ระบบคอมพิวเตอร์และสถาปัตยกรรม
(Computer System and Architecture)

บทที่ 8
แคชและหน่วยความจำเสมือน
(Cache & Virtual Memory)

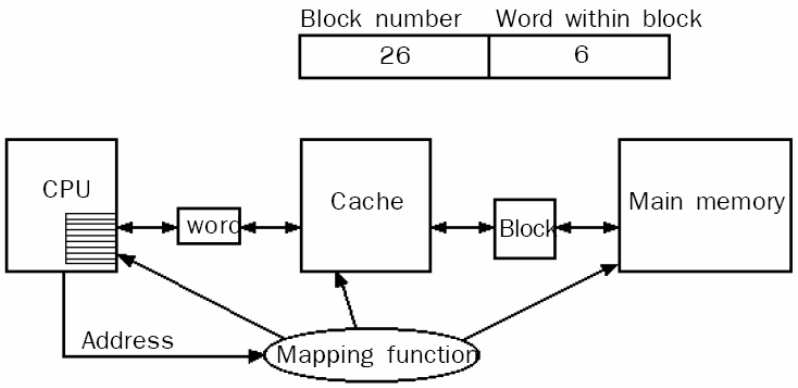
การออกแบบหน่วยความจำ

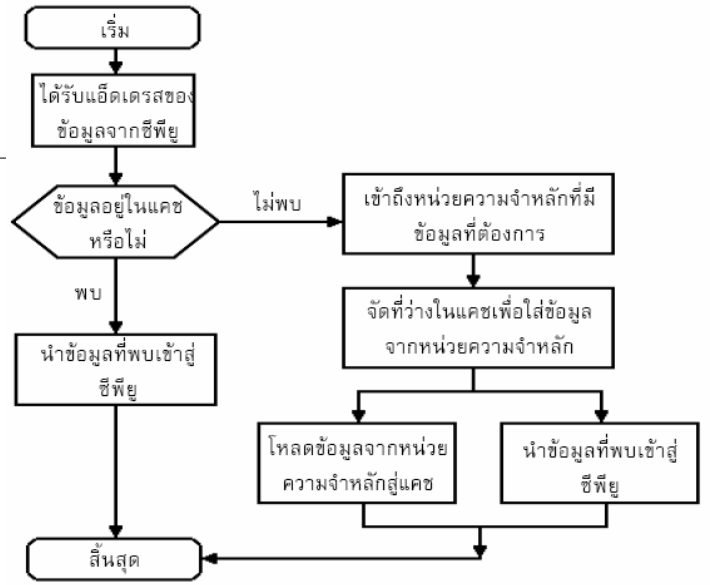
หนึ่งในปัญหาที่ยากต่อการออกแบบระบบคอมพิวเตอร์ คือการจัดการกับหน่วยความจำ หน่วยความจำของคอมพิวเตอร์ที่มีความหลากหลายและแตกต่างกันทั้งชนิด, เทคโนโลยี และความแตกต่างในด้านประสิทธิภาพและราคา และไม่มีเทคโนโลยีใดเทคโนโลยีหนึ่งที่เป็นคำตอบที่ดีที่สุดของการออกแบบหน่วยความจำในระบบคอมพิวเตอร์ ด้วยเหตุผลดังกล่าวทำให้การออกแบบหน่วยความจำในระบบคอมพิวเตอร์ที่ดีแบบหนึ่งคือ การแบ่งหน่วยความจำของคอมพิวเตอร์ออกเป็นชั้นๆ (Hierarchy) โดยสามารถแบ่งออกเป็นหน่วยความจำภายใน (Internal) ที่ซีพียูสามารถเข้าถึงข้อมูลได้โดยตรง และหน่วยความจำภายนอก (External) ที่ซีพียูเข้าถึงได้โดยผ่านไอโอโมดูล (I/O module)

แคช (Cache)

หน่วยความจำแคช (Cache) เป็นหน่วยความจำที่จัดวางไว้ระหว่างซีพียู และหน่วยความจำหลัก และโดยทั่วไปแล้วเวลาที่ใช้ในการเข้าถึงข้อมูลที่อยู่ในแคช ก็จะมีความเร็วมากกว่าเวลาที่ใช้ในการเข้าถึงข้อมูลที่อยู่ในหน่วยความจำหลักประมาณ 5-10 เท่า โดยหน่วยความจำแคชนี้ เป็นส่วนหนึ่งในลำดับชั้นของหน่วยความจำที่มีความเร็วที่สุด และมีความเร็วเกือบใกล้เคียงกับตีพียูเลยทีเดียว ความคิดพื้นฐานคือ การเก็บคำสั่ง หรือข้อมูลที่ถูกเรียกใช้งานบ่อย ๆ ลงในแคช ทำให้ค่าเฉลี่ยของการเข้าถึงข้อมูลมีค่าใกล้เคียงกับเวลาที่ใช้ในการเข้าถึงข้อมูลในแคช

แคช (Cache)





การออกแบบหน่วยความจำแคช

- ประสิทธิภาพการทำงานของแคชนั้น สามารถวัดได้จาก อัตราการพบข้อมูล หรือ **hit ratio**
- เมื่อซีพียูต้องการนำคำสั่งหรือข้อมูลเข้า และพบว่าข้อมูลดังกล่าวถูกเก็บไว้ในแคชแล้วนั้น เราจะเรียกขั้นตอนนี้ว่า “พบ” หรือ **hit**
- ถ้าไม่พบข้อมูลที่ต้องการในแคช เนื่องจากข้อมูลดังกล่าวอยู่ในหน่วยความจำหลัก เราก็จะเรียกว่า “พลาด” หรือ **miss**
- อัตราส่วนของจำนวนที่พบข้อมูล (hit) หารด้วยจำนวนครั้งที่ตัวประมวลผลทำการเรียกคำสั่ง หรือข้อมูล (hit บวก miss) ก็คืออัตราการพบข้อมูล หรือ **hit ratio** นั่นเอง

การใช้ฟังก์ชันในการเชื่อมโยงข้อมูล (Mapping)

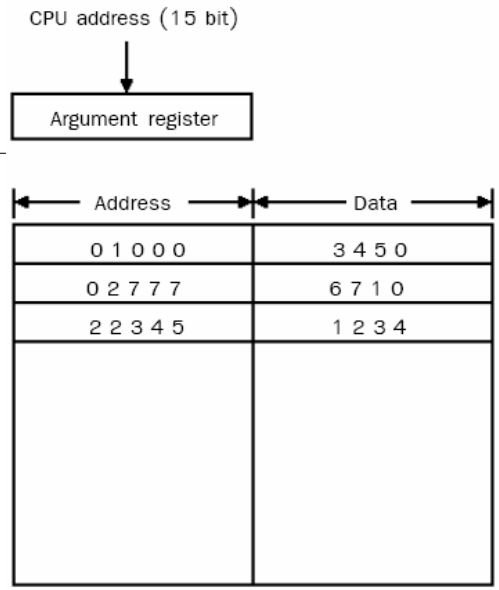
- ต้องมีกลยุทธในการวางข้อมูล (Placement) - ที่ที่เราจะนำบล็อกของข้อมูลวางลงในแคช
- ต้องมีกลยุทธในการแทนที่ข้อมูล(Replacement) - บล็อกที่เราจะนำข้อมูลมาแทนที่เมื่อไม่มีการพบข้อมูลที่ต้องการในแคช
- ต้องมีนโยบายในการอ่านและเขียนข้อมูล - วิธีจัดการ การอ่านและการเขียนข้อมูลลงในแคช ที่ขึ้นอยู่กับการพบ(hit) และพลาด (miss) ของข้อมูล

