (MHz)	Prefetch	Data rate (MT/s)	Transfer rate (GB/s)	Voltage (V)
SDRAM	100-166	100-166	1n	100-166	0.8-1.3	3.3
DDR	133-200	133-200	2n	266-400	2.1-3.2	2.5/2.6
DDR2	133-200	266-400	4n	533-800	4.2-6.4	1.8
DDR3	133-200	533-800	8n	1066-1600	8.5-14.9	1.35/1.5
DDR4	133-200	1066-1600	8n	2133-3200	17-21.3	1.2



Cache

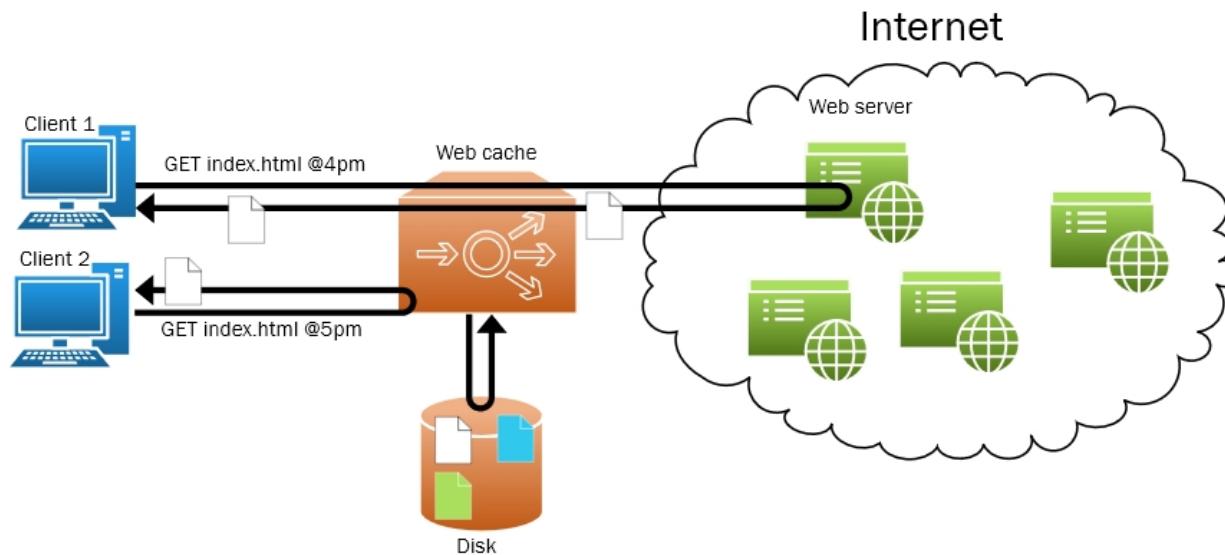
- ส่วนของ Instruction และ Data จะเก็บไว้ใน DRAM ซึ่งมีข้อดีที่มีความจุสูง แต่ก็มีข้อด้อย คือ ความเร็วต่ำ (high latency) โดยถ้าเทียบกับความเร็วของ CPU ทุกวันนี้อาจต้องใช้เวลารอถึง 150 clock cycles
- สมมติ CPU 3 GHz ดังนั้น 1 clock cycles = $1/3\text{GHz} = 0.33\text{ ns}$
ดังนั้น 50 ns = 150 clock cycles

Memory technology	Typical access time	\$ per GB in 2008
SRAM	0.5–2.5 ns	\$2000–\$5000
DRAM	50–70 ns	\$20–\$75
Magnetic disk	5,000,000–20,000,000 ns	\$0.20–\$2



Cache

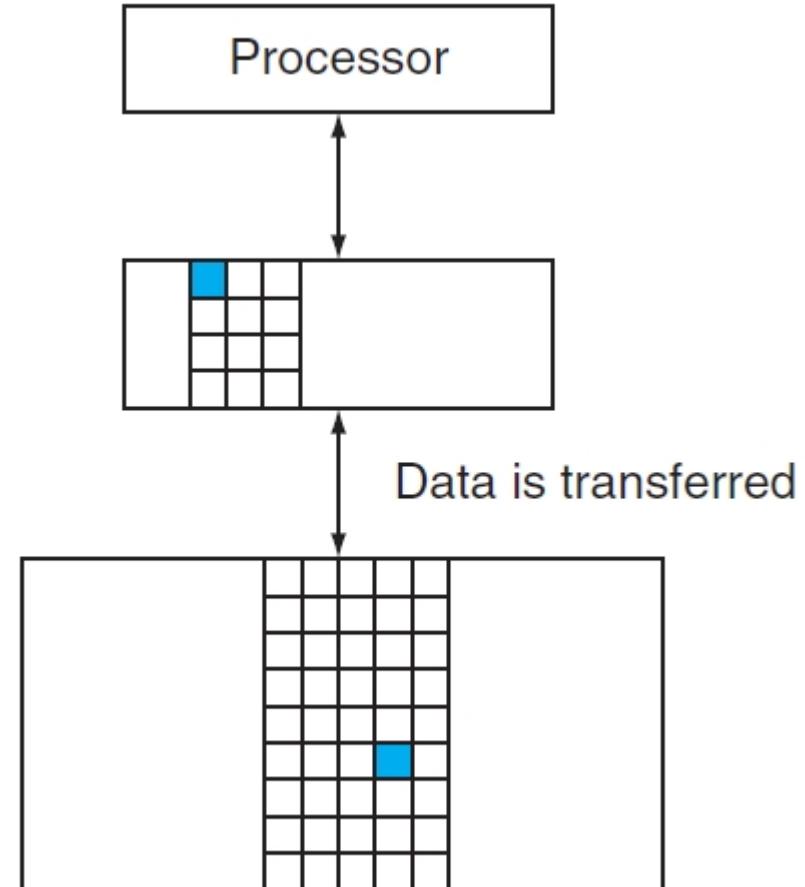
- ดังนั้นจึงได้มีการสร้างที่เก็บข้อมูลไว้ใน Processor ในโครงสร้างที่เรียกว่า Cache โดย Cache จะใช้เทคโนโลยี SRAM ซึ่งมีความเร็วสูงกว่า แต่มีความหนาแน่นน้อยกว่า และมีราคาแพงกว่ามาใช้
- ในเครือข่ายอินเทอร์เน็ตก็มีการนำ Cache มาใช้เช่นกันเพื่อความรวดเร็ว



Cache



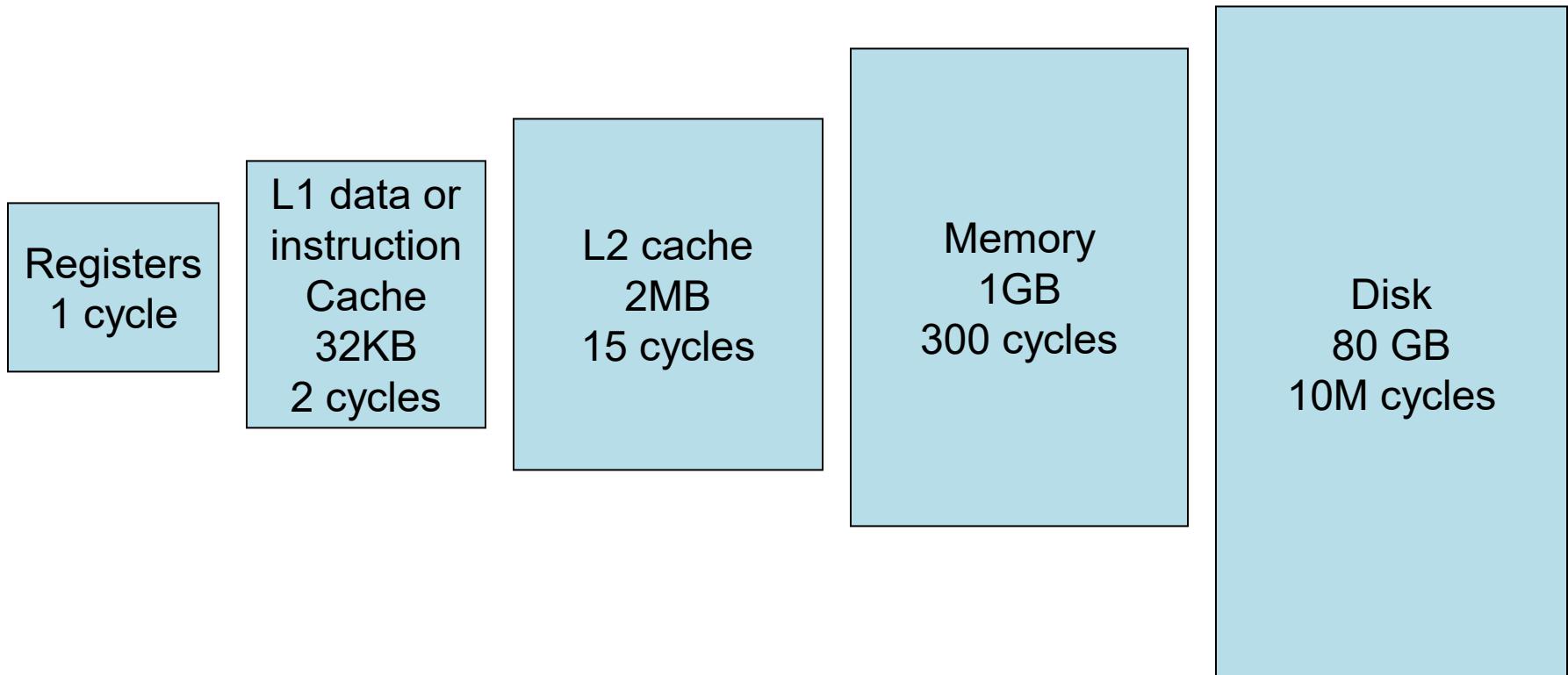
- Cache ทำงานอย่างไร
 - แคช เป็นเพียง copy ข้อมูล จาก แหล่งข้อมูลต้นฉบับ
 - เมื่อ CPU จะใช้ข้อมูลจะหาในแคชก่อน ถ้าไม่พบจะไปอ่านในหน่วยความจำ
 - แคช จะมีขนาดที่เล็กกว่าแหล่ง ข้อมูล ต้นฉบับ แต่มีความเร็วสูงกว่า
 - แคชอาจมีหลายระดับชั้น โดยแต่ละ ระดับจะเล็กลง แต่เร็วขึ้น
 - แคชจะมีประโยชน์ เมื่อมีการใช้ข้อมูลที่ ออยู่ในแคชซ้ำ





