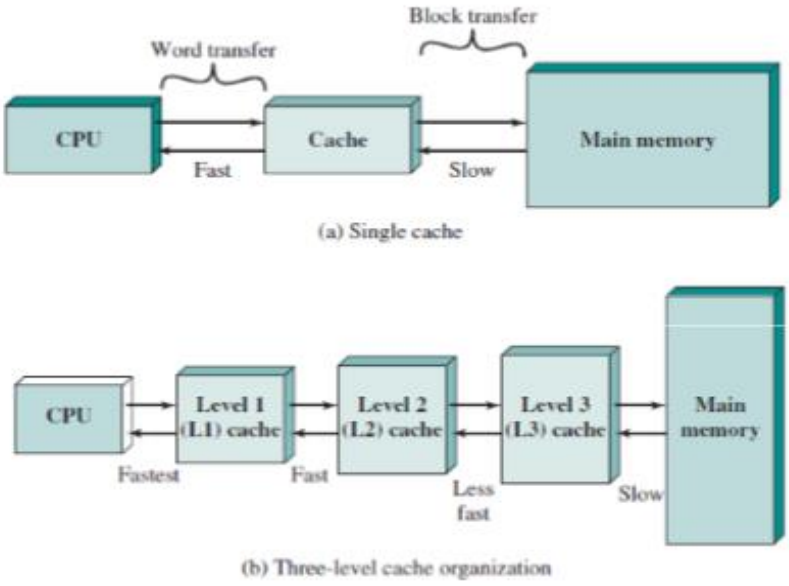
สรุปแนวข้อสอบ Com Arc Final 59

มีทั้งหมด 3 ข้อใหญ่ คะแนนเต็ม 80 คะแนน

1. Cache Memory 3 ข้อ 30 คะแนน

1.1 หลักการของ Cache ทำไมถึงมี Cache มันมีประโยชน์ยังไง 5 คะแนน



รูปที่ 4.3 Cache and Main Memory

ทำไมต้องต้องมี Cache ?

เพราะหน่วยความจำนั้นตามความเร็วของ CPU ไม่ทัน ซึ่งเป็นเหตุทำให้ CPU ต้องเสียเวลาในการรอ จึงสร้าง Cache เพื่อนำเอาคำสั่งข้อมูล ที่ใช่บ่อยๆ ไปไว้ใน Cache ซึ่ง Cache จะมีความเร็วสูงกว่าหน่วยความจำหลัก เพราะ Cache ถูกออกแบบมาเพื่อผสมผสานการใช้งานหน่วยความจำที่มีราคา แพง ความเร็วสูง กับหน่วยความจำที่มีขนาดความจุมาก ราคาถูก ความเร็วต่ำ

หลักการทำงาน

Cache จะ copy ข้อมูลบางส่วนจากหน่วยความจำ ซึ่งก็คือข้อมูลที่ใช้บ่อยๆ เมื่อ Processor ต้องการอ่านข้อมูลในหน่วยความจำ ก็จะเช็คใน Cache ก่อนว่ามีไหม ถ้ามี ก็ส่งไปให้ Processor แต่ถ้าไม่มีใน Cache ก็จะโหลด Block (Block เก็บหลายๆ word รวมกันเป็น Block) จากหน่วยความจำที่มีข้อมูลนั้นมายัง Cache แล้วส่งต่อให้ Processor ตามลำดับ

Cache มีทั้งหมด 3 ระดับ ดังรูปที่ 4.3 b

ระดับที่ 1 เรียก Caehe L1 จะทำงานเร็วสุด แต่ความจุน้อยสุด

ระดับที่ 2 เรียก Caehe L2 จะทำงานช้ากว่า L1 แต่ความจุมากกว่า L1

ระดับที่ 3 เรียก Caehe L3 จะทำงานช้ากว่า L2 แต่ความจุมากกว่า L2

1.2 Mapping Function : Direct , Associative, ฯลฯ จะมีตารางมาให้แล้วเติมข้อมูลลงใน Cache 20 คะแนน มี 3 แบบหลักๆ คือ

1.Direct Mapping

เป็นวิธีที่ง่ายที่สุด โดยให้ 1 block ของหน่วยความจำ เท่ากับ 1 line ของ cache โดยการย้ายข้อมูลจาก block ที่ j ไป line ที่ i จะได้ดังสมการนี้ i = j modulo m เมื่อ i = หมายเลขตำแหน่ง line ใน cache, j = หมายเลขตำแหน่ง block ในหน่วยความจำ และ m = จำนวนของ line ทั้งหมด ใน cache เช่น

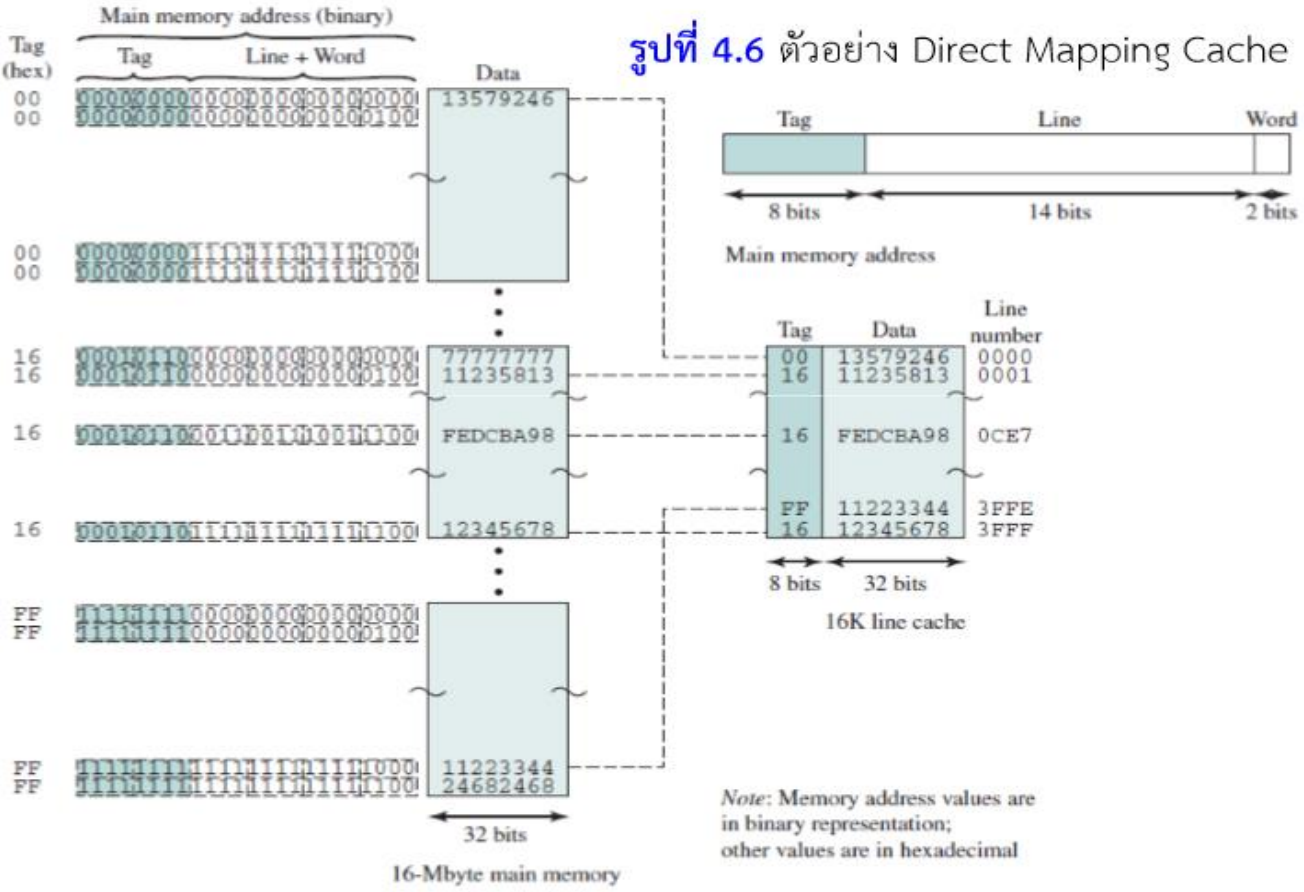
ย้าย block ที่ 0 ถึง block ที่ m-1 ไปยัง Cache จะได้

