สถาปัตยกรรมคอมพิวเตอร์พื้นฐาน (Intro to Computer Architecture)

Chapter 1

Computer Organization and Architecture

ปรับปรุงจากเอกสารของ
อ. ณัฐวุฒิ สร้อยดอกสน (NSD)

อ.เอิญ สุริยะฉาย (ENS)
ภาควิชาวิทยาการคอมพิวเตอร์และสารสนเทศ
มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าพระนครเหนือ

สถาปัตยกรรมคอมพิวเตอร์

สถาปัตยกรรมคอมพิวเตอร์ (Computer Architecture)

- หมายถึงโครงสร้างของระบบคอมพิวเตอร์ที่โปรแกรมเมอร์ของระบบจะต้องเข้าใจในภาษาเครื่องเพื่อเขียนโปรแกรมให้เครื่องทำงานได้อย่างถูกต้อง ประกอบไปด้วย
  - รีจิสเตอร์
  - หน่วยความจำ
  - ชุดคำสั่ง (Instruction Set) ฯลฯ

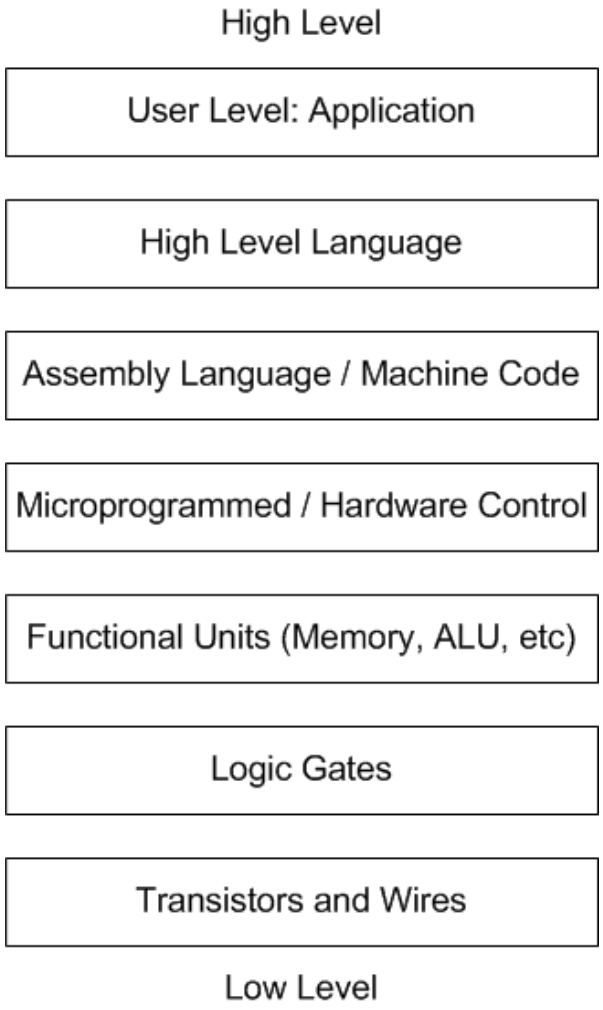
สถาปัตยกรรมคอมพิวเตอร์

สถาปัตยกรรมชุดคำสั่ง (Instruction Set Architecture: ISA)

- ถูกนำมาใช้ในการอธิบายถึงโครงสร้างของคอมพิวเตอร์ที่โปรแกรมเมอร์มองเห็น
  - ตระกูลของโปรเซสเซอร์ที่สามารถรันไบนารีโค้ดเดียวกันถือเป็นสถาปัตยกรรมตัวเดียวกัน
  - แต่บางครั้งโปรเซสเซอร์คนละตระกูลสามารถที่จะรันคำสั่งเดียวกันได้ เช่น IBM, Intel และ SUN สามารถเอ็กซิคิวต์คำสั่งที่มาจากสถาปัตยกรรมที่หลากหลายได้

โครงสร้างของคอมพิวเตอร์

- แบ่งออกได้เป็น 7 ระดับ (Level)



โครงสร้างของคอมพิวเตอร์

- **ระดับผู้ใช้งานหรือแอปพลิเคชัน**
  (User or Application Program Level)
  - เป็นระดับที่ผู้ใช้งานสามารถโต้ตอบกับคอมพิวเตอร์ผ่านทางโปรแกรมต่างๆ
- **ระดับภาษาระดับสูง**
  (High-Level Language Level)
  - เป็นระดับของผู้พัฒนาโปรแกรม (Programmer) โดยใช้ตัวแปลภาษา (**Compiler**) ที่ทำหน้าที่แปลจากภาษาระดับสูงเป็นภาษาเครื่อง (Machine Code) เพื่อนำไปใช้กับคอมพิวเตอร์
  - เช่น ภาษาซี ปาสคาล หรือจาวา

โครงสร้างของคอมพิวเตอร์

- **ระดับภาษาแอสเซ็มบลีหรือภาษาเครื่อง** (Assembly Language or Machine Code Level)
  - โดยภาษาเครื่องจะอยู่ในรูปแบบของ Binary Code เช่น 111001 แทนการบวก เป็นรูปแบบที่จดจำได้ยาก
  - ต่อมามีผู้พัฒนาภาษา **Assembly** โดยมีตัวแปลภาษาคือ “แอสเซ็มเบลอร์” (Assembler) เพื่อให้ใช้งานได้สะดวกยิ่งขึ้น
  - ตัวอย่างของภาษาเช่น

ภาษาแอสเซ็มบลี	ภาษาเครื่อง	ความหมาย
ADD	00111001	การบวก
MOVE	00010110	การย้ายค่า

    - หมายเหตุ ระดับของภาษาเครื่องจะสัมพันธ์กับการออกแบบทางด้านฮาร์แวร์ เช่นรีจิสเตอร์ ที่ถูกใช้ในการโอนถ่ายข้อมูล โดยชุดคำสั่งของเครื่องจะเรียกว่า Instruction Set

โครงสร้างของคอมพิวเตอร์

- **ระดับควบคุม**
  (Control Level)
  - เป็นระดับของสัญญาณควบคุมที่มีผลต่อการถ่ายโอนข้อมูลระหว่างรีจิสเตอร์ เป็นลักษณะของคำสั่งในแบบของไมโครโปรแกรม
  - **ไมโครโปรแกรม** (Microprogram) เป็นภาษาระดับต่ำเพื่อควบคุมฮาร์ดแวร์
- **ระดับหน่วยฟังก์ชัน**
  (Functional Unit Level)
  - เป็นการถ่ายโอนข้อมูลระหว่างรีจิสเตอร์ที่หน่วยควบคุมย้ายเข้าออกจากหน่วยฟังก์ชัน เช่น การบวก ลบ คูณ หาร รวมถึงรีจิสเตอร์ภายในซีพียู ALU และหน่วยความจำของเครื่อง

โครงสร้างของคอมพิวเตอร์

- **ระดับลอจิกเกตและทรานซิสเตอร์**
  (Logic gate and Transistor Level)
  - เป็นระดับในส่วนของ**ฮาร์ดแวร์** ของเครื่องที่อาศัยสัญญาณนาฬิกา กระแสไฟฟ้า ความต้านทาน ฯลฯ

ความเข้ากันได้ (Compatibility)

- โครงสร้างของระบบคอมพิวเตอร์โมเดลใหม่หรือคอมพิวเตอร์รุ่นใหม่ๆ จะมีสถาปัตยกรรมใหม่ที่ซอฟต์แวร์เดิม ไม่สามารถรันได้
- จึงเกิดปัญหาความคอมแพติเบิลของฮาร์ดแวร์และซอฟต์แวร์ ทำให้ผู้ใช้งานต้องชลอการสั่งซื้อคอมพิวเตอร์รุ่นใหม่ๆ เนื่องจากต้องรอซอฟต์แวร์รุ่นใหม่ๆ ออกมาเพื่อให้สามารถใช้งานได้กับคอมพิวเตอร์รุ่นใหม่ๆ

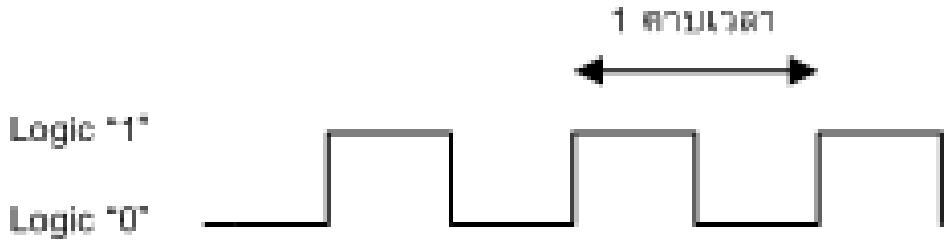
สถาปัตยกรรมคอมพิวเตอร์

สถาปัตยกรรมคอมพิวเตอร์ (Computer Architecture)

- ระบบคอมพิวเตอร์โดยทั่วไปจะมีโปรเซสเซอร์เพียงตัวเดียว หรือที่เรียกกันว่าซีพียู (Central Processor Unit)
- ระบบที่มีประสิทธิภาพสูงในสมัยใหม่จะมีโปรเซสเซอร์มากกว่า 1 ตัว เช่น
  - โปรเซสเซอร์แบบเวกเตอร์ (Vector Processor)
  - โปรเซสเซอร์แบบขนาน (Parallel Processor)
- ระบบที่มีโปรเซสเซอร์เพียงตัวเดียวจะเป็นโปรเซสเซอร์แบบอนุกรมหรือที่เรียกว่าโปรเซสเซอร์แบบสเกลาร์ (Scalar Processor)

การวัดความเร็วของคอมพิวเตอร์

- สัญญาณนาฬิกาในระบบคอมพิวเตอร์ถูกใช้เป็นตัวควบคุมการทำงานของอุปกรณ์ต่างๆในระบบ โดยใช้คริสตัลขนาดเล็กเป็นตัวควบคุมสัญญาณนาฬิกาของระบบ
  - หน่วยที่ใช้วัดสัญญาณนาฬิกาจะใช้หน่วย **“เฮิรตซ์” (Hertz)**
  - สัญญาณนาฬิกาจำนวน 1 ลูกต่อวินาทีหรือมีคาบเวลาเท่ากับ 1 วินาที
  - ถ้า 1 GHz หรือ 1 กิกะเฮิรตซ์หมายความว่ามีสัญญาณนาฬิกาจำนวนพันล้านลูกต่อวินาที เพราะฉะนั้นสัญญาณนาฬิกา 1 ลูกจะมีคาบเวลาเพียง 1 นาโนวินาที



การวัดความเร็วของคอมพิวเตอร์

- การคำนวณค่าความเร็วที่เพิ่มขึ้น (**Speedup**) ของการประมวลผลสามารถหาได้จากอัตราส่วนระหว่างเวลาในการทำคำสั่งของซีพียูรุ่นเก่ากับเวลาในการทำคำสั่งของซีพียูรุ่นใหม่
  - เช่น ถ้าหากซีพียูตัวเก่าทำงานโปรแกรมหนึ่งใช้เวลา 30 วินาที ซีพียูตัวใหม่ทำงานกับโปรแกรมเดิมใช้เวลา 20 วินาทีสามารถหาได้จากสูตรดังนี้

$$Speedup = \frac{Execution\ Time_{old}}{Execution\ Time_{new}} = \frac{30}{20} = 1.50$$

การวัดความเร็วของคอมพิวเตอร์

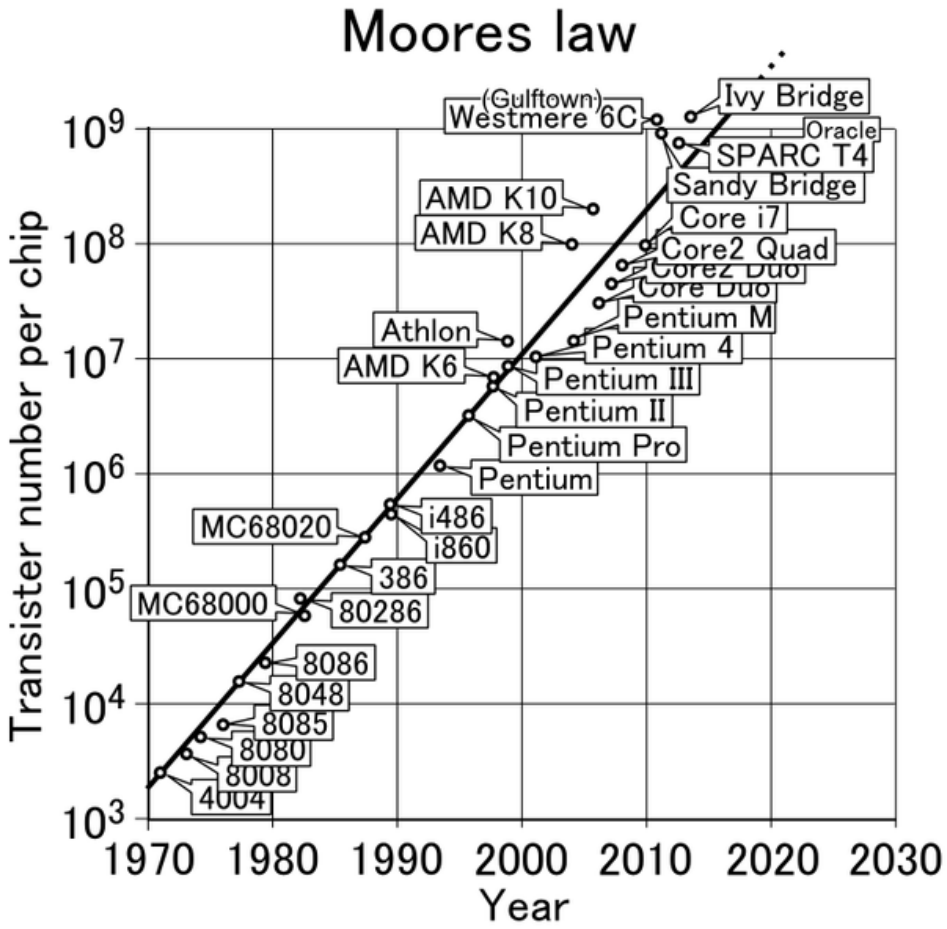
- ถ้าเปรียบเทียบการรันระหว่างซีพียูรุ่นเก่ากับซีพียูรุ่นใหม่ โดยซีพียูรุ่นเก่ารันโปรแกรมได้ 35 ครั้งใน 1 ชั่วโมงแต่ซีพียูรุ่นใหม่รันโปรแกรมเดียวกันได้ 48 ครั้งใน 1 ชั่วโมงเพราะฉะนั้นสามารถหาความเร็วที่เพิ่มขึ้นได้จาก

$$Speedup = \frac{S_{new}}{S_{old}} = \frac{48}{35} = 1.37$$

  - หรือมีความเร็วเพิ่มขึ้น โดยพิจารณาจากตัวเลขหลังทศนิยมหรือคิดเป็นเปอร์เซ็นต์ได้
  - แต่วิธีการนี้จะใช้ได้ต่อเมื่อซีพียูมีสถาปัตยกรรมภายในเหมือนกัน

Moore's law

- จำนวนทรานซิสเตอร์จะเพิ่ม เท่าตัวประมาณทุก ๆ 2 ปี



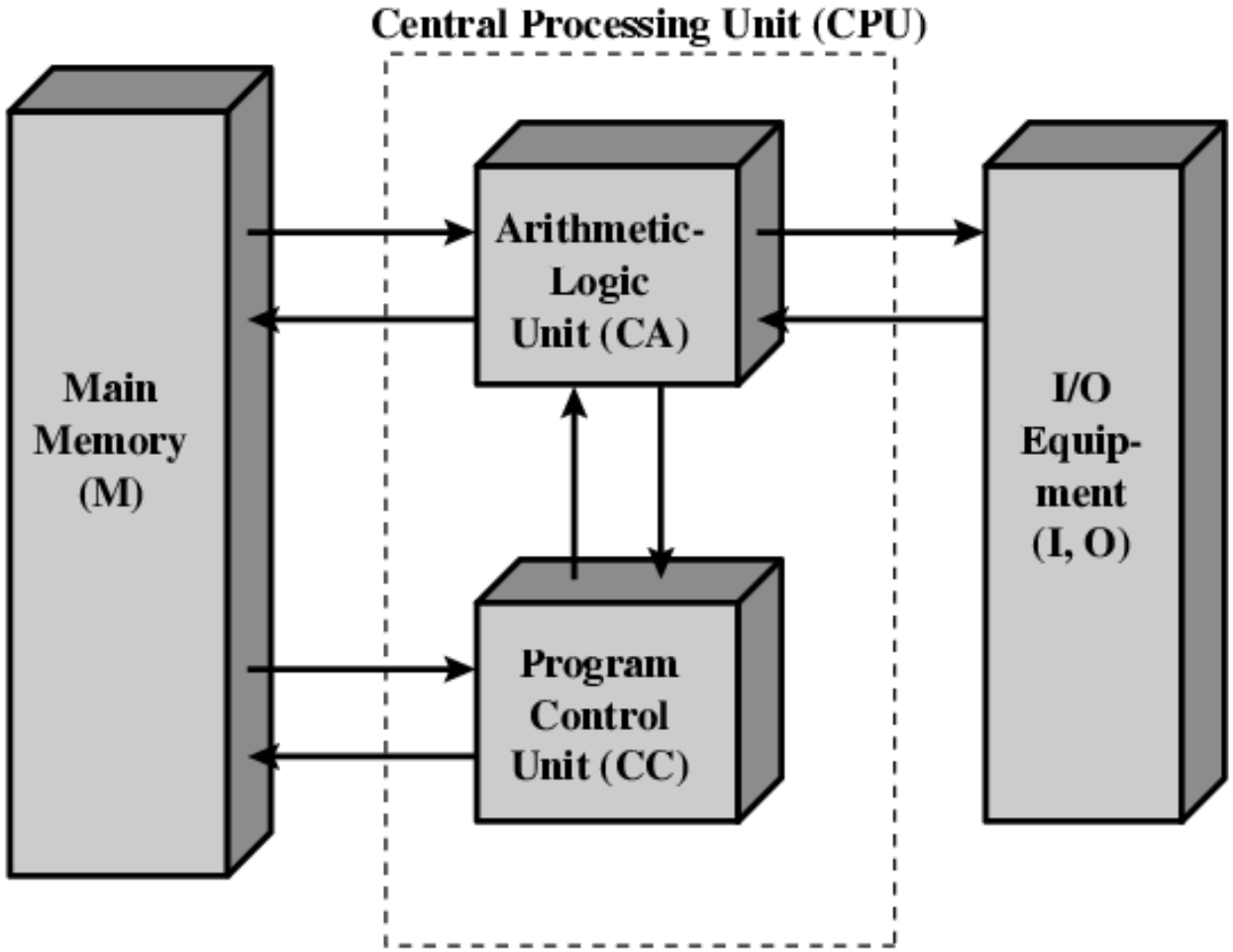
วิวัฒนาการของไมโครโปรเซสเซอร์

- ประสิทธิภาพในการทำงานของซีพียูจะสูงขึ้นถ้ามีจำนวนทรานซิสเตอร์ในตัวเพิ่มมากขึ้นและความถี่ของสัญญาณนาฬิกาเพิ่มสูงขึ้น
- กระบวนการในการพัฒนาซีพียูรุ่นใหม่ที่ออกมาในปัจจุบันจะใช้วิธีการปรับปรุงกระบวนการประมวลผลภายใน แทนการปรับสัญญาณนาฬิกาให้เร็วขึ้นเช่น
  - ซีพียูแบบ **Dual-core**
  - ซีพียูแบบ **Multi-core**
- ทำให้ซีพียูทำงานได้เร็วขึ้นแม้ความถี่ของสัญญาณนาฬิกาจะไม่สูงขึ้นก็ตาม
  - เนื่องจากข้อจำกัดคุณสมบัติทางไฟฟ้าของสารกึ่งตัวนำในการผลิตซีพียู

เครื่องจักร Von Neumann

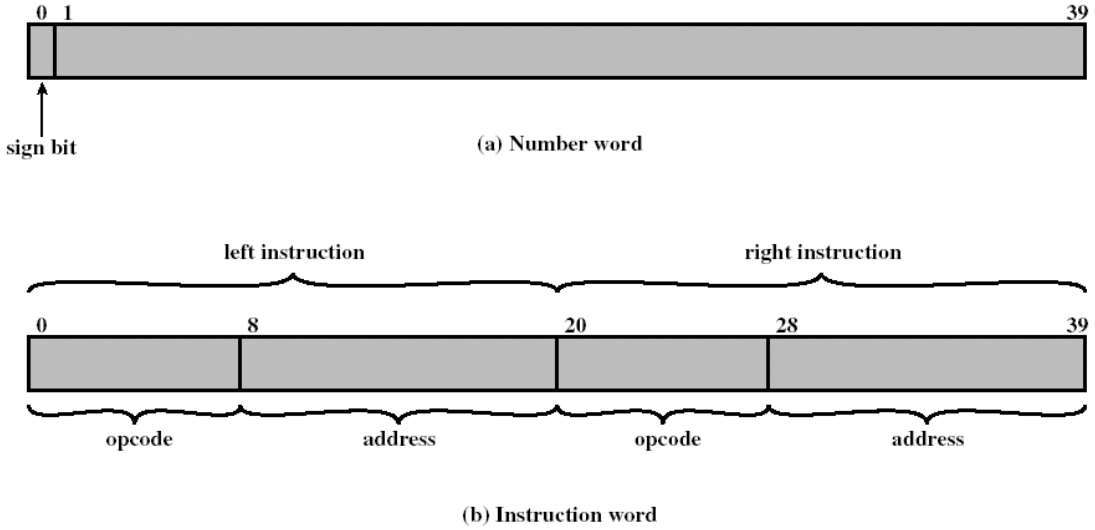
- John Von Neumann ได้เสนอแนวคิดของสถาปัตยกรรมคอมพิวเตอร์ ที่ทำให้เกิดคอมพิวเตอร์ต้นฉบับของเครื่องคอมพิวเตอร์ในปัจจุบันนี้ขึ้นมา และได้ตั้งชื่อในว่า **IAS** (Institute for Advanced Studies) ซึ่งสร้างสำเร็จในปี พ.ศ. 2495 โดยโครงสร้างทั่วไปจะประกอบไปด้วย
  - หน่วยความจำหลักสำหรับเก็บข้อมูลและคำสั่ง
  - หน่วยประมวลผลคณิตศาสตร์และตรรกะ
  - หน่วยควบคุม ทำการประมวลผลคำสั่งในหน่วยความจำ
  - อุปกรณ์ I/O (อ่านและบันทึก) ข้อมูลที่ควบคุมการทำงานโดยหน่วยควบคุม

เครื่องจักร Von Neumann

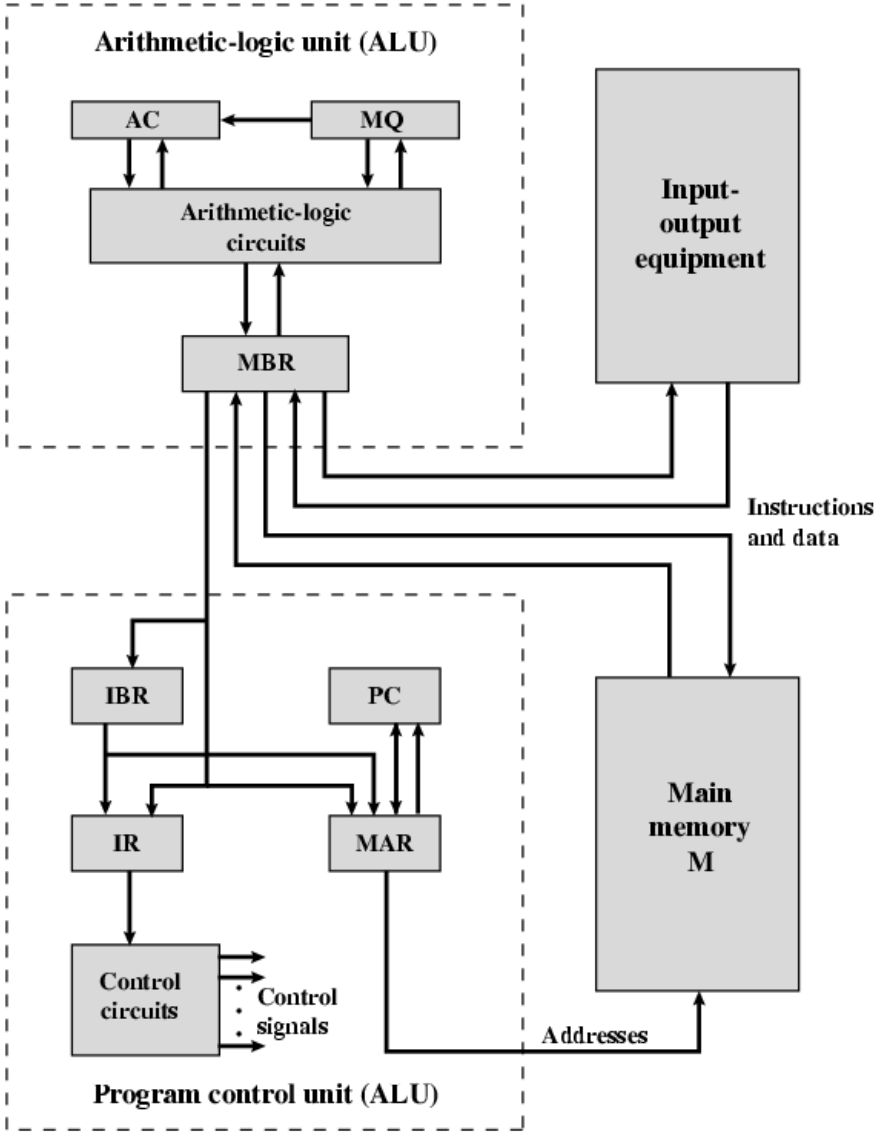


เครื่องจักร Von Neumann

- คอมพิวเตอร์ IAS นี้มีส่วนต่างๆคล้ายกับคอมพิวเตอร์ในยุคปัจจุบัน
  - มีหน่วยความจำหลักที่สามารถใช้เก็บข้อมูลและคำสั่งได้ 1,000 ตำแหน่ง (1 กิโลเวิร์ด)
  - 1 เวิร์ดสามารถจัดเก็บข้อมูลได้ 40 บิต
  - เก็บค่าตัวเลขเป็นแบบ Fixed Point
  - 1 คำสั่งมีขนาด 20 บิต Opcode 8 บิต, Address 12 บิต (1 เวิร์ดจะมี 2 คำสั่ง)



โครงสร้างของคอมพิวเตอร์ IAS



โครงสร้างของคอมพิวเตอร์ IAS

- **Accumulator (AC), Multiplier quotient (MQ)** ใช้เก็บข้อมูลหรือผลลัพธ์ที่ได้จากการทำงานของ ALU
- **Memory Buffer Register (MBR)** ทำหน้าที่เก็บข้อมูลขนาด 1 เวิร์ด ที่บันทึกลงในหน่วยความจำหรืออ่านข้อมูลจากหน่วยความจำ
- **Instruction Buffer Register (IBR)** ทำหน้าที่จัดเก็บคำสั่ง (20 Bits) ทางฝั่งขวาของแต่ละเวิร์ดเพื่อรอการประมวลผล (เก็บข้อมูลชั่วคราว)
- **Program Counter (PC)** จัดเก็บที่อยู่ (Address) ของคำสั่งต่อไปที่จะถูกนำมาประมวลผล
- **Instruction Register (IR)** จัดเก็บ Opcode คำสั่งขนาด 8 บิตของคำสั่งที่กำลังจะประมวลผล
- **Memory Address Register (MAR)** เก็บตำแหน่งที่อยู่ของข้อมูลขนาด 1 เวิร์ดที่กำลังจะถูกบันทึกหรืออ่านข้อมูลมาเก็บใน MBR

