บทที่ 6

หน่วยประมวลผล

6.1 หน้าที่ของหน่วยประมวลผล

6.2 ส่วนประกอบของหน่วยประมวลผล

6.3 การทำงานของหน่วยประมวลผล

6.4 เอกสารอ้างอิงและเว็บไซต์ที่ควรรู้

วัตถุประสงค์

- ส่วนประกอบหลักของหน่วยประมวล
- หน้าที่และการทำงานของหน่วยประมวลผล
- เทคนิคการทำไปป์ไลน์

หน่วยประมวลผลกลางหรือซีพียู เปรียบเสมือนหัวใจหลักของคอมพิวเตอร์ ทำหน้าที่ควบคุมการทำงานของคอมพิวเตอร์ ซีพียูประกอบขึ้นจากวงจรอิเล็กทรอนิกส์ที่ซับซ้อนบรรจุอยู่ภายในชิปตัวหนึ่ง มักเรียกชิปนี้ว่าโปรเซสเซอร์ ผู้พัฒนาชิปในปัจจุบันได้พัฒนาชิปให้มีประสิทธิภาพมากขึ้น ดังจะเห็นได้จากพัฒนาชิปให้มีความเร็วเพิ่มขึ้น และมีขนาดเล็ก หรือการพัฒนาชิปเป็นดูอัลคอร์ (dual core) หรือควอดคอร์ (quad core) ซึ่งเป็นการรวมเอาสองหรือสี่โปรเซสเซอร์ไว้ในชิปตัวเดียวกัน ในคอมพิวเตอร์เครื่องหนึ่งๆ จะต้องมีซีพียูอย่างน้อยหนึ่งตัวหรืออาจมีมากกว่าหนึ่งขึ้นไปได้

6.1 หน้าที่ของหน่วยประมวลผล

หน่วยประมวลผลจะทำงานตามคำสั่งที่ผู้ใช้ป้อนเข้าไป โดยจะทำงานตามขั้นตอนหลักดังต่อไปนี้

1. **ดึงคำสั่ง** (fetch instruction) ซีพียูจะทำการอ่านคำสั่งในหน่วยความจำหลักเข้ามาเก็บไว้ภายใน
2. **แปลความหมาย** (interpret instruction) จากนั้นจะทำการแปลความหมายของคำสั่ง เพื่อให้รู้ว่าคำสั่งนั้นสั่งให้ทำงานอะไร
3. **ดึงข้อมูล** (fetch data) ในกรณีที่คำสั่งนั้นสั่งให้ทำงาน โดยมีการอ่านข้อมูลจากหน่วยความจำหรืออุปกรณ์ภายนอก
4. **ประมวลผลข้อมูล** (process data) ทำการประมวลผลคำสั่งซึ่งอาจเป็นการประมวลผลทางคณิตศาสตร์หรือตรรกะ
5. **บันทึกข้อมูล** (write data) เมื่อได้ผลลัพธ์จากการประมวลผลคำสั่ง อาจมีการบันทึกผลลัพธ์ไปไว้ในหน่วยความจำหลักหรืออุปกรณ์ภายนอก

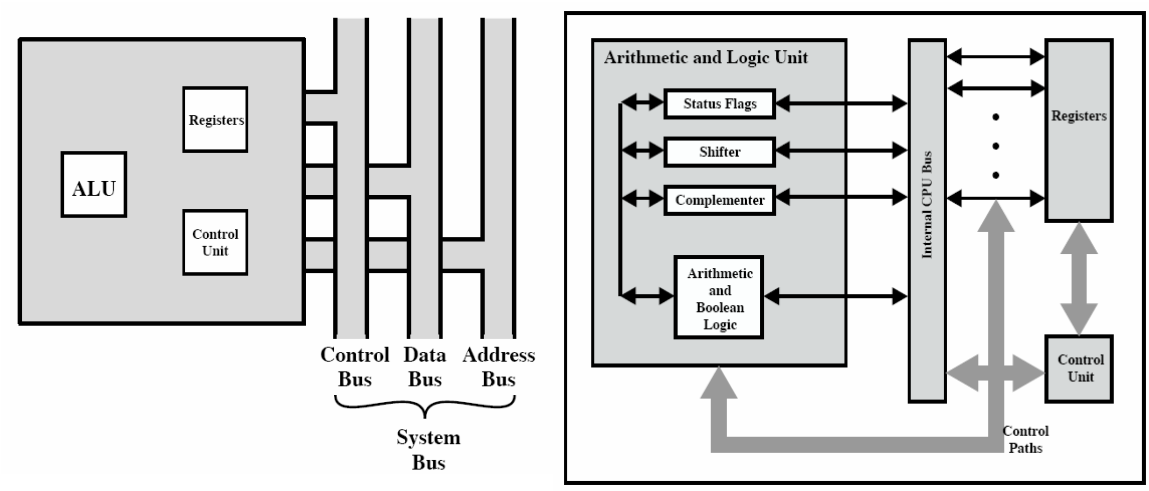
6.2 ส่วนประกอบของหน่วยประมวลผล

จากหน้าที่ที่กล่าวมาแล้วข้างต้น จะเป็นว่าซีพียูจะมีการติดต่อกับหน่วยความจำหลัก และอุปกรณ์ภายนอก โดยการขนถ่ายข้อมูลหรือคำสั่งระหว่างกัน ซึ่งสามารถทำได้โดยผ่านบัสระบบที่ประกอบไปด้วยบัสตำแหน่ง บัสข้อมูล และบัสควบคุม ดังแสดงในรูป 6.1a ในการประมวลคำสั่ง ซีพียูจำเป็นต้องมีหน่วยความจำที่จัดเก็บข้อมูลและคำสั่งไว้ชั่วคราว จำเป็นต้องมีส่วนที่ทำหน้าที่ในการควบคุม และทำหน้าประมวลผลคำสั่ง

ซีพียูจึงประกอบด้วย 3 ส่วนประกอบหลัก ได้แก่

- **หน่วยควบคุม** (Control Unit) ทำหน้าที่ควบคุมการทำงานของส่วนประกอบต่างๆ ให้ทำงานตามคำสั่ง รวมถึงการเคลื่อนย้ายข้อมูลและคำสั่งเข้าออกจากซีพียู
- **หน่วยประมวลผลคณิตศาสตร์และตรรกะ** (Arithmetic and Logic Unit – ALU) ทำหน้าที่คำนวณทางคณิตศาสตร์และเปรียบเทียบทางตรรกะ
- **รีจิสเตอร์** (Register) ทำหน้าที่เป็นเก็บคำสั่ง ข้อมูล และแอดเดรสไว้ชั่วคราว

โดยมีบัสภายในเพื่อใช้ในการเชื่อมต่อระหว่างกัน ดังแสดงในรูป 6.1b จะสังเกตได้ว่า โครงสร้างภายในคอมพิวเตอร์และโครงสร้างภายในซีพียูเองมีความคล้ายคลึงกัน คือ ประกอบด้วยส่วนประกอบขนาดเล็กต่างๆ ที่เชื่อมต่อกันด้วยบัส