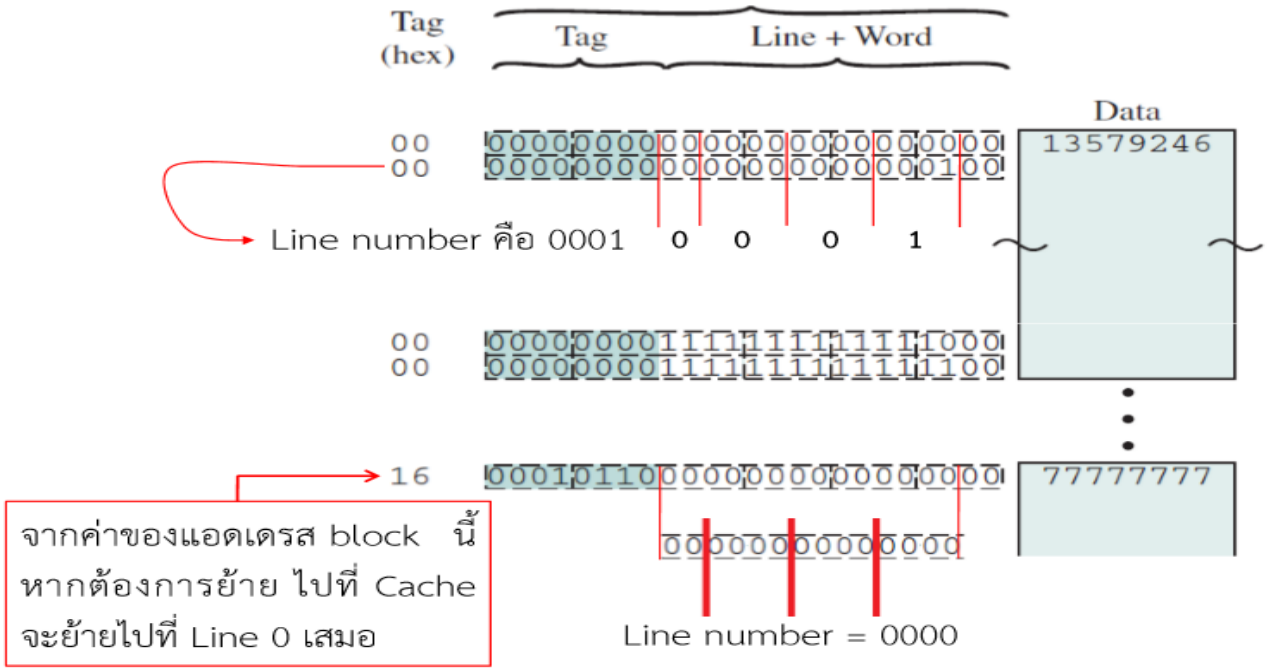
block ที่ 0 ถูกโหลดไปยัง Cache ที่มีหมายเลข Line เท่ากับ 0

block ที่ 1 ถูกโหลดไปยัง Cache ที่มีหมายเลข Line เท่ากับ 1

block ที่ m-1 ถูกโหลดไปยัง Cache ที่มีหมายเลข Line เท่ากับ m-1

ดัง 2 ภาพนี้


Main memory address (binary)
Tag
(hex)
Tag
Line + Word
Data
13579246
77777777
11235813
FEDCBA98
12345678
11223344
24682468
32 bits
16-Mbyte main memory
Tag
Line
Word
8 bits
14 bits
2 bits
Main memory address
Tag
Data
Line number
0000
0001
0CE7
3FFE
3FFF
8 bits
32 bits
16K line cache
Note: Memory address values are in binary representation; other values are in hexadecimal


Tag
(hex)
Tag
Line + Word
Data
13579246
77777777
Line number คือ 0001
0 0 0 1
จากค่าของแอดเดรส block นี้
หากต้องการย้าย ไปที่ Cache
จะย้ายไปที่ Line 0 เสมอ
Line number = 0000

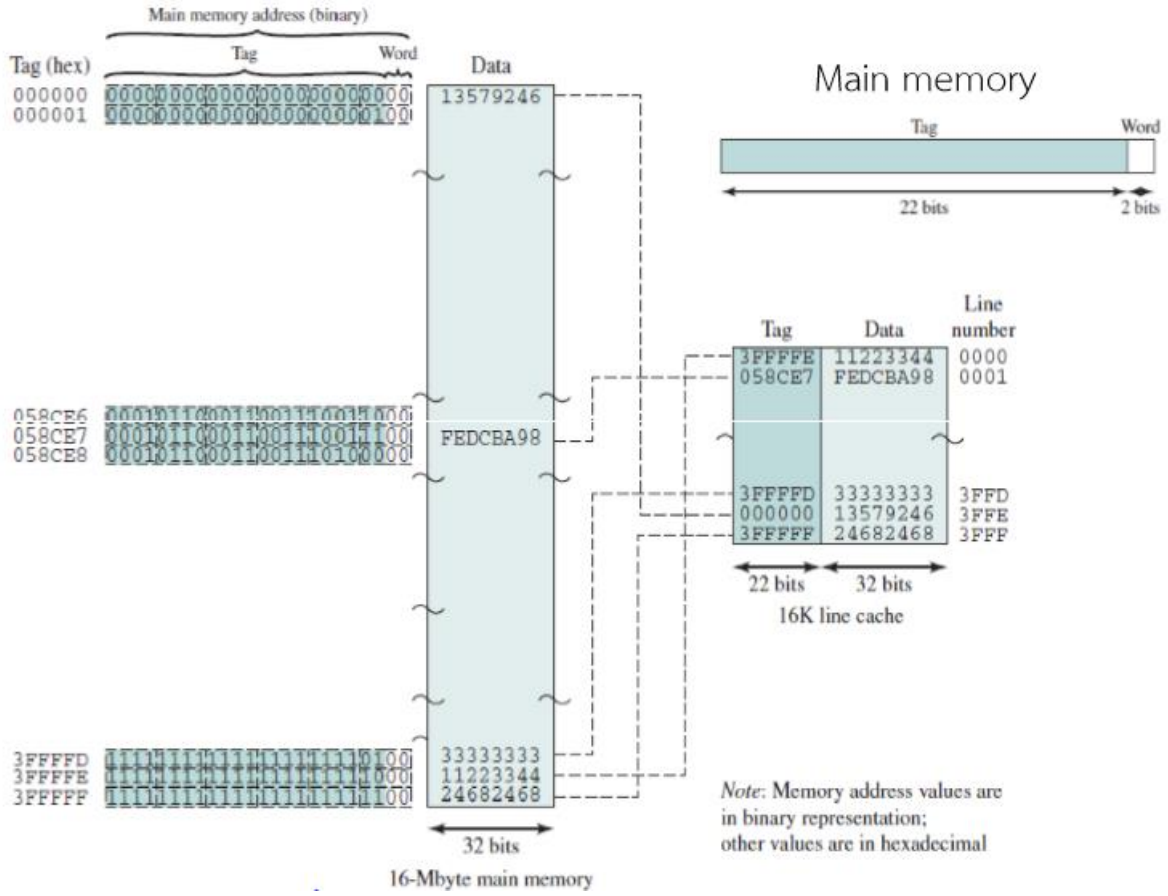
จะเห็นว่าในหน่วยความจำจะมี 8 บิต ที่ใช้เป็น Tag เพื่อให้ cache ชี้มายัง Tag ที่ต้องการ และในแต่ละ Tag จะมี Line + Word ทั้งหมด 16 บิต โดยจะแบ่งเป็น Line 14 บิต (ใช้เป็นเลขฐาน 2) และ Word 2 บิต โดยที่ Line ในหน่วยความจำนั้น จะถูกใช้เป็น Line number ใน cache (โดยใช้เป็นเลขฐาน 16 โดยแปลงมาจาก Line ซึ่งจะต้องใช้ 16 บิตในการแปลง เพราะ 4 บิตของฐาน 2 จะแปลงได้ 1 บิตในฐาน 16 แต่ Line มี 14 บิต ไม่ถึง 16 บิต ดังนั้น บิตแรกของ Line number จะใช้แค่ 2 บิตแรก ใน Line) ซึ่ง Line number นี้จะเป็นตัวชี้ Data ที่แตกต่างกันออกไป โดยใน Tag เดียวกัน Data แต่ละตัว จะมี Line number ไม่ซ้ำกัน แต่จะมี Line number ซ้ำกัน ใน Tag อื่น ซึ่งการที่ Line number ซ้ำกันนี้จะมีข้อเสียอยู่

