Cache สิ่งจำเป็นสำหรับคอมพิวเตอร์ของคุณ

ด้วยซอฟต์แวร์ระดับบิ๊กเหล่านี้ทำให้ปริมาณข้อมูลที่ผ่านเข้าออกในเครื่องคอมพิวเตอร์พลอยสูงไปขึ้นไปด้วย ไม่น่าแปลกใจแต่อย่างใด ที่เครื่องของคุณจะทำงานได้ช้าลงไปถนัดใจเมื่อต้องเจอซอฟต์แวร์ขนาด 60 เมกะไบต์ ที่ทำงานกับหน่วยความจำขนาด 8-16 เมกะไบต์ ด้วยข้อมูลเพียงน้อยนิดที่สามารสเก็บได้ เมื่อเทียบกับข้อมูลมหาศาลที่ต้องใช้ทำให้การถ่ายเทข้อมูลระหว่าง CPU หน่วยความจำ และฮาร์ดดิสก์เกิดขึ้นครั้งแล้วครั้งเล่า อันเป็นเหตุสำคัญที่ทำให้เครื่องทำงานช้าลง อย่างช่วยไม่ได้

นอกจากนั้นสำหรับผู้ที่ใช้ซอฟต์แวร์กราฟฟิกที่ความระเอียดสูงๆ และใช้สีแบบ True Color ก็คงซาบซึ้งดีกับความล่าช้าในการทำงาน แน่นอนหน่วยความจำสัก 16 เมกะไบต์คงจะพอช่วยได้ หรือจะให้ดีก็ 32 เมกะไบต์ เครื่องคงจะดีขึ้นทันใจ แต่การเพิ่มหน่วยความจำเข้าไปขนาดนั้นก็ถือว่าเป็นภาระอันใหญ่หลวงของเงินในกระเป๋าเช่นกันสำหรับคนยากจนจะมีหนทางใดที่จะช่วยลดปัญหาที่เกดขึ้นโดยไม่ต้องเสียเงินทองมากมาย มีวิธีหนึ่งที่จะช่วยคุณได้คือการใช้หน่วยความจำแคช (Cache Memory) และดิสก์แคช (Disk Caching)

แคช คืออะไร

แคชถือกำเนิดมาจากความไม่ลงรอยระหว่างอุปกรณ์ที่ต้องการใช้ข้อมูลกับอุปกรณ์เก็บข้อมูล ในยุคสมัยเริ่มแรกของคอมพิวเตอร์นั้น การทำงานของส่วนต่างๆในคอมพิวเตอร์ยังไม่เร็วมากนัก (อาจเรียกว่าช้าหากจะเปรียบกับปัจจุบัน) ไม่ว่าจะเป็นCPU หน่วยความจำ ดิสก์ไดรฟ์หรือเครื่องพิมพ์ก็ตาม แม้ว่าจริงๆแล้วCPUกับหน่วยความจำจะทำงานเร็วกว่าดีสก์ไดรฟ์และเครื่องพิมพ์เป็นร้อยเท่าก็ตาม IBM FC ที่มีให้ใช้เฉพาะฟลอปปี้ดิสก์นั้น ผู้ใช้เครื่องก็ไม่รู้สึกอะไรมากนักกับความช้าของการอ่านเขียนดิสก์ หรือการพิมพ์ อาจเนื่องมาจากความเคยชินก็ได้

แต่ขณะที่คอมพิวเตอร์ได้พัฒนาก้าวหน้าไปเรื่อยๆนั้น พัฒนาการขององค์ประกอบต่างๆ ในเครื่องคอมพิวเตอร์ก็ได้แตกต่างและห่างออกไปเรื่อยๆCPUเป็นองค์ประกอบของเครื่องคอมพิวเตอร์ที่พัฒนาไปได้รวดเร็วมากที่สุด หากจะเทียบกับเครื่อง IBM PC/XT แล้วCPU Pentium มีความเร็วมากกว่าถึง 500 เท่า ในขณะที่หน่วยความจำทำงานเร็วขึ้นเพียง 3-4 เท่า ฮาร์ดดิส์เร็วขึ้น 10 เท่า และดิสก์ไดรฟ์แทบจะไม่ได้ทำงานเร็วขึ้นเลย ความแตกต่างอย่างมหาศาลของการพัฒนานี้ได้ทำให้ช่องว่างระหว่างความเร็วขององค์ประกอบต่างๆ กว้างออกไปเรื่อยๆ

คงไม่มีประโยชน์อะไร ที่จะพัฒนาCPUที่มีความเร็วมหาศาลแต่ต้องกลับมารออุปกรณ์รอบข้างที่ทำงานช้ากว่าอย่างมากมาย ดังนั้งจึงได้มีการค้นคว้าหาวิธีที่จะมาอุดช่องว่างที่เกิดขึ้นนี้ให้ได้ และนั้นก็คือที่มาของแคช (Cache)

แคชเป็นชื่อเรียกแหล่งเก็บข้อมูลความเร็วสูงที่ทำหน้าที่เป็นตัวกลางระหว่างอุปกรณ์ในเครื่องคอมพิวเตอร์ที่มีความเร็วต่างกันมาก ถ้าเป็นตัวกลางระหว่างหน่วยความจำกับCPU ก็เรียกว่าหน่วยความจำแคช และหากเป็นตัวกลางระหว่างหน่วยความจำหรือCPUกับฮาร์ดดิสก์ ก็เรียกว่าดิสก์แคช

หน่วยความจำแคช

หน่วยความจำแคชเป็นชื่อเรียกหน่วยความจำส่วนหนึ่งในเครื่องคอมพิวเตอร์ที่ทำหน้าที่เป็นตัวกลางระหว่างCPUกับหน่วยความจำปกติ โดยหน่วยความจำส่วนนี้จะมีความเร็วสูงกว่าหน่วยความจำปกติค่อนข้างมาก (ต่อไปจะเรียกว่าหน่วยความจำ และจะเรียกหน่วยความจำแคชว่าแคช) โดยทั่วไปแคชจะมีความเร็วสูงกว่าหน่วยความจำประมาณ 4-6 เท่า แคชมักจะใช้หน่วยความจำชนิดสแตติก (Static Memory) ซึ่งมีความเร็วสูงแต่มีราคาแพงและมีความจุต่อพื้นที่ต่ำในขณะที่หน่วยความจำปกติจะใช้หน่วยความจำชนิดไดนามิก (Dynamic Memory) ที่มีราคาถูกและมีความจุต่อพื้นที่สูงกว่าแต่มีความเร็วที่ต่ำกว่า (เป็นหน่วยความจำเหมือนกัน แต่เป็นคนละชนิด)

ในระบบคอมพิวเตอร์ที่มีแคชนั้นการติดต่อกับหน่วยความจำทุกๆ ครั้งจะต้องผ่านทางแคชเสมอ ไม่ว่าจะเป็นการอ่านหรือการเขียนก็ตาม โดยในการอ่านข้อมูลข้อมูลนั้นCPUจะอ่านข้อมูลจากหน่วยความจำปกติแต่การอ่านข้อมูลนี้จะผ่านมาที่แคชก่อน จากนั้นแคชจะตรวจสอบว่าข้อมูลที่ต้องการนั้นอยู่ที่แคชหรือไม่ ซึ่งหากเป็นช่วงเริ่มการทำงานตั้งแต่เปิดเครื่องใหม่ก็จะไม่มีข้อมูลอะไรอยู่ในแคชเลย ดังนั้นแคชก็จะผ่านการอ่านข้อมูลนี้ไปยังหน่วยความจำเพื่ออ่านเอาข้อมูลที่CPUต้องการออกมา ซึ่งข้อมูลนั้นก็จะถูกส่งผ่านแคชเข้าสู่CPUเพื่อใช้งานต่อไป

ตอนนี้จะข้ออธิบายการอ่านข้อมูลของแคชเพิ่มเติม ซึ่งตรงนี้มีความสำคัญมากคือว่า ในขณะที่แคชกำลังอ่านข้อมูลจากหน่วยความจำเพื่อส่งต่อไปยังCPUนั้น มันจะถือโอกาสอ่านข้อมูลข้างเคียงกับข้อมูลที่ต้องการมาด้วย ซึ่งตรงนี้จะต่างจากการอ่านข้อมูลของCPUในระบบที่ไม่มีแคช เพราะโดยทั่วไปเมื่อCPUต้องการอ่านข้อมูลจากหน่วยความจำ มันจะอ่านเฉพาะข้อมูลที่ต้องการเท่านั้น เช่น หากต้องการอ่านข้อมูล 1 ไบต์ก็จะอ่านข้อมูลมาเพียง 1 ไบต์เท่านั้น

ข้อมูลข้างเคียงที่อ่านขึ้นมาพร้อมๆกันนั้น จะมากหรือน้อยก็ขึ้นอยู่กับการออกแบบระบบของคอมพิวเตอร์นั้นๆ เช่น อาจจะอ่านมาครั้งละ 1 กิโลไบต์ เป็นต้น ซึ่งการติดต่อระหว่างหน่วยความจำกับแคชนี้จะควบคุมด้วยแคชคอนโทลเลอร์ แคชคอนโทลเลอร์จะแบ่งหน่วยความจำออกเป็นส่วนๆเรียกว่าบล็อก แต่ละบล็อกจะมีขนาดเท่าๆกัน เช่น 1 กิโลไบต์ เป็นต้น และในขณะเดียวกันก็จะแบ่งแคชออกเป็นบล็อกเช่นเดียวกัน โดยขนาดของบล็อกในแคชและหน่วยความจำจะต้องเท่ากัน

ดังนั้นในขณะที่แคชอ่านข้อมูลในหน่วยความจำเพื่อส่งให้กับCPUนั้นแคชจะอ่านข้อมูลทั้งบล็อกเลย แล้วจึงส่งข้อมูลที่CPUต้องการไปให้CPU โดยข้อมูลในบล็อกนั้นก็ยังคงอยู่ในแคชต่อไป ถึงตอนนั้นอาจยังมีผู้สงสัยว่าในเมื่อขั้นตอนการอ่านข้อมูลค่อนข้างซับซ้อนเช่นนี้ จะไม่ทำให้การอ่านข้อมูลช้าลงหรือ? ซึ่งคำตอบก็คือ ไม่ เพราะขั้นตอนทั้งหมดถูกควบคุมด้วยแคชคอนโทลเลอร์ ซึ่งเป็นฮาร์ดแวร์ที่อยู่บนเมนบอร์ด ดังนั้นขั้นตอนการอ่านทั้งหมดจึงสามารถทำไปพร้อมๆ กับอ่านข้อมูลตามปกติได้

จากที่กล่าวมาทั้งหมดนี้เป็นเพียงวิธีการนำข้อมูลที่อยู่ในหน่วยความจำเข้ามาอยู่ในแคชเท่านั้น ซึ่งจะเห็นได้ว่า CPU ไม่ได้ใช้ประโยชน์จากแคชเลย CPU จะใช้ประโยชน์จากแคชได้ก็ต่อเมื่อข้อมูลที่ CPU ต้องการอยู่บนแคชเท่านั้น เพราะเมื่อข้อมูลที่ต้องการอยู่บนแคช CPUก็สามารถอ่านข้อมูลจากแคชเข้ามาได้เลย โดยไม่จำเป็นต้องเข้าไปข้องเกี่ยวกับหน่วยความจำอีกและเนื่องจากการทำงานของแคชนั้นเร็วกว่าหน่วยความจำอยู่หลายช่วงตัว ดังนั้น CPUจึงเสียเวลาในการอ่านข้อมูลน้อยลง ถึงตอนนี้คงพอเข้าใจว่าแคชช่วยให้คอมพิวเตอร์เร็วได้อย่างไร

เมื่อประกอบกับลักษณะการทำงานของคอมพิวเตอร์ที่มักจะมีการอ่านข้อมูลที่อยู่ใกล้เคียงกันเป็นส่วนใหญ่ด้วยแล้ว ก็ได้ทำให้ CPUสามารถหาข้อมูลในแคชพบในอัตราที่มากอย่างไม่น่าเชื่อ หากกำหนดขนาดของแคชที่พอเหมาะ เช่น CPU386 ที่มีหน่วยความจำ 4-8 เมกะไบต์ ก็ควรจะใช้แคช 128 กิโลไบต์ และ CPU486 ที่มีหน่วยความจำ8-16 เมกะไบต์ ก็ควรใช้แคช 256 กิโลไบต์ ซึ่งหากใช้แคชตามที่กำหนดไว้ข้างต้นนี้แล้วก็จะทำให้ CPUพบข้อมูลในแคชได้ถึงกว่า 90 ครั้งต่อการอ่านข้อมูล 100 ครั้งในการใช้งานไปซึ่งอัตราการพบข้อมูลในแคชต่อการอ่านข้อมูลทั้งหมดนี้ จะเรียกว่า Hit Rate และ มีหน่วยเป็นเปอร์เซ็นต์

ด้วยอัตรา Hit Rate ที่สูงมากนี้เองที่ชี้ให้เห็นว่าในสภาพการทำงานทั่วไปนั้น CPUจะใช้เวลาในการติดต่อกับแคชเป็นส่วนใหญ่ และด้วยความเร็วที่สูงมากของแคชนี่เอง (เทียบกับหน่วยความจำ) จึงทำให้การทำงานของเครื่องเร็วขึ้นตามไปด้วยดั้งนั้นจะเห็นได้ว่า แคชเป็นองค์ประกอบที่สำคัญมากในเครื่องคอมพิวเตอร์ อันจะขาดไปเสียไม่ได้ ลองนึกภาพ CPUที่ต้องทำกับหน่วยความจำที่ช้ากว่าแคชถึง 4 เท่า ว่าจะทำให้ช้าลงได้มากแค่ไหน

สำหรับในแง่ของการเขียนข้อมูลนั้น ก็จะใช้หลักการคล้ายๆกับการอ่านโดยที่ CPUจะเขียนข้อมูลลงบนแคชแทนที่จะเขียนข้อมูลลงในหน่วยความจำ แคชที่ทำงานเร็วกว่าจะช่วยให้ CPUสามารถกลับไปทำงานอื่นได้เร็วขึ้น แทนที่จะต้องมารอการทำงานของหน่วยความจำ แต่สำหรับการเขียนแคชจะเขียนเฉพาะข้อมูลลงในหน่วยความจำเท่านั้น จะไม่เข้าไปยุ่งเกี่ยวกับข้อมูลข้างเคียง คือไม่ได้เขียนลงไปทั้งบล็อก ซึ่งถือเป็นจุดแตกต่างระหว่างการอ่านและการเขียน

มีศัพท์เกี่ยวกับแคชอยู่ 2 คำ ที่จะอธิบายไว้ตรงนี้ คือคำว่า Write Through และ Write Back ทั้ง 2 ค่าเป็นสิ่งที่ใช้ระบุโหมดการเขียนของแคช โดยการเขียนแบบ Write Through นั้นจะเป็นการเขียนในแคชไปพร้อมๆกับเขียนลงในหน่วยความจำ ซึ่งวิธีนี้จะมีวิธีที่ข้อมูลไม่หายไปอย่างแน่นอน แต่การเขียนแบบ Write Back นั้นCPUจะเขียนข้อมูลลงในแคชและกลับไปทำงานต่อ จากนั้นส่วนควบคุมแคชจะนำข้อมูลนั้นไปเขียนลงในหน่วยความจำภายหลัง โดยไม่ต้องไปยุ่งเกี่ยวกับCPUอีกเลยในหน่วยความจำแคชที่มีประสิทธิภาพมักจะเป็นระบบ Write Back เพราะให้ความเร็วในการทำงานที่ดีกว่า แต่สำหรับดิสก์แคชที่ดีแล้วควรจะใช้ระบบ Write Through มากกว่าเพราะมีความปลอดภัยสูงกว่า

เมื่อ CPUอ่านข้อมูล 1ครั้ง จะทำให้แคชอ่านข้อมูล 1 บล็อกมาจากหน่วยความจำ และหาก CPUอ่านข้อมูลที่ไม่มรในแคช ก็จะทำให้แคชอ่านข้อมูลต้องอ่านอีก 1 บล็อกมาจากหน่วยความจำ สมมุติว่า

มีแคช 256 กิโลไบต์ และกำหนดขนาดของบล็อกไว้ที่ 1 กิโลไบต์ ก็จะทำให้แคชสามารถเก็บข้อมูลในบล็อกต่างๆได้ถึง 256 บล็อก ซึ่งในสภาพการทำงานจริงแคชจำนวน 256 บล็อก หรือ 256กิโลไบต์นี้ เดี๋ยวก็เต็มแล้ว

ทีนี้ปัญหาคือว่า หาแแคชเต็มแล้วจะทำอย่างไรหากจะต้องอ่านข้อมูลเข้ามาเพิ่มอีกบล็อกหนึ่ง คำตอบก็คือ ต้องนำข้อมูลในแคชทิ้งไปหนึ่งบล็อก เพื่อนำที่ว่างในบล็อกนั้นมาใส่ข้อมูลที่พึงอ่านเข้ามา แต่ปัญหาที่เกิดขึ้นอีกก็คือ จะนำข้อมูลใดใน 256 บล็อกทิ้งดี ซึ่งตรงนี้ผู้ออกแบบแคชจะต้องออกแบบกติกาไว้ เช่น ให้นำบล็อกที่เก่าที่สุดทิ้งไป (หมายถึงข้อมูลที่อ่านเข้ามาแรกสุด) หรือนำบล็อกที่ใช้งานน้อยที่สุดทิ้งไปแต่โดยทั่วไปแล้วจะนำบล็อกที่เก่าที่สุดทิ้งไป

ถึงตรงนี้อาจมีผู้ถามว่า ทำไมไม่เพิ่มแคชเข้าไปมากจะได้เติมง่าย อันที่จริงการจะเพิ่มแคชเข้าไปนั้นไม่ใช้เรื่องยาก แม้ว่าการเพิ่มแคชจะทำให้ค่าใช้จ่ายเพิ่มมากขึ้นก็ตาม เพราะแคชราคาแพง แต่นั้นก็ไม่ใช่ปัญหาหากมันจะทำให้เครื่องเร็วขึ้นไปอีก แต่ที่เขาไม่เพิ่มกันเพราะว่ามันไม่ประโยชน์ เพราะการเพิ่มแคชจาก 256 กิโลไบต์ ไปเป็น 512 กิโลไบต์นั้น จะช่วยให้ Hit Rate เพิ่มขึ้นเพียง 2-3 เปอร์เซ็นต์เท่านั้น ซึ่งนับว่าไม่คุมค่าเลย

แคชช่วยให้คอมพิวเตอร์ทำงานได้เร็วขึ้น แต่สุดท้ายก่อนที่จะผ่านเรื่องหน่วยความจำแคชไปจะกล่าวถึงแคชอีกประเภทหนึ่ง ก็คือแคชที่เรียกว่าแคชภายใน (Internal Cache)

แคชภายในก็คือหน่วยความจำแคชธรรมดาแต่ว่าจะถูกบรรจุเข้าไปในตัว CPUเลย ก็เพื่อให้ CPUมีความเร็วในการทำงานสูงสุดเท่าที่จะทำได้นั้นเอง เพราะการติดต่อกับแคชภายนอกนั้น CPUจะต้องเสียเวลาบางส่วนให้กับการทำงานของระบบบัส นอกจากนั้นการทำงานของแคชภายในยังสามารสทำให้มีความเร็วสูงกว่าแคชภายนอกได้อีกด้วย

ในปัจจุบัน CPUที่มีแคชมากที่สุดก็ยังมีแค่ 32 กิโลไบต์เท่านั้น แต่แม้ว่าจะมีแคชขนาดเล็กแค่นี้ก็ตาม แต่ก็ช่วยให้การทำงานของ CPUเร็งขึ้นเป็นอย่างมาก เพราะการทำงานของ CPUที่มีการใช้งานข้อมูลที่ใกล้เคียงกันมากนั้นเอง

แต่ด้วยขนาดที่จำกัดของแคชภายในก็ได้ทำให้ Hit Rate ของแคชภายในไม่มากนัก (น้อยกว่า 90 แต่จะเป็นเท่าไรนั้นยังไม่มีการรายงานผล แต่หากจะให้เดาก็น่าจะมี Hit Rate อยู่ประมาณ 60-70) ดังนั้นแคชภายนอกจึงยังมีความจำเป็นอยู่ จะใช้แต่เพียงแคชภายในไม่ได้โดยจะเรียกแคชภายนอกว่า Cache Level2 (แปลว่า แคชระดับ 2 แต่กระนั้นก็มีเครื่องคอมพิวเตอร์ 486 บางยี่ห้อ)ที่ขายเครื่องโดยไม่ได้ใส่แคชภายในมาให้ด้วย แต่ก็โฆษณาว่ามีแคชซึ่งก็หมายถึงแคชภายในนั้นเอง เครื่องคอมพิวเตอร์แบบนี้จะไม่สามารสทำงานได้อย่างเต็มที่เลย นอกจากนั้นใน CPUบางตัว ยังได้นำเอาแคชคอนโทลเลอร์เข้าไปอยู่ใน CPUด้วยเลย ทั้งนี้ก็เพื่อให้แคชภายนอก และแคชภายในทำงานประสานกันได้อย่างดีที่สุด อันจะทำให้ประสิทธิภาพดีที่สุดนั้นเอง

การออกแบบชิพ 68020 นี้มีจุดมุ่งหมายให้ใช้หน่วยความจำแคชทั้งที่อยู่ภายในชิพ และที่เป็นชิพภายนอก เพื่อประสิทธิภาพการทำงานของชิพ

กรณีหน่วยความจำแคชภายในชีพ คำสั่งการทำงานของ 68020 สามารสทำงานโดยการอ่านคำสั่งจากหน่วยความจำแคชภายใน เพื่อลดเวลาการอ่านคำสั่งลงไปได้มาก เพราะไม่ต้องใช้สัญญาณส่งออกไปภายนอก

คำสั่งที่เก็บไว้ในหน่วยความจำแคชภายในจะได้รับการอ่านได้รวดเร็วมาก เพราะการเฟตนี้จะอ่านโดยตรงจากหน่วยความจำภายใน โดยไม่ต้องเสียเวลาในไซเคิลที่ทำงานภายนอก ในการทำงานกับบัสภายนอก 68020 มีหน่วยความจำแคชขนาด 64 เวิร์ด ทุกๆเวิร์ดประกอบด้วย tag field ,valid bit และส่วนที่เก็บคำสั่ง 32 บิต ส่วนของ tag field ประกอบด้วยส่วนของบิตแอดเดรสส่วนบน 24 บิ และค่าของ FC2 ซึ่งเป็นระหัสฟังก์ชันที่กำหนดออกไปทางขา FC2 ของชิพ นั่นคือหน่วยความจำ 4 กิกะไบต์ของ 68020 แบ่งออกเป็นบล็อกๆละ 256 ไบต์ ซึ่งเมื่อเทียบกับแคชมี 64 เวิร์ด แบบยาวซึ่งก็คือ 256 ไบต์