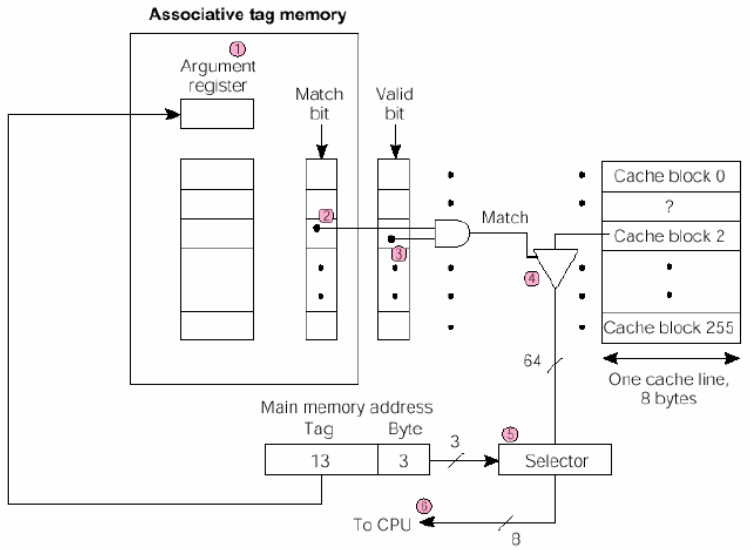
ชนิดของการเชื่อมโยงข้อมูล

- การเชื่อมโยงแบบสัมพันธ์ (Associative mapping)
- การเชื่อมโยงแบบโดยตรง (Direct mapping)
- การเชื่อมโยงแบบกลุ่มสัมพันธ์ (Block-set associative mapping)

การเชื่อมโยงแบบสัมพันธ์ (Associative mapping)

- เป็นวิธีการที่เร็วและยืดหยุ่นที่สุดของการจัดการกับหน่วยความจำแคช
- วิธีการจัดการหน่วยความจำแบบนี้ แคชจะเก็บทั้งแอ็ดเดรสและข้อมูลในหน่วยความจำ ซึ่งจะทำให้แคชสามารถเก็บข้อมูลจากเวิร์ดใดก็ได้จากหน่วยความจำหลัก
- การเชื่อมโยงแบบนี้เป็นวิธีการที่มีความยืดหยุ่นสูง และใช้ความจุของแคชอย่างเต็มที่
- มีจุดอ่อนหรือข้อเสียเช่นเดียวกัน คือเมื่อมีการอ่านข้อมูลของแคชในแต่ละครั้ง ฟังก์ชั่นนี้จะต้องทำการค้นหาแอ็ดเดรสทั้งหมดภายในแคชให้ตรงกับที่อ้างอิงถึง ซึ่งการอ่านหรือการค้นหาแอ็ดเดรสแบบลำดับ (Sequential) ซึ่งใช้เวลามาก



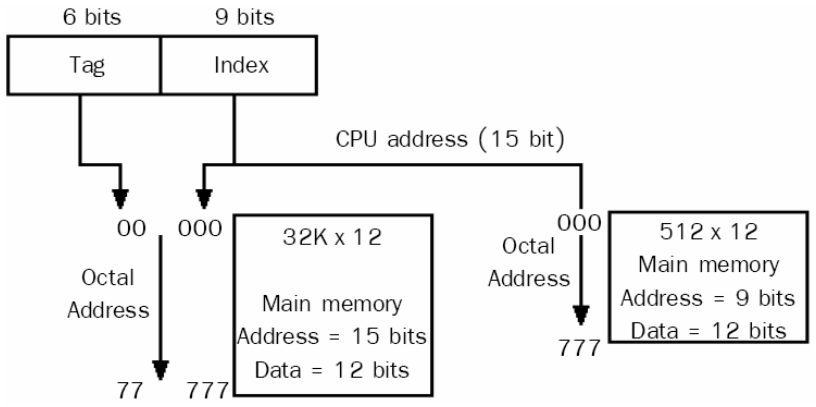


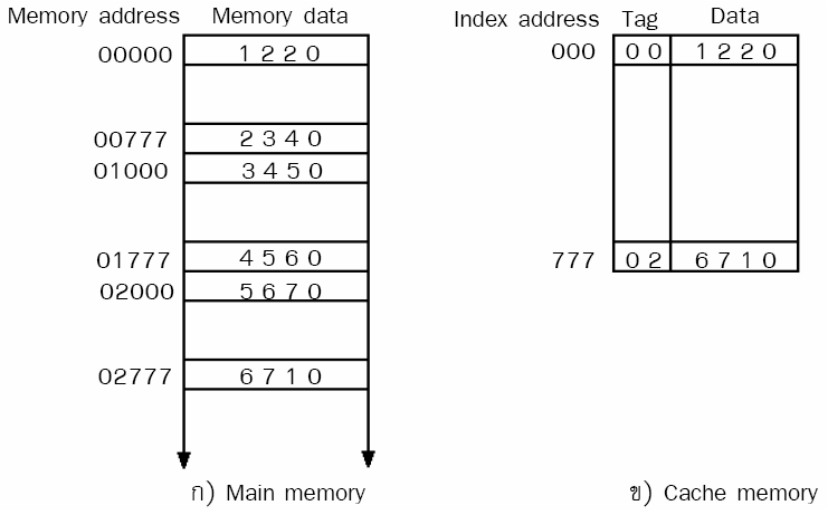
การเชื่อมโยงแบบโดยตรง (Direct mapping)

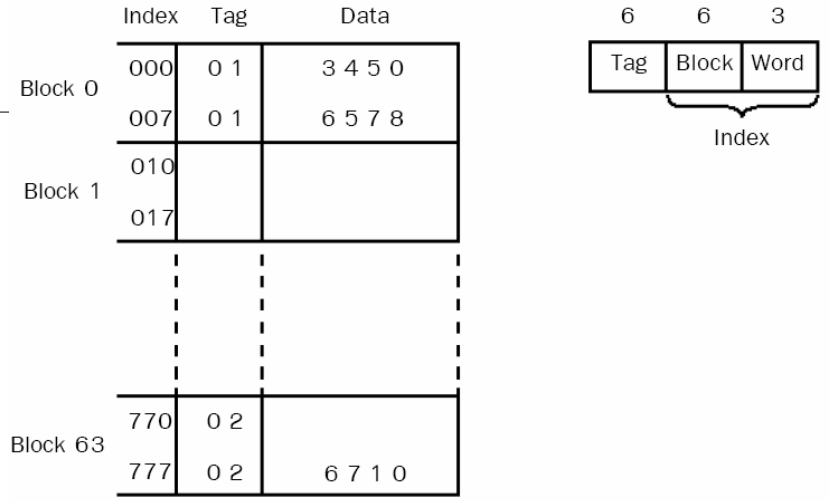
- อาจจะเรียกว่าเป็นการเชื่อมโยงแบบสุ่ม
- Address ของซีพียูที่มี 15 บิต จะแบ่งออกเป็น 2 ส่วน คือ
  - 9 บิตสำหรับเป็นอินเด็กซ์ (index)
  - 6 บิตสำหรับเป็นแท็ก (tag)
- จำนวนบิตที่อยู่ใน Index fieldจะเท่ากับจำนวนบิตที่ต้องการใช้ในการเข้าถึงหน่วยความจำแคช
- แต่ละเวิร์ดของข้อมูลในแคชจะประกอบด้วยข้อมูลจริงและ Tag
- การเข้ามาครั้งแรกบิตของ Tag จะถูกบันทึกคู่กับข้อมูลจริง