ข้อดี : ง่ายและถูก

ข้อเสีย : วิธีนี้อาจจะมี 2 block ที่มีหมายเลข Line number ซ้ำกันเมื่อ โหลดมายัง Cache แล้วต้องอยู่ตำแหน่งเดียวกัน ซึ่งอยู่ตำแหน่งเดียวกันไม่ได้ ดังนั้นจะมีเพียง 1 block เท่านั้นที่อยู่ใน Cache ได้ จึงต้องทำการสลับเพื่อให้ตัวที่จะใช้เข้าไปแทนตัวเดิม แทนที่จะมี Hit ratio สูงๆ (การค้นพบข้อมูล ยิ่งสูงยิ่งเร็ว) กลับต้องมาเสียเวลาในการสลับข้อมูล

2. Associative Mapping

วิธีนำมาแก้ปัญหาของ Direct Mapping ที่มี Line number ซ้ำกัน โดย 1 block จะมี 1 Tag เท่านั้น ดังนั้นเมื่อเป็นแบบนี้ จะไม่มีทางซ้ำกันแน่นอน

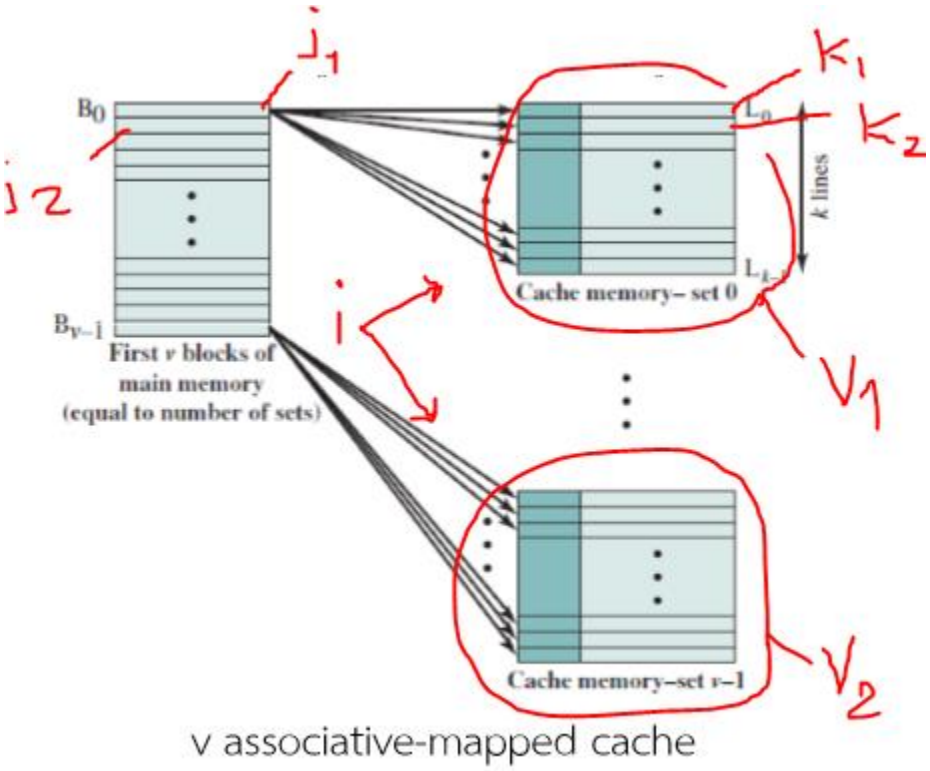


จากภาพนี้ ก็จะเห็นว่า Tag ของหน่วยความจำนั้น จะใช้บิตของ Line ใน Direct เพื่อมารวมเป็น Tag ซึ่งจะไม่ได้ใช้ Line เป็นตัวบอก Line number เหมือน Direct แต่จะใช้ Tag เป็นตัวกำหนดเอง โดย Tag จะมี 22 บิต เดิมจากวิธีเก่าจะมี 8 บิต แต่พอรวมกับบิตของ Line ด้วยอีก 14 ก็จะเป็น 22 บิต ซึ่งก็ใช้หลักการเดิมในการแปลงมาเป็นฐาน 16 โดยจะได้ฐาน 16 ทั้งหมด 6 บิต ซึ่งจริงๆต้องใช้ 24 บิตของฐาน 2 ในการแปลง (4*6 = 24) เป็นฐาน 16 แต่มีแค่ 22 บิต โดยให้ 1 บิตแรกของฐาน 16 แปลงจาก 2 บิตแรกของฐาน 2 เท่านั้น สังเกตุว่า บิตแรกของ Tag จะเป็นได้แค่เลข 0 1 2 3 เพราะใช้แค่ 2 บิต 2^2 = 4 จะได้แค่ 4 เลขเท่านั้น แล้วทีนี้ ก็จะไม่มีตัวบอก line number ใน cache แล้ว ดังนั้น cache จะรัน line number ด้วยตัวมันเองไปเรื่อยๆ โดยเริ่มตั้งแต่ 0000 ถึง 3FFF ซึ่งก็คือ 16838 ตำแหน่ง หรือ 2^14-1 หรือ 16K (16*1024) ซึ่งก็จะสามารถรันตำแหน่งได้เท่ากับความจุของ cache นั่นเอง

3.Set Associative Mapping

วิธีนี้จะนำข้อดีของทั้ง 2 แบบก่อนหน้านี้มารวมกัน และมันก็จะคือวิธีที่ดีที่สุด โดยจะแบ่ง Cache ออกเป็น Set และในแต่ละ Set จะมีแบ่งออกเป็น Lines อีกที มีอยู่ 2 แบบย่อย คือ v associative - mapped กับ k direct - mapped

