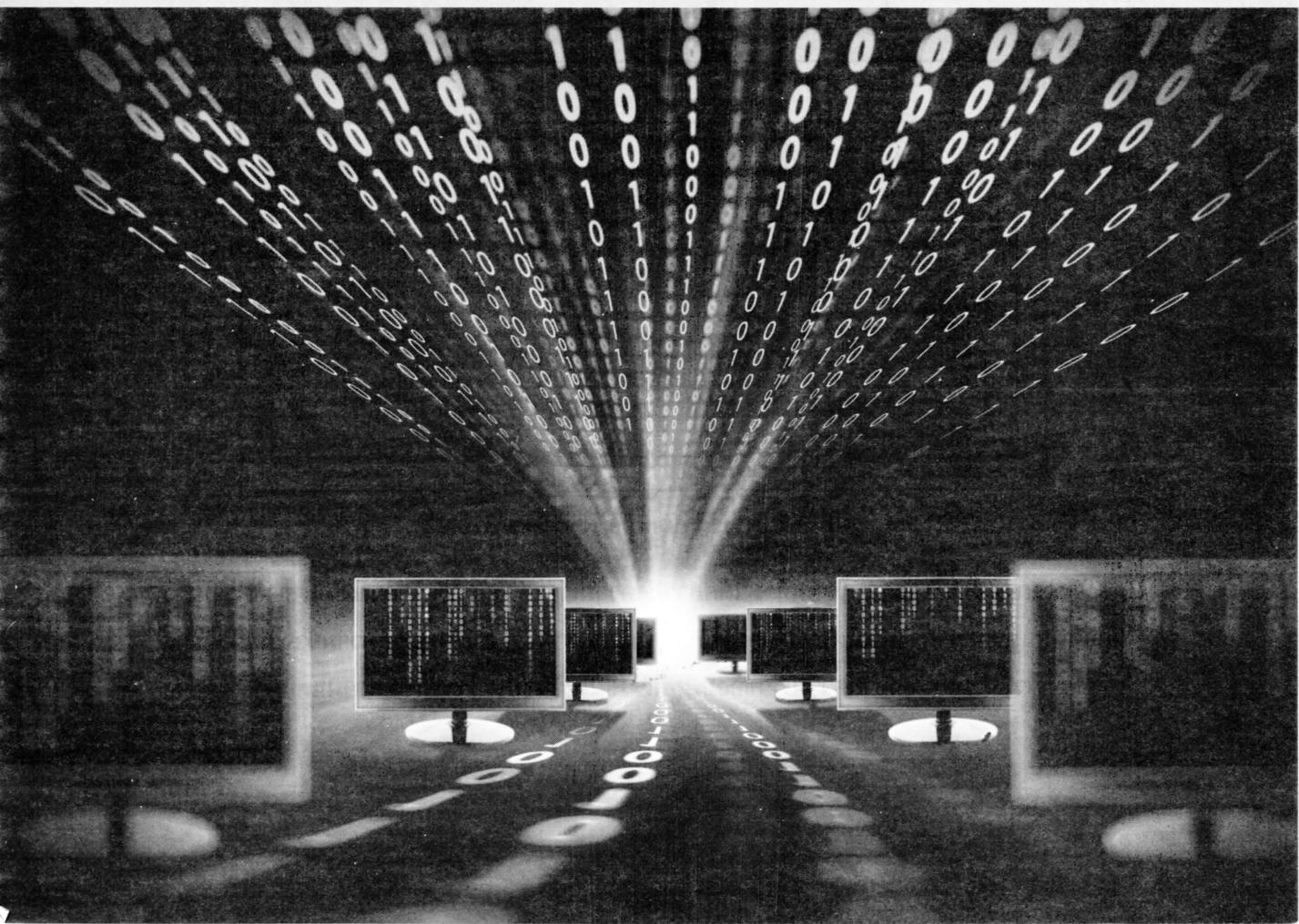


9

หน่วยความจำแบบเชิงชั้น (Memory Hierarchy)

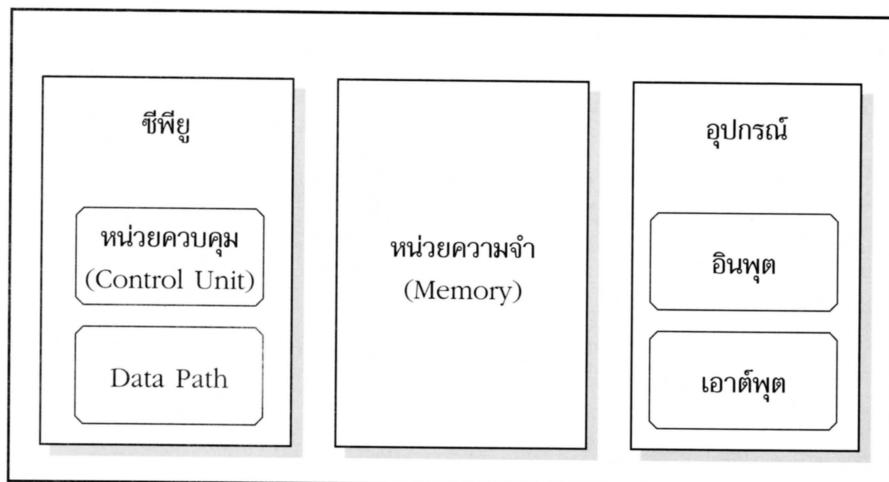


เครื่องคอมพิวเตอร์มีส่วนประกอบที่สำคัญอีกอย่าง ได้แก่ หน่วยความจำ การออกแบบหน่วยความจำนั้นมีความสำคัญในการเพิ่มประสิทธิภาพของการทำงานของโปรแกรม ซึ่งต้องอาศัยระบบหน่วยความจำที่ประกอบด้วยหน่วยความจำหลายระดับที่มีขนาดและมีกลไกการทำงานที่เหมาะสม

ในบทนี้จะกล่าวถึงองค์ประกอบของคอมพิวเตอร์ในส่วนของการจัดการหน่วยความจำ โดยเริ่มต้นจะกล่าวถึงหน่วยความจำระดับต่างๆ รวมทั้งปัจจัยที่เกี่ยวข้องกับราคาและคุณภาพของหน่วยความจำ รวมทั้งกล่าวถึงหน่วยความจำเสมือน (Virtual Memory) และแคช ความล้มเหลวของปัจจัยต่างๆ ที่เกี่ยวกับประสิทธิภาพของการทำงานของระบบหน่วยความจำ

■ 9.1 ความสำคัญของหน่วยความจำในเชิงสมรรถนะ

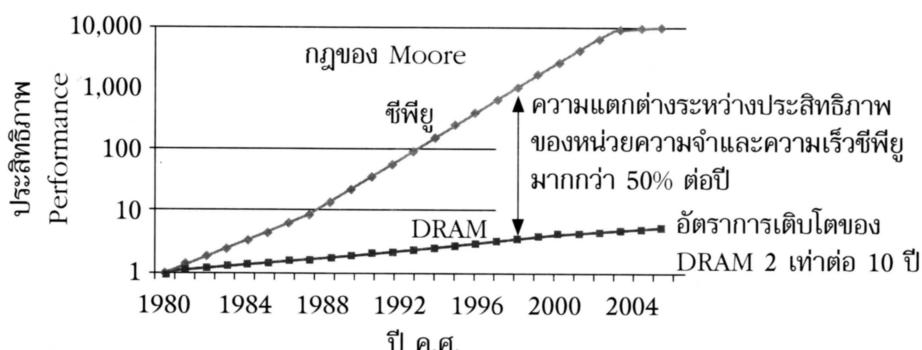
หน่วยความจำเป็นองค์ประกอบที่สำคัญของคอมพิวเตอร์ดังรูปที่ 9.1 จะเห็นว่าหน่วยความจำเป็นองค์ประกอบที่อยู่ภายใต้ออกข้อมูลชีพีย์



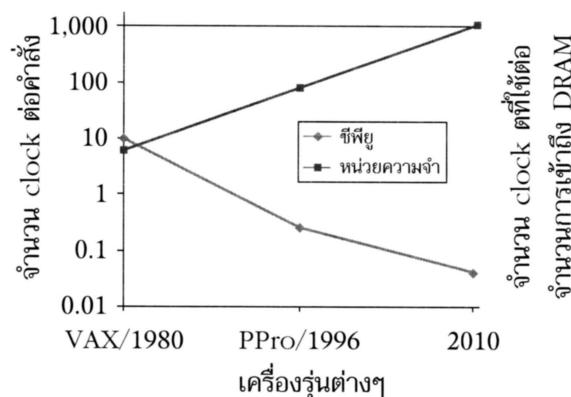
รูปที่ 9.1 โครงสร้างเมนบอร์ดของคอมพิวเตอร์

จากกฎของ Moore ในบทที่ 1 ได้กล่าวถึงความแตกต่างของอัตราการเติบโตของความเร็วของชิปีย์และอัตราการเติบโตของความเร็วของหน่วยความจำ แม้ว่าหน่วยความจำจะมีเวลาการเข้าถึงที่เร็วขึ้น แต่เนื่องจากความเร็วของชิปีย์ที่เร็วขึ้นในปัจจุบัน จะทำให้มีความแตกต่างระหว่าง

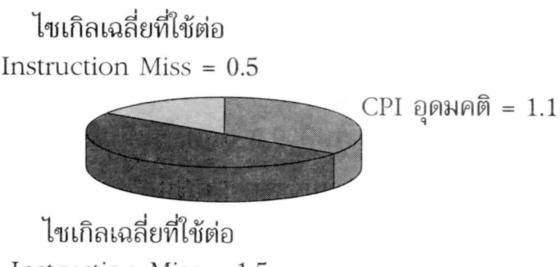
ความของห้องส่องมากขึ้น ดังนั้น จะเห็นว่าความเร็วของการเข้าถึงหน่วยความจำยังช้าเมื่อเทียบกับความเร็วของชิปปี้ การออกแบบระบบหน่วยความจำที่ดีจะมีส่วนช่วยให้เวลาในการเข้าถึงรวมทั้งการใช้งานหน่วยความจำเร็วขึ้น รูปที่ 9.2 แสดงให้เห็นถึงความแตกต่างของความเร็วของชิปปี้ และความเร็วของการเข้าถึงหน่วยความจำของแต่ละเครื่อง ยิ่งเครื่องคอมพิวเตอร์เร็วมากขึ้นเท่าไร ความแตกต่างนี้ก็จะมีมากขึ้นเท่านั้น



รูปที่ 9.2 Speed Gap ตามกฎของ Moore



รูปที่ 9.3 ความเร็วของการเข้าถึง DRAM สำหรับเครื่องรุ่นต่างๆ



รูปที่ 9.4 ใช้เกลลี่ยที่ต้องใช้ไปแต่ละส่วน กำหนดให้ CPI อุดมคติเป็น 1.1 และ Data Miss¹

จะเสียเวลาเพิ่มอีก 1.5 ใช้เกลลี่ย ส่วน Instruction Miss จะเสียเวลาเพิ่มอีก 0.5 ใช้เกลลี่ย

โดยทั่วไปการทำงานของโปรแกรมหนึ่งๆ ประกอบด้วยการใช้คำสั่งหลายรูปแบบ เช่น ประกอบด้วยคำสั่งที่เกี่ยวกับการทำงานของ ALU 50% คำสั่งประเกท Load/Store 30% และคำสั่งเกี่ยวกับการกระโดดหรือ Branch 20% และถ้าใน 30% ของคำสั่ง Load/Store นี้อาจจะก่อนให้เกิดการ Miss จำนวน 10% แต่ละครั้งของการ Miss จะต้องใช้เวลาถึง 50 ใช้เกลล์ในการเข้าไปอ่านข้อมูลในหน่วยความจำหลักและนำเข้ามาอย่าง凸缓 ถ้าสมมติให้ CPI เดิมที่ไม่ได้คิดเวลาที่เกิดจาก Miss นี้ เพาบกับ 1.1 ให้ลองคำนวณ CPI ใหม่เมื่อพิจารณาผลของ Miss นี้

$$\text{CPI} = \text{CPI}_{\text{เดิม}} + \text{จำนวน Stall เกลลี่ยต่อหนึ่งคำสั่ง}$$

$$= 1.1 (\text{ใช้เกลลี่ย}) + (0.30 (\text{จำนวนคำสั่ง Load/Store}) \text{ ต่อคำสั่ง}) \times 0.10 (\text{จำนวนเฉลี่ยของการ Miss ต่อหนึ่งคำสั่งที่เกี่ยวกับหน่วยความจำ}) \times 50 (\text{ใช้เกลลี่ยที่ใช้ต่อการ Miss หนึ่งครั้ง})$$

$$= 1.1 \text{ ใช้เกลลี่ย} + 1.5 \text{ ใช้เกลลี่ย} = 2.6 \text{ ใช้เกลลี่ย}$$

จะเห็นว่า 58% ของเวลาทั้งหมดจะเสียไปกับการรอการทำงานของหน่วยความจำให้เสร็จสิ้น

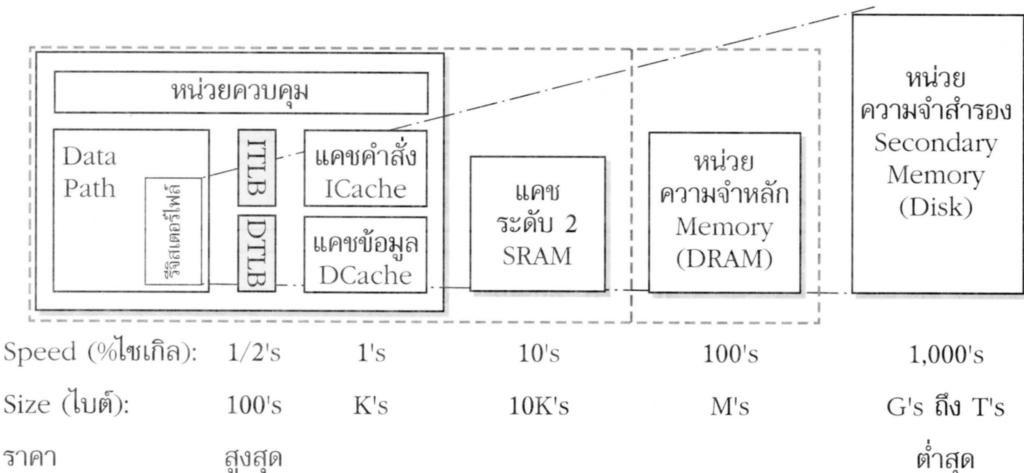
ดังนั้น จึงมีความจำเป็นต้องออกแบบระบบหน่วยความจำที่ดี เพื่อลดเวลาการเข้าหน่วยความจำโดยรวมให้ได้ ซึ่งจะทำให้เวลาในการทำงานของโปรแกรมทั้งหมดเร็วขึ้นนั่นเอง โดยในการออกแบบนี้จะอาศัยหลักการของการทำงานแบบบานาน และการออกแบบเป็นระดับชั้นมาก่อน

¹ การ Miss หมายถึงการเข้าถึงข้อมูลใน凸缓 แต่ข้อมูลที่ต้องการนั้นไม่ได้อยู่ใน凸缓

Data Miss เป็นการ Miss เนื่องจากการเข้าถึงข้อมูลในหน่วยความจำข้อมูลหรือ凸缓ข้อมูล และ Instruction Miss เป็นการ Miss เนื่องจากการเข้าถึงหน่วยความจำคำสั่งหรือ凸缓คำสั่ง

9.2 การแบ่งระดับหน่วยความจำและปัจจัยที่เกี่ยวข้อง

ในระบบคอมพิวเตอร์หนึ่งๆ มีหน่วยความจำได้หลายรูปแบบนับตั้งแต่รีจิสเตอร์ไปจนถึง DRAM และหน่วยความจำหลัก และตัวอย่างในส่วนของแอดเดรสสามารถมีแคชได้หลายระดับอีกด้วยกัน การจัดลำดับการเข้าถึงแคชจะกำหนดให้แคชระดับที่ 1 เป็นระดับที่เร็วที่สุด และอยู่ใกล้กับชิปซีพียูมากที่สุด



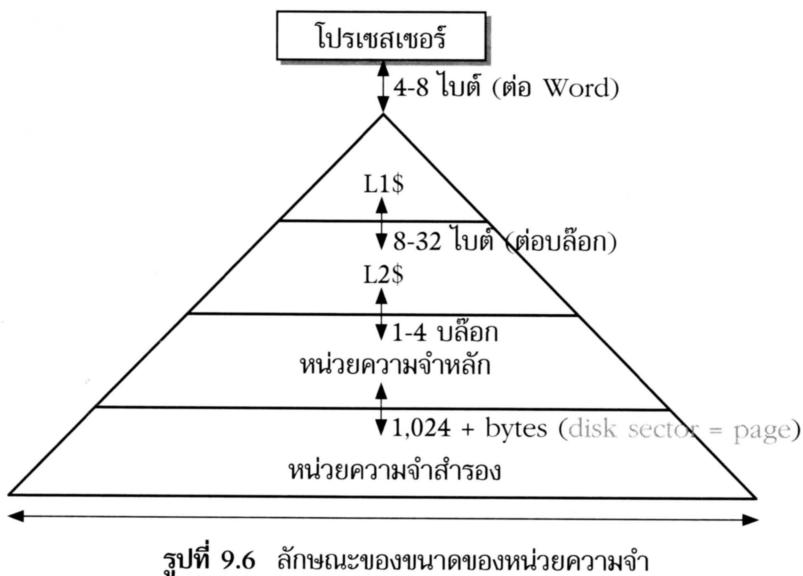
รูปที่ 9.5 ตัวอย่างของระดับชั้นของหน่วยความจำ

รูปที่ 9.5 และรูปที่ 9.6 แสดงตัวอย่างของระดับชั้นของหน่วยความจำ ได้แก่ รีจิสเตอร์/_FIFO², DTLB³, แคชคำสั่ง (ICache), แคชข้อมูล (DCache) และระดับถัดไป ได้แก่ SRAM, DRAM และฮาร์ดดิสก์ตามลำดับ ในรูปที่ 9.5 แสดงเวลาในการเข้าถึงข้อมูลแต่ละระดับ (ระบุเป็น微秒 เท่านั้นที่อยู่ในวงกลมที่ใช้) ขนาด (มีหน่วยเป็นไบต์) ราคากองแต่ละระดับจะเรียงจากราคามากไปน้อย และเรียงตามขนาดในแต่ละระดับจากเล็กไปใหญ่⁴

² ITLB ย่อมาจาก Instruction Table Look aside Buffer เป็นแคชพิเศษหรือบัฟเฟอร์ที่เก็บคำสั่ง

³ DTLB ย่อมาจาก Data Table Look aside Buffer เป็นแคชพิเศษหรือบัฟเฟอร์ที่เก็บข้อมูล

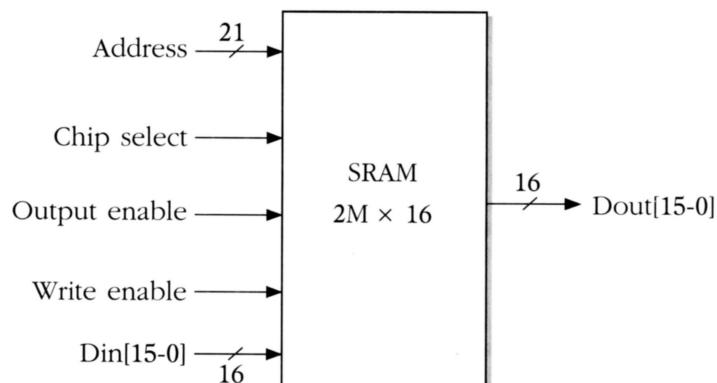
⁴ ข้อมูลในปี ค.ศ. 2004



■ 9.3 หน่วยความจำแรม (Random Access Memory: RAM)

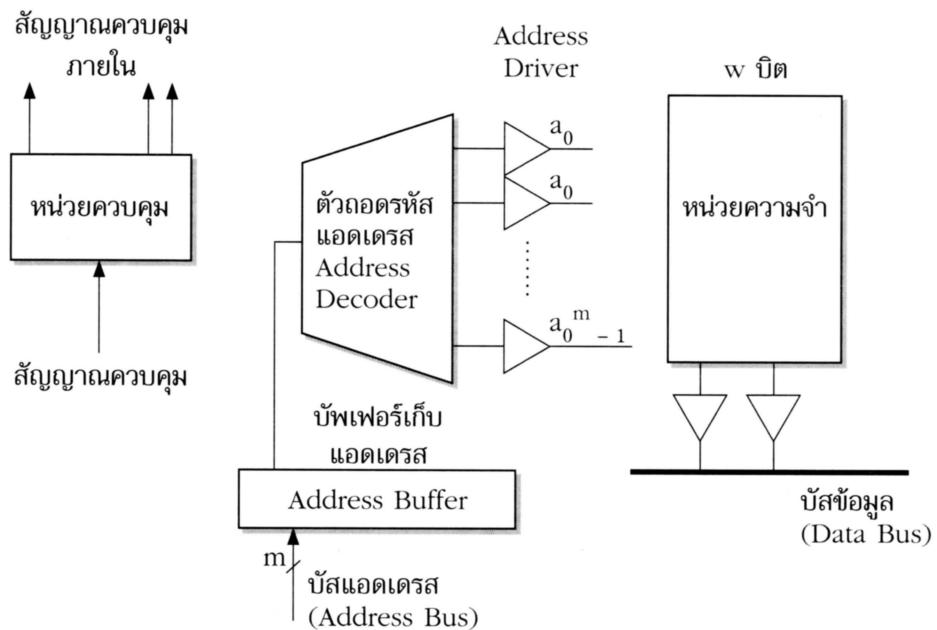
หน่วยความจำแรม (RAM) แบ่งเป็น 2 รูปแบบ ได้แก่ SRAM และ DRAM ตัว SRAM เป็น Static RAM ซึ่งข้อมูลจะถูกเก็บไว้จนกว่าจะปิดเครื่อง ส่วนใหญ่ใช้สำหรับเป็นแคช SRAM มีราคาแพงเนื่องจากทำจากเทคโนโลยีราคาสูง ที่มีความจุน้อย ดังรูปที่ 9.7 Din เป็นขาที่ໄร์ป้อนข้อมูลเข้าขนาด 16 บิต และ Address เป็นขาแอดเดรสที่ต้องการเขียนข้อมูลไป SRAM นี้มีความกว้างเท่ากับ 16 บิต มีจำนวนแควรเท่ากับ 2048 (หรือ 2M)

ขา Chip select มีไว้เพื่อเลือกใช้ SRAM ตัวนี้ (Enable เมื่อ Chip select = 1) สัญญาณ Output enable = 1 ใช้เพื่อต้องการอ่านค่าจาก SRAM โดยกำหนดตำแหน่งที่จะอ่านด้วย Address และจะได้ข้อมูลออกมากที่ Dout และให้สัญญาณ Write enable = 1 ใช้เพื่อกำหนดร่วมต้องการเขียนไปที่ SRAM นี้ โดยระบุตำแหน่งที่จะเขียนด้วยสัญญาณ Address



รูปที่ 9.7 โครงสร้าง SRAM

การออกแบบภายในรูปที่ 9.7 เป็นลักษณะแบบหนึ่งมิติ ประกอบด้วยแอดเดรสขนาด m บิต โดยสามารถเข้าถึงแอดเดรสได้ทั้งหมด 2^m ตำแหน่ง ดังรูปที่ 9.8