เมื่อ 68020 ทำการเฟตคำสั่ง จะกระทำในลักษณะเหมือนกับติดต่อกับหน่วยความจำภายนอก และติดต่อกับหน่วยความจำแคชไปพร้อมกัน CPUจะทำการตรวจสอบแคชเพื่อดูว่าแอดเดรสของคำสั่งที่ต้องการอยู่ในแคชหรือไม่ การตรวจสอบนี้ทำได้โดยดูจาก tag field ที่ใช้สำหรับอ้างอิงกับแอดเดรส 24 บิตบน โดยเปรียบเทียบกับค่าแอดเดรสที่ส่งออกมา และหากตรงกันก็จะเซต valid bit เพื่อแสดงสถานะ hit หลังจากนั้นก็อ่านข้อมูลจากแคชและส่งสัญญาณจบไซเคิลเพื่อทำงานไซเคิลต่อไปได้

หากข้อมูลใน tag field ไม่ตรงกับแอดเดรสก็จะแสดงสถานะ miss เป็นผลทำให้ valid bit ได้รับการเคลียร์ และจะทำงานไซเคิลภายนอกในการอ่านข้อมูลจากหน่วยความจำ

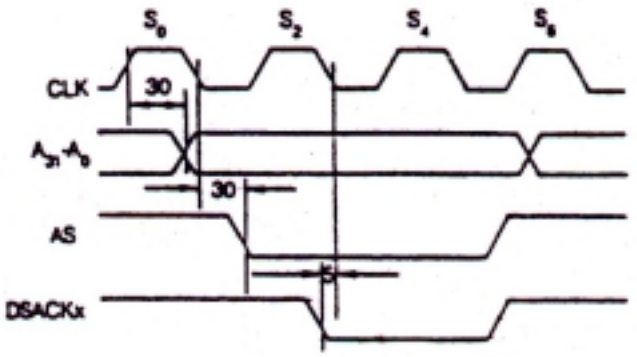
การใช้แคชภายนอกชิพ

การออกแบบแคชภายนอกร่วมกับ 68020 ทำเพื่อเพิ่มประสิทธิภาพโดยรวมของหน่วยความจำแคชที่ใช้เป็นได้ทั้งที่กำหนดแอดเดรสแบบลอจิกหรือพิสิคัลก็ได้ การเพิ่มหน่วยความจำแคชนี้จะเพิ่มในลักษณะที่เป็นที่เก็เฉพาะคำสั่งหรือข้อมูลอย่างเดียวก็ได้

ในที่จะกล่าวถึงหลักการของการใช้หน่วยความจำแคชที่เป็นแบบลอจิคัลสำหรับข้อมูลวิธีนี้ไม่ใช้วิธีที่ดีทีสุดสำหรับการต่อหน่วยความจำแคชกับ 68020 แต่เป็นระบบที่เข้าใจระบบการเชื่อมต่อกับสัญญารได้ง่าย

โดยปกติ การใช้หน่วยความจำหลักของ CPU สเปกของช่วงเวลาที่วิกฤติจะอยู่ในช่วงเวลาจาก address valid ไปจนถึง data valid แต่สำหรับเมื่อใช้กับหน่วยความจำแคชจึงเป็นเรื่องการหาทางลด wait state ลงในช่วงเวลาจาก address valid ไปยังช่วง data valid โดยเมื่อพบข้อมูลในแคช (hit) ก็จะไปทริกให้ไซเคิลการทำงานนั้นสิ้นสุดลง สำหรับตัว 68020 เหตุการณ์นี้จะเกิดขึ้นได้โดยสัญญาณที่ส่งมาบอก CPUให้เสร็จสิ้นบัสไซเคิลคือ DSACKX ซึ่งก็คือสัญญาณออกการถ่ายเทข้อมูลเสริมแล้ว หรือ BERR คือสัญญาณบัสมีข้อผิดพลาด และสัญญาณ HALT ซึ่งจะหยุดระบบ

ในกรณีนี้เมื่อ CPUส่งแอดเดรสออกมาเพื่อต้องการอ่านข้อมูลในแอดเดรส ข้อมูลแอดเดรสนี้จะได้รับการ



เปรียบเทียบหรือเข้าไปแมตช์กับหน่วยความจำแคช หากพบ(hit) หน่วยความจำแคชก็จะส่งสัญญาณ DSACK ซึ่งเป็นผลมาจากสัญญาณ HIT ที่มาจากวงจรแคชเมื่อเกิดการ hit การแอกเซสข้อมูลจากหน่วยความจำก็เป็นอันยกเลิก หากไม่พบในหน่วยความจำแคช หรือเกิด miss ไซเคิลการทำงานกับหน่วยความจำก็เกิดขึ้นจนครบ

จากรูปเป็นสัญญาณการทำงานที่ความถี่สัญญาณนาฬิกา 16.67 MHz และสัญญาณที่แสดงนี้ไม่มี wait state สังเกตช่องว่างที่ใช้ใน s0 s2 s4 เมื่อCPUส่งแอดเดรส a31-a0 ออกมาและมีการแสดงแอดเดรสด้วยสัญญาณสโตรบ AS หน่วย ความจำแคชใช้เวลาอีกประมาณ 55 ns เพื่อตรวจสอบแอดเดรส หลังจากนั้นก็ส่งสัญญาณ DSACK กลับออกมาเพื่อยืนยันการ hit เมื่อยืนยันแล้วข้อมูลจากหน่วยความจำแคชต้องพร้อมก่อนสัญญาณขาลงของ s4 อย่างน้อย 5 นาโนวินาที

การอับเกรดแคช

cache คือหน่วยความจำความเร็วสูงที่อยู่ระหว่าง CPUและหน่วยความจำของคุณที่สมมติฐานสองข้อที่รองรับการใช้งานของหน่วยความจำ cache ในการเพิ่มประสิทธิภาพการทำงานของระบบคือ

1 เมื่อเพิ่มโปรแกรมติดต่อเข้ากับตำแหน่งของหน่วยความจำปกติ
มันก็มักจะติดต่อเข้ากับตำแหน่งเดิมอีกครั้งต่อไป

2 เมื่อใดก็ตามที่โปรแกรมติดต่อเข้ากับตำแหน่งของหน่วยความจำปกติมันก็มักจะติดต่อเข้ากับตำแหน่งที่ใกล้เคียงกันด้วยในครั้งต่อไป

ดังนั้น การเก็บตำแหน่งที่มีความถี่ในการใช้งานบ่อยๆ ไว้ในหน่วยความจำความเร็วสูงหรือแคชก็สามารถเพิ่มประสิทธิภาพการทำงานของระบบขึ้นได้เช่นกัน

ในเครื่องทดสอบของเรา ไม่มีหน่วยความจำแคชที่มีเพียงแต่ช็อกเกตว่างๆที่สามารถอัปเกรดหน่วยความจำนี้ต้องการให้เราพิจารณษเลือกชิพที่เราใช้ให้แน่นอน ปัญหาในเรื่องนี้ก็คือระบบแคชของคุณประกอบขึ้นจากชิพสองส่วนด้วยกันคือ

cache RAM เป็นที่เก็บบรรจุสิ่งต่างๆ ของหน่วยความจำแคช ขณะที่ tag RAM เป็นเหมือนตัวชี้ (index) ไปที่ cache RAM เป็นตัวแสดงว่าอะไรเก็บอยู่ในนั้น การซื้อชิปที่แตกต่างกันและสับสนได้การอัปเกรดในส่วนนี้แนะนำว่าทางที่ดีควรให้ผู้ชำนาญมาเป็นผู้เลือกซื้อและติดตั้งให้หรือติดต่อกับตัวแทนจำหน่ายเครื่องของคุณ

เหมือนดังเช่นการอัปเกรดส่วน CPU ก็คือ ทางเราค่อนข้างผิดหวังกับการอัปเกรดในส่วนนี้ มีความสามารถในการทำงานเพิ่มขึ้นเพียง 3 เปอร์เซ็นต์ บนระบบที่มีหน่วยความจำอยู่ 4 เมกะไบต์ การใส่ส่วนความจำแคช และเปลี่ยน CPU ที่เร็วขึ้นสร้างความแตกต่างที่ค้อนข้างมาก แต่มันไม่มากเท่ากับการเพิ่มของหน่วยความจำ ทางเราได้พิสูจน์แล้วว่า รายละเอียดของเครื่องที่อัปเกรดส่วนของแคช แล้วสามารถเพิ่มประสิทธิภาพได้เด่นชัดก็คือ ระบบเครื่องที่อย่างน้อยใช้ตัวประมวนผลความเร็ว 50 MHz และมีหน่วยความจำ 8 เมกะไบต์ ถ้าคุณได้อัปเกรดส่วนของ CPUและหน่วยความจำไปแล้ว การใส่หน่วยความจำแคชเข้าไปจาก 0 ไปเป็น 256 กิโลไบต์จะทำให้ประสิทธิภาพการทำงานของระบบดีขึ้นอีก 10 เปอร์เซ็นต์ สำหรับราคาที่ต้องจ่ายประมาณ 100 เหรียญสหรัฐ

การใช้ดิสก์แคช

ดิสก์แคชและแรมดิสก์เป็นโปรแกรมที่อยู่ในตระกูลเดียวกัน กล่าวคือมันจะใช้หน่วยความจำเกินพันไบต์ซึ่ง MS-DOS ไม่สามารถใช้ประโยชน์ได้ เพื่อสร้างพื้นที่หน่วยหนึ่ง ๆ เพื่อทำอะไรสักอย่างที่ทำให้ระบบรวมของระบบพีซีดีหรือเร็วขึ้น

ส่วนกรณ๊ดิสก์แคชหรือดิสก์บัฟเฟอร์หรือที่เก็บข้อมูลชั่วคราวของดิสก์ เป็นพื้นที่หน่วยความจำที่ใช้สำหรับเก็บข้อมูลจากดิสก์ที่อ่านขึ้นมา โดยเมื่อ MS-DOS อ่านข้อมูลจากดิสก์ สำเนาชุดที่หนึ่งจะถูกเก็บไว้บริเวณที่หน่วยความจำที่ดิสก์แคชนี้ เพื่อเวลา MS-DOS ต้องการอ่านข้อมูลเดิมอีกครั้งหรืออ่านข้อมูลที่อยู่ติดกับชุดข้อมูลชุดเดิมจะได้อ่านที่ดิสก์แคชแทน ซึ่งเร็วกว่าการอ่านจากดิสก์โดยตรง

การใช้โปรแกรมดิสก์แคช SMARTDrive

โปรแกรม SMARTDrive จะทำตัวเหมือนเป็นแคชสมบูรณ์แบบ คือ การอ่านแคชมันจะอ่านข้อมูลที่ซีพียูต้องการจากดิสก์ขึ้นมาเก็บไว้ที่ตัวมัน นอกจากนั้นมันจะดึงข้อมูลที่อยู่รายรอบข้อมูลที่เราต้องการอ่านขึ้นมาด้วย การอ่านข้อมูลครั้งต่อไปจะเป็นข้อมูลที่อยู่ติดกันหรือรายรอบจากข้อมูลนี้ สำหรับการเขียนแคช คือ การเขียนข้อมูลใด ๆ ที่จะเกิดกับดิสก์ การเกิดขึ้นกับดิสก์แคชก่อนเพื่อความรวดเร็ว จาก

นั้นเมื่อถึงเวลาหนึ่ง จึงมีการอัปเดตฮาร์ดดิสก์หรือฟลอปปี้ดิสก์ภายหลัง การสร้างดิสก์แคชสำหรับรูปคำสั่งของ SMARTDrive มีดังนี้

C:\dos\smartdrv.exe [[drive[+:-]…]

Drive เป็นชื่อของดิสก์ไดรฟ์ที่เราต้องการสร้างดิสก์แคช เช่น ไดฟ์ A: หรือไดฟ์ C:

การสร้างดิสก์แคช

การสร้างดิสก์ทำได้โดยเพิ่มคำสั่งอีกบรรทัดในไฟล์ AUTOEXEC.BAT ดังนี้

C:\dos\smartdrv/v

สวิตซ์ตัวเลือก /V หมายถึงให้แสดงสถานะการสร้างหรือการโหลดว่ามีข้อผิดพลาดอะไรหรือไม่ ซึ่งสวิตซ์นี้เราอาจเอาออกได้ถ้าแน่ใจว่าไม่มีอะไรผิดพลาดต่อไป อย่าลืมดูว่าไดเรกทอรี่ที่เรากำหนดใน คำสั่งนี้ (C:\DOS) มีไฟล์ SMARTDrive อยู่จริง

กระบวนการสร้างดิสก์แคชมีเพียงเท่านี้ เมื่อคุณแก้ไขไฟล์ AUTOEXEC.BAT เรียบร้อยแล้ว และบูตเครื่องใหม่คุณจะเห็นข้อความดังต่อไปนี้

Microsoft SMARTDrive Disk Cache version 4.1

Copyright 1991,1993 Microsoft Corp.

Cach Size:2,097,152 bytes

Cach size while runnimg Wimdows: 2,097,152 bytes

Disk Caching Status

Drive	read cache	write cache	buffering
A:	yes	no	no
B:	yes	no	no
C:	yes	no	no

For heip,type”Smartdrv /?”

The memory-resident portion oF SMARTDrive s loaded

ถ้าคุณพิมพ์คำสั่ง smartdrv ที่ดอสพรอมต์ จะเป็นการตรวจสอบสถานะแคชของไดรฟ์ต่างๆซึ่งบางทีคุณต้องอ่านตรวจสอบดู หรือเพื่อหยุดใช้งานมันชั่วคราวโดยใช้สวิตซ์ตัวเลือก-(ให้ออกจาก Windows ก่อนจึงเลือกใช้สวิตซ์นี้ได้) ตัวอย่างเช่น

smartdrv/s

จะเป็นการรายงานสถานะเพิ่มเติมของโปรแกรมนี้ โดยจะมีการบอก cache hit ซึ่งเป็นค่าบอกจำนวนครั้งที่มีการอ่านดิสก์แคชแทนการอ่านดิสก์ และ cache misses บอกจำนวนครั้งที่อ่านดิสก์เพราะหาข้อมูลที่ต้องการในดิสก์แคชไม่ได้ สัดส่วนระหว่างเลขสองจำนวนนี้ (cache hits หารด้วย cache misses จะเป็นตัวบอกประสิทธิภาพของโปรแกรม SMART Drive ที่มีต่อระบบของคุณ)

ถ้าพิมพ์ smartdrv / c จะเป็นการสั่งให้ SMART Drive ข้อมูลลงดิสก์ทันที แล้วล้างหน่วยความจำแคชสำหรับเขียน คำสั่งนี้เหมาะสำหรับทุกครั้งที่ต้องการจะปิดเครื่อง และถ้าพิมพ์ smartdv / r จะเป็นการรีเซตหรือสั่งให้โปรแกรม SMART Drive ทำตัวเหมือนกับเพิ่งเปิดเครื่องใหม่ ๆ

ถ้าคุณมีหน่วยความจำมากพิเศษ ขอให้ทำเป็นดิสก์แคชแทนเพื่อนำหน่วยความจำที่ไม่ได้ใช้มาทำประโยชน์ และเพื่อเพิ่มประสิทธิภาพโดยรวมรวมของระบบคอมพิวเตอร์ ดิสก์แคชช่วยเร่งความเร็วในการอ่านเขียนฮาร์ดดิสก์ให้เร็วขึ้น

- ดิสก์แคชเป็นส่วนหนึ่งของหน่วยความจำ ใช้เก็บสำเนาข้อมูลที่อ่านจากดิสก์ เผื่อกรณีที่ต้องการอ่านข้อมูลชุดเดิม หรือข้อมูลชุดเดิมจะไม่เสียเวลาอ่านดิสก์แคชแทน ด้วยวิธีนี้ทำให้เครื่องพีซีทำงานได้ดีและเร็วขึ้น
- โปรแกรมดิสก์แคชที่มาพร้อมกับ MS-DOS ก็คือ SMART Drive มันจะสร้างแคชสำหรับฟลอปปี้ดิสก์และฮาร์ดดิสก์โดยอัตโนมัติ ประโยชน์ก็คือเพิ่มความสามารถของฮาร์ดดิสก์ให้ดียิ่งขึ้น ทำงานเร็วขึ้น

การเร่งความเร็วดิสก์ด้วย DISK CACHING

การทำ DISK CACHING โดยการที่หน่วยความจำบางส่วนไว้เป็นที่เก็บข้อมูลที่ต้องอ่านจากดิสก์บ่อยๆ อาศัยหลักการที่ว่าในการทำงานกับดิสก์นั้นในการทำงานระยะหนึ่งๆ จะมีการอ่าน/เขียนข้อมูลกับไฟล์เพียงจำนวนจำกัดเท่านั้น ไม่ได้ใช้ทุกไฟล์บนดิสก์ทั่วไปหมด และที่ใช้ในแต่ละไฟล์ก็อาจไม่ได้ใช้ทั่วไปทั้งไฟล์แต่ใช้เฉพาะบางส่วน ดังนั้น หากสามารถก็เอาข้อมูลส่วนที่ใช้บ่อยๆเหล่านั้นใช้ในส่วนความจำที่เป็น RAM แทน เวลาที่จะเลือกใช้อีกครั้งหนึ่งก็จะไม่เสียเวลาอ่านจากดิสก์ใหม่ แต่จะอ่านจาก RAM ได้ทันที ทำให้ทำงานได้เร็วขึ้นนับร้อยเท่า หน่วยความจำ RAM นี่เรียกว่าเป็น Cache Memory สำหรับดิสก์หรือ Disk Cache นั่นเอง

หน่วยความจำที่กันไว้นี้มีขนาดเล็กกว่าดิสก์มาก ดังนั้นข้อมูลที่จะเก็บไว้เป็นเฉพาะที่ใช้บ่อยจริงๆ โดยจะมีซอฟต์แวร์ที่อยู่ในลักษณะของ ดีไวซ์ไดร์เวอร์ ตัวหนึ่งคอยดูแลเรื่องนี้ให้ ส่วนใดที่ไม่ได้ใช้นานแล้วจะถูกเอาออกไป เพื่อให้มีที่ว่างสำหรับส่วนที่ใช้บ่อย ๆ แทน หลักการจัดสรรที่แบบนี้เรียกว่า Least Recently Used (LRU) คือส่วนที่ใช้น้อยที่สุดก็จะถูกเอาออกไปก่อน การแบ่งเนื้อที่ใน Disk Cache นี้จะแบ่งเป็นส่วน ๆ เรียกว่า page เวลาเอาข้อมูลเข้าหรือออกจาก Disk Cache ก็จะทำทั้ง page ขนาดของ Disk Cache และ page นี้อาจจะขึ้นอยู่กับซอฟต์แวร์ที่ดูแลเรื่องนี้แต่ละตัว เช่น Disk Cache อาจมีขนาด 64 กิโลไบต์ โดยแบ่งเป็น 16 page มีขนาด page ละ 4 กิโลไบต์หรือเท่ากับ 8 เซคเตอร์บนดิสก์ก็ได้ เป็นต้น

การอ่านโปรแกรมจากดิสก์โปรแกรมดังกล่าวจะมาดูในหน่วยความจำในส่วนที่เป็น Disk Cache ก่อน หากมี page ใดมีข้อมูลที่ต้องการอยู่ก็จะส่งโปรแกรมที่ต้องการข้อมูลนั้นนำไปใช้ได้เลย หากไม่มีก็จะไปอ่านมาจากดิสก์ และนำมาเก็บแทน page ที่ใช้น้อยที่สุดใน Disk Cache ขณะนั้น ส่วน page ที่จะถูกแทนที่นั้นก็จะถูกนำกลับไปเขียนลงในดิสก์ก่อน หรือถูกทับไปเฉย ๆ เลยก็ขึ้นอยู่กับว่าเวลาที่เราต้องการเขียนข้อมูลลงบนดิสก์นั้น ซอฟต์แวร์ที่เป็น Desk Caching จัดการให้เราอย่างไรซึ่งเราทำได้ 2 วิธีใหญ่ ๆ คือ

1. Write-through วิธีนี้เมื่อเขียนข้อมูลกับดิสก์จะเขียนลงทั้งใน Disk Cache และดิสก์จริงเลย ทำให้ข้อมูลทั้งสองถูกต้องตรงกันเสมอ วิธีนี้มีข้อดีคือ ถ้าเกิดไฟดับหรือเหตุขัดข้อง ข้อมูลสุดท้ายที่บันทึกไปจะไม่สูญหายเพราะเก็บไว้ในดิสก์แล้ว ถ้าซอฟต์แวร์ที่ทำ Disk Caching ใช้วิธีนี้เวลาเอา page ใหม่เข้ามาก็จะถูกทับแทนที่ได้เลย ไม่ต้องนำ page เก่าเขียนลงดิสก์อีก แต่วิธีนี้จะไม่ช่วยให้การเขียนข้อมูลกับดีสก์เร็วขึ้น คงเร็วขึ้นเฉพาะตอนอ่านเท่านั้น
2. Write-Cache คือการเขียนข้อมูลกับดีสก์จะทำเฉพาะหน่วยความจำที่เป็น Disk Cache เท่านั้นยังไม่ลงดีสก์จริงจนกว่า page นั้นจะถูกของใหม่เข้ามาแทนที่ จึงจะนำไปเขียนในดีสก์จริง วิธีนี้จะทำการเขียนข้อมูลกับดีสก์เร็วขึ้นเท่า ๆ กับการ read แต่มีข้อเสียคือ ถ้าเกิดการขัดข้อง เช่น ไฟฟ้าดับหรือเครื่อง hang อาจจะทำให้ข้อมูลสุดท้ายหายไปและข้อมูลในดีสก์อาจไม่ใช้ข้อมูลล่าสุดที่เราทำงานค้างไว้ก็ได้

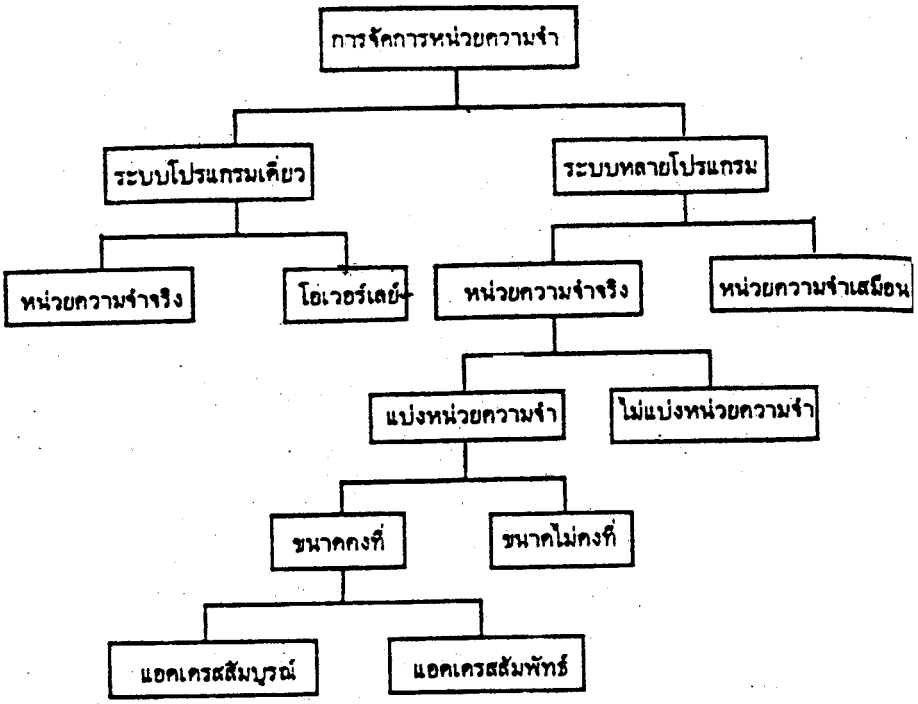
Disk Caching จะมีการ read / write กับดิสก์จริงแทรกอยู่เป็นระยะ ๆ แต่มีข้อดีคือไม่ต้องห่วงว่าที่ใน RAM จะเต็มเพราะซอฟต์แวร์จะดูแลการใช้ที่ใน Disk Cache ให้โดยอัตโนมัติ ส่วนใดไม่ใช้บ่อยก็นำออกไป Disk Caching ผู้ใช้ยังไม่ต้องจัดการก๊อปปี้ไฟร์ข้อมูลที่จะใช้ไปใส่ RAM Disk ในตอนต้นและก๊อปปี้มาใส่ดิสก์จริงก่อนปิดเครื่อง รวมทั้งไม่ต้องจัดการตั้งให้โปรแกรมใช้งาน RAM Disk