(a) โครงสร้างซีพียูกับบัสระบบ (b) โครงสร้างภายในของซีพียู

รูปที่ 6.1 โครงสร้างของซีพียู

รีจิสเตอร์ภายในซีพียูอาจแบ่งได้เป็นหลายประเภท แต่ในที่นี้จะแบ่งออกเป็นสองกลุ่มหลักคือ

1. **รีจิสเตอร์ที่ผู้ใช้มองเห็นได้** (User visible register) โดยรีจิสเตอร์ในกลุ่มนี้จะอนุญาตให้ผู้เขียนโปรแกรมใช้คำสั่งในการอ้างอิงถึงและนำมาใช้งานได้ เพื่อลดการอ้างอิงถึงข้อมูลในหน่วยความจำ ซึ่งสามารถแบ่งออกเป็นประเภทย่อยๆ ได้ดังนี้
    - **รีจิสเตอร์สำหรับใช้งานทั่วไป** (General purpose register) ซึ่งสามารถนำไปใช้งานได้หลากหลาย ขึ้นอยู่คำสั่งที่ผู้เขียนโปรแกรมกำหนด อาจจะใช้ในการเก็บข้อมูล หรือใช้เก็บแอดเดรส ซึ่งอาจมีรีจิสเตอร์ในกลุ่มนี้บางตัวถูกกำหนดไว้ใช้งานเฉพาะอย่างได้
    - **รีจิสเตอร์สำหรับเก็บข้อมูล** (Data register) จะเรียกรีจิสเตอร์กลุ่มนี้ว่า Accumulator
    - **รีจิสเตอร์สำหรับเก็บแอดเดรส** (Address register) อาจเป็นรีจิสเตอร์สำหรับใช้งานทั่วไป หรือใช้ในการอ้างอิงเกี่ยวกับแอดเดรสโดยตรง เช่น รีจิสเตอร์เซกเมนต์
    - **รีจิสเตอร์สำหรับเก็บรหัสเงื่อนไข** (Condition code register) ใช้เก็บเงื่อนไขในการทำงานที่สอดคล้องกับผลลัพธ์ที่เกิดจากการประมวลผลของซีพียู กลุ่มบิตที่เก็บค่าเหล่านี้ในรีจิสเตอร์จะเรียกว่า Flags เช่น การคำนวณทางคณิตศาสตร์ อาจให้ผลลัพธ์ที่มีค่าเป็นบวก เป็นลบ เป็นศูนย์ หรือเกิดโอเวอร์โฟล์วได้ โดยค่า flag เหล่านี้จะถูกกำหนดค่าขึ้นตามผลลัพธ์ที่ประมวลได้ เพื่อใช้ในการทดสอบค่าตามเงื่อนไขที่ต้องการได้ โดยปกติรีจิสเตอร์กลุ่มนี้จะสามารถอ่านได้อย่างเดียว ผู้ใช้ไม่สามารถกำหนดค่าโดยการเขียนโปรแกรมได้ จัดเป็นรีจิสเตอร์สำหรับการควบคุมประเภทหนึ่ง แต่ในบางครั้งก็อนุญาตให้สามารถเปลี่ยนแปลงค่าได้ในกรณีที่มีการเรียกใช้โปรแกรมย่อย โดยจะสามารถบันทึกค่าของรีจิสเตอร์กลุ่มนี้ไว้ก่อน เมื่อทำการประมวลผลโปรแกรมย่อยเสร็จจึงนำค่าในรีจิสเตอร์เดิมมาบันทึกกลับคืน

2. **รีจิสเตอร์ที่ผู้ใช้มองไม่เห็น หรือรีจิสเตอร์สำหรับการควบคุม และรายงานสถานการณ์ทำงาน** (Control and status register) จะใช้โดยหน่วยควบคุม เพื่อควบคุมการทำงานของซีพียู หรืออาจมีการเรียกใช้งานโดยคำสั่งพิเศษของระบบปฏิบัติการ เพื่อควบคุมการประมวลผลโปรแกรม โดยส่วนใหญ่จะไม่อนุญาตให้ผู้ใช้มองเห็นรีจิสเตอร์ในกลุ่มนี้ อาจมีบางส่วนที่ยอมให้ใช้คำสั่งในการประมวลผลได้ ก็ต่อเมื่อเป็นคำสั่งของระบบปฏิบัติการ รีจิสเตอร์ในกลุ่มนี้ ส่วนใหญ่สามารถแบ่งได้เป็น 4 ประเภท ดังต่อไปนี้

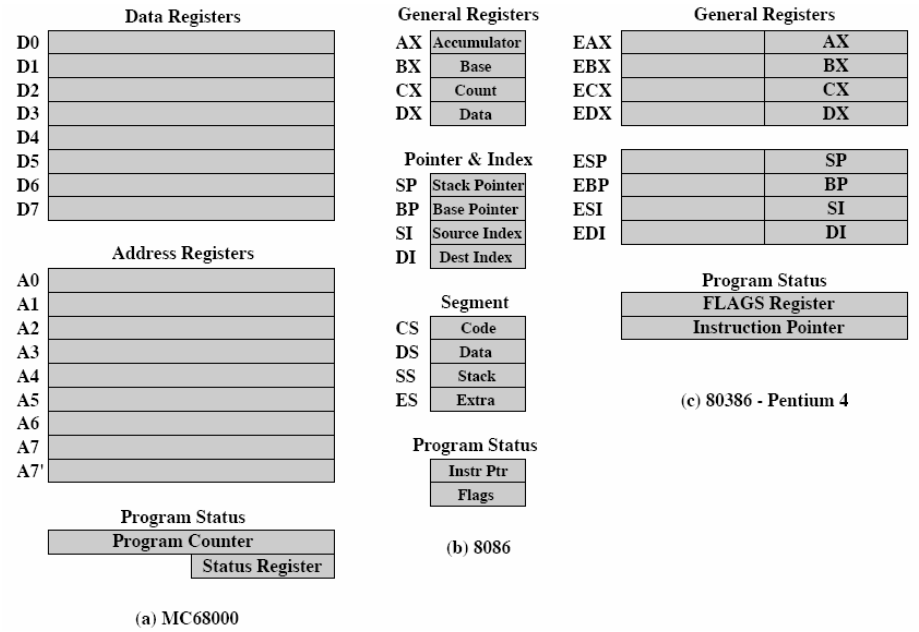
   - PC (Program counter) ใช้เก็บแอดเดรสของคำสั่งที่จะประมวลผลในลำดับถัดไป
   - IR (Instruction register) ใช้เก็บคำสั่งที่อ่านเข้ามาจากหน่วยความจำหลัก
   - MAR (Memory address register) ใช้เก็บแอดเดรสของหน่วยความจำหลัก
   - MBR (Memory buffer register) ใช้เก็บข้อมูลขนาด 1 เวิร์ดที่อ่านเข้ามาเก็บในซีพียู หรือ ข้อมูลที่ซีพียูต้องการบันทึกลงในหน่วยความจำหลัก