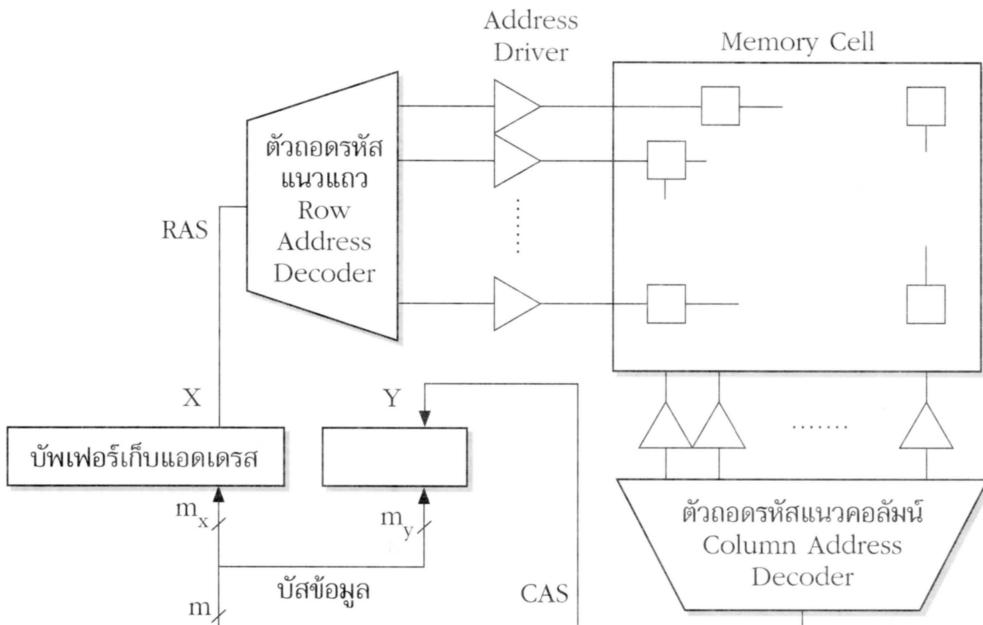


รูปที่ 9.8 ลักษณะทางโครงสร้างของ RAM แบบ 1 มิติ

รูปที่ 9.8 เป็นหน่วยความจำขนาด $2^m \times w$ บิต โดยที่ w คือ ขนาดของเวิร์ดจำนวน 1 เวิร์ด จะมีตัวถอดรหัสแอดเดรสเพื่อทำการเลือกแอดเดรสของหน่วยความจำที่ต้องการเข้าถึงในการอ่านข้อมูลจากหน่วยความจำ ตำแหน่งของข้อมูลที่จะอ่านจะถูกกำหนดโดยไดรฟ์เวอร์ของแอดเดรส (Address Driver) ข้อมูลจะถูกส่งออกมายังบัสข้อมูลและถ้าเป็นการเขียน ก็จะมีการส่งข้อมูลจาก Data Bus ไปยังหน่วยความจำ โดยจะกำหนดตำแหน่งโดยไดรฟ์เวอร์ของแอดเดรส เช่นกัน

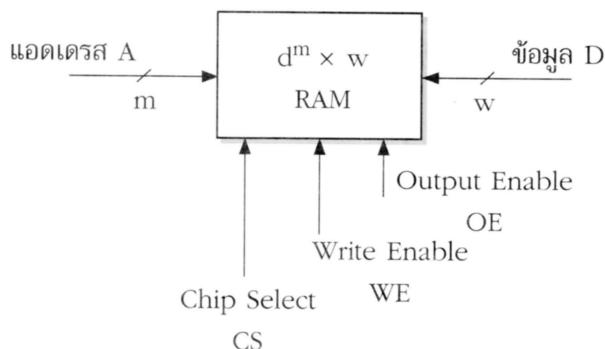
สำหรับ DRAM มักจะมีความจุมากกว่า SRAM จะใช้เป็นหน่วยความจำหลักการเข้าถึงข้อมูลจะซ้ำกับ SRAM ต้องการไฟเลี้ยงเพื่อกระตุน (Refresh) ข้อมูลให้คงอยู่เป็นช่วงๆ

การออกแบบ DRAM ในลักษณะ 2 มิติที่สามารถทำได้เพื่อประยัด จำนวน Address Driver ดังรูปที่ 9.9 จะใช้สัญญาณ RAS (Row Access Strobe) เพื่อถอดรหัสสัญญาณแอดเดรส และสัญญาณ CAS (Column Access Strobe) เพื่อถอดรหัสสัญญาณคอลัมน์



รูปที่ 9.9 RAM 2 มิติ

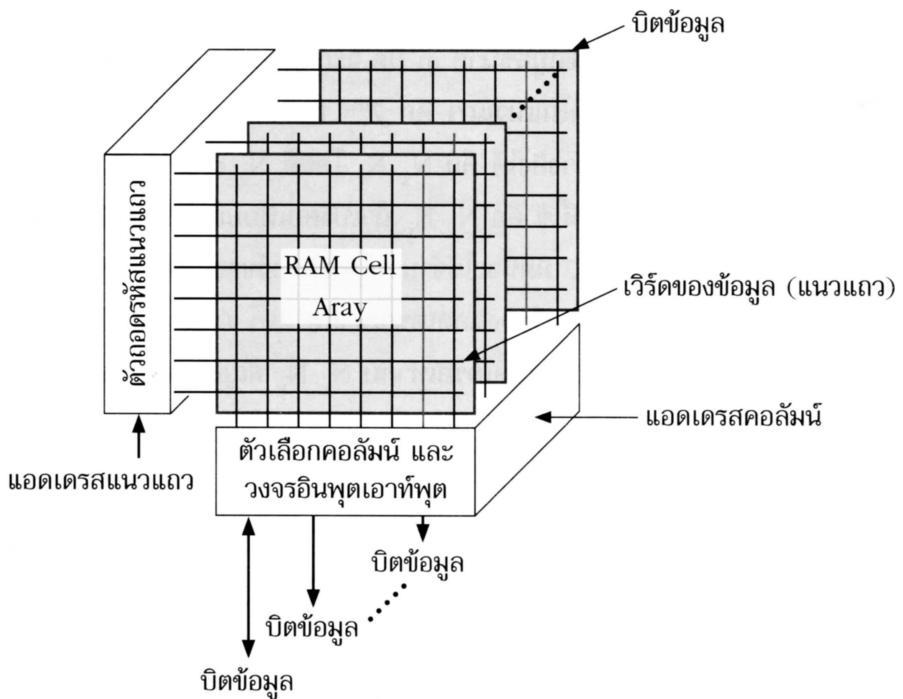
ลักษณะหน่วยความจำถูกจัดในรูปแบบ 2 มิติโดยจัดเป็นแนวแถว (Row) และคอลัมน์ (Column) สำหรับแอดเดรส ข้อมูลขนาด m บิต จะถูกแบ่งเป็น 2 ส่วนขนาด m_x และ m_y บิต โดยจำนวนบิตของแอดเดรสที่ได้ในแนวแถว คือ 2^{m_x} และแนวคอลัมน์ คือ 2^{m_y} ดังนั้น จำนวน Cell ของหน่วยความจำที่มีอยู่อ้างอิงได้ คือ $N_x N_y$ โดยที่ $N_x \leq 2^{m_x}$ และ $N_y \leq 2^{m_y}$ โดยวิธีนี้ จำนวนไดรฟ์เรอร์ของแอดเดรสที่ใช้ คือ $N_x N_y$ ถ้าเปรียบเทียบกับวิธี 1 มิติ สำหรับจำนวน Cell ของหน่วยความจำที่เท่ากัน วิธี 2 มิตินี้จะใช้จำนวนไดรฟ์เรอร์ของแอดเดรสที่น้อยกว่า เท่ากับ จำนวน N ตำแหน่ง จะใช้จำนวนไดรฟ์เรอร์ของแอดเดรสเท่ากับ $2\sqrt{n}$ (เพราะ $N_x = N_y \sqrt{n}$) แต่สำหรับ รูปแบบ 1 มิติ จะใช้ไดรฟ์เรอร์ของแอดเดรสเท่ากับ $N_x N_y$ สัญลักษณ์ของ RAM ดังรูปที่ 9.10



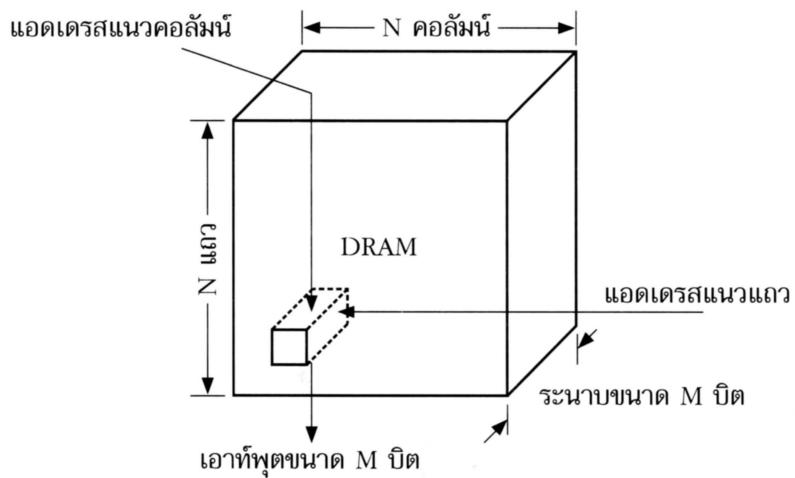
รูปที่ 9.10 ลักษณะทางโครงสร้างของ RAM

ในรูปที่ 9.10 เป็น RAM ที่มีจำนวนแอดเดรส ขนาด 2^m และขนาดเวิร์ด คือ w บิต สัญญาณ CS เป็นสัญญาณ Enable ของ RAM นี้ ส่วนขา WE ใช้สำหรับการเขียนข้อมูลใน RAM และ ขา OE ใช้สำหรับการอ่านข้อมูลจาก RAM ข้อมูลจะออกมาจาก RAM ทางขาข้อมูล D หรือมองอีกรูปแบบได้ดังรูปที่ 9.11 หนึ่งระบบคือตามรูปที่ 9.9 ซึ่งถูกนำร่องมาเรียงกันเพื่อให้สามารถเข้าถึงแบบเวิร์ดได้ และรูปที่ 9.12 แสดงภาพโดยย่อ

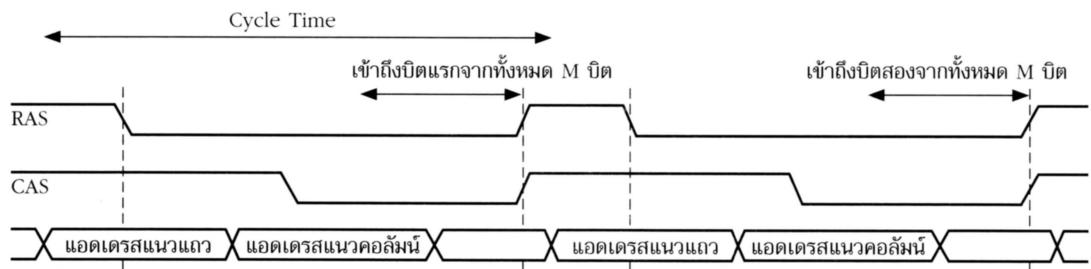
รูปที่ 9.13 แสดงการให้สัญญาณเมื่อต้องการอ่านข้อมูลจาก DRAM โดยจะให้สัญญาณ RAS ก่อน และตามด้วย CAS จากนั้นจึงได้ข้อมูลในเวิร์ดต่อไปก็ให้สัญญาณ RAS ก่อน และตามด้วย CAS จึงอ่านข้อมูลในเวิร์ดได้



รูปที่ 9.11 DRAM ในรูปที่แบบบานง่ายๆ

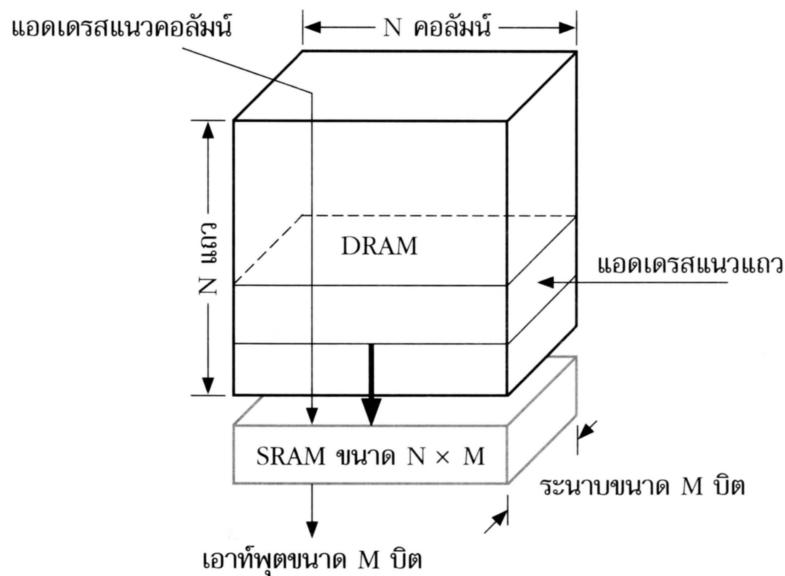


รูปที่ 9.12 กล่องของ DRAM ขนาด w บิต

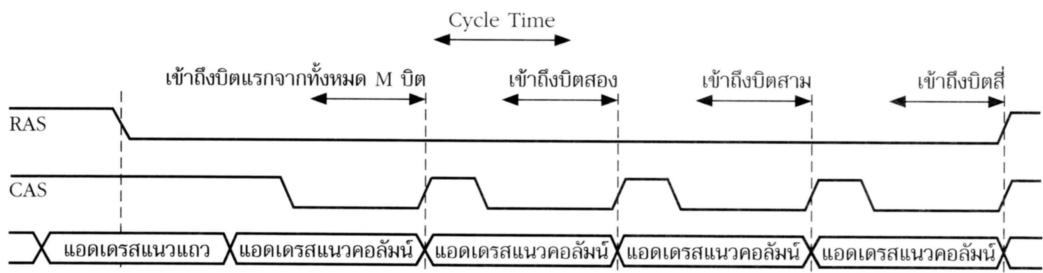


รูปที่ 9.13 การให้สัญญาณของ DRAM

ในการกรณีดังกล่าว หากต้องการปรับปรุงการเข้าถึงข้อมูลในกรณีที่อ่านข้อมูลในແຄາเดียวกันอย่างต่อเนื่อง อาจส่งสัญญาณ RAS ครั้งเดียวจากนั้นตามด้วย CAS แบบนี้เรียกว่า Page Mode DRAM มีโครงสร้างดังรูปที่ 9.14 จะเห็นว่าสามารถอ่านข้อมูลในແຄາเดียวกันทั้งหมด DRAM ออกมากได้ SRAM ขนาด $N \times M$ จากรูปที่ 9.15 จะเห็นว่าเราให้สัญญาณ RAS ครั้งแรกจากนั้นจะให้สัญญาณของ CAS และจะอ่านเวิร์ดต่อไปออกมากอย่างต่อเนื่องได้

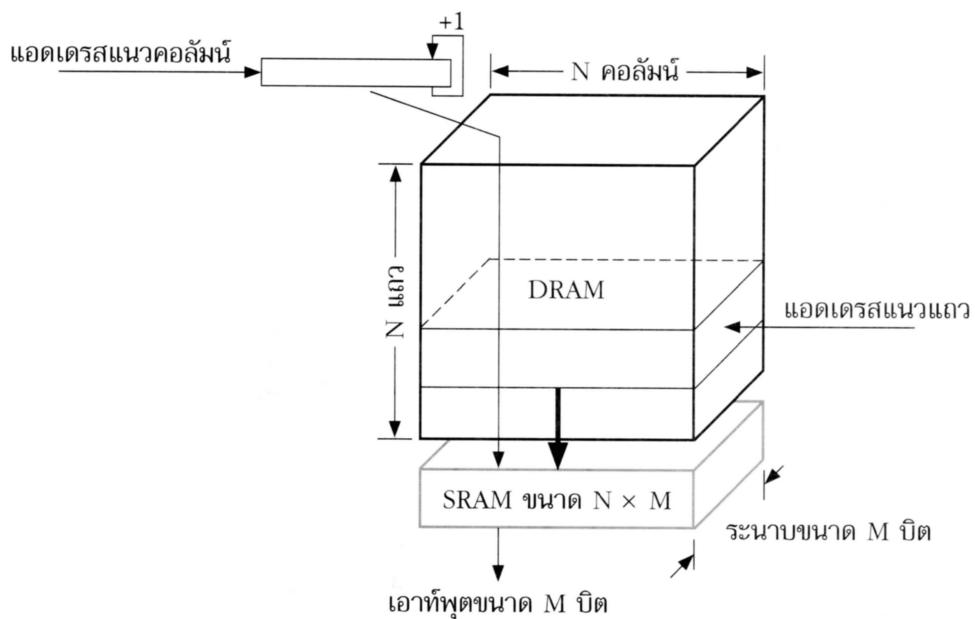


รูปที่ 9.14 โครงสร้าง DRAM แบบ Page Mode

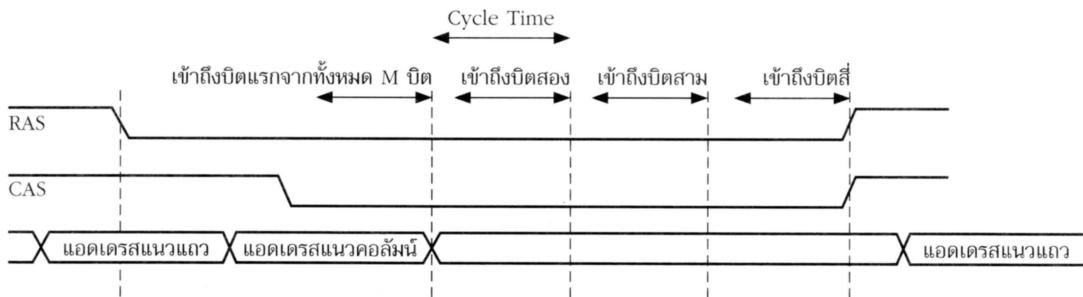


รูปที่ 9.15 การให้สัญญาณสำหรับ DRAM แบบ Page Mode

ในอีกรูปที่แบบหนึ่งเรียกว่า Synchronous DRAM (SDRAM) จะอ่านข้อมูลจาก DRAM ในแถวเดียวกันและคอลัมน์ติดกันอย่างต่อเนื่องดังรูปที่ 9.16 ซึ่งจะมีตัวเพิ่มค่าแอดเดรสของแต่ละคอลัมน์อย่างต่อเนื่อง รูปที่ 9.17 จะพบว่าการป้อนสัญญาณ RAS และ CAS เพียงครั้งแรกครั้งเดียวทำให้ข้อมูลถูกอ่านออกมากได้อย่างต่อเนื่อง



รูปที่ 9.16 โครงสร้าง SDRAM

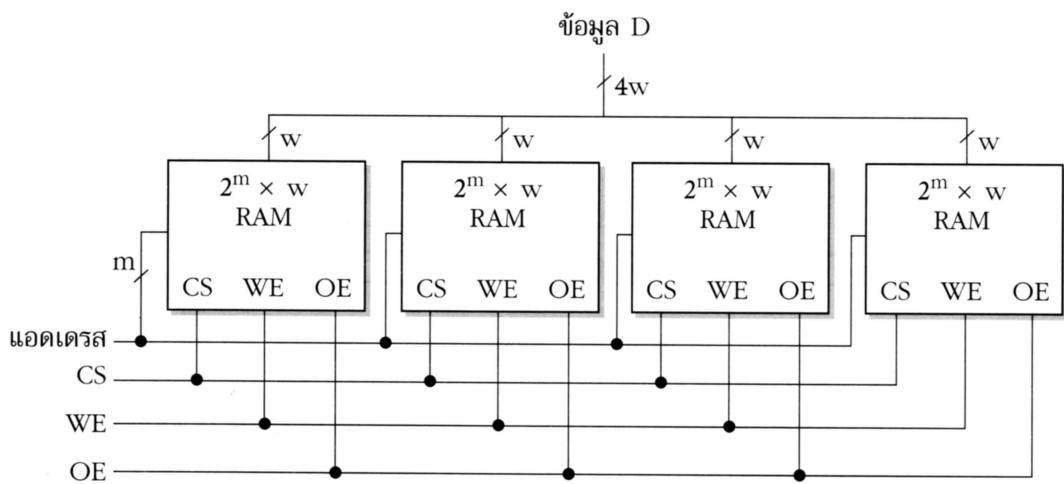


รูปที่ 9.17 การให้สัญญาณของ SDRAM

นอกจากนี้ยังมีเทคโนโลยี DRAM อื่นๆ อีก เช่น DDR SDRAM เป็นการอ่านข้อมูลออก มาด้วยอัตราเป็นสองเท่าโดยอ่านทั้งขาขึ้นและขาลง และยังมี FastPage DRAM, Synch DRAM, DDR-SDRAM, DDR-2 ในปัจจุบันได้พัฒนาไปเป็น DDR-3 เป็นต้น

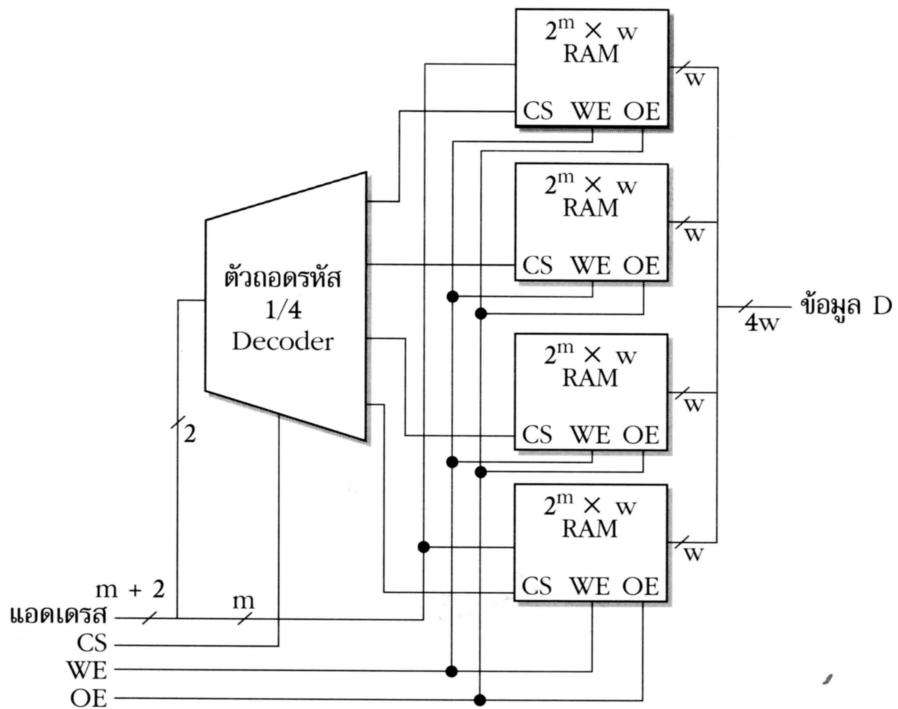
เมื่อแบนด์วิลด์ของซีพียูเพิ่มเป็นสองเท่าแต่ Latency ของหน่วยความจำซึ่งเพียง 1.2 – 1.4 เท่า ดังนั้น จะต้องมีเทคนิคอื่นๆ ช่วยเพื่อให้ Latency ของหน่วยความจำลดลงมากกว่านี้ เช่นการขยายขนาดด้านกว้างเพื่อให้อ่านข้อมูลได้ทีละมากๆ หรือใช้วิธีการ Interleave