Memory Hierarchy

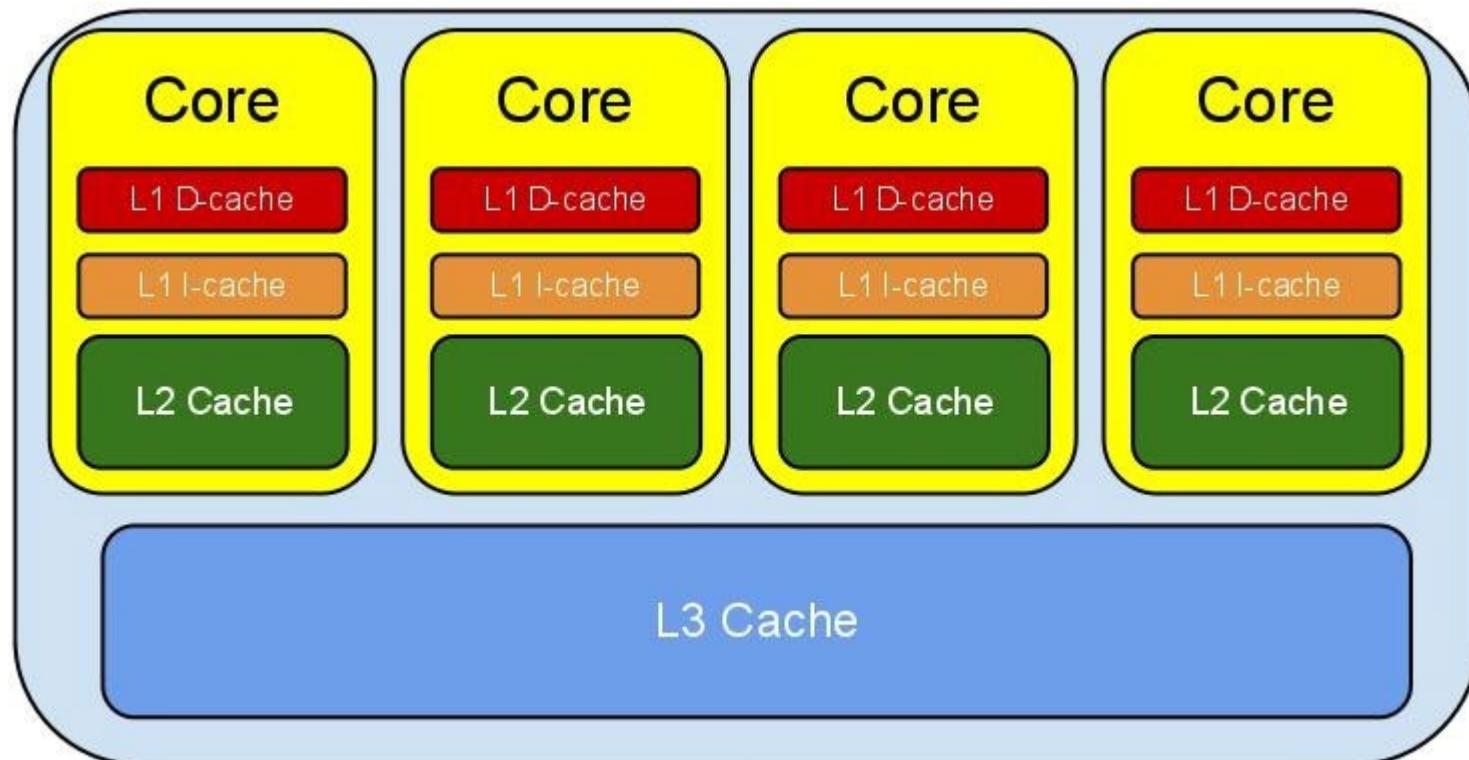
- ระบบคอมพิวเตอร์ในปัจจุบัน จะมีการจัดหน่วยความจำเป็นลำดับชั้น โดยยิ่งใกล้ Processor จะยิ่งเร็ว ยิ่งไกล Processor จะช้า แต่มีความจุมากขึ้น





Cache

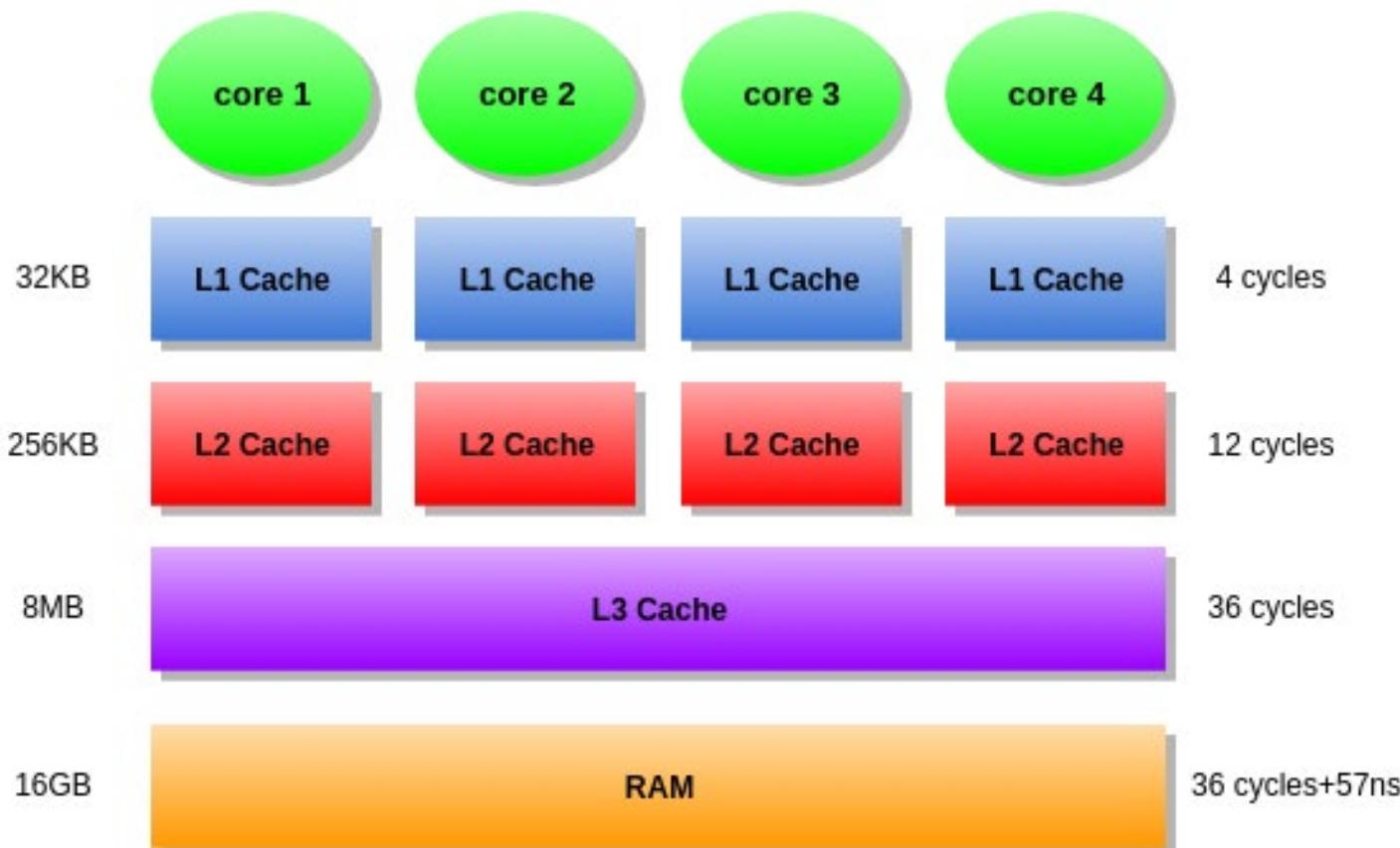
- CPU ในปัจจุบัน มักจะใช้ Cache 3 ระดับ คือ L1 D-Cache+L1 I-Cache, L2 Cache และ L3 Cache