การเชื่อมโยงแบบโดยตรง (Direct mapping)

- ฟิลด์ที่เป็นIndexจะถูกนำมาใช้เป็นaddress สำหรับการเข้าถึง cache
- Tag Field ของ address ที่ซีพียูต้องการจะถูกเปรียบเทียบกับ ข้อมูลTag ที่อยู่ใน cache ถ้าข้อมูลของ Tag ทั้งสองตรงกันก็จะพบข้อมูล
- ถ้า Tag ไม่ตรงกันแสดงว่าไม่พบข้อมูล จะต้องไปหาที่ต้องการในหน่วยความจำหลักต่อไป



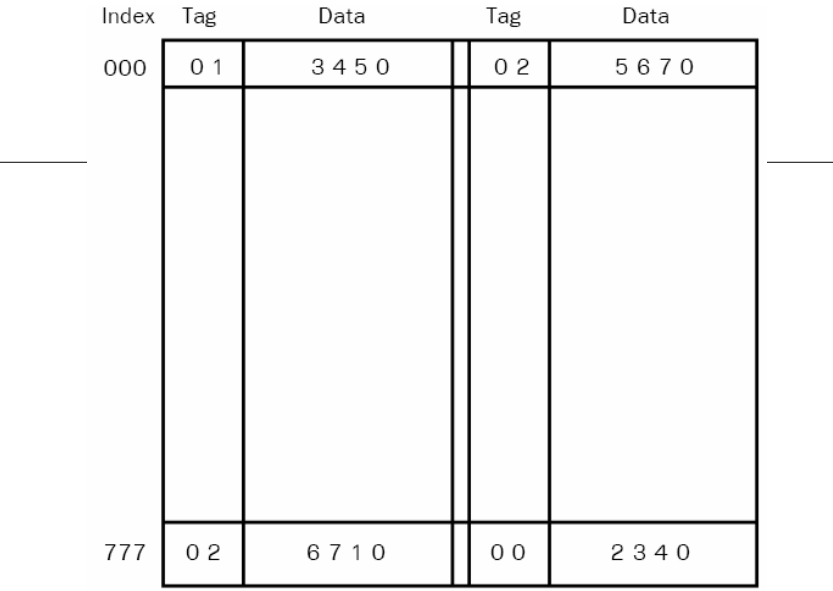




การเชื่อมโยง cache แบบโดยตรงสำหรับ Block ข้อมูล 8 เวิร์ด

การเชื่อมโยงแบบกลุ่มสัมพันธ์ (Block-set associative mapping)

- แต่ละเวิร์ดที่อยู่ในcacheสามารถบันทึกข้อมูลที่มีฟิลด์ของอินเด็กซ์เหมือนกันได้มากกว่า 1 เวิร์ดขึ้นไป
- ข้อมูลจริงของแต่ละเวิร์ดนั้นจะถูกบันทึกพร้อมกับ tag ของมัน
- จำนวนของ tag พร้อมข้อมูลในหนึ่งเวิร์ดนั้นจะถูกกำหนดเป็นกลุ่ม



วิธีการสับเปลี่ยนข้อมูล (Replacement)

- Data Blockใหม่ถูกนำเข้ามาบันทึกลงใน Cache ต้องมีการเขียนทับ Data Block เดิม
- สำหรับ cache ที่มีการเชื่อมโยงแบบโดยตรง data block ที่อยู่ในตำแหน่งของ Indexจะถูกเขียนทับ
- สำหรับcache ที่มีการเชื่อมโยงแบบสัมพันธ์และกลุ่มสัมพันธ์จะต้องมีอัลกอริทึ่ม หรือวิธีการสับเปลี่ยนข้อมูล ช่วยในการทำงาน

วิธีการสับเปลี่ยนข้อมูล (Replacement)

- วิธีการสับเปลี่ยนแบบใช้น้อยที่สุดในปัจจุบันออกไปก่อน (LRU) เป็นการสับเปลี่ยนที่เอาบล็อกของข้อมูลที่อยู่ในแคชนานที่สุดและไม่ได้ถูกเรียกใช้ในช่วงเวลาที่ผ่านมาเลยออกไปก่อน
- การสับเปลี่ยนแบบเข้ามาก่อนออกก่อน (FIFO) ซึ่งจะทำการสับเปลี่ยนบล็อกของข้อมูลที่อยู่ในแคชนานที่สุดออกไปก่อน
- การสับเปลี่ยนนำข้อมูลที่ใช้จำนวนครั้งน้อยที่สุดออกไปก่อน (LFU) เป็นการสับเปลี่ยนบล็อกของข้อมูลที่มีจำนวนครั้งของการเรียกใช้งานน้อยที่สุดออกไป

วิธีการเขียนข้อมูล (Write Policy)

- ระบบจะต้องมีการจัดการกับการเขียนข้อมูลลงในcache
- การเขียนจะต้องถูกกระทำในเวิร์ดที่อยู่ในแคช และทำการเปลี่ยนค่าที่อยู่ในหน่วยความจำหลักด้วยเพื่อให้ข้อมูลอัปเดท
- ถ้ามีปัจจัยอื่น ๆ ที่ทำให้หน่วยความจำหลักมีการเปลี่ยนแปลงข้อมูลแล้ว แคชก็จะต้องทำการอัปเดทค่าของข้อมูลด้วย
- ในระบบที่มีซีพียูมากกว่า 1 ตัวและมีแคชแยกเป็นของตนเอง การเปลี่ยนแปลงข้อมูลของแคชหนึ่งอาจทำให้ข้อมูลในแคชอื่นผิดพลาดไปด้วยถ้าไม่มีการอัปเดทข้อมูล

วิธีการเขียนข้อมูล (Write Policy)

- **การเขียนทั้งหมด** (Write through) เป็นวิธีที่ง่ายและใช้งานมากที่สุด คือให้มีการอัปเดทข้อมูลในหน่วยความจำหลักทุกครั้งที่มีการเขียนข้อมูลเกิดขึ้น พร้อมกับการปัปเดทข้อมูลที่อยู่ในแคช ที่มีแอ็ดเดรสตรงกัน วิธีนี้จะมีข้อดีก็ตรงที่ข้อมูลที่อยู่ในหน่วยความจำหลักนั้นจะมีข้อมูลตรงกับข้อมูลที่อยู่ในแคช
- **การเขียนทีหลัง** (Write-back) ในวิธีนี้การปรับเปลี่ยนข้อมูลจะกระทำในแคชเท่านั้น แต่ตำแหน่งของข้อมูลดังกล่าวก็จะถูกบันทึกโดยใช้บิตพิเศษที่เรียกว่า flag ทำการกำหนดว่าเมื่อเวิร์ดนั้นถูกนำออกมาจากแคชเมื่อใด ก็จะต้องทำการอัปเดทข้อมูลในหน่วยความจำด้วย