ในการขยายขนาดของ RAM สามารถขยายขนาดได้ทั้งแนวกว้างและแนวสูง และยังสามารถสร้าง RAM ที่มีขนาดใหญ่เรียก RAM $M_{N', w'}$ โดยใช้ RAM พื้นฐานจำนวน N ตัว และแต่ละตัว มีความกว้าง w บิต (เรียกว่า RAM $M_{N, w}$) โดยที่ $N' > N$ และ $w' < w$ ในการสร้างจะต้อง เอา RAM $M_{N, w}$ มาต่อเป็นระยะขนาด $p \times q$ ของ โดยที่ $p = \left\lceil \frac{N'}{N} \right\rceil$ และ $q = \left\lceil \frac{w'}{w} \right\rceil$ เช่น $p = 1$, $q = 4$ จะได้ RAM ที่มีเวิร์ดขนาด $4w$ บิต ดังรูปที่ 9.18



รูปที่ 9.18 การขยาย RAM ให้กว้างขึ้น โดย $p=1$ $q=4$

ถ้าต้องการ RAM ขนาดใหญ่ขึ้นโดยที่ความกว้าง w บิตเท่าเดิม และกำหนดให้ $p = 1$, $q = 1$ และใช้ตัวถอดรหัส (Decoder) มาช่วยดังรูปที่ 9.19



รูปที่ 9.19 การขยาย RAM ให้ใหญ่ขึ้นโดยที่ขนาดเวิร์ดเท่ากับ w บิต

ในการทำให้การอ่านและเขียนข้อมูลจาก RAM เร็วขึ้น (หรือทำให้อัตราส่งเร็วขึ้น) มีวิธีทำได้แก่

- ใช้ขนาดเวิร์ดที่ใหญ่ขึ้น คือ $w = S_n$ โดยให้สามารถส่งหรือรับข้อมูลได้ครั้งละ S_n บิต ในการทำเขียนนี้มีข้อเสียคือ ต้องการวงจรพิเศษที่ทำงานได้เร็ว สามารถเข้าถึงข้อมูลได้ S_n บิตได้พร้อมกัน สำหรับการเขียนข้อมูลใน RAM ต้องทำการรวมข้อมูลให้ได้เป็นจำนวน S_n บิตในเวลา t_m พร้อมกัน
- ทำให้สามารถเข้าถึงข้อมูลได้ครั้งละมากกว่า 1 เวิร์ด วิธีการคือ จะแบ่ง RAM ออกเป็น N ช่อง (หรือ Bank) ได้แก่ Bank ที่ $M_0, M_1 \dots M_{n-1}$ และสามารถทำการอ่านและเขียน Bank ทั้งหมด N Bank พร้อมกัน ในเวลาภายใน T_m

ทั้งสองวิธีนี้ต้องการการแปลงการส่งรับข้อมูลจากแบบขนาน (Parallel) มาเป็นแบบลำดับ (Serial) หรือจากแบบลำดับเป็นแบบขนาน รวมทั้งต้องการวงจรฮาร์ดแวร์พิเศษ ปกติแล้วข้อมูลขนาด S เวิร์ดที่ติดกันซึ่งพิจฉาณอ่านหรือเขียนข้อมูลโดยอ่านครั้งละ 1 เวิร์ด และเข้าต้องถึงข้อมูลจำนวน S ครั้ง ดังนั้น การใช้เทคนิคที่เรียกว่า Memory Interleaving ทำให้ข้อมูลขนาด S เวิร์ด เหล่านี้กระจายออกจากกันและสามารถเข้าถึงได้พร้อมกันได้ โดยกระจายข้อมูลออกไป S Bank ตามวิธีของการ Interleave คือ จะกำหนดแอดเดรส A_i ให้กับหน่วยความจำ Bank ที่ M_j เมื่อ $j = i \bmod S$ นั่นคือ แต่ละ Bank จะหมายถึงแอดเดรสดังนี้

$$M_0 : A_0 \ A_s \ A_{2s} \dots$$

$$M_1 : A_1 \ A_{s+1} \ A_{2s+1} \dots$$

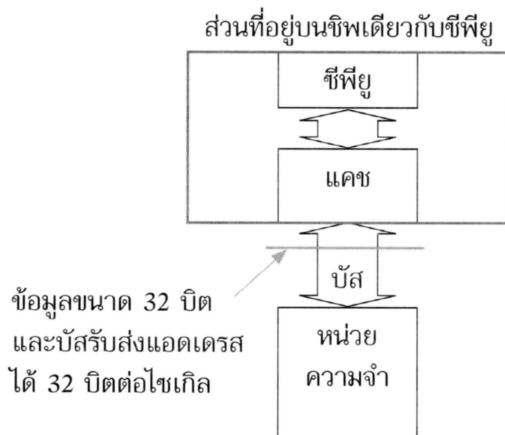
被称为 S-Way Interleave

ในการประมาณว่าค่า Interleave Factor S ว่าควรมีค่าเป็นเท่าไร ขึ้นอยู่กับว่าจำนวนเวิร์ดที่ต้องการเข้าถึงในหนึ่งไฟเกิลเมเทอร์ และขึ้นกับเวลาในการเข้าถึงของหน่วยความจำของแต่ละ Bank ว่าเป็นเท่าไร

ในการอ่านข้อมูลที่มีพื้นที่ติดกันจาก Bank เริ่มต้นจะส่งแอดเดรสออกไปก่อนไปที่ Bank แรก ที่พบข้อมูลนั้น จนกว่าจะทำการอ่านข้อมูลจาก Bank นั้นขึ้นมา การอ่านข้อมูลจาก Bank นี้จะใช้เวลาขึ้นกับการเข้าถึงหน่วยความจำ การอ่านจากแอดเดรสต่อไป ก็จะส่งแอดเดรสไปยัง Bank ถัดไปเพื่อทำการเข้าถึงข้อมูลใน Bank ถัดไป ทำเช่นนี้ไปเรื่อยๆ ถ้าเวลาในการเข้าถึงหน่วยความจำ

เป็น s ดังนั้น ควรจะมีจำนวน Bank อย่างน้อยเท่ากับ s เท่านั้น เพื่อว่าเมื่อส่งแอดเดรสไปถึง Bank ที่ s แล้ว ข้อมูลจาก Bank แรกที่อ่านໄວ่ตอนแรกจะมาถึงพอดี ทำให้ชีพีย์ไม่ต้องหายุดรอข้อมูล สังเกตว่าการเข้าถึงหน่วยความจำแบบนี้เป็นลักษณะการ Interleave นั่นเอง นอกจากนี้ ควรจะพิจารณาด้วยว่าการกระจายข้อมูลในแต่ละ Bank ควรเป็นไปอย่างเหมาะสม เพราะถ้าข้อมูลที่ต้องการอยู่ใน Bank เดียวกันทั้งหมดก็จะเกิดการชนกันที่ Bank นั้นบ่อยๆ ทำให้ไม่สามารถทำการ Interleave การเข้าถึงข้อมูลได้และทำให้ต้องรอให้การเข้าถึงข้อมูลแต่ละครั้งใน Bank นั้นเสร็จสิ้นก่อน

พิจารณาการเปรียบเทียบวิธีการปรับปรุงรูปแบบการจัดหน่วยความจำดังนี้ สมมติให้โครงสร้างพื้นฐานเป็นดังรูปที่ 9.20 ขนาดของหน่วยความจำกว้าง 1 เวิร์ดและบัสกว้าง 1 เวิร์ด เท่านั้น สมมติให้ใช้เวลา 1 นาทีในการส่งแอดเดรสและ Cycle Time ของ DRAM เป็น 25 นาทีในการเข้าถึงข้อมูล และใช้เวลา 1 นาทีในการส่งเวิร์ดกลับมา ให้พิจารณา หาแบบวิดส์ของบัสระหว่างหน่วยความจำและแคช (จำนวนไชเกิลที่ใช้ในการเข้าถึงและการโอนข้อมูลไปยังแคชต่อไชเกิล)



รูปที่ 9.20 โครงสร้างหน่วยความจำตัวอย่าง

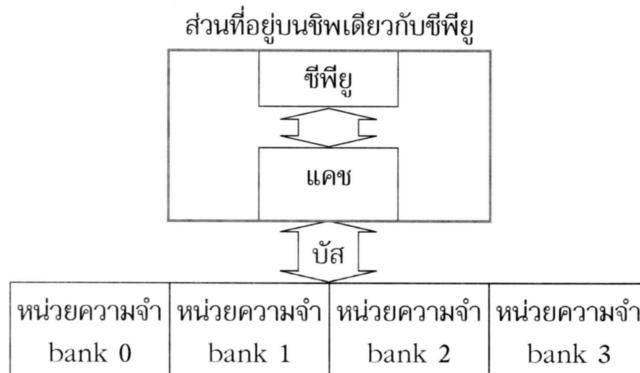
สมมติให้ขนาดบล็อก = 1 เวิร์ด ถ้าเกิดการ Miss ในแคช จะใช้เวลาที่ไชเกิลในการนำข้อมูลเข้ามายังแคช

ในกรณีรูปที่ 9.20 จะใช้เวลาเท่ากับ $1 + 25 + 1 = 27$ ไชเกิล และจะทำให้ได้แบบวิดส์ (จำนวนไบต์ที่รับส่งต่อหนึ่งไชเกิล) เท่ากับ $4/27 = 0.148$ (กรณีคิด 1 เวิร์ด = 32 บิตหรือ 4 ไบต์)

ในกรณีรูปที่ 9.20 ขนาดบล็อก = 4 เวิร์ดจะใช้เวลาเท่ากับ $1 + 25 \times 4 + 1 = 102$ นาที ก็จะทำให้ได้แบบตัวต่อตัว เท่ากับ $(4 + 4)/102 = 0.157$

ในกรณีรูปที่ 9.20 ใช้ขนาดบล็อก = 4 เวิร์ด และใช้ Fast Page Mode DRAM จะใช้เวลาเท่ากับ $1 + 25 \times 1 + 3 \times 8 + 1 = 51$ นาที ก็จะทำให้ได้แบบตัวต่อตัว เท่ากับ $(4 + 4)/51 = 0.314$

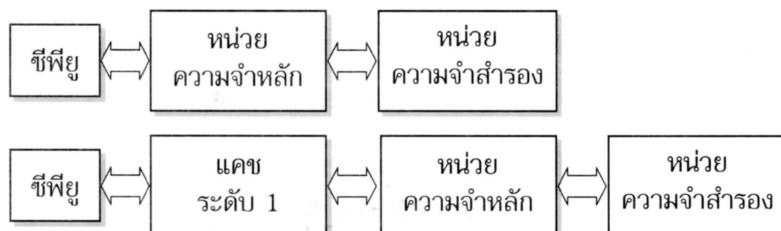
ในกรณีรูปที่ 9.21 ขนาดบล็อก = 4 เวิร์ด ใช้ Fast Page Mode DRAM จะใช้เวลาเท่ากับ $1 + 25 + 3 + 1 = 30$ นาที ก็จะทำให้ได้แบบตัวต่อตัว $(4 + 4)/30 = 0.533$



รูปที่ 9.21 ตัวอย่างการแบ่งหน่วยความจำเป็นหลายๆ Bank

9.4 ระบบหน่วยความจำ (Memory System)

ในหน่วยความจำของคอมพิวเตอร์โดยทั่วไปมีการแบ่งระดับตั้งเรื่องรูปที่ 9.22



รูปที่ 9.22 ระดับของหน่วยความจำในคอมพิวเตอร์

ซึ่งจะเพิ่มต่อไปเรื่อยๆ ความจำที่เข้าถึงได้เร็วที่สุดได้แก่ จีโนมิกส์ เทอร์ จำนวนนี้จะเพิ่มตามลำดับ ในการอ้างถึงข้อมูลในหน่วยความจำซึ่งต้นทางข้อมูลในหน่วยความจำที่ใกล้ก่อนเสมอ

คุณลักษณะของหน่วยความจำในระดับ i คือ

1. $C_i > C_{i+1}$ โดยที่ C_i คือ ราคาต่อหน่วยของระดับที่ i
2. $t_{A_i} < t_{A_{i+1}}$ โดย t_{A_i} เป็น เวลาที่ใช้ในการเข้าถึงหน่วยความจำในระดับที่ i
3. $S_i < S_{i+1}$ โดยที่ S_i คือ ความจุของหน่วยความจำในระดับที่ i

ในระดับใกล้จากซีพียูมากที่สุด ขนาดหรือความจุของหน่วยความจำจะมีมากที่สุด ราคาก็จะถูกกลง แต่จะหน่วยความจำเหล่านี้จะใช้เวลาในการเข้าถึงช้ากว่าหน่วยความจำที่อยู่ใกล้กับซีพียู ดังนั้น ให้ ฮาร์ดดิสก์เป็นระดับที่ $i + 1$ และระดับที่ i คือ RAM ใน การค้นหาในหน่วยความจำ ถ้าข้อมูลไม่อยู่ ในแคชก็จะไปหาข้อมูลนั้นในหน่วยความจำและหน่วยความจำสำรอง เช่น ฮาร์ดดิสก์ตามลำดับถ้า ข้อมูลถูกพบในหน่วยความจำสำรอง ข้อมูลก็จะถูกส่งต่อขึ้นไปเรื่อยๆ จากระดับบนลงล่างจนข้อมูลถูกส่งมาถึงซีพียู

ในการใช้หน่วยความจำที่มีประสิทธิภาพ จะต้องทำให้ระบบใช้เวลาที่เข้าไปถึงข้อมูลน้อยที่สุด นั่นคือ ต้องพยายามหาวิธีที่จะทำให้ข้อมูลที่ต้องการใช้งานมาอยู่ในหน่วยความจำที่ใกล้กับซีพียูมากที่สุด เมื่อซีพียูต้องการใช้ข้อมูลนั้น ก็จะได้ไปเอาข้อมูลได้ง่ายและเร็ว

หน่วยความจำเสมือน (Virtual Memory)

การใช้หน่วยความจำเสมือนมีจุดมุ่งหมายเพื่อให้ซีพียูสามารถทำงานกับโปรแกรมที่มีขนาดใหญ่กว่าหน่วยความจำหลักที่มีอยู่ได้ โดยผู้ใช้งานจะไม่รู้สึกว่าได้ใช้หน่วยความจำเสมือนอยู่เลย และไม่มีความยุ่งยากให้เขียนโปรแกรมแต่อย่างไร โปรแกรมเมอร์ยังสามารถเขียนโปรแกรมได้ตามปกติ ประโยชน์ของการมีหน่วยความจำเสมือนมีหลายอย่าง ได้แก่

1. เพื่อให้เกิดการใช้ข้อมูลร่วมกันในหน่วยความจำในระบบแบบผู้ใช้หลายคน (Multi-users)
2. เพื่อเป็นการจัดการการใช้หน่วยความจำให้กับผู้ใช้อย่างมีประสิทธิภาพ ผู้ใช้และนัก โปรแกรมเมอร์ไม่ต้องกังวลถึงขั้นตอนการจัดการตั้งกล่าว

3. เพื่อทำให้ขนาดของโปรแกรมที่จะทำงานในเครื่อง ไม่เขียนกับขนาดที่แท้จริงของหน่วยความจำที่มีอยู่
4. เพื่อทำให้ใช้เวลาในการเข้าถึงหน่วยความจำน้อย และลดค่าใช้จ่ายในการซื้อหน่วยความจำ ทำให้ไม่จำเป็นต้องซื้อหน่วยความจำขนาดใหญ่มากๆ เพื่อทำงานกับโปรแกรมขนาดใหญ่ขึ้น

ในการใช้หน่วยความจำเสมือน จะต้องกำหนดแอดเดรสของหน่วยความจำเสมือนที่จะไปเอาข้อมูลที่เรียกว่า Logical Address หรือแอดเดรสเสมือน (Virtual Address) และผู้ใช้จะรู้จักแต่แอดเดรสเสมือนซึ่งแท้จริงแล้วจะเป็นตำแหน่งในหน่วยความจำจริง หรืออาจจะเป็นตำแหน่งใดๆ ในหน่วยความจำสำรอง โดยผู้ใช้จะไม่ต้องรับรู้การหาตำแหน่งข้อมูลที่แท้จริงจากแอดเดรสเสมือนนี้

Locality of Reference

การอ้างถึง (Reference) จะหมายถึงการอ้างถึงข้อมูลในหน่วยความจำ โดยปกติแล้วข้อมูลที่ต้องการใช้งานในโปรแกรมหนึ่งๆ มักอยู่ใกล้กัน หรือมีแอดเดรสติดกัน คุณสมบัตินี้เรียกว่า Locality ลักษณะของ Locality นี้จะช่วยในการคำนวณว่าข้อมูลที่ต้องการใช้ในเวลาต่อไปอยู่ที่ตำแหน่งใดตัวอย่างเช่นพิจารณาการทำงานของโปรแกรม ลักษณะของโปรแกรมโดยมากจะทำงานตามลำดับตั้งแต่ต้นโปรแกรมไปห้ายังโปรแกรม ซึ่งแอดเดรสของแต่ละคำสั่งจะเป็นแอดเดรสที่ต่อเนื่องกัน ความสัมพันธ์เช่นนี้เรียกว่าเป็นความสัมพันธ์ในเชิงพื้นที่ ที่เรียกว่า Spatial Locality เพราะแต่ละคำสั่งที่จะถูกอ่านเข้ามาทำงานตามลำดับ ซึ่งโดยทั่วไปจะอยู่ในหน่วยความจำในพื้นที่ต่อเนื่องกัน

ในตัวอย่างของโปรแกรมที่มีลูป และภายในลูปจะมีกลุ่มของคำสั่งที่ถูกทำงานเข้ากันในช่วงเวลาหนึ่ง ดังนั้น คำสั่งเหล่านี้ที่ทำซ้ำๆ กันจะสัมพันธ์กันในแนวของเวลา หรือเรียกว่ามี Locality เวลา (Temporal Locality) ณ เวลาหนึ่งๆ กลุ่มของคำสั่งเหล่านั้นจะต้องอยู่ในหน่วยความจำ เพราะเป็นการทำงานซ้ำๆ ของลูปนั้นๆ ในช่วงเวลาจะยังคงอยู่

Working Set คือ เขตของแอดเดรสของล็อกข้อมูลที่ต้องการใช้ภายในช่วงเวลาหนึ่ง ในการทำงานของโปรแกรม จะต้องมีการเก็บ Working Set นี้ไว้ในหน่วยความจำระดับล่างที่ใกล้กับชีพยูที่สุด เพื่อให้ชีพยูสามารถเข้าถึงได้เร็วเนื่องจากใช้มีงานข้อมูลเหล่านี้บ่อย ปกติแล้ว Working Set จะเปลี่ยนไปตามช่วงเวลาของการทำงานของโปรแกรม แต่การเปลี่ยนแปลงภายใน Working Set จะเป็นไปอย่างช้า ๆ

ตัวอย่างในโค้ดต่อไปนี้

```
for (i=0; i < 3; i++)
    for (j=0; j < 100; j++)
        a[i][j] = b[j][0] * b[j+1][0]
```

พิจารณาการใช้ข้อมูลในโปรแกรม ถ้าการจัดเก็บอะเรย์เป็นแบบแนวๆ ในหน่วยความจำ กล่าวคือ จะจัดเก็บเรียงลำดับติดกันดังนี้ $a[0][0], a[0][1], a[0][2], \dots, a[0][99], a[1][0], a[1][1], \dots$ จะเห็นว่าการอ้างถึงข้อมูลในอะเรย์ a ตามลักษณะในโปรแกรมนี้จะได้ประโยชน์จาก Locality เหิงพื้นที่ เพราะเป็นการอ้างถึงข้อมูลที่ติดกันเรียงต่อกันไป แต่สำหรับอะเรย์ b จะได้ประโยชน์จาก Locality เหิงเวลาถึงแม้ว่าจะไม่ได้ทำการอ้างถึงข้อมูลใน b ที่เรียงกันตามพื้นที่ในหน่วยความจำที่ติดกัน (เพราะการอ้างถึง b เป็นแบบ $b[0][0], b[1][0], b[2][0], \dots$) แต่ว่าการอ้างถึงค่าใน b ในแต่ละรอบของ i เป็นแบบเดียวกันดังนั้นจึงเป็นประโยชน์จาก Locality เหิงเวลาเนื่องจากในช่วงเวลาที่ใกล้กันได้มีการใช้ข้อมูลกลุ่มเดียวกัน

ราคาและประสิทธิภาพ

จุดมุ่งหมายอันหนึ่งในการมีหน่วยความจำหลายระดับเท่านี้ คือ ต้องการให้โปรแกรมทำงานได้เร็วโดยรวม ซึ่งความเร็วในการทำงานนี้ได้รวมไปถึงเวลาที่ต้องไปอ่านเขียนข้อมูลในหน่วยความจำด้วย ในส่วนนี้เราต้องการให้เวลาเฉลี่ยในการเข้าถึงหน่วยความจำน้อยที่สุดที่เป็นไปได้ นั้นคือ เท่ากับเวลาที่ใช้ในการเข้าถึงหน่วยจำระดับที่ใกล้ชิดกันที่สุด จึงจะดีกว่าเร็วที่สุด แต่ราคารวมของเครื่องนั้นจะต้องมีราคาถูก เสมือนว่าเครื่องนั้นใช้เพียงหน่วยความจำที่ถูกที่สุดคือใช้หน่วยความจำที่อยู่ใกล้ชิดกันนั่นเอง

ปัจจัยที่มีผลกับประสิทธิภาพ ได้แก่

1. สถิติการอ้างถึงแอดเดรสต่างๆ ของข้อมูลและแอดเดรสของคำสั่ง ในเวลาที่ใช้พิจารณา กับโปรแกรมว่าโปรแกรมนั้นต้องการเข้าถึงแอดเดรสใดบ้าง
2. เวลาในการเข้าถึงข้อมูลในหน่วยความจำในแต่ละระดับ หรือ t_{A_i}
3. ความจุของหน่วยความจำแต่ละระดับ S_i
4. ขนาดของบล็อกของข้อมูลที่ที่อ่านเข้ามา (S_{P_i}) ในหน่วยความจำแต่ละระดับที่มีความจุ ทั้งหมดเป็น S_i

5. อัลกอริธึมในการตัดสินใจว่าจะเอาบล็อกข้อมูลใดออกไปเมื่อต้องการอ่านข้อมูลชุดใหม่เข้ามาแทนที่บล็อกในหน่วยความจำ (Replacement Policy)

ส่วนราคาของหน่วยความจำต่อหน่วยบิตสำหรับระบบที่มีหน่วยความจำมากกว่า 2 ระดับขึ้นไปหาได้จาก

$$C = \frac{C_1 S_1 + C_2 S_2}{S_1 + S_2}$$

โดยที่ C_i คือ ราคาต่อบิตของหน่วยความจำในระดับที่ i และ S_i คือขนาดของหน่วยความจำในระดับที่ i (หน่วยเป็นบิต)