Cache

- ความเร็วของ Cache ของ Core i7 เทียบกับ Clock Cycles





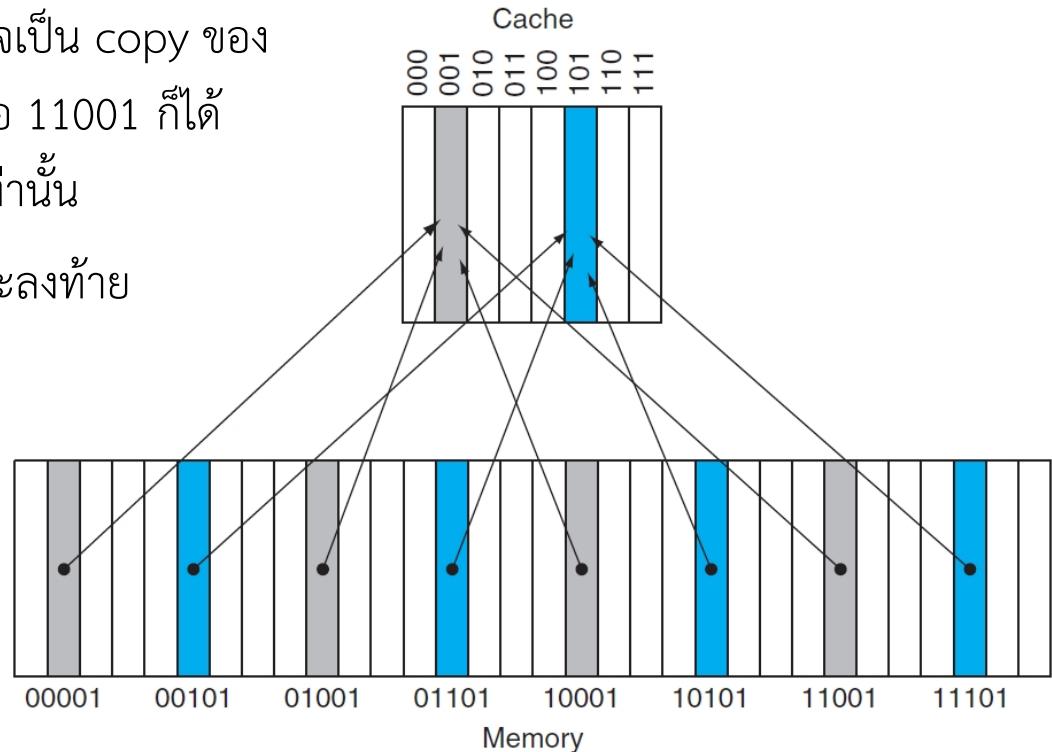
Locality

- Locality vs Cache
 - Cache จะมีประโยชน์ เมื่อข้อมูลมี Locality
 - Temporal locality หมายถึง การทำงานที่เมื่อมีการใช้ข้อมูลใด ก็อาจจะใช้ข้อมูลนั้นอีกในอนาคตอันใกล้ (ใช้ซ้ำ)
 - Spatial locality หมายถึง การทำงานที่เมื่อมีการใช้ข้อมูลใด ก็อาจจะใช้ข้อมูลที่อยู่ใกล้ๆ กันอีกในอนาคตอันใกล้ (ใช้ข้างเคียง)
- หากการทำงานพบข้อมูลใน Cache เรียกว่า **Hit** ถ้าไม่พบเรียกว่า **Miss**
- ถ้าไม่ใช้ Cache ค่า access time สำหรับข้อมูลอาจมีค่าสูงถึง 150 cc.
- แต่หากใช้ Cache ขนาด 32 KB ที่ทำงานได้ใน 2 cc. โดยสมมติ hit rate = 95% ค่า access time เนลี่ยจะเท่ากับ $0.95 \times 2 + 0.05 \times 150 = 9.4$ cc
- จากข้อกำหนดข้างต้น หาก hit rate = 90% ค่า access time เนลี่ย = ?



Accessing the Cache

- หน่วยความจำ Cache จะมี Address โดยเริ่มจาก 0
- หน่วยความจำ Cache จะมีขนาดเล็กกว่าหน่วยความจำหลัก ดังนั้นใน 1 ตำแหน่งจะเป็น copy ของข้อมูลได้หลายตำแหน่ง (mapping)
- จากรูปตำแหน่ง 001 ของ Cache อาจเป็น copy ของ 00001 หรือ 01001 หรือ 10001 หรือ 11001 ก็ได้ แต่ต้องเป็น copy ของอันใดอันหนึ่งเท่านั้น
- จะสังเกตว่า address ที่เป็นต้นฉบับจะลงท้ายด้วย 001
- ต้นฉบับมี 32 ตำแหน่ง Cache มี 8 ตำแหน่ง





Accessing the Cache

- การกำหนดตำแหน่ง cache จะใช้วิธี mod ได้เท่าไร จะใช้ตำแหน่งนั้น
- บรรทัด 1 แอดเดรส 22 (10110) อ้างถึง cache ตำแหน่ง 110 เนื่องจากอ้างเป็นครั้งแรก จึงเป็น miss
- บรรทัด 3 แอดเดรส 22 (10110) อ้างถึง cache ตำแหน่ง 110 ครั้งที่ 2 จึงเป็น hit
- แต่ถ้ามี แอดเดรส 11110 จะอ้างถึง cache ตำแหน่ง 110 เช่นกัน แต่เป็นค่า address (เป็น miss)

Decimal address of reference	Binary address of reference	Hit or miss in cache	Assigned cache block (where found or placed)
22	10110_{two}	miss (5.6b)	$(10110_{\text{two}} \bmod 8) = 110_{\text{two}}$
26	11010_{two}	miss (5.6c)	$(11010_{\text{two}} \bmod 8) = 010_{\text{two}}$
22	10110_{two}	hit	$(10110_{\text{two}} \bmod 8) = 110_{\text{two}}$
26	11010_{two}	hit	$(11010_{\text{two}} \bmod 8) = 010_{\text{two}}$
16	10000_{two}	miss (5.6d)	$(10000_{\text{two}} \bmod 8) = 000_{\text{two}}$
3	00011_{two}	miss (5.6e)	$(00011_{\text{two}} \bmod 8) = 011_{\text{two}}$
16	10000_{two}	hit	$(10000_{\text{two}} \bmod 8) = 000_{\text{two}}$
18	10010_{two}	miss (5.6f)	$(10010_{\text{two}} \bmod 8) = 010_{\text{two}}$
16	10000_{two}	hit	$(10000_{\text{two}} \bmod 8) = 000_{\text{two}}$



Accessing the Cache

- เพื่อตรวจสอบ Address ของ Memory จึงต้องเพิ่มส่วนที่เรียกว่า tag เข้าไปในโครงสร้าง cache โดย กำหนดที่ตรวจสอบว่าข้อมูลที่อยู่ใน Cache เป็นข้อมูลของ address ที่ถูกต้องหรือไม่
- ขนาดของ tag คำนวณได้จาก $a-n$ โดย 2^n คือจำนวน block ของแคช และ a คือขนาดของ address ดังนั้นกรณีนี้ tag มีขนาด 2 บิต ($2^3=8$) ; tag = 5-3 = 2
- บิต V กำหนดที่บอกว่า cache ตำแหน่งนั้นถูกใช้หรือไม่
- สมมติลำดับการใช้ 22, 26, 16, 3, 18

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

a. The initial state of the cache after power-on

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	10 _{two}	Memory (10110 _{two})
111	N		

b. After handling a miss of address (10110_{two})

Accessing the Cache



Index	V	Tag	Data
000	N		
001	N		
010	Y	11 _{two}	Memory (11010 _{two})
011	N		
100	N		
101	N		
110	Y	10 _{two}	Memory (10110 _{two})
111	N		

c. After handling a miss of address (11010_{two})

Index	V	Tag	Data
000	Y	10 _{two}	Memory (10000 _{two})
001	N		
010	Y	11 _{two}	Memory (11010 _{two})
011	Y	00 _{two}	Memory (00011 _{two})
100	N		
101	N		
110	Y	10 _{two}	Memory (10110 _{two})
111	N		

e. After handling a miss of address (00011_{two})

Index	V	Tag	Data
000	Y	10 _{two}	Memory (10000 _{two})
001	N		
010	Y	11 _{two}	Memory (11010 _{two})
011	N		
100	N		
101	N		
110	Y	10 _{two}	Memory (10110 _{two})
111	N		

d. After handling a miss of address (10000_{two})

Index	V	Tag	Data
000	Y	10 _{two}	Memory (10000 _{two})
001	N		
010	Y	10 _{two}	Memory (10010 _{two})
011	Y	00 _{two}	Memory (00011 _{two})
100	N		
101	N		
110	Y	10 _{two}	Memory (10110 _{two})
111	N		

f. After handling a miss of address (10010_{two})



Exercise

- จากตาราง กำหนดการใช้งานเป็น 4,7,10,13,16,28,23,18,13,28,4,7,10 ให้แสดงการเปลี่ยนแปลงใน Cache กำหนดให้ address ขนาด 5 บิต