3.1 v associative - mapped

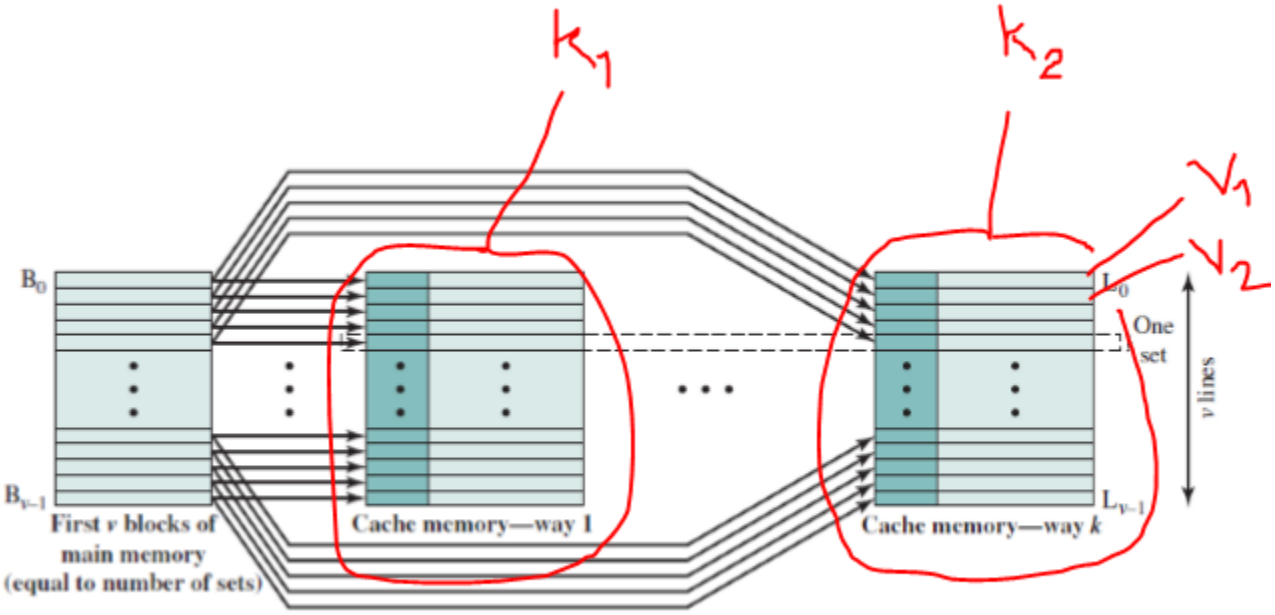


v associative-mapped cache

วิธีนี้ จะแบ่ง cache เป็น v set และในแต่ละ set จะมี k lines ซึ่งสามารถหาตำแหน่งได้จาก i = j mod v ซึ่ง i = ลำดับ set ใดๆ, j = ลำดับ block , v = จำนวน set ทั้งหมดใน cache , k = จำนวน line ในแต่ละ set และ m = v*k ซึ่ง m ก็คือจำนวน line ทั้งหมด ใน cahe

วิธีคิดเช่น เรามี set ทั้งหมด 4 set 0 1 2 3 แล้วไปโหลดข้อมูลบล็อคที่ B6 มา ก็จะเอามาเข้าสูตร i = 6 % 4 = 2 ก็จะได้ว่า i = 2 ซึ่งก็จะบอกว่าข้อมูลที่ไปโหลดมานี้ ให้ไปอยู่ set ที่ 2 แต่จะอยู่ใน line ไหนก็ได้นั่นหมายถึงจะเอาไว้ใน k ตำแหน่งที่เท่าไหร่ก็ได้ ใน set 2 ถ้า line นั้นว่างอยู่ แต่ถ้าข้อมูลมีแอดเดรสเดียวกัน ต้องสับเปลี่ยนตัวเก่าออก เพราะแอดเดรสเดียวกันอยู่ด้วยกันไม่ได้

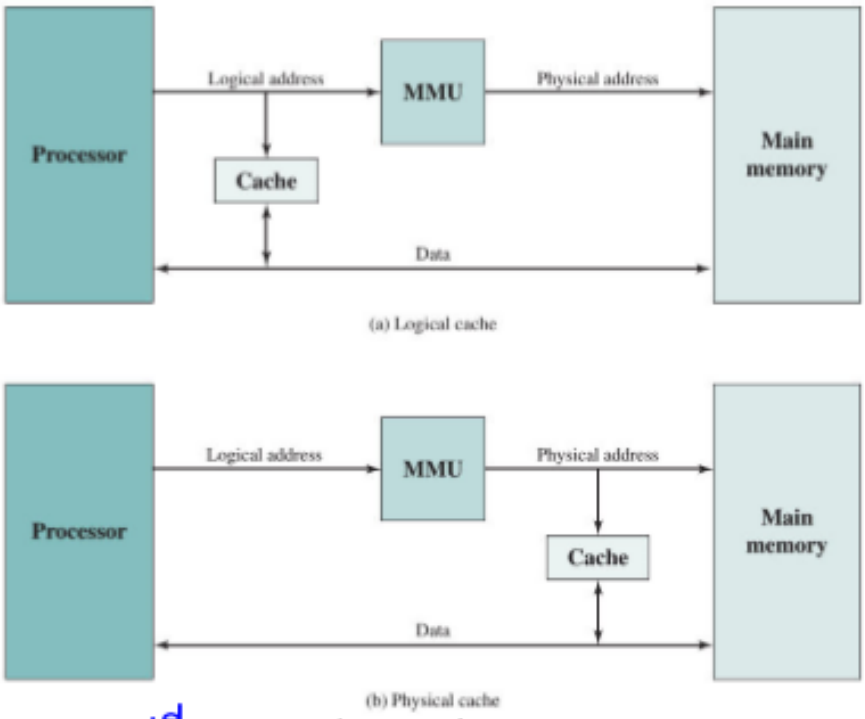
3.2 K direct - mapped



K direct – mapped caches

K direct - mapped โดยจะแบ่ง cache ออกเป็น k - way โดยแต่ละ way จะมี v line ซึ่ง v line ใน way ก็คล้ายกับ k line ในแบบข้างบน โดยวิธีนี้จะเจ๋งกว่า เพราะแต่ละ way จะใช้ แอดเดรสเดียวกัน แต่แยก way ในการเก็บข้อมูล ทำให้สามารถเก็บแอดเดรสเดียวกัน แต่ข้อมูลต่างกันได้ แยก way กี่อัน ก็คือ k เท่านั้น เช่นในรูป 2 อัน ก็คือ 2 – way และ 4 – way คือวิธีที่ดีที่สุด

1.3 Logical และ Physical Cache 5 คะแนน



รูปที่ 4.3 Cache and Main Memory

Logical Cache หรือ แอดเดรสเสมือน (Virtual Cache) : CPU เข้าถึง Cache โดยตรงไม่ผ่าน MMU (Memory Management Unit) เป็นฮาร์ดแวร์ที่ทำหน้าที่แปลงแอดเดรสเสมือนให้เป็น Physical address ที่ชี้ไปยังหน่วยความจำหลัก โดยจะใช้แอดเดรสเสมือน เพราะ Cache ไม่ได้เป็นส่วนหนึ่งของหน่วยความจำ

Physical Cache : CPU เข้าถึง Cache โดยผ่าน MMU โดย Cache จะเป็นส่วนหนึ่งของหน่วยความจำ ดังนั้น แอดเดรสจะเป็นส่วนหนึ่งของหน่วยความจำด้วย