การ Hit หมายถึง การพบข้อมูลที่ต้องการในหน่วยความจำที่ใกล้ชีพยูที่สุด เช่น แคล

- Hit Rate หมายถึง อัตราส่วนของจำนวนครั้งการค้นพบในหน่วยความจำที่ใกล้ชีพยูที่สุดต่อจำนวนครั้งในการอ้างถึงข้อมูลทั้งหมด
- Hit Time หมายถึง เวลาในการเข้าถึงข้อมูลที่ต้องการในหน่วยความจำที่ค้นพบคำนวนได้จาก

เวลาเข้าถึงหน่วยความจำ (Memory Access Time) + เวลาที่ใช้ในการตัดสินว่าค้นพบหรือไม่ (Hit Time)

Miss หมายถึง กรณีที่ข้อมูลที่ต้องการไม่อยู่ในระดับที่ใกล้ชีพยู กล่าวคือต้องไปเอาข้อมูลนั้นมาจากหน่วยความจำระดับล่าง

- Miss Rate = 1 - (Hit Rate)
- Miss Penalty เป็นเวลาที่ใช้ในการนำบล็อกข้อมูลที่ต้องการมายังหน่วยความจำที่ใกล้ชีพยูในกรณีที่ข้อมูลนั้นไม่อยู่ในหน่วยความจำระดับใกล้ชีพยู และรวมเวลาในการแทนที่บล็อกในระดับนั้น ๆ (Block Replacement Time)

โดยทั่วไปแล้ว Hit Time จะมีค่าน้อยกว่า Miss Penalty มากๆ

กำหนดให้เกณฑ์ของ Hit Ratio (H) คือสัดส่วนของจำนวนครั้งของการค้นพบข้อมูลที่ต้องการในหน่วยความจำที่ใกล้ชีพยูที่สุดเทียบกับจำนวนครั้งในการอ้างถึงข้อมูลทั้งหมด

ในทางตรงกันข้าม Miss Ratio คือ $1-H$ ดังนั้น สำหรับเครื่องที่มีหน่วยความจำ 2 ระดับ จะหาได้จาก

$$H = \frac{N_1}{N_1 + N_2}$$

โดยที่ N_1 คือ จำนวนครั้งในการอ้างถึงแอดเดรสที่อยู่ในหน่วยความจำระดับแรกที่สุดใน M_1 และ N_2 คือ จำนวนครั้งในการอ้างถึงข้อมูลทั้งหมด เมื่อข้อมูลที่ต้องการอยู่ในหน่วยความจำ M_2 (แต่ไม่อยู่ใน M_1)

เราแบ่งการ Miss ของแคชออกเป็น 3 แบบ ดังนี้

- **Compulsory Miss** เกิดจากเมื่อเริ่มต้นรันโปรแกรมนั้นข้อมูลที่ต้องใช้ยังไม่อยู่ในแคช เป็นการ Miss ที่จำเป็นเนื่องจากโปรแกรมเพิ่มเริ่มทำงาน
- **Conflict (Collision) Miss** เกิดจากการที่ Replacement Policy นั้นกำหนดให้ ข้อมูลหลายๆ บล็อกไปจัดเก็บยังตำแหน่งเดียวกัน หรือเรียกว่าเป็นการชนกันของ ตำแหน่ง ซึ่งอาจจะแก้ปัญหาได้โดยการเพิ่มขนาดของแคช หรือเพิ่มตีกรีของ Associativity ดังจะกล่าวต่อไป
- **Capacity Miss** เนื่องจากขนาดของแคชเล็กเกินไปทำให้ไม่สามารถบรรจุข้อมูลที่ เป็น Working Set ทั้งหมดของโปรแกรมนั้นๆ ได้

เวลาที่ใช้ในการเข้าถึงข้อมูลโดยเฉลี่ย (Average Access Time) จะคิดได้จาก $t_A = Ht_{A_1} + (1 - H)t_{A_2}$ โดยที่ t_{A_1} และ t_{A_2} คือเวลาของการเข้าถึงข้อมูลใน M_1 และ M_2 ตาม ลำดับ ซึ่ง $t_{A_2} = t_{A_1} + t_B$ โดยที่ t_B คือเวลาในการนำข้อมูลในหน่วยความจำ M_2 มาไว้ในหน่วยความจำ M_1 และจากนั้นเข้าไปเอาข้อมูลในหน่วยความจำ M_1 ซึ่งใช้เวลา t_{A_1}

ดังนั้น เมื่อแทนค่าเข้าไปเราจะได้ $t_A = Ht_{A_1} + (1 - H)t_B$ ให้ Access Time Ratio $r = \frac{t_{A_2}}{t_{A_1}}$ คือความแตกต่างระหว่างเวลาในการเข้าถึงข้อมูลที่อยู่ในหน่วยความจำ M_1 และหน่วยความจำ M_2

ให้ Access Efficiency $e = \frac{t_{A_1}}{t_A}$ คือ ความแตกต่างระหว่างเวลาที่ใช้ในการเข้าถึงข้อมูลโดยเฉลี่ยและเวลาที่ใช้ในการเข้าถึงข้อมูลในหน่วยความจำที่เร็วที่สุด

$$e = \frac{t_{A_1}}{t_A}$$

$$\begin{aligned}
 &= \frac{t_{A_1}}{Ht_{A_1} + (1 - H)t_{A_2}} \\
 &= \frac{1}{\frac{H}{t_{A_1}} + \frac{1 - H}{t_{A_2}}} \\
 &= \frac{1}{H + (1 - H)\frac{t_{A_2}}{t_{A_1}}} \\
 &= \frac{1}{H + (1 - H)r} \\
 &= \frac{1}{r + (1 - r)H} \\
 \text{ดังนั้น} \quad e &= \frac{1}{r + (1 - r)H}
 \end{aligned}$$

พิจารณาประโยชน์ของการใช้พื้นที่ของหน่วยความจำที่ใช้วัดได้จาก $u = \frac{S_u}{S}$ โดยที่ S คือ พื้นที่ของหน่วยความจำทั้งหมดที่มีอยู่ และ S_u คือ พื้นที่ของหน่วยความจำที่ได้ใช้จริง

สาเหตุของพื้นที่ในหน่วยความจำที่ไม่มีประโยชน์มีหลายอย่าง เช่น

1. พื้นที่ว่างเนื่องจากมีการแบ่งหน่วยความจำเป็นบล็อก และขนาดของข้อมูลที่นำมาใส่ไม่เต็มบล็อก (Internal Fragment)
2. พื้นที่ที่เก็บข้อมูลอยู่ แต่ข้อมูลที่ถูกเก็บไม่ได้มีการนำไปใช้ หรือไม่ได้อ้างถึง
3. พื้นที่ที่เก็บข้อมูลที่ไม่สามารถเข้าถึงได้ เช่น ไฟล์ที่ถูกซ่อนในเครื่องคอมพิวเตอร์ หรือไฟล์ที่ถูกจำกัดการเข้าถึง

ในการออกแบบระบบจัดการหน่วยความจำต้องการให้มีหลักการของ Transparency นั่นคือ ผู้ใช้จะไม่รับรู้ความยุ่งยากในการจัดการหน่วยความจำเลย ไม่รับรู้ถึงว่ามีการใช้หน่วยความจำเมื่อใด หรือไม่รับรู้ในการโอนย้ายข้อมูลไปมาระหว่างส่วนต่างๆ ของเครื่อง ผู้ใช้ไม่รู้รายละเอียดและไม่กังวลกับการเปลี่ยนแปลงแอดเดรสของข้อมูลในหน่วยความจำจริง และไม่ว่าบล็อกใดจะถูกจัดเก็บไว้ที่ใด ผู้ใช้สามารถเข้าถึงได้โดยไม่ต้องคำนึงถึงรายละเอียดทางกายภาพของหน่วยความจำ

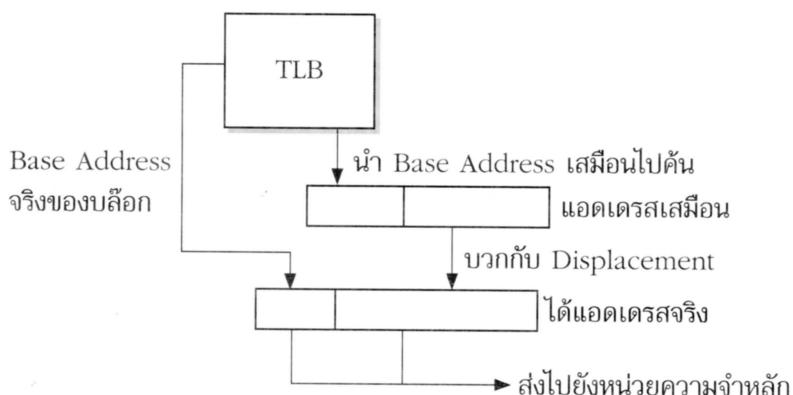
การแปลงแอดเดรส (Address Translation)

การแปลงค่าแอดเดรสของข้อมูลในหน่วยความจำเสมือนไปเป็นแอดเดรสของข้อมูลในหน่วยความจำจริง ในที่นี้เรารายกแอดเดรสของข้อมูลในหน่วยความจำเสมือนว่าเป็นแอดเดรสเสมือน และเรียกแอดเดรสของข้อมูลในหน่วยความจำจริงว่า Real Address (หรือ Physical Address) ปกติแล้วการแปลงแอดเดรสนี้จะเป็นหน้าที่ของซอฟต์แวร์ระบบ เช่น อาจเป็นหน้าที่ของระบบปฏิบัติการตัวแปลงภาษา Linker Loader ตัวจัดการหน่วยความจำ เวอร์ชัลเมมชีน เป็นต้น

เราเรียกการแปลงแอดเดรสที่เกิดขึ้นในขั้นตอนการคอมไพล์เรียกว่า Static Translation และเรียกการแปลงแอดเดรสที่เกิดขึ้นระหว่างการทำงานของโปรแกรมว่าเป็น Dynamic Translation และเรียกหน่วยที่ทำหน้าที่จัดการการแปลงแอดเดรสนี้ว่า หน่วยจัดการหน่วยความจำ (Memory Management Unit/MMU)

เมื่อหน่วยความจำแบ่งเป็นบล็อกในการอ้างถึงข้อมูล จะทำการคำนวณ Effective Address ของข้อมูลโดยใช้รูปแบบคือ แอดเดรสเริ่มต้น (Base Address) + ระยะห่าง (Displacement หรือ Offset) โดยที่ Base Address คือแอดเดรสเริ่มต้นของบล็อกที่ข้อมูลอยู่และ Displacement คือ ระยะห่างของข้อมูลเทียบจากจุดเริ่มต้นภายในบล็อกนั้น

ในการแปลงแอดเดรสของหน่วยความจำเสมือนนั้น จะต้องมีตารางการ Map เก็บ Base Address ของบล็อกต่างๆ ที่ใช้อยู่ขณะนั้น มี Translation Lookaside Buffer (TLB) จะเป็นแคชพิเศษที่เก็บ Base Address ของบล็อกข้อมูลที่ใช้บ่อย โครงสร้างของการแปลงแอดเดรสเป็นดังรูปที่ 9.23



รูปที่ 9.23 ขั้นตอนการทำแปลงแอดเดรส

ในรูปที่ 9.23 แอดเดรสเสมือนจะแบ่งเป็น 2 ส่วน คือ Base Address และ Displacement นั้น ส่วนของ Base Address จะถูกนำเข้าไปค้นหาใน TLB ซึ่งค้นหาแอดเดรสจริงของบล็อกข้อมูลนั้น ในส่วนของ Displacement จะถูกมาบวกกับแอดเดรสจริงเป็นแอดเดรสที่เก็บข้อมูลจริงในหน่วยความจำภายในบล็อก แต่ถ้าไม่สามารถหา Base Address จริงใน TLB ได้ก็จะเข้าไปหาในตารางการ Map หน่วยความจำและนำเอา Base Address จริงของข้อมูลนั้นมาเก็บไว้ใน TLB

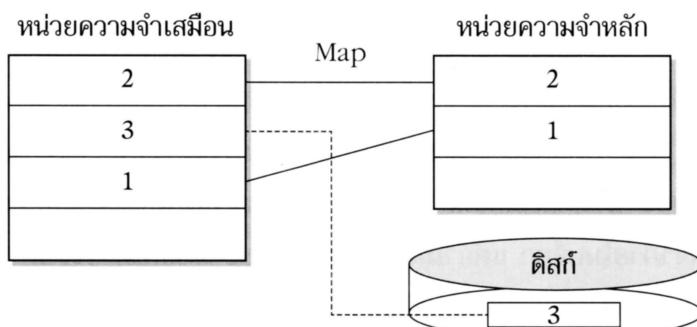
บางครั้งบล็อกของข้อมูลในหน่วยความจำอาจมีขนาดเท่ากันทุกบล็อกหรือที่เรียกว่าเป็น Page จากรูปที่ 9.23 โดยทั่วไปแล้ว 1 แผง (Entry) ของ TLB จะเก็บแอดเดรสเสมือนของ Page จะเก็บแอดเดรสจริงของ Page เก็บ Id ของเจ้าของ Page นั้นและเก็บบิตต่างๆ ที่แสดงสถานะของ Page นั้น

ถ้าแบ่งหน่วยความจำเป็นส่วนๆ โดยที่แต่ละส่วนมีขนาดไม่เท่ากัน จะเรียกว่าเป็นการแบ่งเป็น Segment พิจารณาตัวอย่าง เช่น เวลาเขียนโปรแกรมในภาษาอาสน์แซมบลี โปรแกรมจะถูกแบ่งออกเป็น Segment เช่น Data Segment, Code Segment เป็นต้น ในการเข้าถึงข้อมูลใน Segment จะเป็นลักษณะเดียวกับแบบ Page ซึ่งจะใช้รูปที่แบบที่ประกอบด้วย Segment Base Address และ Displacement ถ้าข้อมูลที่ต้องการไม่อยู่ในหน่วยความจำ ก็จะนำ Segment ที่อยู่ของข้อมูลนั้นเข้ามาทั้งหมด ตารางที่เก็บการ Map ระหว่างแอดเดรสเสมือนกับแอดเดรสจริงนี้ เรียกว่าเป็น Segment Table (ซึ่งเรียกว่า Page Table ในระบบ Page) ในหนึ่ง Entry ของตารางนี้จะประกอบด้วยแอดเดรสจริงของ Segment ที่ข้อมูลนั้นๆ อยู่ ถ้า Segment นั้นอยู่ในหน่วยความจำจริง นอกจากนี้ จะมีบิตพิเศษต่างๆ ที่บอกคุณสมบัติของ Segment เช่น Presence Bit

บอกว่า Segment นั้นอยู่ในหน่วยความจำจริงหรือไม่ Copy Bit ใช้บอกว่า Segment นั้นได้ถูกคัดลอกไปใช้หลายที่หรือไม่รวมทั้งยังมีการบอกรึ่งจำนวนเริร์ด ที่มีใน Segment นั้น

ประโยชน์ของการแบ่งแบบ Segment ได้แก่ การใช้ Segment ร่วมกันระหว่างผู้ใช้หลายคน หรือโปรแกรมหลายโปรแกรมในระบบผู้ใช้หลายคนโดยสามารถแบ่ง Segment ได้เป็นหลายประเภท ขึ้นกับการลักษณะการใช้งาน และขนาดของ Segment ก็ยืดหยุ่นได้ เป็นต้น

การแบ่งหน่วยความจำเป็น Page หรือบล็อกที่มีขนาดเท่ากันทั้งหมด โดย Page จะถูก Map เข้ากับเฟรมแต่ละเฟรมของหน่วยความจำจริงดังรูปที่ 9.24



รูปที่ 9.24 การ Map ระหว่างหน่วยความจำเสมือนและหน่วยความจำจริง

Page Table หมายถึง ตารางที่เก็บข้อมูลการ Map ระหว่างแอดเดรสจริงและแอดเดรสเสมือน และ Page Fault หมายถึงการที่ Page ข้อมูลที่ต้องการไม่อยู่ในหน่วยความจำจริง

ในการจัดการหน่วยความจำแบบ Page เป็นการจัดเก็บข้อมูลที่ง่ายกว่าแบบ Segment เพราะแต่ละบล็อกจะเก็บข้อมูลขนาดเท่ากัน แต่ถ้าจะเรียบเทียบกับ Segment แล้ว การใช้แบบ Page จะมีโอกาสที่จะเหลือพื้นที่ที่ไม่ได้ใช้ประโยชน์อันเนื่องจากข้อมูลที่เข้ามาไม่เต็ม Page ซึ่งเรียกว่า Internal Fragmentation ส่วนกรณี Segment จะมีโอกาสที่พื้นที่ในหน่วยความจำไม่ได้ใช้ประโยชน์ เพราะขนาดของพื้นที่ที่เหลือนั้นไม่พอเพียงที่จะอ่าน Segment ของข้อมูลทั้งหมดเข้ามาซึ่งกรณีนี้เรียกว่า External Fragmentation

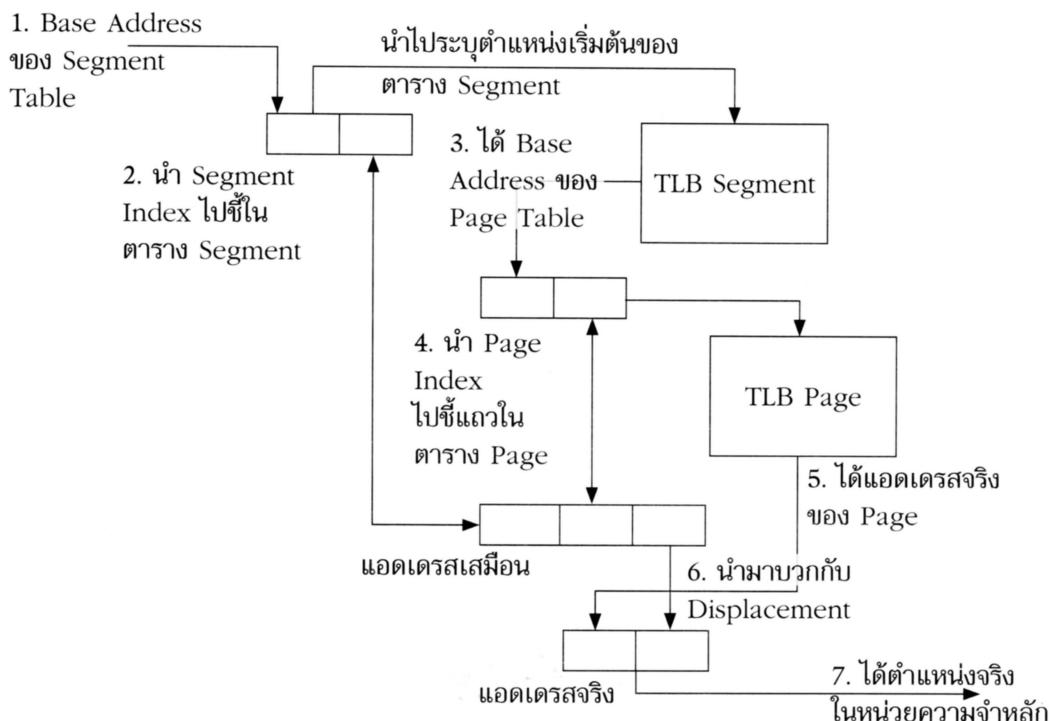
บางครั้งอาจจะมีการนำแบบ Page และ Segment เพื่อให้เกิดประโยชน์สูงสุด กล่าวคือ ในหนึ่ง Segment จะแบ่งเป็น Page ย่อยๆ ตั้งนั้น แอดเดรสเสมือนจะต้องเก็บพิลต์ ตั้งนี้

Segment Index: Page Index: Displacement

และจะใช้ TLB 2 ตัวสำหรับเก็บ Segment และ Page Table ตามลำดับ ขั้นตอนในการแปลงแอดเดรสเสมีອนเป็นแอดเดรสจริจจะเป็น ดังนี้

1. นำ Segment Index ไปค้นใน TLB ของ Segment Table เพื่อหา Base Address จริงของ Page Table ที่ต้องการ เพราะ 1 Segment มีหลาย Page
2. นำ Page Base Address มารวมกับ Page Index ดูว่าเป็น Page ได้ และไปค้นใน TLB ของ Page Table เพื่อหา Base Address จริงของ Page ที่ต้องการ
3. จากแอดเดรสจริจที่ได้มารวมกับ Displacement เป็นแอดเดรสจริจของข้อมูลที่ต้องการ

ขั้นตอนเหล่านี้แสดงดังรูปที่ 9.25



รูปที่ 9.25 ระบบการจัดการหน่วยความจำที่ใช้ทั้ง Page และ Segment

ขนาดของ Page

ขนาดของ Page เป็นสิ่งสำคัญดังที่ได้กล่าวมาแล้ว เพราะขนาดของ Page จะมีผลกระทบต่อการใช้ประโยชน์ของพื้นที่ของหน่วยความจำ และอัตรารับส่งข้อมูล เพราะเมื่อ Page ที่ต้องการไม่อยู่ในหน่วยความจำหลักนั้น หรือมี Page Fault นั่นเอง ก็จะต้องส่งทั้ง Page ที่ต้องการนั้นเข้ามาในหน่วยความจำ ถ้าขนาดของ Page ใหญ่ จะทำให้ต้องมีการรับส่งข้อมูลทั้ง Page มาก นอกจากนี้มีโอกาสจะทำให้เกิดพื้นที่ของส่วนที่ใช้ไม่ได้ภายใน Page (Internal Fragmentation) เพราะว่าจะมีพื้นที่บางส่วนใน Page นั้นๆ ที่ไม่ได้ใช้ประโยชน์ เนื่องจากโปรแกรมหรือข้อมูลที่ต้องการอยู่ใน Page มีขนาดเล็กกว่าพื้นที่ของ Page นั้นๆ และถ้าขนาดของ Page เล็กเกินไปจะทำให้ Page Table มีขนาดใหญ่ เพราะจะมีจำนวน Page มากและ Page Table ต้องเก็บข้อมูลของทุก Page ที่เป็นไปได้ว่าแต่ละ Page อยู่ที่ไหนในหน่วยความจำจริงหรืออยู่ในหน่วยความจำสำรองแม้ว่าจะเป็นการเพิ่มประโยชน์การใช้พื้นที่ก็ตาม

ในระบบที่มีทั้ง Segment และ Page Table โดยหนึ่ง Segment แบ่งเป็น Page หลาย Page

ให้ S_s เป็นขนาดโดยเฉลี่ยของ Segment (ขนาดเป็นเริร์ด)