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

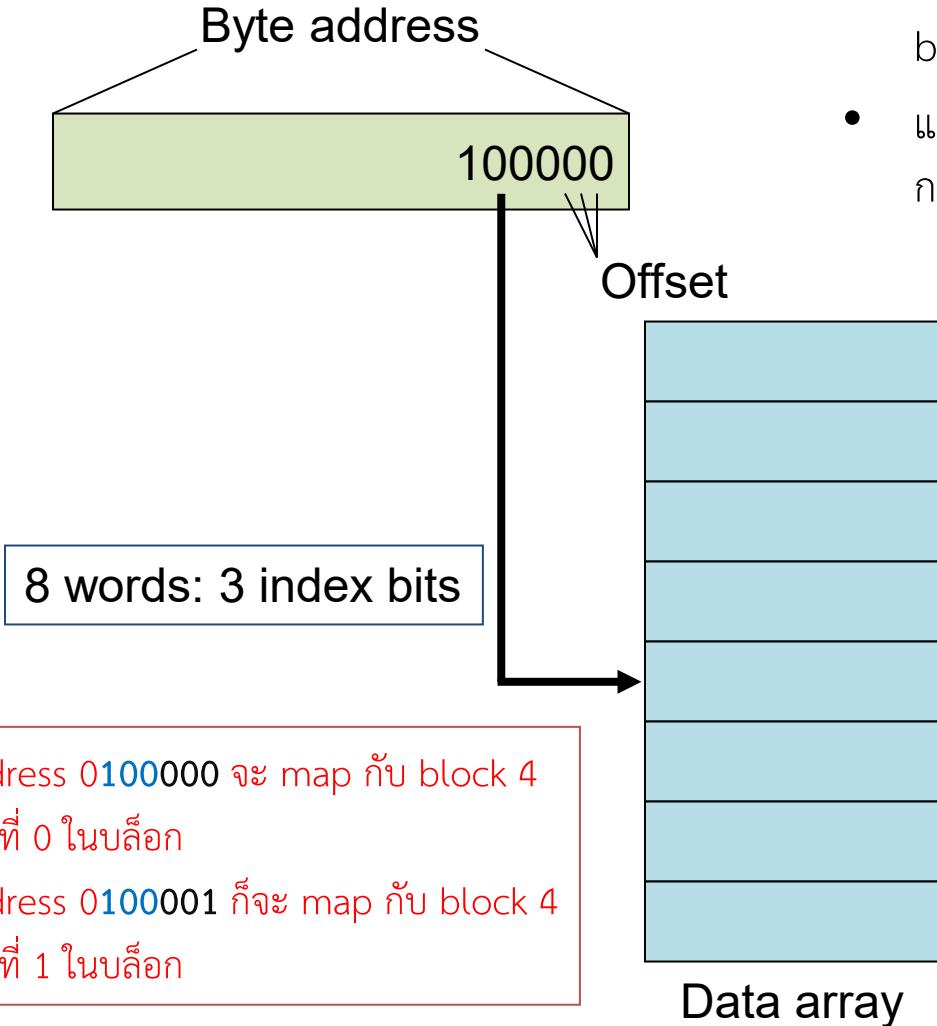
- The initial state of the cache after power-on



Accessing the Cache

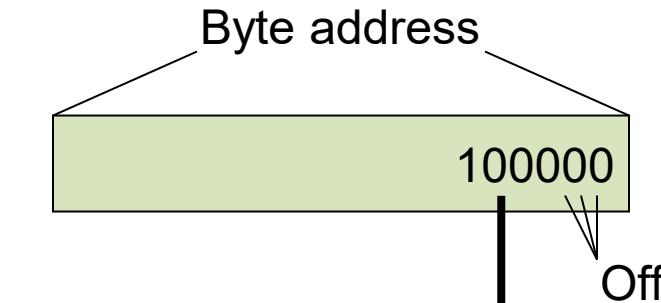
- การ miss กรณีที่ไม่มีข้อมูลมาก่อนเรียกว่า **Compulsory miss**
- การ miss กรณีที่ address ไม่ตรง (รูป f) จะเรียกว่า **Conflict miss**
- วิธีการทำงานที่ผ่านมา เรียกวิธีการนี้ว่า Direct-Mapped เนื่องจากเป็นการ Map ระหว่างตำแหน่งในหน่วยความจำกับตำแหน่งของแคชโดยตรง
- แต่ตัวอย่างที่ผ่านมา ขนาดของ Cache มีขนาด 1 ไบต์ทำให้ได้ประโยชน์แต่เพียง **Temporal locality** อย่างเดียว
- ดังนั้นหากกำหนดขนาดของ Cache ให้มากกว่า 1 ไบต์ ก็จะได้ประโยชน์ในเรื่อง **Spatial locality** ด้วย
- ตัวอย่างในหน้าถัดไป แต่ละ entry ของ cache จะมีขนาด 8 ไบต์ โดยจะเรียกแต่ละ entry ว่า **line** หรือ **block**
- การ map ระหว่าง address กับ cache จะแบ่งเป็น 2 ส่วน คือ offset ใช้ในการระบุตำแหน่งใน block และ index ใช้ในการระบุ block

Accessing the Cache



- Offset จะซึ่งตำแหน่งใน block จากรูปแคชแต่ละ block จะมี 8 ไบต์ ดังนั้นจึงใช้ offset จำนวน 3 บิต
- แคชมี 8 block ดังนั้นจะใช้ 3 บิตเพื่อเป็น index ใน การ map (ในรูป 100 คือ entry ที่ 4)
- จะเห็นว่ากรณีจะได้ประโยชน์ในเรื่อง Spatial Locality ด้วย เพราะการโหลดจาก หน่วยความจำมาที่แคชจะโหลดเป็นบล็อก

Accessing the Cache

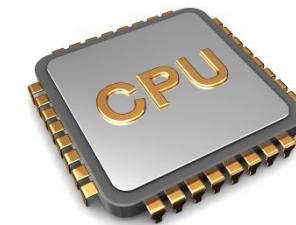


Address 0100000 จะ map กับ block 4
Address 0100001 ก็จะ map กับ block 4
Address 1100000 ก็จะ map กับ block 4



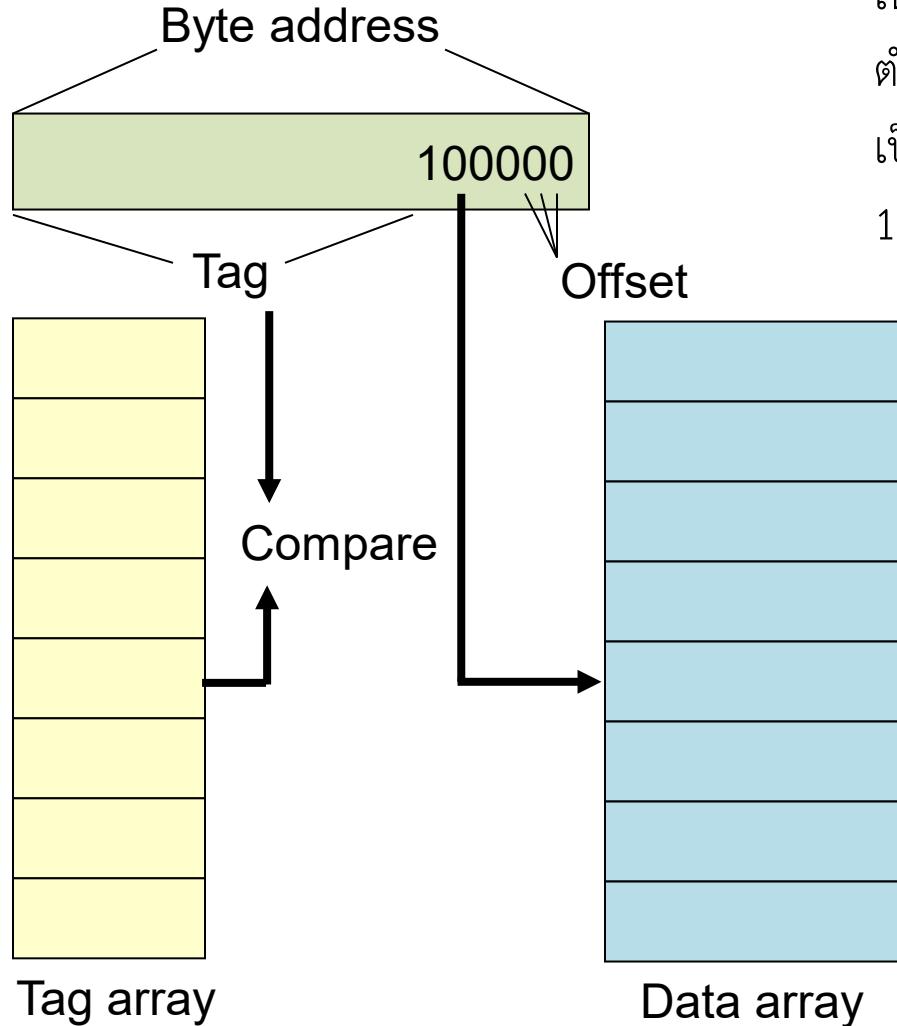
Data array

- แคชแบบ Direct-Mapped แต่ละ Address ใน memory จะ map กับตำแหน่งใน cache เพียงตำแหน่งเดียว
- แคชแต่ละตำแหน่งอาจถูก map กับหลาย address (ตามตัวอย่าง) จึงไม่รู้ว่าข้อมูลใน cache เป็นของ address ใดกันแน่