จำนวนของแคช

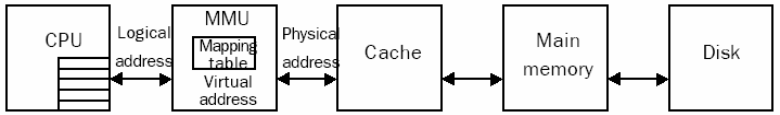
- **แคชที่มีหลายลำดับชั้น**
  - แคชที่อยู่บนชิพ (On-chip cache) หรือที่เรียกว่าแคช L1
  - แคชภายนอก ที่เรียกว่าแคช L2
- **การใช้งานแคชแบบเก็บข้อมูลรวมหรือแยก** ถ้าใช้แบบรวมข้อมูลจะมีข้อดีคือ
  - การเก็บแบบรวมจะมีอัตราการพบข้อมูลมากกว่าการเก็บแบบแยก
  - การออกแบบและการนำไปใช้งานในการเก็บแบบรวมจะง่ายกว่าการเก็บแบบแยก
- แนวโน้มการเก็บข้อมูลจะมุ่งไปทางการเก็บแบบแยก ซึ่งมีข้อดีตรงที่ช่วยลดการแข่งขันของแคชในการนำข้อมูลเข้าและแปลข้อมูล

การเริ่มต้นใช้งานแคช

- เมื่อมีกระแสไฟฟ้าเลี้ยงคอมพิวเตอร์
- เมื่อหน่วยความจำหลักโหลดโปรแกรม
- เริ่มต้นมาจากหน่วยความจำอื่น
- ใช้บิตพิเศษ (valid bit) เพื่อตรวจสอบความถูกต้องของแคช เนื่องจากในแคชมักมีข้อมูลขยะ

หน่วยความจำเสมือน (Virtual Memory)

- ใช้เทคโนโลยีของดิสก์เป็นหน่วยความจำสำรอง หรือหน่วยความจำเสมือน
- หน่วยความจำประเภทดิสก์ถูกจัดอยู่ในระดับล่างของหน่วยความจำ

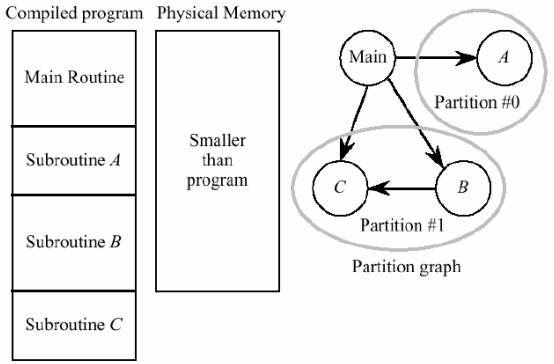


หน่วยความจำเสมือน (Virtual Memory)

- หน่วยความจำเสมือนจำเป็นต้องใช้กระบวนการเชื่อมโยงข้อมูล (Mapping)
- ระบบปฏิบัติการจะทำหน้าที่จัดสรรพื้นที่สำหรับการทำงานบนหน่วยความจำและบนดิสก์ แล้วทำการสลับเปลี่ยนไปมาเมื่อโปรแกรมต้องการ
- ข้อดีก็คือโปรแกรมสามารถมีขนาดใหญ่กว่าหน่วยความจำจริงได้ เนื่องจากมีการจัดสรรไปไว้บนดิสก์

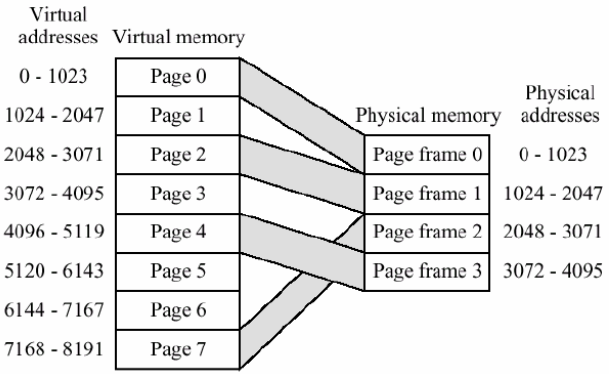
การวางโปรแกรมทับซ้อน (Overlays)

- การใช้วิธีวางโปรแกรมทับซ้อนกัน (Overlay) ทำให้สามารถเขียนโค้ดของตัวเองทับโค้ดอื่น ๆ ได้ตามที่ต้องการ โดยโปรแกรมเมอร์จัดการ



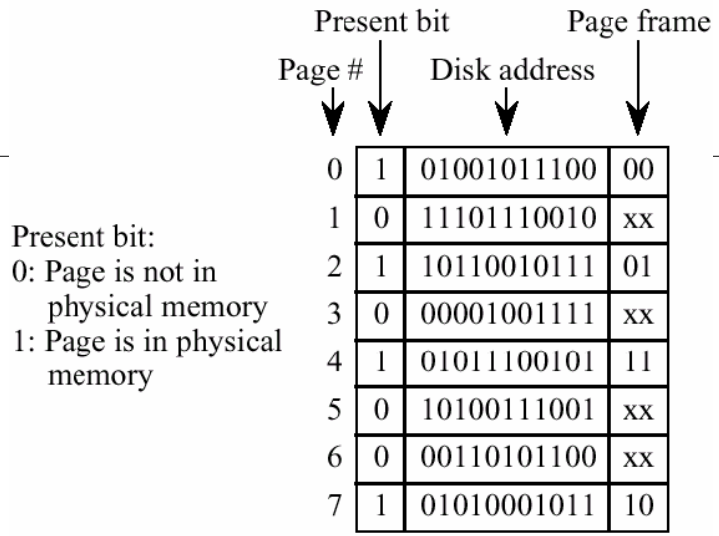
การแบ่งออกเป็นหน้า (Paging)

- การแบ่งพื้นที่และแอ็ดเดรสของหน่วยความจำเสมือนออกเป็น Block ที่เท่ากันที่เราจะเรียกว่า หน้า (Page)



การผิดหน้า (Page Fault)