ให้ S_p เป็นขนาดโดยเฉลี่ยของ Page (เป็นเริร์ด) โดยที่ $S_s >> S_p$ พื้นที่ที่เสียไปในหน่วยความจำ ก็คือพื้นที่ส่วนของ Page Table และพื้นที่ของส่วนที่ใช้ไม่ได้ภายใน (Internal Fragment) โดยที่พื้นที่ส่วนของ Page Table คือ $\frac{S_s}{S_p}$ ซึ่งคือจำนวน Entry ใน Page Table สำหรับ 1 Segment (สมมติว่า 1 Entry ของ Page Table มีขนาด 1 เริร์ด) และโดยเฉลี่ยสมมติให้ Page สุดท้ายใช้ประโยชน์ได้แค่ครึ่ง Page เท่ากับ $\frac{S_p}{2}$ ดังนั้น พื้นที่ที่ไม่ได้ใช้ประโยชน์ต่อ 1 Segment คือ

$$S = \frac{S_p}{2} + \frac{S_s}{S_p} \quad (9.1)$$

ดังนั้น สัดส่วนของขนาด Segment และขนาด Segment รวมกับ Overhead คือ

$$u = \frac{S_s}{S_p + S} \quad (9.2)$$

โดยที่ S_s คือ ขนาดของ 1 Segment ถ้านำทั้งสองสมการ 9.1 และ 9.2 มารวมกันได้ว่า

$$u = \frac{2 S_s S_p}{S_p^2 + 2S(1 + S_p)}$$

ในการหาขนาดของ Page ที่ดีที่สุด เพื่อให้เกิดประโยชน์ต่อการใช้พื้นที่ในหน่วยความจำสูงสุด หรือ Maximize ค่าในตัวแปร u เราจะทำการหาอนุพันธ์ของ S เทียบกับ S_p โดยให้ $\frac{dS}{dS_p} = 0$ จะได้ว่า $S_p^{\text{OPT}} = \sqrt{2S_s}$ และเมื่อแทนค่าแล้วกลับไปในสมการ (9.2) จะได้ว่า

$$u^{\text{OPT}} = \frac{2S_s S_p}{1 + \sqrt{\frac{2}{S_s}}}$$

ถ้าพิจารณาขนาดของ Page มีผลต่อ Hit Ratio อย่างไร กำหนดการอ้างถึงแอดเดรสได้แก่ A_i และ A_{i+d} โดยที่ระยะห่างระหว่าง 2 แอดเดรสนี้ในหน่วยความจำคือ d กรณีความกว้างเป็นของการพับแอดเดรสนั้น ในหน่วยความจำมีสูง เพื่อเป็นการเพิ่ม Hit Ratio จะทำได้ดังนี้

- ค่า d น้อย หมายถึงว่าแอดเดรส A_i และ A_{i+d} มีโอกาสที่จะอยู่ภายใน Page เดียวกัน ดังนั้น ควรจะกำหนดขนาดของ Page ให้มากกว่า d เพื่อเพิ่มอัตราการ Hit
- ถ้า A_i และ A_{i+d} อยู่ใน Page ที่ต่างกัน แต่ว่า Page ที่เก็บแอดเดรสทั้งสองถูกอ้างอิงบ่อย ความกว้างจะเป็นของการที่จะพับแอดเดรส A_{i+d} ในหน่วยความจำจะเพิ่มขึ้นเมื่อเก็บจำนวน Page ไว้ในหน่วยความจำได้มากขึ้น อีกนัยหนึ่งก็คือการลดขนาดของ Page ลง เพื่อให้หน่วยความจำบรรจุได้หลาย Page

จะเห็นว่าทั้งสองกรณีนี้เป็นกรณีที่ขัดแย้งกันอยู่ ดังนั้น จึงควรมีการจัดขนาดของ Page ที่เหมาะสมที่พอดีระหว่างทั้งสองกรณีนี้

การจัดสรรหน่วยความจำ

เป็นการจัดสรรพื้นที่ในหน่วยความจำเพื่อมาเก็บ Page ที่ต้องการใช้ปกติแล้ววิธีการเรียกใช้ Page มี 2 วิธี คือ Anticipatory Paging ซึ่งต้องมีการทำนายว่าในอนาคตจะใช้ Page ใด โดยการทำนายจะดูว่าในระยะเวลาล้านๆ ในอนาคต รูปแบบของการอ้างถึงแอดเดรสเป็นอย่างไร อีกวิธีเรียกว่า Demand Paging โดยจะมีการนำ Page ที่ต้องการใช้เข้ามาในหน่วยความจำเมื่อมีการร้องขอเท่านั้น

เมื่อมีการนำ Page หนึ่งๆ เข้ามาแล้ว จะต้องมีการเลือกว่าจะเอา Page นั้นไปวางไว้ที่ตำแหน่งใดในหน่วยความจำ วิธีการเลือกพื้นที่ที่จะวางบล็อกหนึ่นเรียกว่า Replacement Policy นั้นเอง ตามปกติแล้ว ในการจัดสรรพื้นที่จะต้องเก็บข้อมูลต่อไปนี้ไว้ด้วย เช่น มีพื้นที่ใดบ้างที่ได้ถูกใช้

งานอยู่ เก็บไว้เพื่อให้บังยั่งว่างอยู่ เก็บได้หากหรือว่า พื้นที่ได้ในหน่วยความจำคราวเป็นผู้ใช้ เป็นต้น

หลังจากมีการจัดสรรพื้นที่ในหน่วยความจำมาใช้ เมื่อเลิกใช้พื้นที่นั้นแล้วจะต้องคืนพื้นที่ให้กับระบบเพื่อให้สามารถนำพื้นที่นั้นมาใช้ใหม่อีก ไปได้ การคืนพื้นที่เรียกว่า Deallocation

ลักษณะของการจัดสรรพื้นที่ในหน่วยความจำมาใช้แบ่งได้เป็น 2 ลักษณะ คือ แบบ Preemptive และ Nonpreemptive

- แบบ Nonpreemptive เป็นลักษณะของการหาพื้นที่ว่างสำหรับเก็บล็อกข้อมูลโดยจะนำบล็อกข้อมูลเข้ามาในหน่วยความจำ โดยที่ไม่ไปเยี่ยนทั่วบล็อกต่างๆ ที่มีอยู่แล้วในหน่วยความจำ วิธีการที่จัดว่าเข้าลักษณะ Nonpreemptive ได้แก่
 1. First Fit เป็นวิธีที่นำบล็อกไปวางในพื้นที่หน่วยความจำในบล็อกที่ว่างที่พบอันแรก
 2. Best Fit เป็นวิธีการนำบล็อกข้อมูลไปไว้ในหน่วยความจำโดยหาพื้นที่ว่างที่มีขนาดใกล้เคียงกับขนาดบล็อกข้อมูลที่นำเข้ามา
 3. Worst Fit เป็นวิธีที่นำบล็อกข้อมูลไปวางไว้ในพื้นที่ว่างที่ใหญ่ที่สุดที่พบ โดยหวังว่าเมื่อใส่ข้อมูลไปในพื้นที่ว่างนั้นแล้วจะยังมีพื้นที่ว่างเหลือที่เพียงพอทำให้放入บล็อกข้อมูลอื่นได้อีก
- แบบ Preemptive เป็นลักษณะของการนำพื้นที่ที่ใช้แล้วมาใช้ใหม่ เมื่อพื้นที่ในหน่วยความจำได้ถูกใช้หมดแล้วโดยที่จะเลือกว่าจะเอาพื้นที่ตรงตำแหน่งใดใช้ได้อีก วิธีการนี้มี 2 ประเภทย่อย ได้แก่
 1. Relocation คือ การย้ายบล็อกของข้อมูลในหน่วยความจำให้ไปอยู่ในพื้นที่ติดกัน หรือเป็นการทำ Memory Compaction เพื่อร่วมเอาพื้นที่ว่างเล็กๆ ที่เกิดขึ้นเป็นพื้นที่ว่างขนาดใหญ่และนำมาใช้เก็บบล็อกข้อมูลขนาดใหญ่ได้อีก
 2. การใช้ Replacement Policy เมื่อใช้วิธีการ Compaction ดังกล่าวแล้วยังไม่เหลือที่ว่างที่จะอ่านข้อมูลเข้ามาจำต้องมีการเลือกเอาว่า จะเอาข้อมูลใหม่ไปแทนที่บล็อกข้อมูลใดในหน่วยความจำ วิธีการที่ใช้เลือกนี้ เรียกว่า Replacement Policy โดยบล็อกข้อมูลที่จะถูกแทนที่อาจจะได้มีการแก้ไขในหน่วยความจำ (Dirty Block) หรืออาจจะเป็นบล็อกที่ไม่ได้มีการเปลี่ยนแปลง (Clean Block) ถ้าบล็อกนั้นมีการแก้ไขแล้ว

จะต้องมีการเขียนข้อมูลกลับไปยังดิสก์เพื่อสำรองบล็อกข้อมูลให้ถูกต้องก่อน สำหรับ Replacement Policy ในที่นี้จะกล่าวถึง 3 แบบ คือ

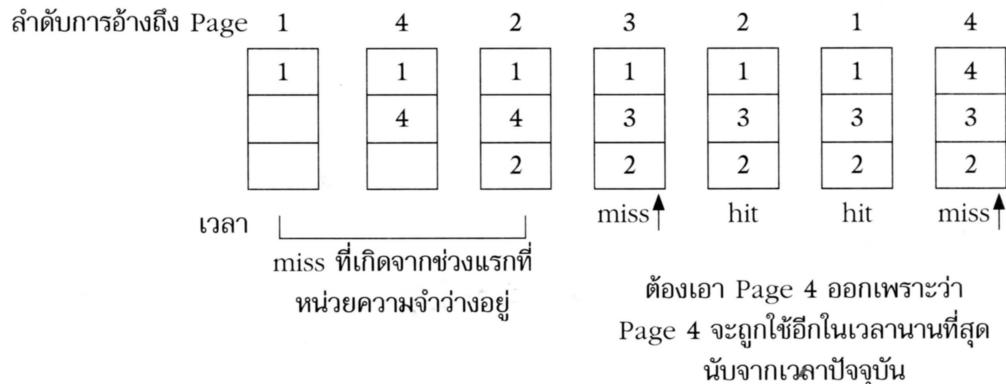
- OPT (Optimal Page Replacement) เป็นการเลือกเอา Page ที่จะໄปให้แล้วในอนาคตอันที่ใกล้ออก ข้อเสียของวิธีนี้คือจะต้องรู้การอ้างถึงแอดเดรส ทั้งหมดหรือลำดับของแอดเดรสของข้อมูลที่จะໄป
- LRU (Least Recently Used) จะใช้ริจิสเตอร์เป็นตัวนับว่าในอดีตมีการอ้างถึงแอดเดรสนั้นกี่ครั้งและจะเลือกเอาบล็อกที่ไม่ได้ใช้หรือมีการอ้างถึงอย่างน้อยที่สุดในอดีตออกไป โดยดูจากค่าในริจิสเตอร์ตัวนับนั้น
- FIFO (First In First Out) จะแทนที่บล็อกที่นำเข้ามาก่อนเสมอ

การหาค่าอัตราการ Hit หาได้จาก $H = \frac{N_1}{N_1 + N_2}$ โดยที่ N_1 คือ จำนวนครั้งในการอ้างถึงแอดเดรสในหน่วยความจำ M_1 ที่ใกล้กับชีพียุมากที่สุด และความน่าจะเป็นของการเกิด Page Fault คือ $1 - H$

ตัวอย่าง สมมติให้การอ้างถึงแอดเดรสที่ต้องการอ่านเข้ามาในหน่วยความจำเป็นลำดับ ดังนี้

1 4 2 3 2 1 4

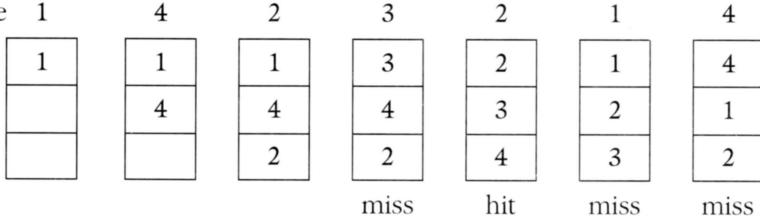
ถ้าหน่วยความจำจริงสามารถเก็บได้เพียง 3 บล็อก เริ่มต้นกำหนดให้หน่วยความจำว่าง ถ้าใช้ Replacement Policy คือ OPT มีการทำงานดังรูปที่ 9.26



รูปที่ 9.26 การจัดการ Replacement Policy แบบ OPT

ถ้าใช้ Replacement Policy แบบ LRU มีการทำงานดังรูปที่ 9.27

ลำดับการอ้างถึง Page 1

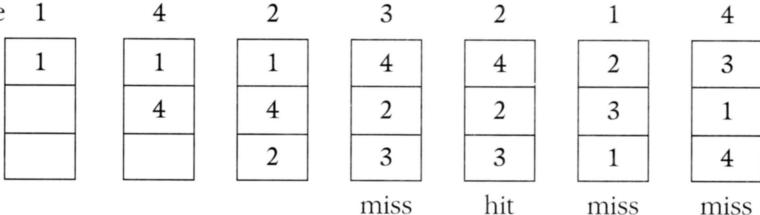


ເລົາ Page 4 ອອກພະຈານໄດ້ໃຫ້
ໃນອດືອນນານທີ່ສຸດ

รูปที่ 9.27 การจัดการ Replacement Policy แบบ LRU

ถ้าใช้ Replacement Policy แบบ FIFO ซึ่งมีการทำงานดังรูปที่ 9.28

ลำดับการอ้างถึง Page 1



ເອົາ Page 1 ອອກ

รูปที่ 9.28 การจัดการ Replacement Policy แบบ FIFO

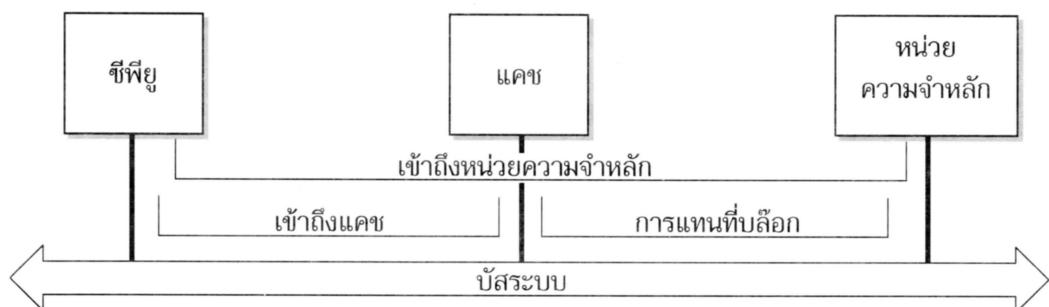
จะเห็นว่า LRU จะมีการจำว่ามี Page ใดที่ถูกเรียกใช้น้อยที่สุดในอดีตบ้าง และจะเอา Page นั้นออก ถ้ามีมากกว่า 1 Page ที่ถูกใช้น้อยเท่ากัน ก็สุ่มเลือกเอาออก 1 Page ส่วน FIFO จะเอา Page ที่นำเข้ามาเก็บอนกอกไปก่อนเสมอ และ OPT จะพิจารณาจากการอ้างถึงตำแหน่งในอนาคต (วิธีนี้เป็นลักษณะที่เรียกว่า Offline เพราะต้องมีแอดเดรสที่จะใช้ทั้งหมดล่วงหน้า) OPT จะนำแอดเดรสที่ไม่ใช้ในอนาคตอันใกล้ออกไป ส่วนถ้าไม่ทราบแอดเดรสที่จะใช้ วิธี LRU ก็จะให้ประสิทธิภาพที่ดี นอกจากนี้ การเพิ่มจำนวน Page ในหน่วยความจำจะลดจำนวน Page Fault ได้ด้วย

9.5 แคช (Cache)

แคชเป็นบัฟเฟอร์พิเศษอยู่ระหว่างหน่วยความจำหลักและชีพียู มีราคาแพงกว่าหน่วยความจำหลัก และชีพียู สามารถเข้าถึงข้อมูลในแคชได้เร็วกว่าหน่วยความจำหลัก

โครงสร้างของระบบแคชประกอบด้วยแคชที่เก็บข้อมูล โดยหน่วยความจำข้อมูลของแคชจะแบ่งเป็น Line หรือ บล็อกแต่ละบล็อกเรียกว่า Cache Line (หรือ Cache Block) ในแต่ละ Line จะมี Tag ซึ่งเป็นแอดเดรสที่เก็บเป็นฟิลด์ เพื่อบอกว่าข้อมูลในบล็อกนี้อยู่หน่วยความจำหลักตำแหน่งใด หน่วยความจำล้วนที่เก็บ Tag ส่วนนี้เรียกว่าหน่วยความจำ Tag (Cache Tag Memory)

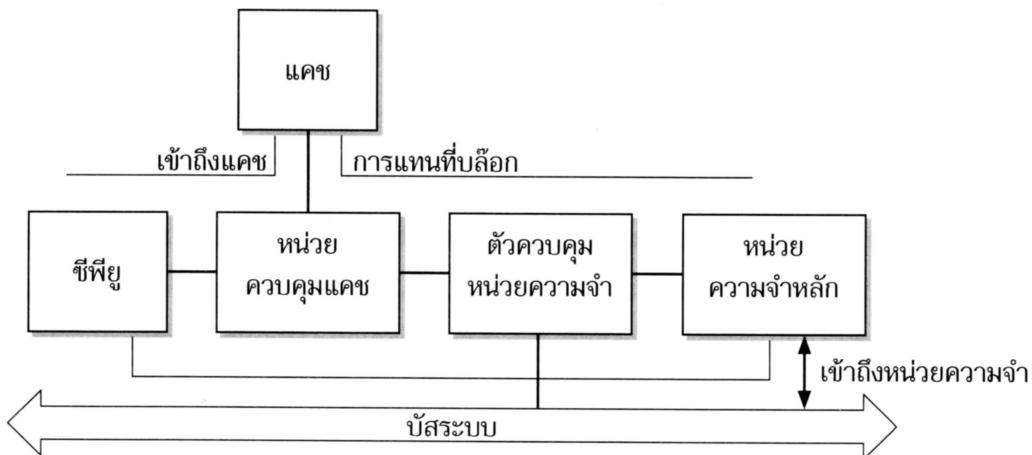
ปกติแล้วในการค้นหาข้อมูลในแคชจะต้องเสียเวลาในการตรวจสอบในแคชเพื่อหาว่าในหน่วยความจำที่เก็บ Tag ว่าเก็บแอดเดรสของข้อมูลที่ต้องการหาหรือไม่ และถ้าใช่ก็จะไปเอาข้อมูลจาก Cache Line ที่สัมพันธ์กันมา รูปแบบของการออกแบบการวางแคชในระบบที่เป็นไปได้ได้แก่ แบบ Look Aside และแบบ Look Through ในแบบ Look Aside แสดงดังรูปที่ 9.29



รูปที่ 9.29 แคชแบบ Look Aside

เมื่อต้องการหาข้อมูล จะเปรียบเทียบแอดเดรสของข้อมูลที่ต้องการหาในแคชก่อน ถ้าพบ แอดเดรสในแคชก็จะส่งข้อมูลที่สัมพันธ์กันในแคชนั้นมาให้ชีพียูเลย และถ้าไม่พบจะไปหาข้อมูลนั้น ในหน่วยความจำหลัก และส่งเข้ามาเก็บในแคช โดยจะทำการเลือกว่าจะเก็บข้อมูลไว้ที่ใดในแคช และใช้ Replacement Policy เลือกบล็อกที่จะแทนที่

ส่วนรูปที่แบบของ Look Through เป็นดังรูปที่ 9.30



รูปที่ 9.30 แคชแบบ Look Through

ชีพียูจะติดต่อกับแคชโดยใช้บัสภายใน (Local Bus) พิเศษ ส่วนบัสระบบ (System Bus) จะได้ใช้ในการกิจอื่นๆ ลักษณะนี้ทำให้การเข้าถึงแคช และหน่วยความจำเข้าถึงได้พร้อมกัน ชีพียูจะไม่ส่งคำร้องขอไปยังหน่วยความจำหลักเลย แต่จะส่งคำร้องขอไปยังแคช ยกเว้นกรณีมีการ Miss เท่านั้น

ในการทำงานของแคชมีขั้นตอนที่เกิดขึ้นดังนี้

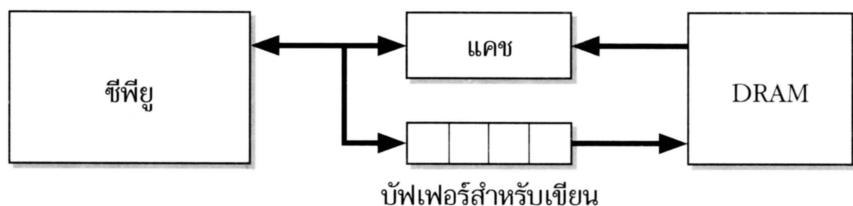
1. ส่งแอดเดรสที่ต้องการมาเปรียบเทียบกับแอดเดรสของ Tag ที่เก็บในหน่วยความจำที่เก็บ Tag
2. ถ้าพบ Tag ที่ต้องการ ตำแหน่งที่เก็บ Tag จะสัมพันธ์กับ Cache Line ในหน่วยความจำข้อมูลของแคช ดังนั้น จึงไปที่ Cache Line นั้น เอาข้อมูลที่ต้องการและใช้ Displacement เพื่อรบุว่าเป็นไปต่ำใดใน Cache Line นั้นๆ

อีกนัยหนึ่งในการจัดเก็บข้อมูลและ Tag มองเป็น Associative Array ของ (Tag, Data) ก็ได้ ลังเกตว่าในที่นี้หน่วยความจำข้อมูลของแคช จะสัมพันธ์กับข้อมูลในหน่วยความจำหลักด้วยใน Tag แอดเดรสนั้นๆ

ถ้าเป็นการเขียนข้อมูลก็จะเหมือนกับการอ่าน ในการนี้สำหรับการ Hit แต่ต้องคำนึงถึงความสอดคล้อง (Consistency) ของข้อมูลในแคช และหน่วยความจำหลักด้วย เมื่อข้อมูลไม่อยู่ในแคช ก็ให้นำเข้ามาจากหน่วยความจำหลัก

Cache Consistency คือ การรักษาความสอดคล้องของข้อมูลระหว่างระดับ โดยข้อมูลในแคชจะต้องเหมือนกับข้อมูลในหน่วยความจำหลักเสมอ เนื่องจากมีการเก็บข้อมูลเหมือนกันในหน่วยความจำมากกว่า 1 แห่งขึ้นไป วิธีการรักษาความสอดคล้องนั้นมีหลายอย่าง เช่นการใช้บิตพิเศษเรียกว่า Change Bit (C) เป็นตัวบอกว่า Cache Line นั้นได้มีการแก้ไข (Dirty) หรือไม่ ถ้า C=1 แปลว่า Cache Line นั้นไม่มีการแก้ไข (Clean) ถ้ามีการแก้ไข ก็จะต้องทำการเขียนข้อมูลในแคชกลับไปยังหน่วยความจำและติดสก์ เพื่อให้ถูกต้องสอดคล้องกันด้วย

การปรับข้อมูลให้สอดคล้องกันระหว่างข้อมูลในหน่วยความจำหลัก และภายในแคชนั้น (Consistency) ควรมีมากเท่าใดทั้งนี้ขึ้นกับวิธีการ ในวิธี Write Back คือ การเขียนกลับไปทุกครั้งที่แคชนั้นมีการเปลี่ยนแปลงซึ่งแตกต่างกับวิธี Write Through ที่มีการเขียนกลับไปทุกครั้งที่มีการแก้ไขข้อมูลในแคช ในวิธีการ Write Back นั้นจะต้องระวังว่าอาจจะมีความไม่สอดคล้องระหว่างข้อมูลในหน่วยความจำหลักและข้อมูลในแคชเกิดขึ้นช้ากว่าบ้าน้ำง ในขณะที่วิธี Write Through ข้อมูลในระบบหน่วยความจำจะมีความสอดคล้องตลอดเวลา แต่ต้องทำการเขียนไปยังหน่วยความจำบ่อยกว่า ซึ่งจะก่อให้เกิดการใช้งานบัสข้อมูลมากได้ โดยเฉพาะถ้ามีการเปลี่ยนแปลง Cache Line เดิมปอยๆ ที่ดำเนินไปต่อต่างกันแต่ใน Line เดียวกันนั้นจะต้องถูกโอนไปยังหน่วยความจำหลักทุกครั้งที่มีการเปลี่ยนแปลงซึ่งต่างจากวิธีการ Write Back ที่จะลดจำนวนครั้งของการรับส่งข้อมูลไปได้ โดยเฉพาะอย่างยิ่งเมื่อ Cache Line มีขนาดใหญ่การรับส่งข้อมูลปอยๆ ดังกรณีจะทำให้เกิดการชนกันของการส่งรับข้อมูลได้ ทำให้การรับส่งข้อมูลนั้นช้าลง ในการ Write Through จะจะใช้บัฟเฟอร์สำหรับการเขียน (Write buffer) มาช่วยเก็บข้อมูลที่จะเขียน ไปเพื่อให้ลดเวลาในการรอการเขียนให้เสร็จสิ้นได้