8-byte words

The Tag Array



- เนื่องจากมีหลาย address ที่ map ลงใน cache ตำแหน่งเดียวกัน ทำให้มีรูปว่าในตำแหน่งที่ 4 เป็นข้อมูลของ address 0100000 หรือ 1100000 กันแน่ จึงต้องเพิ่ม tag เพื่อตรวจสอบ
- Tag คือ ข้อมูลส่วนที่เหลือของ address เมื่อหัก index กับ offset แล้ว ทำหน้าที่ตรวจสอบว่าเป็น Address ที่ถูกต้อง หรือไม่



Accessing the Cache

- จากตัวอย่าง แม้จะมีการอ้างแอดเดรส 100000_2 แต่เนื่องจาก cache มีขนาด 8 ไบต์ ดังนั้นในการโหลดข้อมูลจากหน่วยความจำจะโหลดมาทั้ง block คือ $100000-100007$
- ดังนั้นหากในครั้งต่อไป มีการอ่านที่ตำแหน่ง 100001 ก็ยังคงเกิด hit เนื่องจาก Spatial locality สามารถอ่านจาก cache ได้เลย
- ขนาดของ block = 2^m เมื่อ m คือจำนวนบิต offset หรือ $m = \log_2 \text{blocksize}$
- ดังนั้นขนาดของ tag คำนวณได้จาก $a-(n+m)$ โดย 2^n คือจำนวน block ของแคช
- จากตัวอย่างข้างต้น สมมติว่า address มีขนาด 16 บิต blocksize มีขนาด 8 ไบต์ ดังนั้น $m = 3$ และจำนวน block เท่ากับ 8 ดังนั้น $n = 3$ ดังนั้นขนาดของ tag คือ $16-(3+3) = 10$ บิต
- กำหนดให้ address ขนาด 32 บิต มี cache จำนวนหนึ่ง โดย block ขนาด 32 ไบต์ จำนวน 256 block ให้หาขนาดของ tag และขนาดของ cache

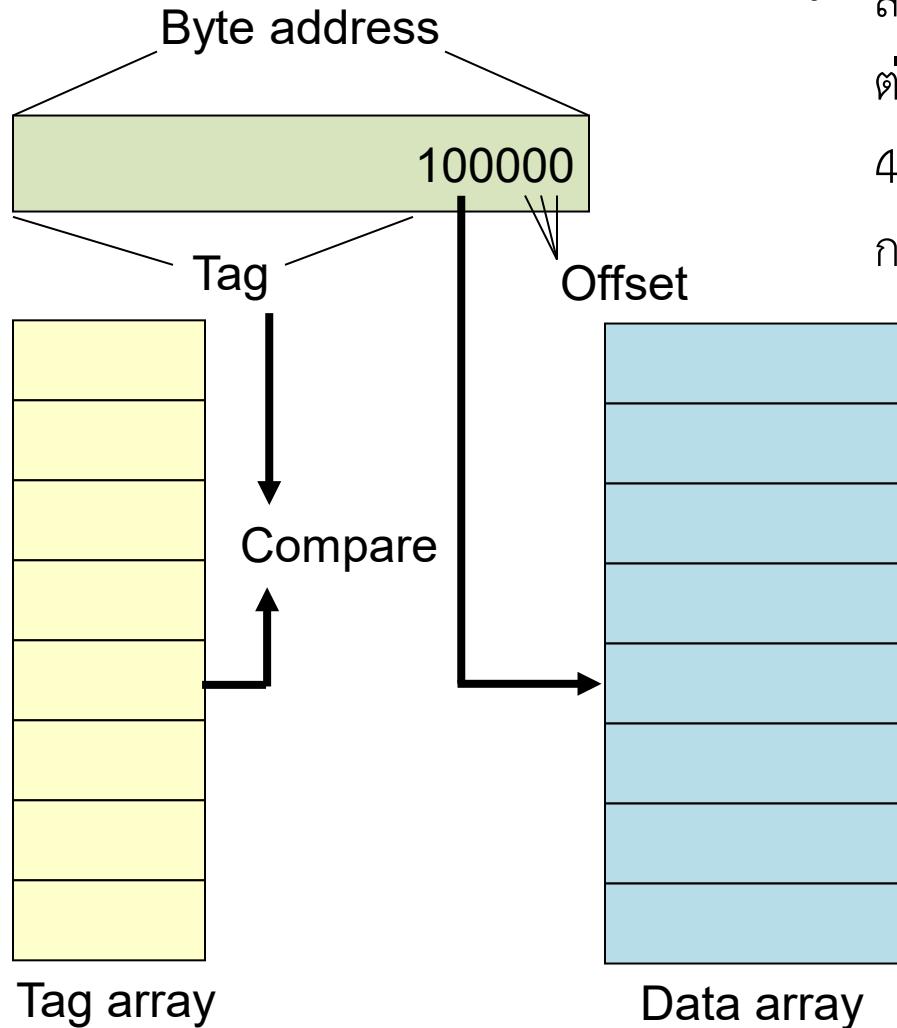
Exercise



- กำหนด Address ขนาด 32 บิต แต่ละบล็อกมีขนาด 32 ไบต์ โดยแคชมีขนาด 64 KB
 - จงหาขนาดของ offset, index และ tag
 - ข้อมูล address : 0000 1234, 1000 5555, F000 ABCD อัญญาบล็อกได และ offset ได



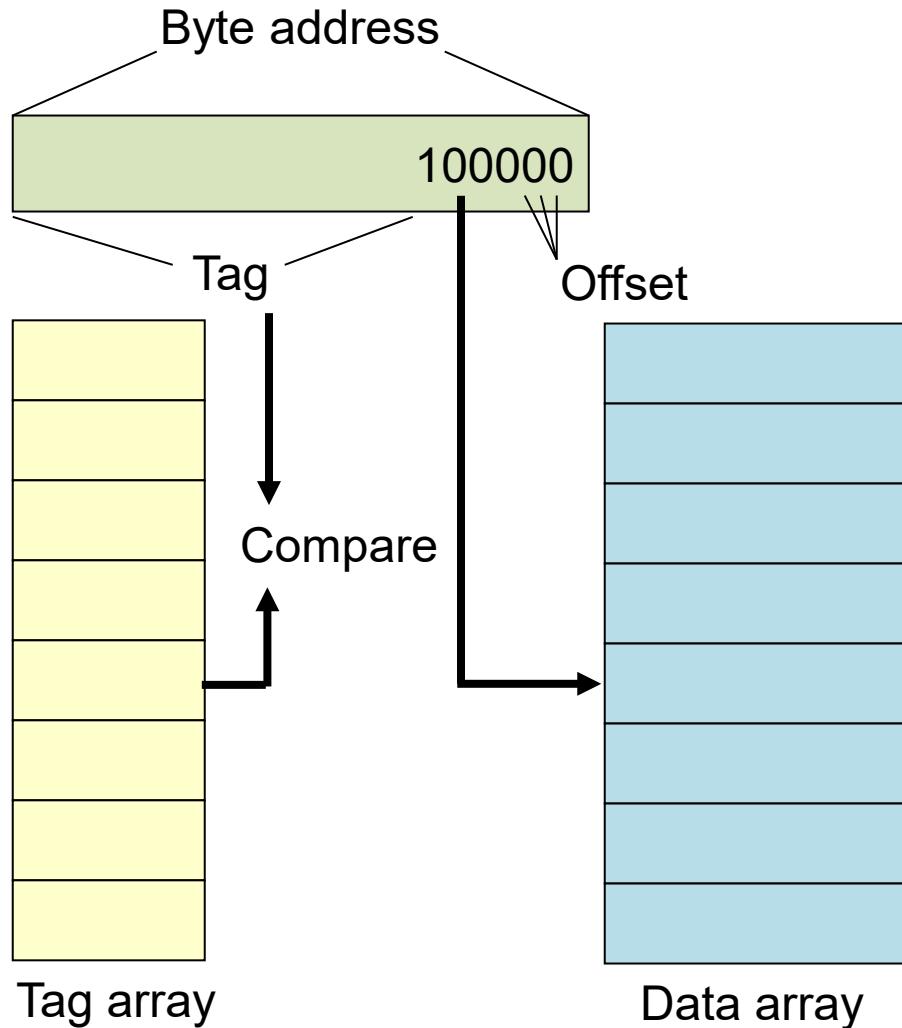
Example Access Pattern



- สมมติว่ามีการเรียกใช้ข้อมูลใน address ต่อไปนี้
4,7,10,13,16,68,73,78,83,88,4,7,10
การเรียกใช้ครั้งใดที่ hit และครั้งใดที่ miss



Example Access Pattern



Addr.	ฐานสอง	Index	Tag	Miss /Hit
4	00 000 100	000	00	Miss
7	00 000 111	000	00	Hit
10	00 001 010	001	00	Miss
13	00 001 101	001	00	Hit
16	00 010 000	010	00	Miss
68	01 000 100	000	01	Miss
73	01 001 001	001	01	Miss
78	01 001 110	001	01	Hit
83	01 010 011	010	01	Miss
88	01 011 000	011	01	Miss
4	00 000 100	000	00	Miss
10	00 001 010	001	00	Miss

Exercise



- กำหนด Address ขนาด 16 บิต แต่ละบล็อกมีขนาด 32 ไบต์ โดยแคชมีขนาด 32 บล็อก
 - จงหาขนาดของ offset, index และ tag
 - สมมติว่ามีการเรียกใช้ข้อมูลใน address ต่อไปนี้
0, 4, 16, 132, 232, 160, 1024, 30, 140, 3100, 180, 2180
การเรียกใช้ครั้งใดที่ hit และครั้งใดที่ miss