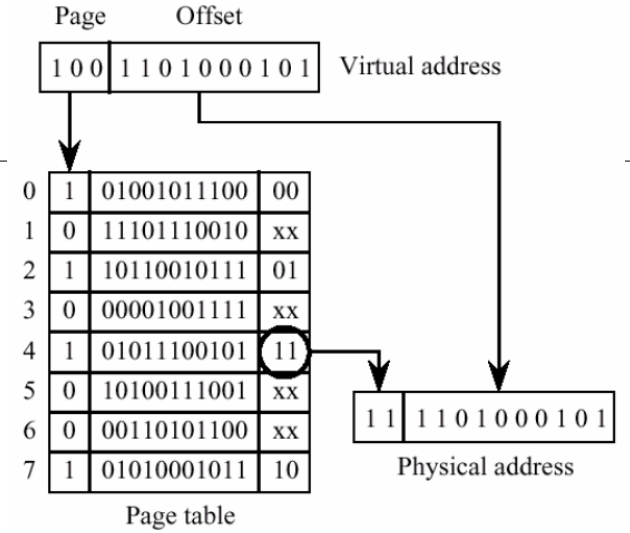
- การผิดหน้า (Page fault) เกิดจากมีหน่วยความจำเสมือนที่ถูกอ้างอิงถึงนั้นไม่ได้อยู่ในหน่วยความจำหลัก ทำให้เกิดกระบวนการ
  - ถ้าเฟรมของหน้าใดหน้าหนึ่งถูกกำหนดว่าจะมีการเขียนโปรแกรมหรือข้อมูลทับ เนื้อหาในเฟรมหน้านั้นก็จะถูกเขียนลงในหน่วยความจำสำรอง เพื่อให้มีการบันทึกค่าดังกล่าวก่อนที่เฟรมของหน้านั้นจะถูกเขียนข้อมูลทับ
  - หน้าของข้อมูลที่เราต้องการเข้าถึงและเก็บไว้ในหน่วยความจำสำรองจะถูกเขียนลงในหน่วยความจำหลัก (ในเฟรมหน้าที่เราได้กำหนดไว้ในข้อ 1)
  - ตารางหน้า จะถูกอัปเดทข้อมูลการเชื่อมโยงระหว่างหน่วยความจำเสมือนเข้ากับหน่วยความจำหลักใหม่
  - ทำงานอื่น ๆ ต่อไป



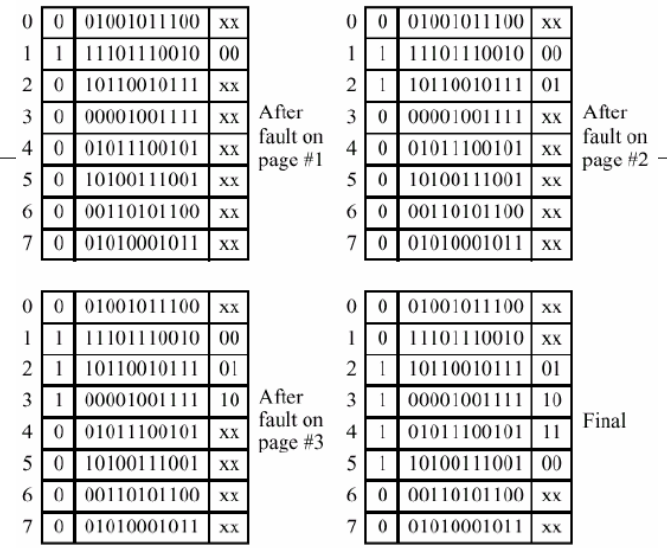
ตารางหน้าของหน่วยความจำเสมือน

ตารางหน้า (Page Table)

- ตารางหน้ามีจำนวนเรกคอร์ดเท่ากับจำนวนหน้าที่มีในหน่วยความจำเสมือน และฟิลด์ต่าง ๆ ดังนี้
  - Present bit
  - Disk address
  - Page frame



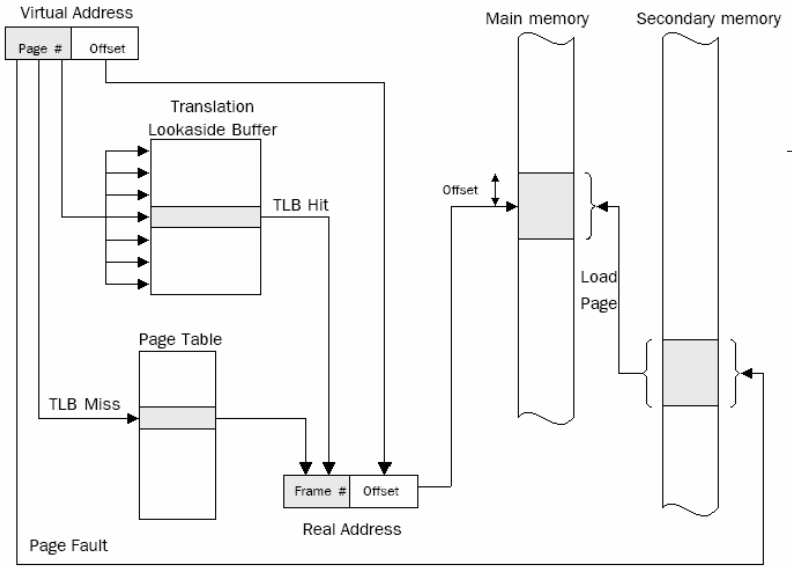
การแปลงแอ็ดเดรสของหน่วยความจำเสมือนเป็นแอ็ดเดรสของหน่วยความจำหลัก

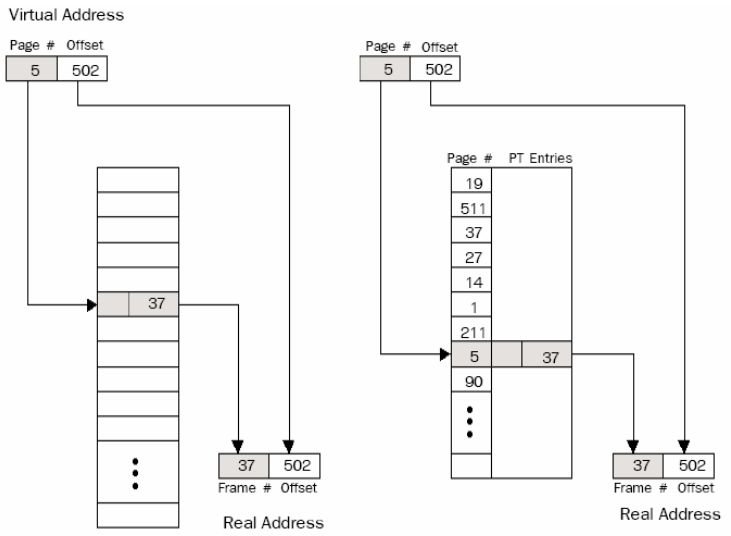


ตัวอย่างการจัดการตารางหน้าเมื่อมีการทำงานของโปรแกรม

บัพเฟอร์ค้นหาที่อยู่ (Translation Lookaside Buffer)

- ปกติเมื่อมีการอ้างถึงหน่วยความจำเสมือน จะมีการเรียกหน่วยความจำหลัก 2 ครั้ง (ทำให้ใช้เวลาเป็น 2 เท่า)
  - ดึงแถวของตารางหน้า
  - ดึงข้อมูลที่ต้องการออกจากหน่วยความจำหลัก
- เป็นบัพเฟอร์ที่ทำหน้าที่เก็บแถวของตารางหน้าที่ใช้งานล่าสุด





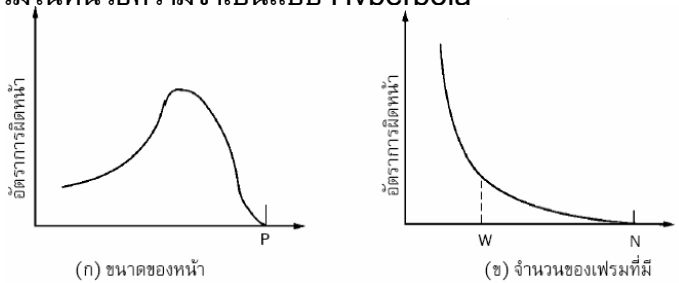
การเปรียบเทียบการเชื่อมโยงระหว่าง Page Table และ TLB

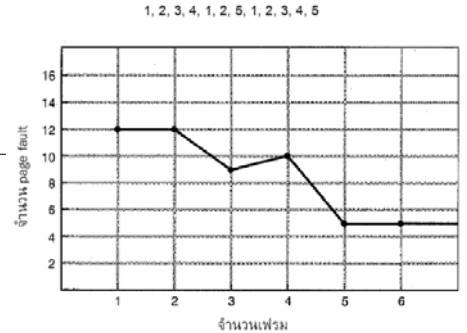
การสับเปลี่ยนหน้า (Page Replacement Alg.