หน่วยความจำเสมือน

ระบบคอมพิวเตอร์ถูกพัฒนาให้รันโปรแกรมได้หลาย ๆ โปรแกรมพร้อมกันโปรแกรมแต่ละโปรแกรมจึงแย่งกันใช้หน่วยความจำ เมื่อมีโปรแกรมเข้ามาในระบบเพิ่มมากขึ้น ปัญหาที่ตามมาคือ ไม่มีหน่วยความจำเพียงพอสำหรับโปรแกรมเหล่านี้ วิธีที่ง่ายที่สุดคือ การเพิ่มหน่วยความจำเข้าไปในระบบแต่จะทำให้เสียค่าใช่จ่ายมากเกินไป เพราะหน่วยความจำมีราคาแพงและต้องเพิ่มเป็นจำนวนมากด้วยและในบางกรณีเราเพิ่มหน่วยความจำเต็มขีดความสามารถของโครงสร้างระบบแล้ว หน่วยความจำก็ยังไม่พอใช้งาน เมื่อเป็นเช่นนี้นักคอมพิวเตอร์จึงคิดหาวิธีอื่นที่จะแก้ปัญหาหน่วยความจำไม่พอ วิธีที่ทำกันมากในระบบใหญ่คือ การทำหน่วยความจำเสมือน (Virtual Memory)

ประเภทของการจัดการหน่วยความจำ

เพื่อให้เข้าใจถึงการจัดการหน่วยความจำประเภทต่างที่กล่าวมา เราจะแบ่งแยกหน่วยความจำได้ดังรูป



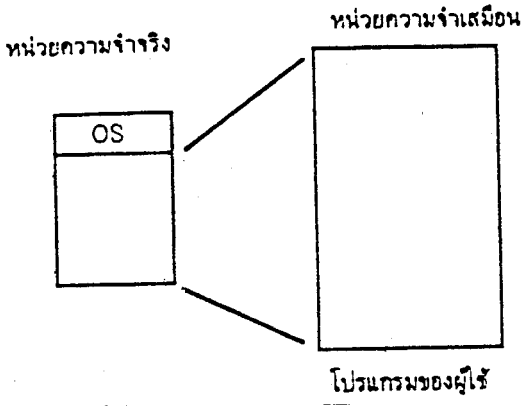
รูป การจัดการหน่วยความจำ

ในระบบโปรแกรมเดี่ยว การจัดการหน่วยความจำแบบหน่วยความจำจริงก็คือการใช้หน่วยความจำแบบธรรมดด โดยที่โปรแกรมของผู้ใช้จะไม่โตเกินขนาดของหน่วยความจำที่มีอยู่ ส่วนการทำโอเวอร์เลย์นั้นจะต่างกับการทำหน่วยความจำจริงตรงที่ขนาดหน่วยความจำที่มีอยู่ได้โดย การโหลดเอาส่วนที่ใช้งานลงไปก่อน ส่วนใหนที่ยังไม่ได้ใช้เก็บไว้ในดิสก์ ซึ่งทั้งหมดนี้ผู้ใช้จะต้องเขียนโปรแกรมให้โปรแกรมเป็นผู้จัดการเองทั้งหมด ลักษณะการจัดการแบบโอเวอร์เลย์นี้มีลักษณะการทำงานเหมือนหน่วยความจำเสมือน

ในระบบหลายโปรแกรม การจัดหน่วยความจำแบบเพิ่มหน่วยความจำจริงนั้น แบ่งเป็นการจัดการแบ่งพื้นที่ในหน่วยความจำ และไม่ได้แบ่งหน่วยความจำ การแบ่งหน่วยความจำยังแบ่งย่อยได้อีก 2 ประเภทคือ ส่วนแบ่งย่อยมีขนาดแน่นอน และขาดไม่คงที่ อย่างไรก็ตามมีข้อจำกัด เช่นเดียวกับในโปรแกรมเดี่ยว คือ ขนาดของโปรแกรมจะต้องไม่โตเกินกว่าขนาดของหน่วยความจำที่มีอยู่ในระบบ ดังนั้นเราอาจแบ่งการจัดการหน่วยความจำออกเป็น 2 ประเภทใหญ่ ๆ คือ ระบบหน่วยความจำจริง และระบบหน่วยความจำเสมือน โดยที่ในระบบหน่วยความจำจริง ขนาดของโปรแกรมจะต้องไม่โตเกินกว่าขนาดของหน่วยความจำที่มีอยู่ ลบด้วยขนาดของหน่วยความจำที่เป็นส่วนของ OS อยู่ ส่วนระบบหน่วยความจำเสมือน ขนาดของโปรแกรมจะโตเท่าใดก็ได้

แนวคิดของหน่วยความจำเสมือน

ผู้เขียนโปรแกรมต้องไม่สนใจว่าในระบบคอมพิวเตอร์ที่เขาใช้อยู่นั้นมีหน่วยความจำขนาดเท่าใด เขารับทราบเพียงว่ามีหน่วยความจำขนาดมหาศาลให้เขาใช้ได้ OS จะจัดการให้ผู้ใช้สามารถใช้หน่วยความจำได้มากกว่าหน่วยความจำจริงที่มีอยู่มากหลายเท่าตัว ตัวอย่างเช่น ในระบบมีหน่วยความจำอยู่ 10 เมกกะไบต์ แต่ผู้ใช้สามารถเขียนหรือรันโปรแกรมขนาด 100-1000 เมกกะไบต์ได้ ดังรูปต่อไปนี้



รูป หน่วยความจำเสมือนมีขนาดใหญ่กว่าหน่วยความจำจริง

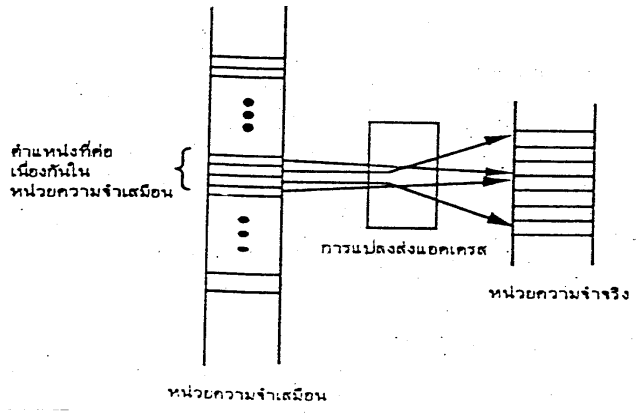
แนวคิดของการทำหน่วยความจำเสมือนนี้มีลักษณะการทำเช่นเดียวกับการทำโอเวอร์แลย์ นั่นคือ โปรแกรมทั้งโปรแกรมไม่ได้ถูกใช้งานพร้อมกันหมด ตัวอย่างเช่น โปรแกรมทั่ว ๆ ไปจะทำงานจากส่วนต้นโปรแกรมค่อย ๆ เลื่อนมาจนกระทั่งท้ายโปรแกรม ในขณะที่โปรแกรมกำลังทำงานอยู่ที่ส่วนต้นโปรแกรม ที่ปลายของโปรแกรมยังไม่ถูกใช้งาน ดังนั้นในช่วงที่โปรแกรมทำงานอยู่ที่ส่วนต้น ๆ ก็ไม่มีความจำเป็นต้องโหลดเอาส่วนท้าย ๆ โปรแกรมให้ลงไปในหน่วยความจำให้เปลืองเนื้อที่ เมื่อโปรแกรมทำงานมาจนกระทั่งถึงส่วนที่ยังไม่ได้โหลดเข้าสู่หน่วยความจำ OS จะทำการดึงส่วนนั้นมาจากหน่วยความจำสำรอง ซึ่งเก็บโปรแกรมทั้งโปรแกรมไว้ โดยที่อาจจะโหลดไปทับส่วนต้น ๆ ของโปรแกรมก็ได้

จากหลักการนี้เราสามารถรันโปรแกรมที่มีขนาดใหญ่กว่าหน่วยความจำที่มีอยู่ได้ ข้อแตกต่างระหว่างการทำโอเวอร์เลย์กับระบบหน่วยความจำเสมือน คือ แบบโอเวอร์เลย์ผู้ใช้จะต้องเขียนโปรแกรมจัดการเอง ส่วนระบบหน่วยความจำ OS จะจัดการให้เองทั้งหมด ผู้ใช้ไม่ต้องรับรู้การทำงานส่วนนี้

การแปลงส่งแอดเดรส (address mapping)

เนื่องจากขนาดของโปรแกรมมีขนาดโตกว่าขนาดของหน่วยความจำ ดังนั้น แอดเดรสที่อ้างอิงภายในโปรแกรมจึงต่างกับแอดเดรสของหน่วยความจำจริง แอดเดรสในหน่วยความจำจริงนั้น เรียกว่าแอดเดรสจริง ส่วนแอดเดรสที่อ้างอิงภายในโปรแกรม เรียกว่า แอดเดรสเสมือน

โปรแกรมจะมีการอ้างแอดเดรสด้วยแอดเดรสเสมือนเท่านั้น แต่ว่าโปรแกรมซึ่งรันอยู่บนหน่วยความจำจริงซึ่งใช้แอดเดรสจริง ดังนั้นจึงต้องมีกลไกการแปลงแอดเดรสเสมือนให้เห็นแอดเดรสจริง ณ โปรแกรมส่วนนั้นถูกวางลงไป วิธีการแปลงนี้เรียกว่า การแปลงส่งแอดเดรส ดังรูป ถึงแม้ว่าแอดเดรสเสมือนในโปรแกรมจะเรียงลำดับต่อเนื่องกัน แต่ไม่จำเป็นเมื่อแอดเดรสเหล่านั้นถูกแปลงเป็นแอดเดรสจริงแล้วจะต้องเรียงต่อกันด้วย ลักษณะเช่นนี้เรียกว่า การต่อเนื่องเทียม (artificial contiquity)



รูป แอดเดรสแม็ปปิ้งและความต่อเนื่องเทียม

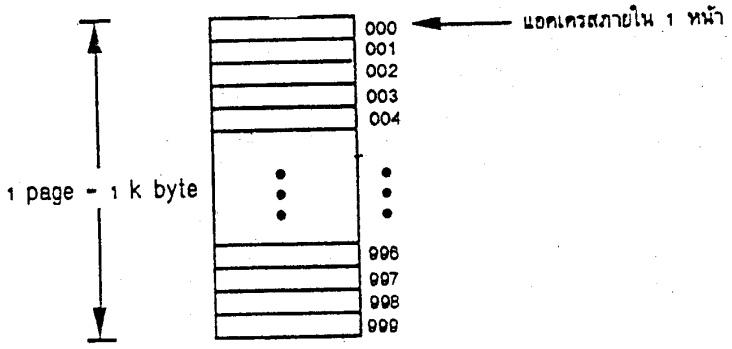
การแบ่งบล็อก

OS จะแบ่งโปรแกรมออกเป็นส่วนย่อย ๆ หลาย ๆ ส่วนเรียกว่า บล็อก (block) ถ้าแบ่งให้แต่ละบล็อกเท่ากันหมดเรียกว่า page แต่มีขนาดไม่เท่ากันจะเรียกว่า เซกเมนต์ (segment) โดยทั่วไปเซกเมนต์จะมีขนาดใหญ่กว่าหน้ามาก ๆ แต่มีกลไกการทำงานที่ยุ่งยากกว่า

หน่วยความจำเสมือนระบบหน้า (paging system)

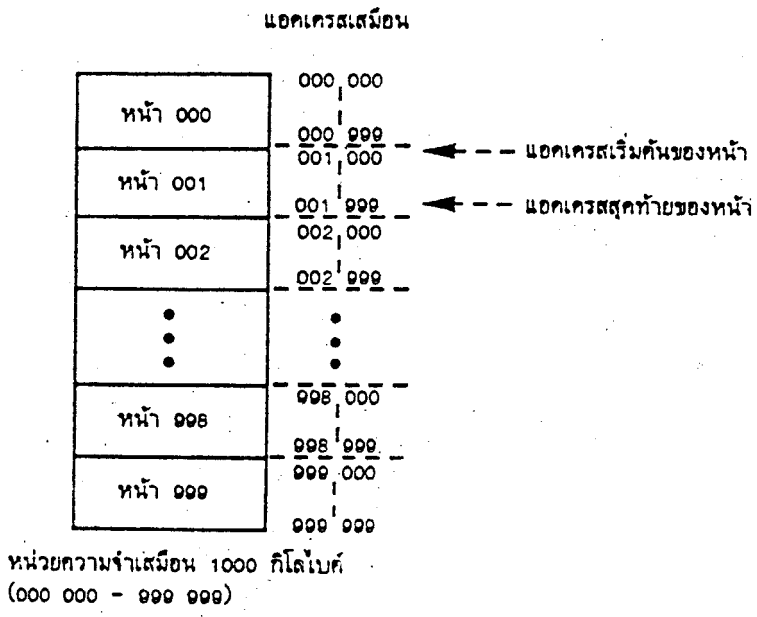
ในระบบที่แบ่งบล็อกเป็นหน้า มีการทำงานที่ง่ายและเหมาะแก่การจะใช้อธิบายให้เกิดความเข้าใจในหลักการทำงาน

สมติว่าระบบมีหน่วยความจำอยู่ 100 กิโลไบต์ แต่อาศัยระบบหน่วยความจำเสมือน ทำให้ผู้เขียนสามารถเขียนหรือรันโปรแกรมที่มีขนาด 1000 กิโลไบต์ ดังนั้นแอดเดรสเสมือนจะเป็นเลข 6 หลัก ส่วนแอดเดรสจริงมีเพียง 5 หลัก สมติว่า OS กำหนดว่า 1 หน้ามีขนาด 1 กิโลไบต์ ถ้าจะอ้างถึงแอดเดรสต่าง ๆ ภายใน 1 หน้าจะต้องใช้เลข 3 หลัก ดังรูป

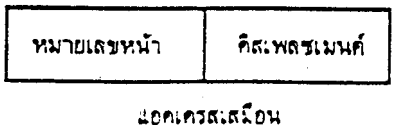


รูป แอดเดรสภายในหน้า

เนื่องจากหน่วยความจำเสมือนมีขนาดโตได้ถึง 1000 กิโลไบต์ ดั้งนั้นโปรแกรมจึงมีจำนวนเท่ากับ 1000 หน้า คือตั้งแต่หน้า 000 ถึง 999 ดังรูปด้านล่าง ขอให้สังเกตแอดเดรสเสมือนที่แสดงไว้ เลข 3 หลักนั้นจะเป็นค่าแอดเดรสใน ๆ หน้านั้น ๆ ส่วนที่ 2 นี้ เรียกว่า ดีสเพรสเมนต์ (displacement) เช่น แอดเดรสเสมือนที่ 243765 มีหมายเลขหน้าเป็น 243 และมีดีสเพรสเมนต์ 765

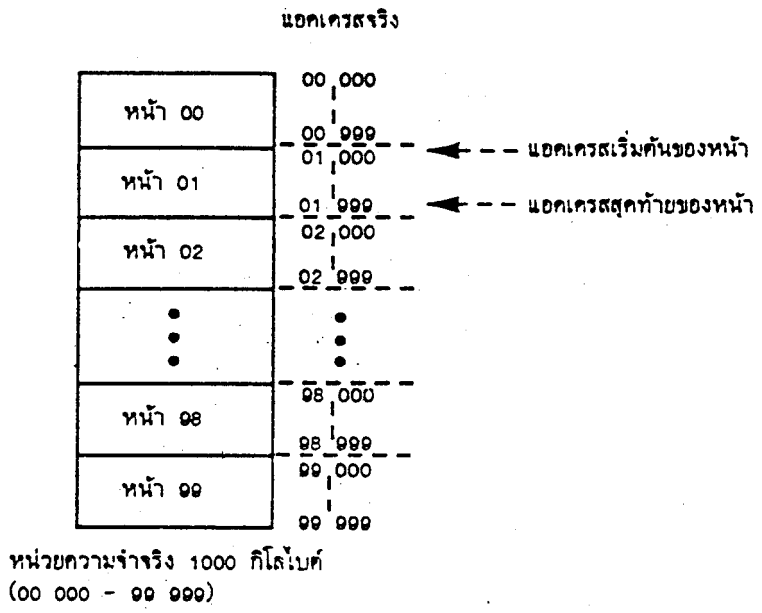


รูป การแบ่งหน้าในหน่วยความจำเสมือน



รูป การแบ่งแอดเดรสเสมือน

ในส่วนของหน่วยความจำจริง OS ก็ต้องแบ่งหน่วยความจำออกเป็นหน้า และมีขนาดเท่ากับหน้าในหน่วยความจำเสมือนเช่นกัน เนื่องจากหน่วยความจำจริงมีขนาด 100 กิโลไบต์ ดังนั้นหน่วยความจำจริงจึงมีขนาดเท่ากับ 100 หน้า คือตั้งแต่หน้า 00 ถึง 99 ดังรูปด้านล่าง จะสังเกตว่า 2 หลักแรกของแอดเดรสจริงบ่งบอกถึงหมายเลขหน้า และ 3 หลักหลังจะเป็นดีสเพรสเมนต์ภายในหน้า ซึ่งมีลักษณะเหมือนกับในหน่วยความจำเสมือน



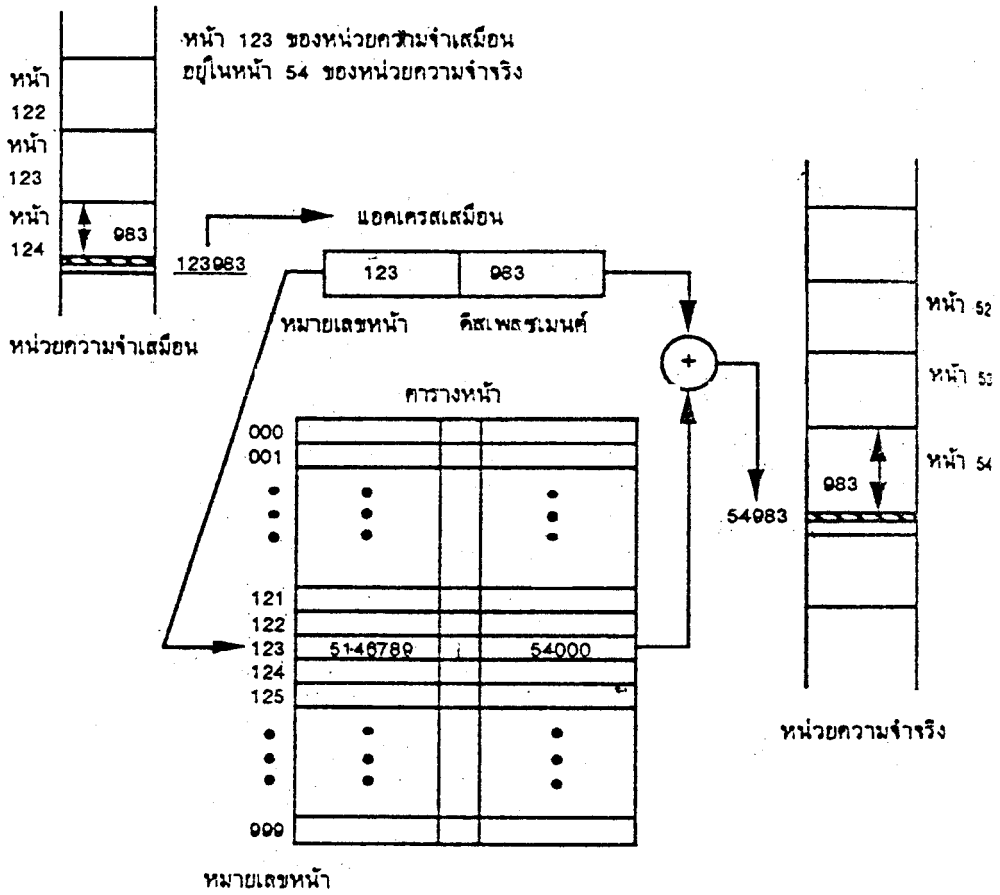
รูป การแบ่งหน้าในหน่วยความจำจริง

โปรแกรมของผู้ใช้ไม่ว่าจะมีขนาดแค่ไหนก็ต้องถูกเก็บไว้ในหน่วยความจำสำรอง ซึ่งส่วนมากจะเป็นดิสก์เพราะเป็นอุปกรณ์เก็บข้อมูลที่มีความเร็วสูง เมื่อผู้ใช้สั่งรันโปรแกรม OS จะโหลดเอาโปรแกรมจากดิสก์เข้าไปในหน่วยความจำครั้งละ 1 หน้า นั้นคือ 1 หน้าโปรแกรมจะเข้าไปใช้เนื้อที่ในหน่วยความจำ 1 หน้า เช่นกัน เมื่อโปรแกรมหนึ่งครอบครองหน้าไหนในหน่วยความจำแล้ว โปรแกรมอื่นจะใช้หน้านั้นอีกไม่ได้ โปรแกรมจะต้องครอบครองหน่วยความจำจริงเป็นจำนวนเต็มของหน้า

วิธีแปลงส่งแอดเดรสแบบ DAT OS จะสร้างตาราวไว้ให้แต่ละโปรเซสตารางนี้เรียกว่า ตารางหน้า (page table) ดังรูปต่อไป ระบบที่สมมตินี้ หน่วยความจำเสมือนมีขนาด 1000 หน้า ดังนั้นตารางหน้านี้จะมีอยู่ 1000 ช่อง ช่องที่000 ถึง 999 ช่อง 000 สำหรับหน้า 000, ช่อง 001 สำหรับหน้า 001, และต่อ ๆ ไป ในตารางจะมี 3 คอลัมน์ คอลัมน์แรกจะบอกที่อยู่ของหน้านั้น ๆ ในดิสก์ เพื่อที่จะได้รู้ว่าหน้าต่าง ๆ ของโปรแกรมเก็บไว้ที่ไหนในดิสก์ คอลัมน์ที่ 2 จะบ่งบอกว่าหน้านั้น ๆ อยู่ในหน่วยความจำจริงหรือไม่ เช่น ถ้าในคอลัมน์ที่ 2 เป็น 0 หมายถึงหน้านั้นไม่ได้อยู่หน่วยความจำ ถ้าค่าในคอลัมน์ที่ 2 เป็น 1 หมายถึง หน้านั้นอยู่ในหน่วยความจำ ซึ่งจะอยู่ที่ไหนนั้นดูได้จาก คอลัมน์ที่ 3 คอลัมน์ที่ 3 จะบ่งบอกถึงแอดเดรสเริ่มต้นของหน้าในหน่วยความจำจริง