รูปที่ 9.31 การใช้บัฟเฟอร์สำหรับการเขียน

นอกจากนี้ในกรณีของการเกิด Miss สำหรับการเขียนนั้น ถ้าข้อมูลที่ต้องการจะเขียนไม่อยู่ในแคชจะทำอย่างไรนั้น วิธีการจัดการนั้นมี 2 วิธี คือ

- Write Allocate เป็นการนำล็อกข้อมูลที่ต้องการเขียนเข้ามา Cache Line ด้วยกรณีที่มีการ Miss และจึงเขียนไปยัง Cache Line นั้น
- No-Write Allocate จะไม่นำบล็อกข้อมูลที่ต้องการเขียนเข้ามาใน Cache Line จะให้เขียนไปที่หน่วยความจำเลย

พิจารณาผลการเลือกทั้งสองวิธีในตัวอย่างง่ายๆ นี้

สมมติให้แอดเดสเป็นแบบ Fully Associative (ดังจะได้กล่าวในหัวข้อถัดไป) โดยมีจำนวน Cache Line มากๆ และเริ่มต้นให้ทุก Line ของแอดเดสว่างทั้งหมด ถ้ามีการอ้างถึงหน่วยความจำตามลำดับดังนี้

WriteMEM[100]⁵, WriteMEM[100], ReadMEM[200], WriteMEM[200], WriteMEM[100]

ถ้าใช้วิธี No-Write Allocate จะมีการ Miss กี่ครั้งและ Hit กี่ครั้ง

จากตัวอย่างลำดับการ Miss ได้ดังนี้

1. WriteMEM[100] เป็นการ Miss ครั้งแรก
2. WriteMEM[100] เป็นการ Miss เพราะครั้งแรกใช้ No-Write Allocate
3. ReadMEM[200] เป็นการ Miss อีกครั้ง
4. WriteMEM[200] เป็นการ Hit
5. WriteMEM[100] เป็นการ Miss ทำให้มีการ Miss จำนวน 4 ครั้งนั่นเอง

ถ้าใช้วิธี Write-Allocate ผลจะเป็นดังนี้

1. WriteMEM[100] เป็นการ Miss ครั้งแรก
2. WriteMEM[100] เป็นการ Hit เพราะใช้ Write Allocate
3. ReadMEM[200] เป็นการ Miss อีกครั้ง

⁵ WriteMEM[100] หมายถึงเขียนไปยังหน่วยความจำตำแหน่ง 100 และ ReadMEM[100] หมายถึงอ่านหน่วยความจำตำแหน่ง 100

4. WriteMEM[200] เป็นการ Hit

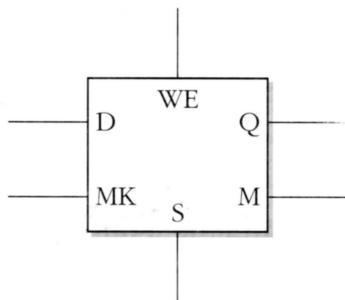
5. WriteMEM[100] เป็นการ Hit

ทำให้มีการ Miss จำนวน 2 ครั้งนั่นเอง

จะเห็นว่าในกรณีนี้จะมีจำนวนครั้งที่หาพบในแคชเพิ่มขึ้น เนื่องจากการนำบล็อกเข้ามาและมีโอกาสส่วนบล็อกนั้นจะได้ใช้ต่อไปในอนาคต

ในการออกแบบแคชนั้นนอกจากคำนึงถึงวิธีการเขียนแล้ว ยังมีประเด็นอื่นๆ ทางด้านฮาร์ดแวร์อีก เช่น เราจะรู้ได้อย่างไรว่าข้อมูลที่ต้องการอยู่ในแคช หรือไม่ และถ้าอยู่แล้วจะค้นหาข้อมูลอย่างไร ในการหาข้อมูลที่ต้องการ จะต้องใช้แอดเดรสของ Tag โดยต้องการฮาร์ดแวร์มาช่วยทำการเปรียบเทียบ และถ้าวิธีการหาไม่ดี ก็จะทำให้การหาข้อมูลในแคชช้าได้ เช่น ถ้าค้นหา Tag ในแคชทั้งหมดก็จะช้า แต่ถ้ามีการจัดกลุ่มของ Tag ออกเป็นกลุ่มๆ เพื่อช่วยในการค้นหา Tag ก็จะทำให้การค้นหาเร็วขึ้นแต่อาจจะทำให้ฮาร์ดแวร์ขับข้อนั้น โดยการจัดกลุ่มจะจัดกลุ่มคละหนึ่งบล็อก (Direct Map) หรือกลุ่มละ 2 บล็อก (2-Way Set Associative) หรือแบบกลุ่มละ n บล็อก (Fully Associative) ก็ได้ ในการจัดกลุ่มนั้นวิธีการค้นหาบล็อกก็มีผลด้วย

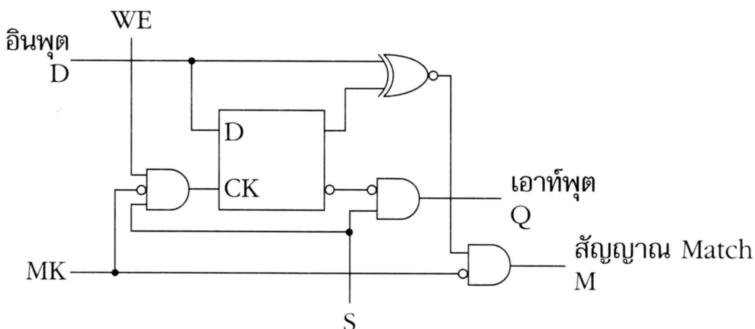
ในการอิมเพลเม้นต์ฮาร์ดแวร์สำหรับแบบ Associative Memory Cell ดังรูปที่ 9.32 ในการทำงานเริ่มต้นแอดเดรสที่เข้ามายังถูก Mask บิตเพื่อเอาเฉพาะส่วนที่เป็น Tag ออกมา และใช้ Tag ในการเปรียบเทียบกับข้อมูลในหน่วยความจำ Tag ทุกดัว ถ้า Tag ที่จะค้นหาไม่ค่าตรงกับค่าในหน่วยความจำที่เก็บ Tag มากกว่าหนึ่งตัว จะใช้ Select Logic ทำการเลือกว่าจะเอาข้อมูลที่สัมพันธ์กับ Tag ตัวใดออกมาน โดยแต่ละเวิร์ดของข้อมูลจะมีวงจรเปรียบเทียบเป็นของตัวเอง



รูปที่ 9.32 สัญลักษณ์ของ Associative Memory Cell

ในรูปที่ 9.32 ขา WE คือ Write Enable ขา MK คือ การ Mask อินพุต สัญญาณเอาท์พุต M ให้ผลว่า หากหรือไม่ ถ้า $M=1$ คือ หาก และถ้า $M=0$ คือหากไม่พบ ส่วนขา Q คือ ขาที่ส่งข้อมูลออก D คือ ขาที่ส่งข้อมูลเข้าไปเขียน และ S คือ สัญญาณการเลือกใช้ Cell นี้

ลักษณะภายในของ Associative Memory ใช้ D Flip Flop ดังรูปที่ 9.33

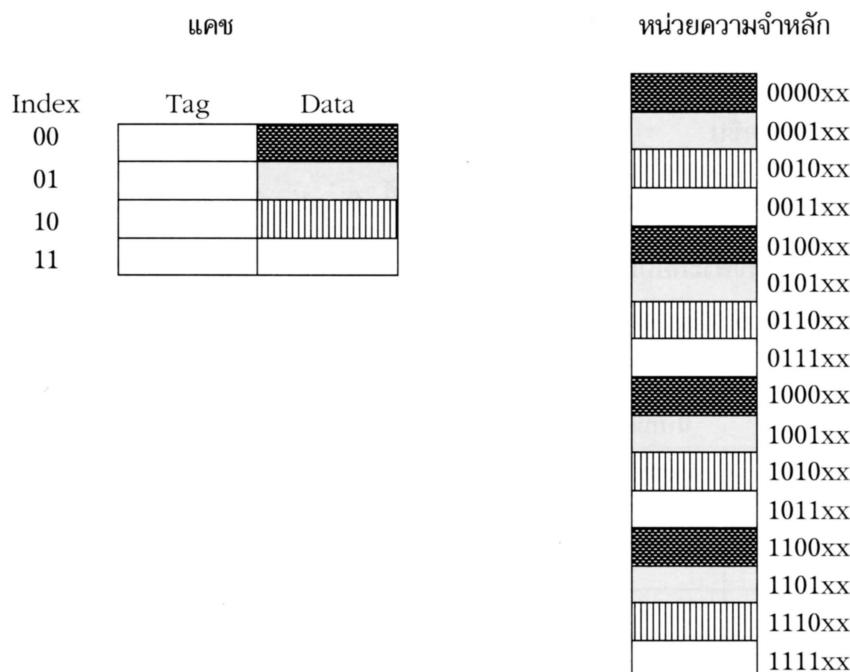


รูปที่ 9.33 ลักษณะภายในของ Associative Memory

- ถ้า $WE=1$ เป็นการเขียน จะเอาสัญญาณ WE เชื่อมกับสัญญาณนาฬิกาเพื่อช่วยกำหนดข้อมูลใหม่
- ในรูปที่ 9.33 เกต XNOR ใช้เพื่อเปรียบเทียบว่าค่าใน D-Flip Flop และค่าอินพุต D เมื่ອนกันหรือไม่ เพื่อใช้ในการค้นหา Tag ถ้าเมื่อกันให้สัญญาณ $M=1$ ถ้าไม่เมื่อกันจะให้สัญญาณ $M=0$
- ถ้า $S=1$ จะ Enable ให้เลือก Memory Cell นี้ได้ ถ้าเป็นการอ่านให้อ่านจากขา Q ซึ่งจะเป็นข้อมูลเอาท์พุต แต่ถ้าเป็นการเขียนให้อ่านจากขา D จะเป็นข้อมูลอินพุต
- ถ้า $MK=1$ จะทำให้สัญญาณ $M=0$ เพื่อเขียน โดยไม่สนใจข้อมูลใน D-Flip Flop นอกจากนี้ยังใช้ในการ Disable อินพุตด้วย

Direct Mapping

เป็นการแบ่งแคชออกเป็นส่วนๆ เรียกว่าเขตและแบ่งหน่วยความจำออกเป็นเขตที่สัมพันธ์กัน ดังรูปที่ 9.34



รูปที่ 9.34 แคชแบบ Direct Mapping

ในรูปที่นี้แคชมี 4 เขตเท่านั้น ในแต่ละเขตมี 1 Cache Line สองบิตล่างของตำแหน่งของหน่วยความจำใช้ระบุบอกว่าต้องการไปต่อใน Cache Line ระบุโดย xx สมมติว่า 1 Cache Line เก็บได้ 1 เวิร์ด ที่มีขนาด 32 บิต พงก์นั้นการแปลงจากบิตล็อกของหน่วยความจำไปยัง Cache Line ในที่นี้คือ

$$j = i \pmod{S_1}$$

กำหนดให้เขตที่ j ใน Cache Line $M_1(j)$ เป็นกลุ่มที่ตรงกันในหน่วยความจำ $M_2(i)$ และกำหนดให้ S_1 คือ จำนวนกลุ่มย่อยภายในเขต ซึ่งสำหรับ Direct Mapped จะมีแค่ 1 กลุ่มต่อ 1 เขตเท่านั้น ในที่นี้จะมี 4 เขตจึงใช้เพียงสองบิตลำดับระบุเขตที่ต้องการ และไปสัมพันธ์กับสองบิตตรงกลางของตำแหน่งในหน่วยความจำ

ถ้าพิจารณาจำนวนพื้นที่ที่ต้องการใช้ในการจัดระบบล็อกแบบนี้ ให้จำนวนบิตข้อมูลที่ใช้ในการเลือกเขตมีความยาว s บิต (เพราะว่า $S_1 = 2^s$) นอกจากนี้ ยังต้องการ t บิตเพื่อเก็บแอดเดรสของ Tag ในหน่วยความจำของ Tag และ d บิตสำหรับ Displacement ภายใน Cache Line (สำหรับเขตที่มีขนาด 2^d เวิร์ด) ดังนั้น ในการอ้างถึงหนึ่งครั้งจะใช้ความยาวทั้งหมดคือ $s + t + d$

ในตัวอย่างข้างต้น เป็นวิธีการ Direct Mapping เมื่อจำนวนเขต = 2 จะมีบล็อกข้อมูลที่ถูกกำหนดเข้ากับพื้นที่เดียวกันมากขึ้นอัตราการ Hit ก็จะลดลงทำให้มีโอกาสนำบล็อกข้อมูลเข้าออกแคชมากขึ้น

พิจารณาตัวอย่าง การอ้างถึงข้อมูลดังนี้ 0 1 2 3 4 3 4 15

ถ้าแคชประกอบด้วย 4 บล็อก จะเห็นว่าจะเกิดการ Miss ใน การอ้างถึง 0, 1, 2, 3, 4, 15 จะมีทั้งหมด 6 Miss จะได้อัตราการ Hit = $2/8 = 0.25$

0 miss

00	Mem(0)

1 miss

00	Mem(0)
00	Mem(1)

2 miss

00	Mem(0)
00	Mem(1)
00	Mem(2)

3 miss

00	Mem(0)
00	Mem(1)
00	Mem(2)
00	Mem(3)

4 miss

01	Mem(4)
00	Mem(1)
00	Mem(2)
00	Mem(3)

3 hit

01	Mem(4)
00	Mem(1)
00	Mem(2)
00	Mem(3)

4 hit

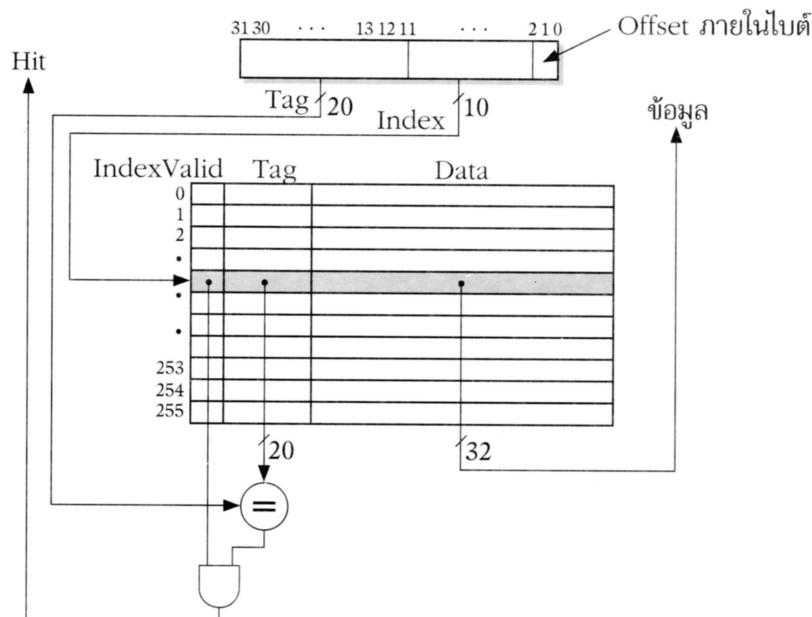
01	Mem(4)
00	Mem(1)
00	Mem(2)
00	Mem(3)

15 miss

01	Mem(4)
00	Mem(1)
00	Mem(2)
11	Mem(15)

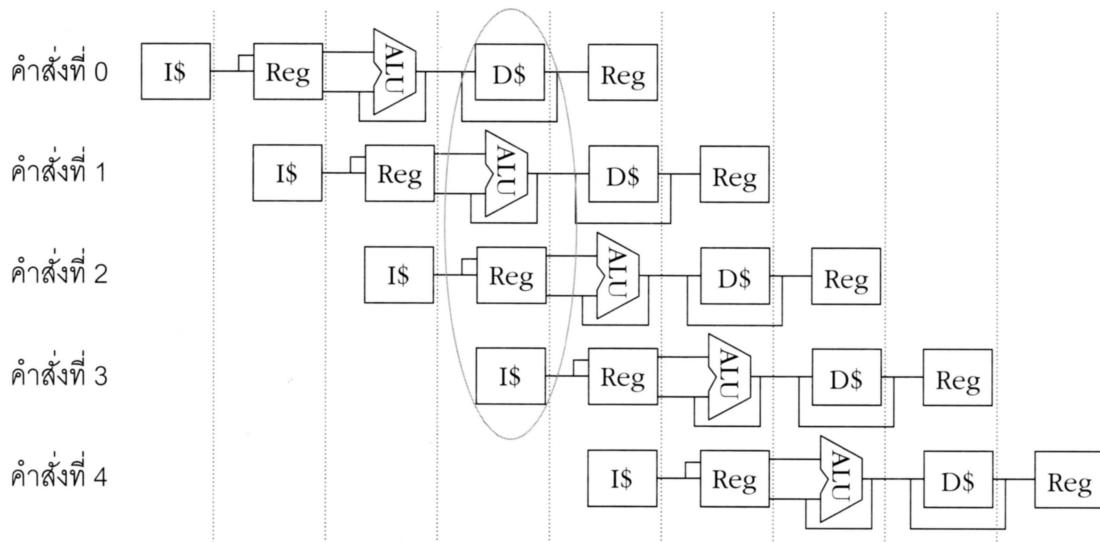
ถ้าออกแบบเป็นแบบ Direct Map โดยให้ 1 เวิร์ดต่อบล็อก จะเป็นดังรูปที่ 9.35 สำหรับแอดเดรสขนาด 32 บิตจะใช้บิต 0-1 เพื่อระบุไปยังภายในบล็อกนั้น เนื่องจากหนึ่งบล็อกมีเพียง 4 ไบต์ ส่วนบิตที่ 2-11 ใช้ระบุ Index ซึ่งอ้างได้ถึง 1024 บล็อก และบิตที่ 12-31 ใช้เก็บ Tag

ในการทำงานจะทำการดึง Index ออกมานำเพื่อไปยังแอดร์รัที่ต้องการ จากนั้นจะตรวจสอบค่า Tag จำนวน 20 บิตภายในแวนั่นด้วยตัวเปรียบเทียบว่าตรงกันที่ระบุไว้ในแอดเดรสหรือไม่ ถ้าตรงกับ Tag ก็จะให้สัญญาณ Hit=1 และข้อมูลจำนวน 32 บิต จะถูกดึงออกมาจากพิล์ด Data ในแคลเดียร์กัน และจะนำเอา Offset ไประบุใบต์ที่ต้องการในบล็อกอีกครั้งหนึ่ง



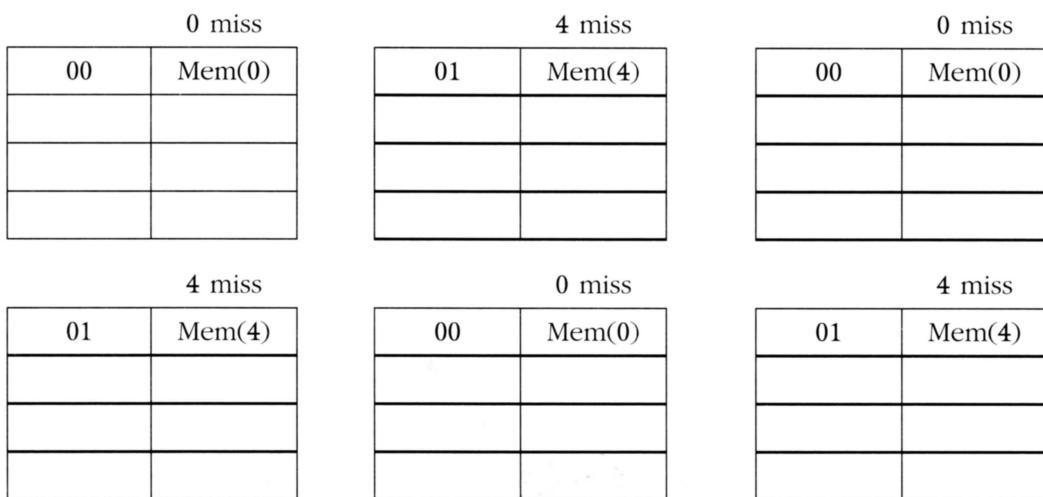
รูปที่ 9.35 การออกแบบ Direct Map พื้นที่ 1 เวิร์ด ต่อบล็อก

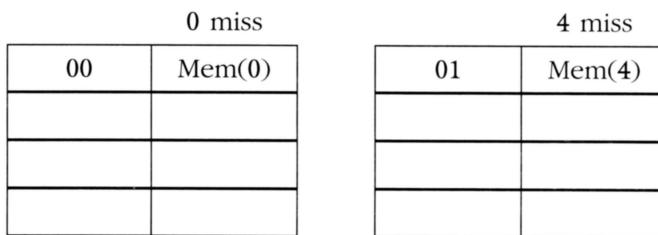
ในส่วนของไปป์ไลน์นั้นจะต้องคำนึงถึงการใช้หน่วยความจำให้ขั้นตอนต่างๆ ที่เกี่ยวข้อง ได้แก่ การเฟตช์ การเข้าถึงหน่วยความจำ เพื่อจะหลีกเลี่ยง Structure Hazard ที่จะเกิดขึ้นดังรูปที่ 9.36 จะเห็นว่าความมีแคชสำรองเก็บทั้งคำสั่งและข้อมูล เพื่อแก้ปัญหา Structure Hazard ดังกล่าว



รูปที่ 9.36 โอกาสเกิด Structure Hazard เกี่ยวกับการใช้หน่วยความจำ

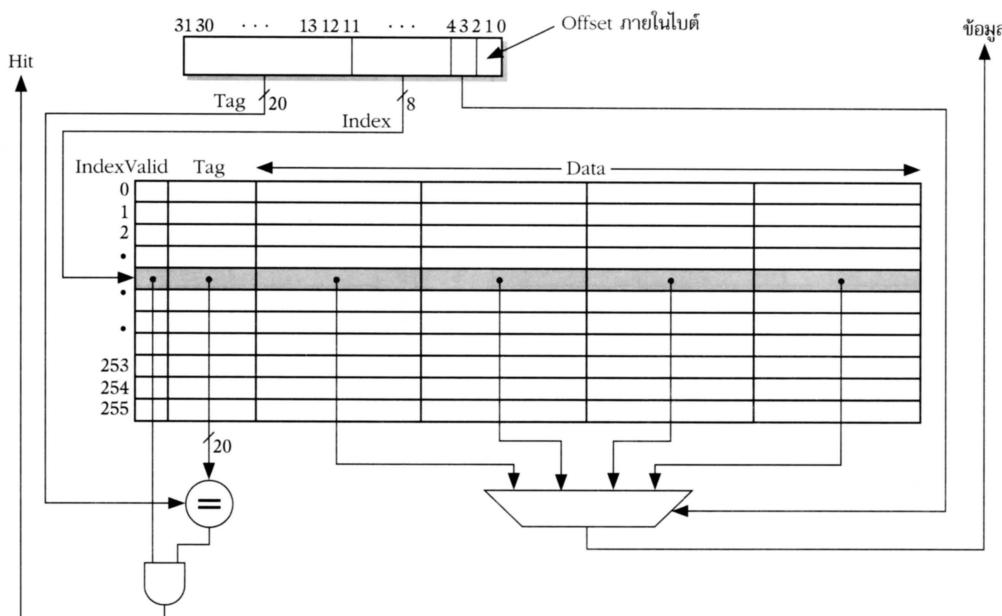
สำหรับกรณีของการอ้างถึงตำแหน่ง 0, 4, 0, 4 ... ในโครงสร้างแบบเดียวกันจะทำให้เกิดการ Miss ตลอดเวลา ซึ่งหมายถึงการใช้กลุ่มละ 1 บล็อกอาจไม่เหมาะสม เนื่องจากทั้งสองแอดเดรส ข้อมูลจะถูกกำหนดไปยังบล็อกที่มีแอดเดรสเดียวกันตลอดเวลา