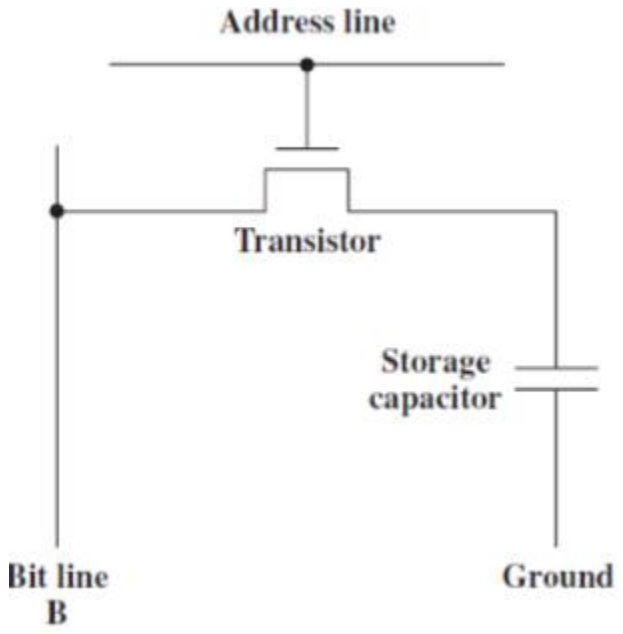
ข้อดีของ Logical Cache เข้าถึงเร็วกว่า Physical Cache เพราะต่อตรงจาก CPU

ข้อเสีย ในเรื่องของการกำหนดตำแหน่งให้กับ Application เพราะใน Logical Cache แต่ละ Application จะต้องกำหนดให้เป็นตำแหน่งเดียวกัน แต่เวลาชี้ไปยัง Physical Cache จะไม่ใช้ตำแหน่งเดียวกัน เพราะ Application คนละตัวจะโหลดคนละตำแหน่ง ดังนั้นตอนโหลด Block มายัง Cache ต้องนำรายละเอียดของ Application มาด้วยหรือต้องมีการเพิ่มบิตพิเศษ ในแต่ละ Line ใน Cache เพื่อชี้ไปยัง Physical Cache ให้ต่างตำแหน่งกัน

2. Internal RAM 3 ข้อ 20 คะแนน

2.1 หลักการพื้นฐานของ SRAM, DRAM

Dynamic RAM (DRAM) จะเก็บข้อมูลโดยใช้ ค่าของ คาปาซิเตอร์

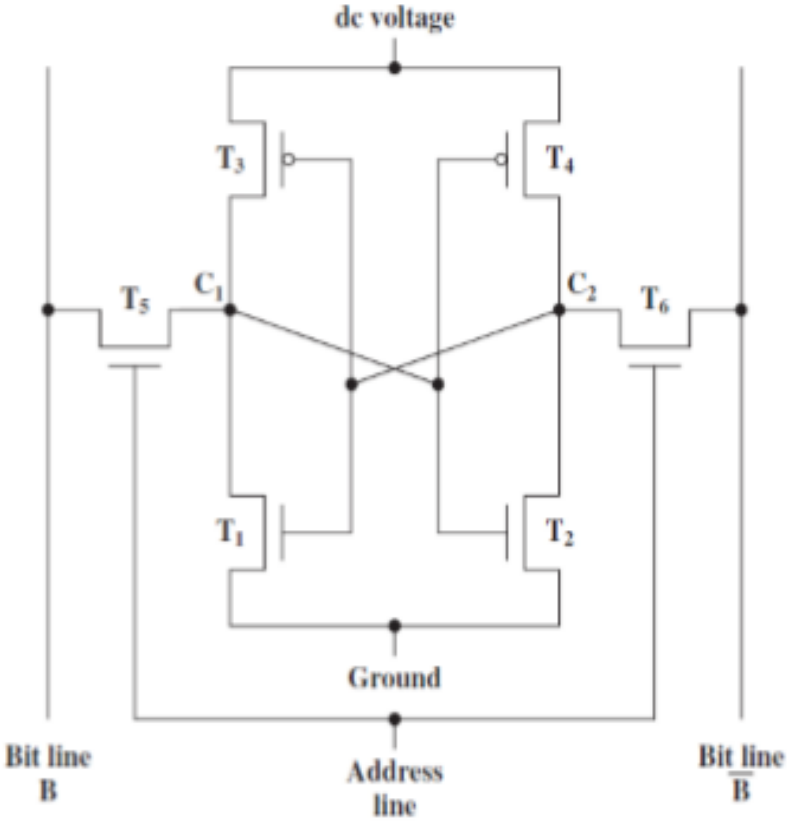


จะเขียนข้อมูลโดย DRAM ส่งข้อมูลมาที่ bit line พร้อมกับสัญญาณแอดเดรส เมื่อมีสัญญาณแอดเดรส จะทำให้ทรานซิสเตอร์ ON แล้วแรงดันที่ขา bit line จะถูกชาร์จมาที่ Storage Capacitor แล้วเมื่อสัญญาณแอดเดรสหายไปก็เสร็จการเขียนข้อมูล

จะอ่านข้อมูลโดยให้สัญญาณแอดเดรสมีค่าเท่ากับ 1 แล้วทรานซิสเตอร์ ON จะทำให้แรงดันที่ Capacitor ผ่านทรายซิสเตอร์ไปที่ขา bit line แล้วส่งต่อไป Sense Amplifier เพื่อเปลี่ยนสถานะแรงดันตามที่ตรวจจับได้ แล้วประจุที่ Capacitor จะ discharge

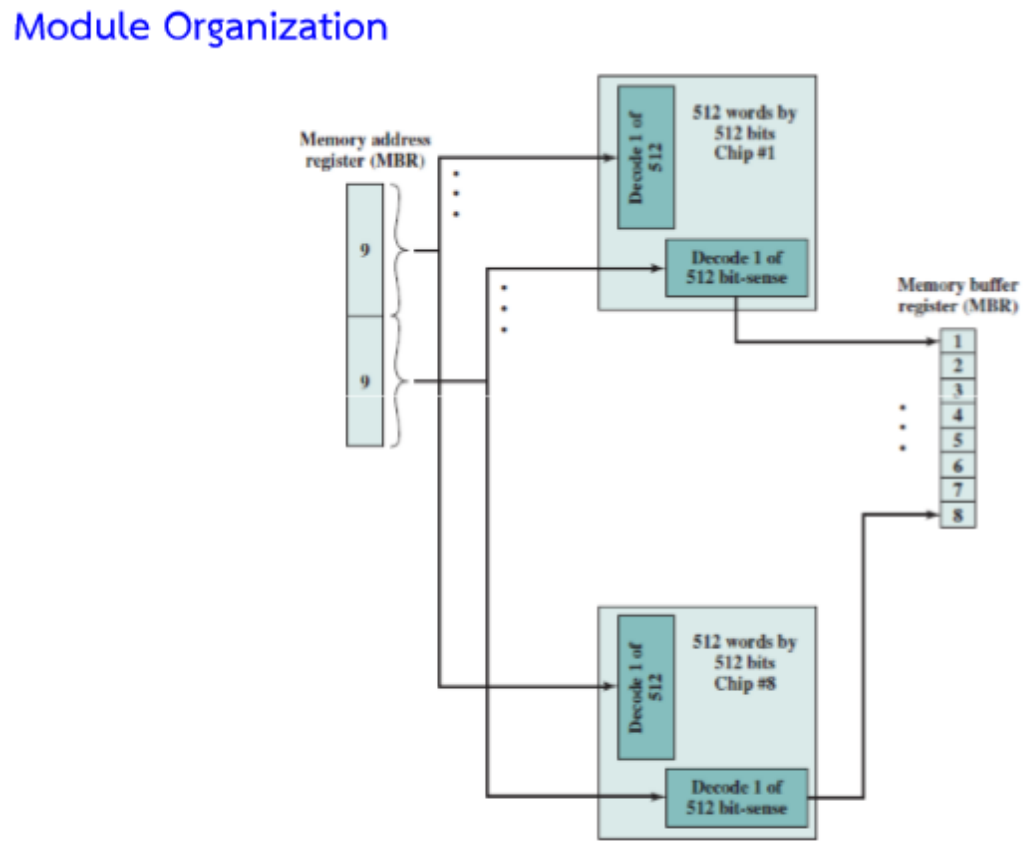
ดังนั้น DRAM จะต้องรีเฟรชข้อมูลที่ Capacitor ทุกๆคาบเวลาเพื่อให้ข้อมูลไม่หาย ซึ่งจะยุ่งยากไม่น้อย แต่ก็มีข้อดี คือง่ายต่อการสร้าง และราคาถูก