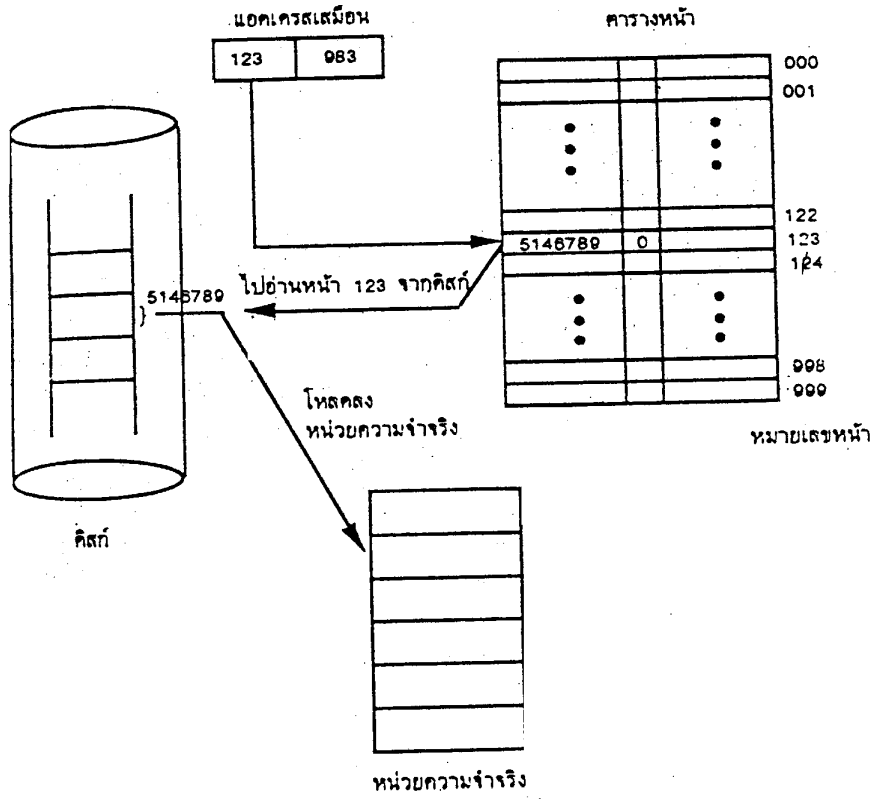
สมมติว่าต้องการทราบว่าแอดเดรสเสมือน 123983 ไปอยู่ ณ ที่ใดในหน่วยความจำ OS จะนำเอาแอดเดรสเสมือนมาหาค่าหมายเลขหน้าและดีสเพรสเมนต์ ในกรณีนี้หมายเลขหน้าคือ 123 และดีสเพรสเมนต์ คือ 983 OS ตรวจสอบที่ตารางหน้าในช่องที่ 123 ดังรูปต่อไป สมมติว่าหน้านี้อยู่ในหน่วยความจำจริงแล้ว คอลัมน์ที่ 2 จะมีค่าเป็น 1 จากนั้น OS จะอ่านค่าในคอลัมน์ที่ 3ของตารางค่าเพื่อหาค่าแอดเดรสเริ่มต้นของหน้าในหน่วยความจำจริง สมมติว่าหน้า 123 ของโปรแกรมอยู่ที่หน้า 54 ของหน่วย

ความจำจริง ค่าในคอลัมน์ที่ 3 ก็จะมีค่าเป็น 54000 OS ก็จะเอาค่าแอดเดรสในช้องที่ 3 นี้มาบวกกับดีสเพรสเมนต์ของแอดเดรสเสมือน (54000+983) ก็จะได้ 54983 ในหน่วยความจำจริงที่ตรงกับแอดเดรส 123983 ของโปรแกรม (ในหน่วยความจำเสมือน)



รูป การแปลงแอดเดรสในระบบหน้า

ถ้าสมมติว่าหน้า 123 ไม่ได้อยู่ในหน่วยความจำ คอลัมน์ที่ 2 จะมีค่าเป็น 0 เมื่อ OS พบว่าหน้า 123 ไม่ได้อยู่ในหน่วยความจำ OS จะอ่านค่าอยู่ในคอลัมน์ที่ 1 คือ 5146789 จากนั้น OS ก็จะไปอ่านข้อมูลที่แอดเดรส 5146789 ในดิสก์ เป็นจำนวนหนึ่งหน้า (1 กิโลไบต์) และหาหน้าที่ว่างในหน่วยความจำจริง สมมติว่าหน้า 03 ในหน่วยความจำจริงว่างอยู่ OS ก็จะนำหน้า 128 ซึ่งอ่านมาจากดิสก์ไปวางที่หน้า 03 และ OS ทำการแก้ข้อมูลในตารางหน้าที่ช่อง 123 โดยเปลี่ยนคอลัมน์ที่ 2 เป็น 1 และใส่ค่า 03000 ลงในคอลัมน์ที่ 3 ดังรูปต่อไป ต่อไปนำค่าดีสเพรสเมนต์ไปบวกกับค่าแอดเดรสเริ่มต้นของหน้า 03 (03000+983) ได้ค่าแอดเดรสจริง 03983 ซึ่งตรงกับแอดเดรสเสมือน 123983 ของโปรแกรม



รูป หน้า 123 ไม่อยู่ในหน่วยความจำ

ถ้าทุกหน้าในหน่วยความจำจริงถูกใช้หมด ก็จะไม่มีหน้าว่างสำหรับเอาหน้า 123 ของโปรแกรมลงไปวาง ในกรณีนี้ OS จำเป็นต้องเลือกเอาหน้าใดหน้าหนึ่งในหน่วยความจำจริงเพื่อเอาหน้า 123 ไปวางทับ หน้าใดจะถูกเลือกนั้นขึ้นอยู่กับยุทธวิธีการแทนที่ (replacement strategy) ซึ่งจะกล่าวในภายหลัง ถ้า OS จะเลือกที่วางหน้า 123 ของโปรแกรมลงในหน้า 78 ของหน่วยความจำจริง และสมมติว่าหน้า 78 นี้เก็บหน้า 001 ของโปรแกรมเดียวกันเอาไว้ เมื่อหน้า 78 ของหน่วยความจำจริงถูกเลือกแล้ว OS จะต้องตรวจสอบก่อนว่าในหน้า 78 นี้มีการแก้ไขข้อมูลหรือเนื้อหาใด ๆ หลังจากที่หน้า 001 ของโปรแกรมถูกโหลดลงมาหรือไม่ ถ้ามี OS ต้องนำเนื้อหาหรือข้อมูลภายในหน้า 78 นี้เขียนกลับคืนไปที่ดิสก์ ณ ตำแหน่งหน้า 001 ของโปรแกรมที่ถูกเก็บไว้ (1495789) ถ้าไม่มีการแก้ไขข้อมูลก็ไม่ต้องเขียนกลับลงไป เพราะการเขียนทำให้เสียเวลา จากนั้น OS จึงค่อยนำหน้า 123 มาวางลงในหน้า 78 ของหน่วยความจำจริง (ดังรูปต่อไป) และทำการแก้ไขข้อมูลในตารางหน้าโดยเปลี่ยนคอลัมน์ที่ 2 ของช่อง 001 เป็น 0 (เพราะตอนนี้ไม่อยู่ในหน่วยความจำแล้ว) แก้คอลัมน์ที่ 2 ของช้อง 123 ให้เป็น 1พร้อมกับใส่แอดเดรสเริ่มต้นของหน้า 78 (78000) ลงไปในคอลัมน์ที่ 3 นำค่าแอดเดรสเริ่มต้นของหน้า 78 ไปบวกกับดีสเพรสเมนต์ (78000 +983) ก็จะได้แอดเดรส 78983 ในหน่วยความจำจริงที่ตรงกับแอดเดรส 123983 ของโปรแกรม

การที่ OS ตรวจสอบจากตารางหน้าแล้วพบว่า หน้าที่ต้องการนั้นไม่ได้อยู่ในหน่วยความจำจริง เรียกว่าเกิดความผิดพลาดของ page ถ้าอัตราส่วนขนาดของหน่วยความจำเสมือนต่อขนาดของหน่วยความจำจริงมีมาก อัตราการเกิดการผิดพลาดของหน้าก็จะมีค่ามากด้วยเช่นกัน การเกิดการผิดพลาดของหน้ามากจะไม่เป็นผลดีต่อการกระทำของระบบ กล่าวคือ การทำงานของโปรแกรมของผู้ใช้ทำงานช้าลง เพราะต้องเสียเวลารอให้ OS ไปโหลดเอาหน้าต่าง ๆ จากดิสก์ลงไปไว้ในหน่วยความจำ

การทำงานของระบบหน้ามีลักษณะเช่นนี้เหมือนกันหมด ต่างกันที่ขนาดของหน่วยความจำจริง หน่วยความจำเสมือนและขนาดของหน้าเท่านั้นสรุปคือ OS จะ

- กำหนดขนาดของหน้าที่แน่นอน
- แบ่งแอดเดรสออกเป็น 2 ส่วนคือ หมายเลขหนา และ ดีสเพรสเมนต์
- สร้างตารางหน้าของแต่ละโปรเซส
- การแปลงแอดเดรสเสมือนเป็นแอดเดรสจริงในหน่วยความจำต้องใช้ตารางหน้าช่วย โดย มีลำดับการทำงานดังนี้

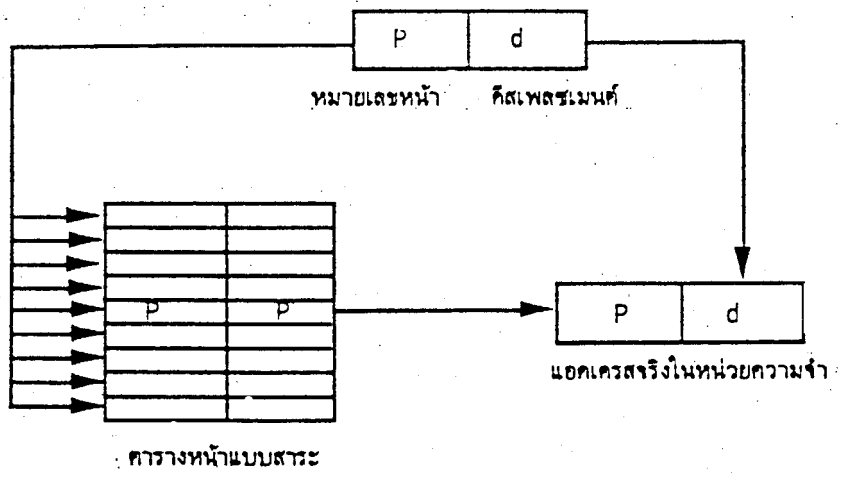
1. ตรวจสอบดูว่าหมายเลขหน้าในแอดเดรสเสมือนอยู่ในหน่วยความจริงหรือไม่
2. ถ้าอยู่ นำค่าแอดเดรสเริ่มต้นของหน้านั้นในหน่วยความจำจริงที่ได้จากตารางหน้ามาบวกกับดีสเพรสเมนต์ ก็จะได้แอดเดรสจริง
3. ถ้าไม่อยู่ ให้ไปโหลดหน้านั้นมาจากดิสก์ ณ ตำแหน่งที่บ่งไปไว้ในตารางหน้า
4. หาหน้าว่างในหน่วยความจำจริง โหลดหน้านั้นลงไป แก้ไขข้อมูลในตารางหน้าและนำค่าแอดเดรสเริ่มต้นในหน่วยความจำของหน้านี้ไปบวกกับ displacement ก็จะได้แอดเดรสจริง
5. ถ้าไม่มีหน้าว่าง ต้องเลือกเอาหน้าหนึ่งออกจากหน่วยความจำ (หน้าใดถูกเลือกขึ้นอยู่กับยุทธวิธีแทนที่ที่ OS ใช้) ถ้าหน้าที่ถูกเลือกนี้มีการแก้ไขเนื้อหาใด ๆ ก็ตามนับตั้งแต่เริ่มเข้ามาอยู่ในหน่วยความจำต้องเขียนหน้านี้กลับไปในดิสก์ ณ ตำแหน่งที่บ่งไว้ในหน้าตารางหน้าของโปรเซสที่เป็นเจ้าของหน้านี้ จากนั้นจึงค่อยโหลดหน้าใหม่ทับลงไป แก้ไขข้อมูลในตาราง และนำแอดเดรสเริ่มต้นของหน้าในหน่วยความจำจริงบวกกับดีสเพรสเมนต์ ก็จะได้แอดเดรสในหน่วยความจำจริง

การแปลงส่งแบบสาระ (associative mapping)

วิธีการแปลงส่งแอดเดรสที่กล่าวมาแล้ว เรียกว่าเป็นการแปลงส่งแบบตรง (direct mapping) ยังมีวิธีการแปลงแอดเดรสอีกวิธีหนึ่งซึ่งมีความเร็วสูงมาก เรียกว่าเป็นการแปลงแบบส่งสัมภาระ (associative mapping) ดังรูปต่อไป วิธีการแปลงเป็นดังนี้ หมายเลขหน้าจากแอดเดรสเสมือนจะถูกส่งเข้า

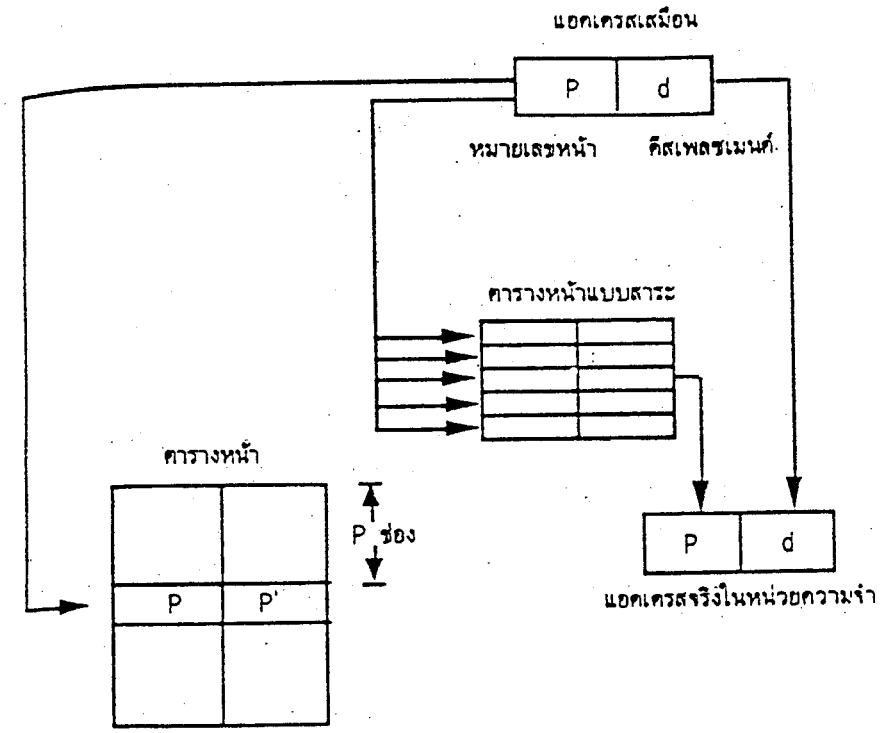
ไปตรวจสอบในตารางที่เรียกว่าตารางหน้าแบบสัมภาระ (associative gage table) พร้อมกันทุกช่อง และจะได้ค่าแอดเดรสเริ่มต้นของหน้าในหน่วยความจำจริงที่เก็บหน้านี้ออกมาไว้ทันที

แอดเดรสที่ได้นี้ก็จะนำไปบวกกับดีสเพรสเมนต์ผลลัพธ์ที่ได้คือแอดเดรสจริงในหน่วยความจำ แต่ถ้าเกิดการผิดพราดของหน้า OS ก็ต้องทำวิธีเดียวกับวิธีการแปลงส่งแบบตรงคือ โหลดหน้านั้นมาจากดิสก์ และหาหน้าที่ว่างในหน่วยความจำเพื่อวางหน้าใหม่ลงไป ถ้าไม่มีหน้าว่างก็โหลดหน้าใดหน้าหนึ่งในหน่วยความจำจริง แล้วจึงค่อยทำการแปลงส่งแอดเดรส



รูป การแปลงส่งแบบสาระ

ตารางหน้าแบบมีสาระจะมีจำนวนช่องเท่ากับตารางหน้าธรรมดา แต่อาศัยวงจรการสร้างทางฮาร์ดแวร์ที่พิเศษ ทำให้สามารถค้นหาในหน่วยความจริงได้พร้อมกันทีเดียวทั้งตาราง จึงเร็วกว่าการใช้ตารางหน้าธรรมดาหลายเท่า แต่จะมีราคาแพงมาก จึงนิยมสร้างให้มีขนาดของตามตารางหน้าและใช้งานร่วมกับตารางหน้าแบบธรรมดาด้วยการแปลงส่งแอดเดรส วิธีนี้เรียกว่าแบบผสมระหว่างแบบสาระและแบบตรง (combined associative / direct mapping) หรือเรียกสั้นๆ ว่าแบบผสม ดังแสดงใน



รูป การแปลงแบบผสม

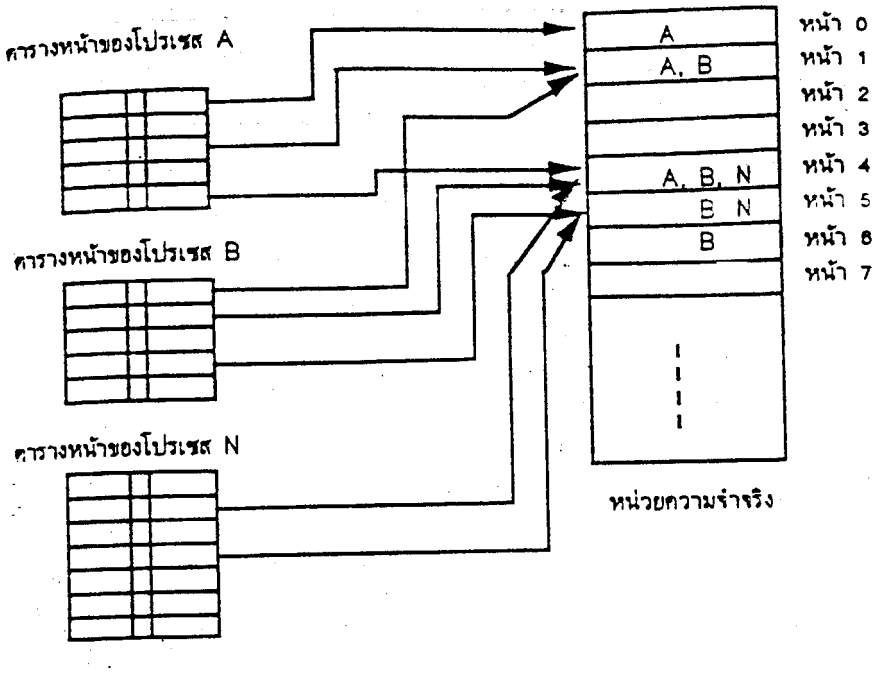
หลักการทำงานของการแปลงแบบผสมเป็นดังนี้ หมายเลขหน้าในแอดเดรสเสมือน (P) จะถูกนำไปตรวจสอบตารางหน้าแบบสาระก่อนโดยตรวจสอบทางคอลัมน์ซ้ายในตาราง ถ้าพบหมายเลขหน้า (หรือแอดเดรสเริ่มต้นของหน้า) ในหน่วยความจำจริง (P) แต่ถ้าไม่พบ OS จะนำหมายเลขหน้า P ไปใช้กับในตารางหน้าแบบผสมจะใช้การแปลงส่งแบบสาระก่อน (เพราะทำงานได้เร็วกว่าหลายเท่า) ถ้าแปลงไม่ได้เนื่องจากหมายเลขหน้าที่ต้องการไม่ได้ถูกเก็บอยู่ในหน้าทดลองแบบสาระ OS ก็จะใช้วิธีแปลงส่งแบบตรง

ระบบการใช้หน้าร่วม (sharing page system)

ในการทำงานระบบผู้ใช้หลายคน ณ ขณะหนึ่งผู้ใช้แต่ละคนอาจเรียกใช้โปรแกรมเดียวกันพร้อมกันก็ได้ เช่น นาย ก , ข และ ค เรียกใช้โปรแกรมเอดิเตอร์ (editor) โปรแกรมเดียวกันพร้อมกันทั้งสามคน ในกรณีเช่นนี้ OS จะต้องนำโค้ดโปรแกรมเอดิเตอร์ตัวเดียวกันนี้ โหลดเข้าไปในหน่วยความจำถึง 3 ครั้ง โปร แกรมเอดิเตอร์จะปรากฏอยู่ในหน่วยความจำถึง 3 แห่ง แต่ผู้ใช้สามารถใช้โค้ดโปรแกรมร่วมกันได้ เราก็สามารถประหยัดเนื้อที่ของหน่วยความจำได้มาก เพราะโค้ดของโปรแกรมเอดิเคเตอร์จะปรากฏอยู่ในหน่วยความจำเพียงแห่งเดียว

ในปัจจุบัน การเขียนโปรแกรมนิยมให้ส่วนโค้ดคำสั่งและข้อมูลแยกออกจากกัน เมื่อโปรแกรมทำงานส่วนที่เป็นข้อมูลจะมีการเปลี่ยนแปลงแก้ไขตลอดเวลา แต่ส่วนที่เป็นโค้ดคำสั่งล้วนๆ โดยมากแล้วจะไม่มีการแก้ไข โค้ดคำสั่งซึ่งไม่สามารถเปลี่ยนแปลงและแก้ไขได้เรียกว่า รีเอ็นแทรนโค้ด (reentrant code) ในระบบหลายผู้ใช้ ถ้ามีโปรแกรมที่ถูกใช้หลาย ๆคน ส่วนที่เป็นรีเอ็นแทรนนี้จะปรากฏได้ในหน่วยความจำเพียงแห่งเดียว และให้ผู้ใช้ทุกคนสามารถใช้โค้ดส่วนนี้ร่วมกันได้ ทำให้ประหยัดเนื้อที่ในหน่วยความจำ แต่ ผู้ใช้แต่ละคนจะมีส่วนที่เป็นข้อมูลของตนเอง เพราะข้อมูลของแต่ละคนของผู้ใช้ไม่สามารถใช้ร่วมกันได้