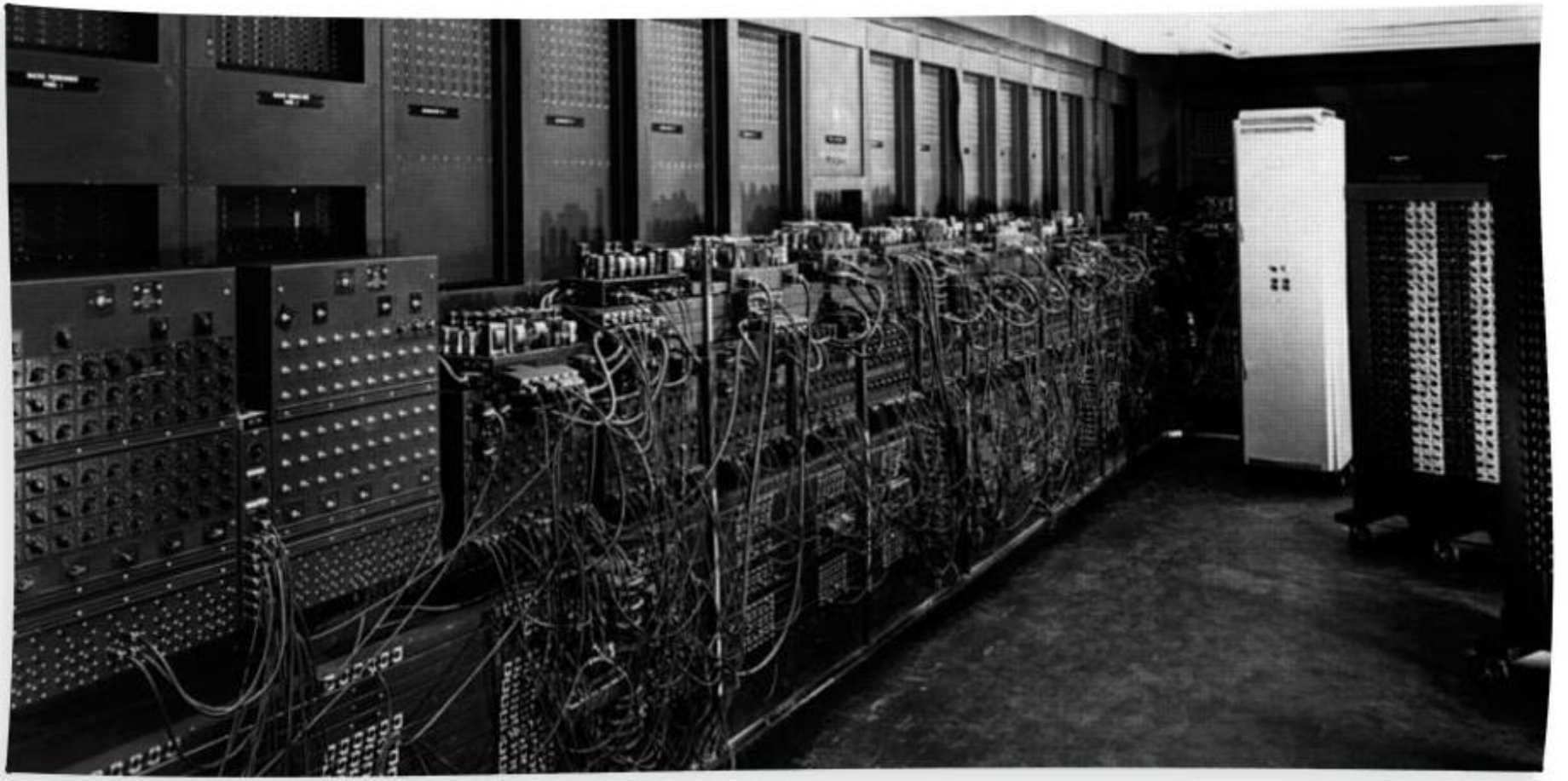
วิวัฒนาการของคอมพิวเตอร์

- วิวัฒนาการของคอมพิวเตอร์และประสิทธิภาพ พิจารณาจาก
  - การเพิ่มความเร็วในการทำงาน CPU
  - การลดขนาดของอุปกรณ์ประกอบ
  - การเพิ่มปริมาณหน่วยความจำ
  - การเพิ่มขีดความสามารถในการจัดเก็บข้อมูล
  - ความเร็วในการทำงาน
- การจัดสมดุลระหว่างองค์ประกอบต่างๆ โดยจะไม่เพิ่มประสิทธิภาพด้านใดด้านหนึ่งที่ส่งผลให้ประสิทธิภาพด้านอื่นลดลง เช่น ความเร็วในการประมวลผลข้อมูลของ CPU จะสูงมากในขณะที่ความเร็วในการอ่าน-เขียน ข้อมูลที่หน่วยความจำเพิ่มขึ้นเพียงเล็กน้อย จึงได้มีการนำเอาเทคนิคต่างๆ มาใช้เพื่อช่วยลดความไม่สมดุลนี้ลง

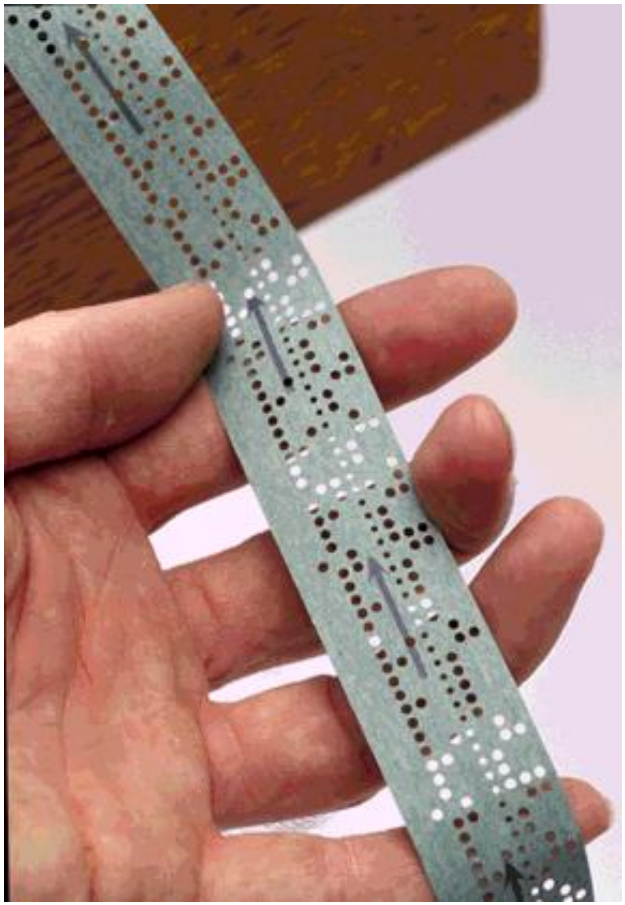
คอมพิวเตอร์ยุคที่ 1 (ค.ศ. 1940 - 1953)

- ยุคที่คอมพิวเตอร์ใช้ **หลอดสูญญากาศ** (Vacuum Tube) เป็นอุปกรณ์ในการประมวลผล
- ในปี ค.ศ. 1946 John Mauchy และ J.P. Eckert แห่งมหาวิทยาลัย Pennsylvania ได้พัฒนาคอมพิวเตอร์ดิจิทัลเครื่องแรกขึ้นมา มีชื่อว่า **"ENIAC"** (Electronic Numerical Integrator and Computer)
  - ตัวเครื่องมีน้ำหนักมากกว่า 30 ตัน ใช้พื้นที่ติดตั้ง 1,500 ตารางฟุต
  - ใช้หลอดสุญญากาศมากกว่า 18,000 หลอด
  - สามารถประมวลผลบวกเลขได้ 5,000 ครั้งต่อวินาที
  - ใช้สำหรับคำนวณตารางวิถีกระสุนปืนใหญ่ โดยใช้เวลาประมาณ 15 วินาที
  - ภาษาคอมพิวเตอร์ในยุคนี้คือภาษาเครื่อง (Machine Language)
  - สื่อบันทึกข้อมูลได้แก่ เทปกระดาษ และบัตรเจาะรู

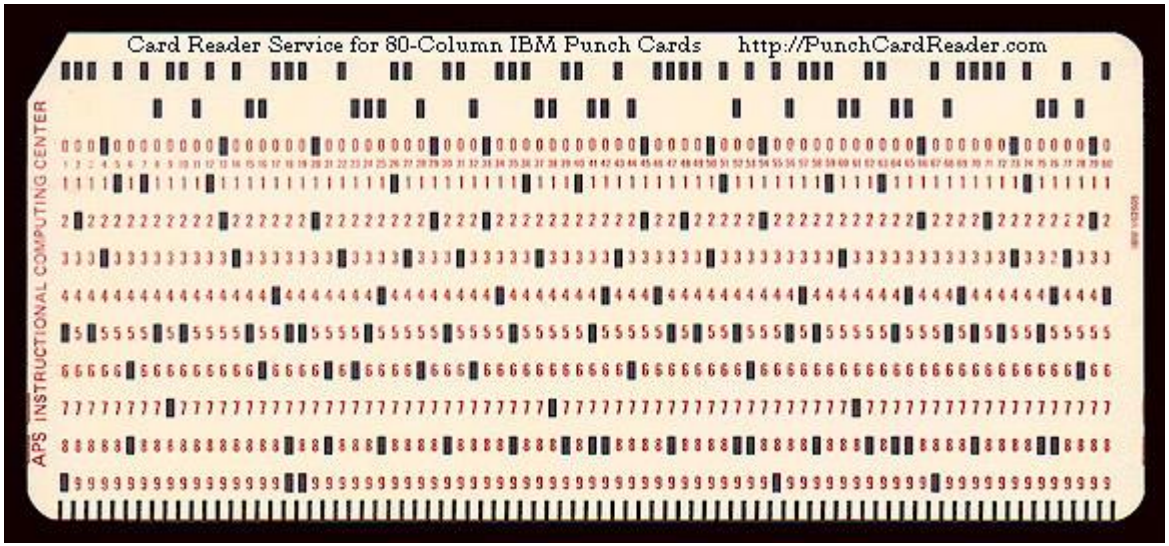
เครื่องคอมพิวเตอร์ ENIAC



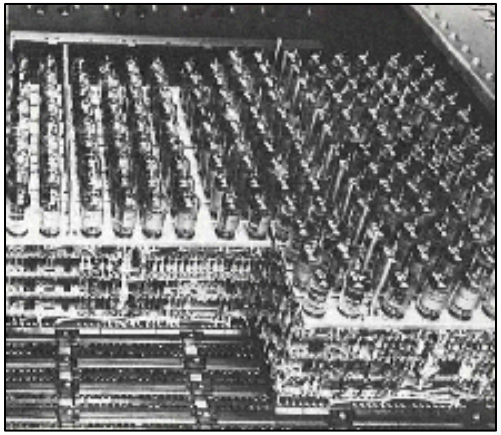
เครื่องคอมพิวเตอร์ ENIAC

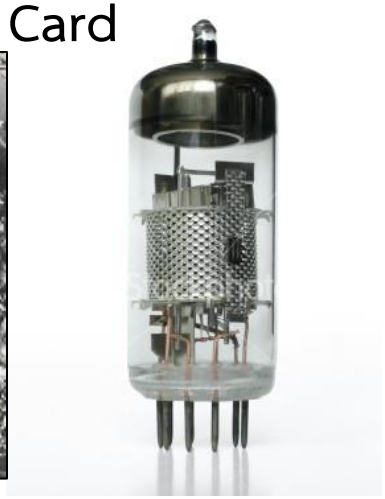


Paper tape



Punch Card





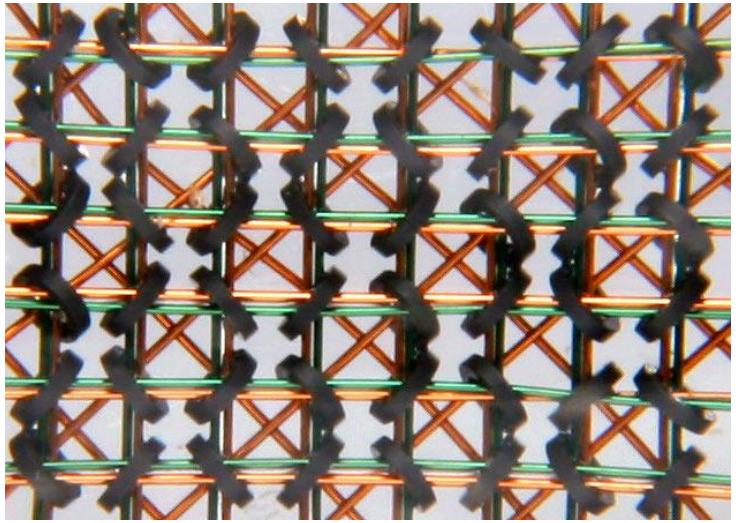
Vacuum Tube

คอมพิวเตอร์ยุคที่ 2 (ค.ศ. 1953 - 1963)

- เป็นยุคที่คอมพิวเตอร์ใช้ **ทรานซิสเตอร์** (Transistor) เข้ามาใช้แทนหลอดสุญญากาศ ทำให้คอมพิวเตอร์มีขนาดเล็กลง ใช้พลังงานน้อยลงและมีราคาถูก
  - หน่วยความจำหลักวงแหวนแม่เหล็ก (Magnetic Core)
  - สื่อบันทึกข้อมูลยังคงใช้เทปแม่เหล็ก (Magnetic Tape) และบัตรเจาะรูเป็นหน่วยความจำสำรอง
  - ภาษาคอมพิวเตอร์พัฒนาขึ้นมาเป็นภาษาสัญลักษณ์ (Symbolic Language) หรือภาษาแอสเซมบลี (Assembly Language)
  - และมีการพัฒนาภาษาระดับสูงขึ้นใช้งานคือภาษา FORTRAN, COBOL และ ALGO
  - ระบบปฏิบัติการทำงานในลักษณะของการประมวลผลแบบกลุ่ม (Batch Processing)

คอมพิวเตอร์ยุคที่ 2 (ทรานซิสเตอร์)

Transistor

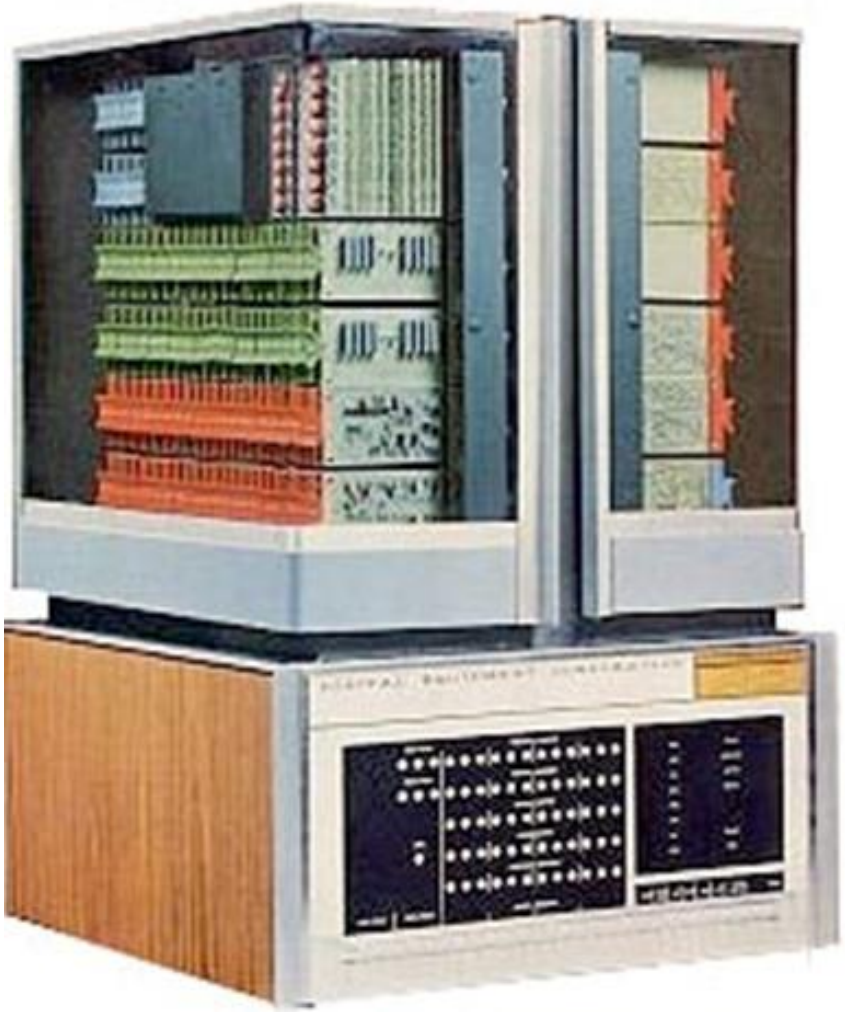


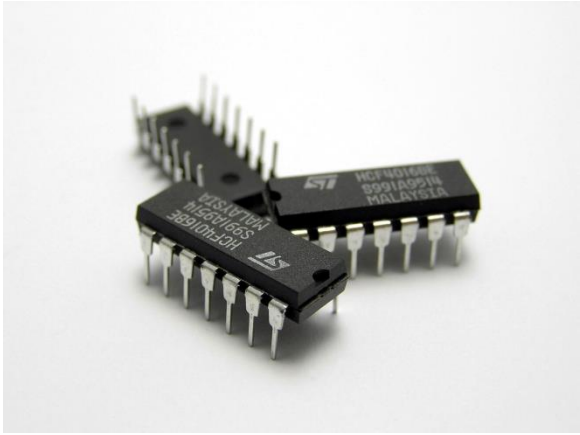
Magnetic Core

คอมพิวเตอร์ยุคที่ 3 (ค.ศ. 1963 - 1972)

- เป็นยุคที่คอมพิวเตอร์ใช้**แผงวงจรรวม** (Integrated Circuit: **IC**) โดยแผงวงจรรวมประกอบไปด้วยทรานซิสเตอร์หลายตัวรวมกัน ทำให้คอมพิวเตอร์มีขนาดเล็กลง ใช้พลังงานน้อยลง ความร้อนลดลงแต่มีความเร็วที่เพิ่มขึ้น
  - หน่วยความจำหลักวงแหวนแม่เหล็ก (Magnetic Core)
  - สื่อบันทึกข้อมูลยังคงใช้เทปแม่เหล็ก (Magnetic Tape)
  - มีการใช้ภาษาระดับสูง เช่น ภาษาปาสคาล ภาษาเบสิก
  - ระบบปฏิบัติการแบบมัลติโปรแกรมมิ่ง (Multi-Programming) และระบบแบ่งเวลา (Time Sharing)
  - เป็นยุคที่เกิดมินิคอมพิวเตอร์ขึ้น ได้แก่รุ่น PDP-8 พัฒนาขึ้นโดยบริษัท Digital Equipment Corporation (DEC)

เครื่องมินิคอมพิวเตอร์ PDP-8 บริษัท DEC





Integrated Circuit

คอมพิวเตอร์ยุคที่ 4 (ค.ศ. 1972 - 1984)

- เป็นยุคที่มีการผลิตแผงวงจรขนาดใหญ่ (Large-scale Integrated Circuit: **LSI**) บรรจุวงจรต่างๆ ไว้มากขึ้น และในยุคนี้ได้มีการผลิตเครื่องคอมพิวเตอร์ขนาดเล็กขึ้นมา (**Microcomputer**)
  - เกิดไมโครโปรเซสเซอร์ (Microprocessor) ตัวแรกของโลกขึ้นมาคือ Intel 4004
  - ประมวลผลขนาด 4 บิต ทำงานที่ความถี่ของสัญญาณนาฬิกา 108 kHz
  - ระบบปฏิบัติการยูนิกส์ (Unix) ค.ศ. 1972 ได้รับการพัฒนาในยุคนี้

Apple Computer

IBM-PC

คอมพิวเตอร์ยุคที่ 5 (ค.ศ. 1984 - ปัจจุบัน)

- เป็นยุคที่มีใช้การผลิตแผงวงจรขนาดใหญ่มาก (Very Large-scale Integrated Circuit: VLSI) โดยภายในตัว VLSI ได้บรรจุทรานซิสเตอร์ได้เป็นล้านตัว
  - พัฒนารูปแบบการโต้ตอบบนหน้าจอแบบกราฟฟิก (GUI)
  - นำเครือข่ายอินเทอร์เน็ตเข้ามาใช้ในองค์กร
  - ภาษาคอมพิวเตอร์ได้ถูกพัฒนาไปสู่การเขียนโปรแกรมเชิงวัตถุ (Object Oriented Programming) เช่นภาษา C++
  - นำเทคโนโลยีคอมพิวเตอร์มาประยุกต์ใช้ด้านองค์ความรู้
  - ระบบผู้เชี่ยวชาญ (Expert System)
  - ระบบปัญญาประดิษฐ์ (Artificial Intelligence)

ประเภทของคอมพิวเตอร์

- การพิจารณาความสามารถของคอมพิวเตอร์เพื่อนำมาใช้งาน ให้ถูกต้องตามลักษณะของงาน พิจารณาได้จากความเหมาะสม ประสิทธิภาพ ขนาด ราคา โดยเรียงลำดับจากขนาดใหญ่ไปจนถึงขนาดเล็ก ดังนี้

1. ซูเปอร์คอมพิวเตอร์ (Supercomputer)
2. เมนเฟรมคอมพิวเตอร์ (Mainframe Computer)
3. มินิคอมพิวเตอร์ (Minicomputer)
4. ไมโครคอมพิวเตอร์ (Microcomputer)

ซูเปอร์คอมพิวเตอร์ (Supercomputer)

- เป็นคอมพิวเตอร์ขนาดใหญ่ที่สุดและมีความเร็วสูงที่สุด เหมาะกับการประมวลผลข้อมูลที่มีความซับซ้อน รองรับโปรเซสเซอร์ได้จำนวนมหาศาล ตัวเหมาะกับงานประเภทที่ต้องมีการคำนวณทางด้านวิทยาศาสตร์ เช่น
  - การถอดรหัสทางพันธุกรรม
  - การพยากรณ์อากาศ
  - การแพร่กระจายของกัมมันตภาพรังสี
  - การสร้างแบบจำลองในโรงงาน

Summit

เมนเฟรมคอมพิวเตอร์

- Mainframe Computer เป็นคอมพิวเตอร์ขนาดใหญ่ เหมาะกับงานที่ประมวลผลข้อมูลพร้อมกันเป็นจำนวนมาก ซึ่งส่วนใหญ่จะใช้งานทางด้านฐานข้อมูลในธุรกิจขนาดใหญ่ ที่มีผู้ใช้จำนวนมากต้องการเข้าถึงข้อมูลในเวลาเดียวกันเหมาะกับงานเช่น
  - ธนาคาร
  - สายการบิน
  - การจัดเก็บภาษี
  - การบริการสืบค้นข้อมูลสารสนเทศ

มินิคอมพิวเตอร์ (Mini Computer)

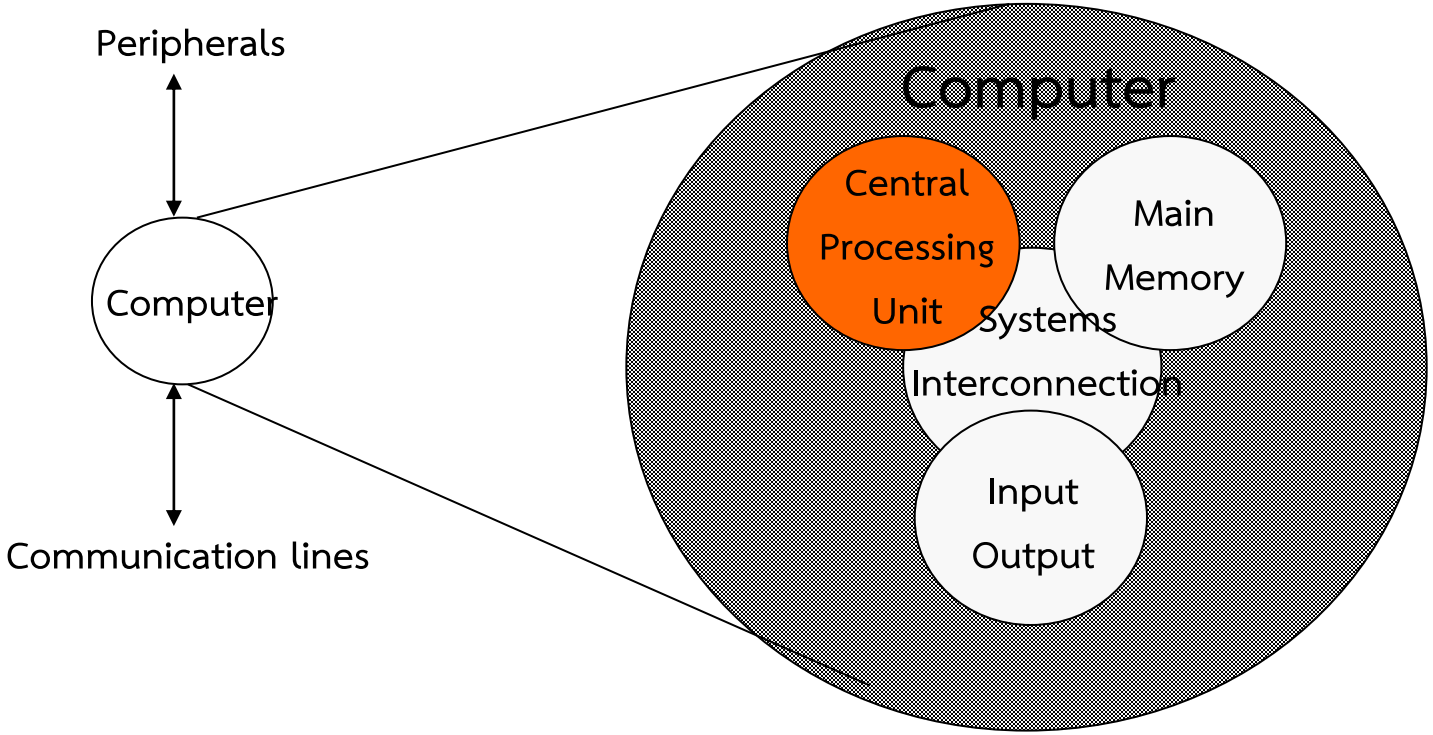
- เป็นคอมพิวเตอร์ที่ใช้ในธุรกิจขนาดเล็ก เพื่อช่วยในด้านการประมวลผลให้เกิดความรวดเร็ว โดยความสามารถจะอยู่ระหว่างเครื่องคอมพิวเตอร์เมนเฟรมกับเครื่องเวิร์กสเตชัน บางครั้งเราเรียกกันว่า มิดเรนจ์คอมพิวเตอร์ (Mid-range Computer) สามารถนำไปใช้ในงานต่างๆ เช่น
  - ธุรกิจขนาดกลาง
  - โรงงานอุตสาหกรรม
  - ธุรกิจโรงพยาบาล
  - ฯลฯ

ไมโครคอมพิวเตอร์ (Microcomputer)

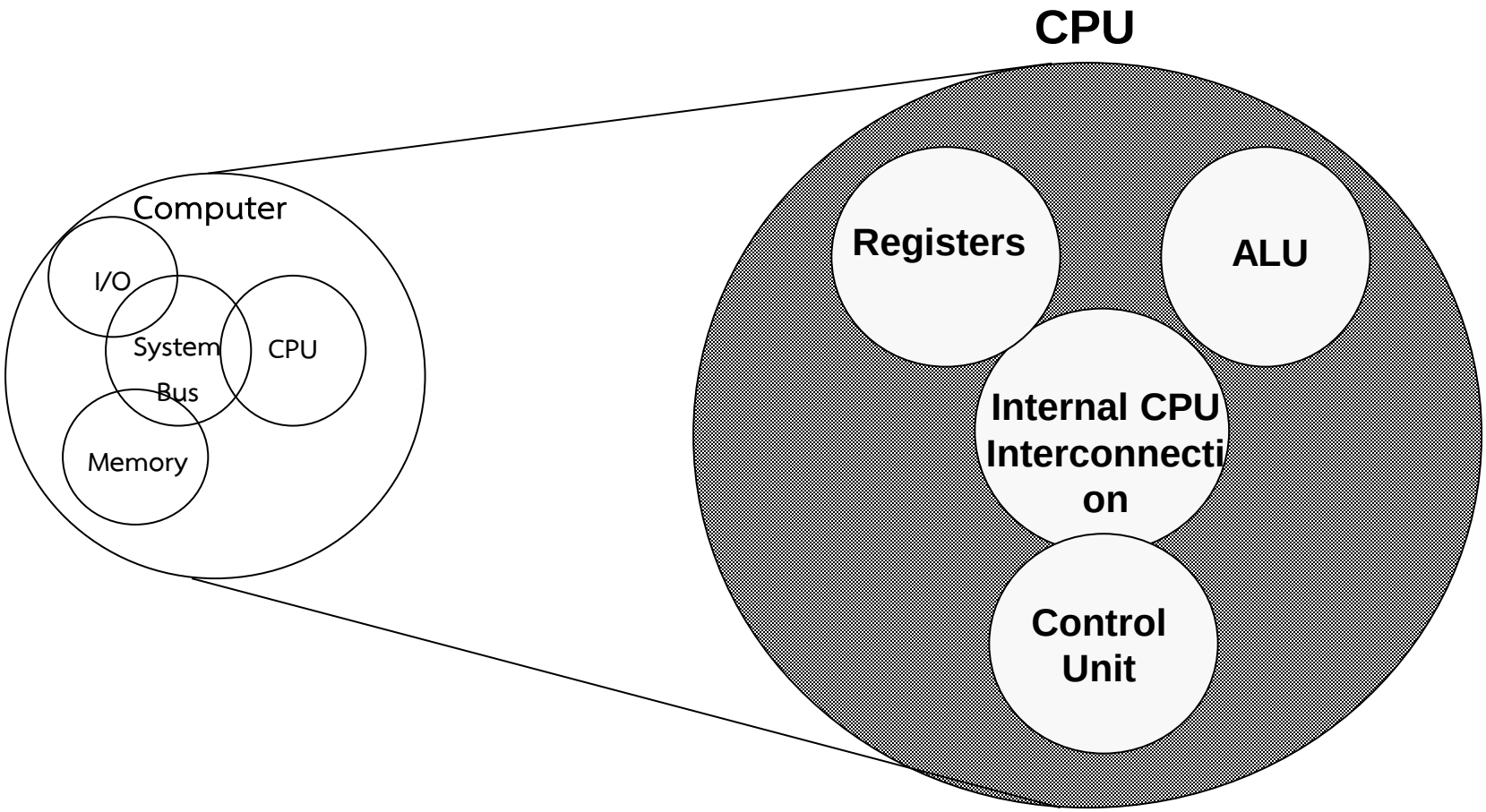
- ถูกออกแบบมาสำหรับผู้ใช้งานเพียงคนเดียว (Single User) บางครั้งเราเรียกกันอีกชื่อหนึ่งว่าคอมพิวเตอร์ส่วนบุคคล (Personal Computer: PC) สามารถประยุกต์ใช้กับงานได้หลายประเภท ซึ่งมีการทำงานลอกเลียนแบบมาจากบริษัทไอบีเอ็มและเรียกกันว่าพีซีคอมแพทิเบิ้ล (PC Compatible)