   โดย PC และ MAR จะมีการเชื่อมต่อโดยตรงเข้ากับแอดเดรสบัส ส่วน MBR จะเชื่อมต่อเข้ากับบัสข้อมูล ในขณะที่ IR อาจมีการแลกเปลี่ยนข้อมูลและคำสั่งกับหน่วยความจำหลักโดยผ่านทางรีจิสเตอร์ MAR และ MBR (ดูรูป 3.2 ในบทที่ 3) รีจิสเตอร์ส่วนที่ผู้ใช้มองเห็นได้จะสามารถขนถ่ายข้อมูลจาก MBR โดยอัตโนมัติ เพื่อนำไปใช้งานได้ รีจิสเตอร์ทั้ง 4 ตัวนี้ จะใช้ในการเคลื่อนย้ายข้อมูลระหว่างหน่วยความจำหลักกับซีพียู ในส่วนของ ALU ที่ทำหน้าประมวลผลกับข้อมูลก็จะต้องสามารถอ่านข้อมูลจากรีจิสเตอร์ MBR และรีจิสเตอร์สำหรับใช้งานทั่วไปได้โดยตรง หรืออาจมีรีจิสเตอร์ที่ทำหน้าที่เป็นบัฟเฟอร์คั่นระหว่าง ALU กับรีจิสเตอร์ต่างๆ อีกทีหนึ่ง

ซีพียูแต่ละรุ่นก็มีการจัดองค์ประกอบของรีจิสเตอร์ที่แตกต่างกัน ดังแสดงตัวอย่างในรูป 6.2 ในการออกแบบรีจิสเตอร์ภายในซีพียู มีข้อที่ควรคำนึงถึงหลายประการด้วยกัน ประเด็นสำคัญก็เช่น รีจิสเตอร์สำหรับใช้ทั่วไป ควรกำหนดหน้าที่การทำงานเฉพาะอย่างให้กับรีจิสเตอร์แต่ละตัว หรือควรจะเปิดให้ใช้งานทั่วไปทั้งหมดไม่เฉพาะเจาะจง การกำหนดหน้าที่การทำงานสำหรับรีจิสเตอร์จะมีผลทำให้คำสั่งมีขนาดเล็กลง ทำงานได้ไวขึ้นแต่ก็มีความยืดหยุ่นน้อย ในขณะที่ถ้าไม่มีการกำหนดหน้าที่ชัดเจน ก็จะต้องใช้จำนวนบิตในการอ้างอิงถึงรีจิสเตอร์เหล่านี้มากขึ้น ทำให้คำสั่งเครื่องมีขนาดใหญ่ขึ้นและมีความซับซ้อนมากขึ้น แต่ก็มีความยืดหยุ่นคล่องตัวในการทำงานมากขึ้นด้วย

ประเด็นถัดมา คือ จำนวนของรีจิสเตอร์สำหรับการใช้งาน ควรจะมีจำนวนเท่าไหร่จึงจะเหมาะสม เพราะจำนวนรีจิสเตอร์ก็มีผลต่อการออบแบบชุดคำสั่งด้วย ยิ่งจำนวนรีจิสเตอร์มากขึ้น ก็อาจมีผลให้จำนวนบิตที่ใช้สำหรับการอ้างอิงถึงในคำสั่งมากขึ้นด้วย ถ้ารีจิสเตอร์มีจำนวนน้อยก็ทำให้ต้องอ้างอิงถึงหน่วยความจำหลักบ่อยขึ้น แต่จำนวนรีจิสเตอร์ที่มากเกินไปก็ไม่ได้ช่วยลดความถี่ในการอ้างอิงถึงหน่วยความจำหลักให้น้อยลงเสมอไป จำนวนรีจิสเตอร์ที่เหมาะสมนั้นอยู่ที่ 8-32 ตัว

อีกประเด็นหนึ่ง คือ เรื่องขนาด (ความยาว) ของรีจิสเตอร์ รีจิสเตอร์นั้นควรจะต้องมีขนาดใหญ่พอที่จะอ้างถึงแอดเดรสที่มากที่สุดได้ และจะต้องมีขนาดใหญ่พอที่จะเก็บข้อมูลได้เกือบทุกชนิด โดยในบางเครื่องอาจจะยอมให้ใช้รีจิสเตอร์สองตัวที่อยู่ติดกัน เพื่อใช้ในการเก็บข้อมูลที่มีขนาดความยาวเป็นสองเท่าของขนาดข้อมูลปกติ เช่น ข้อมูล double int หรือ long int ในภาษาซี



รูป 6.2 ตัวอย่างการจัดโครงสร้างของรีจิสเตอร์ของไมโครโปรเซสเซอร์

การออกแบบซีพียูจำเป็นคำนึงถึงรีจิสเตอร์กลุ่มหนึ่งที่เรียกว่า Program Status Word (PSW) อีกด้วย รีจิสเตอร์ในกลุ่มนี้จะใช้เก็บข้อมูลสถานะการทำงานของซีพียู ดังต่อไปนี้

- Sign ใช้เก็บบิตเครื่องหมายของผลลัพธ์ที่เกิดจากการประมวลผลคำสั่งทางคณิตศาสตร์ครั้งล่าสุด
- Zero บิตนี้จะถูกกำหนดให้มีค่าเป็น 1 เมื่อผลลัพธ์ที่เกิดจากการประมวลผลคำสั่งทางคณิตศาสตร์ครั้งล่าสุดมีค่าเป็น 0
- Carry บิตนี้จะถูกกำหนดให้มีค่าเป็น 1 เมื่อผลลัพธ์ที่เกิดจากการประมวลผลคำสั่งทางคณิตศาสตร์ครั้งล่าสุด มีการยืมเลขจากหลักที่สูงกว่า หรือมีการทดเลขที่มีค่าเกินบิตสูงสุด
- Equal บิตนี้จะถูกกำหนดให้มีค่าเป็น 1 เมื่อผลลัพธ์ที่เกิดจากการเปรียบเทียบทางตรรกะมีค่าเท่ากัน
- Overflow ใช้เก็บบิตเพื่อบอกให้ทราบว่าผลลัพธ์ที่เกิดจากการประมวลผลคำสั่งทางคณิตศาสตร์ครั้งล่าสุดเกิดโอเวอร์โฟล์ว
- Interrupt enable/disable ใช้ในการสั่งให้อนุญาตหรือไม่อนุญาตให้ใช้อินเทอร์รัพท์
- Supervisor ใช้บอกสถานะการทำงานของซีพียูว่าขณะนั้นผู้ใช้เป็น user หรือ supervisor ทั้งนี้เพราะคำสั่งเครื่องบางคำสั่ง และเนื้อที่ในหน่วยความจำบางส่วนจะสงวนให้ใช้ได้เฉพาะเมื่อซีพียูอยู่ในสถานะ supervisor เท่านั้น

นอกจากนี้ การออกแบบซีพียูบางรุ่น อาจจะมีการใช้รีจิสเตอร์พิเศษจำนวนหนึ่งที่เกี่ยวข้องกับการควบคุมการทำงาน หรือการบอกสถานะการทำงานที่นอกเหนือไปจากรีจิสเตอร์ PSW เช่น รีจิสเตอร์ที่เก็บตัวชี้ตำแหน่ง (pointer) ที่ใช้ไปยังบล็อกข้อมูลในหน่วยความจำหลัก หรือรีจิสเตอร์สำหรับอินเทอร์รัพท์เวกเตอร์ การออกแบบรีจิสเตอร์สำหรับการควบคุมและรายงานสถานการณ์ทำงานยังมีองค์ประกอบที่สำคัญอีกหลายประการ องค์ประกอบหนึ่งที่สำคัญ ก็คือการสนับสนุนการทำงานของระบบปฏิบัติการ ดังนั้นผู้ออกแบบจึงควรเข้าใจถึง