จากหลักการนี้ ถ้าระบบมีการใช้หน้าร่วม OS จะทำการตรวจสอบหน้าแต่ละหน้าในหน่วยความจำว่า หน้าใดสามารถแบ่งให้ผู้ใช้คนอื่นสามารถใช้ร่วมกันได้ (หน้านั้นเป็นรีเอ็นแทรนโค้ด) หรือไม่ ถ้าเป็นไม่จำเป็นต้องโหลดหน้านั้นลงไปอีก และแน่นอนหน้าใดที่เก็บข้อมูลของผู้ใช้ไม่สามารถให้ผู้อื่นใช้คนอื่นมาร่วมใช้ได้ ดังรูปต่อไปแสดงการทำงานระบบการใช้หน้าร่วม



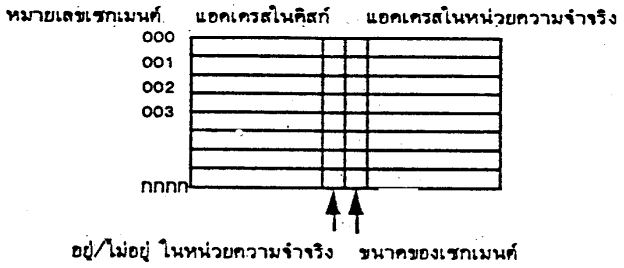
รูป การใช้งานหน้าในระบบหน้าร่วม

หน่วยความจำเสมือนระบบเซกเมนต์ (segmentation system)

หน่วยความจำระบบเซกเมนต์ มีลักษณะการทำงานคล้ายระบบหน้ามากต่างกันตรงที่ขนาดของบล็อกไม่จำเป็นต้องเท่ากัน จึงทำให้มีความซับซ้อนการทำงานเพิ่มมากขึ้น ตารางเซกเมนต์ (segment table) ซึ่งทำหน้าที่เช่นเดียวกับตารางหน้า จะคอลัมน์เพิ่มขึ้นจากตารางหน้าอีกหนึ่งคอลัมน์ เป็นคอลัมน์ที่เก็บขนาดของเซกเมนต์เอาไว้ ทั้ง

นี้เพื่อทำให้ OS ทราบว่าแต่ละเซกเมนต์มีขนานเท่าไร การอ่านหรือเขียนข้อมูลจากหน่วยความจำสำรอง การหาเนื้อที่ในหน่วยความความจำจริงจะกระทำตามขนาดเซกเมนต์

ระบบหน้า	ระบบเซกเมนต์
ตารางหน้า หมายเลขหน้า (page number) ดีสเพลซเมนต์ ตารางหน้าแบบสาระ ระบบหน้าร่วม	ตารางเซกเมนต์ หมายเลขเซกเมนต์ (segment number) ดีสเพลซเมนต์ ตารางเซกเมนต์แบบสาระ ระบบเซกเมนต์ร่วม



รูป ตารางเซกเมนต์

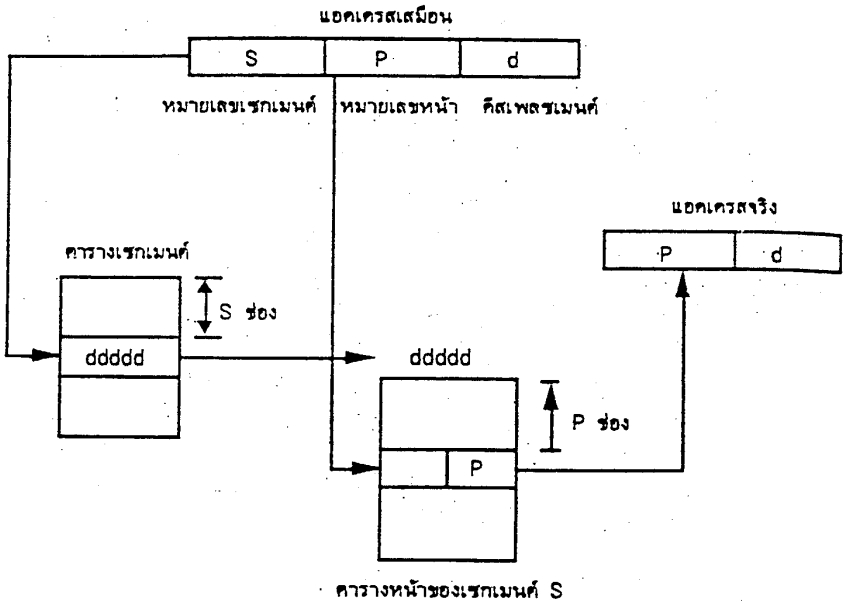
หน่วยความจำเสมือนระบบผสมหน้าและเซกเมนต์

วิธีการทำหน่วยความจำเสมือนแบบผสม (combing paging/segmentation system)ได้รวมเอาลักษณะการทำงานของระบบหน้าและระบบเซกเมนต์เาด้วยกัน กล่าวคือระบบจะแบ่งหน่วยความจำออกเป็นหน้าที่มีนานเท่ากัน ในโปรแกรมของผู้ใช้ จะถูกแบ่งออกเป็นเซกเมนต์ภายใน เซกเมนต์จะถูกแบ่งออกเป็นหลายๆหน้า ดังนั้นขนานของเซกเมนต์จะเป็นจำนวนเท่าของหน้า แต่ละเซกเมนต์ของโปรแกรมไม่จำเป็นต้องอยู่เรียงกันในหน่วยความจำ และแต่ละหน้าภายในเซกเมนต์เดียวกันก็ไม่จำเป็นต้องอยู่เรียงติดกันในหน่วยความจำจริง การผสมเอาระบบหน้าและเซกเมนต์เข้าด้วยกันทำให้ประสิทธิภาพการทำงานของระบบดีขึ้น

ระบบผสมนี้ แอดเดรสเสมือนจะกแบ่งเป็น3ส่วนคือ หมายเลขเวกเมนต์ หมายเลขหน้าและดีสเพรสเมนต์ ดังรูป

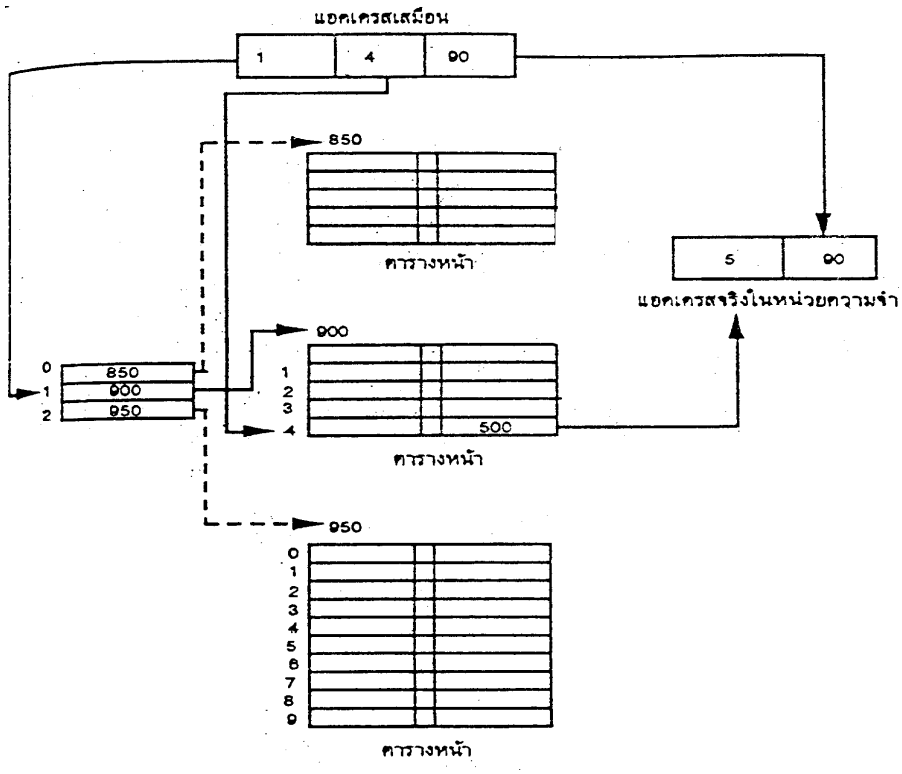
วิธีการแปลงแอดเดรสจะซับซ้อนขึ้นอีกระบบหนึ่งจากระบบหน้า โดยแต่ละโปรเซสจะมีตารางเซกเมนต์อยู่ 1 ตาราง แต่ละเซกเมนต์จะมีตารางหน้าของมันเอง ดังนั้นโปรเซสจึงมีตารางหน้าเท่ากับจำนวน

เซกเมนต์ของโปรเซสนั้น จำนวนช่องของตารางหน้าขึ้นอยู่กับจำนวนหน้าของเซกเมนต์นั้น ค่าที่เก็บอยู่ในเซกเมนต์ คือ แอดเดรสเริ่มต้นของตารางหน้าของเซกเมนต์นั้น การทำแอดเดรสทำดังรูปต่อไป นำหมายเลขเซกเมนต์ (S) ของแอดเดรสเสมือนไปหาแอดเดรสของตารางหน้าที่เก็บในเซกเมนต์ เมื่อทราบว่าตารางหน้าอยู่ที่ใดในหน่วยความจำแล้วใช้หมายเลขหน้า P เพื่อหาค่าแอดเดรสของหน้า P'ในหน่วยความจำ นำค่าแอดเดรสนี้บวกกับค่าดีสเพรสเมนต์ ก็จะได้แอดเดรสจริงในหน่วยความจำ



รูป ตารางหน้าของเซกเมนต์ S

สมมติว่าโปรเวสหนึ่งมี 3 เซกเมนต์ (2-3) เซกเมนต์ 0 และ 1 มี 5 หน้า และเซกเมนต์ 2 มี 10 หน้าแต่ละหน้ามีขนาด 100 ไบต์ (00-99) ดังนั้นดีสเพรสเมนต์เป็นเลข 2 หลัก รูปด้านบนแสดงตารางเซกเมนต์และตารางหน้าของเซกเมนต์นี้ สมมติว่าต้องการแปลงแอดเดรสเสมือน 1490 เป็นแอดเดรสจริง หมายเลขเซกเมนต์คือ 1 หมายเลขหน้าคือ 4 ดีสเพรสเมนต์คือ 90 ขั้นแรกอ่านค่าเก็บไว้ในช่องที่ 1 ของตารางเซกเมนต์ (ดูรูปถัดไป) ได้ค่า 900 หมายความว่าตารางหน้าของเซกเมนต์ 1 อยู่ที่แอดเดรส 900 ต่อมาตรวจสอบค่าในช่องที่ 4 ของตารางหน้าที่แอดเดรส 900 เพื่อดูว่าหน้าที่ 4 ของเซกเมนต์ 1 เก็บอยู่ที่หน้าใดในหน่วยความจำเมื่อได้ค่าแอดเดรสของหน้าในหน่วยความจำแล้ว (500) ก็นำไปบวกกับดีสเพรสเมนต์ ก็จะได้แอดเดรสจริงในหน่วยความจำคือ 590



รูป

ยุทธวิธีแทนที่

เมื่อเกิดผิดพลาดของหน้าในระบบหน้า และในหน่วยความจำไม่มีหน้าใดว่างอยู่เลยก่อนที่ OS จะโหลดเอาหน้าใหม่เข้ามาในหน่วยความจำนั้น OS ต้องตัดสินใจก่อนว่าควรเลือกหน้าใดเพื่อที่จะวางหน้าใหม่ทับลงไป สิ่งที่ OS ในการตัดสินใจเลือกหน้าเรียกว่า ยุทธวิธีแทนที่ (replacement strategy) ซึ่งมีอยู่ 5 วิธีด้วยกันคือ

1. แบบสุ่ม (random) เลือกโดยการสุ่มเดา ทุกหน้ามีโอกาศถูกเลือกเท่ากันหมด

2. FIFO (first in first out) หน้าใดถูกโหลดเข้ามาในหน่วยความจำก่อนก็จะถูกเลือกออกไป ก่อนเรียงตามลำดับเวลา

3. NFU (not frequently used) เลือกหน้าที่ถูกใช้น้อยที่สุด ทั้งนี้เพราะหน้าที่ถูกใช้น้อยโอกาศที่จะถูกใช้ในเวลาต่อมาก็จะน้อยด้วย ดังนั้นจึงควรเอาออกจากหน่วยความจำ แต่วิธีนี้อาจทำให้หน้าที่ถูกโหลดเข้ามาถูกเลือกออกไปได้ เพราะหน้าที่พึ่งถูกโหลดเข้ามาใหม่ จำนวนการใช้ย่อมน้อยกว่าหน้าที่มีอยู่ก่อนแล้ว

4. LRU (least recently used) แต่ละหน้าจะมีการบันทึกเวลาในการใช้ครั้งหลังสุดไว้ หน้าใดที่ไม่ได้ใช้มานานที่สุดก็จะถูกเลือก วิธีนี้จะผลเสียต่อโปรแกรมที่มีการทำงานแบวนรอบ (loop) และวงรอบมีขนาดใหญ่มาก ๆ (หลาย ๆ หน้า)

5. NUR (not used recently) แต่ละหน้าจะมีบิตกำกับอยู่ 2 บิต คือบิตอ้างอิง (referent bit)และบิตแก้ไข (modify bit) เมื่อหน้าถูกโหลดเข้าไปในหน่วยความจำ 2 บิต นี้จะเป็น 0 เมื่อใดที่มีการอ้างถึงหน้าใดบิตอ้างอิงของหน้านั้นจะเป็น 1 และเมื่อนั้นถูกแก้ไขอะไรบางอย่างภายในหน้าบิตแก้ไขจะเป็น 1

ทุก ๆ ช่วงเวลาหนึ่ง เช่น 10 มิลลิวินาที บิตอ้างอิงของทุก ๆ หน้าจะถูกเปลี่ยนเป็น 0 หมด ดั้งนั้นจาก 2 บิต

นี้ เราจะแบ่งประเภทของหน้าออกเป็น 4 ประเภทคือ

1. ไม่มีการอ้างถึง ไม่มีการแก้ไข
2. ไม่มีการอ้างถึง มีการแก้ไข
3. มีการอ้างถึง ไม่มีการแก้ไข
4. มีการอ้างถึง มีการแก้ไข

(**หมายเหตุ** จะเห็นว่าการแก้ไขหน้าโดยไม่มีการอ้างถึงนั้นเป็นไปไม่ได้ แต่ที่เกิดหน้าประเภทที่ 2 ได้นั้น เป็นเพราะมีการเปลี่ยนบิตอ้างอิงทุกๆระยะเวลาหนึ่งให้เป็น 0) การเลือกหน้าจะเลือกตามประเภทของหน้า คือ เลือกหน้าในประเภทที่ 1 ก่อน ถ้าไม่มีเลือกหน้าในประเภทที่ 2 , 3และ 4 ตามลำดับ

การเลือกหน้าในระบบหน้าที่ใช้ยุทธวิธีแทนที่เพียงอย่างเดียว แต่สำหรับในระบบเซกเมนต์ต้องคำนึงถึงยุทธวิธี การวางด้วย (best fit , first fit , worst fit) เพื่อให้เกิดว่างในหน่วยความจำเหมาะสมที่สุดทั้งนี้เพราะแต่ละเซกเมนต์ไม่เท่ากัน

ยุทธวิธีการเฟตช์

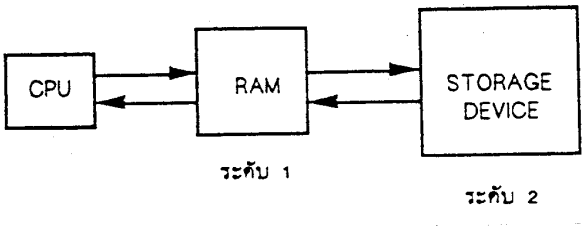
ยุทธวิธีการเฟตช์ (fetch strategy) หมายถึง การโหลดหน้าหรือเซกเมนต์จากดิสก์เข้าไปในหน่วยความจำ แบ่งออกได้ 2 วิธี คือ

1. การเฟตช์แบบต้องการ (demand fetch) วิธีนี้ OS จะโหลดเฉพาะหน้าหรือเซกเมนต์ที่ต้องการใช้เท่านั้นเข้าไปในหน่วยความจำ

2. การเฟตช์แบบคาดเดา (anticipate fetch) OS จะมีการคาดเดาว่าหน้าหรือเซกเมนต์ไหนจะถูกใช้เป็นหน้าหรือเซกเมนต์ต่อไป และจะโหลดหน้าหรือเซกเมนต์นั้นเข้าไปไว้ในหน่วยความจำล่วงหน้า (ก่อนเกิดการถูกใช้จริง) ซึ่งทำให้โปรแกรมทำงานได้เร็วขึ้น (ไม่เสียเวลารอขณะเกิดความผิดพลาดของหน้า) แต่ระบบต้องมีการทำงานเพิ่มขึ้นและบางครั้งเกิดการคาดเดาที่ผิดพลาดด้วย ทำให้หน้าหรือเซกเมนต์ที่ถูกโหลดเข้าไปล่วงหน้าไม่ได้ถูกใช้งาน

ลำดับขั้นของหน่วยความจำ

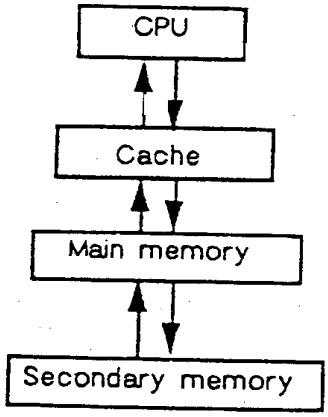
ระบบหน่วยความจำเสมือน สามารถทำให้ผู้ใช้ใช้หน่วยความจำขนาดใหญ่กว่าหน่วยความจำจริงได้ ก็เพราะอาศัยการเก็บข้อมูล (หรือโปรแกรม) ไว้ในหน่วยความจำสำรอง ลักษณะการเคลื่อนย้ายข้อมูลจะมีการส่งไปมาระหว่างหน่วยความจำสำรอง กับหน่วยความจำหลัก ดังรูป ลักษณะเช่นนี้เรียกว่า หน่วยความจำ 2 ระดับ หมายถึงว่า ข้อมูลมีการขนย้ายจากหน่วยความจำประเภทหนึ่งไปยังหน่วยความจำอีกประเภทหนึ่ง



รูป หน่วยความจำ 2 ระดับ

เราอาจมีหน่วยความจำได้หลายระดับ หน่วยความจำระดับต่ำจะมีคุณสมบัติที่ตรงกันข้ามกับหน่วยความจำที่อยู่ในระดับสูงกว่า คือหน่วยความจำในระดับที่ต่ำลงจะมีราคาที่แพงขึ้น ความเร็วในการเข้าถึงสูงขึ้น ความจุต่ำลง แต่ละหน่วยความจำในระดับสูงกว่าจะมีราคาถูกกว่า ความเร็วในการเข้าถึงต่ำลง ความจุสูงขึ้น

ในทศวรรษที่ 1960 มีการพัฒนาหน่วยความจำให้เป็นลำดับชั้นมากกว่า 1 หรือ 2 ระดับ หน่วยความจำแคช (cache memory) จึงได้ถูกพัฒนาขึ้นมาใช้งาน แคชมีความเร็วกว่าหน่วยความจำแรมหลายเท่า ขณะเดียวกันก็มีราคาแพงกว่ามาก จึงมีการนำแคชขนากเล็กมาใช้เท่านั้น แคชจะอยู่ระหว่าง CPUกับหน่วยความจำหลัก ดังรูป เมื่อ CPU ต้องการข้อมูล CPU กำหนดตำแหน่งของข้อมูลที่ต้องการด้วยแอดเดรส ค่าแอดเดรสนี้จะถูกนำไปตรวจสอบในแคชด้วยความเร็วสูง ถ้าในแคชมีข้อมูลในตำแหน่งที่บ่ง โดยแอดเดรส CPU ก็จะได้รับข้อมูลในแอดเดรสจากแคชทันที แต่ถ้าไม่มีข้อมูลนั้นจะถูกส่งมาจากหน่วยความจำหลักมาเก็บลงในแคชก่อนแล้วค่อยส่งไปให้ CPU การขนย้ายข้อมูลระหว่างแคชกับแรม เป็นไปด้วยวงจรทางฮาร์ดแวร์ ไม่มีส่วนเกี่ยวข้องกับโปรแกรมใดๆ ทั้งสิ้น (รวมทั้งตัว OS เองด้วย) CPU บางตัวถูกสร้างให้มีแคชติดมาด้วย ทำให้การทำงานของ CPU มีความเร็วสูงขึ้น



รูป ระบบหน่วยความจำ 3 ระดับ โดยใช้ Cache

Pipeline

Pipelining เป็นเทคนิคในการทำงาน โดยที่คำสั่งถูกปฏิบัติไปพร้อม ๆ กันหลาย ๆ คำสั่ง Pipelining เป็นเทคนิคที่ใช้เพื่อให้ CPU ทำงานได้เร็วขึ้น

Pipeline จะมีลักษณะเหมือนสายพานการผลิต ซึ่งมีหลายขั้นตอน จะทำงานไปพร้อม ๆ กัน ในคอมพิวเตอร์ ขั้นตอนในการทำงานแบบ Pipeline จะทำตามคำสั่งให้เสร็จสมบูรณ์ในขั้นตอนของมันเองเท่านั้น ขั้นตอนการทำงานที่แตกต่างกัน จะทำงานในลักษณะที่แตกต่างกันไป ขั้นตอนแต่ละขั้นตอนเรียกว่า “Pipe Stage”หรือ “Pipe Seqment” แต่ละขั้นตอนจะทำงานต่อเนื่องกันไปเป็นทอด ๆ

ในการทำงานแบบสานพาน throughout ถูกให้ความหมายเหมือนกับเป็นจำนวนของรถที่ผลิตต่อ 1 ชั่วโมง โดยดูว่ารถจะถูกผลิตออกมาบ่อยแคไหน

แต่การทำงานแบบ Pipeline throughout ถูกให้ความหมายโดยดูที่คำสั่งเกิดขึ้นใน pipeline ว่าเกิดขึ้นบ่อยแค่ไหน เพราะว่า Pipe Stage จะเชื่อมโยงต่อกัน ดังนั้น แต่ละขั้นตอนต้องพร้อมที่จะทำงานไปพร้อม ๆ กัน เวลาที่ถูกใช้ในระหว่างการเคลื่อนที่ของคำสั่งจากขั้นตอนหนึ่งไปยังอีกขั้นตอนหนึ่งเราเรียกว่า “Machine Cycle” เพราะว่าทุก ๆ ขั้นตอนทำงานพร้อม ๆ กัน ดังนั้น ความยาวของ “Machine Cycle” จะดูจากเวลาที่ต้องใช้ใน Pipe Stage ที่ทำงานช้าที่สุด

เป้าหมายของผู้ที่ออกแบบ Pipeline คือต้องการให้ความยาวของ Pipeline แต่ละขั้นตอนเกิดความสมดุล ถ้าขั้นตอนแต่ละขั้นตอนสมดุลกันแล้วเวลาที่ใช้ต่อ 1 คำสั่งใน Pipeline จะเท่ากับ