โครงสร้างของคอมพิวเตอร์

- ประกอบด้วยองค์ประกอบที่สำคัญ 4 ส่วนคือ



CPU (Central Processing Unit)



CPU (Central Processing Unit)

- เปรียบเสมือนเป็นสมองของเครื่องคอมพิวเตอร์ โดยทำหน้าที่ในการคำนวณค่าต่าง ๆ ตามคำสั่งที่ได้รับ และควบคุมการทำงานของส่วนประกอบอื่น ๆ ทั้งหมด ในระบบไมโครคอมพิวเตอร์

- **หน่วยคำนวณทางคณิตศาสตร์และตรรกะ** (Arithmetic and Logic Unit : ALU)
  - เป็นหน่วยที่ทำหน้าที่ประมวลผลโดยใช้วิธีคณิตศาสตร์
    - เช่น บวก ลบ คูณ หาร
  - หรือ ทำหน้าที่ประมวลผลทางตรรกะ
    - เช่น AND OR NOT COMPLEMENT เป็นต้น รวมทั้งยังทำหน้าที่ในการเปรียบเทียบค่าต่าง ๆ

CPU (Central Processing Unit)

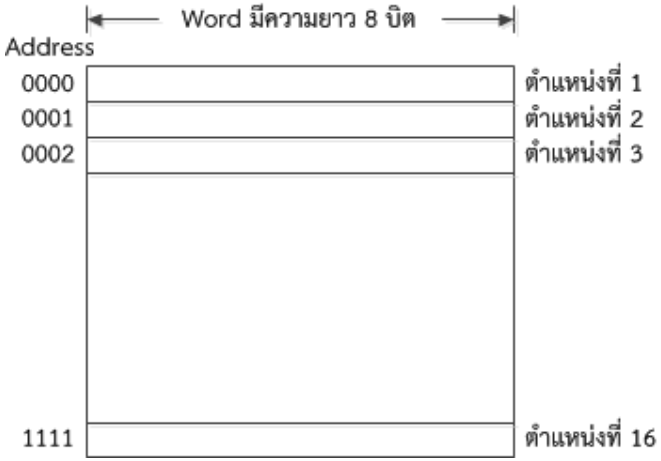
- **หน่วยเก็บข้อมูลชั่วคราว** (Register)
  - เป็นหน่วยความจำขนาดเล็ก ทำหน้าที่เป็นที่จัดเก็บข้อมูล ชั่วคราวก่อนที่จะถูกนำไปประมวลผล
  - การอ้างถึงข้อมูลของ Register จะมีความเร็วเท่ากับความเร็วของหน่วยประมวลผลกลาง เพราะเป็นหน่วยความจำส่วนที่อยู่ภายในหน่วยประมวลผลกลางจึงไม่ต้องไปอ้างถึงภายนอกหน่วยประมวลผล
- **หน่วยควบคุม** (Control Unit)
  - ทำหน้าที่ควบคุมการของระบบคอมพิวเตอร์ทั้งหมด กำหนดจังหวะการทำงานต่าง ๆ ของคอมพิวเตอร์และส่วนประกอบอื่น ๆ ของ CPU
  - ทำหน้าที่ควบคุมการส่งข้อมูลระหว่างหน่วยต่าง ๆ ในคอมพิวเตอร์

Memory

- **หน่วยความจำหลัก** (Main Memory)
  - ทำหน้าที่จัดเก็บ Instruction ข้อมูลหรือโปรแกรม เพื่อที่จะถูกเรียกใช้งานจาก CPU
- **หน่วยความจำภายนอก** (External Memory)
  - หรือ**หน่วยความจำสำรอง** (Secondary Memory) ใช้จัดเก็บข้อมูลเหมือนหน่วยความจำหลัก แต่ CPU ไม่สามารถเข้าถึงหน่วยความจำภายนอกได้โดยตรง

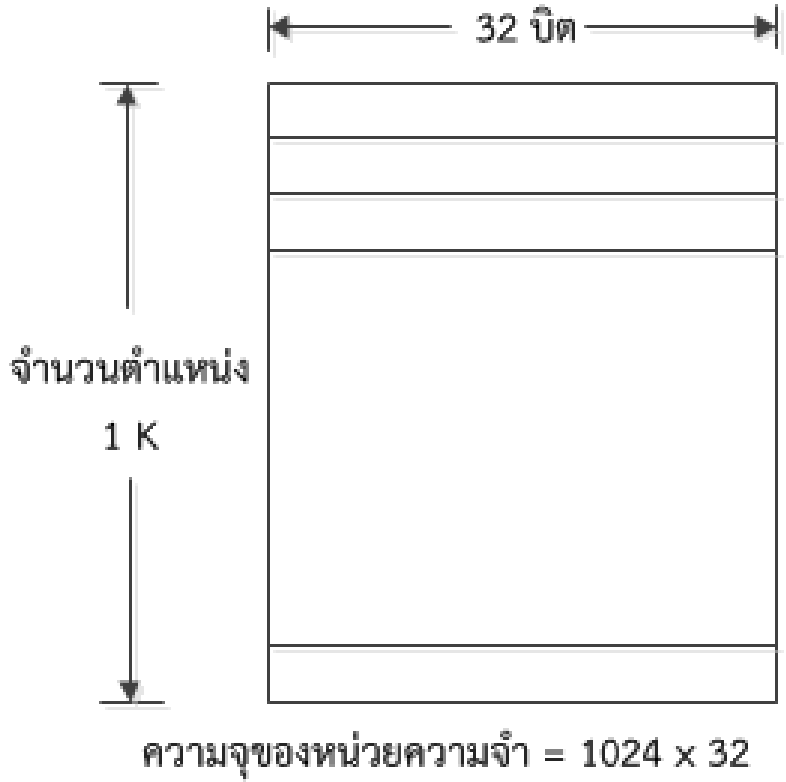
หน่วยความจำหลัก (Main Memory)

- จะมีการจัดพื้นที่และตำแหน่งสำหรับจัดเก็บ Instruction และข้อมูล โดยแต่ละตำแหน่งจะมีค่าแอดเดรส (Address) ที่ไม่ซ้ำกัน
  - ตำแหน่ง (Address) ของหน่วยความจำ ถูกนำมาใช้เพื่ออ้างถึงตำแหน่งของข้อมูลที่เก็บในหน่วยความจำหลัก
  - ขนาดความจุของหน่วยความจำหาได้จากตำแหน่งคูณกับความยาวของ Word เช่น หน่วยความจำหลักมี 16 ตำแหน่งและความยาว Word มีขนาด 4 บิต เพราะฉะนั้นความจุรวมของหน่วยความจำหลักหาได้จาก 16 x 4 บิต เท่ากับ 64 บิตหรือ 8 ไบต์



หน่วยความจำหลัก (Main Memory)

- การอ้างถึงตำแหน่งของหน่วยความจำจะถูกระบุด้วย Address ที่แตกต่างกัน จำนวน Address บนหน่วยความจำเป็น 2^n ตำแหน่ง

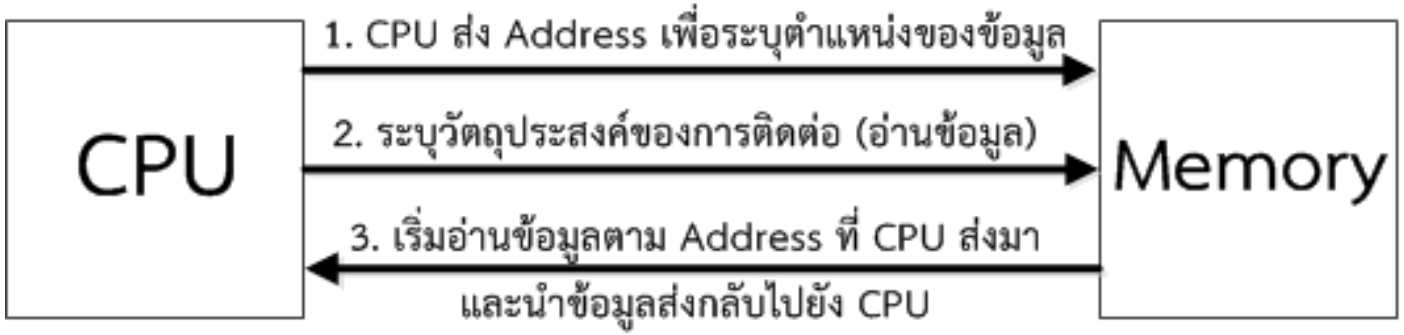


หน่วยความจำหลัก (Main Memory)

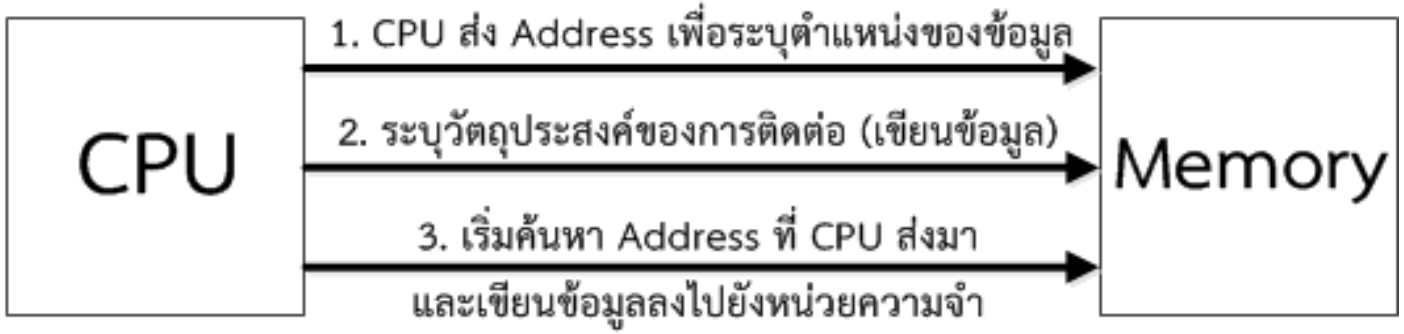
- **RAM** (Random Access Memory)
  - เป็นหน่วยความจำที่ใช้จัดเก็บข้อมูลชั่วคราว สามารถลบและเขียนข้อมูลใหม่ได้ สามารถจัดเก็บข้อมูลได้ในขณะที่มีไฟฟ้าหล่อเลี้ยงเท่านั้น ถูกนำมาใช้เป็นหน่วยความจำหลักของคอมพิวเตอร์
- **ROM** (Read Only Memory)
  - เป็นหน่วยความจำที่ไม่สามารถบันทึกข้อมูลใหม่ลงไปได้ แต่สามารถจัดเก็บข้อมูลโดยไม่จำเป็นต้องมีไฟฟ้าหล่อเลี้ยง ดังนั้น ROM จึงถูกนำมาใช้จัดเก็บชุดคำสั่ง “**ROM Bootstrap**”
    - ROM Bootstrap ใช้เพื่อบอกให้ CPU ทราบว่าตอนเปิดเครื่องต้องเริ่มต้นการทำงานด้วยคำสั่งใดบ้าง โดยปกติชุดคำสั่งจะถูกบันทึกมาจากโรงงาน

การติดต่อระหว่าง CPU และ หน่วยความจำ

- การอ่านและเขียน



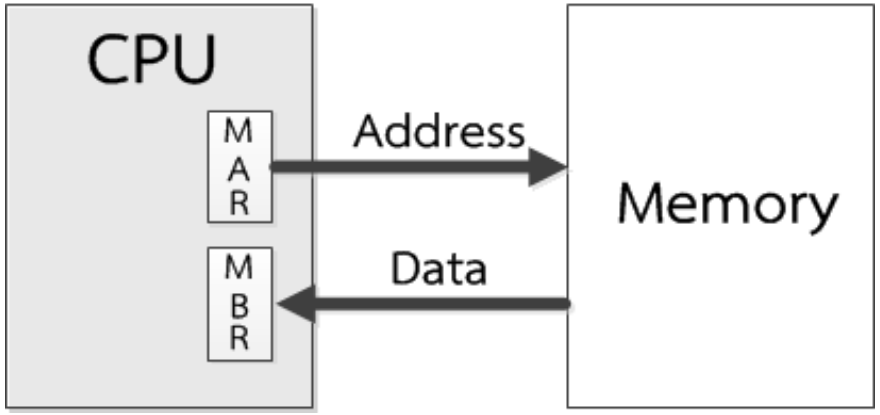
รูปแสดงการอ่านข้อมูลบนหน่วยความจำ



รูปแสดงการเขียนข้อมูลบนหน่วยความจำ

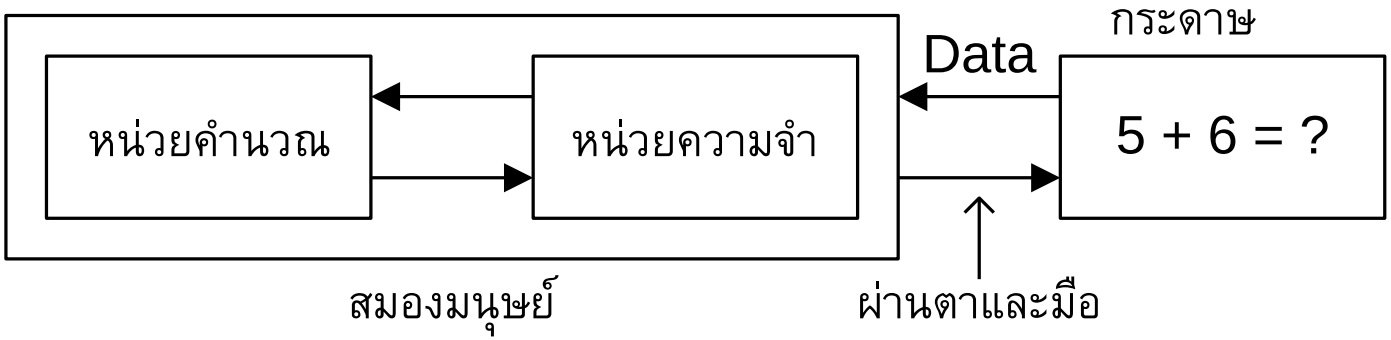
การติดต่อระหว่าง CPU และ Register

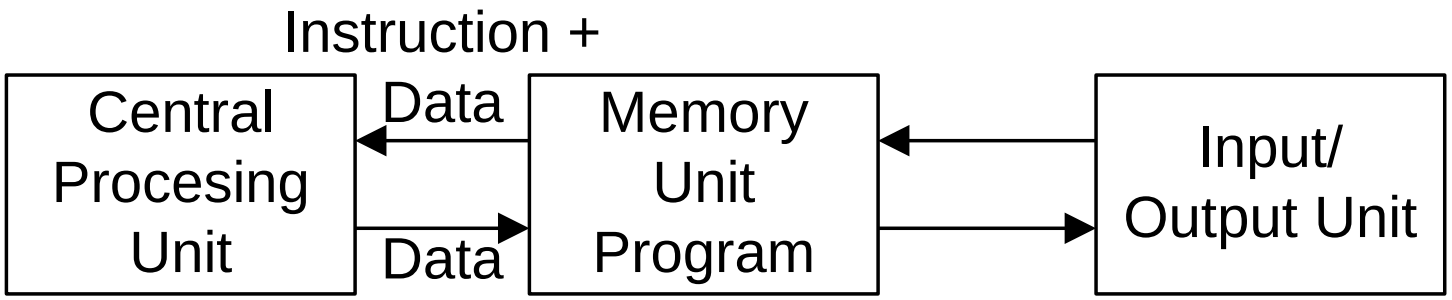
- CPU จะใช้ Register ซึ่งเป็นหน่วยความจำชั่วคราวที่อยู่ใน CPU 2 ตัวคือ MAR (Memory Address Register) และ MBR (Memory Buffer Register) เพื่ออ่านและเขียนข้อมูลลงหน่วยความจำ
  - CPU จะนำค่า Address ที่ต้องการเก็บลงใน MAR ก่อนนำไปชี้ตำแหน่งบนหน่วยความจำ
  - MBR ใช้เก็บข้อมูลจาก CPU ในกรณีการเขียนข้อมูลหรือเก็บข้อมูลจากหน่วยความจำในกรณีของการอ่านข้อมูล



คอมพิวเตอร์กับการคำนวณ

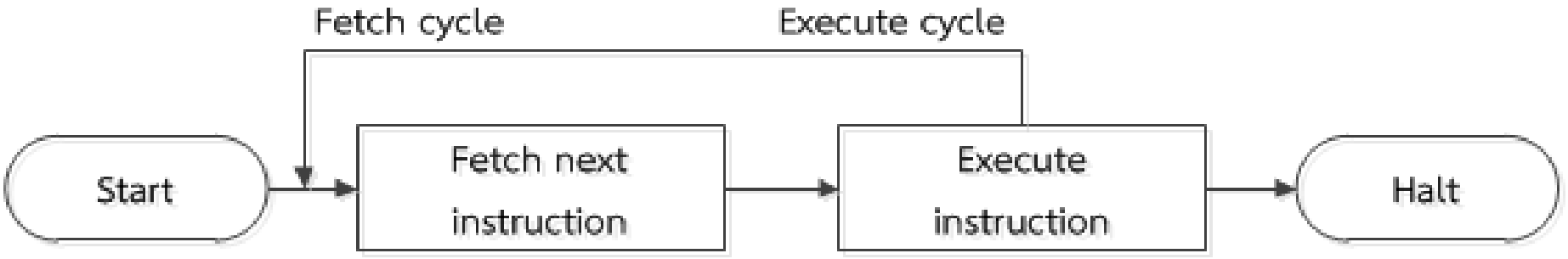
- การคำนวณของคอมพิวเตอร์เป็น von Neumann's Machine





วงรอบคำสั่งของ CPU (Instruction Cycle)

- การที่ CPU ทำการประมวลผลคำสั่งต่างๆ สามารถแบ่งขั้นตอนการทำงานออกเป็นวงรอบคำสั่งได้ดังนี้
  1. **Fetch** เป็นระยะที่ CPU ทำหน้าที่ดึง Instruction (Fetch Instruction) จากหน่วยความจำหลัก
  2. **Execute** เป็นระยะที่ CPU ทำหน้าที่ตีความ Instruction (Execute Instruction) เพื่อปฏิบัติงานตาม Instruction จากนั้นจึงเตรียมทำ Instruction ต่อไป



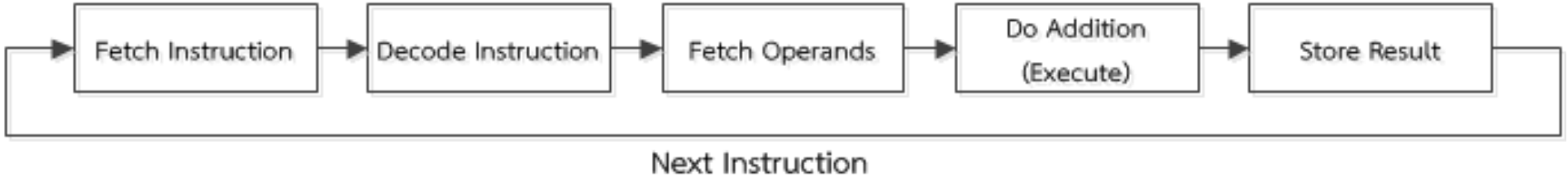
รูปแบบของ Instruction

- รูปแบบโดยทั่วไปของ Instruction ประกอบไปด้วย 2 ส่วน ได้แก่
  1. **Operation Code** หรือ OpCode เป็นรหัสคำสั่งที่ต้องการดำเนินการ เช่น ADD, SUB, STORE และ LOAD เป็นต้น
  2. **Operand** ทำหน้าที่เก็บ Address ที่อ้างอิงข้อมูลที่อยู่ในหน่วยความจำที่ต้องการ

รูปแบบของ Instruction

- หากนำ Instruction บางชนิดมาประมวลผลบน CPU บางครั้งอาจมีรายละเอียดที่ต่างกันเช่น Instruction ชนิด ADD

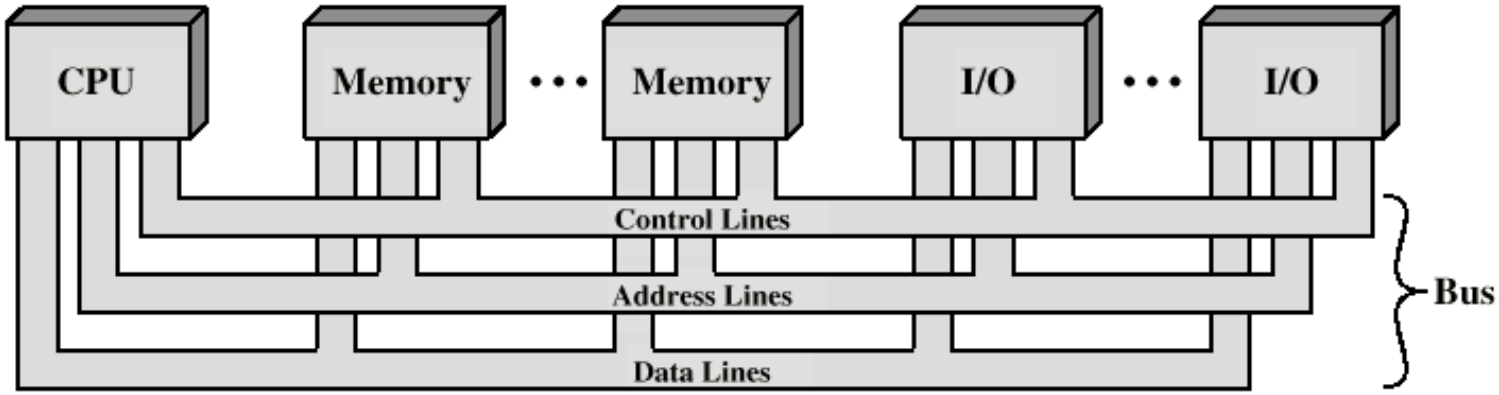
- กระบวนการของ Instruction Cycle จะได้ดังรูป



Bus

- System Interconnection หรือบัส (**Bus**)
  - การเชื่อมต่อระหว่าง CPU และหน่วยความจำ

1. แอดเดรสบัส (**Address Bus**) เป็นกลุ่มสัญญาณที่ใช้สำหรับอ้างถึงตำแหน่งของหน่วยความจำและตำแหน่งพอร์ตอินพุตเอาต์พุต
2. บัสข้อมูล (**Data Bus**) เป็นกลุ่มสัญญาณที่ใช้ในการรับส่งข้อมูล
3. บัสควบคุม (**Control Bus**) เป็นกลุ่มสัญญาณที่ใช้ควบคุม CPU และควบคุมอุปกรณ์ที่ CPU ต้องการติดต่อด้วย



Input/Output Unit

- **หน่วยอินพุต** (Input Unit)
  - ทำหน้าที่รับข้อมูลจากผู้ใช้งานเข้าสู่หน่วยความจำหลัก และจะทำการเปลี่ยนประเภทของข้อมูลที่รับเข้ามาให้อยู่ในรูปแบบที่ CPU สามารถประมวลผลได้
  - เช่น การกดแป้นพิมพ์ จะต้องเปลี่ยนสัญญาณทางไฟฟ้าไปเป็นรหัสคอมพิวเตอร์
- **หน่วยเอาต์พุต** (Output Unit) ทำหน้าที่แสดงผลลัพธ์จากการประมวลผลของคอมพิวเตอร์ สามารถแบ่งออกได้เป็น 2 ประเภทคือ
  - หน่วยแสดงผลชั่วคราว (Soft Copy) เช่น จอภาพ
  - หน่วยแสดงผลถาวร (Hard Copy) เช่น ผลลัพธ์จากเครื่องพิมพ์
- **พอร์ต** (Port)
  - เป็นช่องทางในการติดต่อกับหน่วยอินพุตและเอาต์พุต โดยเชื่อมต่อผ่านทางระบบบัส โดยแต่ละพอร์ตจะมีแอดเดรสประจำตัวของพอร์ตนั้นๆ

Note

ข้อมูล
(Data)

Chapter 2

Computer Organization and Architecture

ปรับปรุงจากเอกสารของ
อ. ณัฐวุฒิ สร้อยดอกสน (NSD)

อ.เอิญ สุริยะฉาย (ENS)
ภาควิชาวิทยาการคอมพิวเตอร์และสารสนเทศ
มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าพระนครเหนือ

ข้อมูล

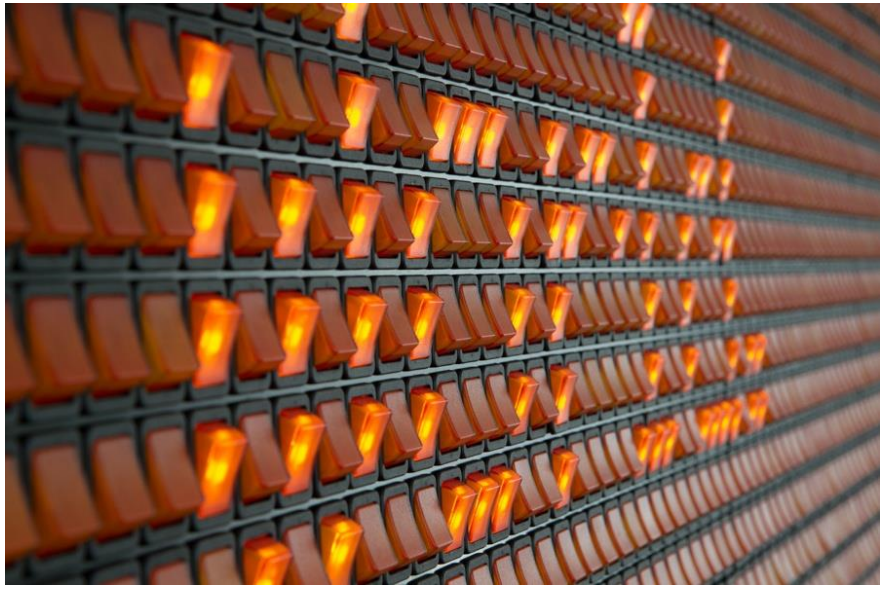
- **ข้อมูล (Data)** ที่คอมพิวเตอร์สามารถประมวลผลได้จะต้องอยู่ในรูปแบบของสัญญาณทางไฟฟ้า ซึ่งเป็นสัญญาณดิจิทัลที่ถูกแทนด้วยลอจิก “0” และ “1” โดยนำข้อมูลทั่วไปมาเปลี่ยนให้สามารถประมวลผลได้ด้วยวิธีการเข้ารหัส (Coding)
  - และเมื่อคอมพิวเตอร์ประมวลผลแล้วต้องนำข้อมูลนั้นส่งออกไปทางเอาต์พุต จะต้องนำข้อมูลนั้นไปทำการถอดรหัส (Decoding) เพื่อให้ได้เอาต์พุตตามที่ต้องการ
  - รหัสใช้แทนตัวเลขที่สามารถนำไปคำนวณได้จะใช้รหัสเลขฐานสอง (Binary Code) และรหัส BCD (Binary Coded Decimal)
  - รหัสที่ใช้แทนตัวอักษรจะใช้รหัส ASCII (American Standard Code for Information Interchange), Unicode และ EBCDIC

ระบบตัวเลข

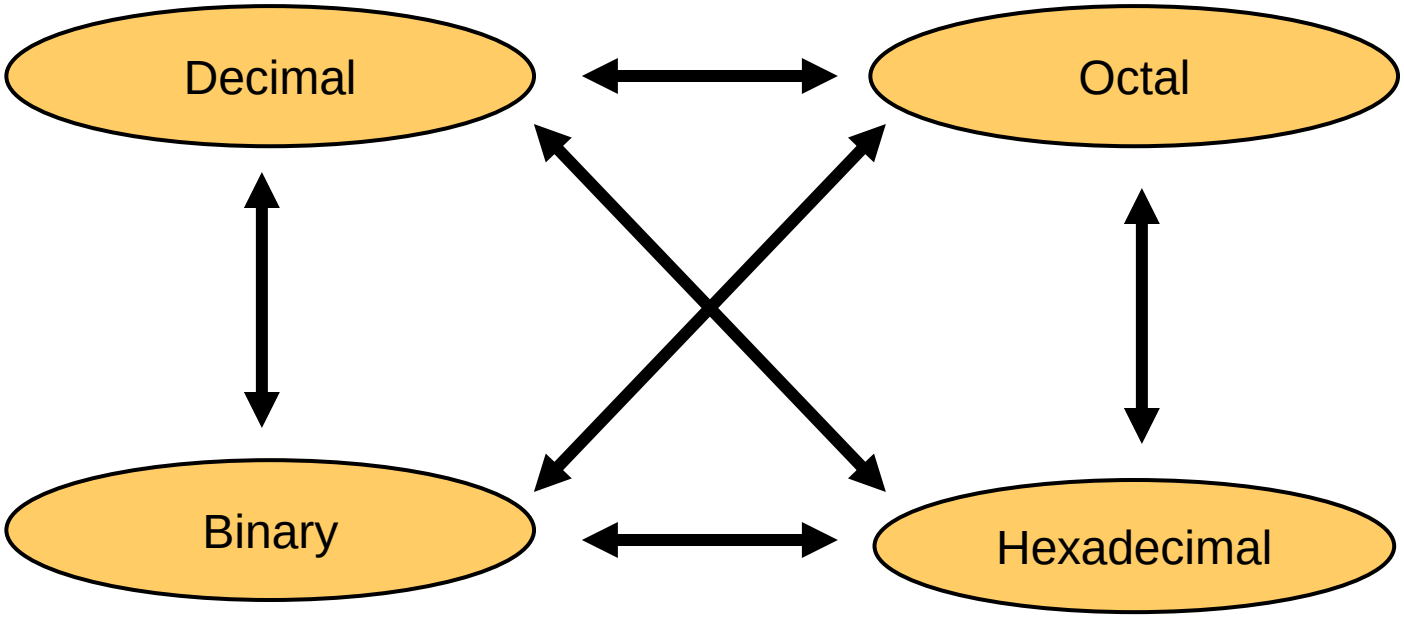
System	Base	Symbols	Used by humans?	Used in computers?
Decimal	10	0, 1, … 9	Yes	No
Binary	2	0, 1	No	Yes
Octal	8	0, 1, … 7	No	No
Hexa-decimal	16	0, 1, … 9, A, B, … F	No	No