Static RAM (SRAM) จะเก็บข้อมูลโดยใช้ Flip-Flop



โครงสร้างจะใช้ทรานซิสเตอร์หลายตัว เมื่อเทียบกับ DRAM แต่ข้อดีคือไม่ต้องรีเฟรช และทำงานได้เร็วกว่า แต่ข้อเสียก็คือ วงจรเยอะก็ต้องจ่ายเยอะ ถูกนำไปใช้เป็น cache

2.2 การนำ DRAM มาต่อเป็น Memory Module น่าจะบอกว่า ถ้าใช้ DRAM เท่านี้ ความจุจะเท่าไหร่ หรือต้องการความจุเท่านี้ ต้องใช้ DRAM เท่าไหร่ แล้วให้วาดรูปโมดูลด้วย



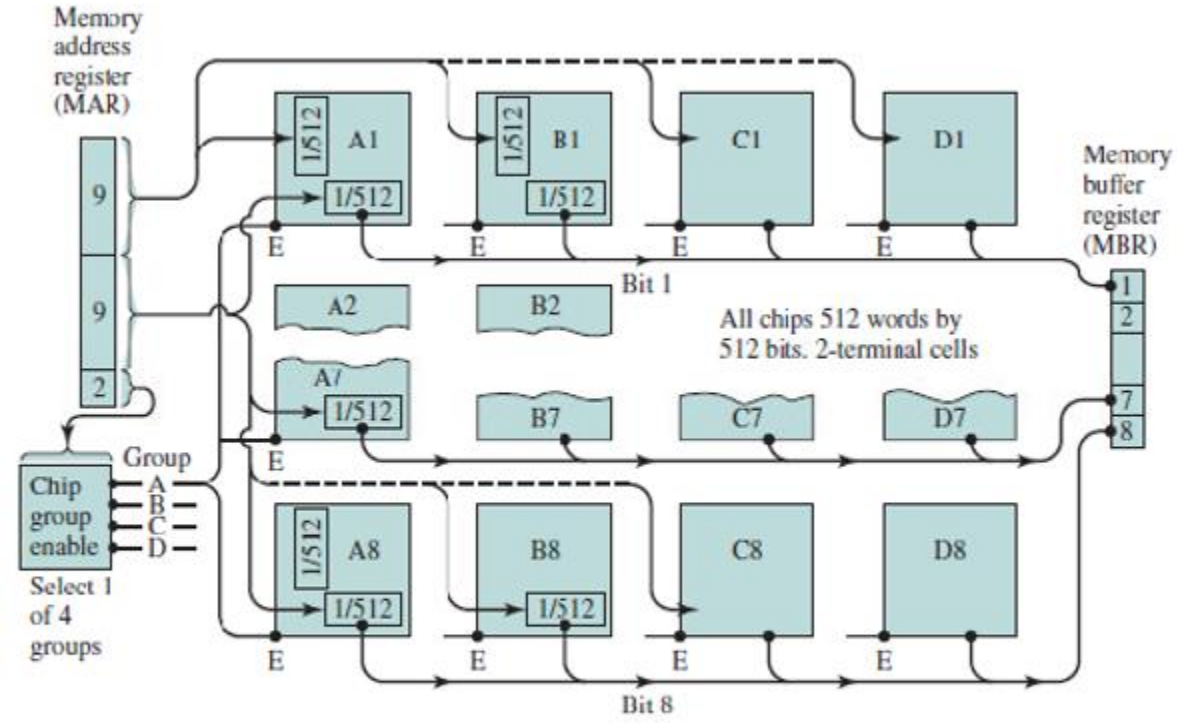
รูปที่ 5.7 256 KByte Memory organization

เช่นรูปนี้จะเห็นว่ามีชิป DRAM 512*512 อยู่ในรูป 2 ตัว ซึ่งจริงๆมีทั้งหมด 8 ตัว โดยชิปจะต่อไปที่ MBR 8 บล็อค ซึ่งคือบล็อคละ 1 บิต 8 บล็อคทั้งหมด 8 บิต หมายความว่าประกอบด้วย word ขนาด 8 บิต ทั้งหมด 256 K word ซึ่ง 256 K ก็คือความจุของโมดูลนี้ นั่นคือ 256*1024 = 262144 จะสามาถบอกตำแหน่งด้วยขนาด log2(262144) = 18 บิต ซึ่ง 2^18 ก็คือ 262144 ตำแหน่งนั่นเอง

จารย์อาจจะให้ DRAM ขนาด 512*512 มา แล้วความจุจะได้เท่าไหร่ เราก็ฟอลโล่ตามข้างบนเลย เอา 512*512 ได้ 262144 แล้วเอาไปหาร 1024 เพื่อให้หน่วยเป็น K ก็จะได้ 256 K ออกมา แล้ววาดรูปตามนี้ไป

หรืออีกเคส จารย์อยากได้ความจุ 256 K ถามว่าจะใช้ DRAM เท่าไร เราก็เอา 256*1024 ก็จะได้ 262144 แล้วไป ถอนรูท(262144) ก็จะได้ 512 ก็เอาเลขนี้ มาเป็น DRAM 512*512 เพราะถ้าคูณกลับก็ได้จะได้ 262144 ตามเดิม