หลักการทำงานของระบบปฏิบัติการที่จะนำมาใช้ด้วย การจัดวางข้อมูลสำหรับการควบคุมระหว่างรีจิสเตอร์และหน่วยความจำหลักก็เป็นเรื่องหนึ่งที่สำคัญ ผู้ออกแบบจะต้องตัดสินใจว่าจะใช้ข้อมูลสำหรับการควบคุมจำนวนมากน้อยเพียงใด เพื่อเก็บไว้ในรีจิสเตอร์ และเพื่อเก็บไว้ในหน่วยความจำหลัก เพื่อเกิดความสมดุลระหว่างค่าใช้จ่ายและความเร็วในการทำงาน

6.3 การทำงานของหน่วยประมวลผล

การทำงานเริ่มต้นจากหน่วยควบคุมอ่านคำสั่งจากหน่วยความจำมาเก็บไว้ในซีพียู แปลความหมายคำสั่ง แล้วจึงสั่งให้หน่วยประมวลผลทางคณิตศาสตร์และตรรกะประมวลผลตามต้องการ จากนั้นจึงบันทึกผลลัพธ์ที่ได้ไว้ในรีจิสเตอร์ เพื่อส่งต่อไปยังหน่วยความจำหลัก ซึ่งการทำงานของคำสั่งหนึ่งคำสั่งนี้ จะเป็นการเรียกใช้กลุ่มคำสั่งย่อยลงไปอีก ชุดคำสั่งย่อยนี้เรียกว่า ไมโครโปรแกรม (Micro program) การทำงานของไมโครโปรแกรมสำหรับคำสั่งหนึ่ง ก็คือ วงรอบการทำงานของคำสั่งหนึ่งวงรอบนั่นเอง ซึ่งได้อธิบายไว้แล้วในบทที่ 3 ดังนั้น ในบทนี้จะขอทบทวนและอธิบายเพิ่มเติมการทำงานของซีพียูในบางส่วน

วงรอบคำสั่ง

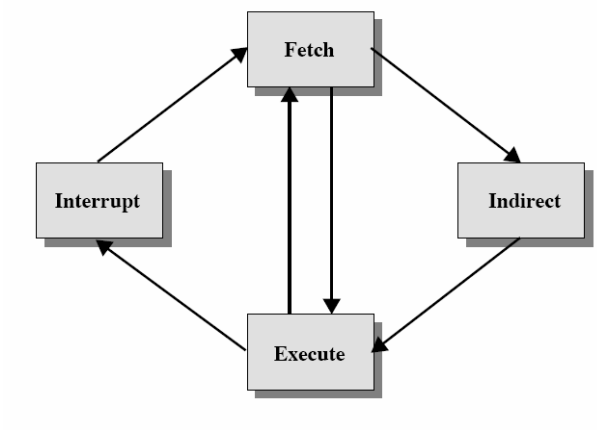
วงรอบของคำสั่งหนึ่งๆ จะประกอบด้วยการทำงานพื้นฐาน 2 ขั้นตอน คือ

1. การดึงคำสั่ง (Fetch) ทำการอ่านคำสั่งที่จะถูกประมวลผลลำดับถัดไปเข้ามาเก็บไว้ในซีพียู
2. การประมวลผล (Execute) ทำการแปลความหมายคำสั่ง และทำงานตามที่ต้องการ

ในวงรอบคำสั่ง (Instruction cycle) หนึ่ง นอกจากจะประกอบด้วยวงรอบคำสั่งพื้นอาจมีวงรอบย่อยเพิ่มเติมอีก ได้แก่

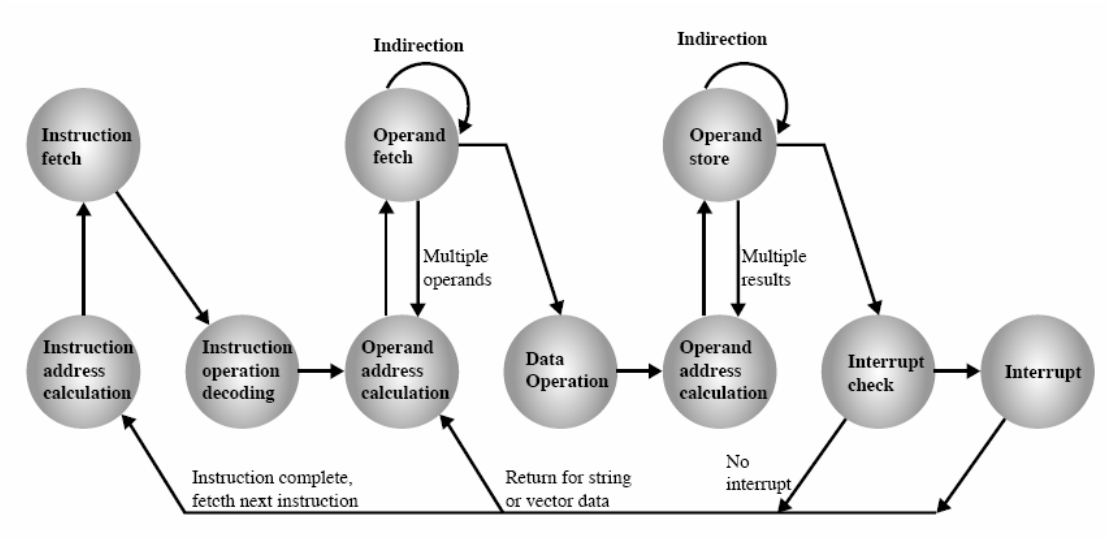
- วงรอบอินเทอร์รัพท์ (Interrupt cycle) กรณีที่อนุญาตให้ใช้อินเทอร์รัพท์ และมีอินเทอร์รัพท์เกิดขึ้น ซีพียูจะต้องบันทึกสถานการณ์ทำงานของโปรแกรมปัจจุบัน และให้บริการแก่อินเทอร์รัพท์ที่ร้องขอ
- วงรอบอินไดเร็ก (Indirect cycle) กรณีที่มีการอ้างอิงถึงตัวถูกกระทำ (operand) ที่เก็บไว้ในหน่วยความจำหลัก โดยเป็นการอ้างอิงแบบโดยอ้อม หรือที่เรียกว่า indirect addressing ซึ่งจะทำให้มีการอ่านข้อมูลในหน่วยความจำเพิ่มขึ้นอีกครั้งหนึ่งเสมอ