Time per instruction on unpipelined machine

Number of pipe stages

การใช้ Pipeline จะส่งผลทำให้เวลาการทำงานต่อคำสั่ง 1 คำสั่งลดลง โดยจะขึ้นอยู่กับเกณฑ์ที่เราใช้ด้วย การลดลงอาจมองได้ว่าเป็นการลดจำนวนของ Cycle ต่อคำสั่ง(CPI) ถ้าจุดเริ่มต้นเป็นเครื่องกลที่ใช้ clock cycle หลาย ๆ ตัวการใช้Pipeline จะดูเหมือนเป็นการลด CPI แต่ถ้าจุดเริ่มต้นเป็นเครื่องกลที่ใช้ clock cycle 1ตัวต่อ 1 คำสั่ง การใช้ Pipeline จะดูเหมือนเป็นการลดเวลาในการทำงานของตัว clock cycle

ในบทเรียนนี้เราจะศึกษาเกี่ยวกับ Pipeline แบบที่ใช้ DLX และแบบที่เป็นแบบปกติเราใช้ DLX เพราะว่าการใช้งานไม่ซ้ำซ้อนและง่ายในการนำเสนอสิ่งสำคัญต่าง ๆ ของ Pipeline และเพื่อที่จะให้เข้าใจง่ายขึ้น ในบทนี้เราจะรวมการกระโดดข้ามคำสั่งของ DLX หัวใจสำคัญในการใช้ Pipeline ในบทนี้คือ การปรยุกต์ใช้คำสั่งที่สมบรูณ์แบบ

A Simple Implementation of DLX

(การปฏิบัติงานขั้นพื้นฐานของ DLX)

ในส่วนนี้จะแสดงการทำงานขั้นพื้นฐานในแบบที่ใช้ clock cycle 5 ตัว เราจะใช้การทำงานขั้นพื้นฐานนี้เพื่อลด CPI unpipeline เป็นการทำงานที่ไม่ใช่ว่าจะมีประสิทธิภาพสูงสุด คำสั่ง DLX ทำงานได้ในแบบที่มี clock cycle 5 ตัว ซึ่ง clock cycle ทั้ง 5 ตัว ก็คือ

1. Instruction Fetch cycle (IF)

IR --------- Mem [PC]

NPC ------ PC + 4

การทำงาน ส่ง PC และรับคำสั่งจาก Memory ไปยัง Struction register (IR) เราจะเพิ่ม PC โดยการบวกด้วย 4 IR จะถูกใช้เพื่อรับคำสั่งที่จำเป็น

2. Instruction Decode /register fetch cycle (ID)

A------ Regs [IR 6……10] ;

B ------Regs [IR 11…..15] ;

Imm ----- ((IR 16) 16 # # IR 16…….31)

การทำงาน การถอดคำสั่งและเข้าถึง register file ผลของ register ถูกอ่านในแบบ general register ซึ่งมันเป็นไปได้เพราะว่ามันอยู่ในบริเวณที่กำหนดไว้ในคำสั่ง DLX เทคนิคแบบนี้ เรียกว่า fixed

field decoding เพราะว่าบางส่วนของคำสั่งตั้งอยู่ในบริเวณที่กำหนด ดังนั้น Sign-Extended จะถูกคำนวณในขั้นตอนนี้ด้วย

3. Execution / Collective address

ALU จะทำงานบนจำนวนที่ได้ทำการคำนวณเตรียมเอาไว้แล้วที่เป็น 1 ใน 4 หน่วยงานขึ้นอยู่กับชนิดของโครงสร้างของ DLX

- **Memory Reference**

  ALU Output ------ A + Imm ;

  การทำงาน ALU จะถูกเพิ่ม operands เข้าไปเพื่อผลการทำงานที่มีประสิทธิภาพมากขึ้น ผลของการทำงานจะแสดงผลใน register ALU Output

- **Register - Register ALU instruction ;**

  ALU Output ------ A func B

  การทำงาน ALU จะปฏิบัติงานตาม function code ที่อยู่ใน register A และ register B ผลที่จะได้จะแสดงใน temporary register ALU

- **Register – Immediate ALU instruction**

  ALU Output ------A op Imm ;

  การทำงาน ALU จะปฏิบัติตาม opcode ที่อยู่บน register A และ register Imm ผลที่จะได้จะแสดงใน temporary register ALU

- **Branch**

  ALU ----------NPC + Imm ;

  Cond ----------(A op O)

  การทำงาน ALU จะเพิ่ม NPC ไปใน sign-extended immediate value ที่อยู่ใน Imm เพื่อใช้คำนวณตามเป้าหมายของ branch ที่วางไว้ register A ที่ถูกอ่านใน cycle อันก่อนๆจะถูกตรวจสอบเพื่อจะได้รู้ว่า branch ได้ถูกเอามาใช้ The comparison operation (OP) เป็นความสัมพันธ์ที่เกิดขึ้นจาก branch opcode ตัวอย่างของ op เป็น = = สำหรับคำสั่ง BEQZ

4. Memory access / branch completion cycle (MEM)

เฉพาะคำสั่ง DLX เท่านั้นที่จะทำงาน cycle นี้ ก็มี load ,stores และ branches

- **Memory reference**

  IMD ---- Mem [ALU Output] or

Mem -----[ALU Output] ----- B ;

การทำงาน การเข้าถึงหน่วยความจำ เป็นสิ่งที่จำเป็น ถ้าคำสั่งถูก load ขึ้นมาข้อมูลจะย้อนกลับมาจากหน่วยความจำ และจะถูกกำหนดใน LMD (load memory data) register ถ้ามันเป็นแบบ store ต่อจากนั้นข้อมูลจาก B จะถูกเขียนลงในหน่วยความจำ address จะถูกคำนวณในระหว่างขั้นตอนก่อนหน้านี้ และจะถูกเก็บใน register ALU output

- **Branch**

If (cond) PC ----AUL output else PC ------NPC ;

การทำงาน ถ้าเป็นคำสั่ง branch PC จะถูกแทนที่ด้วย branch destination ที่อยู่ใน register AUL output หรืออีกนัยหนึ่งถูกแทนที่ด้วย PC ที่เพิ่มขึ้นใน register NPC

5. Write-Back cycle (WB)

- **Register – Register ALU instruction**

Regs [IR16…..20] …. ALU Output ;

- **Register - Immediate ALU instruction :**

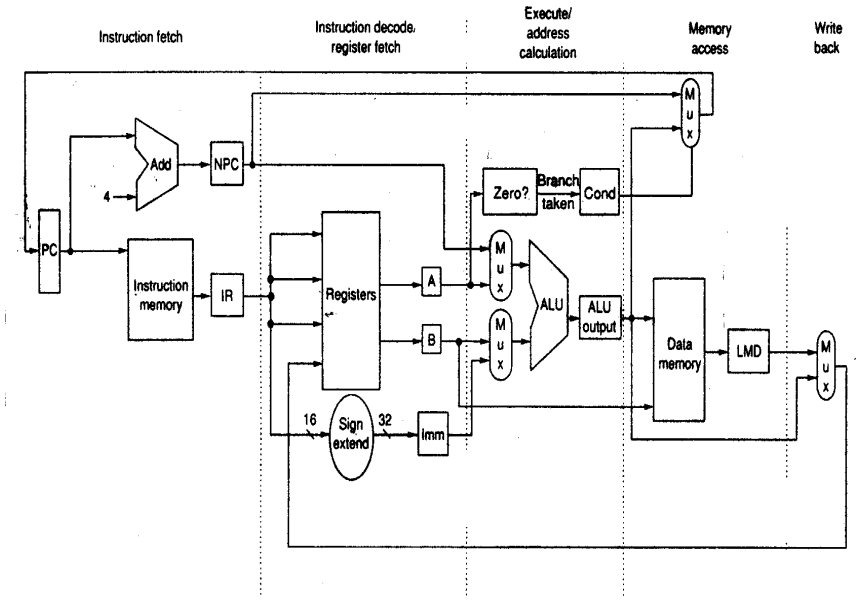
Reps [IB 11….15] ------AUL output;

- **Load instruction**

Rega [IR 11…15] --------LMp;

การทำงาน จะทำการเขียนลงในแฟ้มข้อมูล ซึ่งมาจาก memory system (ระบบของหน่วยความจำ) ที่อยู่ใน LMP หรือมาจาก ALU ที่อยู่ใน ALU output ก็ อยู่ 1 ใน 2 ตัวนี้ด้วย โดยขึ้นอยู่กับคำสั่งตามหน้าที่

รูปจะแสดงว่าการเคลื่อนที่ของคำสั่งบน datapath ที่บริเวณสิ้นสุดของ colock cycle แต่ละตัว ค่าทุกๆๆค่าจะถูกคำนวณ ระหว่างclock cycle ตัวหน้าไปยัง clock cycle a genral – purpose register , the PC , or a temporary register (ตัวอย่างเช่น LMD , ,Imm , A, B ,IR , NPC ,ALU output , or cond) The temporary register จะควบคุมค่าระหว่าง clock cycle สำหรับคำสั่ง 1 คำสั่ง ในขณะที่เครื่องมือในการเก็บข้อมูลตัวอื่นๆ สามารถมองเห็นไเ และ สามรถมองเห็นได้ และควบคุม Value ระหว่าง Successive instructions



คำสั่ง branch และ store จำเป็นต้องมี 4 cycle และคำสั่งอื่นๆ จำเป็นต้องมี 6 cycle สมมุติว่า ความถี่ของ branch = 12% และ store = 5 %

CPI สามรถถูกปรับปรุงโดยไม่มีผลกระทบต่อ Clock rate โดยปฏิบัติตามคำสั่ง AUL ให้เสร็จเรียบร้อยในระหว่าง MEM cycle สมมุติว่าคำว่าคำสั่ง AUL = 47% ของคำสั่งรวมเหมือนกับที่เราคำนวณใน chapter 2 การปรับปรุงนี้จะนำไปสู่ CPI = 4.35 หรือ 4.82 /4.35 = 1.1 การเปลี่ยนแปลงแบบธรรมดาหรือคำสั่งอื่น ๆ จะพยายามลด CPI เราอาจจะเพิ่ม clock cycle การเปลี่ยนแปลง คือ การต้องการให้มีการทำงานมาก ๆ ใน clock cycle ใช่!! มันอาจจะเป็นผลประโยชน์ในการซื้อขาย clock cycle ที่ดีมากขึ้น แต่ถ้าวิเคราะห์รายละเอียดแล้ว ไม่น่าจะเป็นไปได้ที่จะเกิดการปรับปรุงใหม่ โดยเฉพาะถ้าแบ่งงานเบื้องต้นระหว่าง clock cycle ต้องสมดุลกัน

The Basic Pipeline for DLX

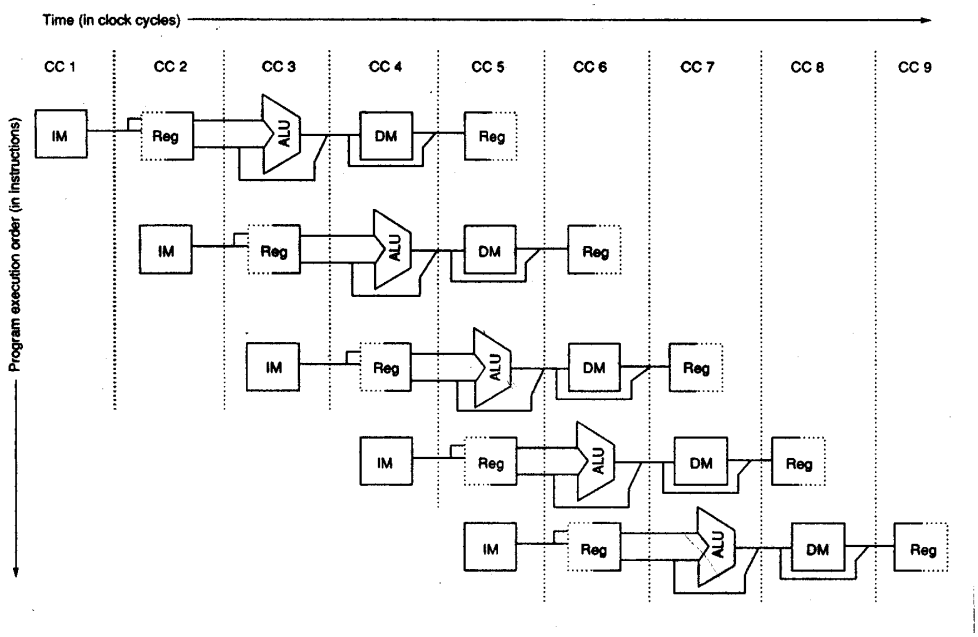
เราสามารถส่งคำสั่งด้วยการที่ไม่เปลี่ยนแปลง โดยจะเริ่มคำสั่งใหม่ในแต่ละ clock cycle แต่ละตัวของ colck cycle จากช่วงแรกจะกลายเป็น pipe stage ผลของการทำงานในรูปแบบนี้ จะถูกแสดงในรูป ในขณะที่แต่ละคำสั่งจะต้องใช้ clock cycle 5 ตัว เพื่อที่จะปฏิบัติให้เสร็จเรียบร้อย ระหว่าง clock cycle แต่ละตัวก็จะมีการเริ่มคำสั่งใหม่ขึ้นมาและทำงานไปตามคำสั่งนั้น ๆ

	Clock number								
Instruction number	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Instruction *i*	IF	ID	EX	MEM	WB				
Instruction *i* + 1		IF	ID	EX	MEM	WB			
Instruction *i* + 2			IF	ID	EX	MEM	WB		
Instruction *i* + 3				IF	ID	EX	MEM	WB	
Instruction *i* + 4					IF	ID	EX	MEM	WB

มันยากที่จะเชื่อว่า การส่งของคำสั่งไปยังส่วนต่าง ๆ จะง่ายแบบนี้เพราะจริง ๆ แล้วมันไม่ใช่ ในส่วนที่กำลังศึกษาอยู่นี้ กับส่วนที่เราจะศึกษาต่อไป เราจะทำให้ DLX pipeline ของเราเป็นจริงขึ้นมาโดยนำไปเกี่ยวข้องกับปัญหาต่าง ๆ ที่เกิดจากการส่งคำสั่ง

การที่เราจะเริ่มศึกษาเราต้องดูว่าอะไรเกิดขึ้น บน clock cycle ทุกตัวของเครื่องกลและย้ำให้แน่ใจว่าเราไม่ได้พยายามปฏิบัติขั้นตอนที่แตกต่างกันด้วยข้อมูลตัวเดียวบน clock cycle ที่เหมือนกัน ยกตัวอย่าง เช่น Sigle ALU ไม่สามารถใช้คำนวณ และ ทำการลบขั้นตอนออกได้ในเวลาเดียวกันเราต้องแน่ใจว่าการทำงานที่ซ้อน ๆ กันใน pipeline นั้น ไม่ขัดแย้งกันคำสั่ง DLX ทำให้ความตีความง่ายขึ้นจะเป็นการแสดงข้อมูล DLX ในรูปแบบของ pipeline หน่วยใหญ่ของการทำงานถูกใช้ในวงจรที่แตกต่างกัน คำสั่งหลาย ๆ คำสั่ง ที่ทำงานซ้อนกัน ก่อให้เกิดการขัดแย้งขึ้นเล็กน้อย มีการสังเกตอยู่ 3 แบบในเรื่องนี้

1. อันดับแรก datapath ทั่วไปที่ศึกษามาก่อนหน้านี้ใช้คำสั่งและข้อมูลของหน่วยความจำที่แยกออกจากกันการใช้ที่เก็บที่แยกจากกันนั้น จะจำกัดข้อขัดแย้งสำหรับหน่วยความจำ หน่วยหนึ่งซึ่งจะเกิดขึ้นระหว่างการรับคำสั่ง และ การเก็บข้อมูล สังเกตได้ว่าถ้า Pipeline มี clock cycle ซึ่งเท่ากับ unpipeline
2. register file ถูกใช้ใน 2 ปแบบ แบบที่ 1 ใช้สำหรับอ่านใน ID และแบบที่ 2 ใช้เขียน WB การใช้นี้จะมีลักษณะเด่นที่สังเกตไดชัด เราจะแสดง register file ใน 2 ที่ ซึ่งหมายความว่า เราต้องการเพื่อปฏิบัติในการอ่านและการเขียนอยู่ใน register เดียวกัน ในตอนนี้เรายังไม่สนใจ ในชุดนี้ แต่เราจะนำไปศึกษาในคราวหลัง
3. ในรูป 3.3 ไม่ได้มีการติดต่อกับ PC ในทุก ๆ clock และพวกนี้ ต้องทำในช่อง IF ในการเตรียมการสำหรับคำสั่งถัดไป ปัญหาจะเกิดขึ้นเมื่อเราคิดเกี่ยวกับผลกระทบของ branchrs ซึ่งจะเปลี่ยนแปลงใน PC ด้วย แต่จะไม่เกิดจนกระทั่งถึงขั้นตอนของ MEM แต่นี่ไม่ใช่ปัญหาใน Multicycle ตั้งแต่ PC ถูกเขียนใน MEM stage ตอนนี้เราจะจัดการ Pipelined datapath ของเราเพื่อใชเขียน PC หรือ ค่าของเป้าหมายของ branch ในตัวแรก ๆ ซึ่งสิ่งนี้ก็ทำให้เกิดปัญหา branch ถูกใช้ได้อย่างไร เราจะอธิบายในส่วนนี้ในรูป



เพราะว่าในทุก ๆ pipe stage ทำงานบน clock cycle และขั้นตอนทั้งหมดใน Pipe stage ต้องเสร็จสมบูรณ์ใน 1 clock cycle และการรวมตัวกันต้องเกิดขึ้น 1 ครั้ง การส่งคำสั่งไปยังส่วนต่าง ๆ จำเป็นต้องผ่านจาก pipe stage หนึ่งไปยัง pipe stage ถัดไปซึ่งต้องบรรจุใน

การแสดง DLX pipeline ด้วย register ที่เหมาะสมเรียกว่า pipeline register หรือ pipeline latches ซึ่งอยู่ระหว่าง pipeline แต่ละช่วง register จะถูกแบ่งด้วยชื่อของช่วงที่ติดต่ออยู่ให้เห็นถึงความสัมพันธ์ของ pipeline แต่ละช่วงได้อย่างชัดเจน

Register ทั้งหมดต้องการที่จะรับค่าชั่วคราวระหว่าง clock cycle ใน 1 คำสั่ง จะถูกจัดเป็นหมู่ ๆ ใน pipeline register ขอบเขตของ IR ซึ่งเป็นส่วนหนึ่งของ IF/ID register จะถูกตั้งชื่อเมื่อถูกใช้จัดหาชื่อของ register

Pipeline register จะประกอบด้วยข้อมูลและการควบคุมจาก pipeline หนึ่งไปยังอีก pipeline หนึ่ง ค่าที่ต้องการสำหรับ pipeline ในช่วงถัดไป จะต้องถูกบรรจุในรูป register และลอกเลียนแบบจาก pipeline หนึ่งไปยัง pipeline จนกว่าความต้องการจะไม่มี ถ้าเราพยายามใช้ register แบบชั่วคราวที่มีอยู่ใน unpipelined data path ก่อนหน้านี้ ค่าต่าง ๆ สามารถถูกเขียนทับก่อนที่จะใช้เสร็จขอบเขตของ register operand ถูกใช้เพื่อเขียนบน load หรือ ALU ซึ่งถูกจัดหามาจาก MEM/ws pipeline register มากกว่าจาก IF/ID register ทั้งนี้เพราะว่าเราต้อง load หรือALU. ในการเขียนจุดหมายของ register จุดหมายปลายทางของ register จุดหมายปลายทางของ register คือ การลอกเขียนแบบจาก pipeline register ตัวหนึ่งไปยังตัวถัดไปจนกระทั่งช่อง WB ซึ่งคำสั่งจะทำงานได้ดีใน 1 ช่อง pipeline ในช่วงเวลาใดเวลาหนึ่ง การกระทำทุก ๆ อย่างจะเกิดขึ้นตามคำ

สั่งที่อยู่ในระหว่าง pipeline register 2 ตัว เราสามารถมองดูการทำงานของ pipeline ได้โดยการตรวจสอบว่ามีอะไรเกิดขึ้นบาง ในแต่ละช่วงเวลาของ pipeline ซึ่งขึ้นอยู่กับลักษณะชนิดของคำสั่ง

รูป แสดงการไหลของข้อมูลจากช่องหนึ่งไปยังอีกช่องหนึ่งซึ่งอยู่ถัดไป สังเกตุว่าการทำงานใน 2 ช่วงแรกจะไม่ขึ้นอยู่กับชนิดของคำสั่ง สาเหตุที่ไม่ขึ้นอยู่กับชนิดของคำสั่งก็เพราะว่าคำสั่งจะไม่ถูกถอดข้อความจนกว่าจะจบขั้นตอนของ ID การทำงานของ IF ขึ้นอยู่กับคำสั่งใน EX/MEM ที่เป็น branch ดังนั้น branch target ถูกใช้ทั้งรับคำสั่งและคำนวณ PC ตัวถัดไป

ในการควบคุม pipeline ทั่ว ๆ ไป เราต้องการแค่ดูว่าจะควบคุมอย่างไรสำหรับ 4 ช่องทางใน data path 2 ช่องทางในช่วงของ ALU จะขึ้นอยู่กับชนิดคำสั่งซึ่งโดย IR ของ ID/EX register

1. Top ALU Input multiplexer ถูกกำหนดโดยคำสั่งเป็น Branch หรือไม่
3. Bottom multiplexer ถูกกำหนดโดยคำสั่งเป็น register - register ALU operation หรือคำสั่งชนิดอื่น ๆ
4. The multiplexer ในช่วงของ IF ใช้ PC ตัวปัจจุบันหรือค่าของ EX /MEM . ALU Output ซึ่ง Multiplexer ตัวนี้ ถูกควบคุมโดย EX/MEM Cond
5. Multiplexer ตัวนี้ถูกควบคุมโดยคำสั่งของ WB load หรือ ALU

Basic Performance Issues in Pipelining (คำสั่งพื้นฐานที่เกิดขึ้นใน Pipelining)

Pipelining เป็นการเพิ่มคำสั่งใน CPU (จนของคำสั่งที่ถูกปฏิบัติต่อเวลาช่วงหนึ่ง) แต่ไม่ใช้การลดเวลาในการทำงานของคำสั่งตัวเดียวโดด ๆ ไม่มีทางที่จะทำงานได้เร็วขึ้น

ความเป็นจริงที่ว่าเวลาการทำงานของแต่ละคำสั่งไม่ได้ลดลงใน pipeline ซึ่งเราจะได้เห็นในเรื่องถัดไป เพิ่มเติม ข้อจำกัดเกิดขึ้นจาก pipeline ซึ่งเกิดจากความไม่สมดุลระหว่างแต่ระช่วงของ pipeline และจาก pipelining Overhead ความไม่สมดุลระหว่างช่วงของแต่ละ pipeline จะทำให้ความเร็วในการทำงานช้ากว่าที่ควรจะเป็นสำหรับ pipeline ช่วงที่ช้าที่สุด Pipeline Overhead เกิดจากการรวมกันของ pipeline register delay กับ clock skew

ถ้าเครื่องของเรามี CPI เรียบร้อยแล้วต่อจากนั้น MD Pipelining จะทำให้เราสามารถใช้เวลาได้น้อยลง เส้นทางในการลำเรียงข้อมูลของช่วงก่อนหน้านี้สามารถทำให้เป็นแบบ Single Cycle ได้โดยการถอดตัวล็อคออกและปล่อยให้ข้อมูลเคลื่อนที่จากวงจรที่ปฏิบัติงานอยู่ไปยังวงจรถัดไป