เลขฐาน2 (Binary)

- เลขฐานสองแต่ละหลักเรียกว่า **“บิต” (Bit)** โดย 1 บิตสามารถแทนได้ 2 สถานะ คือ 0 หรือ 1 ส่วน
  - ข้อมูล **1 ไบต์ (Byte)** จะมีค่าเท่ากับ 8 บิต (หรือ 1 ตัวอักษร) ดังนั้นใน 1 ไบต์สามารถแทนข้อมูลในรูปของเลขฐานสองได้ทั้งหมด 28 หรือ 256 ค่าที่ไม่ซ้ำกัน



การแปลงเลขฐาน



$$25_{10} = 11001_2 = 31_8 = 19_{16}$$

Binary numbers

- เลขฐาน 2 (Binary numbers)
- Digits = {0, 1}

$N = (11010.11)_2$

$= 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2}$

$= 1 \times 16 + 1 \times 8 + 0 \times 4 + 1 \times 2 + 0 \times 1 + 1 \times 1/2 + 1 \times 1/4$

$= (26.75)_{10}$

Powers of 2:

$2^0 = 1$	$2^4 = 16$	$2^8 = 256$
$2^1 = 2$	$2^5 = 32$	$2^9 = 512$
$2^2 = 4$	$2^6 = 64$	$2^{10} = 1024$
$2^3 = 8$	$2^7 = 128$	

1 K (kilo) $= 2^{10} = 1{,}024$
1M (mega) $= 2^{20} = 1{,}048{,}576$
1G (giga) $= 2^{30} = 1{,}073{,}741{,}824$

Octal numbers

- เลขฐาน 8 (Octal numbers)
- Digits = {0, 1, 2, 3, 4, 5, 6, 7}

$N = (127.4)_2$

$= 1 \times 8^2 + 2 \times 8^1 + 7 \times 8^0 + 4 \times 8^{-1}$

$= 1 \times 64 + 2 \times 8 + 7 \times 1 + 4 \times 1/8$

$= (87.5)_{10}$

Powers of 8:

$8^0 = 1$	$8^4 = 4{,}096$	$8^8 = 16{,}777{,}216$
$8^1 = 8$	$8^5 = 32{,}768$	$8^9 = 134{,}217{,}728$
$8^2 = 64$	$8^6 = 262{,}144$	$8^{10} = 1{,}073{,}741{,}824$
$8^3 = 512$	$8^7 = 2{,}097{,}152$	

Hexadecimal numbers

- เลขฐาน 16 (Hexadecimal numbers)
- Digits = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}

$N = (B65F)_{16}$

$= 11 \times 16^3 + 6 \times 16^2 + 5 \times 16^1 + 15 \times 16^0$

$= 11 \times 4096 + 6 \times 256 + 5 \times 16 + 15 \times 1$

$= (46,687)_{10}$

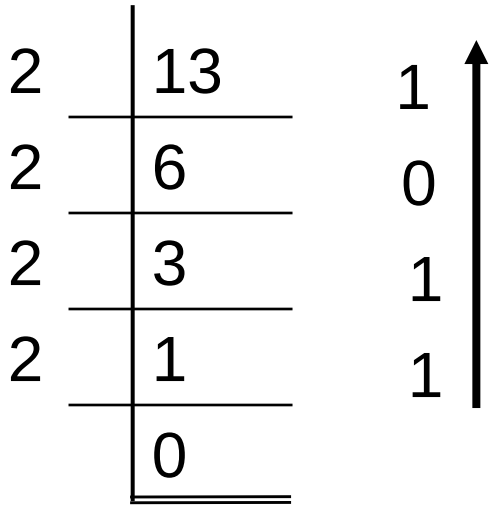
Powers of 16:

$16^0 = 1$	$16^4 = 65,536$	$16^8 = 4,294,967,296$
$16^1 = 16$	$16^5 = 1,048,576$	$16^9 = 68,719,476,736$
$16^2 = 256$	$16^6 = 16,777,216$	$16^{10} = 1,099,511,627,776$
$16^3 = 4,096$	$16^7 = 268,435,456$	

Decimal	Binary	Octal	Hexadecimal
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10

Decimal to Binary

- $13.6875_{10} = ?_2$



$(13)_{10} = (1101)_2$

0.6875 ➔

0.6875	x 2	=	1.375
0.375	x 2	=	0.750
0.750	x 2	=	1.500
0.500	x 2	=	1.000

$0.6875_{10} = 0.1011_2$

$13.6875_{10} = 1101.1011_2$

Decimal to Octal

- $239.1875_{10} = ?_8$

8	239	
8	29	7
8	3	5
	0	3

$239_{10} = 357_8$

0.1875 ➔

$0.1875 \times 8 = 1.5$

$0.500 \times 8 = 4.000$

$0.1875_{10} = 0.14_8$

$(239.1875)_{10} = (357.14)_8$

Decimal to Hexadecimal

- $2748.75_{10} = ?_{16}$

16	2748		
16	171	12	= C
16	10	11	= B
	0	10	= A

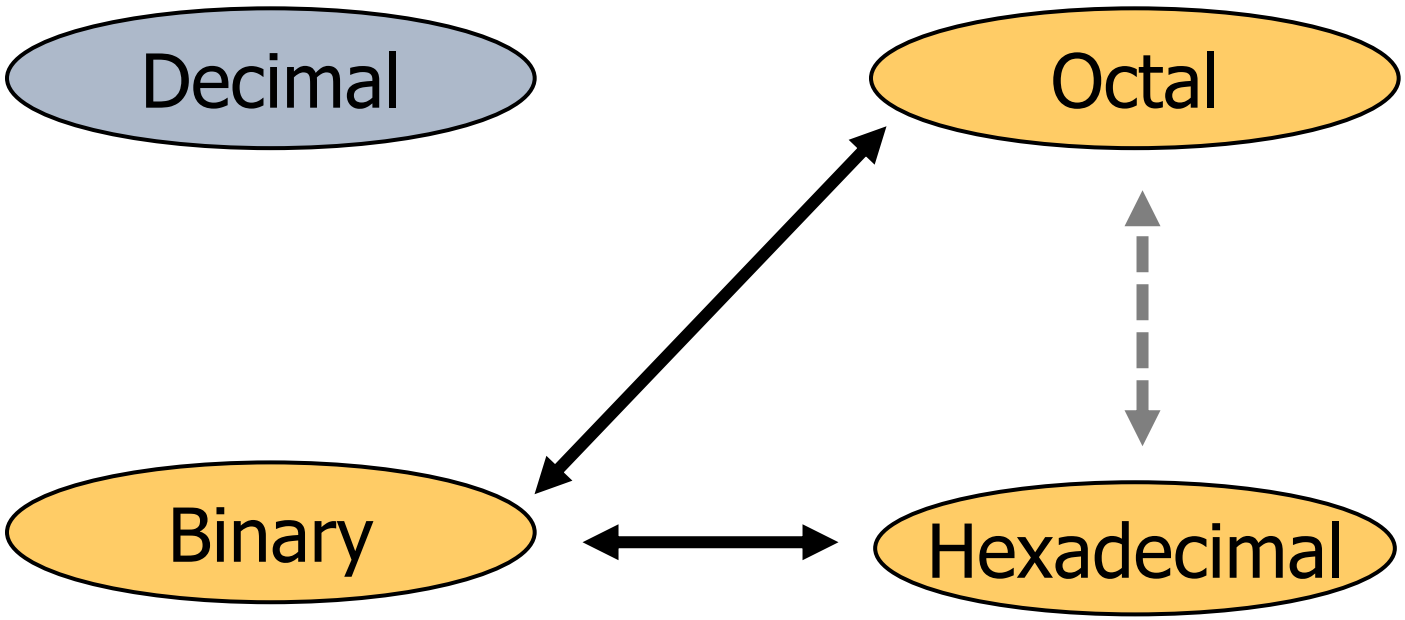
$2748_{10} = ABC_{16}$

0.75 ➔ 0.75 x 16 = 12.000

$0.75_{10} = 0.C_{16}$

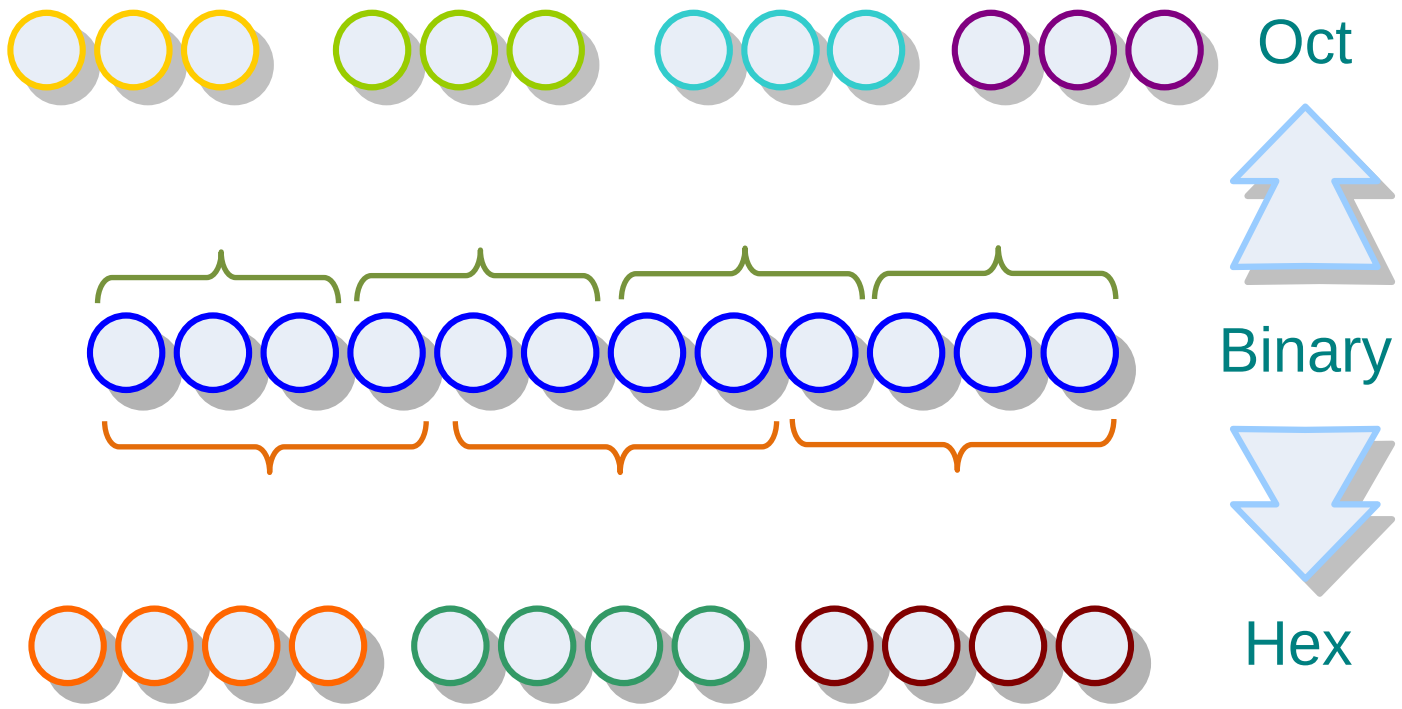
$(2748.75)_{10}=(ABC.C)_{16}$

การแปลงเลขฐาน 2, 8, 16



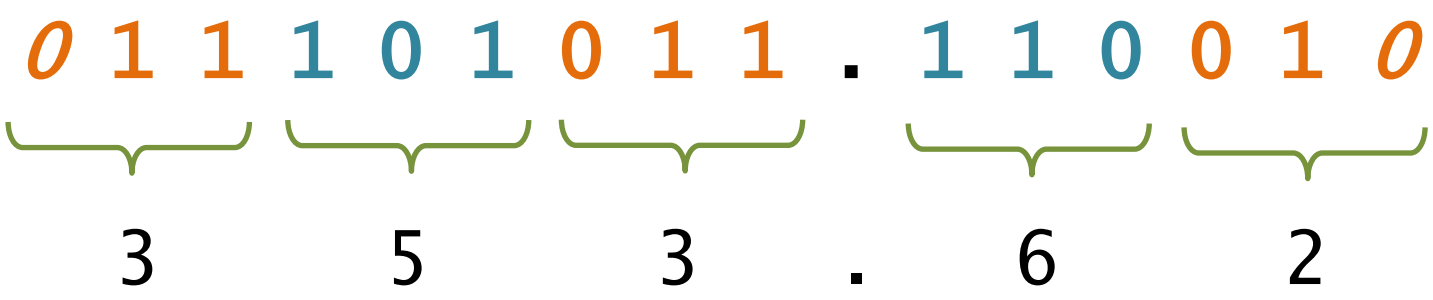
เลขฐานสอง ฐานแปด และฐานสิบหก

- เลขฐานแปด 1 ตัว สามารถแทนได้ด้วย เลขฐานสอง 3 ตัว
- เลขฐานสิบหก 1 ตัว สามารถแทนได้ด้วย เลขฐานสอง 4 ตัว



Binary to Octal

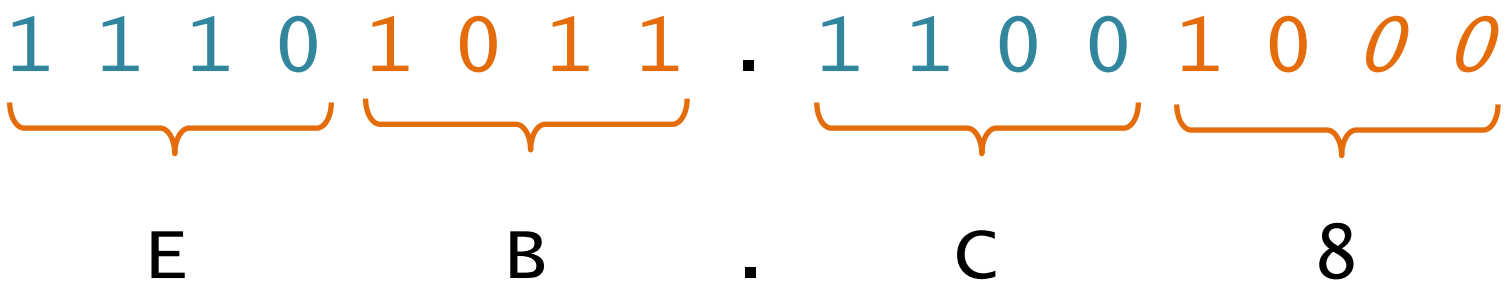
- $11101011.11001_2 = ?_8$



$11101011.11001_2 = 353.62_8$

Binary to Hexadecimal

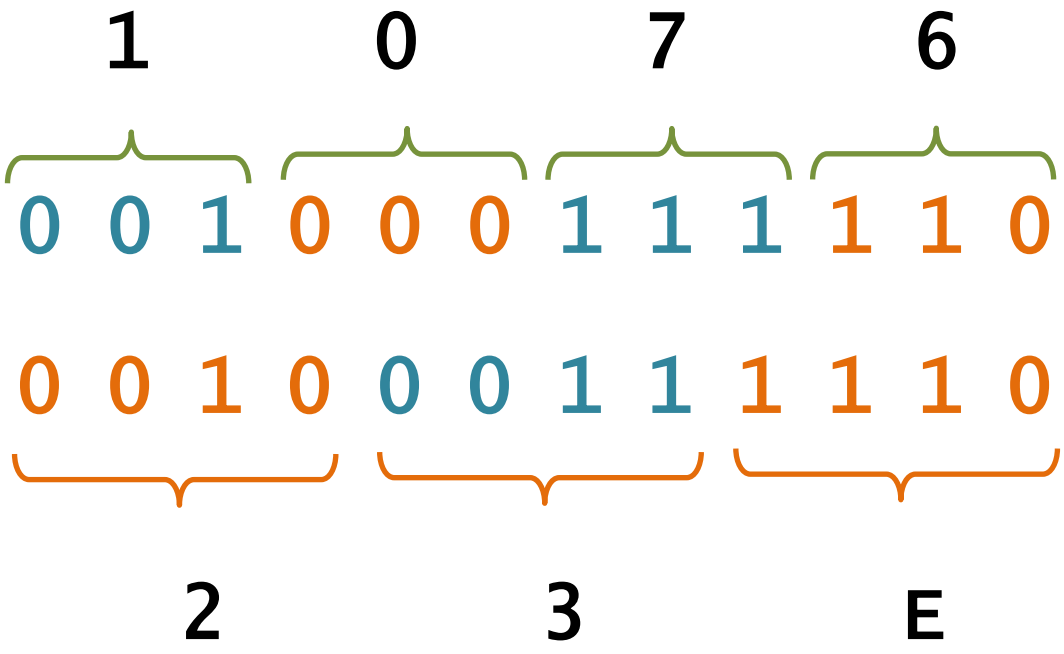
- $11101011.11001_2 = ?_{16}$



$11101011.11001_2 = EB.C8_{16}$

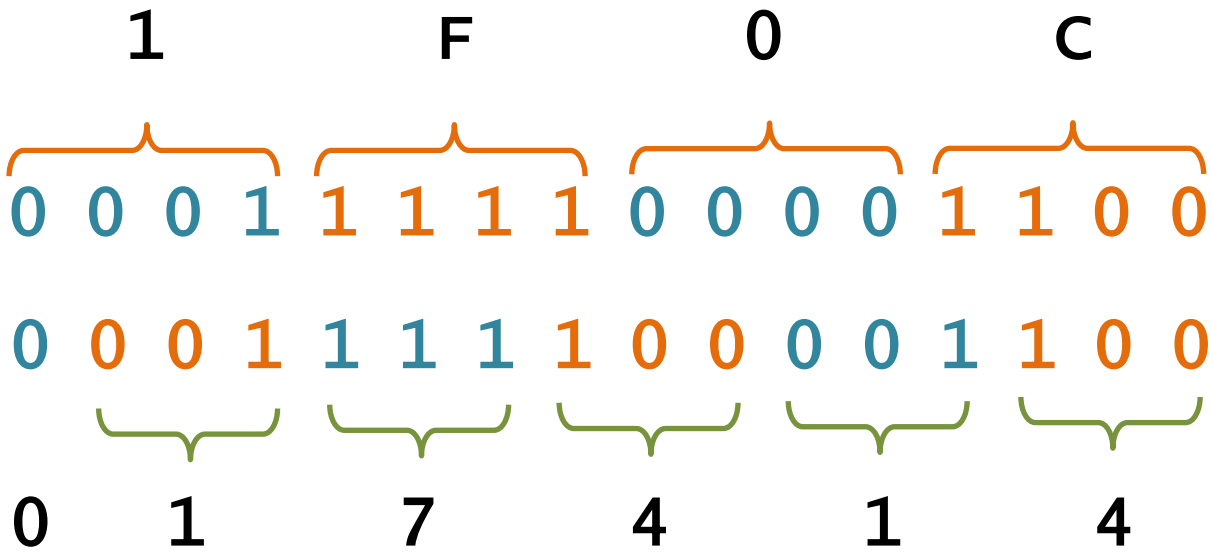
Octal to Hexadecimal

- $1076_8 = ?_{16}$



Hexadecimal to Octal

- $1F0C_{16} = ?_8$



$$1F0C_{16} = 17414_8$$

การบวกเลขฐานสอง

- $010111_2 + 011110_2 = 110101_2$

	1	1	1			
0	1	0	1	1	1	
0	1	1	1	1	0	+
1	1	0	1	0	1	

การลบเลขฐานสอง

- $110111_2 - 011110_2 = 11001_2$

```
    0  10
  1 1 0 1 1 1
  0 1 1 1 1 0   -
  -----------
    1 1 0 0 1
```

การจัดเก็บค่าตัวเลขลงในระบบคอมพิวเตอร์

- การจัดเก็บค่า **ตัวเลขจำนวนเต็ม (Integer)**
  - จะถูกจัดเก็บเป็นเลขฐานสอง
  - จัดเก็บเป็นตัวเลขแบบคิดเครื่องหมายและตัวเลขแบบไม่คิดเครื่องหมายได้
- ถ้าหากมีเลขฐานสองจำนวน n บิต จะสามารถเก็บตัวเลขแบบไม่คิดเครื่องหมายได้ในช่วง 0 ถึง $(2^n - 1)$
  - เช่นข้อมูลขนาด 1 ไบต์ (8 บิต) จัดเก็บตัวเลขได้ในช่วง 0 ถึง 255
- เช่นถ้าต้องการจัดเก็บข้อมูลลงไปในหน่วยความจำเช่น รีจิสเตอร์ที่ใช้จัดเก็บข้อมูลขนาด 8 บิตจะสามารถจัดเก็บค่าได้สูงสุด 2^8 หรือ 256 ค่า เริ่มตั้งแต่ 0 ถึง 255

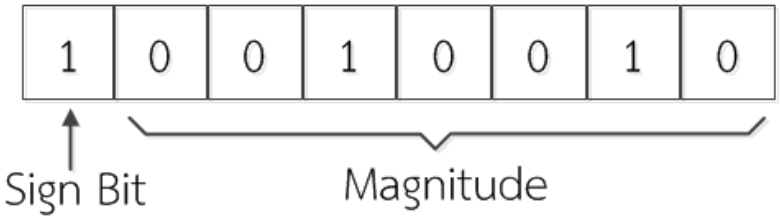
Unsigned Representation

- **การแทนค่าเลขจำนวนเต็มไม่รวมเครื่องหมาย**
  (Unsigned Representation)
  - เช่นข้อมูลขนาด 1 ไบต์ (8 บิต) จัดเก็บตัวเลขได้ในช่วง 0 ถึง 255

0000	0000	=	0
0000	0001	=	1
0010	1001	=	41
1000	0000	=	128
1111	1111	=	255

Sign-Magnitude Representation

- **การแทนค่าเลขจำนวนเต็มรวมเครื่องหมาย** (Sign-Magnitude Representation)
  - หมายถึงการจัดเก็บค่าตัวเลขที่ประกอบด้วยค่าบวกหรือลบ โดยในระบบคอมพิวเตอร์จะไม่ใช้เครื่องหมายบวกและลบ แต่จะพิจารณาจากบิตเครื่องหมาย (Sign Bit)
  - 8 บิตรวมบิตเครื่องหมายจะสามารถแทนค่าตัวเลขได้เพียงครึ่งหนึ่งเท่านั้น เท่ากับ 128 ตัว +127 ถึง -128
  - บิตเครื่องหมาย (Sign Bit) คือบิตซ้าย 0 คือค่าบวก / 1 คือค่าลบ



+18 = 0001 0010

-18 = 1001 0010

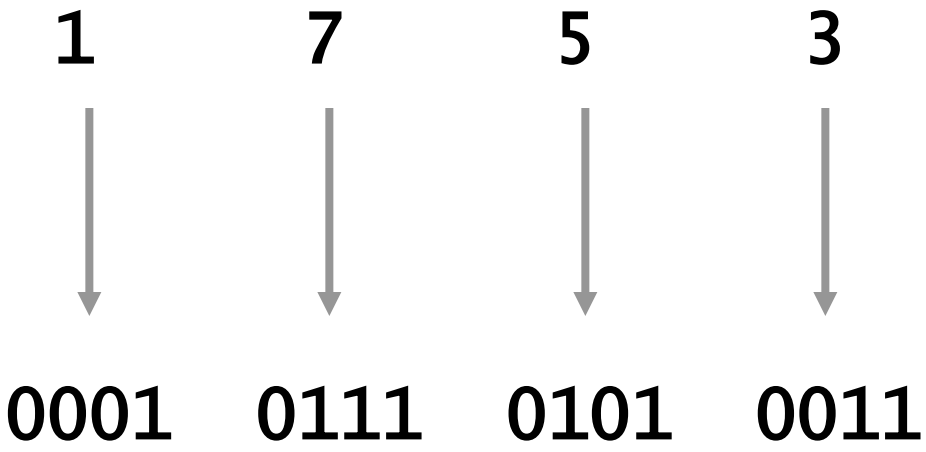
รหัส BCD-8421

- Binary Codes Decimal-8421
- เป็นการเขียนเลขฐานสองแทนเลขฐานสิบแต่ละดิจิต (ไม่ใช่การแปลงฐานสิบเป็นฐานสอง)
- 1 ดิจิตในฐานสิบจะแทนด้วยฐานสองจำนวน 4 ดิจิต
- สังเกต 1010, 1011, 1100, 1101, 1110, และ 1111 ไม่ถูกใช้

Decimal	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

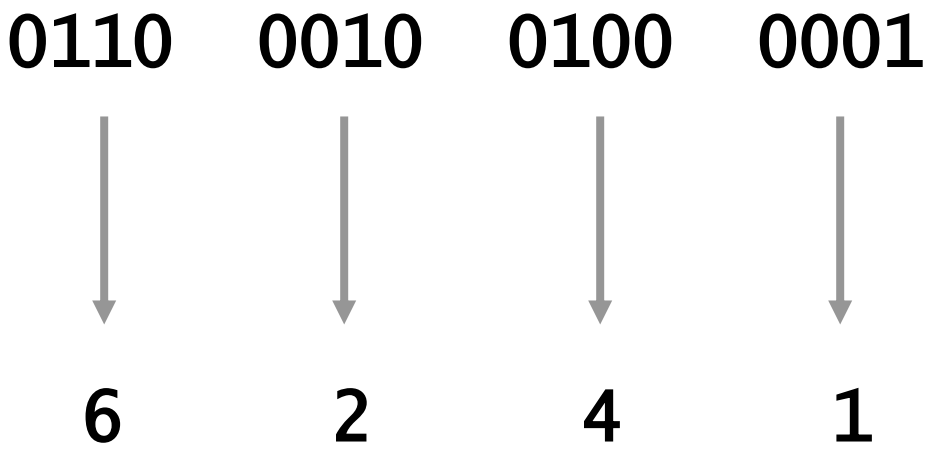
DEC to BCD

- 175310 = ? (in BCD)



BCD to DEC

- 0110001001000001_{BCD} = ?