ดังนั้น ทั้งสองวงรอบจึงอาจถือเป็นขั้นตอนย่อยของวงรอบคำสั่ง ดังแสดงในรูป 6.3



รูปที่ 6.3 วงรอบคำสั่ง

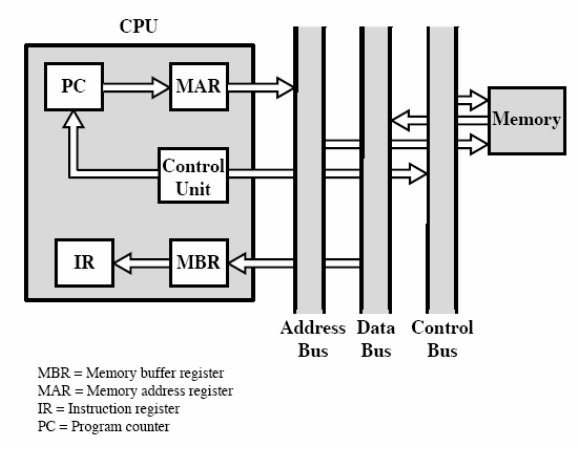
จากรูป การทำงานโดยปกติจะเป็นการทำงานสลับไปมาระหว่าวงรอบย่อย 2 วงรอบคือ วงรอบดึงคำสั่ง (Fetch) และวงรอบประมวลผล (Execute) เมื่อคำสั่งถูกอ่านเข้ามา จะถูกตรวจสอบว่ามีการอ้างอิงแบบ indirect addressing ในคำสั่งนั้นหรือไม่ ถ้ามี จะอ่านตัวถูกกระทำการที่อ้างอิงถึงเข้ามาในวงรอบย่อยอินไดเร็ก (Indirect) ภายหลังการทำงานของวงรอบประมวลผล อาจมีอินเทอร์รัพท์เกิดขึ้น ซีพียูจะตอบสนองโดยการทำงานตามวงรอบอินเทอร์รัพท์ (Interrupt) นั้น ก่อนกลับมาทำงานในวงรอบถึงคำสั่งต่อไป รูปที่ 6.4 แสดงขั้นตอนการทำงานอย่างละเอียดของวงรอบคำสั่งที่ประกอบด้วยวงรอบย่อยต่างๆ จะเห็นว่าถ้ามีตัวถูกกระทำการมีการอ้างอิงถึงแบบ indirect addressing แล้ว จะต้องมีการทำ operand fetch อีกครั้งหนึ่ง เช่นเดียวกับขั้นตอนการทำ operand store หลังจากนั้นจะทำการตรวจสอบอินเทอร์รัพท์เสียก่อน ถ้าพบว่ามีก็จะกระทำการในส่วนของวงรอบอินเทอร์รัพท์ แล้วจึงเริ่มต้นกระบวนการวนรอบต่อไป



รูปที่ 6.4 ไดอะแกรมแสดงสถานะการทำงานของวงรอบคำสั่ง

การไหลเวียนของข้อมูล (Data flow)

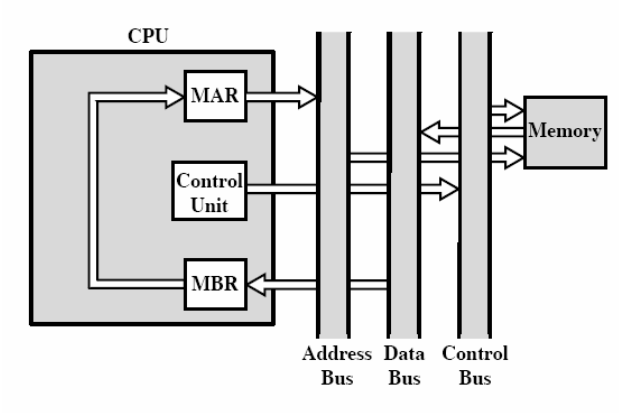
ลำดับการทำงานที่แท้จริงภายในวงรอบคำสั่งจะขึ้นอยู่กับการออกซีพียู แต่เราสามารถแสดงให้เห็นถึงการทำงานทั่วไปที่ต้องเกิดขึ้นได้ โดยสมมุติว่าซีพียูประกอบด้วยรีจิสเตอร์ MAR, MBR, PC และ IR



รูปที่ 6.5 การไหลเวียนของข้อมูลภายในวงรอบดึงคำสั่ง

รูป 6.5 แสดงให้เห็นถึงการไหลเวียนของข้อมูลที่เกิดขึ้นภายในวงรอบดึงข้อมูล เริ่มต้นจาก คำสั่งจะถูกอ่านเข้ามาจากหน่วยความจำ PC จะเก็บแอดเดรสของคำสั่งถัดไปที่จะถูกดึงเข้ามา แอดเดรสใน PC นี้จะถูกส่งไปให้ MAR แล้วส่งต่อไปยังบัสแอดเดรส หน่วยควบคุมจะสั่งให้ทำการอ่านข้อมูลจากหน่วยความจำหลัก ข้อมูลที่ได้จากการอ่านจะถูกส่งผ่านไปยังบัสข้อมูล จากนั้น MBR จะอ่านข้อมูลมาเก็บไว้ แล้วส่งต่อไปยัง IR อีกทอดหนึ่ง ในขณะเดียวกัน PC ก็จะถูกเพิ่มค่าขึ้นอีก 1 เพื่อเตรียมดึงคำสั่งถัดไปเข้ามา

เมื่อวงรอบดึงคำสั่งสิ้นสุด หน่วยควบคุมจะตรวจสอบข้อมูลที่เก็บใน IR ว่ามีตัวถูกกระทำที่ใช้ indirect addressing หรือไม่ ถ้ามีก็จะทำให้เกิดวงรอบอินไดเร็ก โดยมีทิศทางการไหลเวียนของข้อมูลดังแสดงในรูป 6.6

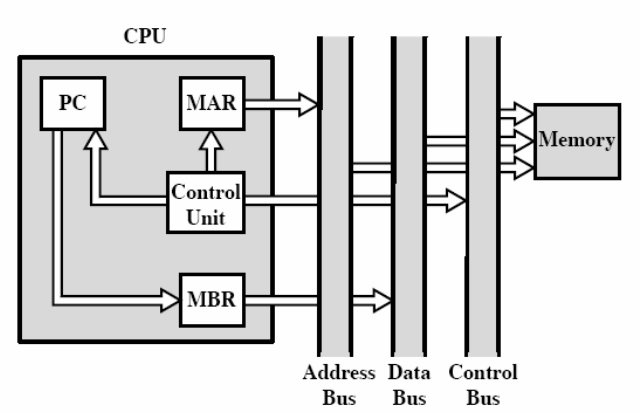


รูปที่ 6.6 การไหลเวียนของข้อมูลภายในวงรอบอินไดเร็ก

เริ่มต้นจากข้อมูลจำนวน N บิตที่อยู่ทางขวาสุดของ MBR ซึ่งเก็บแอดเดรสที่ถูกอ้างอิง ส่งไปยัง MAR จากนั้นหน่วยควบคุมจะสั่งให้อ่านข้อมูลในหน่วยความจำหลัก เพื่อให้ได้แอดเดรสของตัวถูกกระทำที่ต้องการนำไปเก็บไว้ใน MBR

วงรอบดึงข้อมูลและวงรอบอินไดเร็กนี้เป็นการทำงานอย่างง่ายและสามารถคาดเดาได้ ส่วนวงรอบการประมวลผลนั้นสามารถเกิดขึ้นได้หลายรูปแบบ ซึ่งขึ้นอยู่กับคำสั่งเครื่องที่เก็บไว้ใน IR วงรอบนี้อาจเกี่ยวข้องกับการขนถ่ายข้อมูลระหว่างรีจิสเตอร์ การอ่านและบันทึกข้อมูลลงในหน่วยความจำหรืออุปกรณ์รับและแสดงผล และอาจมีการเรียกใช้หน่วย ALU