จึงอาจจะต้องไปใช้การออกแบบ 2-way Set Associative ดังจะได้กล่าวต่อไป

นอกจากจะออกแบบโดยใช้ 1 เวิร์ดต่อบล็อกแล้ว เราอาจจะให้ 4 เวิร์ดต่อบล็อกก็ได้ ในที่นี้จะได้ประโยชน์จาก Locality เทิงเวลา เนื่องจากจะอ่านเวิร์ดที่มีแอดเดรสติดกันมาไว้ในบล็อกเดียวกัน สังเกตว่าในการยังคงจะเพิ่มระยะห่างของบล็อกให้ยกบันบล็อกแรก (Block Offset) เข้าไปด้วย



รูปที่ 9.37 การออกแบบ Direct Map แบบหลายเวิร์ด

การอักแบบยังเป็นแบบ Direct Map โดยให้ 1 เวิร์ดต่อบล็อก และแต่ละแวดมีอยู่ 4 บล็อก จะเป็นดังรูปที่ 9.37 สำหรับแอดเดรสขนาด 32 บิตจะใช้บิต 0-1 เพื่อระบุใบภาระในบล็อกนั้น เนื่องจากหนึ่งบล็อกมีเพียง 4 ใบต์ ใช้บิต 2-3 ระบุบล็อกที่ต้องการซึ่งหงุด 4 บล็อกที่เป็นไปได้ ส่วนบิตที่ 4-11 ใช้ระบุ Index ซึ่งอ้างໄດ้ 256 บล็อกและบิตที่ 12-31 ใช้เก็บ Tag

ในการทำงาน จะทำการตั้ง Index ออกมานี้เพื่อที่ไปยังแทรฟที่ต้องการ จากนั้นจะตรวจสอบค่า Tag จำนวน 20 บิตภายในแควนน์ด้วยตัวเปรียบเทียบ ว่าตรงกันหรือไม่ ถ้าตรง Tag จะให้สัญญาณ Hit=1 และข้อมูลจำนวน 32 บิต จะถูกดึงออกมาจากไฟล์ Data ในแควนน์เดียวกันทั้งหมดออกมานะ 4 บล็อก และใช้ Block Offset เป็นตัวเลือกว่าจะให้สัญญาณจากบล็อกใดออกมานะเป็นเอาท์พุท จากนั้นจะนำเอา Offset ไประบุในต่อไปครั้งหนึ่ง

สำหรับตัวอย่างการ Reference 0, 1, 2, 3, 4, 3, 4, 15 จะเห็นว่าจำนวนครั้งที่ Hit จะมากขึ้น จะได้อัตราการ Hit $4/8 = 0.5$

0 miss		
00	Mem(1)	Mem(0)

1 hit		
00	Mem(1)	Mem(0)

2 miss		
00	Mem(1)	Mem(0)
00	Mem(3)	Mem(2)

4 miss		
01	Mem(5)	Mem(4)
00	Mem(3)	Mem(2)

3 hit		
01	Mem(5)	Mem(4)
00	Mem(3)	Mem(2)

4 hit		
01	Mem(5)	Mem(4)
00	Mem(3)	Mem(2)

15 miss		
01	Mem(5)	Mem(4)
11	Mem(15)	Mem(14)

Set Associative Addressing

จำนวนบล็อกหรือจำนวน Cache Line ในหนึ่งเขตกำหนดให้เป็น k เรียกว่า k -Way Associative ในบล็อกในเขตหนึ่งๆ จะแบ่งเป็น k กลุ่มที่สามารถเข้าถึงได้พร้อมกัน ดังนั้น บล็อกของหน่วยความจำหนึ่งๆ จะสัมพันธ์กับกลุ่มเดียวกันในเขตหนึ่งๆ ก็ได้

ตั้งนั้น Set Associative จึงเป็นรูปที่แบบหัวไปและสำหรับ Direct Mapping จะเป็น Set Associative แบบ $K=1$ และถ้าเป็น Fully Associative ให้ K เท่ากับจำนวนบล็อกในแคชทั้งหมด และ $S_1 = 1$

ในตัวอย่างของการอ้างถึงบล็อก 0, 4, 0, 4 ... ถ้าใช้แบบ 2-Way Set Associative

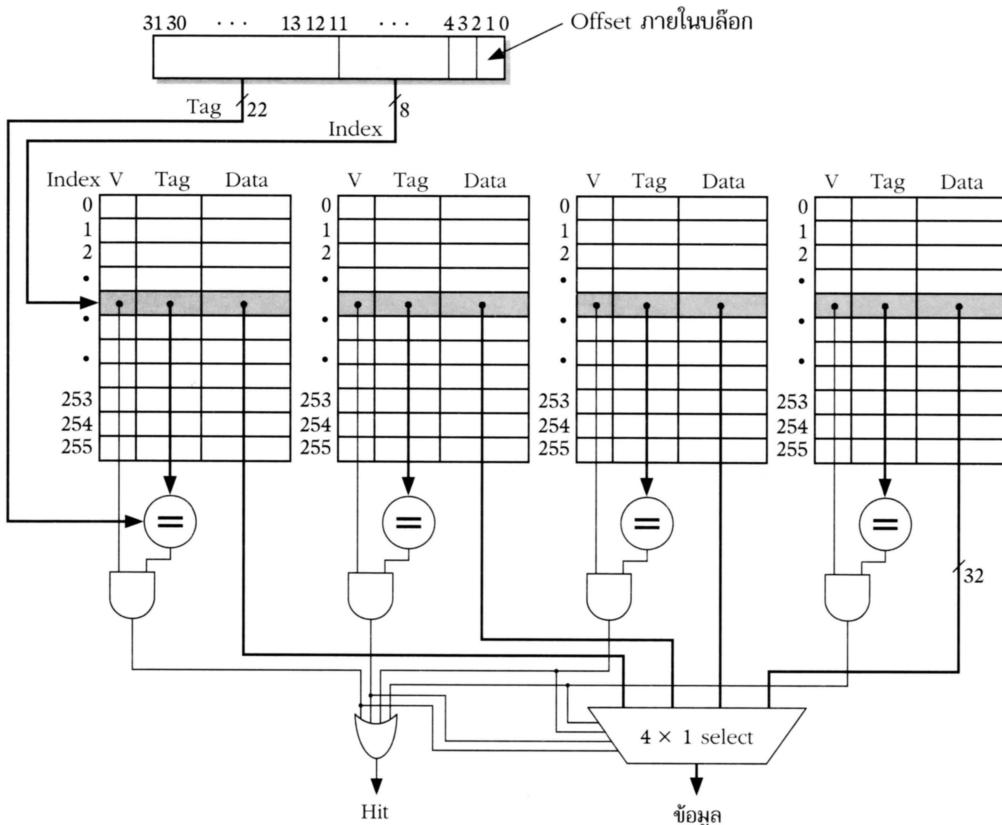
0 miss	
000	Mem(1)

4 miss	
000	Mem(1)
010	Mem(4)

0 hit	
000	Mem(1)
010	Mem(4)

4 hit	
000	Mem(1)
010	Mem(4)

จะทำให้เกิดการ Miss เพียง 2 ครั้งแรกเท่านั้น เพราะเขตสามารถเก็บบล็อกยอดเดรส์ที่ใช้ในเวลาหนึ่งๆ หรือเก็บ Working Set ได้ทั้งหมด



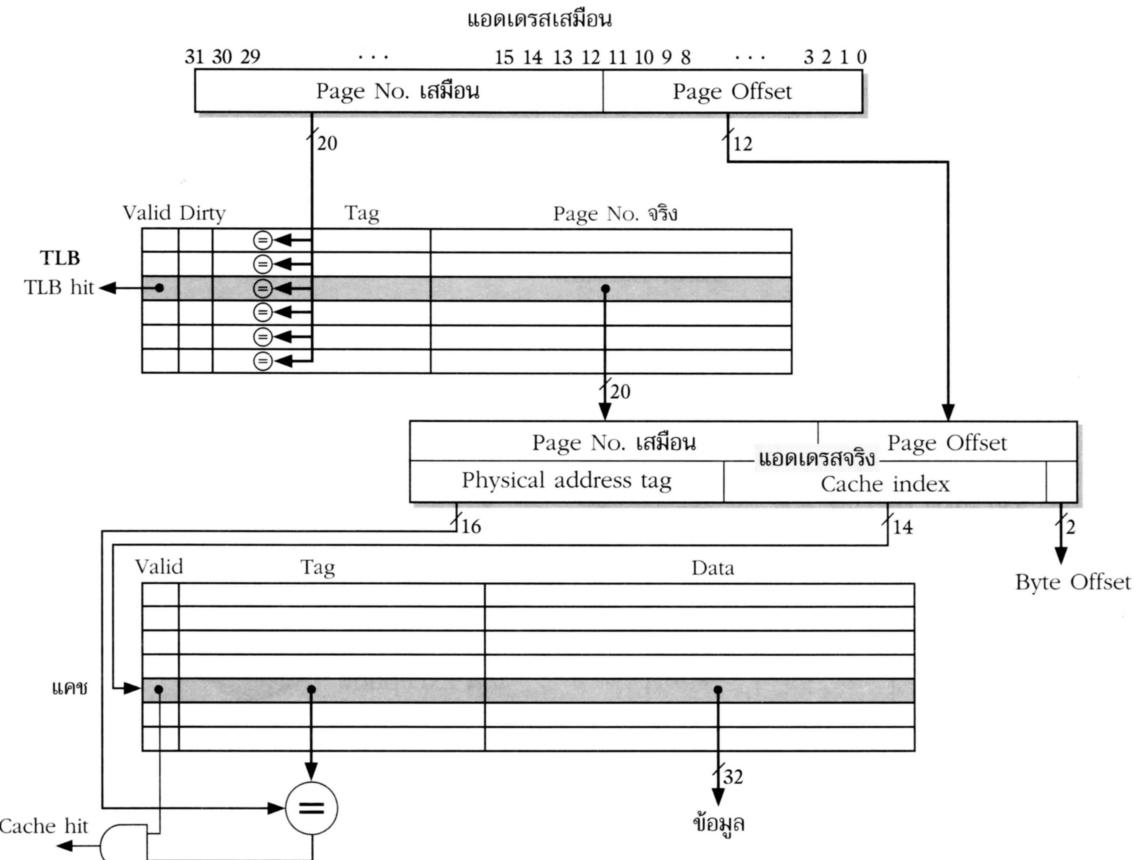
รูปที่ 9.38 ตัวอย่าง 4-Way Set Associative

รูปที่ 9.38 เป็นตัวอย่างของ 4-Way Set Associative ให้ขนาดของแคชทั้งหมดมีจำนวนเวิร์ดเป็น 1024 และ กำหนด 1 เวิร์ดเท่ากับ 32 บิต และ Cache Line เก็บข้อมูลขนาด 1 เวิร์ด จึงใช้ขนาด 2 ไบต์เก็บ Displacement มีทั้งหมด 256 เซต แบ่งเป็น 4 กลุ่ม ($k=4$) จึงใช้ Set Index ขนาด 8 บิต และทำให้ Tag มีขนาด $32-8-2 = 22$ บิต

สำหรับแอดเดรสขนาด 32 บิตจะใช้บิต 0-1 เพื่อระบุในบล็อกนั้น เนื่องจากหนึ่งบล็อกมีเพียง 4 ไบต์ ส่วนบิตที่ 2-9 ใช้ระบุ Index ซึ่งอ้างได้ 256 บล็อก และใช้บิตที่ 10-31 ใช้เก็บ Tag

ในการทำงาน จะทำการดึง Index ออกมานะเพื่อซื้อไปยังแพรที่ต้องการ จากนั้นจะตรวจสอบค่า Tag จำนวน 20 บิตภายในแพรนั้นด้วยตัวเปรียบเทียบ ในการตรวจสอบนี้จะใช้ตัวเปรียบเทียบ 4 ตัวทำการตรวจสอบ Tag พร้อมกันจากทั้ง 4 Way ว่าตรงกับที่ระบุไว้ในแอดเดรสหรือไม่ ถ้าตรง

Tag กับ Tag ตัวใดตัวหนึ่ง จะให้สัญญาณ Hit=1 และข้อมูลจำนวน 32 บิต จะถูกดึงออกมาจากพิล์ด Data ในแรมเดียวกันทั้งหมดถ้าหาก 4 บล็อก และใช้สัญญาณ Hit เป็นตัวเลือกว่าจะให้ข้อมูลจากบล็อกใดออกมาเป็นเอาท์พุท จากนั้นจะนำเอา Offset ไประบุใบ tert ที่ต้องการในบล็อกอีกครั้งหนึ่ง

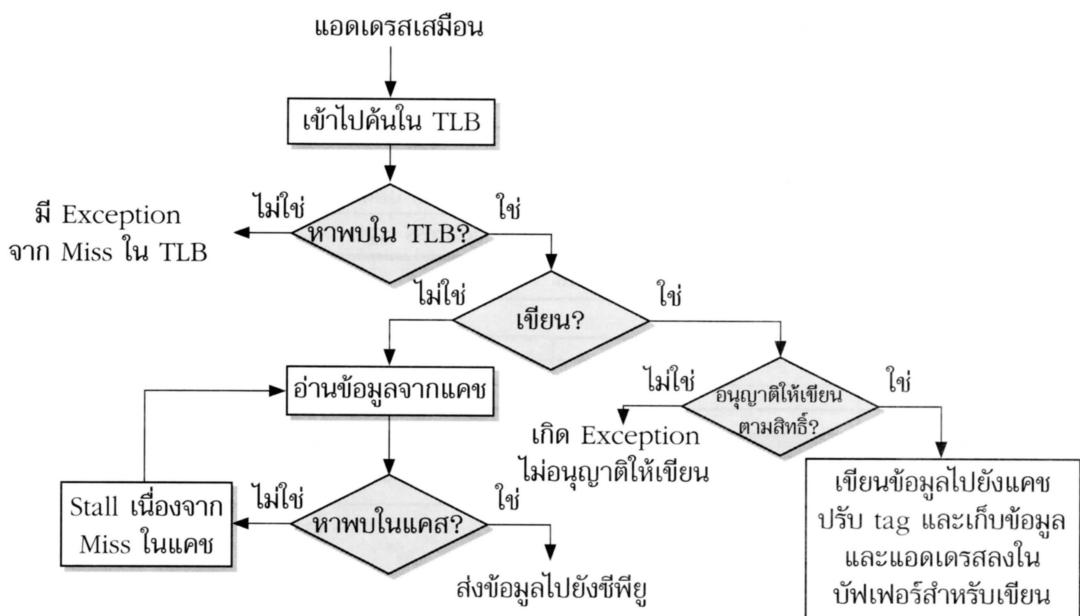


รูปที่ 9.39 เมื่อนำหน่วยความจำเสมือนรวมกับแคช

รูปที่ 9.39 แสดงการรวมระบบหน่วยความจำเสมือนและแคชเข้าด้วยกัน จากแอดเดรสเสมือนจะถูกแปลงค่าและไปค้นหา y ทั้ง TLB ไปทาง Page No. จริงจาก TLB หลังจากการ Hit จากนั้นจะได้แอดเดรสจังวิชและนำไปค้นตามขั้นตอนของแคช ต่อไป

ตารางนี้แสดงกรณีต่างๆ ของการหาใน TLB การหาในแคชและการหาในหน่วยความจำเสมือนกรณีมีการ Miss ใน TLB จะอาจจะหมายถึงการหาพบในแคชซึ่งจะทำการอ่านแอดเดรสมาเก็บไว้ใน TLB แต่ว่าถ้าค้นหาไม่พบในแคช ก็หมายถึงต้องไปนำเอาข้อมูลที่ต้องการมาจากระดับล่างต่อไป

Cache	TLB	หน่วยความจำเสมือน	เป็นไปได้หรือไม่?
Miss	Hit	Hit	ได้แม้ว่า Page table ไม่มีการตรวจสอบ Hit ใน TLB
Hit	Miss	Hit	เมื่อ Miss ใน TLB แต่ Entry นั้นอยู่ใน Page table และจะถูกนำเข้าสู่แคช
Miss	Miss	Hit	เมื่อ Miss ใน TLB แต่ Entry นั้นอยู่ใน Page table
Miss	Hit	Miss	เป็นไปได้



รูปที่ 9.40 Flow การทำงานของการค้นหาใน TLB และแคลช

ในขั้นตอนการทำงาน จะทำการตรวจสอบจาก TLB ก่อนว่าหากพบใน TLB หรือไม่ หากนั้นถ้าพบจะอ่านค่าและเดรสจังก์ ก็เอ้าและเดรสมาร์ก TLB และพิจารณาว่าเป็นการเขียนหรืออ่านถ้าเป็นการอ่านจะไปตรวจสอบในแคลชพบว่าเป็นการ Hit หรือ Miss ถ้าเป็นการ Hit จะส่งข้อมูลนั้นไปบังทีพี่ผู้ต่อไป

ตัวอย่างการคำนวณประสิทธิภาพ

ลองพิจารณาผลของแคชในเชิงประสิทธิภาพ

สำหรับชีพียุคปัจจุบัน $CPI_{ideal} = 2$ และมี Miss Penalty = 100 ไซเกิล พิจารณาโปรแกรมที่ประกอบด้วย คำสั่ง Load/Store 36% และมีอัตราการ Miss ของแคชคำสั่งเป็น 2% อัตราการ Miss ของแคชข้อมูลเป็น 4%

Memory-Stall ใช้จำนวนไซเกิล = $2\% \times 100 + 36\% \times 4\% \times 100 = 3.44$ ไซเกิล

$$\text{ตั้งนั้น } CPI_{stalls} = 2 + 3.44 = 5.44$$

ลองพิจารณาถ้า $CPI_{ideal} = 1$, $CPI_{ideal} = 0.5$ จะพบว่าผลของ Miss Penalty ในจำนวนไซเกิลมีมากขึ้น

สมมติให้อัตราการ Miss ของคำสั่งสำหรับ Benchmark gcc เป็น 2% และอัตราการ Miss ของข้อมูลเป็น 4% ถ้าเครื่องนี้มี $CPI=2$ (เมื่อไม่พิจารณา Stall จากหน่วยความจำ) และมี Miss Penalty เท่ากับ 40 ไซเกิล โปรแกรมบนเครื่องนี้เร็วขึ้นเท่าไร ถ้าโปรแกรมไม่มีการ Miss เลย สมมติให้ความถี่ของคำสั่ง Load/Store ในโปรแกรม gcc เป็น 36%

เมื่อ I เป็น Instruction Count จะได้ Instruction Miss ใช้จำนวนไซเกิล = $I \times 2\% \times 40 = 0.80I$ ไซเกิล

Data Miss ใช้จำนวนไซเกิล = $I \times 36\% \times 4\% \times 40 = 0.58I$ ไซเกิล

Stall จากหน่วยความจำใช้จำนวนไซเกิล ทั้งหมด = $0.8I + 0.58I = 1.38I$ ไซเกิล จะได้ Stall จากหน่วยความจำเฉลี่ยต่อคำสั่ง = 1.38

CPI กรณีคิด Stall จากหน่วยความจำ = $2 + 1.38 = 3.38$

สัดส่วนของ CPI เมื่อพิจารณา Stall จากหน่วยความจำ เทียบกับกรณีที่ชีพียุคทำงานและไม่มี Stall เลยเป็น $3.38/2 = 1.69$

นั่นคือเครื่องที่ไม่มี Stall เลยจะทำงานเร็วกว่า 1.69 เท่า

จากโจทย์เดียวกัน ถ้าเราลด $CPI=1$ สัดส่วนของ CPI เมื่อพิจารณา Stall จากหน่วยความจำ กับชีพียุคปัจจุบัน = $2.38/1 = 2.38$ เท่า

ดังนั้น เวลาที่ใช้ทำงานในกรณีของการพิจารณา Stall จะเพิ่มจาก $1.38/3.38 = 41\%$ เป็น $1.38/2.38 = 58\%$

ในกรณีเดียวกัน ถ้าเราเพิ่มอัตราสัญญาณนาฬิกาเป็นสองเท่า เครื่องจะทำงานเร็วขึ้นอย่างไร ถ้าอัตราสัญญาณนาฬิกาเร็วเป็นสองเท่า Miss Penalty ที่ใช้จะเพิ่มจาก 40 เป็น 80 ไซเกล ดังนั้น Miss ในจำนวนไซเกลต่อคำสั่ง $= (2\% \times 80) + 36\% \times (4\% \times 80) = 2.75$

โปรแกรมเครื่องใหม่นี้จะมี CPI = $2 + 2.75 = 4.75$

ซึ่งจะทำให้เร็วกว่าโดยพิจารณาจาก

$$\frac{\text{ExectionTime}_{\text{SlowClock}}}{\text{ExectionTime}_{\text{FastClock}}} \\ = \frac{\frac{\text{IC} \times \text{CPI}_{\text{SlowClock}} \times \text{ClockCycle}}{\text{IC} \times \text{CPI}_{\text{FastClock}} \times \frac{\text{ClockCycle}}{2}}}{\frac{3.38}{4.75 \times \frac{1}{2}}} = 1.41$$

จะเห็นว่าเมื่อเพิ่มอัตราสัญญาณนาฬิกาจะทำให้ผลของ Miss Penalty มากขึ้น และเมื่อลด CPI แบบที่ไม่คิดผลของหน่วยความจำเลย ผลของ Miss Penalty ก็จะมากขึ้นเช่นกัน

การเพิ่มดีกรีของ Associativity ก็จะมีผลต่อขนาดของที่เก็บ Tag เช่นกัน พิจารณากรณีรูปแบบแอดดิชันนี้

สมมติให้แอดดิชัน มีบล็อกทั้งหมด 4K ขนาดบล็อกและเดรสนานาด 32 บิต (4 เวิร์ด) จงหาจำนวน Set และจำนวนบิตที่ใช้สำหรับ Tag ทั้งหมดที่สำหรับกรณีรูปแบบ Direct-Mapped, 2-Way, 4-Way, Fully-Set Associative

- กรณี Direct-Mapped: สำหรับบล็อกจำนวน 4K จะใช้ 12 บิต สำหรับ Set Index ดังนั้น Tag จะใช้บิต = $28-12 = 16$ บิต จำนวนบิตทั้งหมดสำหรับหน่วยความจำ Tag เป็น $16 \times 4K = 64K$ บิต