- การพิจารณาการสับเปลี่ยนหน้าขึ้นอยู่กับหน่วยความจำและจำนวนโปรเซสที่อนุญาตให้เข้าไป
- ถ้าให้หน่วยความจำแก่โปรเซสใด ๆ น้อยเท่าไหร่ ก็จะทำให้มีจำนวนโปรเซสเข้ามาใช้งานหน่วยความจำมากขึ้นเท่านั้น ทำให้ความน่าจะเป็นของระบบที่จะสามารถค้นพบโปรเซสอย่างน้อยหนึ่งโปรเซสที่พร้อมที่จะทำงานมากขึ้น ทำให้เราเสียเวลาในการสลับหน้าน้อยลง

การสับเปลี่ยนหน้า (Page Replacement Alg.)

- ถ้ามีจำนวนหน้าของโปรเซสหนึ่ง ๆ ที่สามารถอยู่ในหน่วยความจำหลักน้อย (จำนวนเฟรมน้อย) ก็จะทำให้อัตราการเกิดการผิดหน้าสูงมากขึ้น เพราะว่าความสัมพันธ์ระหว่างอัตราการผิดหน้าและจำนวนเฟรมในหน่วยความจำเป็นแบบ Hyperbola

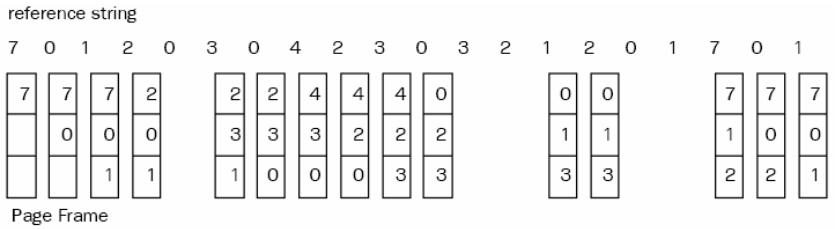




เมื่อทดลองหา จำนวนครั้งของการผิดหน้าเทียบกับจำนวนเฟรมที่มีจะได้กราฟในรูป จะสังเกตได้ว่า เมื่อหน่วยความจำมีเฟรมทั้งหมด 4 เฟรม จะเกิดการผิดหน้าทั้งหมด 10 ครั้ง ซึ่งมากกว่า เมื่อมีทั้งหมด 3 เฟรม ผลลัพธ์ที่ไม่ปกตินี้ เราเรียกว่า ***ปรากฏการณ์เบลาดี้*** *(Balady's anomaly)* ซึ่งแสดงให้เห็นว่า วิธีการสับเปลี่ยนหน้าบางแบบ ก็อาจจะทำให้ อัตราการผิดหน้าเพิ่มขึ้นได้ ถึงแม้เราจะเพิ่มจำนวนเฟรมขึ้นก็ตาม

การสับเปลี่ยนแบบมาก่อน-ออกก่อน

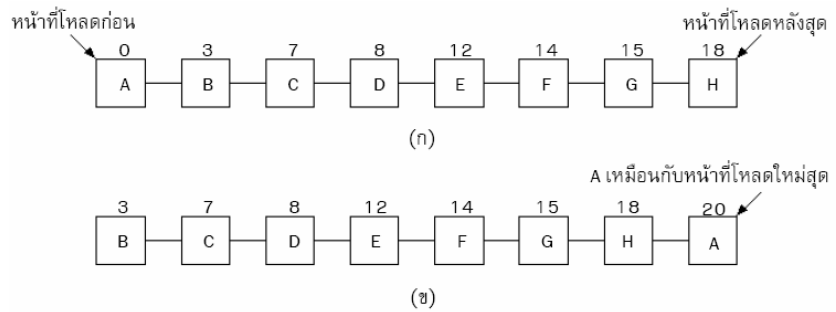
- หน้าที่เข้ามานานที่สุดออกก่อน



การสับเปลี่ยนแบบให้โอกาสครั้งที่สอง

- ปรับปรุงการสับเปลี่ยนแบบมาก่อนออกก่อนเพื่อป้องกันการเปลี่ยนหน้า
- ตรวจสอบบิต R (Referenced) ของหน้าที่เข้ามานานที่สุด ถ้าบิต R มีค่าเป็น 0 แสดงว่าเป็นหน้าเก่า ไม่ได้ใช้งาน ระบบจะสับเปลี่ยนทันที
- แต่ถ้าบิต R มีค่าเป็น 1 ก็กำหนดบิต R เป็น 0 แล้วกลับไปเข้าแถวใหม่ พร้อมเปลี่ยนแปลงเวลาของหน้านั้นเหมือนหน้าที่เพิ่งเข้ามาใหม่

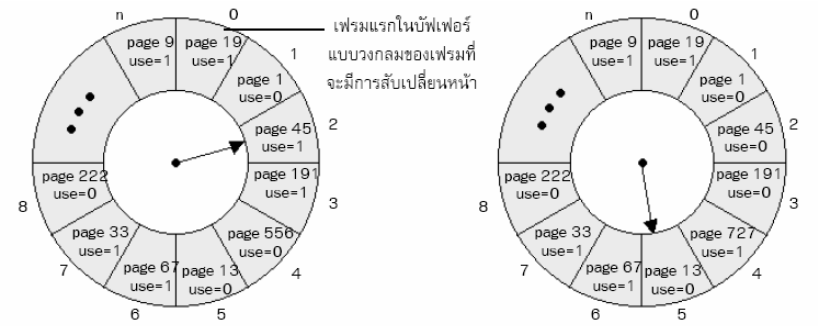
การสับเปลี่ยนแบบให้โอกาสครั้งที่สอง



การสับเปลี่ยนแบบวงรอบนาฬิกา

- ปรับปรุงการสับเปลี่ยนแบบให้โอกาสครั้งที่สองเพื่อลดเวลาที่ใช้ในการย้ายหน้า
- เรียง frame ทุก frame เป็นรูปวงกลม และมีเข็มนาฬิกาชี้ไปที่หน้าที่เก่าที่สุด
- เมื่อมีการผิดหน้า หน้าที่มีเข็มนาฬิกาชี้อยู่จะถูกตรวจสอบ ถ้าบิต R มีค่าเป็น 0 หน้านั้นก็จะถูกสับเปลี่ยนออกไป และหน้าใหม่ก็จะถูกใส่เข้ามาในตำแหน่งเดิม
- เข็มนาฬิกาก็จะทำการเลื่อนไปด้านหน้า 1 ตำแหน่ง
- แต่ถ้าบิต R ถูกกำหนดเป็น 1 ก็ให้ลบค่าของบิตนั้นเป็น 0 และเลื่อนเข็มไปหน้าถัดไป