Exercise

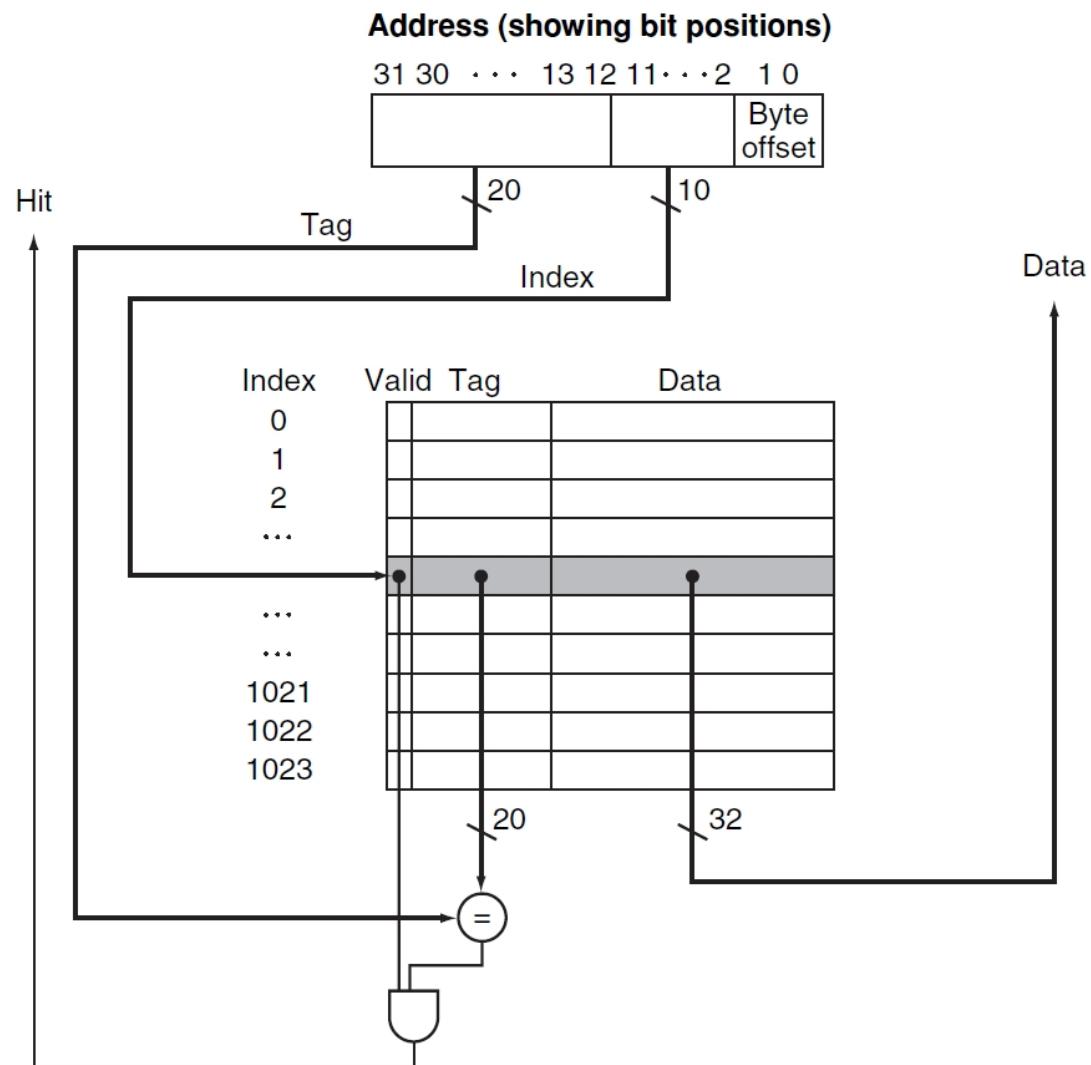
- บล็อกละ 32 ไบต์ ดังนั้น offset = 5 บิต ($2^5=32$) และมี 32 บล็อก ดังนั้น index = 5 บิต
- ดังนั้น tag = $16-5-5 = 6$ บิต

Addr.	ฐานสอง	Index	Tag	Miss/Hit
0	0000 0000 0000	00000	00	Miss
4	0000 0000 0100	00000	00	Hit
16	0000 0001 0000	0000	00	Hit
132	0000 1000 0100	00100	00	Miss
232	0000 1110 1000	00111	00	Miss
160	0000 1010 0000	00101	00	Miss
1024	0100 0000 0010	00000	01	Miss
30	0000 0001 1110	00000	00	Miss
140	0000 1000 1110	00100	00	Hit
3100	1100 0001 1100	00000	11	Miss
180	0000 1011 0100	00101	00	Hit
2180	1000 1000 0100	00100	10	Miss



Hardware of Direct-Mapped

- จักรูป block size ขนาด 4 ไบต์
ดังนั้น offset = 2 บิต
- จำนวน block = 1024
ดังนั้น index = 10 บิต
- Tag = $32 - (n+m)$
 $= 32 - (10+2)$
 $= 20$
- ขนาดของ cache
 $32 \times 1024 = 4\text{KB}$
- ขนาดของ cache ทั้งหมด
 $(32 + 20 + 1) * 1024$
 $= 147 \text{ Kbit}$