และอีกเคส เคสนี่จะมึนหน่อย

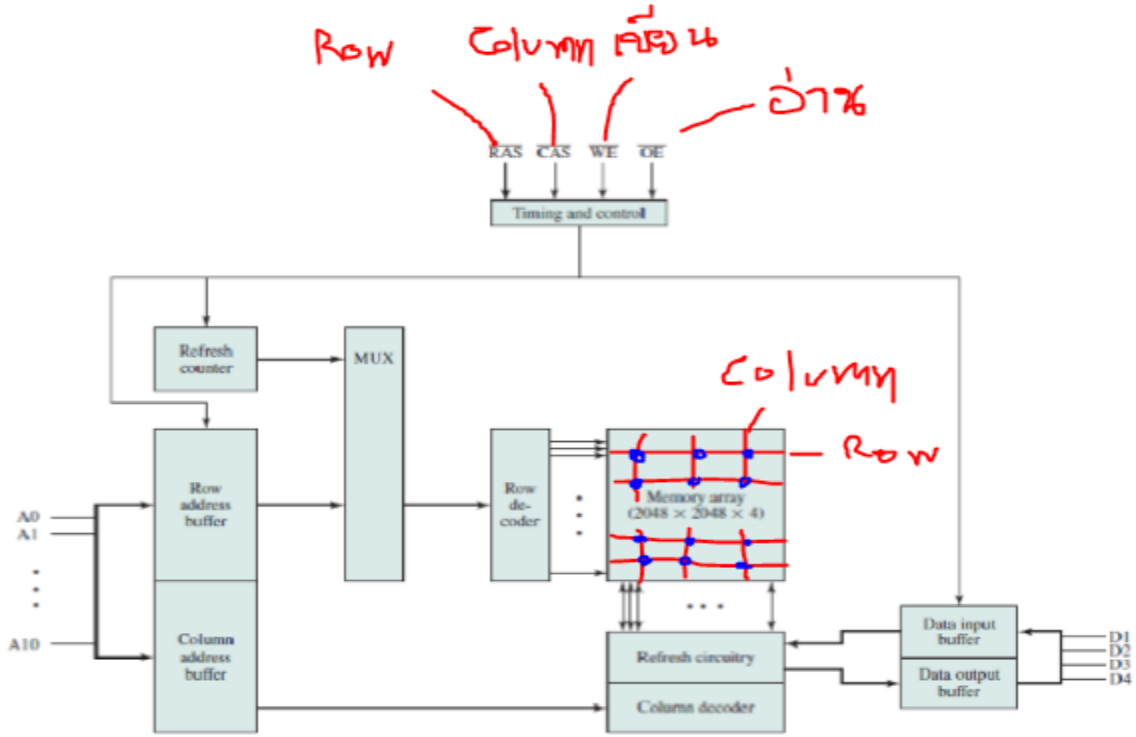


รูปที่ 5.8 1 MByte Memory organization

ซึ่งจะเห็นว่า โครงสร้างจะคล้ายรูป 256 K แต่จะมีชิป DRAM ตั้ง 4 ตัวต่อ 1 บิต ถ้ามองแบบไม่คิดเลขเลยก็ง่ายๆ 1 ตัว ได้ 256 K ดังนั้น 4 ตัวก็จะได้ 256*4 = 1024 = 1 M แต่พอลองคิดดูมันจะมีรายละเอียดนิดหน่อย คือสังเกตุว่า มันจะใช้ DRAM เท่าเดิมคือ 512*512 โดย DRAM เท่านี้คือ 18 บิต ซึ่ง 1 M = 1*1024*1024 = 1,048,576 ถอด log2(1048576) ได้ 20 บิต ซึ่งอีก 2 บิตนั่น มันจะเป็นตัวบอกว่า จะใช้ DRAM ตัวไหนในแต่ละแถว โดย 2 บิตก็จะเลือกได้ 2^2 = 4 ตำแหน่ง ซึ่งก็คือพอดีกับ DRAM 4 ตัวพอดี

2.3 อธิบายการทำงานของ DRAM Chip จะมีบล็อคไดอะแกรมมาให้ อธิบายการทำงาน

#เดาว่าน่าจะบล็อคนี้ แต่ถ้าไม่ใช้บล็อคนี้ หลักการก็ประมาณนี้ เพราะหัวใจของ DRAM คือการลดขาสัญญาณแอดเดรสให้น้อยลง ด้วยการ Multiplex



รูปที่ 5.5 16 M bit DRAM (4 M x 4)

บล็อคนี้เป็นชิปขนาด 2048*2048 และเก็บข้อมูลได้ 4 บิต เลยเป็น 2048*2048*4 = 16,777,216 หรือ 16 M ซึ่งก็คือขนาดของชิป โดยสามารถหาว่าชิปนี้จะใช้สายสัญญาณแอดเดรสกี่เส้น โดยเอา log2 ไปถอดมา จะได้ log2(2048) = 11 เส้น แต่จริงๆต้องใช้ 22 เส้น คือ Row 11 Colum 11 แต่ DRAM ใช้หลักการ Multiplex จึงใช้แค่ 11 เส้น โดยจะควบคุมผ่านขาสัญญาณ RAS และ CAS ว่าจะเป็น Row หรือ Column และขาสัญญาณจาก WE และ OE ว่าจะเขียนหรืออ่านข้อมูล โดย 11 เส้นนี้ ก็จะต่อเข้ากับ Row ก็จะต่อเข้ากับตัวถอดรหัสของ Row decoder และ column ก็จะต่อกับ column decoder เพื่อเชื่อมต่อสัญญาณที่ส่งเข้ามา แล้วส่งออกไป โดยทางออกของสัญญาณจะมีสายสัญญาณอีก 4 เส้นมาจากบัฟเฟอร์สำหรับการบันทึกและอ่านข้อมูล Data input & output buffer และส่วนของ Refresh Circuit จะต้อง disable ram ก่อน โดย Refresh Counter จะสร้าง แอดเดรสของ Row ที่ refresh จนครบทุก row โดยจะทำการอ่านและเขียนกลับไปในตำแหน่งเดิม

3. External Memory 3 ข้อ 30 คะแนน

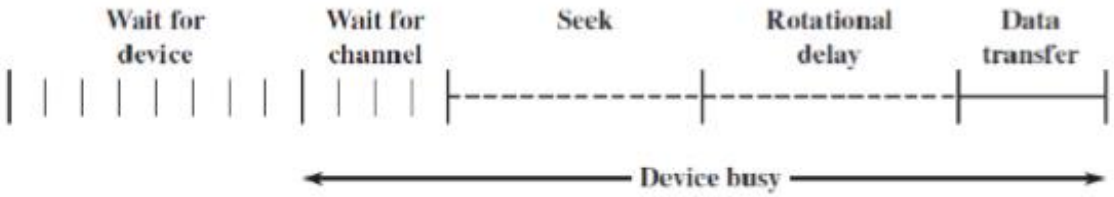
3.1 อธิบายหลักการของหัวอ่าน – บันทึก

ข้อมูลจะถูกบันทึกจากหัวอ่านและหัวบันทึก โดยหัวจะนิ่งอยู่กับที่ โดย Disk จะเป็นส่วนเคลื่อนที่อ่านและบันทึกข้อมูลเอง

การบันทึกข้อมูล จะป้อนสัญญาณไฟฟ้าที่เป็นพัลส์ตามรูปแบบข้อมูลที่จะบันทึก ให้ขดลวดของหัวบันทึก เพื่อเกิดสนามแม่เหล็กตามรูปแบบข้อมูลที่ผิวของ disk ที่เคลือบด้วยสารแม่เหล็ก แล้วก็บันทึกลงหัวบันทึกข้อมูลที่ทำจากสารที่ทำให้เกิดแม่หล็กไฟฟ้าได้ง่าย

การอ่านข้อมูล เมื่อก่อนใช้หัวอ่านทีเป็นขดลวด เมื่อผ่านผิวที่เป็นสนามแม่เหล็ก แล้วจะเกิดกระแสที่มีทิศขึ้นกับขั้วแม่เหล็กที่บันทึกลง disk ซึ่งเป็นแนวคิดที่ตรงข้ามกับการบันทึก แต่ hard disk ปัจจุบัน ใช้ MR (Magnetoresistive) Sensor โดยให้ค่าความต้านทานทางไฟฟ้าที่เปลี่ยนค่าตามสนามแม่เหล็กที่ใกล้เซนเซอร์ แล้วเปลี่ยนเปลี่ยนสัญญาณไฟฟ้าได้ ข้อดี ทำให้ความจุสูงและอ่านเร็วขึ้น