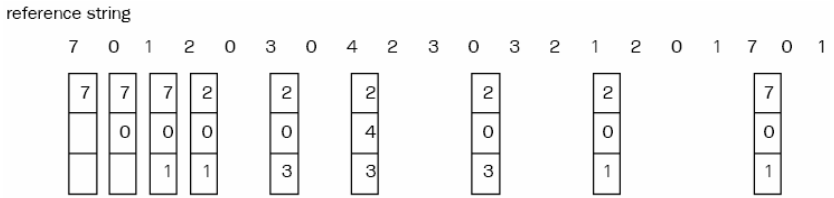
การสับเปลี่ยนแบบวงรอบนาฬิกา



(ก) สถานะของบัฟเฟอร์ก่อนที่จะมีการสับเปลี่ยนหน้า (ข) สถานะของบัฟเฟอร์หลังจากสับเปลี่ยนหน้าต่อไป

การสับเปลี่ยนแบบที่ดีที่สุด

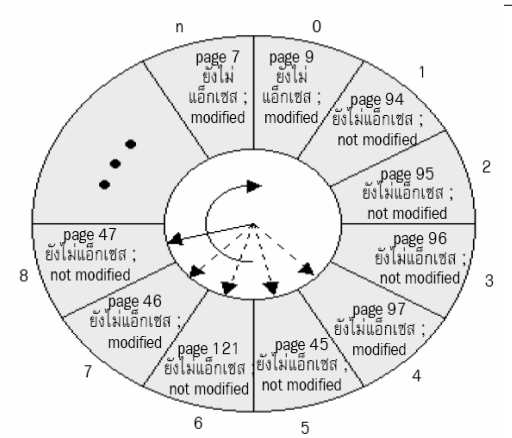
"ให้เลือกสับเปลี่ยนหน้าที่จะไม่ถูกเรียกใช้งาน และมีระยะเวลารอการเรียกใช้ที่นานที่สุด"



การสับเปลี่ยนแบบที่ไม่ได้ใช้งาน-ออกก่อน

- ปรับปรุงการสับเปลี่ยนแบบวงรอบนาฬิกา โดยพิจารณาบิตสำคัญในตารางหน้า
- พิจารณาจากสถิติว่าหน้าใดกำลังถูกใช้ หรือไม่ได้ใช้งาน
- บิต R ถูกกำหนดเป็น 1 เมื่อหน้านั้นถูกเรียกใช้งาน
- บิต M ถูกกำหนดเป็น 1 เมื่อหน้านั้นมีการเปลี่ยนแปลง
- บิตทั้งสองอยู่ในแถวของตารางหน้า จะมีการเปลี่ยนแปลงค่าทุกครั้งเมื่อมีการใช้งาน หรือถูกอ้างอิงถึง

การสับเปลี่ยนแบบที่ไม่ได้ใช้งาน-ออกก่อน

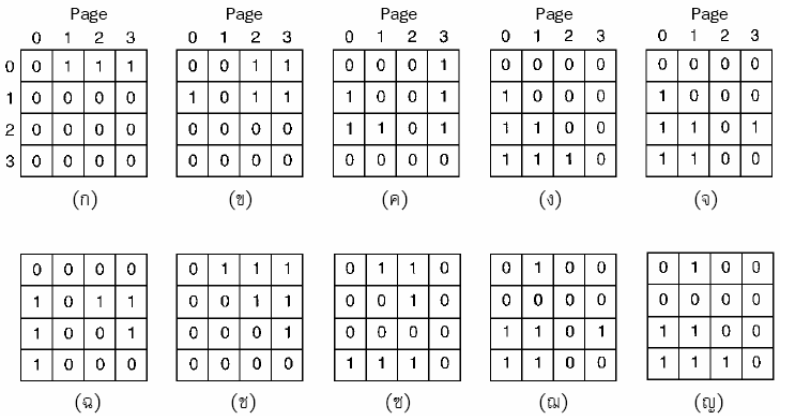


การสับเปลี่ยนแบบใช้งานน้อยที่สุด-ออกก่อน

- มีการบันทึกเวลา
- จะสับเปลี่ยนเอาหน้าที่ไม่ได้ใช้งานเป็นเวลานานออก (ตัวเลขเวลาน้อยที่สุด)
- ระบบจะต้องมีลิงค์ลิสต์ทุก ๆ หน้าในหน่วยความจำ ซึ่งเป็นเรื่องยากเนื่องจากจะต้องมีการลิสต์จะต้องปรับเปลี่ยนทุกครั้งที่มีการอ้างถึง

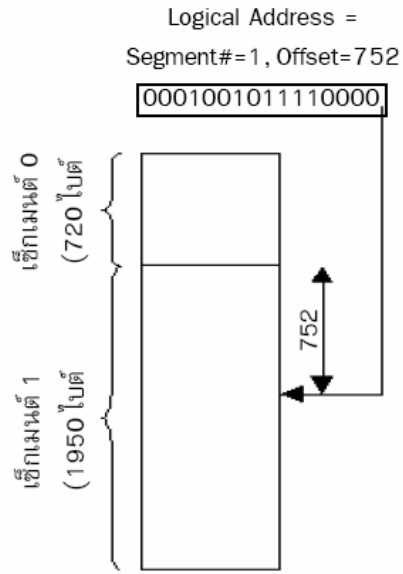
การสับเปลี่ยนแบบใช้งานน้อยที่สุด-ออกก่อน

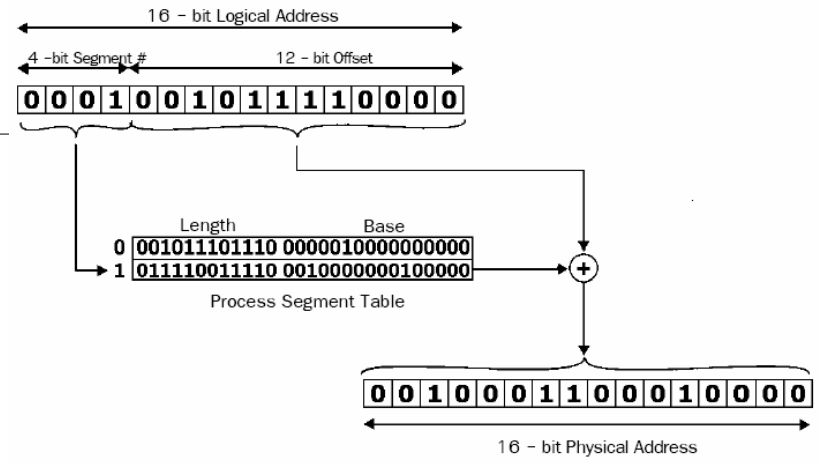
0 1 2 3 2 1 0 3 2 3



การแบ่งเป็น Segment

- แบ่ง Process หรือโปรแกรม ออกเป็นส่วน ๆ โดยแต่ละส่วนนั้นไม่จำเป็นต้องมีความยาวเท่ากัน (แต่อาจจะมีการกำหนดให้อยู่ภายใต้ขนาดที่ใหญ่ที่สุดที่ระบบกำหนด)
- มีการแปลงแอ็ดเดรสทางตรรกเป็นแอ็ดเดรสจริงในหน่วยความจำเหมือนการแบ่งเป็นหน้าที่ต้องใช้ข้อมูล 2 ส่วน คือ เลขที่ของเซ็กเมนต์ และระยะออปเซ็ทจากเซ็กเมนต์นั้น (offset)