Example เวลาที่จำเป็นต้องใช้สำหรับการทำงาน 5 หน่วย (ใน 5 วงจร) มีดังนี้ 10 ns 8 ns, 10 ns , 7 ns สมมุติว่าการทำงานแบบ pipeline จะทำให้เพิ่มมาอีก 1 ms จะหาความเร็วของ,Single Cycle data path

Answer uepipelined machime ทำงานตามคำสั่งทั้งหมดใน Simgle clock cycle เพราะฉะนั้น clock cycle time จะเท่ากับผลบวกของเวลาในแต่ละขั้นตอน

Average instruction execution time = 10 + 8 + 10 +10 + 7 = 45 ns

เวลาของ clock cycle ใน pipeline (Average instruction time Pipeling)จะเท่ากับเวลาที่ใช้มากที่สุดใน 1 stage บวกกับ 1 ns ที่เพิ่มมา ในที่นี้คือ 10 +1 ns

$$\text{Specd up form pipelining} = \frac{\text{Average instruction time unpipelined}}{\text{Average instruction time pipelined}}$$

$$= 45\text{ns}/11\text{ns} = 4.1 \text{ times}$$

เป็นเพราะว่า latches ที่อยู่ใน pipelined มีส่วนสำคัญในเรื่องความเร็วของ clock ผู้ออกแบบได้มอง latches ว่าเป็นตัวที่จะกำหนดความเร็วของ clock

The Earl latch มีความคิดเกี่ยวกับ latch อยู่ 3 ข้อ ซึ่งจะใช้ประโยชน์ได้เป็นอย่างดีในเรื่องของ pipeline

1. ความสัมพันธ์ที่ไม่มีปฏิกิริยาต่อความบิดเบือนของ clock
2. ความล่าช้าในการทำงานของ latch จะคงที่อยู่ที่ 2 ช่วงเสมอเผยหลีกเลี่ยงความบิดเบือนในข้อมูลที่ผ่านมายัง latch
3. ใน 2 ช่วงที่กล่าวมาใน 2 ข้อ สามารถทำงานใน latch โดยไม่ล่าช้าได้ ซึ่งหมายความว่า 2 ช่วงที่ว่านี้สามารถทำงานซ้อนกันได้ด้วย latch ดั้งนั้นเวลาที่เพิ่มขึ้นมาก็จะไม่มี

THE Major Hurdle of Pipelining - Pipeline Hazards

มีสถานะการณ์ที่เรียกว่าการกีดขวาง ซึ่งจะขัดขวางการเคลื่อนที่ของคำสั่งจากที่หนึ่งไปยังอีกที่หนึ่ง สิ่งกีดขวางนี้จะลดความเร็วในการทำงานของ pipeline แบ่งได้ออกเป็น 3 แบบ

1. Structural Hazard เกิดจากความขัดแย้งอันเนื่องมาจาก HW ไม่สามารถรองรับการรวมกันของคำสั่งที่ทำให้ทำงานซ้อนๆ กันได้
2. Data Hazard เกิดขึ้นเมื่อคำสั่งหนึ่งต้องขึ้นกับผลของคำสั่งที่ก่อนหน้า ซึ่งทำงานแบบซ้อนๆ กันใน pipeline
3. Control Hazard เกิดจาก pipeline ของ branch และคำสั่งอื่น ๆ ที่เปลี่ยนแปลง PC

ในการที่จะกำจัด Hazard นั้น จำเป็นต้องยอมให้คำสั่งใน pipeline ทำงานได้ในขณะที่ตัวอื่น ๆ กำลังล่าช้าอยู่

Performance of Pipelines With Stall

การถ่วงเวลาทำให้การทำงานของ pipeline ลดประสิทธิภาพลงจากความเป็นจริงของการทำงานจริง ๆ จงดูที่สมการที่ใช้หาความเร็วจาก pipeline โดยเริ่มจากสมการที่เขียนไปแล้วในช่วงแรก

$$\text{Speedup From Pipelining} = \frac{\text{Average instruction time unpipelined}}{\text{Average instruction time pipelined}}$$

$$= \frac{\text{CPI Unpipelined} * \text{Clock cycle Unpipeline}}{\text{CPI Unpipelined} * \text{Clock cycle pipeline}}$$

การทำงานของ pipeline ทำให้ลด CPI หรือ clock cycle time มันเป็นเรื่องปรกติที่จะใช้ CPI ในการศึกษา pipeline ซึ่งเราจะเริ่มจากจุดนี้ The ideal CPI ใน pipeline จะเป็น 1 เสมอ

$$\text{CPI pipelined} = \text{Ideal CPI} + \text{pipeline stall clock cycle per instruction}$$

$$= 1 + \text{pipeline stall clock cycle per instrution}$$

แต่ถ้าเราไม่สนใจเวลาที่เพิ่มขึ้นใน pipeline และสมมุติว่าแต่ละช่วงสมดุลกันหมดต่อจากนั้น cycle time ของเครื่องกลทั้ง 2 จะเท่ากัน ซึ่งนำไปสู่

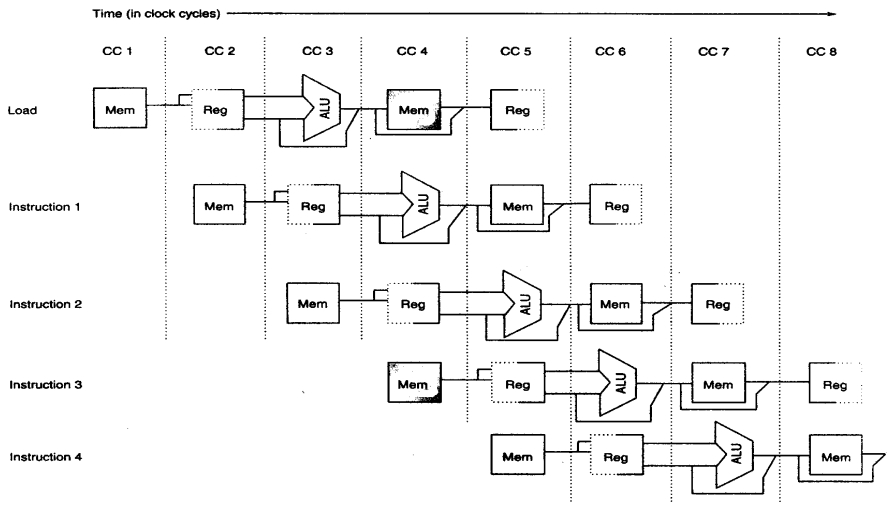
$$\text{Speedup} = \frac{\text{CPI pipelined}}{= 1 + \text{pipeline stall clock cycle per instrution}}$$

Structurol Hazards

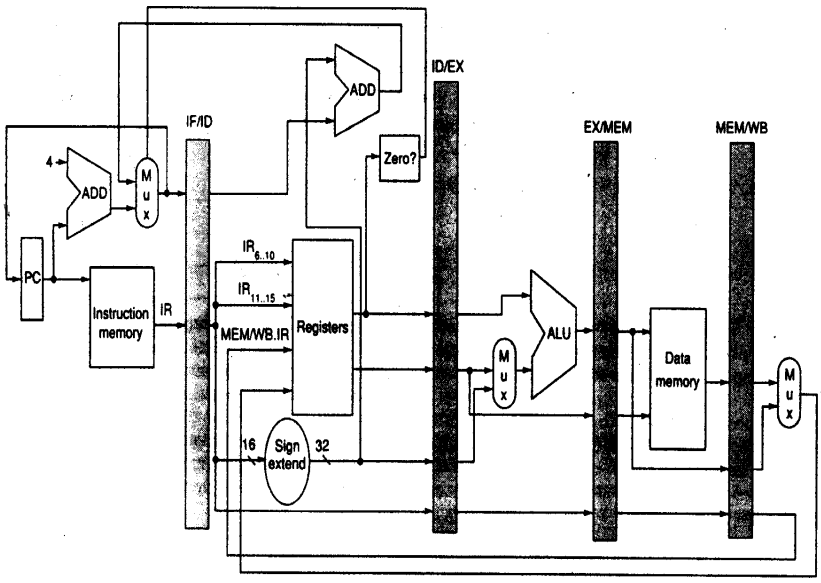
เมื่อเครื่องกลเป็นแบบ pipelined การทำงานแบบซ้อนๆ กัน ของคำสั่งจำเป็นต้องใช้การส่งคำสั่งและการทำซ้ำของวิธีการเพื่อที่จะทำให้เกิดการรวมกันของคำสั่งใน pipeline ถ้าการรวมกันของคำสั่งปรับเข้าหากันไม่ได้ เนื่องมาจาก วิธีการขัดแย้งกันซึ่งเราเรียกว่า Structural Hazard ตัวอย่างที่เห็นได้ชัดเกี่ยวกับเรื่อง Structural Hazard จะเกิดขึ้นเมื่อ มีบางคำสั่งไม่ได้เป็นการทำงานแบบ pipeline ผลที่ตามมาคือ คำสั่งที่ใช้การทำงานแบบ unpipeline จะไม่สามารถทำงานได้ 1 อย่าง ต่อ 1 clock cycle และอีกทางหนึ่งที่ Structural Hazard จะเกิดขึ้นได้ก็คือ เมื่อวิธีการไม่ได้ทำซ้ำ ๆ กันจนเพียงพอที่จะนำมารวมกันได้

Pipeline บางตัวมีการแบ่ง Single – Memory กันใช้สำหรับข้อมูลและคำสั่งซึ่งจะส่งผลให้การขัดแย้งกันได้ ดังรูป การที่จะแก้ปัญหานี้ เราต้องถ่วงเวลาของ pipeline สำหรับ clock cycle ดังรูป

Stage	Any instruction		
IF	IF/ID.IR ← Mem[PC]; IF/ID.NPC,PC ← (if EX/MEM.cond {EX/MEM.ALUOutput} else {PC+4});		
ID	ID/EX.A ← Regs[IF/ID.IR$_{6..10}$]; ID/EX.B ← Regs[IF/ID.IR$_{11..15}$]; ID/EX.NPC ← IF/ID.NPC; ID/EX.IR ← IF/ID.IR; ID/EX.Imm ← (IF/ID.IR$_{16}$)^{16}##IF/ID.IR$_{16..31}$;		
	ALU instruction	**Load or store instruction**	**Branch instruction**
EX	EX/MEM.IR ← ID/EX.IR; EX/MEM.ALUOutput← ID/EX.A *func* ID/EX.B; or EX/MEM.ALUOutput ← ID/EX.A *op* ID/EX.Imm; EX/MEM.cond ← 0;	EX/MEM.IR← ID/EX.IR EX/MEM.ALUOutput ← ID/EX.A + ID/EX.Imm; EX/MEM.cond ← 0; EX/MEM.B← ID/EX.B;	EX/MEM.ALUOutput ← ID/EX.NPC+ID/EX.Imm; EX/MEM.cond ← (ID/EX.A *op* 0);
MEM	MEM/WB.IR ← EX/MEM.IR; MEM/WB.ALUOutput ← EX/MEM.ALUOutput;	MEM/WB.IR ← EX/MEM.IR; MEM/WB.LMD ← Mem[EX/MEM.ALUOutput]; or Mem[EX/MEM.ALUOutput] ← EX/MEM.B;	
WB	Regs[MEM/WB.IR$_{16..20}$] ← MEM/WB.ALUOutput; or Regs[MEM/WB.IR$_{11..15}$] ← MEM/WB.ALUOutput;	Regs[MEM/WB.IR$_{11..15}$] ← MEM/WB.LMD;	



จากรูปแสดง pipeline data path แบบที่มีการถ่วงเวลาการถ่วงเวลาเราเรียกว่า pipeline bubble หรือ bubble และเราจะเห็น stall ชนิดอื่น ๆ เมื่อเราพูดเรื่อง Data hazard



แสดงการถ่วงเวลาโดยการแสดงวงจร เมื่อไม่การทำงานเกิดขึ้นและย้ายคำสั่งที่ 3 ไปทางขวา ซึ่งการล่าช้าของงานเริ่มและจบใน 2 วงจร

Control Hazards

โครงสร้างของ hazard ทำให้สิ่งที่มีประสิทธิภาพเสียไปเมื่อแยกทำ มันอาจจะไม่เปลี่ยน PC บางเวลาในปัจจุบันมีค่า +4 การเรียกกลับถ้าจะเปลี่ยน PC มันจะเป็นเป้าหมายของที่อยู่ ซึ่งทำการแยกตลอด อาจจะทำหรือไม่ทำ ถ้าคำสั่งทำการแยกหาก PC เป็นปรกติก็จะไม่เปลี่ยนจนกระทั่งจบ memory ซึ่งเป็นระเบียบที่ง่ายเกี่ยวกับการแยกสาขาของ pipeline ในไม่ช้าก็จะเสียหาย จนกระทั่งให้ memory ทำเป็นตอน

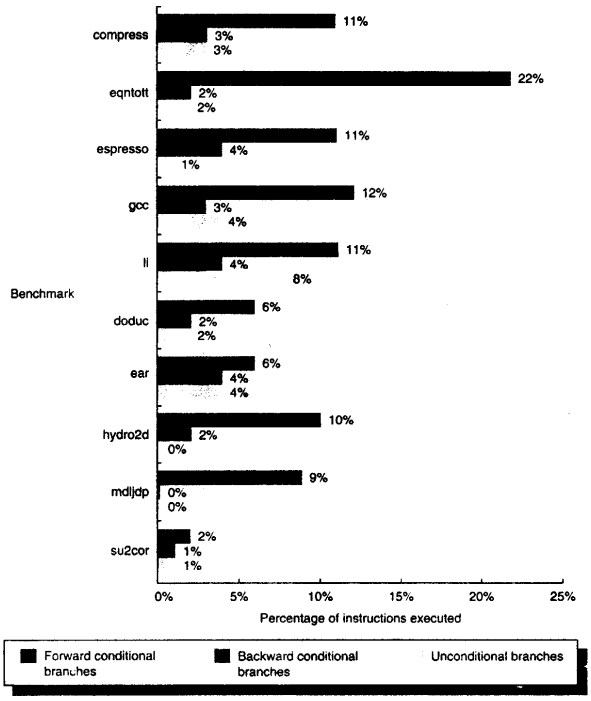
Branch Behavior in programs

การแยกจะมีผลกระทบ เราควรจะมองที่พฤติกรรมที่รับเกี่ยวกับบทบาทการแยกและการกระโดด ความสามารถจะลดน้อยลง การวิจาร์ณเกี่ยวกับการแสดงความถี่ควบคุมเกี่ยวกับการไหลของกระบวนการ SPEC เป็นกลุ่มย่อยของ DLX ซึ่งจะทำให้เครื่องเสียในระหว่างการแยกและการกระโดดแสดงเงื่อนไขเกี่ยวกับการแยกด้วยการทำลาย ซึ่งจะส่งผลและห่างไกลต่อการแยกเมื่อประสิทธิภาพของการทำ pipeline อาจขึ้นอยู่กับสิ่งใดสิ่งหนึ่งหรืออาจจะไม่แยกก็ได้ ข้อมูลกลายเป็นคำตอบซึ่งมี 2 ลำดับ

Peducing Pipeline Branch Penalties

เป็นวิธีการปฏิบัติร่วมกันกับ pipeline ซึ่งเป็นเหตุให้ผ่านการแยกโดยล่าช้า ผู้วางแผนการคำนวณเวลามีส่วนประกอบเล็ก ๆ อยู่ที่การคิดค้นและการแยกที่แน่นอน ซึ่งการแยกทำให้การนำความรู้สู่ระบบคอมพิวเตอร์ จากการวางแผนและแยกลักษณะการทำงานของเครื่องหลังจากอธิบายได้แล้วสิ่งเหล่านี้

จะเป็นสูตร เราสามารถตรวจสอบเวลาในการคำนวณแยกตามที่มีการบอกล่วงหน้า เมื่อแยกแล้วทำให้มีประสิทธิภาพทุกอย่างที่อยู่ในเทคโนโลยี ในแผนคำนวณเวลาที่ลดน้อยลง ความถี่ของการแยกที่เปลี่ยนแลงไม่คงที่ การแยกเป็นการหยุดนิ่งในการส่งซึ่งจะยึดหรือทำลายบ้างมีการแนะนำหลังจากแยกจนกระทั่งสู่จุดหมายของความรู้ การดึงดูดความสนใจเกี่ยวกับวิธีแก้ไขระดับรกเริ่มในความง่ายของมันทั้งคู่เพราะ HW และ SW มันเป็นวิธีแก้ไขที่ไม่มีใน pipeline ในการตรวจสอบในการแยกกฎเป็นการยึดและไม่สามารถให้ลดน้อยลงได้ผ่านทางโปรแกรมในฐานของการยอมรับ HW โดยทำต่อไปไม่หยุด จากรูป



ทุก ๆ สาขาที่เป็นการเอามากำหนดการแยกเป็นการถอดระหัสและที่อยู่ของจุดหมายเป็นการคำนาณแยกและเริ่มนำมาปฏิบัติเพราะใน DLX pipeline จะไม่ทราบถึงจุดหมายในบางระบบมีการสนับสนุนอย่างแน่นอนโดยจัดให้เหมาะสำหรับ สำหรับโดยการแยกสาขาของจุดหมายให้รู้ก่อน ผู้คำนวณสามารถปรับปรุงแก้ไขประสิทธิภาพโดยผ่านผู้จัดตั้งระหัส ดังนั้นตัวมากสุดที่เกิดขึ้นบ่อยทั้งที่มีคุณสมบัติเท่ากันใน HW ที่มีตัวเลือกแบบแผนที่จัดเตรียมมากสุดที่มีโอกาส เพื่อที่จะคำนวณ ที่จะแก้ไขปรับปรุงประสิทธิภาพ

รูป 27

Untaken branch instruction	IF	ID	EX	MEM	WB				
Instruction *i* + 1		IF	ID	EX	MEM	WB			
Instruction *i* + 2			IF	ID	EX	MEM	WB		
Instruction *i* + 3				IF	ID	EX	MEM	WB	
Instruction *i* + 4					IF	ID	EX	MEM	WB

Taken branch instruction	IF	ID	EX	MEM	WB				
Instruction *i* + 1		IF	idle	idle	idle	idle			
Branch target			IF	ID	EX	MEM	WB		
Branch target + 1				IF	ID	EX	MEM	WB	
Branch target + 2					IF	ID	EX	MEM	WB

จากรูป เป็นการกำหนดแผนงานล่วงหน้าด้านบนของช่องแต่ละคู่ จะบอกถึงระหัสก่อนจะกำหนดแผนงานล่วงหน้าส่วนที่อยู่ด้านกลางแสดงให้เห็นถึงแผนงานล่วงหน้าของรหัส (a) จะถ่วงเวลาทางเดินเป็นการวางแผนงาน เนื่องจากไม่ต้องการยุ่งเกี่ยวกับใครจะมีการแนะนำก่อนที่จะมีการแยกเป็นตัวเลือกที่ดีที่สุด (b) และ (c) เป็นส่วนที่คล้าย (a) ในรหัสจะลำดับเหตุการณ์ เพราะ (b) และ (c) ใช้ประโยชน์จาก R 1 ในการแยกและปรับให้เหมาะสำหรับการใช้งาน เพื่อป้องกัน ADD แนะนำจากสิ่งที่ปรากฏอยู่จากการเคลื่อนที่หลังจากการแยกใน (b) ถ่วงเวลาในการแยก เป็นการกำหนดแผนงานล่วงหน้า เพราะว่ามันสามารถผ่านจากสิ่งหนึ่งไปยังอีกสิ่งหนึ่งตามเส้นทาง (b) เป็นตัวที่มีโอกาสได้สูง ในที่สุดการแยกอาจเป็นการกำหนดแผนงานล่วงหน้าผ่านเข้าไปจนกระทั่งใน (c) ส่วนนี้ทำให้มีประสิทธิภาพในกฎ เพราะ (b) หรือ (c) ซึ่งต้องเป็น OK เป็นการปฏิบัติ SUB

Summary : Performance of the DLX INTEGER Pipeline

จบส่วนนี้บน hazard ถึงการค้นพบและทำลายผ่านออกสู่สายตาทั้งหมด การกระจายอย่างช้า ๆ ในการหมุนเวลาจากจำนวนเต็มที่มีโอกาสวิ่งบน DLX ของ pipeline กับ SW เป็นเพราะ pipeline ที่กำหนดล่วงหน้าไว้แล้ว

Instruction Set Complication

DLX การแนะนำจะมีมากกว่า 1 ผลลัพธ์ และ DLX pipeline การเขียนนั้นจะมีผลลัพธ์เพียงจบเกี่ยวกับการแนะนำเป็นการปฏิบัติงาน เพื่อต้องการให้สมบูรณ์ เป็นการเรียกตรวจสอบใน DLX เป็น pipeline ที่เป็นตัวเลขทุกอย่างมีการแนะนำ โอกาสที่จะไปถึงจนจบจาก memory นำมาแสดงและไม่แนะนำให้มีการเปลี่ยนแปลก่อนนำมาแสดง ดังนั้นความถูกต้องสมบูรณ์ อย่างตรงไปตรงมา สิ่งที่เพิ่มเติมในระบบตรงกับการปรับให้เหมาะกับการใช้งาน ผู้ดำเนินงานต้องตัดสินใจให้เหมาะ ซึ่งจะไม่มีการเปลี่ยนแปลงในส่วนนี้ทำให้ยุ่งยากต่อข้อสรุปโอกาสที่ปรับให้เหมาะกับรหัส มีการวางหลักเวลาหลังจากการแยกในระบบที่มากตรงกันอย่างแน่นอนเพื่อ

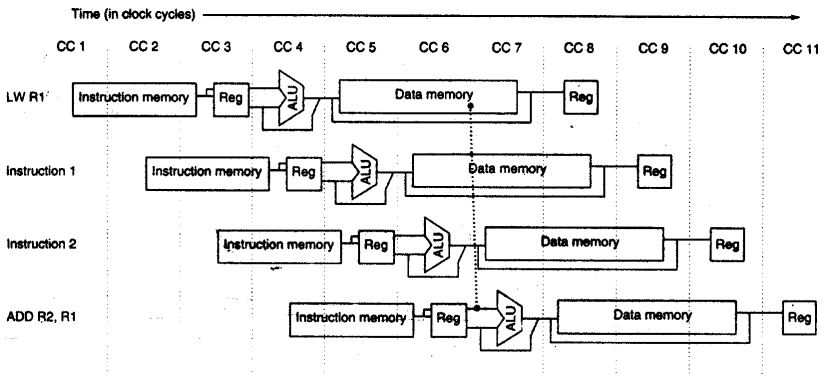
จัดให้เหมาะกับรหัด โดยส่วนนี้ปฏิบัติโดยการยืดเวลาออกไปในการปรับให้เหมาะกับการประเมินค่า