รหัสเกินสาม (Excess-3 Code)

- Excess-3 Code จะมากกว่าค่าของ BCD 8421 อยู่ 3

Decimal **3** **5** **9**

Excess-3 **0110 1000 1100**

Decimal	BCD	Excess-3
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000
6	0110	1001
7	0111	1010
8	1000	1011
9	1001	1100

คอมพลีเมนต์ (Complement)

- **คอมพลีเมนต์** หรือ ส่วนเติมเต็ม คือ การกำหนดเลขฐานนั้น ๆ เพิ่มขึ้นสำหรับการลบ
- มี 2 แบบ คือ
  - (b-1)'s Complement
  - (b)'s Complement

คอมพลีเมนต์ในระบบเลขฐานสอง

- คอมพลีเมนต์ในระบบเลขฐานสอง มี 2 แบบ คือ
  - 1's Complement คือ การนำค่าสูงสุดในระบบเลขฐานสิบ ที่มีจำวนดิจิต เท่ากับตัวที่ต้องการหา มาตั้งแล้วลบด้วยค่าที่ต้องการหา

เช่น

```
1 1 1 1 1 1
1 1 0 0 1 0   -
0 0 1 1 0 1
```

1's Complement ของ 110010

  - 2's Complement คือ 1's Complement+1

เช่น

```
0 0 1 1 0 1
          1   +
0 0 1 1 1 0
```

2's Complement ของ 110010

การลบเลขด้วยวิธี 1's complement

- มีวิธีดังนี้

1. นำเอา **ตัวลบ** ไปทำ 1's
2. นำเอา **ตัวตั้ง** มาบวกกับ **ตัวลบ** (ที่ทำเป็น 1's แล้ว)
   (แต่จำนวนหลักของ**ตัวตั้ง**และ**ตัวลบ**ต้อง **เท่ากัน**)
3. ผลที่ได้จากการบวก
   - **ถ้ามีตัวทด** ให้นำไป**บวก**เพิ่มจากผลที่ได้ คำตอบจะมีมาเป็นค่าบวก
   - **ถ้าไม่มีตัวทด** ให้นำผลที่ได้ ไปทำ 1's อีกครั้งหนึ่ง คำตอบจะออกมาเป็นค่าลบ

การลบเลขด้วยวิธี 1's complement

- เช่น **110110** – **100001** ด้วยวิธี 1's complement
- หา 1's ของ **100001** คือ 011110
- นำเอา **ตัวตั้ง** มาบวกกับ **ตัวลบ** (ที่ทำเป็น 1's แล้ว)

```
  1 1 0 1 1 0
  0 1 1 1 1 0   +
1 0 1 0 1 0 0
```

- ผลที่ได้กรณีนี้ มีตัวทด ให้นำไปบวกเพิ่มจากผลที่ได้
  0 1 0 1 0 0 + 1 = 1 0 1 0 1

การลบเลขด้วยวิธี 2's complement

- มีวิธีดังนี้

1. นำเอา **ตัวลบ** ไปทำ 2's
2. นำเอา **ตัวตั้ง** มาบวกกับ **ตัวลบ** (ที่ทำเป็น 2's แล้ว)
   (แต่จำนวนหลักของ**ตัวตั้ง**และ**ตัวลบ**ต้อง **เท่ากัน**)
3. ผลที่ได้จากการบวก
   - **ถ้ามีตัวทด** ให้ตัดตัวทดทิ้ง คำตอบจะออกมาเป็นค่าบวก
   - **ถ้าไม่มีตัวทด** ให้นำผลที่ได้ ไปทำ 2's อีกครั้งหนึ่ง คำตอบจะออกมาเป็นค่าลบ

การลบเลขด้วยวิธี 2's complement

- เช่น **110110** – **100001** ด้วยวิธี 2's complement
- หา 2's ของ **100001** คือ 011111
- นำเอา **ตัวตั้ง** มาบวกกับ **ตัวลบ** (ที่ทำเป็น 2's แล้ว)

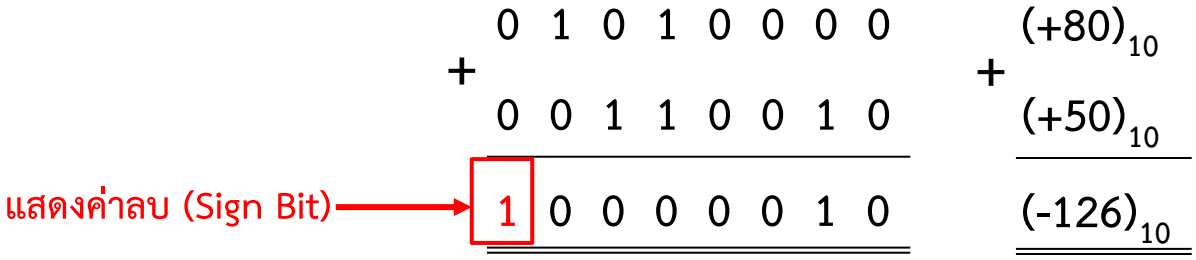
```
  1 1 0 1 1 0
  0 1 1 1 1 1   +
1 0 1 0 1 0 1
```

- ผลที่ได้กรณีนี้ให้ตัดตัวทดทิ้ง คำตอบจะออกมาเป็นค่าบวก

  = 1 0 1 0 1

โอเวอร์โฟลว์ (Overflow)

- คือผลลัพธ์ของตัวเลขที่มีค่าเกินกว่าที่จะรับได้ ทำให้การแสดงผลค่าตัวเลขที่เป็นผลลัพธ์เกิดข้อผิดพลาดขึ้น
  - ถ้าชุดตัวเลขสองค่าที่มีค่า Sign เหมือนกัน (Large Magnitudes) มาบวกกัน จะเกิดปรากฏการณ์โอเวอร์โฟลว์ขึ้น
  - เช่น (+80) และ (+50) โดยมีรูปแบบ 8 บิตพร้อมบิตเครื่องหมาย ผลลัพธ์ที่ได้ควรจะเป็น (+130) แต่ผลลัพธ์ที่ได้จากการคำนวณจะได้ (-126)



  - ค่าบวกสูงสุดของ 8 บิตรวมเครื่องหมายใน Two's Complement คือ (+127)

รหัสแทนข้อมูล

- **ANSI** (American National Standards Institute)
  - ใช้แทนตัวเลข ตัวอักษร และ สัญลักษณ์ต่าง ๆ
  - Standard ASCII มีขนาด 7 บิต มีทั้งหมด 27 เท่ากับ 128 ตัวอักษร
  - Extended ASCII (IBM ASCII) มีขนาด 8 บิต มีทั้งหมด 28 เท่ากับ 256 ตัวอักษร (ปัจจุบัน)

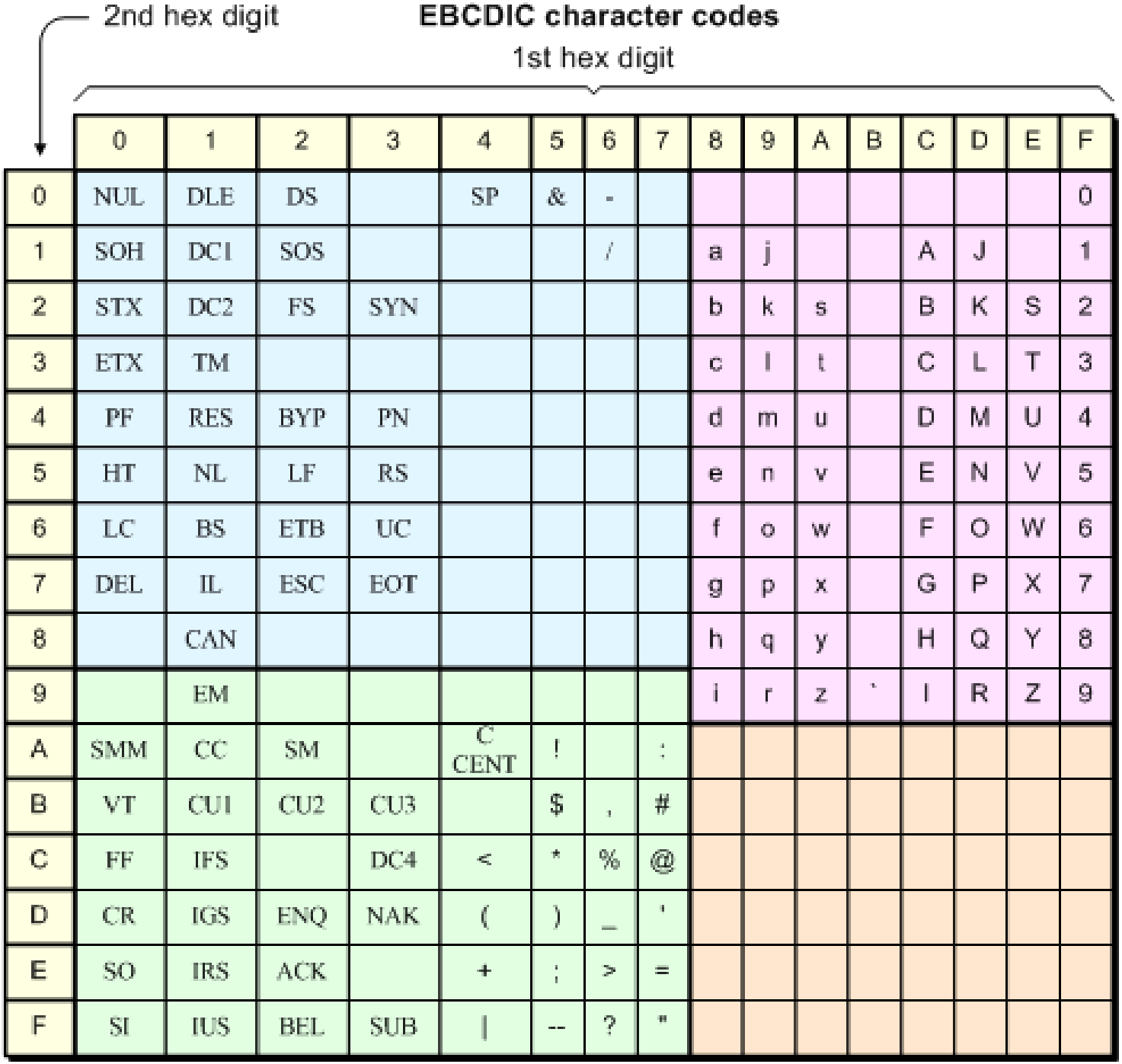
Dec	Hx	Oct	Char	Dec	Hx	Oct	Chr	Dec	Hx	Oct	Chr	Dec	Hx	Oct	Chr
0	0	000	NUL (null)	32	20	040	Space	64	40	100	@	96	60	140	`
1	1	001	SOH (start of heading)	33	21	041	!	65	41	101	A	97	61	141	a
2	2	002	STX (start of text)	34	22	042	"	66	42	102	B	98	62	142	b
3	3	003	ETX (end of text)	35	23	043	#	67	43	103	C	99	63	143	c
4	4	004	EOT (end of transmission)	36	24	044	$	68	44	104	D	100	64	144	d
5	5	005	ENQ (enquiry)	37	25	045	%	69	45	105	E	101	65	145	e
6	6	006	ACK (acknowledge)	38	26	046	&	70	46	106	F	102	66	146	f
7	7	007	BEL (bell)	39	27	047	'	71	47	107	G	103	67	147	g
8	8	010	BS (backspace)	40	28	050	(	72	48	110	H	104	68	150	h
9	9	011	TAB (horizontal tab)	41	29	051	)	73	49	111	I	105	69	151	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	74	4A	112	J	106	6A	152	j
11	B	013	VT (vertical tab)	43	2B	053	+	75	4B	113	K	107	6B	153	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	76	4C	114	L	108	6C	154	l
13	D	015	CR (carriage return)	45	2D	055	-	77	4D	115	M	109	6D	155	m
14	E	016	SO (shift out)	46	2E	056	.	78	4E	116	N	110	6E	156	n
15	F	017	SI (shift in)	47	2F	057	/	79	4F	117	O	111	6F	157	o
16	10	020	DLE (data link escape)	48	30	060	0	80	50	120	P	112	70	160	p
17	11	021	DC1 (device control 1)	49	31	061	1	81	51	121	Q	113	71	161	q
18	12	022	DC2 (device control 2)	50	32	062	2	82	52	122	R	114	72	162	r
19	13	023	DC3 (device control 3)	51	33	063	3	83	53	123	S	115	73	163	s
20	14	024	DC4 (device control 4)	52	34	064	4	84	54	124	T	116	74	164	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	85	55	125	U	117	75	165	u
22	16	026	SYN (synchronous idle)	54	36	066	6	86	56	126	V	118	76	166	v
23	17	027	ETB (end of trans. block)	55	37	067	7	87	57	127	W	119	77	167	w
24	18	030	CAN (cancel)	56	38	070	8	88	58	130	X	120	78	170	x
25	19	031	EM (end of medium)	57	39	071	9	89	59	131	Y	121	79	171	y
26	1A	032	SUB (substitute)	58	3A	072	:	90	5A	132	Z	122	7A	172	z
27	1B	033	ESC (escape)	59	3B	073	;	91	5B	133	[	123	7B	173	{
28	1C	034	FS (file separator)	60	3C	074	<	92	5C	134	\	124	7C	174	|
29	1D	035	GS (group separator)	61	3D	075	=	93	5D	135	]	125	7D	175	}
30	1E	036	RS (record separator)	62	3E	076	>	94	5E	136	^	126	7E	176	~
31	1F	037	US (unit separator)	63	3F	077	?	95	5F	137	_	127	7F	177	DEL

รหัสแทนข้อมูล

- **รหัส EBCDIC**

  (Extended Binary Code Decimal Interchange Code)

  - ถูกพัฒนาขึ้นโดยบริษัทไอบีเอ็มเพื่อใช้ในเครื่องไอบีเอ็มเมนเฟรมและมินิคอมพิวเตอร์
  - แทนสัญลักษณ์ได้ทั้งหมด 256 ตัวอักษร (28)
  - ใช้ 8 บิตแทนตัวอักษร 1 ตัวอักษร



2nd hex digit \ 1st hex digit	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	DLE	DS		SP	&	-									0
1	SOH	DC1	SOS				/		a	j			A	J		1
2	STX	DC2	FS	SYN					b	k	s		B	K	S	2
3	ETX	TM							c	l	t		C	L	T	3
4	PF	RES	BYP	PN					d	m	u		D	M	U	4
5	HT	NL	LF	RS					e	n	v		E	N	V	5
6	LC	BS	ETB	UC					f	o	w		F	O	W	6
7	DEL	IL	ESC	EOT					g	p	x		G	P	X	7
8		CAN							h	q	y		H	Q	Y	8
9		EM							i	r	z	`	I	R	Z	9
A	SMM	CC	SM		C CENT	!		:								
B	VT	CU1	CU2	CU3		$	,	#								
C	FF	IFS		DC4	<	*	%	@								
D	CR	IGS	ENQ	NAK	(	)	_	'								
E	SO	IRS	ACK		+	;	>	=								
F	SI	IUS	BEL	SUB	|	--	?	"								

รหัสแทนข้อมูล

- **รหัส UNICODE**
  - ถูกพัฒนาขึ้นมาล่าสุด เพื่อนำมาใช้กำหนดตัวอักษรของประเทศอื่นๆ เพื่อให้เพียงพอต่อการใช้งานในแต่ละภาษา นอกเหนือจากภาษาอังกฤษ เช่นภาษาจีน ภาษาญี่ปุ่นที่มีตัวอักษรหรือสัญลักษณ์เกินกว่า 256 ตัวอักษร
  - แทนตัวอักษรและสัญลักษณ์ได้ทั้งหมด 65536 ตัวอักษร (216)
  - มีขนาด 16 บิตหรือ 2 ไบต์
  - มีใช้ในระบบปฏิบัติการ Windows และในระบบปฏิบัติการ UNIX บางรุ่น

Note

หน่วยคำนวณทางคณิตศาสตร์และตรรกะ (Arithmetic and Logic Unit)

Chapter 3

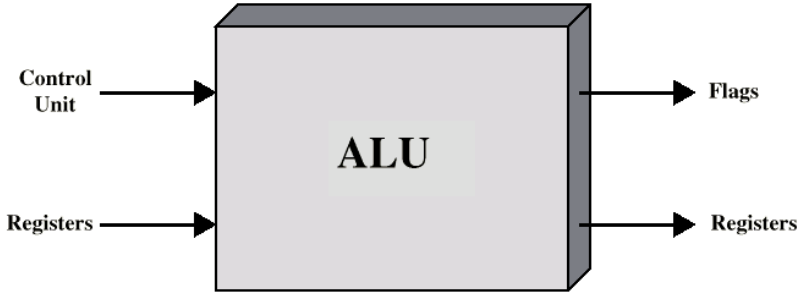
Computer Organization and Architecture

ปรับปรุงจากเอกสารของ
อ. ณัฐวุฒิ สร้อยดอกสน (NSD)

อ.เอิญ สุริยะฉาย (ENS)
ภาควิชาวิทยาการคอมพิวเตอร์และสารสนเทศ
มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าพระนครเหนือ

หน่วยคำนวณทางคณิตศาสตร์และตรรกะ

- หน่วยคำนวณทางคณิตศาสตร์และตรรกะ (Arithmetic and Logic Unit : **ALU**)
  - ทำหน้าที่ประมวลผลทางคณิตศาสตร์ โดยมีหน่วยควบคุม (Control Unit) ทำหน้าที่ควบคุมการทำงานว่าจะให้ ALU ประมวลผลในแบบใด
  - โดยนำข้อมูลจากรีจิสเตอร์มาประมวลผลและเก็บผลลัพธ์ที่ได้จากการประมวลผลไว้ในรีจิสเตอร์
  - และใช้แฟล็ก (Flags) ในการจัดเก็บสถานะที่ได้จากการประมวลผล
    - เช่นถ้าผลลัพธ์ที่ได้จากการประมวลผลมีค่ามากเกินกว่าที่รีจิสเตอร์จะเก็บได้ ก็จะใช้บิตแฟล็กนี้เป็นตัวบอกสถานะของผลลัพธ์ที่เกิดขึ้น



Arithmetic and Logic Unit

การลบเลขด้วยวิธี 2's complement

- เช่น **0000 1110** – **0001 0100** ด้วยวิธี 2's complement
- หา 2's ของ **0001 0100** คือ 1110 1100
- นำเอา **ตัวตั้ง** มาบวกกับ **ตัวลบ** (ที่ทำเป็น 2's แล้ว)

~~1~~

(14)	0000	1110
(-20)	1110	1100 +
	1111	1010

- ให้นำผลที่ได้ ไปทำ 2's อีกครั้งหนึ่ง คำตอบจะออกมาเป็นค่าลบ

= **0000 0110**

เลขบวกสองจำนวน

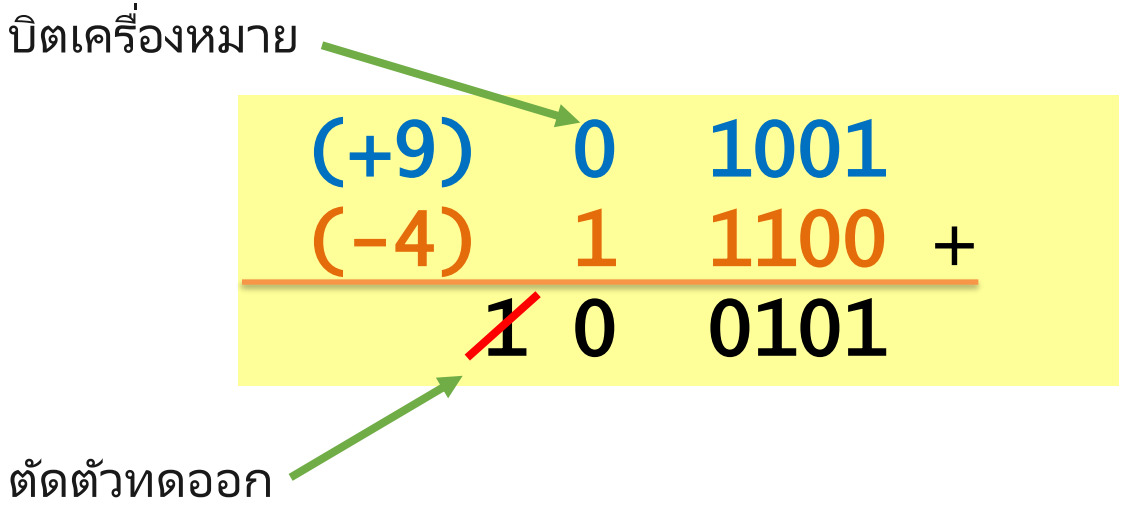
- จะเป็นการบวกค่าเลขบวกสองค่าเข้าด้วยกัน
- ตัวอย่างเช่น การบวกค่า +9 กับ +4 สามารถทำได้ดังนี้

(+9)	0	1001	
(+4)	0	0100	+
	0	1101	

บิตเครื่องหมาย

เลขบวกกับค่าลบของเลขที่น้อยกว่า

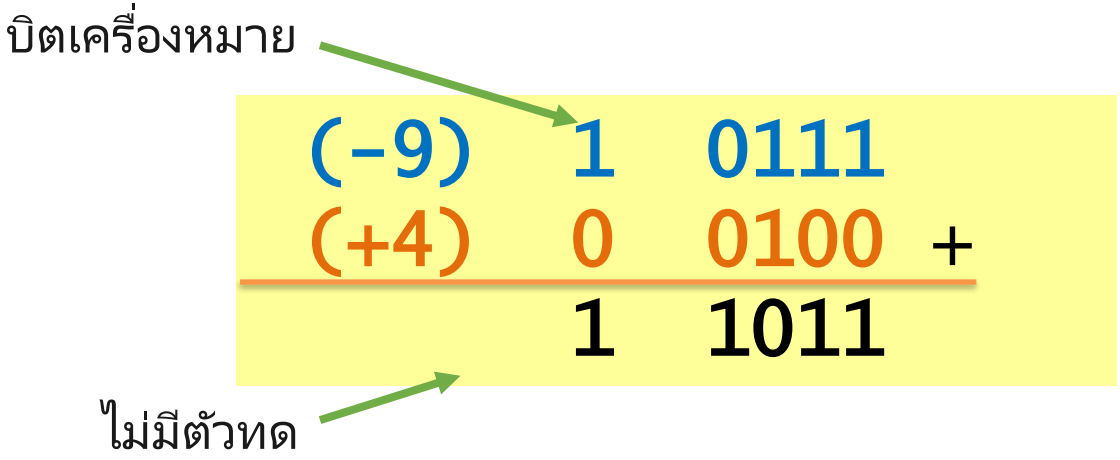
- เช่น การบวกกันระหว่างเลข +9 กับ -4
- โดย -4 มาทำ 2's complement



- บิตเครื่องหมายเป็นศูนย์ ผลลัพธ์จะมีค่าเป็นบวก ตัดตัวทดออก
- ตอบ +5

เลขบวกกับค่าลบของเลขที่มากกว่า

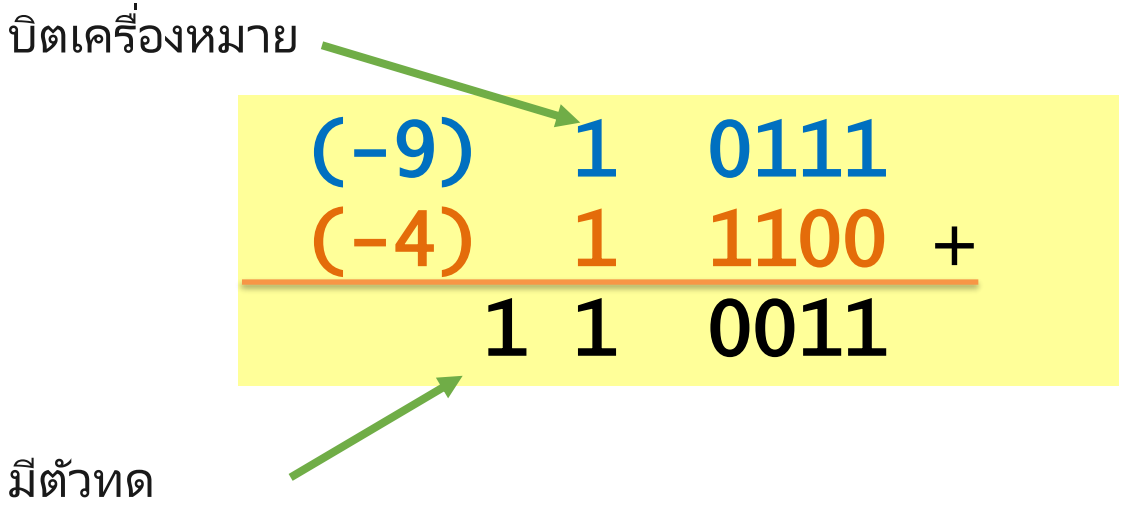
- เช่น การบวกกันระหว่างเลข -9 กับ +4
- โดย -9 มาทำ 2's complement



- ผลลัพธ์ที่ได้จะเป็นเลขลบ ไปทำ 2's อีกครั้งหนึ่ง
- ตอบ -5

เลขลบสองจำนวน

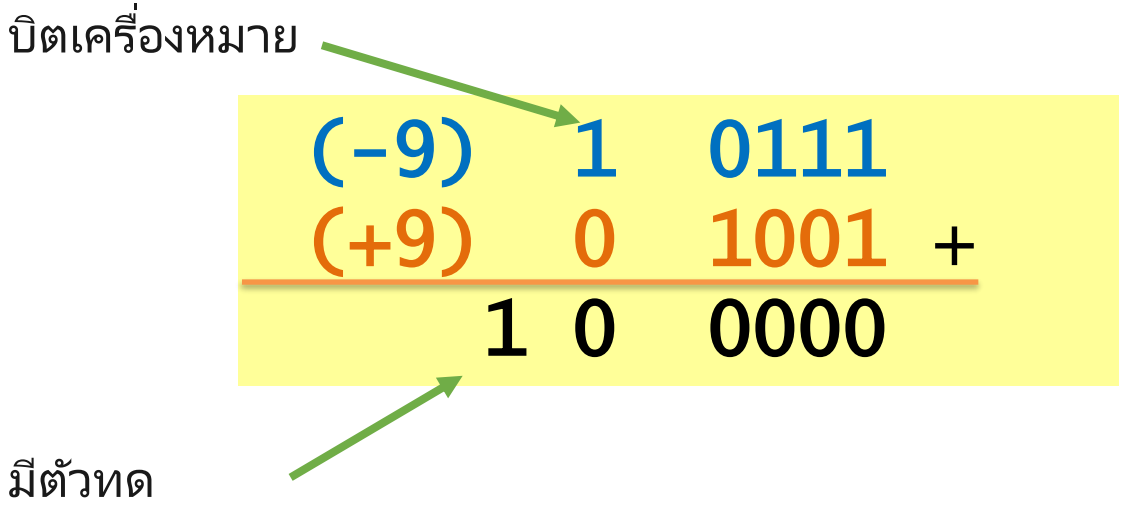
- เช่น การบวกกันระหว่างเลข -9 กับ -4
- โดย -9 และ -4 มาทำ 2's complement



- ผลลัพธ์ที่ได้จะเป็นเลขลบ ไปทำ 2's อีกครั้งหนึ่ง
- ตอบ -13

เลขเดียวกันแต่เครื่องหมายตรงกันข้าม

- เช่น การบวกกันระหว่างเลข -9 กับ -9
- โดย -9 มาทำ 2's complement



- ตอบ 0

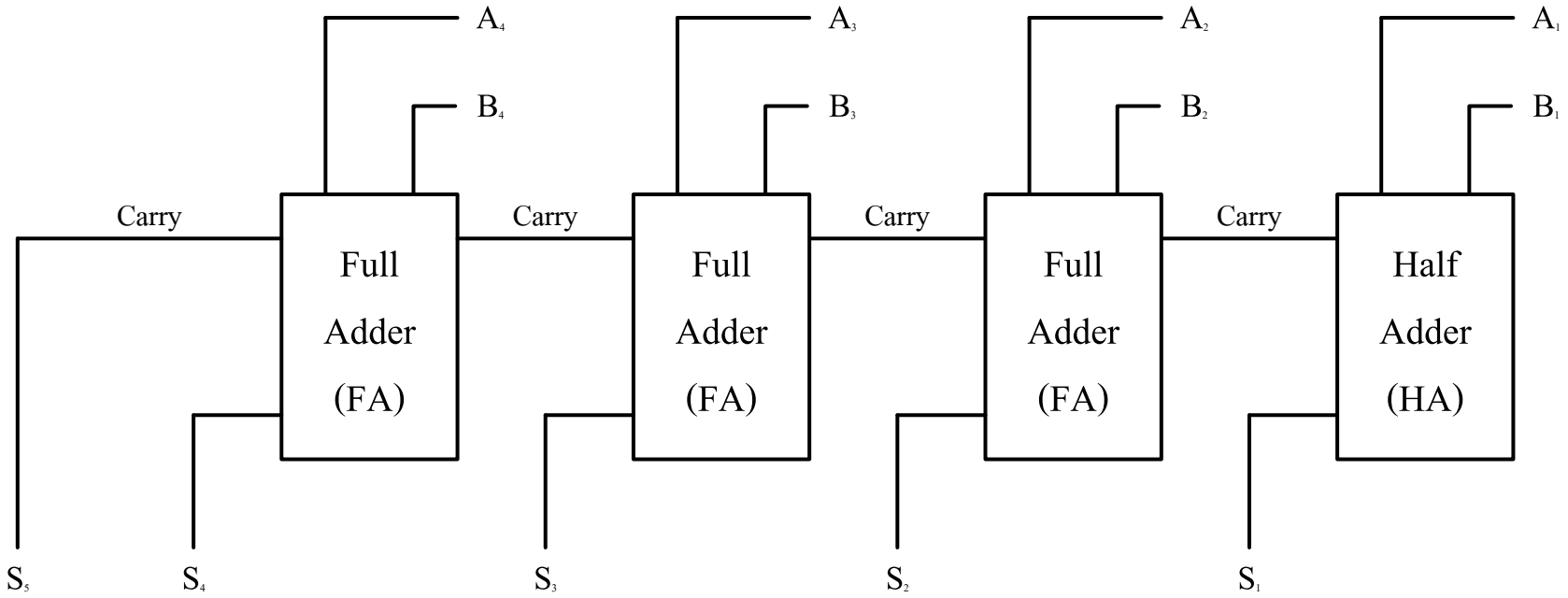
โอเวอร์โฟลว์ (Overflow)

- เกิดเมื่อนำตัวเลขสองจำนวนมาบวกหรือลบกัน โดยผลลัพธ์ที่ได้อาจมีค่ามากเกินกว่าจะจำนวนบิตจะสามารถจัดเก็บได้
  - เช่น 64 +96 โดยใช้เลขฐานสอง 8 บิตแบบคิดเครื่องหมาย

```
(+64)   0100 0000
(+96)   0110 0000 +
        1010 0000
```