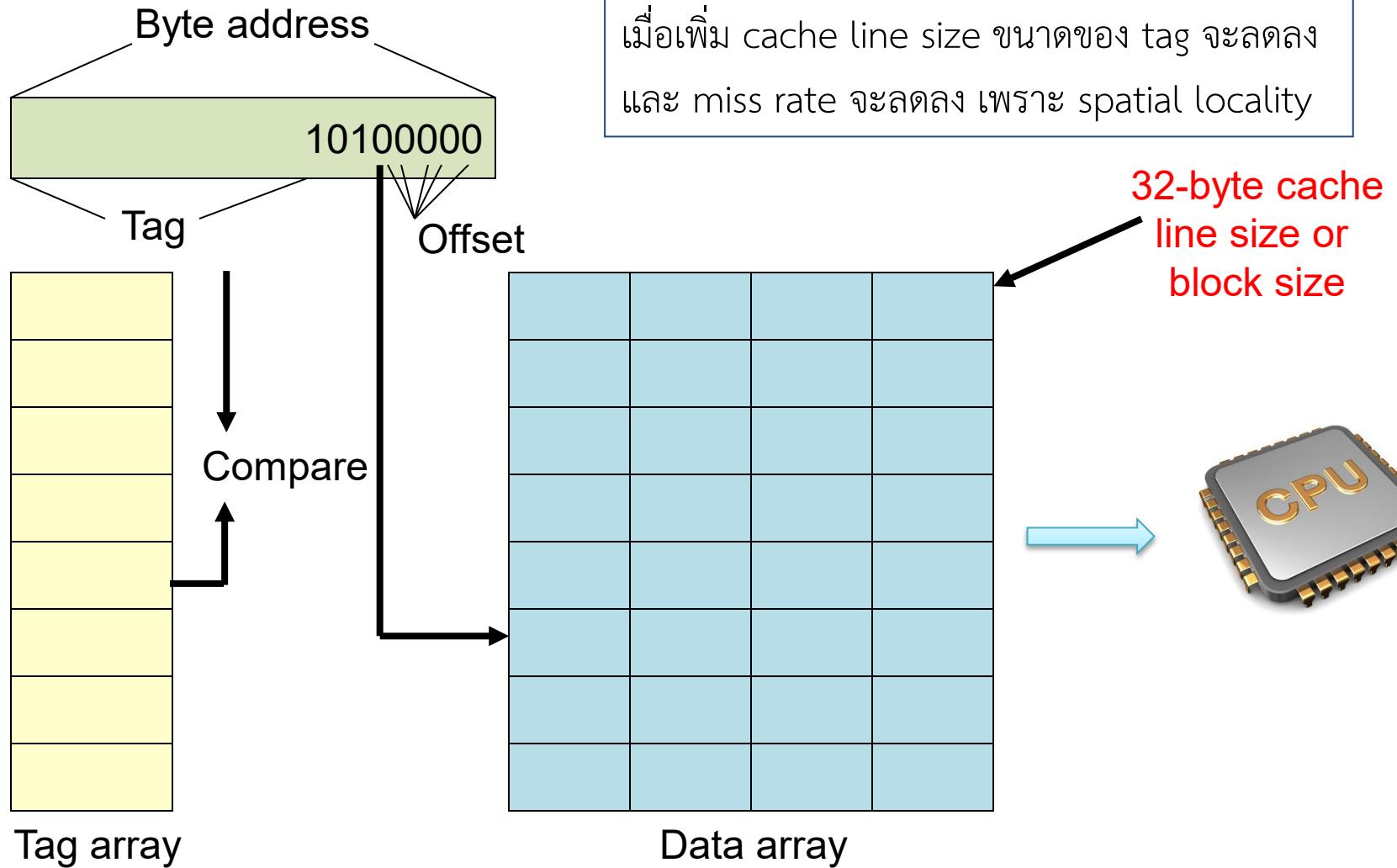




Address

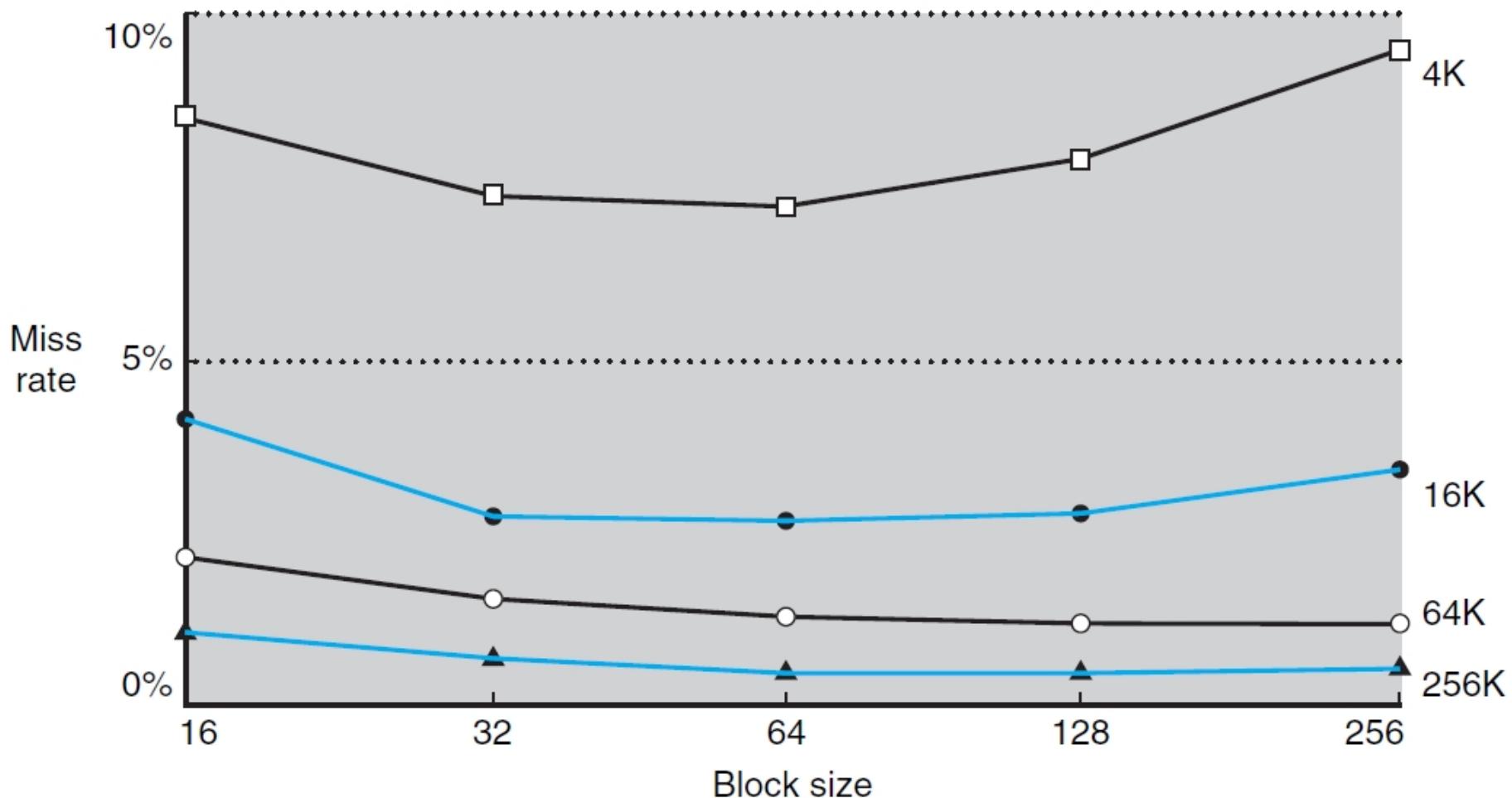
- Block address คือ ตำแหน่งของ block ที่ข้อมูลที่ต้องการอยู่
= (Byte Address / Byte per block) **mod** No. of blocks
- เช่น กำหนดให้ แต่ละ block มี 8 ไบต์ ถ้าเป็น address ที่ 9 block address = $9/8 = 1$ คือ อยู่ block ที่ 1
- โดย offset ใน block = Byte Address **mod** Byte per block
- คำถาม : กำหนดให้มี 16 ไบต์ต่อบล็อก ตำแหน่งที่ 1200 อยู่ใน block address ใด

Increasing Line Size

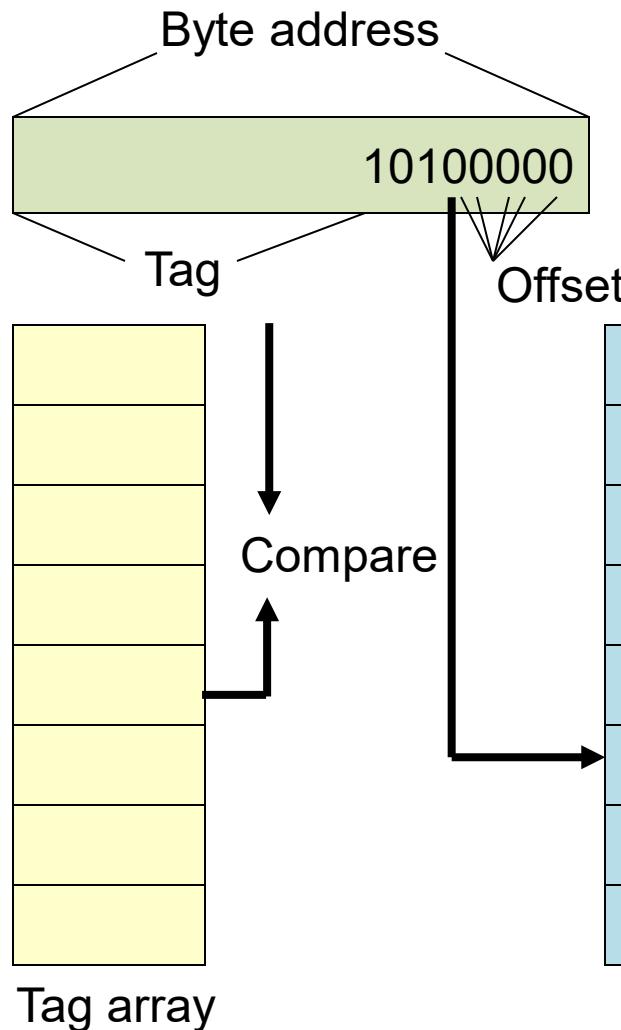




Miss Rate vs Block size



Exercise



สมมติว่ามีการเรียกใช้ข้อมูลใน address ต่อไปนี้

4,7,10,13,16,68,73,78,83,88,4,7,10

การเรียกใช้ครั้งใดที่ hit และครั้งใดที่ miss

32-byte cache
line size or
block size

Data array

Exercise



Addr.	ฐานสอง	Index	Tag	Miss /Hit
4	00 000 100	000	00	Miss
7	00 000 111	000	00	Hit
10	00 001 010	000	00	Hit
13	00 001 101	000	00	Hit
16	00 010 000	000	00	Hit
68	01 000 100	010	00	Miss
73	01 001 001	010	00	Hit
78	01 001 110	010	00	Hit
83	01 010 011	010	00	Hit
88	01 011 000	010	00	Hit
4	00 000 100	000	00	Hit
10	00 001 010	000	00	Hit



Cache Size

- นอกเหนือจากขนาดของ Cache โดยตรงแล้ว Tag เองก็ถือว่าเป็นส่วนหนึ่งของ Cache ด้วย เช่น
 - แคชขนาด 16 KB แต่ละ block มีขนาด 16 ไบต์
 - แปลว่ามีจำนวน 1024 บล็อก
 - สมมติว่า address ขนาด 32 บิต ดังนั้น tag ก็จะมีขนาด $32 - (10 + 4) + 1$ (รวม valid bit)
 - ดังนั้นแคชจะมีขนาด $= (16 \times 8 + 19) * 1024 = 147 \text{ Kbits}$



Cache Misses

- Cache Miss คือ การไม่พบข้อมูลใน Cache จะแบ่งออกเป็น 2 สถานการณ์
 - Write Miss คือ การจะนำข้อมูลไปเก็บ แต่ตำแหน่งของข้อมูลนั้นไม่อยู่ใน Cache ในกรณีนี้มีทางเลือก 2 ทาง
 - Write-allocate เป็นการนำข้อมูลจาก memory มาไว้ที่ cache ด้วย
 - Write-no-allocate เป็นการเขียนข้อมูลลงในหน่วยความจำโดยตรงไม่ผ่าน cache
 - Read Miss คือ การอ่านข้อมูล แต่ไม่พบข้อมูลใน cache ในกรณีจะนำข้อมูลในหน่วยความจำมาบรรจุใน cache เสมอ โดยกรณีที่ cache มี way ที่ว่างก็จะนำมาใส่ใน way ที่ว่าง แต่หากไม่ว่าง ก็จะต้องนำ cache ตัวใดตัวหนึ่งออก
- Cache Penalty คือ ค่าของความล่าช้าที่จะเกิดขึ้นในการทำงาน ในกรณีที่เกิด Cache Miss โดยทั่วไปจะใช้ค่าของ clock cycle ที่เสียไปเนื่องจากการรอ