Extending the DLX Pipeline to Handle multicycle Operations

สิ่งนี้สามารถนำไปใช้ประโยชน์ได้ตามต้องการนั่นคือ DLX เป็นจุดทศนิยมทั้งหมดใน 1 clock cycle หรือเสมอกันทั้งสอง ดังนั้นจะไม่ดีในข้อของการยอมรับเวลาจะช้าหรือเป็นตัวที่ใหญ่มากเกี่ยวกับเหตุผลในหน่วยของทศนิยม หรือเป็นคู่แทนจุดทศนิยม ทศนิยมของ pipeline ต้องการยินยอมให้ช่องที่ซ้อนอยู่กับกระบวนการส่วนนี้เป็นอย่างง่ายดาย ถึงความเข้าใจถ้าวางแผนแนะนำจุดทศนิยมในบางเวลา pipeline เป็นจำนวนจริงของการแนะนำตรงกับสองสิ่งที่มีความหมายในการเปลี่ยนอาจจะลำดับที่ 1 คือ Ex cycle ทำซ้ำจนเวลามากต้องการที่สมบูรณ์ จำนวนเกี่ยวกับลักษณะของการกระทำซ้ำ ๆ สามารถแตกต่างกันได้ เพราะว่ากระบวนการทำงานไม่เหมือนกัน

Performance of

DLXFP Pipeline แต่ละ Type ของจุดในรูปสามารถก่อให้เกิดโครงสร้างการหยุดการแบ่งหน่วยและการหยุดสำหรับ RAW hazards จะแสดงถึงตัวเลขในรอบของการหยุดของทศนิยมขบวนการจัดการแต่ละพื้นฐานข้อเท็จจริงซึ่งสามารถเชื่อถือประสิทธิภาพรอบของการหยุดของแต่ละกระบวนการของ FP มันจะผันแปลตามรูปแบบของลักษณะหน่วยของฟังก์ชันจะแสดงผลสำเร็จของการกระจายออกเป็นรายการย่อย ๆ ของตัวเลขและจุดทศนิยม



การหยุดที่มีอยู่สำหรับ DLXFP เป็นผลรวมของตัวเลขของการหยุดของแต่ละคำสั่ง การตรวจสอบผลลัพธ์ การหยุดทั้งหมดของ FP จะได้ค่าเฉลี่ยของรอบการหยุด การเปรียบเทียบการผลิตจะได้ค่าเฉลี่ยเท่ากับ 0.1 ของการหยุดต่อ 1 คำสั่ง

Crosscutting Issues:

Instruction set design and pipelining

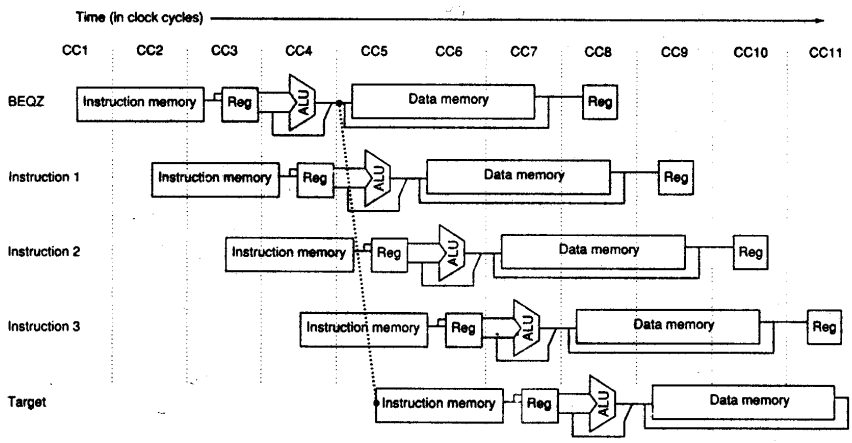
สำหรับอีกหลายปีผลกระทบระหว่างกลุ่มของคำสั่งและเครื่องมือจะเป็นปัญหาเล็ก ๆ และเครื่องมือมันจะไม่ใช้ประเด็นสำคัญในการออกแบบกลุ่มของคำสั่ง ในปี 1980 อุปสรรคต่าง ๆ มันเริ่มจะคี่คลาย และความไม่ลงตัวของ pipeline ความยุ่งยากมันจะเพิ่มขึ้นโดยชุดคำสั่ง

- ค่าความยาวของคำสั่งและเวลา running สามารถเป็นตัวนำที่ไม่สมดุลระหว่างการดำเนินการของ pipeline การป้องกันอันตรายและการบำรุงรักษาต้องทำอย่างเข้มงวดและแน่นอน ยกเว้นบางครั้งข้อดีของมันจะพิสูจน์ได้ว่าเป็นการเพิ่มความยุ่งยาก เช่น ปัญหาที่เกิดขึ้นของ cache คือจะใช้เวลามากในการ run ชุดคำสั่งจนสำเร็จ อย่างไรก็ตาม cache ยังมีข้อดี คือ มันจะเพิ่มความยุ่งยากก็แต่เฉพาะที่ตัวมันทำงานได้
- การแบ่งแยก address ที่เคยใช้มาแล้วสามารถใช้เป็นตัวนำได้ แบ่งประเภทตามความแตกต่าง address ที่กล่าวถึง มันจะใช้ update ในการลงทะเบียน
- สถาปัตยกรรมอนุญาติให้เขียนคำสั่งไว้ในช่องว่างที่ เป็นเช่น เพราะใช้ในรุ่น 80 x 88 สามารถทำความยุ่งยากใน pipeline ได้
- กลุ่มรหัสจำนวนมากมันจะยุ่งยากในการแบ่ง และยากในการตัดสินใจ สำหรับการกำหนดการเลื่อนระยะเวลาที่ล่าช้าของแต่ละปัญหา ตามสภาพแวดล้อมและรูปแบบการตัดสินใจจะกำหนดตามสภาพของรหัสสุดท้ายเงื่อนไข สุดท้ายที่เกิดขึ้นจะไม่ข้อกำหนด มันจะทำให้ยากในการค้นหา

เป็นที่คาดว่าคำสั่ง DLX จะมีความสลับซับซ้อนมาก การถอดรหัสจะแยกออกต่างหาก โดยความต้องการจะทำหลังการลงทะเบียน มันจะมีปริมาณมากขึ้น สำหรับการถ่วงเวลาของ Two-clock cycle ที่ดีคือ Second brand delay เพราะมันต้องชุดที่น้อยที่สุดเป็นอันดับแรก ปี 1983 เป็นการก่อตั้ง Second delay slot แต่มันจะเป็นเพียงครั้งหนึ่งเท่านั้น ยังไม่สมบูรณ์ ตัวนำที่มีประสิทธิภาพจะใช้เวลาเพียง 0.1 clock cycle ต่อ 1 ชุดคำสั่ง ที่เราใช้เปรียบเทียบสำหรับ pipeline คือ VAX 8800 และ MIPR 3000 แม้ว่าเครื่องนี้จะมีองค์ประกอบที่เหมือนกัน แต่ชุดคำสั่งของ VAX ไม่ได้ถูกสร้างขึ้นเพื่อใช้สำหรับ pipeline ผลลัพธ์ที่ได้บน SPEC 89 brank mark MIPSR 3000 มันมีความเร็วระหว่าง 2-4 เวลา สำหรับ MIPS 3000 มันมีข้อดีคือ ใช้เวลาเพียง 2.7 time

Putting it all To Together:The MIPS R4000 Pipeline

ในกลุ่มจะเห็นได้ว่าโครงสร้างของ pipeline และตระกูลของ MIPS 4000 เราจะแนะนำ MIPS-3 สำหรับ R 4000 มันเป็นอุปกรณ์ที่ใช้คำสั่ง 64 bit เหมือนกับ DLX เราสามารถใช้ R4000 ได้ดีกว่า DEX ในการทำเรื่องจนเต็มและ FP โปรแกรมความเร็วของ pipeline คือมันจะอนุญาติให้ บรรจุอัตรา clock ระหว่าง 100-200 MHz โดยจะสลายตัวเป็นจำนวนเต็ม tive-stage pipeline ใน 8 stage เพราะว่าการเข้าที่ซับซ้อนมันเป็นเวลาเฉพาะในระยะวิกฤตที่พิเศษสำหรับ pipeline มันจะสลายทางเข้าของ memory แต่ก่อนเราเรียกชนิดของความลับของ pipeline ว่า superpipelining จากรูปจะแสดงให้เห็นโครงสร้างของ 8 stage pipeline ซึ่งจะสรุปความสัมพันธ์ได้ตาม version



ต่อมารูปจะแสดงถึงความซับซ้อนของโครงสร้างใน pipeline ข้อมูลในหน่วยความจำจะถูกครอบครองเพิ่มเป็นทวีคูณใน cycle ข้อมูลเหล่านี้มันจะเพียงพอใน pipeline ดังนั้น คำสั่งใหม่ ๆ จะเริ่มบน every clock ในความเป็นจริง pipeline จะใช้ข้อมูลหลังจากที่ cache มันค้นพบจนสำเร็จ

หน้าที่การติดตามแต่ละ stage

- การกำหนดคำสั่ง IF ในครั้งแรก PC จะทำการเลือกสิ่งที่มีอยู่จริงโดยทำงานร่วมกับ cache
- การกำหนดคำสั่ง IS ในครึ่งสองคำสั่งจะสำเร็จก็ต่อเมื่อ cache เข้ามาทำงาน
- การถอดรหัสคำสั่ง RF และการกำหนด register จะทำการเช็ค hazard และ cache จะทำการตรวจค้นชุดคำสั่งทั้งหมด
- การปฏิบัติงานของ EX ส่วนนี้จะเป็นส่วนของการคำนวณที่มีประสิทธิภาพ , กระบวนการจัดการของ ALU และ การคำนวณของ branch target และการคำนวณเงื่อนไขต่าง ๆ
- การกำหนดข้อมูลในส่วนของ DF ข้อมูลครึ่งแรกมันจะให้ cache เข้ามาทำงาน

- DS ก็คือ ครึ่งที่สองมันก็จะให้ cache เข้ามาทำงานจนสำเร็จ
- TK มันคือ Tagcheck ข้อสรุปคือ cache จะเข้ามาทำงานในส่วนของ data
- WB การเขียนหลังการบรรจุ และ การชี้ให้เห็นขั้นตอนต่าง ๆ

สิ่งที่เพิ่มเติมลงไปโดยส่วนมากมันจะเพิ่มขึ้น จำนานที่เพิ่มขึ้นมันจะส่งเสริมต่อความต้องการต่อปัจจัยของ pipeline การ load และการ branch delays ในรูปจะแสดงถึงความล่าช้าของ two cycle ตั้งแต่ข้อมูลที่ว่างอยู่บนของ DS รูปจะแสดงถึงการบันทึกโดยใช้ตัวเลขของการกำหนด pipeline ซึ่งมันจะใช้บันทึกในทันทีตามที่มันต้องการบรรจุ และมันจะแสดงความต้องการที่อยู่หน้าผลลัพธ์ของการบรรจุคำสั่งจนถึงจุดหมายปลายทาง ซึ่งมันจะเป็น 3-4 cycle รูปจะแสดงให้เห็นของ branch delays ที่เป็น tree cycle ตั้งแต่เงื่อนไขของ branch ที่ทำการคำนวณระหว่าง EX โครงสร้างทางสถาปัตยกรรมของ MIPH มันจะเป็นแบบ single-cycle delayed branch delay รูปจะแสดงถึงตัวอย่าง อย่างง่ายของ one cycle delay branch คือมันจะมีช่วงต่อสำหรับ 2 รอบที่สูญเปล่า ชุดคำสั่งในการจัดหา branch มันจะบรรยายถึงระดับความสำคัญและสิ่งที่นำมาใช้ในการอุดช่อง branch delay pipeline ข้อบังคับในการใช้ pipeline ที่เชื่อมต่อกันของทั้งสอง cycle บน taken branch และหลาย ๆ data hazard ซึ่งมันจะใช้เลื่อนขึ้นเพื่อบรรจุผลลัพธ์

การเพิ่มจำนวนใน stall สำหรับการบรรจุและการ branch การเพิ่มระดับความลึกของ pipeline มันจะส่งเสริมการดำเนินการของ ALU ใน DLX five-stage pipeline ความก้าวหน้าระหว่าง two register-register ALU จะแนะนำสิ่งที่เกิดขึ้นของ ALU/MEM หรือ MEM/WB register ใน R 4000 pipeline , ผลลัพธ์ 4 อย่างที่เป็นไปได้สำหรับการหลีกเลี่ยง ALU คือ EX/DF , DF/DS , DS/TC และ TC/WB

The Floating-Point Pipeline

องค์ประกอบของ R 4000 floating-point unite จะมีอยู่ 3 ฟังก์ชันคือ การหารของ floating-point , การคูณของ floating – point , การบวกของ floating-point ใน R 3000 การเพิ่ม logic มันจะใช้บน step สุดท้ายของการคูณหรือการหารขบวนการ double-precision FB สามารถสร้างรูปแบบ 2 cycle ให้เพิ่มขึ้นถึง 112 cycle เพื่อการค้นหาอย่างสอดคล้องในการเพิ่มหลาย ๆ หน่วย จะมีอัตราความแตกต่าง floating-point functional unite สามารถกำหนดขั้นตอนความแตกต่างได้ 8 อย่างในรูป

Performance of the R 4000 Pipeline

ในหมวดหมู่นี้เราจะตรวจสอบความหน่วงที่เกิดขึ้นใน SPEC 92 benchmarks ซึ่งมันจะ run บน R 4000 Pipeline จะมีสาเหตุสำคัญอยู่ 4 ประการ

1. Load stalls รูปแบบของการล่าช้าคือ การที่ใช้บรรจุผลลัพธ์ใน 1 หรือ 2 cycle หลังจากการ load
2. Branch stall-to cycle ตามหน่วงที่เกิดขึ้นกับการบวกมันจะไม่สมบูรณ์หรือการยกเลิกช่อง branch delay
3. FP result stalls มันหน่วงเพราะ RAW hazard สำหรับขบวนการ FP
4. โครงสร้างของ FP มันล่าช้าเพราะว่าการจำกัด การเลื่อนขึ้นในรูปแบบของความขัดแย้งสำหรับฟังก์ชันใน FP pipeline

Fallacies and Pitfalls

อันดับแรก WAW hazard มันจะมองเห็นการเชื่อมของสิ่งที่ไม่เคยเกิดขึ้นของสิ่งที่ไม่เคยเกิดขึ้นเพราะว่าไม่มี compiler ที่จะทำให้เกิดขึ้น แต่สามารถทำให้เกิดขึ้นได้โดยการเรียงลำดับแบบไม่คาดคิด

Concluding Remarks

Pipelining และการทำงานต่อไปมันเป็น 1 ในเทคนิคที่มีส่วนสำคัญสำหรับการเพิ่มประสิทธิภาพของ Processors ทางผ่านที่มีประสิทธิภาพของ pipelining เป็นกุญแจสำคัญในการเพิ่มการออกแบบของ designer ในช่วงปลาย 1950 จนถึงตอนกลางของ 1960 ในช่วงท้ายของ 1960 จนถึงช่วงท้ายของ 1970 เป็นช่วงที่ใช้พิจารณาโครงสร้างของ computer ในหลาย ๆ ความคิดที่จัดปรับปรุงแก้ไขราคา ขนาด และความเชื่อคือมันจะได้มาซึ่งคำแนะนำที่สมบูรณ์แบบของเทคโนโลยีในช่วงนี้เป็นช่วงสุดท้ายของ pipelining ต่อไปนี้ pipeline จะไม่ใช้กุญแจสำคัญในการออกแบบในหลายคำสั่ง เพราะเราจะใช้ pipeline น้อยลง และจะเปลี่ยนมาใช้สถาปัตยกรรม VAX ซึ่งมันอาจจะดีที่สุดในช่วงท้าย 1970 หรือก่อน 1980 นักวิจัยเริ่มเข้าใจในชุดคำสั่งที่ซับซ้อน และดำเนินการโดยเฉพาะอย่างยิ่ง pipeline นี้เราจะเลิกใช้ การดำเนินการของ RISC เกิดจากศิลปะแบบง่าย ๆ ในชุดคำสั่งที่ยินยอมอย่างรวดเร็ว และสม่ำเสมอ ในการพัฒนาเทคนิคของ pipeline พวกเราจะเห็นต่อไปที่จะนำเทคนิคเข้ามาใช้อย่างผู้มีประสบการณ์ เทคนิคที่ว่าจะใช้ออกแบบอย่างยากลำบากเพราะมันจะเป็นสถาปัตยกรรมที่สลับซับซ้อนในช่วงปี 1970 ในที่นี้เราจะแนะนำ pipeline และแผนการอย่างง่าย ๆ ของ compiler เพื่อยกระดับประสิทธิภาพมากขึ้น pipeline microprocessor ในอนาคตเราจะปรับปรุงอย่างมีประสิทธิภาพ ในทศวรรษต่อไป microprocessor มันจะแนะนำแบบ

แผนของ branch prediction ถึงความสามารถจะถูกตีพิมพ์มาในแบบใหม่เพื่อแนะนำสิ่งที่เกิดขึ้นใน cycle และใช้ความสามารถของเทคโนโลยีของ compiler ความก้าวหน้าที่จะเกิดขึ้นของเทคโนโลยีจะถูกล่าวถึงต่อไป

Historical perspective and References

ส่วนสำคัญที่จะกล่าวถึงคือความก้าวหน้าของ pipeline เครื่อง RISC เป็นต้นแบบของแนวคิดแบบง่ายต่อการดำเนินการและ pipeline ในข้อคิดเห็นต่าง ๆ นา ๆ ก่อนที่ RISC จะออกเผยแพร่ในช่วง 1980 เพื่อให้ก้าวหน้าอย่างมีประสิทธิภาพ อย่างไรก็ตามการเปรียบเทียบระหว่างการดำเนินการ VAX และ MIPX โดยหลังจากที่ RISC ที่ออกเผยแพร่ ข้อโต้แย้งที่เกิดขึ้นต่อการดำเนินการ ที่ก่อให้เกิดผลประโยชน์ของ RISC โดยข้อความที่แจกจ่ายนี้ส่วนใหญ่จะไม่หน้าเชื่อถือต่อความก้าวหน้าของชุดคำสั่งแบบสถาปัตยกรรม RISC

Instruction – level Parallelism

Concepts and Challenges

เราจะพูดถึง pipeline ที่ปฏิบัติงานอย่างซับซ้อนของคำสั่งจะมีอิสระต่อกัน อาจจะเป็นไปได้ที่จะเกิดการซ้อนกันระหว่างคำสั่งที่เรียก ILP คำสั่งจะประเมินค่าแบบขนานเราจะมองเห็นความสำคัญเป็นแนวกว้างถึงเทคนิค เพื่อแนวความคิด pipeline โดยความสามารถจะเพิ่มขึ้นแบบขนานระหว่างกลางของชุดคำสั่งเราจะเริ่มมองเห็นเทคนิคที่เปรียนไปโดยผลกระทบจะเกิดขึ้นกับข้อมูลและตัวควบคุมและการกลับมาของหัวข้อสำคัญความสามารถของ processor จะส่งเสริมกระบวนการนี้เราจะโต้ตอบกับ compiler โดยใช้ ILP มากขึ้นและตรวจสอบผลลัพธ์จากการเรียนรู้ ILP Pipelining it all together ส่วนนี้จะปกป้อง Power PC 620ซึ่งมันจะสนับสนุนเทคนิค pipeline ที่ก้าวหน้า

ในส่วนนี้เราจะอธิบายถึงการเพิ่มขึ้นและขอบเขตทั้ง program และ processor ซึ่งมันจะสัมพันธ์กันสามารถส่งเสริมระหว่างชุดคำสั่ง เราสรุปได้ว่าส่วนที่เราเห็นเทคนิค compiler อย่างง่ายสำหรับยกระดับการใช้ให้เป็นประโยชน์ของ pipeline ซึ่งมันจะคล้ายกับ compiler

Basic Pipeline Scheduling and Loop Unrolling

การเก็บ pipeline อย่างเต็มรูปแบบจะพูดถึงระหว่างคำสั่งกับการส่งเสริมการค้นหาแบบเรียงลำดับโดยไม่เกี่ยวข้องกับคำสั่งโดยซ้อนกันได้ใน pipeline ความหน่วงใน pipeline จะไม่มีประสิทธิภาพ คำสั่งจะแยกจากกันโดยแหล่งกำเนิดของคำสั่งโดยระยะห่างใน clock cycle compiler สามารถกระทำรายการต่าง ๆ ได้โดยขึ้นอยู่กับจำนานที่ว่างบน ILP ใน program และบนลักษณะต่าง ๆ ของฟังก์ชันใน pipeline โดยจะแสดงในตารางเราจะพูดถึงจำนวนตัวเลขที่มาตรฐาน DLX pipeline branches จะมีความล่าช้าใน 1 clock cycle เราจะรู้ได้ว่าชุดของฟังชันเต็ม

แล้วภายใน pipeline หรือเป็นที่พับในหลาย ๆ ชนิดสามารถบอกถึงประเด็นสำคัญในหลาย ๆ clock cycle และไม่เป็นโครงสร้างของ Hazards

Dependences

ข้อสรุปในหนึ่งคำสั่งจะขึ้นอยู่กับอีกสิ่งหนึ่งในกรณีที่ฉุกเฉินเท่านั้น จะไม่กำหนดรายการการกระทำหรือกระบวนการปฏิบัติ จะตัวอย่างแต่ด้วยการกำหนดที่มากใน program และคล้ายคลึงกันสามรถกระทำได้ในรายละเอียด ความสามรถในการใช้คำสั่งจะคล้ายคลึงกันจะกำหนดคำสั่งและกระทำได้ในประเภทเดียวกัน ถ้าคำสั่งทั้ง 2 คำสั่ง มาทำในประเภทเดียวกันเราจะสามารถปฏิบัติได้พร้อม ๆ กันและในเวลาเดียวกันการทำงานภายนอกของ pipeline จะถ่วงเวลา เพราะไม่ได้ใช้คำสั่งประเภทเดียวกัน เช่นเดียวกันทั้ง 2 คำสั่งก็จะไม่สามารถทำตามคำสั่งได้ คำสั่งสามารถส่งให้ทำงานประเภทเดียวกันของมันแต่มันจะทำงานได้ไม่ดีเท่าทีควร

Another Dynamic Scheduling Approach The Tomasule Approach

การปฏิบัติงานจะยอมให้เข้าไปดำเนินการใน hazards ด้วยการใช้จุดทศนิยม ก่อนที่จะกล่าวถึงจะแนะนำขั้นตอนสั้น ๆ 3 ขั้นตอนคือ

1. Issue-Get คำสั่งจากจุดทศนิยมจะมีกระบวนการประเด็นสำคัญจะมีการจองรักษาจุดทศนิยม การจองจะเป็นการจองโดยอัตโนมัติ ถ้าขั้นตอนบรรจุข้อมูลมันจะเป็นจุดมุ่งหมายในหน่วยความจำ
2. Exeute-it เครื่อง CDB จะรอจนเสร็จการคำนวณและการบันทึกอัตโนมัติ ซึ่ง RAW จะตรวจสอบข้อผิดพลาดที่เกิดจากอันตราย
3. Write result ผลลัพธ์ที่ได้จะเขียนบน CDB และใน register ในการรอผลลัพธ์