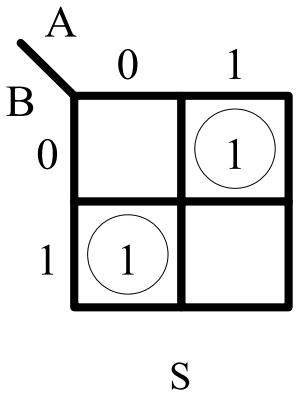
  - ผลลัพธ์ที่ได้บิตเครื่องหมายจะเป็นลบ ซึ่งผิด เพราะว่าค่า +160 ไม่สามารถใช้ข้อมูล 8 บิตเก็บได้ (8 บิตเก็บได้สูงสุดเพียง +127)
  - ใน ALU ของไมโครโปรเซสเซอร์ เมื่อมีการเกิดโอเวอร์โฟลว์ขึ้นจะแสดงออกมาทาง**แฟล็ก** เพื่อบอกว่าการประมวลผลนั้นเกิดการโอเวอร์โฟลว์

วงจรลอจิกของวงจรบวก

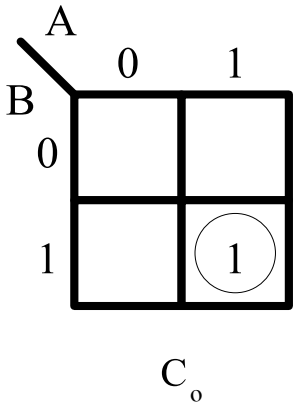


วงจร Half Adder

- วงจร Half Adder คือวงจรที่ใช้ในการบวกเลข Binary 2 Bits เข้าด้วยกัน โดย Truth Table จะมี Input เป็นเลข Binary 2 หลัก และมี 2 Outputs คือผลบวก (Sum) และตัวทด (Carry)



Input		Output		
A	B	Sum	Carry	
0	0	0	0	0 + 0 = 0 ทด 0
0	1	1	0	0 + 1 = 1 ทด 0
1	0	1	0	1 + 0 = 1 ทด 0
1	1	0	1	1 + 1 = 0 ทด 1

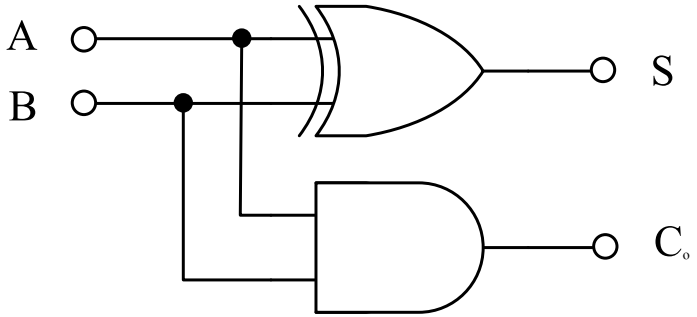


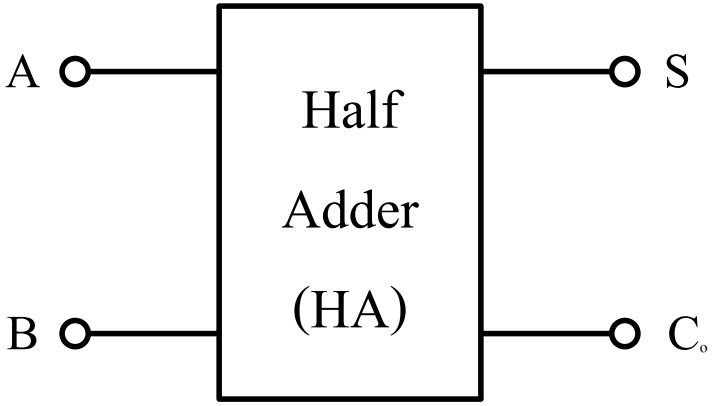
$$S = \overline{A}B + A\overline{B} = A \oplus B$$

$$C = AB$$

วงจร Half Adder

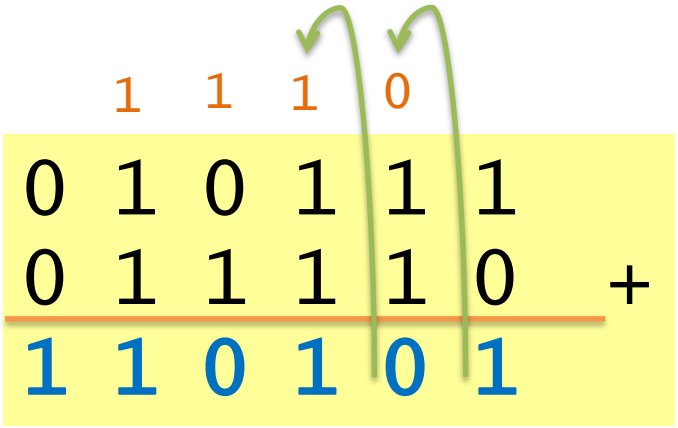
- การต่อวงจร Half-Adder





วงจร Full Adder

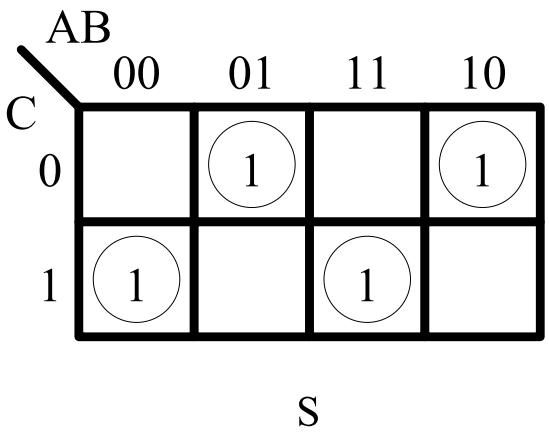
- คือวงจรที่ใช้บวกเลข Binary 2 Bits และตัวทดอีก 1 Bit รวมเป็น 3 Bits เข้าด้วยกัน โดย Truth Table จะมี Input เป็นเลข Binary 2 หลัก และมี Carry in (C_{in}) อีก 1 ตัว และมี 2 Outputs คือผลบวก (Sum) และตัวทด Carry out (C_{out})

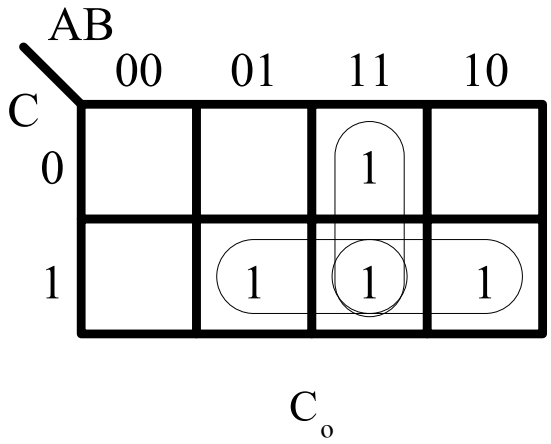


Input			Output		
A	B	C_{in}	Sum	C_{out}	
0	0	0	0	0	0 + 0 + 0 = 0 ทด 0
0	0	1	1	0	0 + 0 + 1 = 1 ทด 0
0	1	0	1	0	0 + 1 + 0 = 1 ทด 0
0	1	1	0	1	0 + 1 + 1 = 0 ทด 1
1	0	0	1	0	1 + 0 + 0 = 1 ทด 0
1	0	1	0	1	1 + 0 + 1 = 0 ทด 1
1	1	0	0	1	1 + 1 + 0 = 0 ทด 1
1	1	1	1	1	1+ 1 + 1 = 1 ทด 1

วงจร Full Adder

- นำค่าจากตารางไปสร้าง Karnaugh Map



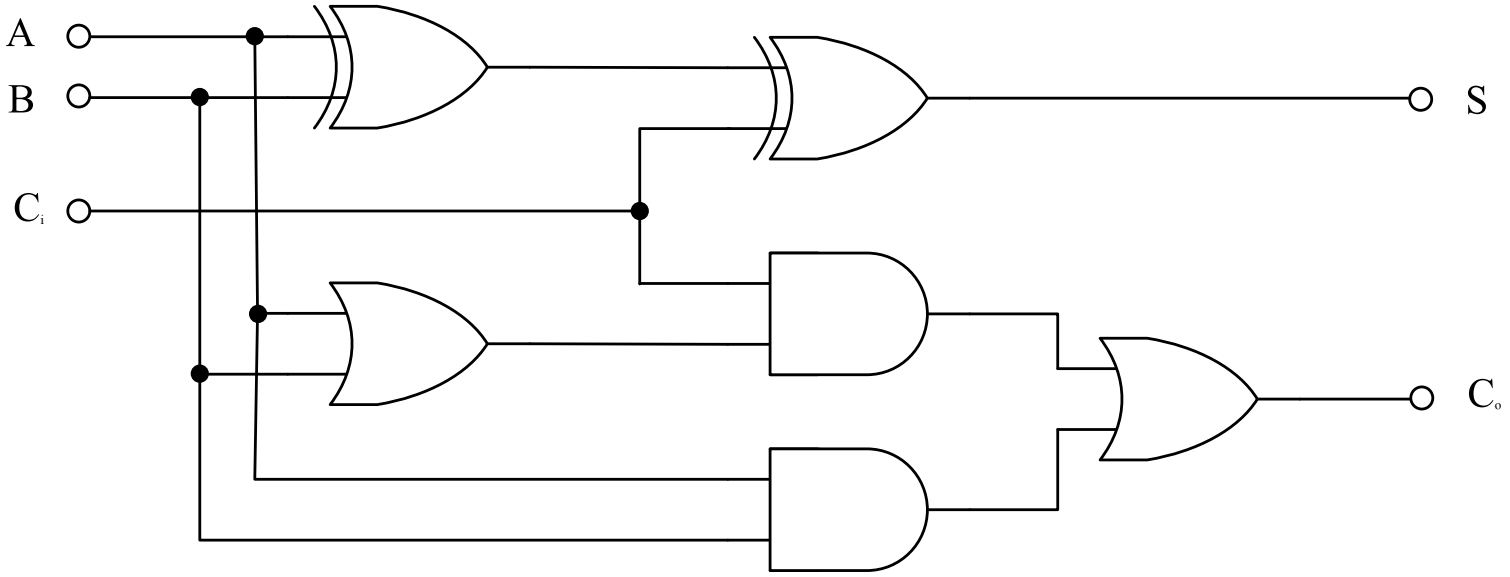


$$S = A \oplus B \oplus C_i$$

$$C_o = AB + (A + B)C_i$$

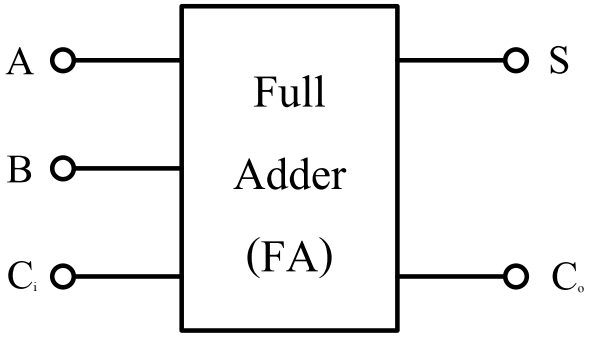
วงจร Full Adder

- การต่อวงจร Full Adder



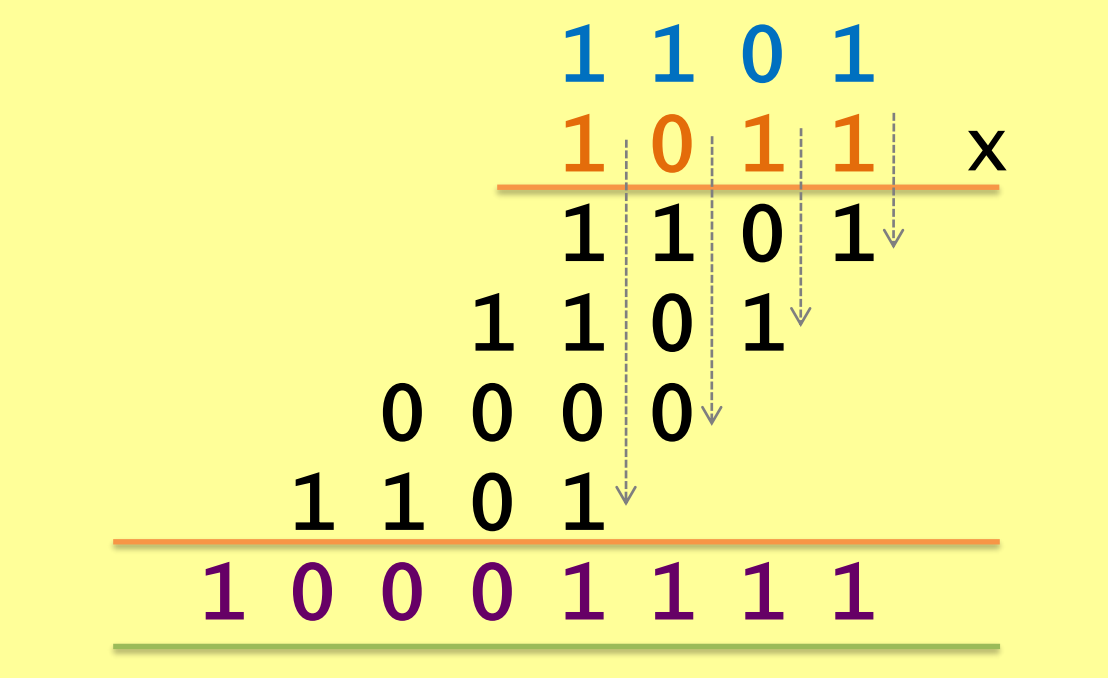
$$S = A \oplus B \oplus C_i$$

$$C_o = AB + (A + B)C_i$$



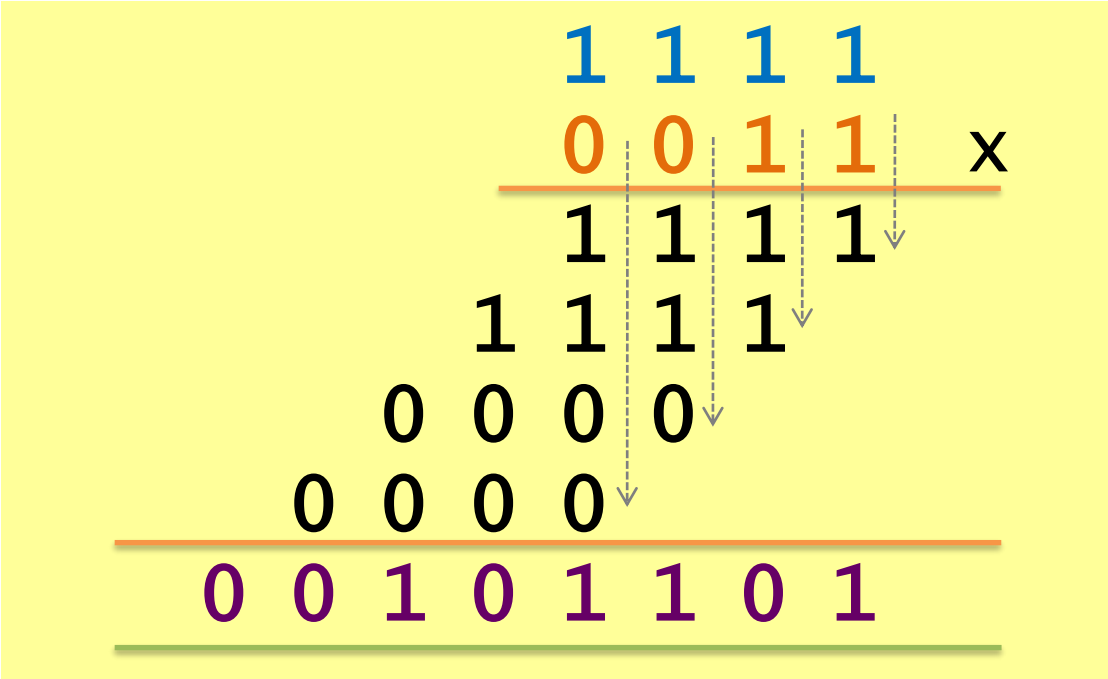
การคูณ (Multiplication)

- การคูณเลขจำนวนเต็มที่ไม่คิดเครื่องหมาย
  - ตัวคูณ (Multiplier) และ ตัวตั้ง (Multiplicand)
  - การคูณจำนวนเต็มจำนวน n บิต จะได้ผลคูณที่มีขนาดสูงสุดเป็น 2n บิตเสมอ



การคูณเลขจำนวนเต็มที่มีเครื่องหมาย

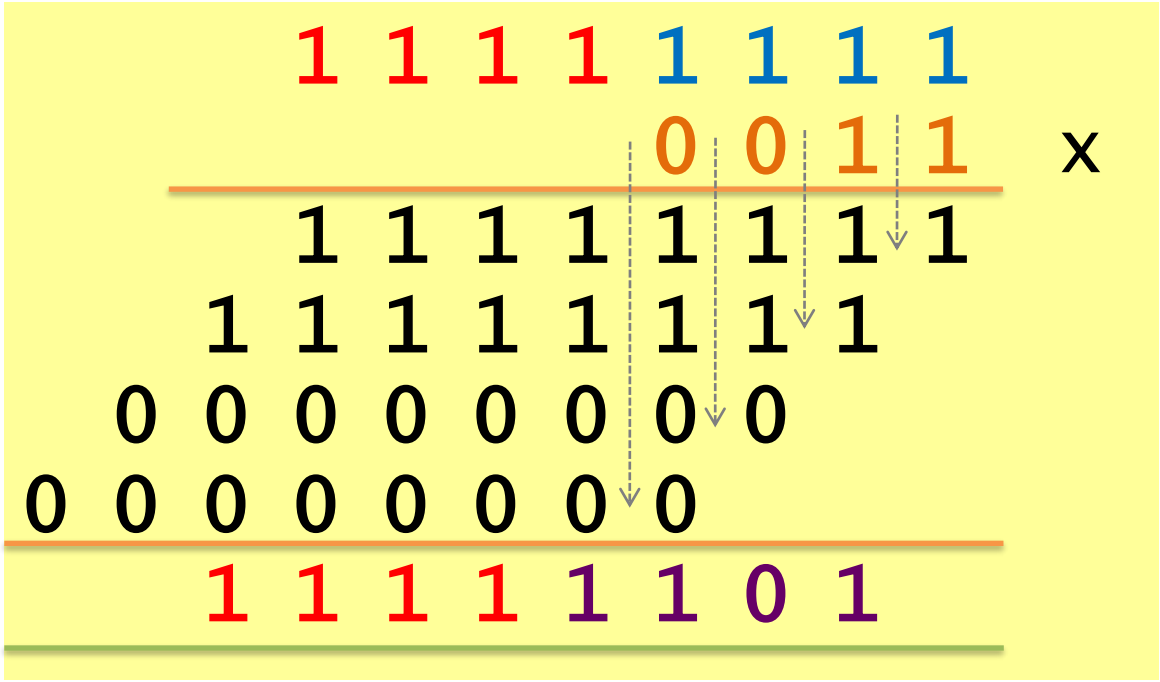
- ตัวอย่างการคูณ (-1) x (+3) โดยใช้ 2's Complement



- ผลลัพธ์ที่ได้จากการคูณ (-1) x (+3) = (+45) เป็นคำตอบที่ไม่ถูกต้อง เนื่องจากบิตเครื่องหมายไม่ได้มีการขยายบิตออกไป

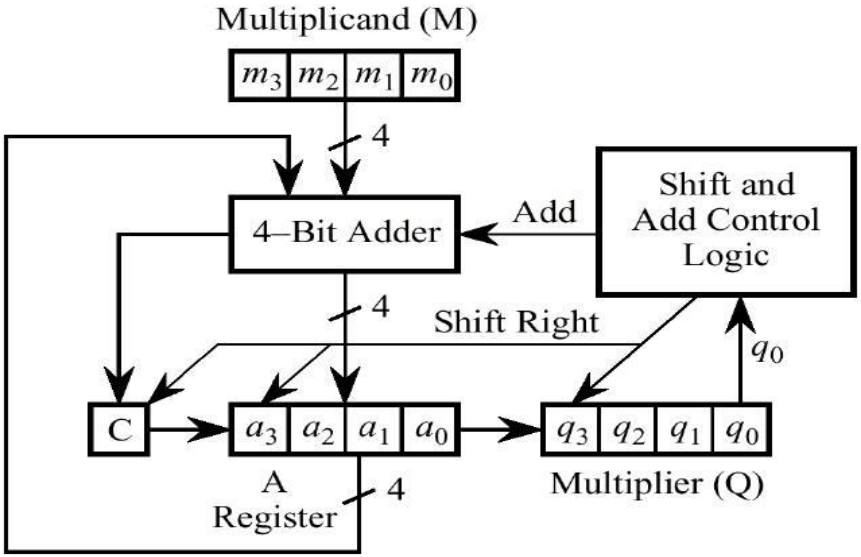
การคูณเลขจำนวนเต็มที่มีเครื่องหมาย

- **วิธีการแก้ไข** ทำได้โดยขยายผลคูณย่อยออกไป มีเพียง 8 บิตขวาสุดของผลลัพธ์ที่คงค่าไว้ ถ้าทั้งตัวตั้งและตัวคูณเป็นจำนวนลบทั้งคู่จะมีการขยายบิตเครื่องหมาย แต่ยังคงค่า 8 บิตขวาสุดของผลลัพธ์ที่ยังคงค่าอยู่



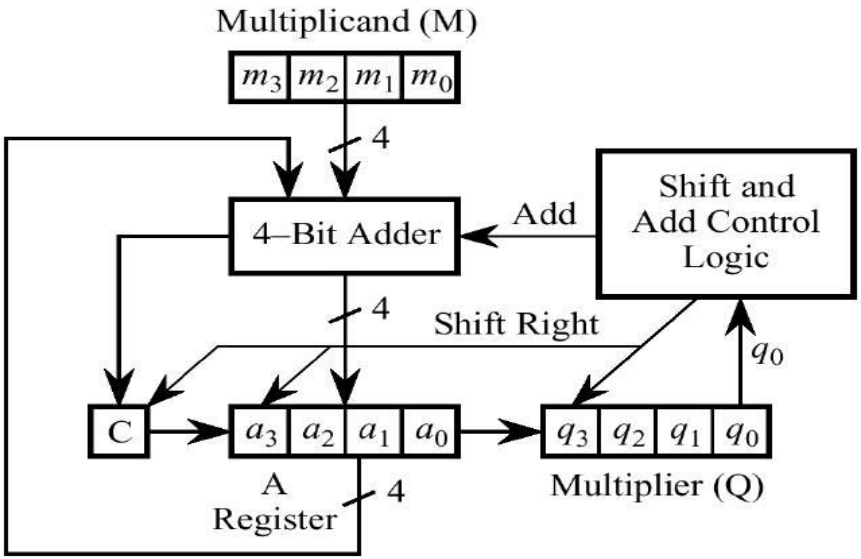
ฮาร์ดแวร์สำหรับการคูณ

- ตัวอย่างการคูณด้วยตัวเลขขนาด 4 บิต
  - ตัวตั้ง (Multiplicand) จะจัดเก็บลงในรีจิสเตอร์ M
  - ตัวคูณ (Multiplier) จะจัดเก็บลงในรีจิสเตอร์ Q
  - รีจิสเตอร์ A และ C จะถูกกำหนดค่าเริ่มต้นเป็นศูนย์
  - ระหว่างการคูณนั้น จะพิจารณาจากบิตขวาสุด (Rightmost Bit)



ฮาร์ดแวร์สำหรับการคูณ

- จากกระบวนการคูณ
  - ระหว่างการคูณนั้น จะพิจารณาจากบิตขวาสุด (Rightmost Bit)
    - ถ้ามีค่าเป็น 1 ให้นำค่าในรีจิสเตอร์ M และ A มาบวกกันและเก็บในรีจิสเตอร์ A จากนั้นใช้บิต 0 ดันเพื่อเลื่อนขวา
    - ถ้ามีค่าเป็น 0 ให้ใช้บิต 0 ดันเพื่อเลื่อนขวา

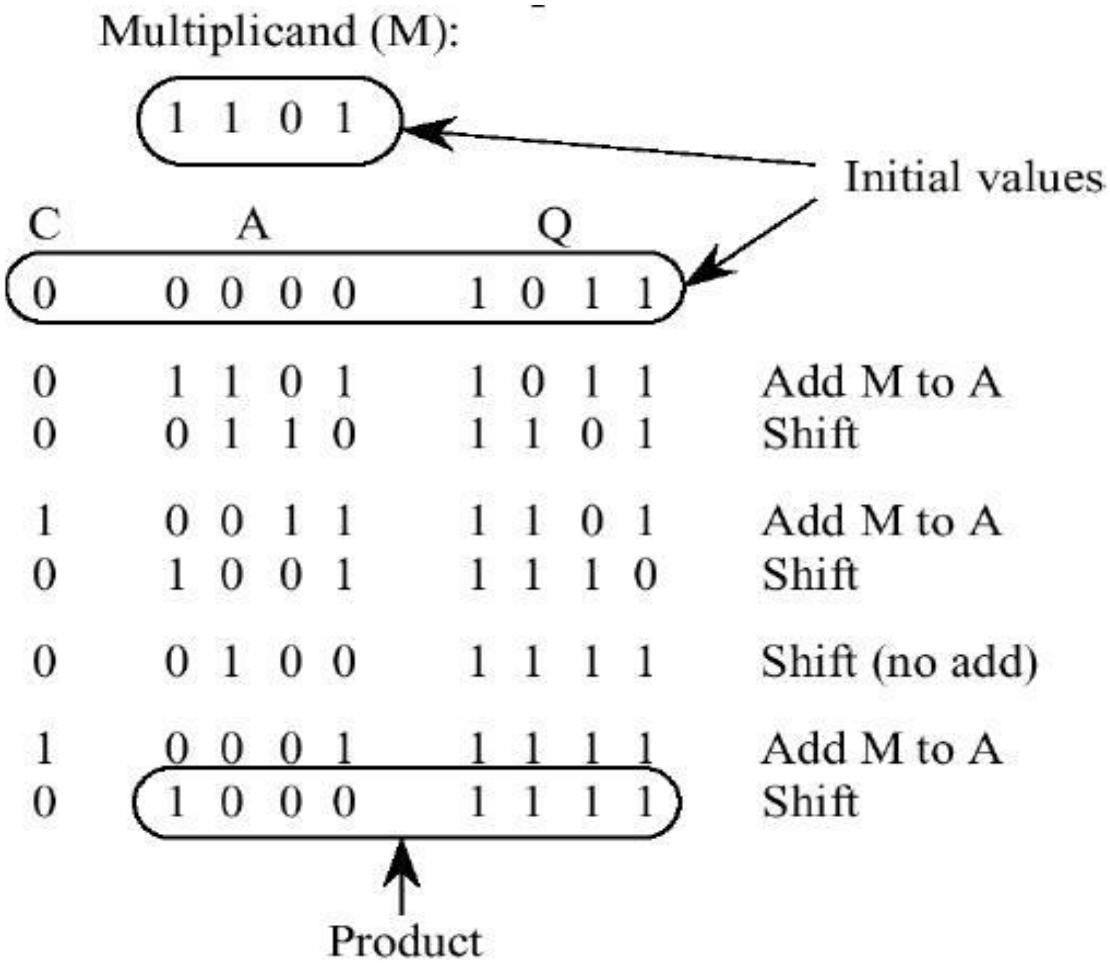


ฮาร์ดแวร์สำหรับการคูณ

- พิจารณา
  1. ADD M to A คือการนำค่าในรีจิสเตอร์ M มาบวกกับค่าในรีจิสเตอร์ A และนำค่าไปเก็บในรีจิสเตอร์ A
  2. การกระทำในข้อ 1 จะกระทำก็ต่อเมื่อบิตขวาสุดของรีจิสเตอร์ Q มีค่าเป็น 1 เท่านั้น
  3. ในการคูณจะทำการเลื่อนขวา ด้วยการใช้บิต 0 เป็นตัวดัน
  4. จำนวนรอบในการทำงาน จะกระทำเท่ากับจำนวนบิต