3.2 Disk I/0 Timing

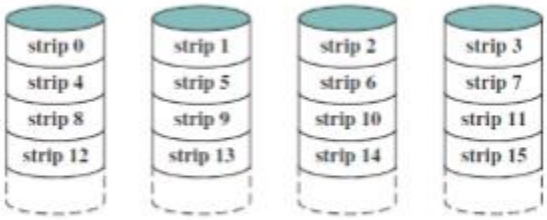


รูปที่ 6.7 Timing of Disk I/O Transfer

Timing คือระยะเวลาทั้งหมดในการถ่ายเทข้อมูลในการอ่านหรือเขียนข้อมูล โดยจะรอทั้งหมด 3 อย่างหลักๆ คือ Seek Time คือเวลาที่รอหัวอ่านไปยัง Track ที่ต้องการอ่าน/เขียน ซึ่งตั้งแต่เริ่มคำสั่ง Seek ช่องสัญญาณ i/0 จะถูกปล่อยให้ process อื่นใช้งาน พอถึง Track ที่ต้องการแล้ว ก็จะรอ Rotational delay ต่อ ก็คือเวลาที่หัวอ่านเคลื่อนที่จาก Track ไปยัง Sector เมื่อถึง Sector แล้ว จะติดต่อกลับไปยัง i/0 อีกครั้งเพื่อถ่ายเทข้อมูล (Transfer) แต่ถ้า i/0 channel ยังไม่ว่าง ก็ต้องรอ disk ในรอบต่อไป ซึ่งเวลาในการรอนี้ ก็จะบวกทบเข้าไปใน Timing เรื่อยๆ

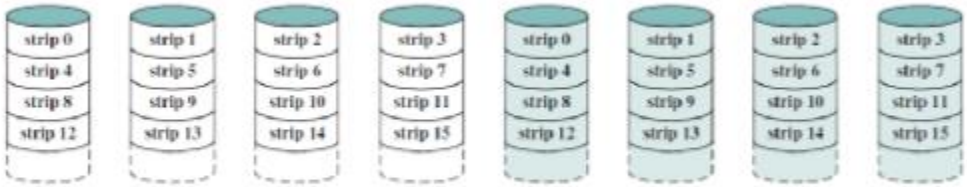
3.3 Hard Disk (RAID)

ใช้ hard disk หลายตัวทำงานร่วมกัน เพื่อความเร็วที่สูงขึ้น และกู้ข้อมูลได้ เรียกว่า RAID มีทั้งหมด 7 แบบ



RAID 0 (Nonredundant)

1. RAID 0 แบ่งข้อมูลเป็นส่วนๆ (Strip) สามารถทำงานได้เร็วขึ้น แต่กู้ข้อมูลไม่ได้

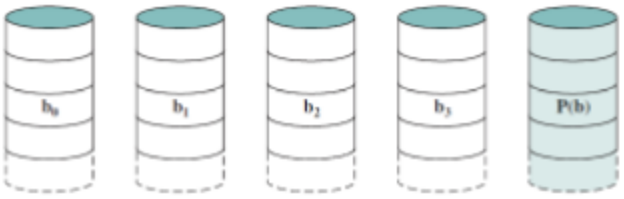


RAID 1 (Mirrored)

2. RAID 1 เหมือน RAID 0 แต่จะมีเพิ่มมาอีก 1 ชุด ทำให้กู้ข้อมูลได้ แต่ราคาแพง

RAID 2 (Redundancy through Hamming code)

3. RAID 2 ทำงานร่วมกัน จะอ่านตำแหน่งเดียวกันทุก disk ใช้ตรวจจับข้อผิดพลาด แต่สิ้นเปลือง



RAID 3 (Bit-interleaved parity)

4. RAID 3 ทำงานคล้ายกับ RAID 2 แต่ใช้ชุดเดียว กู้ข้อมูลได้ โดยใช้ parity bit เช่น

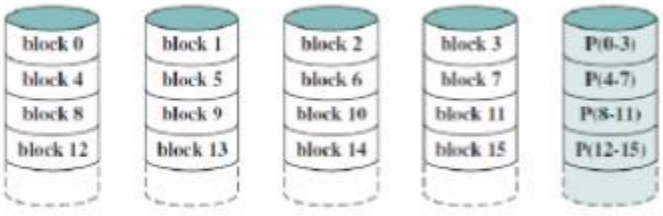
Parity bit → $X4(i) = X3(i) \oplus X2(i) \oplus X1(i) \oplus X0(i)$

where $\oplus$ is exclusive-OR function.

Suppose that drive X1 has failed. If we add $X4(i) \oplus X1(i)$ to both sides of the preceding equation, we get

$$X1(i) = X4(i) \oplus X3(i) \oplus X2(i) \oplus X0(i)$$

ถ้า x1 เสีย ก็จะสามารถกู้ได้ โดยแทน x4 xor x1 เข้าไปในสมการ แล้ว x4 เจอ x4 ก็จะหายไป เพราะ xor เหมือนกันได้ 0 และ x1 เจอ x1 ก็หายไป เหลือสมการใหม่ ที่จะได้ x1 มา ซึ่งก็คือการกู้ข้อมูลใน x1 นั่นเอง



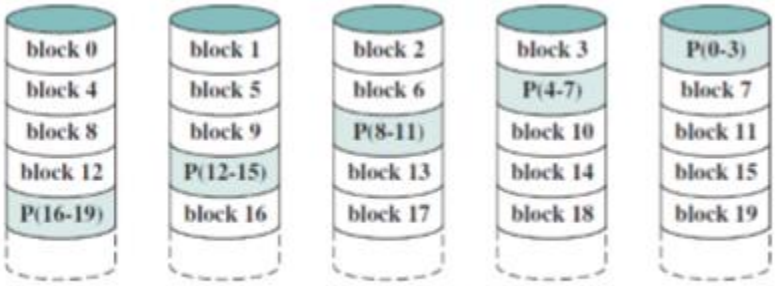
RAID 4 (Block-level parity)

5. RAID 4 Disk แต่ละตัวทำงานอิสระ ไม่เหมือน RAID 2-3 มีจะใช้ 1 disk เก็บ parity เลย แต่ถ้า disk ที่เก็บ parity พังก็พังทั้งหมด

$$X4(i) - X3(i) \oplus X2(i) \oplus X1(i) \oplus X0(i)$$

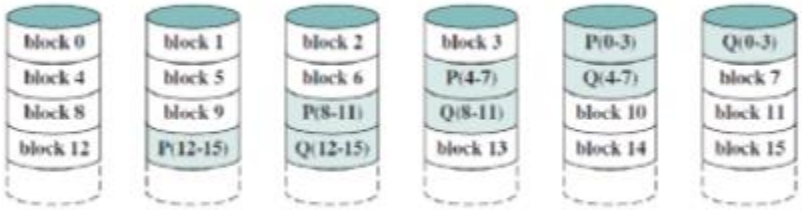
$$\begin{aligned} X4'(i) &= X3(i) \oplus X2(i) \oplus X1'(i) \oplus X0(i) \\ &= X3(i) \oplus X2(i) \oplus X1'(i) \oplus X0(i) \oplus X1(i) \oplus X1(i) \\ &= X3(i) \oplus X2(i) \oplus X1(i) \oplus X0(i) \oplus X1(i) \oplus X1'(i) \\ &= X4(i) \oplus X1(i) \oplus X1'(i) \end{aligned}$$

สามารถ update ข้อมูลได้รวดเร็วมาก เพราะจะใช้บิตของ parity เดิม ซึ่งจะน้อยกว่าเอาบิตทุกบิตมาคิด



RAID 5 (Block-level distributed parity)

6. RAID 5 คล้าย RAID 4 แต่จะกระจาย parity ไปในทุก disk ไม่อยู่ใน disk เดียวเหมือน RAID 4 เพราะถ้า disk ใดพังก็ไม่พังทั้งหมด ใช้กับเครื่อง server



RAID 6 (Dual redundancy)

7. RAID 6 จะใช้ parity 2 ชุด คือ P และ Q เก็บ parity ชุดเดียวกันโดยจะเก็บไว้ 2 disk เพื่อที่จะสามารถกู้ข้อมูลได้ แม้ disk จะเสีย 2 ตัวพร้อมกัน ต้องมีใช้ disk ข้อมูล N เท่ากับ N+2