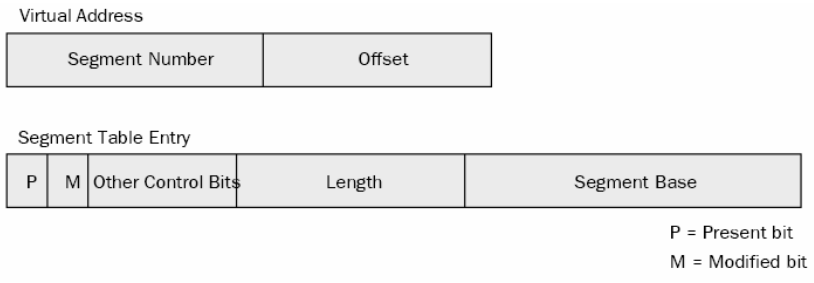


การแปลงแอ็ดเดรสทางตรรกเป็นแอ็ดเดรสจริงของการแบ่งเป็นเซ็กเมนต์

การนำการแบ่งเป็น Segment มาใช้ในหน่วยความจำเสมือน

- อนุญาตให้โปรแกรมเมอร์สามารถมองเห็นว่าหน่วยความจำนั้นประกอบขึ้นจากชิ้นส่วนเล็กๆ และชิ้นส่วนของหน่วยความจำเหล่านี้อาจจะมีขนาดไม่เท่ากัน และไม่คงที่ มีข้อดีหลาย ๆ ข้อต่อโปรแกรมเมอร์
  - การจัดการกับการเพิ่มโครงสร้างของข้อมูลในโปรแกรมสามารถทำได้ง่าย
  - วิธีการนี้สามารถทำให้เราเปลี่ยนแปลง หรือคอมไพล์โปรแกรมแยกออกเป็นส่วน ๆ ได้
  - การแบ่งเป็นเซ็กเมนต์นี้จะอนุญาตให้มีการแบ่งปันข้อมูลระหว่างโปรเซสหลาย ๆ โปรเซสได้
  - การแบ่งแบบนี้อนุญาตให้มีการป้องกันข้อมูลด้วยตัวเองได้

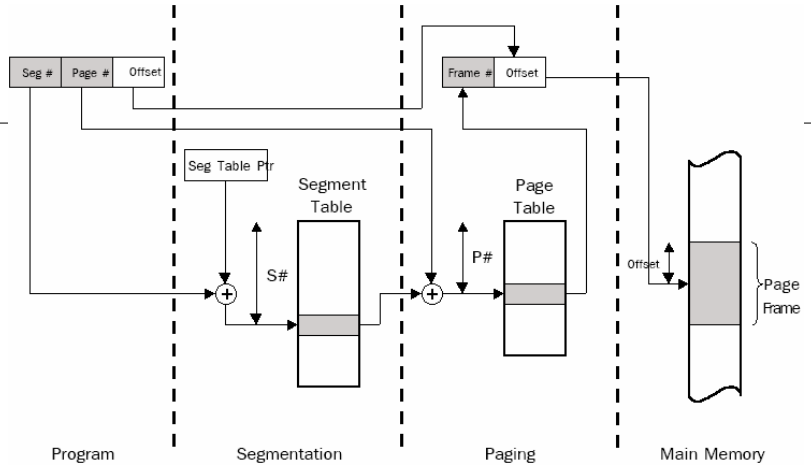
การนำการแบ่งเป็น Segment มาใช้ในหน่วยความจำเสมือน



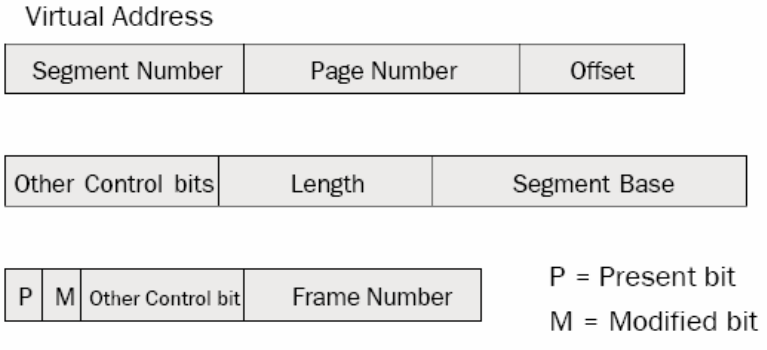
Logical address และตารางการแบ่งเป็น segment

การรวมวิธีการแบ่งเป็นหน้ากับการแบ่ง segment เข้าด้วยกัน

- ข้อดีของการแบ่งเป็นหน้า
  - การปกปิดไม่ให้โปรแกรมเมอร์มองเห็นการทำงานของมัน
  - การป้องกันการสูญเสียพื้นที่ภายนอก (external fragmentation)
  - ทำให้ระบบใช้หน่วยความจำหลักอย่างมีประสิทธิภาพ
  - เนื่องจากหน้าของโปรเซสที่ถูกสลับเข้าและออกจากหน่วยความจำนั้น มีขนาดคงที่ และเท่ากัน จึงมีความเป็นไปได้ที่เราจะพัฒนาอัลกอริทึมที่ ซับซ้อน เพื่อนำไปใช้ในการจัดการกับหน่วยความจำได้
- ข้อดีของการแบ่งเป็นเซ็กเมนต์ ดังที่กล่าวก่อนหน้านี้ และ
  - สามารถจัดการโครงสร้างข้อมูลที่มีขนาดไม่คงที่ได้ดี
  - ทำโปรแกรมเป็นโมดูล
  - สนับสนุนการแบ่งปันและการป้องกันเซ็กเมนต์

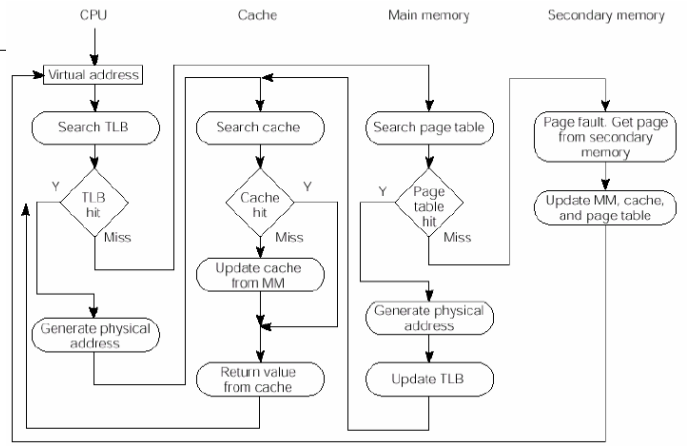


การแปลง address ของระบบ Segmetation/Paging



ตาราง ที่เกิดจากการรวม Segmentation/Paging

แคชและหน่วยความจำเสมือน



การทำงานของระบบหน่วยความจำคอมพิวเตอร์