วงรอบอินเทอร์รัพท์ก็คล้ายกับวงรอบดึงข้อมูลและวงรอบอินไดเร็ก กล่าวคือ เป็นการทำงานอย่างง่ายและคาดเดาได้เช่นกัน โดยมีทิศทางการไหลเวียนของข้อมูลดังแสดงในรูป 6.7



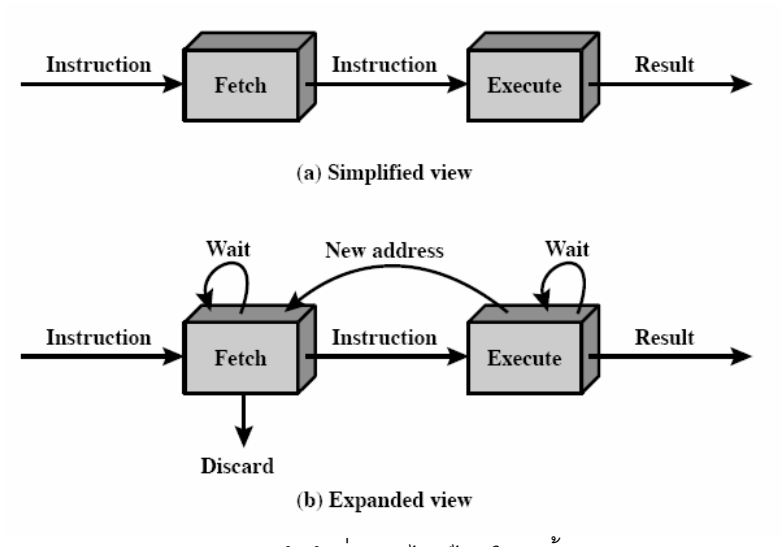
รูปที่ 6.7 การไหลเวียนของข้อมูลภายในวงรอบอินเทอร์รัพท์

เริ่มต้นจากการนำข้อมูลใน PC ไปบันทึกเก็บไว้ในหน่วยความจำ เพื่อที่จะเรียกค่านี้คืนกลับมาใช้ภายหลังทำวงรอบอินเทอร์รัพท์เสร็จสิ้น ดังนั้นค่าใน PC จะถูกส่งไปยัง MBR แล้วถูกส่งเข้าไปยังบัสข้อมูล เพื่อทำการบันทึกลงในหน่วยความจำ ซึ่งหน่วยควบคุมจะเป็นตัวกำหนดตำแหน่งที่จะบันทึกส่งไปให้ยัง MAR จากนั้น จะนำแอดเดรสของโปรแกรมอินเทอร์รัพท์ไปเก็บไว้ใน PC เพื่อเริ่มต้นวงรอบดึงข้อมูลต่อไป

เทคนิคการทำไปป์ไลน์

การเพิ่มประสิทธิภาพของคอมพิวเตอร์สามรถทำได้หลายรูปแบบ เช่น การออกแบบแผงวงจรให้มีความเร็วในการทำงานเพิ่มขึ้น หรืออาจจะเป็นการจัดโครงสร้างภายในซีพียูให้ดีขึ้น เช่น การออกแบบให้มีรีจิสเตอร์หลายตัว การใช้หน่วยความจำแคช อีกวิธีหนึ่งก็คือ เทคนิคการทำคำสั่งแบบไปป์ไลน์ (instruction pipelining)

การทำคำสั่งแบบไปป์ไลน์ เป็นเทคนิคที่คล้ายกับกระบวนการผลิตสินค้าในโรงงานอุตสาหกรรม ซึ่งแบ่งการประกอบสินค้าออกเป็นขั้นตอนต่างๆ โดยการทำงานของขั้นตอนทั้งหมดสามารถทำพร้อมกันได้ (ยกเว้นตอนเริ่มต้นการทำงาน) เมื่อนำมาประยุกต์ใช้กับการประมวลผลคำสั่ง ก็คือการแบ่งการประมวลผลออกเป็นขั้นตอนย่อยๆ ทำให้สามารถนำเทคนิคการทำไปป์ไลน์มาใช้ได้ ตัวอย่างเช่น ถ้าเรามีการแบ่งการประมวลผลออกเป็น 2 ขั้นตอน คือ ขั้นตอนการดึงคำสั่ง และขั้นตอนการประมวลผล ในขณะที่ทำขั้นตอนการประมวลผลอยู่ จะไม่มีการอ้างอิงถึงหน่วยความจำหลัก จึงสามารถใช้ช่วงเวลานี้ในการดึงคำสั่งลำดับถัดไปเข้ามาเตรียมพร้อมไว้ได้ ดังแสดงในรูปที่ 6.8a จะเห็นว่าการทำงานทั้งสองขั้นตอนเกิดขึ้นพร้อมกัน ไปป์ไลน์ประกอบด้วยขั้นตอนที่เป็นอิสระจากกัน ขั้นตอนแรกจะเป็นการดึงคำสั่งมาเก็บไว้ในบัฟเฟอร์ เมื่อขั้นตอนที่ 2 ว่างก็จะขนถ่ายคำสั่งจากบัฟเฟอร์ไปให้ ในขณะที่ขั้นตอนที่ 2 กำลังประมวลผลคำสั่งอยู่ ขั้นตอนแรกก็สามารถใช้ประโยชน์จากการอ้างอิงหน่วยความจำหลัก โดยดึงคำสั่งถัดไปมาเก็บไว้ในบัฟเฟอร์ การทำงานแบบนี้เรียกว่า instruction prefetch หรือ fetch overlap



รูป 6.8 การทำคำสั่งแบบไปป์ไลน์ใน 2 ขั้นตอน

การทำงานในลักษณะนี้จะช่วยให้ซีพียูสามารถประมวลผลได้เร็วขึ้น ถ้าขั้นตอนดึงคำสั่งและขั้นตอนประมวลผลใช้เวลาเท่ากัน เวลาที่ใช้ในการประมวลผลทั้งหมดก็จะลดลงเหลือครึ่งเดียวเท่านั้น แต่ถ้าพิจารณา

อย่างละเอียด ดังแสดงในรูป 6.8b แล้วจะพบว่าระยะเวลาการทำงานไม่สามารถลดลงครึ่งหนึ่งได้ เนื่องจากเหตุผล 2 ประการ คือ

1. เวลาในการประมวลผล มักจะใช้เวลานานกว่าเวลาที่ใช้ในการดึงคำสั่งมาจากหน่วยความจำหลัก ดังนั้น ขั้นตอนการดึงคำสั่งเข้ามาจึงต้องรอเป็นระยะเวลาหนึ่งก่อนจะส่งข้อมูลไปให้ขั้นตอนการประมวลผลได้ และทำงานต่อไปได้
2. คำสั่งที่เกี่ยวกับทางเลือกแบบมีเงื่อนไข (conditional branch) ทำให้ไม่ทราบตำแหน่งที่แน่นอนของคำสั่งที่จะถูกประมวลในลำดับถัดไป ดังนั้นขั้นตอนดึงคำสั่งอาจจำเป็นต้องรอจนกว่าจะทราบตำแหน่งที่แน่นอนของคำสั่งถัดไปจากขั้นตอนการประมวลผล ทำให้ขั้นตอนการประมวลผลเองก็ต้องรอจนกว่าคำสั่งถัดไปจะถูกดึงเข้ามา