Write

- ในการณ์ที่มีการเขียนข้อมูล และตำแหน่งที่ต้องการเขียนอยู่ใน Cache พอดี ก่อนอื่นต้องเข้าใจว่าข้อมูลใน Cache ไม่ใช่ข้อมูลต้นฉบับ แต่เป็นข้อมูล copy เท่านั้น
- กรณีแบบนี้มีวิธีจัดการ 2 รูปแบบ
 - Write-through เมื่อมีการเขียนข้อมูลลงไปใน cache ก็จะเขียนลงในหน่วยความจำลับถัดไปด้วยเลย
 - Write-Back จะเขียนลงเฉพาะใน Cache โดยยังไม่เขียนลงในหน่วยความจำลับถัดไป แต่จะ mark ที่ dirty bit ของบล็อกนั้นเอาไว้ โดยเมื่อมีความจำเป็นต้องเอาบล็อกนั้นออก ค่อยเขียนลงในหน่วยความจำลับถัดไปทีเดียว
- Write-Back จะมีประสิทธิภาพดีกว่า เพราะไม่ว่าจะเขียนลงใน cache กี่ครั้ง ก็จะนำไปเขียนลงในหน่วยความจำครั้งเดียว
- Write-through มีข้อดีที่ง่ายต่อการดำเนินการมากกว่า



Type of Cache Misses

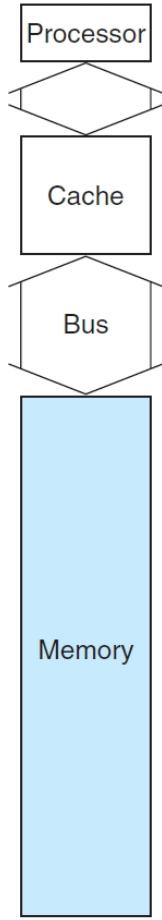
- Compulsory misses : เป็นการ miss ที่เกิดขึ้นจากการอ้างถึงครั้งแรก การ miss ชนิดนี้ ไม่มีวิธีการลดได้ แม้จะมี Cache ที่ใหญ่มากๆ ก็ยังต้องเกิดอยู่ดี
- Capacity misses : เป็นการ miss ที่เกิดจากการมีขนาด cache น้อยกว่าการใช้งานของโปรแกรม เช่น สมมติว่า cache มีขนาด 200 ตำแหน่ง แต่โปรแกรมมีการอ้าง array ขนาด 1,000 ตำแหน่ง (แบบ random) เหตุการณ์ที่จะเกิดขึ้น คือ จะเกิด cache miss จำนวนมาก แต่หากเพิ่ม cache ให้มากขึ้น miss จะลดลง
- Conflict misses : เป็นการ miss ที่เกิดจาก การอ้าง address 2 ตำแหน่ง มีการ map ลงใน cache ตำแหน่งเดียวกัน การเพิ่ม way ใน associative cache จะสามารถลด miss ชนิดนี้ได้ (จะกล่าวถึงต่อไป)



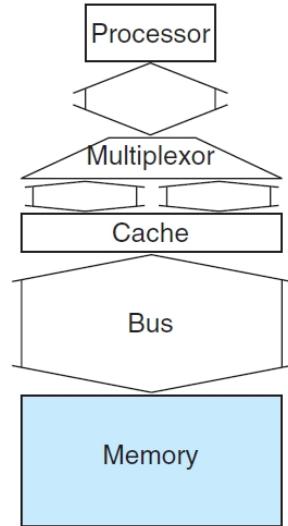
Design Memory Systems

- จากที่กล่าวถึง DRAM ในตอนต้นว่า มีข้อดีที่ ราคาถูก และมีความจุสูง แต่มีข้อเสียที่ ความล่าช้าในการ Access ครั้งแรก
- อย่างไรก็ตาม เราสามารถลด miss penalty ได้โดยการเพิ่มค่า bandwidth เช่น จากเดิม ในการอ้างหน่วยความจำจะใช้เวลาดังนี้
 - 1 memory bus cycle ในการส่ง address
 - 15 memory bus cycle ในการเข้าถึง DRAM (DRAM access initiated)
 - 1 memory bus cycle ในการส่งข้อมูล
- สมมติว่า cache block มีขนาด 16 ไบต์ (4 words) miss penalty เท่ากับ $1 + 4 \times 15 + 4 \times 1 = 65$ memory bus cycle ดังนั้นจำนวนไบต์ที่ส่งต่อ bus clock cycle เท่ากับ $4 \times 4 / 65 = 0.25$

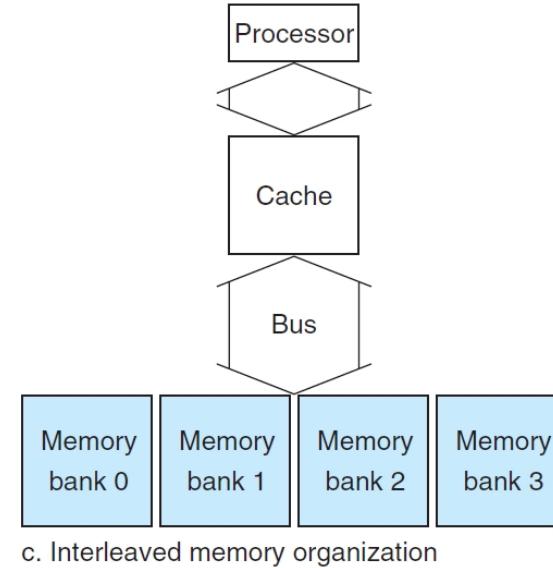
Design Memory Systems



a. One-word-wide
memory organization



b. Wider memory organization



c. Interleaved memory organization



Design Memory Systems

- รูป b ใช้วิธีการขยาย memory bus เพื่อให้ส่งข้อมูลได้มากขึ้น โดยขยายเป็น 2 เท่า ดังนั้น miss penalty จะลดลงจาก 65 เหลือ $1 + (2 \times 15) + 2 \times 1 = 33$ memory bus cycle ดังนั้นจำนวนไบต์ที่ส่งต่อ bus clock cycle เท่ากับ $4 \times 4 / 33 = 0.48$
- แต่รูป b ก็จะเสีย access time เพิ่ม เนื่องจากมีการนำ multiplexer มาใช้
- รูป c มีการจัดโครงสร้างหน่วยความจำเป็น memory bank ทำให้สามารถอ่านหรือเขียนได้หลาย word ในเวลาเดียวกัน ดังนั้น miss penalty จะเท่ากับ $1 + (1 \times 15) + 4 \times 1 = 20$ ดังนั้นจำนวนไบต์ที่ส่งต่อ bus clock cycle เท่ากับ $4 \times 4 / 20 = 0.8$
- ซึ่งจะเห็นว่า miss penalty มีค่าลดลงได้มาก



Cache Performance

- เมื่อนำผลของ miss penalty ซึ่งทำให้โปรแกรมทำงานช้าลง ทำให้สามารถเขียนเป็นสมการได้ดังนี้

$$\text{CPU time} = (\text{CPU execution clock cycles} + \text{Memory-stall clock cycles}) \times \text{Clock cycle time}$$

- **Memory-stall clock cycles** คือ จำนวน clock ที่ต้องรอกรณีที่ cache miss โดยมีค่าเท่ากับ Read-stall cycles + Write-stall cycles
- **Read-stall cycles** = $(\text{Reads/Program}) \times \text{Read miss rate} \times \text{Read miss penalty}$
- **Write-stall cycles** = $((\text{Writes/Program}) \times \text{Write miss rate} \times \text{Write miss penalty}) + \text{Write buffer stalls}$
- ในที่นี้จะถือว่า Write buffer stalls มีค่าน้อย จนสามารถตัดทิ้งได้



Example

- การรันโปรแกรมหนึ่ง มี miss rate ในส่วน instruction cache 2% และ miss rate ในส่วน data cache 4% ถ้า Processor มี CPI = 2 (คิดแบบ Perfected Cache คือ ไม่ miss) กำหนดให้ miss penalty = 100 clock cycles ให้หาว่า Processor ตัวนี้ทำงานด้วยประสิทธิภาพเท่าใด เทียบกับ Perfected Cache โดยสมมติว่าในโปรแกรมมีคำสั่ง load/store อยู่ 36 %
 - กำหนดให้ | คือ คำสั่งทั้งหมดของโปรแกรมนี้
 - Instruction miss cycle = $| \times 2\% \times 100 = 2|$
 - Data miss cycle = $| \times 36\% \times 4\% \times 100 = 1.4|$
 - จำนวน Memory stall cycle ทั้งหมด = $2| + 1.4| = 3.44|$ เมื่อรวมกับ CPI เดิม = 5.44|
 - ประสิทธิภาพเทียบกับ Perfect Cache = $2/5.44 = 0.36$



- ย่อมาจาก Average Memory Access Time เป็นค่าเฉลี่ยของการเข้าถึงหน่วยความจำโดยพิจารณาทั้ง Hit และ Miss
 - = Time for a hit + Miss rate x miss penalty
- เช่น Processor ตัวหนึ่ง มี clock cycles = 1 ns โดยมี miss penalty = 20 clock cycles และมี miss rate = 0.05 miss per instruction กำหนดให้ cache access time = 1 clock cycles
 - AMAT = 1 + 0.05x20 = 2 clock cycles



Thank you!

for your attention