- กรณี 2-Way: สำหรับล็อกจำนวน 4K แบ่งเป็น 2 เขต แต่ละเขตมีล็อกจำนวน 2K จะใช้ 11 บิต สำหรับ Set Index และ Tag ขนาด = $28-11 = 17$ บิต จำนวนทั้งหมดสำหรับหน่วยความจำ Tag เป็น $17 \times 2 \times 2K = 68K$ บิต
 - กรณี 4-Way จะแบ่งเป็น 4 เขต แต่ละเขตมีจำนวนล็อกเป็น 1K ใช้ 11 บิต สำหรับ Set Index และใช้ Tag ขนาด = $28-10 = 18$ บิต จำนวนทั้งหมดสำหรับหน่วยความจำ Tag เป็น $18 \times 4 \times 1K = 72K$ บิต
 - กรณี Fully Set Associative: จะใช้ 1 เขตที่มีล็อกจำนวน 4K ดังนั้นจะใช้ Tag ขนาด 28 บิต จำนวนทั้งหมดสำหรับหน่วยความจำ Tag เป็น $28 \times 4K \times 1 = 112K$ บิต
- จากการคำนวณถ้าใช้รูปที่แบบ Fully Set Associative จะใช้พื้นที่เก็บ Tag มากที่สุด

■ 9.6 ปัจจัยการออกแบบcache

ในการวัดประสิทธิภาพcache โดยทั่วไปเราจะใช้ Hit Ratio จากสมการข้างล่าง

$$t_A = t_{A_1} + (1 - H)t_{A_2}$$

โดยที่ t_A คือ Average Access Time, t_{A_1} คือ เวลาในการเข้าถึงข้อมูลในcache และ t_{A_2} เป็นเวลาในการเข้าถึงข้อมูลในหน่วยความจำหลัก

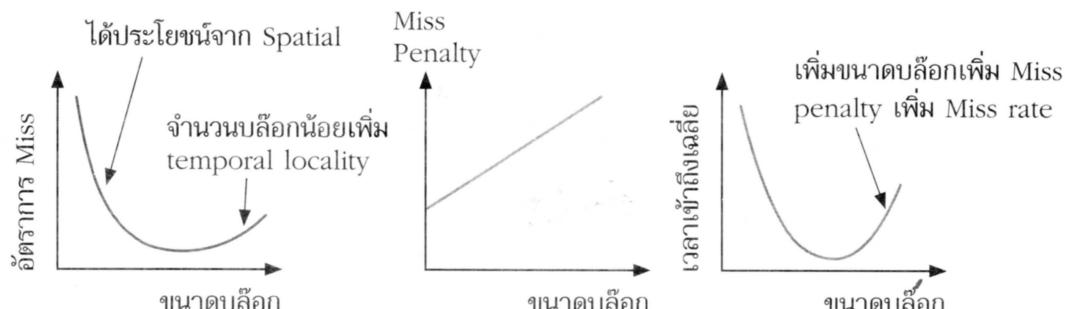
สำหรับ K-Way Associative ขนาดของcache คือ kS_1p_1 โดยที่ K คือจำนวน Way (ซึ่งเป็นจำนวนนับล็อกต่อหนึ่งเขต) S_1 คือ จำนวนเขตและ p_1 คือขนาดของบล็อก (Cache Line)

จุดมุ่งหมายในการออกแบบcache คือ เพื่อให้ได้cacheที่มีประสิทธิภาพที่ดี ประหยัดค่าใช้จ่าย (ในแง่ของขนาดของcacheที่ใช้เพื่อระบายความร้อนจะมีราคาแพง) ในส่วนของประสิทธิภาพทั้งนี้ ขึ้นอยู่กับงานประยุกต์ที่ใช้อยู่ เพราะต้องดูจากการอ้างถึงตำแหน่งต่างๆ ในโปรแกรมนั้นๆ ว่าเป็นอย่างไร ถ้าเพิ่มปัจจัยต่างๆ เช่น k, S_1 หรือ p_1 ก็จะทำให้มีโอกาสเพิ่มจำนวนครั้งของการ Hit แต่ว่าก็จะทำให้ขนาดของcacheเพิ่มขึ้นด้วย นอกจากนี้การเพิ่มก็มีขอบเขต เพราะเมื่อเพิ่มไปมากๆ แล้ว จะเกิดพื้นที่ที่ไม่ได้ใช้ประโยชน์ทำให้เสียพื้นที่และเป็นการสิ้นเปลืองค่าใช้จ่าย ขั้นตอนของการออกแบบcache จึงสรุปได้ดังนี้

- เลือกขนาดของบล็อก p อย่างเหมาะสมโดยมากจะดูจากความกว้างของบล็อกที่อยู่ระหว่างชีพีย์และหน่วยความจำหลัก
- เลือกโปรแกรมตัวอย่างมาทดลองทำการอ้างถึงแอดเดรส เพื่อคาดคะเนการอ้างถึงแอดเดรสต่างๆ ระหว่างการทำงานของโปรแกรม
- จำลองระบบหน่วยความจำโดยเปลี่ยนค่าขนาดของ Set S_1 , Associative Degree K และลอง Replacement Policy ต่างๆ กัน
- นำข้อมูลที่ได้จากการทดลองของแคชแบบต่างๆ มาแสดงเป็นกราฟ และหาข้อดี ข้อเสียระหว่างค่าใช้จ่าย และประสิทธิภาพของแคชแบบเหล่านั้น รูปที่แบบของแคชที่เหมาะสมซึ่งมีประสิทธิภาพที่ยอมรับได้ภายในตัวค่าใช้จ่ายที่กำหนด

ในการออกแบบ จึงต้องพิจารณาหลายๆ ปัจจัย ในการฟังก์ชันล่างแสดงผลของขนาดของบล็อกต่ออัตราการ Miss โดยที่ว่าไปถ้าขนาดหั้งหมดของแคชเท่าเดิม เมื่อเพิ่มขนาดของบล็อกจะลดอัตราการ Miss เพราะเป็นการเพิ่ม Spatial Locality อย่างไรก็ได้ ถ้าขนาดของแคชเล็กเกินไป และบล็อกมีขนาดใหญ่จะทำให้จำนวนบล็อกมีน้อย ซึ่งจะไม่สามารถบรรจุบล็อกข้อมูลที่ต้องการหั้งหมดก็เป็นได้ ดังนั้น ถ้าขนาดของแคชใหญ่เพียงพอ ควรจะแบ่งบล็อกให้มีขนาดพอดีที่ทำให้จำนวนบล็อกที่มีอยู่เพียงพอที่จะบรรจุ Working set บล็อกต่างๆ ได้ และได้ประโยชน์จากพื้นที่ที่ติดกันด้วย

การเพิ่มขนาดของบล็อกยังจะมีข้อเสียหลายๆ อย่างเช่น จะทำให้มี Miss Penalty มากขึ้น จะลดเวลาในการเข้าถึงเฉลี่ยไปส่วนหนึ่ง แต่เมื่อขนาดของบล็อกใหญ่มากไปจะเป็นการเพิ่มเวลาเข้าถึงเฉลี่ยไป ดังรูปที่ 9.41 เป็นการสรุปผลของขนาดของบล็อกต่ออัตราการ Miss ผลต่อ Miss Penalty และผลต่อเวลาการเข้าถึงข้อมูล



รูปที่ 9.41 Trade-off ของการเพิ่มขนาดของบล็อกต่อปัจจัยต่างๆ

นอกจากนี้สำหรับล็อกแบบหลายๆ เวิร์ดเมื่อมีการ Miss เกิดขึ้นจะต้องพิจารณาว่าทำอย่างไรจะทำให้มี Miss Penalty น้อยที่สุด เทคนิคต่างๆ ที่จะนำมาใช้ จะได้กล่าวไว้ในบทถัดไป ดังนี้

กรณี Read Miss (ทั้ง I\$ และ D\$) จะต้องใช้วิธี Early Restart โดยจะ Resume คำสั่งนั้นทันทีที่เวิร์ดที่ต้องการมาถึง ไม่ต้องรอข้อมูลทั้งบล็อก หรือวิธี Request Word First โดยการให้ส่งเวิร์ดที่ต้องการเข้ามาก่อน และระหว่างนั้นก็ทำการอ่านข้อมูลต่อไปให้เต็มบล็อก หรือใช้แคชแบบ Non-Blocking เพื่อให้ชีพียุคทำงานต่อไปได้แม้ว่าต้องรอในส่วนของแคชในกรณี Miss จากการเขียนจะไม่สามารถใช้เทคนิคอะไรช่วยได้ นอกจากต้อง Stall การทำงาน

เช่นเดียวกันถ้าเพิ่มตีกรีของ Associativity จะเป็นการลดอัตราการ Miss แต่ว่าการเพิ่มตีกรีมากเกินไปจะไม่ให้ผลดีขึ้นเท่าที่ควร เพราะว่าอาจจะมีบางส่วนของแคชไม่ได้ถูกใช้งานได้ สำหรับขนาดของตีกรีจะต้องเหมาะสมกับขนาดของแคช ถ้าน้ำหนักของแคชเล็กมาก ตีกรีที่มากก็จะไม่มีประโยชน์อะไรเนื่องจากมีปัญหาจาก Capacity Miss เมื่อน้ำหนักของแคชเหมาะสม จึงควรเลือกตีกรีที่เหมาะสม ไม่น้อยเกินไป ถ้าน้อยเกินไปจะทำให้ไม่สามารถบรรจุบล็อกแอดเดรสที่อาจจะเข้ากันทั้งหมดได้ในเวลาหนึ่ง และถ้ามากเกินไปก็จะเป็นการทำให้ฮาร์ดแวร์ซับข้อนอีกด้วย เป็นการเพิ่มค่าใช้จ่าย และรอบเวลาสัญญาณนาฬิกาในการค้นหาข้อมูล

แต่อีกวิธีหนึ่งที่ยังไม่ได้กล่าวถึงในการลดอัตราการ Miss ก็คือการเพิ่มระดับของแคช โดยทั่วไปในปัจจุบันนี้เราใช้แคช หลายระดับตั้งแต่ L1, L2, L3 โดยใช้ L1 เป็นแคชแบบแยก กล่าวคือแยกกันระหว่างคำสั่งและข้อมูล L2, L3 เป็นแคชเดียวที่เก็บรวมกันทั้งคำสั่งและข้อมูล

สมมติให้ $CPI_{ideal} = 2$, Miss Penalty (กรณีไปหาอย่างหน่วยความจำหลัก) = 100 ไซเกิล มีคำสั่ง Load/Stores อยู่ 36% โดยมีอัตราการ Miss ของ $L1(I) = 2\%$ มีอัตราการ Miss ของ $L1(D) = 4\%$ ถ้าเพิ่มแคช L2 แบบรวมที่มี Miss Penalty = 25 ไซเกิล และมีอัตราการ Miss เป็น 0.5% จะได้ว่า

$$\begin{aligned} CPI_{stalls} &= 2 + 0.02 \times 25 + 0.36 \times 0.04 \times 25 + 0.005 \times 100 + 0.36 \times 0.005 \times 100 \\ &= 3.54 \end{aligned}$$

เปรียบเทียบกับค่า 5.44 เมื่อไม่ได้ใช้ L2

สำหรับเครื่องที่มี CPI เท่ากับ 1.0 มีความถี่ 500 Mhz (รอบสัญญาณนาฬิกาเท่ากับ 2 นาโนวินาที) มี อัตราการ Miss = 5% มีเวลาการเข้าถึง DRAM = 200 นาโนวินาที เครื่องจะเร็ว

ขั้นเท่าไรถ้าจะเพิ่มแคช L2 ที่มีเวลาการเข้าถึง = 20 นาโนวินาที สำหรับทั้งกรณี Hit หรือ Miss และมีขนาดใหญ่ที่จะทำให้ลดอัตราการ Miss ไปยังหน่วยความจำเหลือเป็น 2%

- Miss Penalty ไปยังหน่วยความจำ 200 นาโนวินาที/(2 นาโนวินาทีต่อไซเกิล) = 100 ไซเกิล
- CPI รวม = Base CPI + Stall เคลื่อนของหน่วยความจำต่อคำสั่ง ตั้งนั้น CPI รวม = $1.0 + 5\% \times 100 = 6.0$

สำหรับการเพิ่มระดับ L2 Miss Penalty ในการเข้าถึงระดับ L2 เป็น 20 นาโนวินาที/(2 นาโนวินาทีต่อไซเกิล) = 10 ไซเกิล

- CPI รวม = Base CPI + Stall เคลื่อนของหน่วยความจำระดับ L1 ต่อคำสั่ง + Stall เคลื่อนของหน่วยความจำระดับ L2 ต่อคำสั่ง
- CPI รวม = $1.0 + 5\% \times 10 + 2\% \times 100 = 3.5$

ตั้งนั้นเครื่องแบบที่มี L2 จะมีเร็วกว่าอยู่ $6.0/3.5 = 1.7$ เท่า

ในการออกแบบลักษณะแคชหลายระดับนั้น ระดับ L1 ควรจะให้มีขนาดบล็อกที่เล็ก เพื่อลดเวลาในการ Hit ระดับ L2 ควรจะออกแบบเพื่อให้ลด Miss Penalty และลดอัตราการ Miss เมื่อมีระดับ L2 ควรจะทำให้ Miss Penalty ของระดับ L1 ลดลง แต่อาจจะมีอัตราการ Miss สูง สำหรับระดับ L2 ควรจะเน้นที่การลดอัตราการ Miss มากกว่าการลดเวลาการ Hit

ตารางข้างล่างเป็นข้อมูลของ L1, L2 ที่มักจะใช้ เมื่อปี ค.ศ. 2004 จาก Hennessy (2005)

ปัจจัย	L1	L2
ขนาด (จำนวนบล็อก)	250 – 2,000	4,000 – 250,000
ขนาดทั้งหมด (KB)	16 – 64	500 – 8,000
ขนาดบล็อก (ไบต์)	32 – 64	32 – 128
Miss Penalty (ไซเกิล)	10 – 25	100 – 1,000
Miss rate (ค่า Global Miss Rate สำหรับแคช L2)	2% – 5%	0.1% – 2%

ตารางข้างล่างเปรียบเทียบการออกแบบหน่วยความจำหลายชั้น รูปแบบสถาปัตยกรรม

ลักษณะ	Intel P4	AMD Opteron	ARM Cortex A8	PowerPC 7477A ⁶	Intel Core Microarchitecture
รูปแบบ L1	แคชแยกคำสั่งและข้อมูล	แคชแยกคำสั่งและข้อมูล	แคชแยกคำสั่งและข้อมูล	แคชแยกคำสั่งและข้อมูล	แคชแยกคำสั่งและข้อมูล
ขนาดของ L1	8KB สำหรับทั้งแคชคำสั่งและข้อมูล D\$ และ 96KB แคชคำสั่ง	64KB สำหรับทั้งแคชคำสั่งและข้อมูล	32 KB ทั้งแคชคำสั่งและข้อมูล	32 KB ทั้งแคชคำสั่งและข้อมูล	32 KB ทั้งแคชคำสั่งและข้อมูล
ขนาดของบล็อก L1	64 ไบต์	64 ไบต์	64 ไบต์	32 ไบต์	64 ไบต์
L1 Associativity	4-way	2-way	4-way	8-Way	8-Way
L1 Replacement	~LRU	LRU	Random	Pseudo LRU/ AltiVec LRU	~LRU
รูปแบบการเขียน L1 (แคชข้อมูล)	write-through	write-back	write-back, no-write allocate	write-back (write-through)	write-back (write-through)
รูปแบบ L2	Unified	Unified	Unified	Unified	Unified
ขนาด L2	512KB	1024KB (1MB)	64KB(2MB)	512KB	2.4 MB
ขนาดบล็อก L2	128 ไบต์	64 ไบต์	64 ไบต์	64 ไบต์	64 ไบต์
L2 Associativity	8-way	16-way	8-way	8-way	8-way (16-way)

⁶ มีแคช 3 ระดับเท่านี้ยกเว้น Intel Core

ตารางข้างล่างเปรียบเทียบการออกแบบหน่วยความจำหลายๆ รูปแบบสถาปัตยกรรม (ต่อ)

ลักษณะ	Intel P4	AMD Opteron	ARM Cortex A8	PowerPC 7477A ⁶	Intel Core Microarchitecture
L2 Replacement	~LRU	~LRU	Pseudo Random	~LRU	~LRU
รูปแบบการเขียน L2	write-back	write-back	write-through, write-back, write-allocate	write-back (write-through)	write-back

■ 9.7 สรุป

บทนี้ได้กล่าวถึงการจัดการหน่วยความจำในเครื่องคอมพิวเตอร์ ตั้งแต่ความรู้พื้นฐานในการแบ่งระดับของหน่วยความจำ ประเภทของหน่วยความจำแบบต่าง ๆ แบบเข้าถึงแบบลำดับ แบบเข้าถึงแบบสุ่ม แบบ ROM เป็นต้น ปัจจัยในการวัดประสิทธิภาพและราคาของหน่วยความจำ ได้แก่ เวลาในการเข้าถึง รอบสัญญาณนาฬิกา แบนด์วิดธ์ ราคาต่อบิต เป็นต้น นอกจากนี้ได้กล่าวถึงเทคนิคที่ทำให้การเข้าถึงหน่วยความจำเร็วขึ้น ทั้งการขยายบัส และการทำ Interleave จากนั้นได้กล่าวถึงระบบหน่วยความจำ หน่วยความจำเสมือน การจัดระบบ Page และ Segment ประโยชน์ของการใช้ Locality และประเภททั้งแบบเชิงพื้นที่และเวลา Spatial และ Temporal Locality ประสิทธิภาพของการใช้พื้นที่ในหน่วยความจำ วิธีการจัดสรรการใช้พื้นที่ในหน่วยความจำ ทั้งแบบ Preemptive และ Non-Preemptive ตัวอย่างของ Replacement Policy แบบ OPT, FIFO และ LRU และได้กล่าวถึงแบบ Direct-Mapped และ Set Associative ทั้งในแง่ของฮาร์ดแวร์และปัจจัยที่มีผลต่อการพิจารณาในการเลือกใช้แคชที่มีลักษณะที่เหมาะสมกับระบบทั้งด้านค่าใช้จ่ายและประสิทธิภาพ

คำถ้ามก้ายบท

1. จงอธิบายความสัมพันธ์ของ ราคา ขนาด และความเร็วของหน่วยความจำระดับชั้น
2. ยกตัวอย่างของโ diket ที่ทำให้เห็นประโยชน์ของ Locality เขิงพื้นที่
3. จงอธิบายความหมายของแต่ละพิล๊ด และบอกผลต่อขนาดของพิล๊ด เช่น Tag, Set, Displacement
4. จงบอกรความสัมพันธ์ของ Associativity Degree, ขนาดของบล็อก และจำนวนเขต
5. จงอธิบายการเปลี่ยนใน Data Path ของ MIPS เมื่อพิจารณาแคช
6. Page Replacement Policy มีผลต่อการอุกแบบแคชอย่างไร
7. พิจารณาลำดับการอ้างถึง Page ดังนี้

0, 2, 3, 1, 2, 4, 2, 5, 7

สมมติให้ เริ่มต้นหน่วยความจำว่างเปล่า จำนวน Page ที่บรรจุได้ในหน่วยนี้เป็น 3 Page
ถ้าใช้วิธี FIFO, OPT, LRU วิธีใดจะให้จำนวน Miss ที่น้อยที่สุด
8. พิจารณาการอุกแบบแคช ถ้าให้บล็อกเดรส = 32 บิต และบล็อกมูล = 64 บิต ความจุของแคชทั้งหมดคือ 512 กิโลบิต มี Cache Line มีขนาดเท่ากับ 32 บิต
 - 8.1 เปรียบเทียบขนาดของหน่วยความจำเก็บ Tag ที่ใช้สำหรับวิธี Direct mapped, Fully Associative, 2-Way Set Associative (ให้ระบุเป็นจำนวนบิต)
 - 8.2 ถ้าใช้วิธี 2-Way Set Associative จงหาขนาดจำนวนบิตของ Tag, Set, Offset
9. เปรียบเทียบรูปแบบของหน่วยความจำแบบต่างๆ ได้แก่
 - มีความกว้างบล็อกขนาด 1 เวิร์ด เมื่อขนาดบล็อก = 2 เวิร์ด และขนาดบล็อก = 4 เวิร์ด

- มีความกว้างบล็อก 1 เวิร์ด ใช้ Fast Page Mode เมื่อขนาดบล็อก = 2 เวิร์ด และเมื่อขนาดบล็อก = 4 เวิร์ด
- Interleaved และเมื่อขนาดบล็อก = 4 เวิร์ด

เมื่อต้องการเข้าถึงหน่วยความจำ ต้องมีการ Miss และต้องหยุดไปเป็นไลน์ข้ามคราวเพื่อไปเอาข้อมูล สมมติให้ใช้เวลา 1 ไบเกิล ในการส่งแอดเดรส ใช้เวลา 30 ไบเกิลในการเข้าถึง DRAM และใช้เวลา 2 ไบเกิลในการส่งข้อมูลกลับมา จงหาแบบดีวิดอธ์ของการส่งข้อมูลสำหรับการ Miss หนึ่งครั้งสำหรับกรณีต่างๆ ข้างต้น

10. พิจารณาตัวอย่างโปรแกรมภาษาซีด้านล่าง อธิบายและยกตัวอย่าง Locality เขิงพื้นที่และเขิงเวลา จากตัวอย่างได้ดังนี้

```

while (j < 10) {
    if (x < 5)
        A[j] = B[j]+4;
        some long code
    else
        A[j] = C[j] + B[j]
        another long code
    j++;
}

```

11. พิจารณาการออกแบบระบบหน่วยความจำแบบลำดับชั้น ที่ให้ขนาดของแคชทั้งหมดเป็น 512 กิโลไบต์ มีบล็อกข้อมูลขนาด 64 บิต และมีบล็อกแอดเดรสขนาด 32 บิต กำหนดให้พิจารณาทางเลือก ดังนี้ 1. เมื่อขนาดบล็อกเป็น 512 ไบต์ และ 2. เมื่อขนาดบล็อกเป็น 256 ไบต์ และเลือกระหว่าง 4. วิธี Direct map 5. วิธี 2-Way Set Associative และ 6. วิธี 4-Way Set Associative จงคำนวณขนาดของหน่วยความจำเก็บ Tag ทั้งหมดสำหรับทางเลือกต่อไปนี้

ทางเลือกแรก 1) + 4), 1) + 5), 1) + 6)

ทางเลือกที่สอง 2) + 4), 2) + 5), 2) + 6)

ทางเลือกใดจะให้ขนาดของหน่วยความจำเก็บ Tag ที่เล็กที่สุด

12. จงพิจารณาระบบแคชขนาดเล็กที่บรรจุจำนวนล็อกได้ทั้งหมด 4 บล็อก พิจารณาการออกแบนลักษณะ Direct Mapped, 2-Way Set Associative, Fully Set Associative และถ้ามีการอ้างถึงแอดเดรส ดังนี้ 0, 4, 2, 4, 2, 3, 1, 2, 4 พิจารณา Replacement Policy ดังนี้
- LRU
- FIFO
- วิธีใดจะให้อัตราการ Hit สูงสุด เพราะเหตุใด
13. ประเภทของ Miss มีกี่ประเภทไว้บ้าง
14. Write-Allocate และ Write No Allocate ต่างกันอย่างไร