แม้ว่าเหตุผลทั้งสองประการนี้จะทำให้ประสิทธิภาพของการทำคำสั่งแบบไปน์ไลน์ลดลง แต่ก็ยังคงมีประสิทธิภาพสูงกว่าการทำงานแบบไม่ใช้ไปป์ไลน์

การใช้เทคนิคการเดาเป็นวิธีที่อาจช่วยทำให้ลดข้อจำกัดในเหตุผลข้อ 2 ได้ โดยใช้หลักการง่ายๆ คือ เมื่อขั้นตอนการดึงคำสั่งส่งคำสั่ง branch ถูกส่งไปยังขั้นตอนการประมวลผล ขั้นตอนการดึงข้อมูลก็ทำการดึงคำสั่งถัดไปขึ้นมาจากหน่วยความจำตามปกติ ถ้าการ branch ไม่เกิดขึ้น ก็ทำให้ไม่ต้องเสียเวลารอ แต่ถ้าเกิดคำสั่งที่อยู่ดึงขึ้นมาแล้วก็จะถูกลบทิ้ง แล้วดึงคำสั่งที่ต้องการมาแทน

การแบ่งขั้นตอนให้มีจำนวนมากขึ้น ก็เป็นอีกวิธีที่ช่วยทำให้เพิ่มประสิทธิภาพได้เช่นกัน ตัวอย่างเช่น การแบ่งขั้นตอนการประมวลผลออกเป็น 6 ขั้นตอนดังนี้

- Fetch instruction (IF) เป็นการอ่านคำสั่งลำดับถัดไปเข้ามาเก็บไว้ในบัฟเฟอร์
- Decode instruction (DI) เป็นการแปลความหมายคำสั่ง
- Calculate operands (CO) เป็นการคำนวณแอดเดรสที่แท้จริง (effective address) สำหรับตัวถูกกระทำแต่ละตัว
- Fetch operands (FO) ทำการดึงค่าตัวถูกกระทำแต่ละตัวจากหน่วยความจำหลักเข้ามาในซีพียู ถ้าตัวถูกกระทำเก็บอยู่ในรีจิสเตอร์แล้ว ไม่จำเป็นต้องอ่านเข้ามาใหม่
- Execute instruction (EI) ทำการประมวลผลคำสั่ง และอาจมีการบันทึกผลลัพธ์ไว้ในแอดเดรสของตัวถูกกระทำที่หนดไว้ในคำสั่งนั้นๆ
- Write operand (WO) ทำการบันทึกผลลัพธ์ไว้ในหน่วยความจำหลัก

การแบ่งขั้นตอนการทำงานแบบนี้ ทำให้แต่ละขั้นตอนใช้เวลาทำงานใกล้เคียงกัน รูป 6.9 แสดงแผนผังของเวลาที่ใช้ในการทำคำสั่งแบบไปป์ไลน์ โดยสมมุติให้แต่ละขั้นตอนใช้เวลาทำงานเท่ากัน สามารถทำงานได้ในเวลาเดียวกัน จะเห็นว่า เมื่อแบ่งการทำงานเป็น 6 ขั้นตอน สามารถลดเวลาการประมวลคำสั่ง 9 คำสั่งจาก 54 หน่วยเวลาเหลือ 14 หน่วยเวลาได้

แต่หากมีการ branch อย่างมีเงื่อนไข หรือเกิดอินเทอร์รัพท์ก็จะทำให้ลดประสิทธิภาพในการทำงานลงเช่นกัน รูป 6.10 แสดงแผนผังของเวลาที่ใช้การทำคำสั่งแบบไปป์ไลน์ เมื่อเกิดการ branch ที่คำสั่งที่ 3 ไปยังคำสั่งที่ 15 ซึ่งจะไม่มีทราบจนกว่าจะประมวลผลคำสั่งที่ 3 เสร็จสิ้น จากรูปจะเห็นได้มีการดึงคำสั่งเข้าไปในไปป์ไลน์จนถึงคำสั่งที่ 7 แล้ว คำสั่งทั้งหมดในไปป์ไลน์นี้จะต้องถูกลบทิ้ง จากนั้นจึงเริ่มดึงคำสั่งที่ 15 เข้ามาไปใน

หน่วยเวลาที่ 8 ในช่วงเวลาที่ 9-12 จึงไม่มีคำสั่งใดถูกประมวลผลจนเสร็จสิ้นเลย ถือเป็นช่วงเวลาที่เสียเปล่า (penalty)

Time →

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Instruction 1	FI	DI	CO	FO	EI	WO								
Instruction 2		FI	DI	CO	FO	EI	WO							
Instruction 3			FI	DI	CO	FO	EI	WO						
Instruction 4				FI	DI	CO	FO	EI	WO					
Instruction 5					FI	DI	CO	FO	EI	WO				
Instruction 6						FI	DI	CO	FO	EI	WO			
Instruction 7							FI	DI	CO	FO	EI	WO		
Instruction 8								FI	DI	CO	FO	EI	WO	
Instruction 9									FI	DI	CO	FO	EI	WO

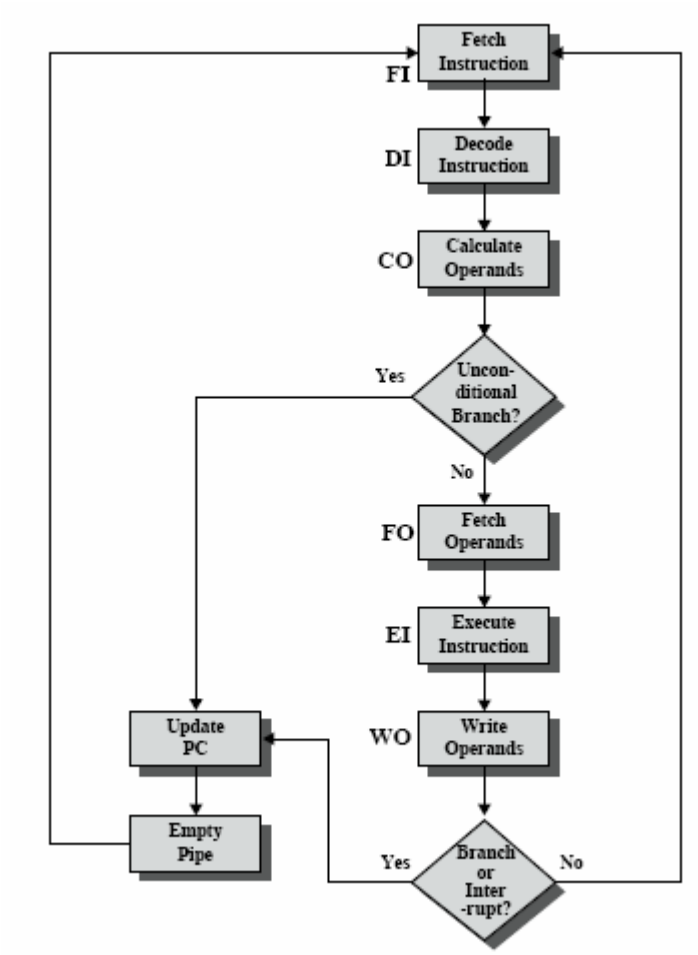
รูป 6.9 แผนผังแสดงเวลาการทำคำสั่งแบบไปป์ไลน์ 6 ขั้นตอน