ฮาร์ดแวร์สำหรับการคูณ

- ตัวอย่าง การคูณโดยมีตัวตั้ง 1101 กับตัวคูณ 1011

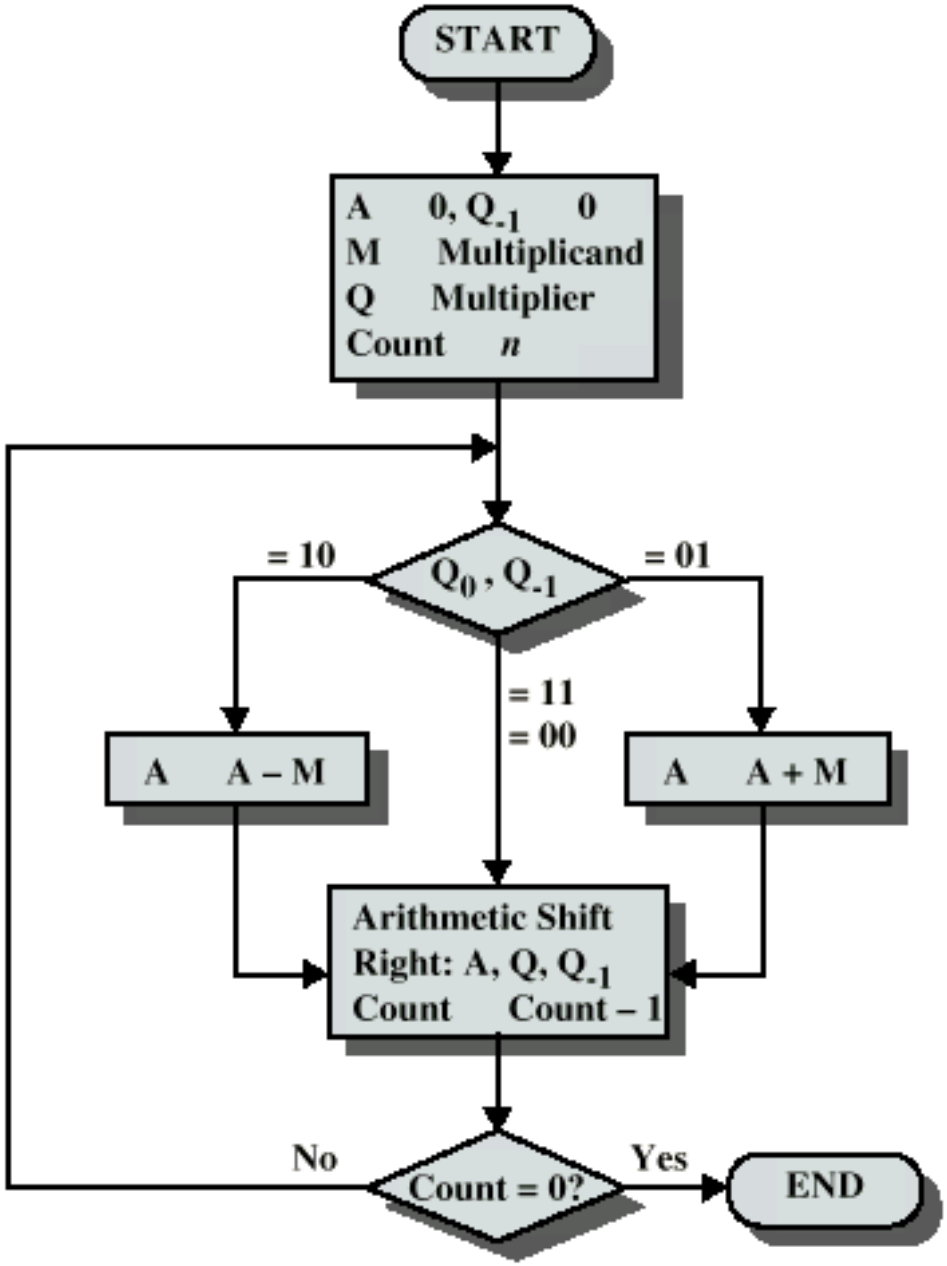


อัลกอริทึมของบูธ

- อัลกอริทึมของบูธ (Booth's Algorithm) สร้างโดย Andrew Booth ได้คิดค้นเทคนิคใหม่ในการคูณเลขแบบทูคอมพลีเมนต์ เป็นการคูณที่เร็วกว่าแบบพื้นฐาน โดยวัตถุประสงค์ของ Booth
- คือจะทำการลดจำนวนครั้งของการบวกในทุกขั้นตอนที่เกิดขึ้นในอัลกอริทึมการคูณแบบพื้นฐาน ซึ่งมีขั้นตอนดังนี้
  - จะเก็บไว้ที่รีจิสเตอร์ Q (Multiplier) และ M (Multiplicand) และมีรีจิสเตอร์ขนาด 1 บิตที่กำหนดเป็น Q_{-1} วางไว้ด้านขวาของบิตที่มีค่าน้อยที่สุด (Q_0) ของรีจิสเตอร์ Q
  - ผลคูณที่ได้จะเก็บไว้ในรีจิสเตอร์ A และ Q
  - เมื่อเริ่มต้นการทำงานจะกำหนดค่า A และ Q_{-1} เป็น 0

อัลกอริทึมของบูธ

- ตัวควบคุมจะตรวจสอบบิตของตัวคูณร่วมกับบิต Q_{-1} (Q_0, Q_{-1}) โดยมีเงื่อนไข
  - ถ้า 2 บิตเหมือนกัน (1,1 หรือ 0,0) ให้ทำการเลื่อนทุกบิตของรีจิสเตอร์ A,Q และ Q_{-1} ไปทางขวา 1 บิต
  - แต่ถ้า 2 บิตต่างกันเป็น (0,1) จะนำค่าจากตัวตั้ง (M) มา**บวก**เข้ากับ A แล้วเก็บไว้ที่ A แล้วจึงทำการเลื่อนบิตไปทางขวา 1 บิต
  - แต่ถ้า 2 บิตต่างกันเป็น (1,0) จะนำค่าจากตัวตั้ง (M) มา**ลบ**ออกจาก A แล้วเก็บไว้ที่ A แล้วจึงทำการเลื่อนบิตไปทางขวา 1 บิต
  - ในการเลื่อนบิตไปทางขวาบิตซ้ายสุดของรีจิสเตอร์ A (A_{n-1}) ไม่เพียงแค่เลื่อนบิตไปที่ A_{n-2} เท่านั้น แต่ยังคงอยู่ใน A_{n-1} ด้วยเพื่อเก็บเครื่องหมายของตัวเลขในรีจิสเตอร์ A


START
A 0, Q-1 0
M Multiplicand
Q Multiplier
Count n
Q0 , Q-1
= 10
= 01
= 11
= 00
A A – M
A A + M
Arithmetic Shift
Right: A, Q, Q-1
Count Count – 1
Count = 0?
No
Yes
END

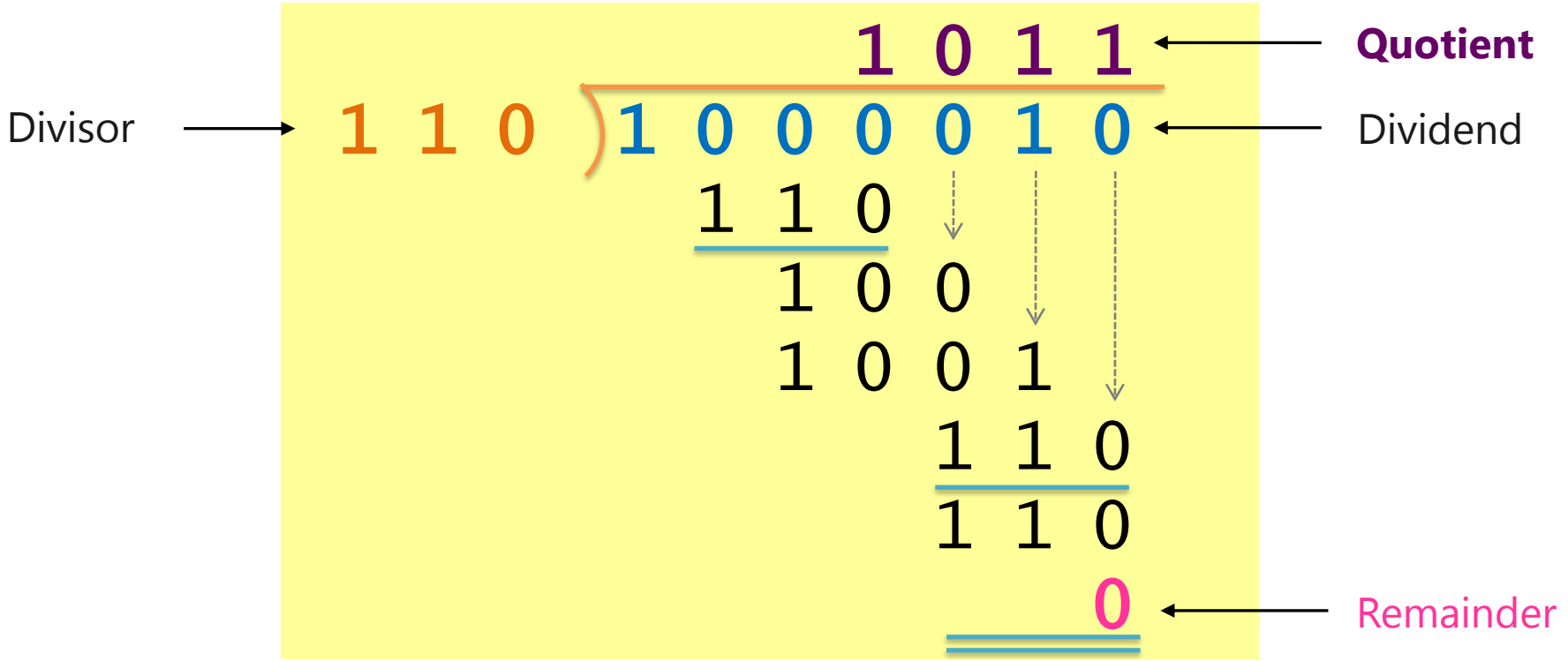
อัลกอริทึมของบูธ

- การคูณ ด้วย 2's Complement ระหว่าง (+7) x (+3)

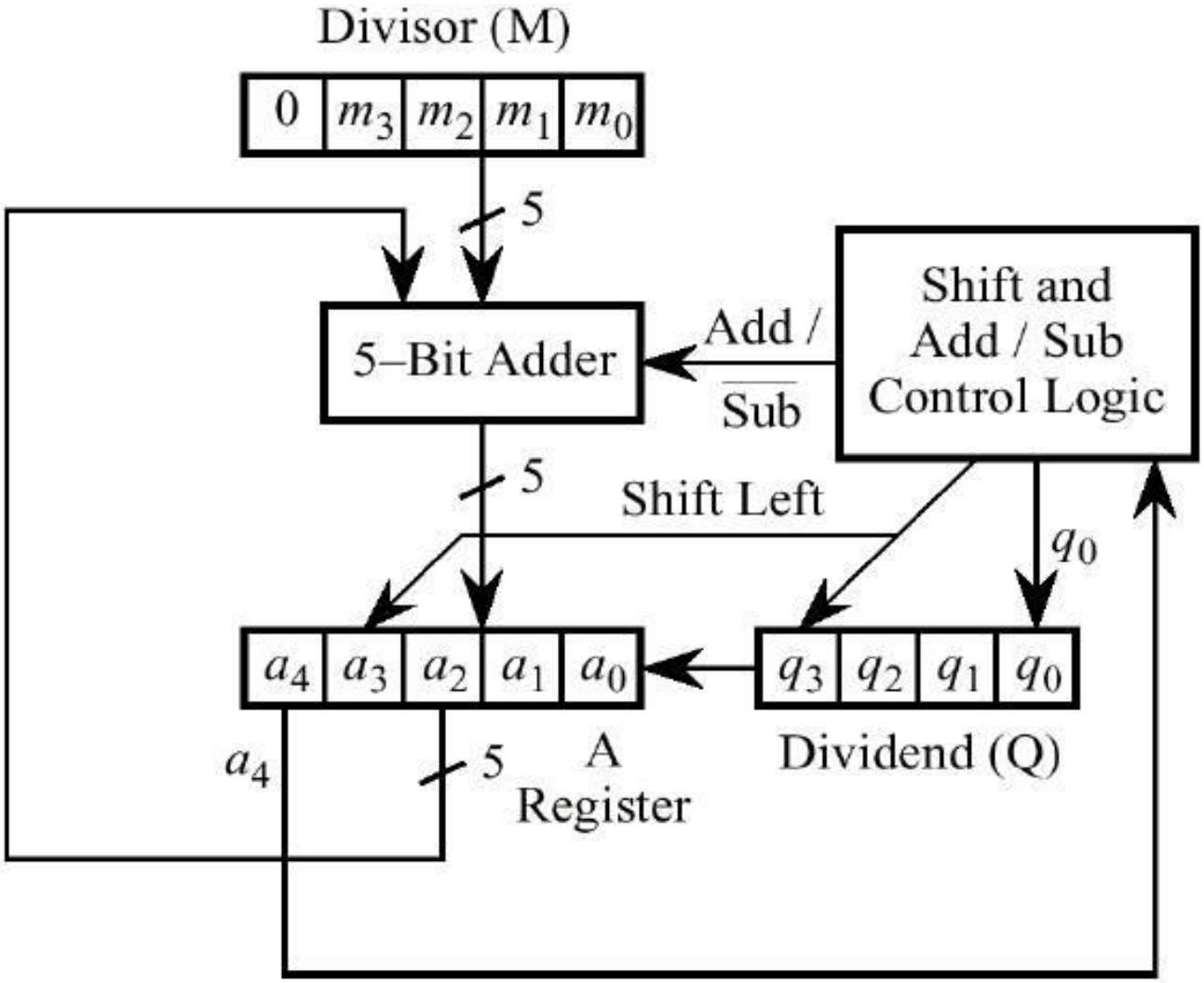
A	Q	Q_{-1}	M		
0000	0011	0	0111	Initial Values	
1001	0011	0	0111	A A - M	First Cycle
1100	1001	1	0111	Shift	
1110	0100	1	0111	Shift	Second Cycle
0101	0100	1	0111	A A + M	Third Cycle
0010	1010	0	0111	Shift	
0001	0101	0	0111	Shift	Fourth Cycle

การหาร (Integer Division)

- การหารเลขจำนวนเต็ม (Integer Division) มีวิธีการที่ซับซ้อนกว่าการคูณ แต่ใช้หลักการเดียวกับการคูณ

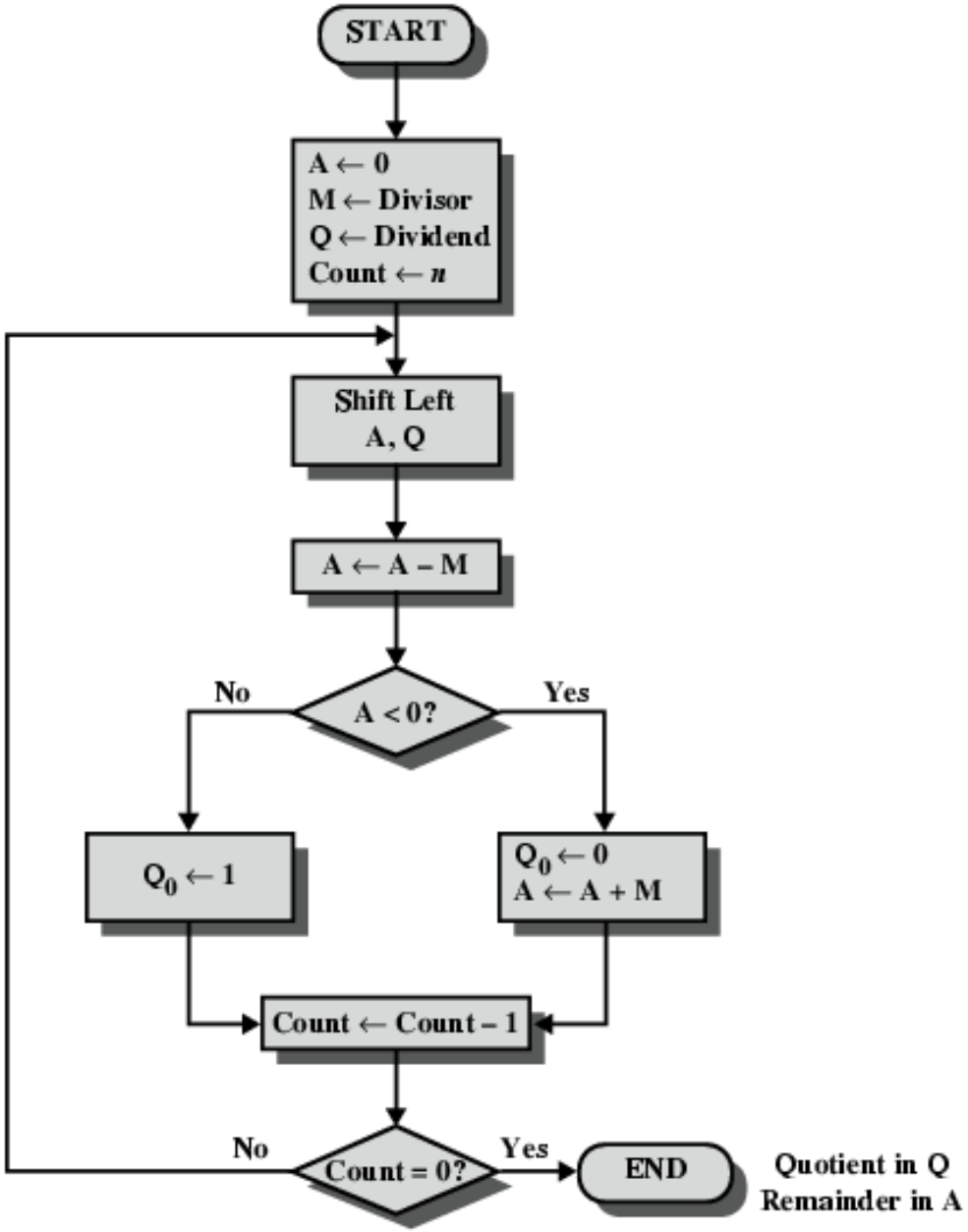


การหาร (Integer Division)

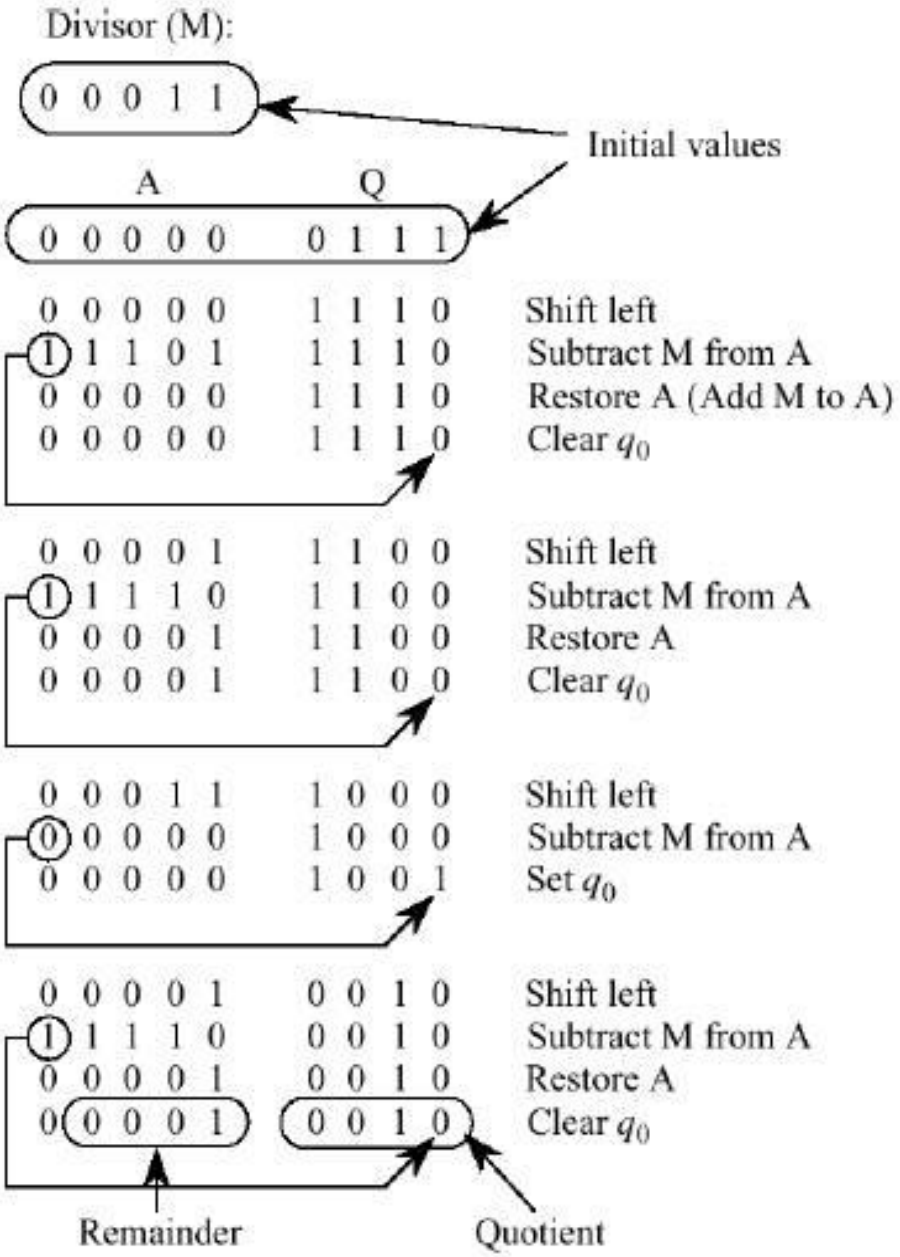


การหาร (Integer Division)

- รายละเอียดขั้นตอนในการหารและรีจิสเตอร์ที่เกี่ยวข้องประกอบไปด้วย
  - ตัวตั้งคือรีจิสเตอร์ Q, ตัวหารคือรีจิสเตอร์ M
  - กำหนดค่าเริ่มต้นให้รีจิสเตอร์ A เป็น 0
  - นำบิต 0 มาดันรีจิสเตอร์ Q เพื่อให้เลื่อนไปทางซ้าย
  - นำค่าของ M ลบออกจาก A (A – M) การลบจะเป็นการบวกด้วยคอมพลีเมนต์
    - ถ้าบิตซ้ายของรีจิสเตอร์ A มีค่าเท่ากับ 1 ให้ดึงค่าเดิมก่อนหน้านั้นมาใช้ จากนั้นกำหนดค่าบิตขวาสุดของรีจิสเตอร์ Q เป็น 0
    - ถ้าบิตซ้ายของรีจิสเตอร์ A มีค่าเท่ากับ 0 ให้เซ็ตบิตขวาสุดของรีจิสเตอร์ Q เป็น 1
  - โดยจำนวนรอบที่ดำเนินการจะกระทำเท่ากับจำนวนบิต


START
A ← 0
M ← Divisor
Q ← Dividend
Count ← n
Shift Left
A, Q
A ← A – M
No
A < 0?
Yes
Q0 ← 1
Q0 ← 0
A ← A + M
Count ← Count – 1
No
Count = 0?
Yes
END
Quotient in Q
Remainder in A

การหารโดย
มีตัวตั้ง 111
กับตัวหาร 11



M = 00011

2's Complement = 11101

การประมวลผลเลขทศนิยม

- เลขทศนิยมจะต้องจัดเก็บข้อมูลเป็นแบบเลขทศนิยม ซึ่งมีวิธีการจัดเก็บต่างจากเลขจำนวนเต็มที่เรียกว่า “Fixed Point”
  - ตัวเลขแบบนี้สามารถนำมาแทนเลขที่เป็นจำนวนบวกและจำนวนลบได้ โดยการนำเลขแบบทูคอมพลีเมนต์ซึ่งจะจัดเก็บข้อมูลอยู่ภายในขอบเขตที่กำหนดค่าหนึ่ง
- แต่ถ้าหากต้องการแทนเลขที่อยู่นอกขอบเขตที่กำหนดก็สามารถนำระบบ**เลขทศนิยมที่เรียกว่า “Floating Point”** มาใช้ได้
  - เลขแบบทศนิยมนี้สามารถนำมาแทนข้อมูลที่มีขนาดใหญ่กว่าได้ ซึ่งสามารถเขียนแทนได้ในรูปแบบของเลขยกกำลัง
  - ซึ่งอาจแสดงจำนวนได้ตั้งแต่ $\pm 10^{-38}$ ถึง 10^{+38}
  - และยังช่วยลดข้อผิดพลาดจากการคำนวณเมื่อเกิดทศนิยมมากเกินไป ทำให้เกิดการสูญหายและความแม่นยำลดลง
  - การใช้เลขทศนิยมใช้สื่อจัดเก็บข้อมูลน้อยกว่าและเพิ่มความเร็วในการคำนวณ

เลขยกกำลัง

- ข้อมูลที่มีขนาดใหญ่ สามารถเขียนให้อยู่ในรูปแบบเลขยกกำลังได้ ตัวอย่างเช่น

$$3,155,760,000$$

- สามารถเขียนให้อยู่ในรูปแบบเลขยกกำลังได้เป็น

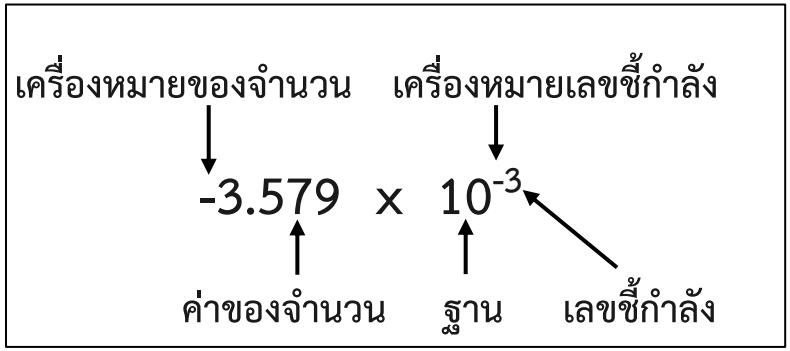
$$3.15576 \times 10^{9}$$

- หรือหากข้อมูลมีจำนวนเล็กมากๆ เช่น 0.00000001 สามารถเขียนเป็นรูปแบบเลขยกกำลังได้เป็น

$$1.0 \times 10^{-8}$$

เลขทางวิทยาศาสตร์

- การเขียนเลขยกกำลังในลักษณะต่างๆ เรียกอีกอย่างหนึ่งว่า “เลขทางวิทยาศาสตร์” (Scientific Notation) โดยทั่วไปการแสดงตัวเลขในลักษณะนี้จะมีส่วนประกอบ 4 ส่วนคือ
  1. เครื่องหมายของจำนวน (+ หรือ -)
  2. ค่าของจำนวน เรียกว่า **“Mantissa”**
  3. เครื่องหมายเลขชี้กำลัง (+ หรือ -)
  4. ตัวเลขชี้กำลัง



- นอกจากนี้ยังต้องมีข้อมูลเพื่อบอกว่าตัวเลขชี้กำลังที่แสดงอยู่นั้นเป็นเลขฐานอะไร และตำแหน่งของจุดทศนิยมอยู่ในตำแหน่งใด

การแทนเลขทศนิยมในคอมพิวเตอร์

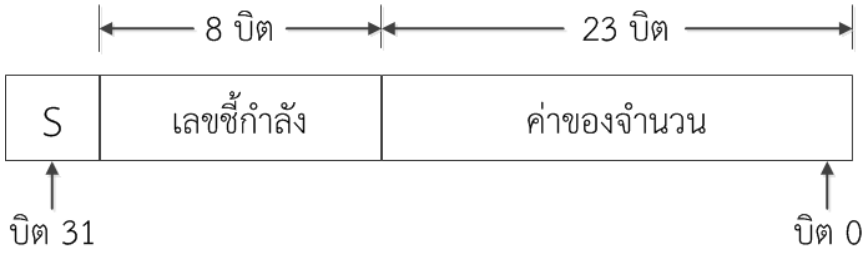
- ตัวเลขทศนิยมสามารถจัดเก็บอยู่ในรูปแบบเลขฐานสองได้ โดยมีการกำหนดว่าเลขฐานสองในส่วนใดจะใช้แทนค่าแมนทิสซา (Mantissa) ส่วนใดจะใช้แทนเลขชี้กำลัง (Exponent) โดยมีรูปแบบทั่วไปดังนี้

SEEMMMMM

  - S คือ Sign เป็นบิตสำหรับจัดเก็บเครื่องหมาย
  - E คือ Exponent เป็นบิตที่ใช้เก็บเลขชี้กำลัง
  - M คือ Mantissa เป็นบิตค่าของจำนวนทศนิยม

การแทนเลขทศนิยมในคอมพิวเตอร์

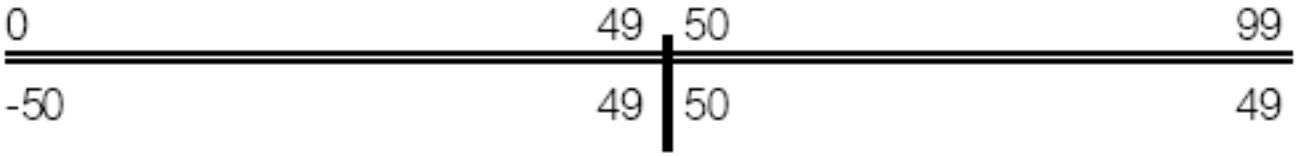
- ถ้าหากเลขทศนิยมนี้ถูกเก็บในหน่วยความจำขนาด 4 ไบต์หรือ 32 บิต การแบ่งข้อมูลในระบบเลขฐานสองจะได้ดังรูป



  - บิตที่ 31 จำนวน 1 บิตแทนบิตเครื่องหมาย
  - บิตที่ 23 ถึงบิตที่ 30 ใช้แทนเลขชี้กำลัง (Exponent)
  - ส่วน 23 บิตจะใช้แทนค่าของจำนวนแมนทิสซา (Mantissa)