รูปที่ 6.11 แสดงผังงานการทำงานของไปป์ไลน์แบบ 6 ขั้นตอนที่มีการ branch หรืออินเทอร์รัพท์เกิดขึ้นได้ การทำคำสั่งแบบไปป์ไลน์ 6 ขั้นตอนสามารถแสดงให้เห็นในมุมมองของลำดับขั้นตอนที่เกิดขึ้นได้ ดังแสดงในรูป 6.12 จะเห็นว่า แนวตั้งจะเป็นเวลาที่ผ่านไปจากจุดเริ่มต้นที่ตำแหน่งบนสุด แต่ละแถวแสดงแต่ละขั้นตอนของไปป์ไลน์ที่เกิดขึ้น ณ เวลาหนึ่งๆ ซึ่งสอดคล้องกับรูป 6.7 และ 6.8

Time → Branch Penalty ↔

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Instruction 1	FI	DI	CO	FO	EI	WO								
Instruction 2		FI	DI	CO	FO	EI	WO							
Instruction 3			FI	DI	CO	FO	EI	WO						
Instruction 4				FI	DI	CO	FO							
Instruction 5					FI	DI	CO							
Instruction 6						FI	DI							
Instruction 7							FI							
Instruction 15								FI	DI	CO	FO	EI	WO	
Instruction 16									FI	DI	CO	FO	EI	WO

รูปที่ 6.10 แผนผังแสดงเวลาการทำคำสั่งแบบไปป์ไลน์โดยมีผลกระทบที่เกิดจากการทำ conditional branch



รูปที่ 6.11 การทำคำสั่งแบบไปป์ไลน์ใน 6 ขั้นตอน

Time

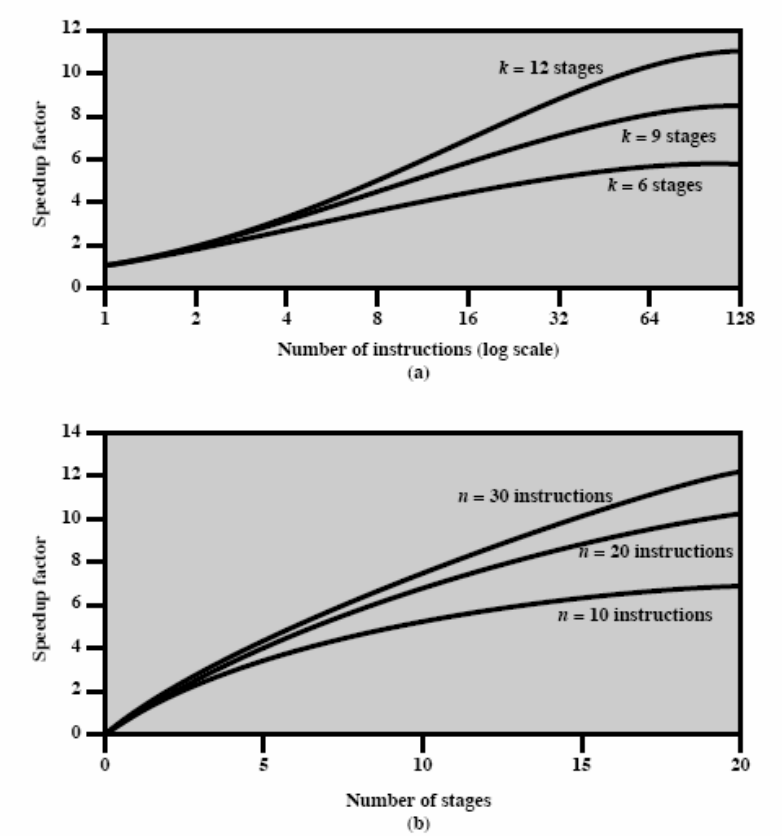
	FI	DI	CO	FO	EI	WO
1	I1					
2	I2	I1				
3	I3	I2	I1			
4	I4	I3	I2	I1		
5	I5	I4	I3	I2	I1	
6	I6	I5	I4	I3	I2	I1
7	I7	I6	I5	I4	I3	I2
8	I8	I7	I6	I5	I4	I3
9	I9	I8	I7	I6	I5	I4
10		I9	I8	I7	I6	I5
11			I9	I8	I7	I6
12				I9	I8	I7
13					I9	I8
14						I9

(a) No branches

	FI	DI	CO	FO	EI	WO
1	I1					
2	I2	I1				
3	I3	I2	I1			
4	I4	I3	I2	I1		
5	I5	I4	I3	I2	I1	
6	I6	I5	I4	I3	I2	I1
7	I7	I6	I5	I4	I3	I2
8	I15					I3
9	I16	I15				
10		I16	I15			
11			I16	I15		
12				I16	I15	
13					I16	I15
14						I16

(b) With conditional branch

รูป 6.12 แผนผังเวลาการทำคำสั่งแบบไปป์ไลน์ 6 ขั้นตอนในอีกรูปแบบหนึ่ง



รูปที่ 6.13 กราฟแสดงปัจจัยการเพิ่มความเร็วด้วยเทคนิคการทำไปป์ไลน์

รูป 6.13 แสดงให้เห็นถึงปัจจัยที่มีผลทำให้ความเร็วของการทำไปป์ไลน์เพิ่มขึ้น เมื่อจำนวนขั้นตอนทั้งหมดในไปป์ไลน์เพิ่มขึ้น ความเร็วในการทำงานก็จะยิ่งเพิ่มมากขึ้นด้วย ซึ่งจะเห็นได้ชัดเจนเมื่อจำนวนคำสั่งที่ถูกประมวลผลเพิ่มมากขึ้น แต่ในทางปฏิบัติ จำนวนขั้นตอนที่เพิ่มขึ้นนี้จะทำให้เกิดความซับซ้อนมากขึ้น ระยะเวลาหน่วงระหว่างขั้นตอนต่างๆ ในไปป์ไลน์เพิ่มขึ้น รวมทั้งมูลค่าต้นทุนของซีพียูอาจสูงขึ้นด้วย

6.4 เอกสารอ้างอิงและเว็บไซต์ที่ควรรู้

Chapter 12 William Stalling. Computer organization and architecture: designing for performance.

http://www.ackadia.com/computer/system-architecture/system-architecture-cpu.php

http://www3.ipst.ac.th/research/assets/web/mahidol/computer(10)/system/processor.htm

http://staff1.kmutt.ac.th/~sumet/combk/ch4/c421.htm

http://yalor.yru.ac.th/~nipon/Archi_html/4122701/e-learn.htm

http://homepage.cs.uri.edu/faculty/wolfe/book/Readings/Reading04.htm

http://www.hardwaresecrets.com/article/209