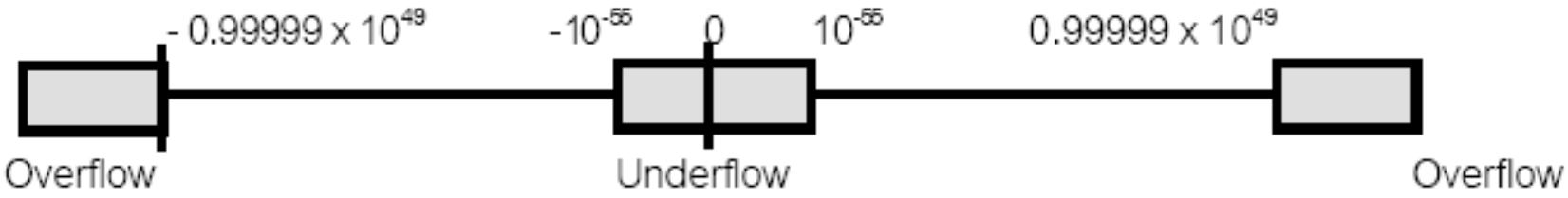
Excess-N

- Excess-N จะใช้การเลือกตัวที่อยู่กึ่งกลาง หากเป็นข้อมูลทางด้านซ้ายของค่ากึ่งกลางจะเป็นลบ แต่หากเป็นข้อมูลทางด้านขวาจะเป็นค่าบวก โดย N คือค่ากึ่งกลางที่เลือก
- ตัวอย่าง Excess-50 (เลือกค่า 50 ซึ่งอยู่กึ่งกลางระหว่างช่วง 0-99 แล้วกำหนดค่าเลขชี้กำลัง ณ จุดที่เลือกเป็น 0)
  - ถ้าเลื่อนไปทางขวา (จาก 50 เป็น 51 เรื่อยไปจนถึง 99) ค่าเลขชี้กำลังจะเป็น 1, 2, 3, ...,49
  - ถ้าเลื่อนไปทางซ้าย (จาก 50 เป็น 49 เรื่อยไปจนถึง 0) ค่าเลขชี้กำลังจะเป็นค่าลบจาก -1, -2, -3, ...,-50)



Excess-N

- โอกาสที่จะเกิด Overflow สำหรับจำนวนทศนิยมเป็นไปได้ยาก ซึ่งการเกิด Overflow นั้นจะต้องมีขนาดใหญ่กว่า -0.9999×10^{49} หรือ 0.9999×10^{49}
- นอกจากนี้ยังสามารถเกิด Underflow ซึ่งเป็นจำนวนที่เล็กเกินกว่าที่จะเก็บได้ ซึ่งเป็นค่าที่อยู่ในช่วง -10^{-55} ถึง 10^{-55} ($0.0001 \times 10^{-50} = 10^{-55}$)



Excess-N

- ข้อควรพิจารณา
- คอมพิวเตอร์จัดเก็บเฉพาะตัวเลขเท่านั้น ไม่มีการเก็บเครื่องหมายหรือจุดทศนิยม ดังนั้นถ้าต้องการแสดงเครื่องหมายและจุดทศนิยม ในการแสดงผลเลขทศนิยมฐานสิบ ตามรูปแบบของเลขทศนิยมจะมีเครื่องหมาย 1 ตัว เลขชี้กำลังสองตัวที่เก็บในแบบ Excess-50 และค่าอีก 5 ตัว
- ค่า 0 แสดงเครื่องหมาย + ในขณะที่ 5 แสดงเครื่องหมาย – เหมือนกับ 1 ที่แสดงเครื่องหมาย - ในคอมพิวเตอร์

Excess-N

- ตัวอย่าง จงหาค่าเลขทศนิยมฐานสิบ เมื่อใช้รูปแบบทศนิยมแบบ SEEMMMMM ที่ใช้ Excess-50 ของจำนวนต่อไปนี้
  - **05323456**
    - 0 หมายถึงจำนวนบวก
    - 53 หมายถึงเลขชี้กำลัง (Excess-50 เลข 53 คือเลข 3) ทำให้ได้ 10^3
    - 23456 คือค่า 23456 (Mantissa)
    - $= 0.23456 \times 10^3$ $= 234.56$
  - 54820000 =
  - 55555555 =
  - 04935791 =

การทำ Normalization เลขทศนิยม

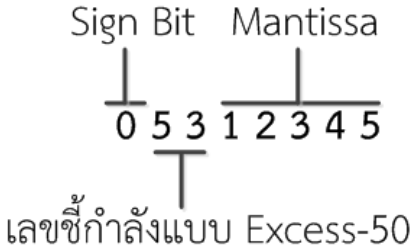
- เป็นการกำหนดรูปแบบของเลขทศนิยมเพื่อให้เกิดความแม่นยำ **(การเก็บจำนวนโดยไม่มีเลข 0 นำหน้า)** โดยการเลื่อนจำนวนไปทางซ้ายโดยการเพิ่มเลขชี้กำลังจนกว่า 0 ที่นำหน้าจำนวนนั้นถูกกำจัดออกไป
  - รูปแบบการเก็บค่าที่เหมาะสมประกอบด้วยเครื่องหมายและค่าของจำนวนที่มี 5 หลัก ดังนั้นรูปแบบมาตรฐานจะเป็น

    .MMMMM x 10EE

  - การแปลงเลขทศนิยมฐานสิบเป็นรูปแบบมาตรฐานสามารถทำได้ 4 ขั้นตอนคือ
    1. จัดจำนวนอยู่ในลักษณะของเลขยกกำลัง
    2. เลื่อนจุดทศนิยมไปทางซ้ายหรือขวา โดยเพิ่มหรือลดเลขชี้กำลังที่สัมพันธ์กันจนกว่าจุดทศนิยมจะอยู่ในตำแหน่งที่ถูกต้อง
    3. ทำให้เป็น Normalization โดยเลื่อนจุดทศนิยมให้สัมพันธ์กับเลขชี้กำลังจนกว่าค่าจะไม่มีศูนย์นำหน้า
    4. ตรวจสอบความแม่นยำ โดยการเพิ่มหรือตัดตัวเลขที่ไม่จำเป็นเพื่อให้อยู่ในรูปแบบมาตรฐาน

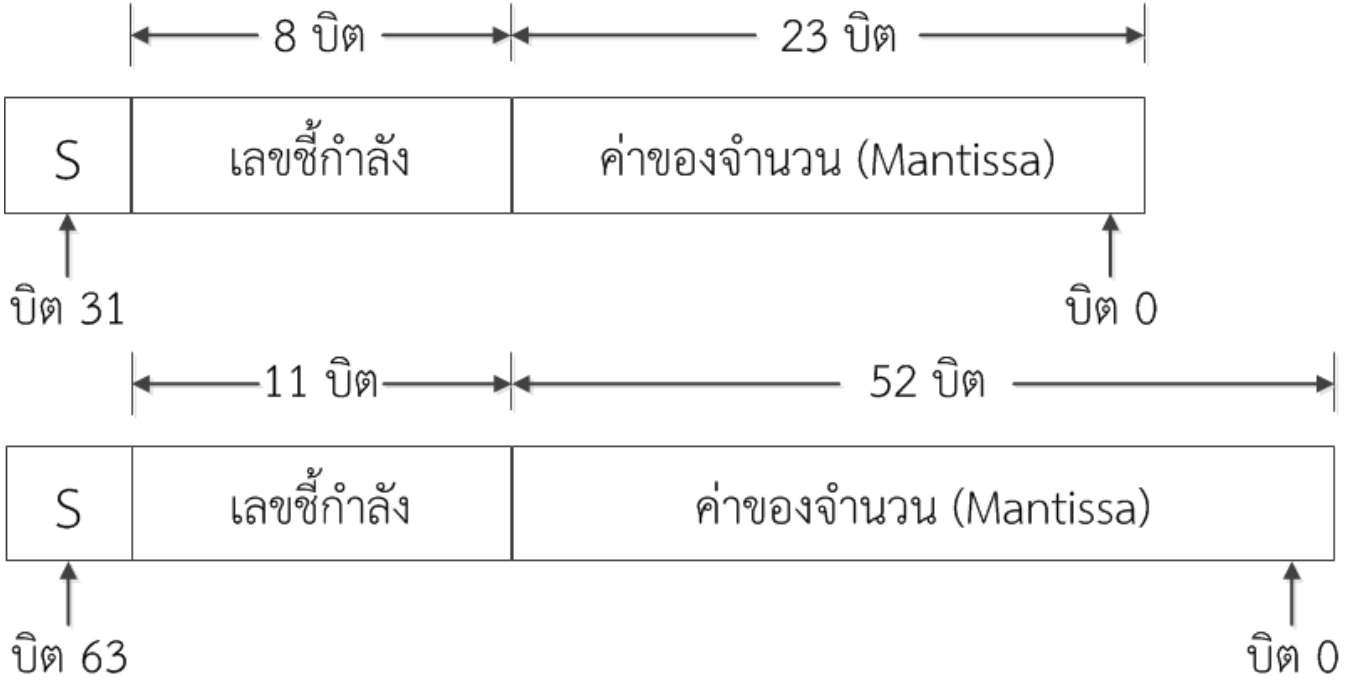
ตัวอย่าง การทำ Normalization

- Ex1 123.4567
  1. จัดจำนวนอยู่ในลักษณะของเลขยกกำลัง >> 123.4567×10^0
  2. เลื่อนจุดทศนิยมไปทางซ้าย 3 ตำแหน่ง >> 0.1234567×10^3
  3. เนื่องจากจำนวนอยู่ในรูปแบบของ Normalization แล้ว (ไม่มี 0 นำหน้าจำนวน) จึงไม่ต้องมีการปรับแต่ง
  4. เนื่องจากเลขมี 7 ตัว แต่รูปแบบมาตรฐานมี 5 ตัว จึงตัด 2 ตัวท้ายที่มีค่าน้อยที่สุดออกไปทำให้ได้ 0.12345×10^3
  5. เลขชี้กำลังเป็น 3 เมื่อใช้รูปแบบ Excess-50 จะต้องใช้ 53 แทนตำแหน่ง EE สำหรับเครื่องหมาย ถ้าเป็น + จะใช้เลข 0 แต่ถ้าเป็น – จะใช้เลข 5 ดังนั้นผลลัพธ์ที่ได้



เลขทศนิยมตามมาตรฐาน IEEE754

- รูปแบบมาตรฐานของเลขทศนิยมขนาด 32 บิตและ 64 บิต เพื่ออำนวยความสะดวกในการแสดงค่าข้ามแพล็ตฟอร์มที่ใช้โปรเซสเซอร์ต่างกัน รูปแบบเลขชี้กำลังแบบ 32 บิตเรียกว่า แบบ Single-precision ส่วน 64 บิตเรียกว่าแบบ Double-precision



เลขทศนิยมตามมาตรฐาน IEEE754

- เมื่อนำมาเขียนในรูปแบบเลขฐานสองจะเขียนได้ดังนี้

1.MMMMMMMMM

- โดยที่ MMMMMMMMM แทนค่าของข้อมูล (Mantissa) สำหรับรูปแบบของเลขชี้กำลังแบบ 32 บิต ถ้าหากเลขชี้กำลังเป็นแบบ Excess-127 จะทำให้เก็บข้อมูลได้ในช่วง 2^{-126} ถึง 2^{127}
- การเก็บข้อมูล 64 บิต ถ้าหากเลขชี้กำลังเป็นแบบ Excess-1023 เมื่อแทนค่าข้อมูลในรูปแบบเลขฐานสิบสามารถแสดงได้ถึงเลขนัยสำคัญ 15 ตัวเลขและเก็บข้อมูลได้ในช่วง 10^{-300} ถึง 10^{300}
- สำหรับวิธีการเปลี่ยนข้อมูลที่อยู่ในรูปแบบ IEEE754 มาเป็นข้อมูลทศนิยมนั้น ถ้าหากเป็นแบบ 32 บิต จะหาค่าได้ดังรูปแบบต่อไปนี้

$$(-1)^{S} \times (1 + \text{ค่าของจำนวน}) \times 2^{(\text{เลขชี้กำลัง} - 127)}$$

เลขทศนิยมตามมาตรฐาน IEEE754

- สำหรับวิธีการเปลี่ยนข้อมูลที่อยู่ในรูปแบบ IEEE754 มาเป็นข้อมูลทศนิยมนั้น ถ้าหากเป็นแบบ 32 บิต จะหาค่าได้ดังรูปแบบต่อไปนี้

$$(-1)^{S} \times (1 + \text{ค่าของจำนวน}) \times 2^{(\text{เลขชี้กำลัง} - 127)}$$

  - โดยที่ S เป็นบิตสูงสุดหรือบิตเครื่องหมาย (ถ้าหากบิตนี้เป็น 0 จะได้ค่าเป็นจำนวนบวก ถ้าเป็น 1 จะได้ค่าเป็นจำนวนลบ) คูณกับข้อมูลค่าแรกที่เป็น 1 ตามด้วยค่าของทศนิยม (Mantissa) ที่เป็นค่าจำนวนขนาด 23 บิต (ซึ่งจะทำให้ข้อมูลมีค่าอยู่ระหว่าง 1 ถึงน้อยกว่า 2 (1.0000 – 1.9999)
  - แล้วคูณกับค่าฐานสองยกกำลังด้วยค่าของเลขชี้กำลังลบออกด้วย 127 (เนื่องจากเป็นการใช้ตัวเลขแบบ Excess-N ถ้าหากข้อมูลมีขนาด k บิต ค่า N นี้จะหาค่าได้จาก $2^{k-1} - 1$ ซึ่งถ้าหากตำเลขชี้กำลังมีขนาด 8 บิต ค่านี้จะมีค่าเท่ากับ 127

เลขทศนิยมตามมาตรฐาน IEEE754

- ตัวอย่าง หากมีข้อมูลเลขทศนิยมที่อยู่ในรูปแบบ IEEE754 ที่มีค่าต่อไปนี้ จงเปลี่ยนให้อยู่ในรูปแบบเลขฐานสิบ

0	0110 1000	101 0101 0100 0011 0100

  - บิตเครื่องหมายเป็น 0 ดังนั้นเป็นเลขบวก
  - เลขชี้กำลังมีค่าเป็น 0110 1000_2 ซึ่งมีค่าเป็นเลขฐานสิบคือ 104 ดังนั้นค่ายกกำลังของสองคือ 104 – 127 ซึ่งจะเท่ากับ -23
  - สำหรับตัวเลขหาได้จากค่า 1 ตามด้วยค่าทศนิยมที่เป็นค่า Mantissa เขียนได้เป็น

1.101 0101 0100 0011 0100 หากเปลี่ยนให้อยู่ในรูปแบบเลขฐานสิบจะได้

$1 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} + 0 \times 2^{-4} + 1 \times 2^{-5} + \ldots + 0 \times 2^{-19}$

  - หรือมีค่าเท่ากับ 1.666115 ดังนั้นเลขฐานสองที่เก็บในรูปแบบที่โจทย์กำหนดสามารถเปลี่ยนเป็นเลขฐานสิบได้เป็น 1.666115×2^{-23} หรือมีค่าประมาณ 1.986×10^{-7}

ชุดคำสั่ง
(Instruction Set)

Chapter 4

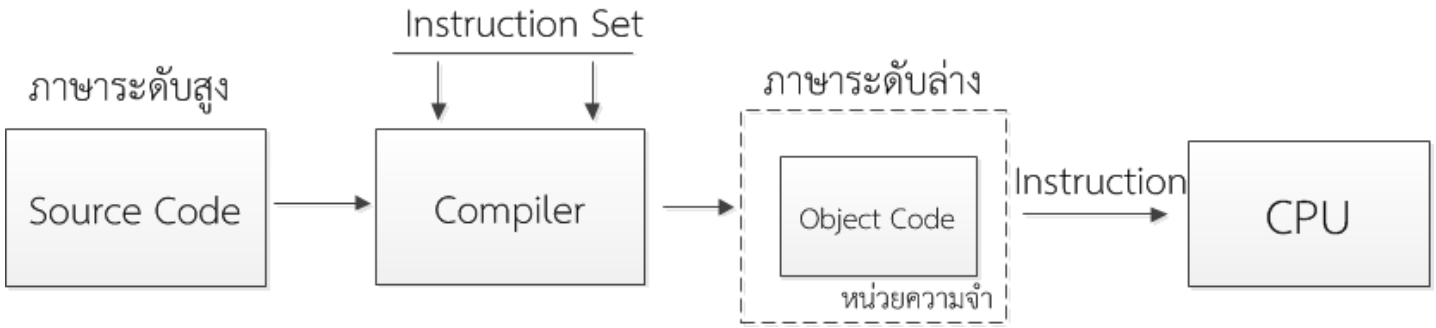
Computer Organization and Architecture

ปรับปรุงจากเอกสารของ
อ. ณัฐวุฒิ สร้อยดอกสน (NSD)

อ.เอิญ สุริยะฉาย (ENS)
ภาควิชาวิทยาการคอมพิวเตอร์และสารสนเทศ
มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าพระนครเหนือ

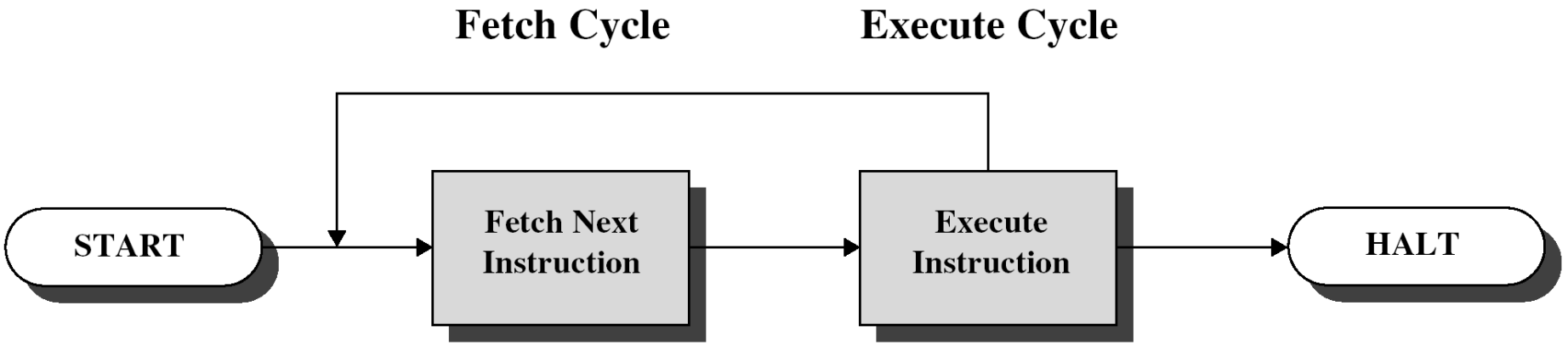
ชุดคำสั่ง (Instruction Set)

- คือกลุ่มของคำสั่งที่ใช้เป็นตัวบอกถึงขั้นตอนการทำงานอย่างเป็นลำดับให้แก่ CPU และ CPU จะทำงานตาม Instruction Set นั้น
  - ภายใน Instruction Set จะประกอบด้วย Instruction มากกว่าหนึ่งตัว ตัวอย่างของ Instruction ได้แก่ ADD, SUBTRACT, MULTIPLY, DIVIDE, COMPARE, MOVE เป็นต้น
  - หน้าที่ของ Instruction Set จะช่วยตัวคอมไพเลอร์ ทำการแปลง Source Code ของโปรแกรมที่เป็นภาษาระดับสูงให้เป็น Object Code ไปเก็บลงในหน่วยความจำหลัก เพื่อรอส่งชุดคำสั่งที่ต้องการประมวลผลไปยัง CPU

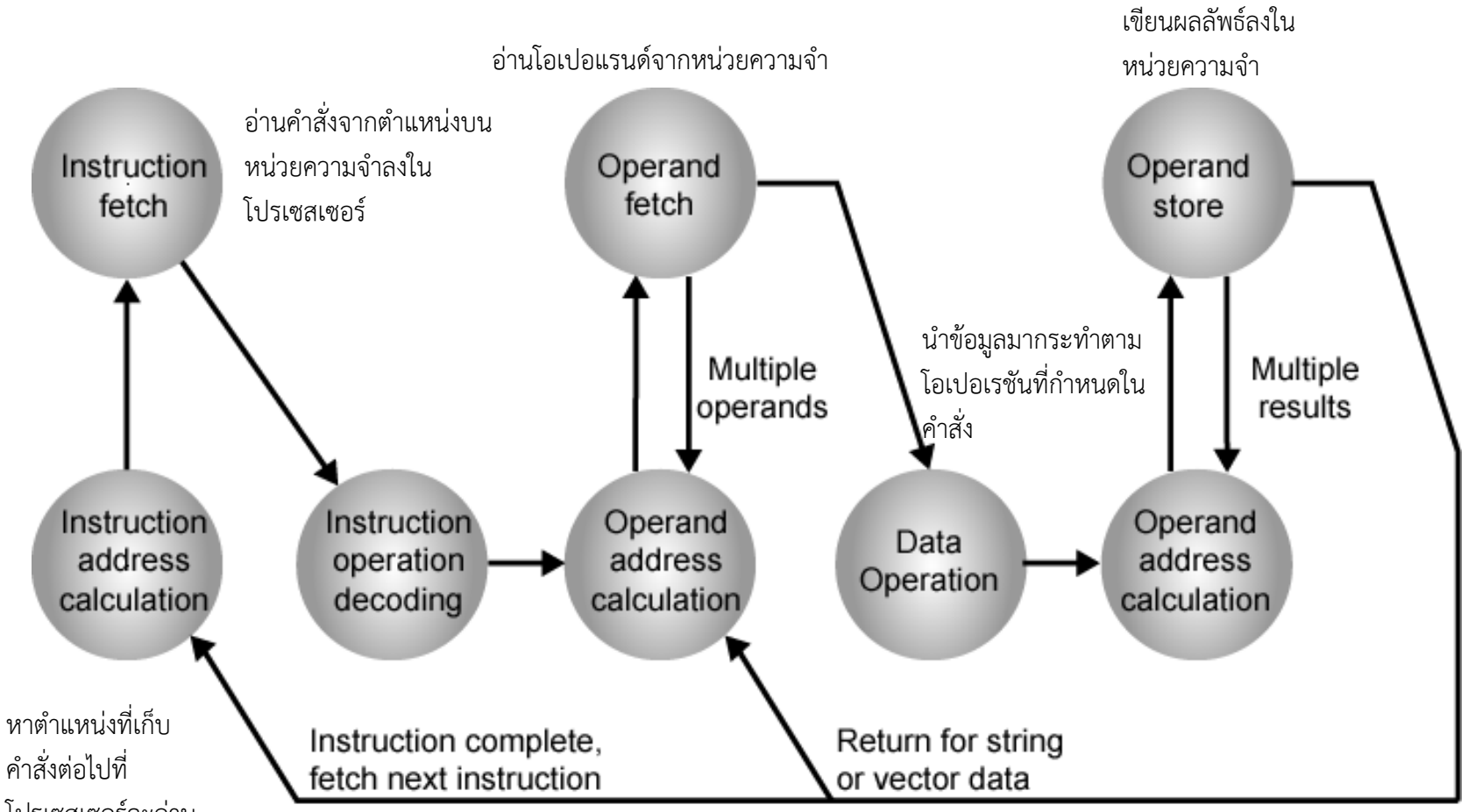


วัฏจักรคำสั่ง (Instruction Cycle)

- กระบวนการในการทำคำสั่งที่เก็บอยู่ในหน่วยความจำประกอบด้วย 2 ขั้นตอนคือ
  1. การอ่านรหัสคำสั่ง (Fetches) จากหน่วยความจำ
  2. การปฏิบัติการ (Execution) หรือการกระทำคำสั่งนั้นๆ
- กระบวนการในการทำคำสั่ง 1 คำสั่งเรียกว่า ไซเคิลคำสั่ง (Instruction Cycle) หรือ วัฏจักรคำสั่ง โดยโปรแกรมจะหยุดทำงานเมื่อถึงคำสั่งที่ให้หยุดการทำงาน หรือเมื่อปิดเครื่อง



ไดอะแกรมของวัฏจักรคำสั่ง



ไดอะแกรมของวัฏจักรคำสั่ง

- ในการอ่านคำสั่ง โปรเซสเซอร์อาจจะอ่านจากหน่วยความจำโดยตรง แต่โดยทั่วไปแล้วโปรเซสเซอร์จะอ่านตำแหน่งที่เก็บคำสั่งต่อไปจากโปรแกรมเคาเตอร์ (Program Counter: PC) ตำแหน่งเหล่านี้จะอยู่บนหน่วยความจำแล้วเอ็กซิคิวต์ตามคำสั่งที่อ่านเข้ามา ซึ่ง PC จะทำการเพิ่มค่าขึ้นเรื่อย ๆ ทำให้โปรเซสเซอร์ทำงานเรื่อยไป
  - **Instruction Address Calculation** (IAC)
    - หาตำแหน่งที่เก็บคำสั่งต่อไปที่โปรเซสเซอร์ที่จะอ่านเข้ามา
  - **Instruction Fetch** (IF)
    - อ่านคำสั่งจากตำแหน่งบนหน่วยความจำลงในโปรเซสเซอร์
  - **Instruction Operation Decoding** (IOD)
    - วิเคราะห์คำสั่งเพื่อพิจารณาประเภทของโอเปอเรชันที่จะกระทำ รวมทั้งโอเปอแรนด์ที่จะใช้

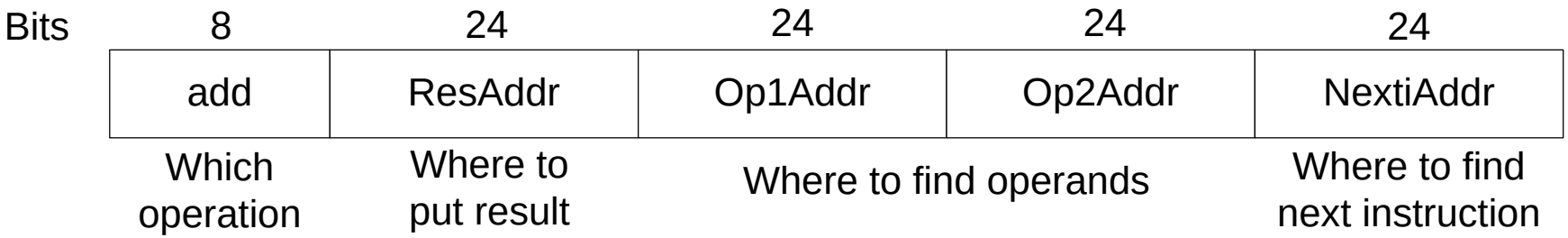
ไดอะแกรมของวัฏจักรคำสั่ง

- **Operand Address Calculation** (OAC)
  - หาตำแหน่งของ โอเปอแรนด์ ถ้าโอเปอเรชันนั้นมีการอ้างถึงโอเปอแรนด์หรือที่มีอยู่ในหน่วยความจำผ่านทางอุปกรณ์ อินพุต/เอาท์พุต
- **Operand Fetch** (OF)
  - อ่านโอเปอแรนด์จากหน่วยความจำหรืออ่านจากอุปกรณ์อินพุต/เอาท์พุต
- **Data Operation** (DO)
  - นำข้อมูลมากระทำตามโอเปอเรชันที่กำหนดในคำสั่ง
- **Operand Store** (OS)
  - เขียนผลลัพธ์ลงในหน่วยความจำหรือส่งออกไปทางอุปกรณ์อินพุต/เอาท์พุต

ส่วนประกอบของคำสั่ง

- ในแต่ละคำสั่งจะประกอบด้วยข้อมูลที่โปรเซสเซอร์ต้องการ เพื่อเอ็กซิคิวต์ตามวัฏจักรคำสั่ง ซึ่งมีส่วนประกอบของคำสั่ง 3 ส่วนดังนี้
  1. **Operation Code** หรือรหัสดำเนินงาน (Opcode) หรืออ้อปโค้ด ทำหน้าที่กำหนดรหัสคำสั่งให้ทำอะไร
  2. **Operand Reference** เป็นการอ้างอิงตัวถูกดำเนินการ
  3. **Next Instruction Reference** การอ้างถึงรหัสคำสั่งถัดไป

Instruction Format (4-Address Instruction)



ประเภทคำสั่ง

- ลองพิจารณาคำสั่งภาษาชั้นสูง เช่น Pascal, Fortran หรือ Basic

$$X = X + Y$$

- ลองสมมติค่าตัวแปร X และ Y เก็บอยู่ที่ตำแหน่ง 123 และ 456 ตามลำดับ โอเปอเรชันนี้จะทำ 3 คำสั่ง คือ
  1. โหลดค่าจากหน่วยความจำตำแหน่ง 123 ลงรีจิสเตอร์
  2. บวกค่าจากหน่วยความจำตำแหน่ง 456 กับค่าในรีจิสเตอร์
  3. เก็บค่าจากรีจิสเตอร์ลงบนหน่วยความจำตำแหน่ง 123
- จะเห็นว่าภาษาชั้นสูงจะใช้สัญลักษณ์ทางคณิตศาสตร์และใช้ตัวแปร ส่วนคำสั่งภาษาเครื่องจะใช้การถ่ายโอนข้อมูลระหว่างหน่วยความจำกับรีจิสเตอร์

การออกแบบชุดคำสั่ง

- ชุดคำสั่ง คือสิ่งที่โปรแกรมเมอร์ใช้ควบคุมซีพียู ดังนั้นการออกแบบชุดคำสั่งเป็นงานที่มีความซับซ้อนเนื่องจากมีผลกระทบกับระบบคอมพิวเตอร์ จะพิจารณาจาก
  - **Operation repertory** : จำนวนโอเปอเรชันที่มีให้เลือกใช้ รวมทั้งความซับซ้อนของโอเปอเรชันที่ควรเป็น
  - **Data type** : ความหลากหลายของประเภทข้อมูลที่ทำโอเปอเรชัน
  - **Instruction format** : ความยาวของคำสั่ง (เป็นบิต) จำนวนแอดเดรส ขนาดของฟิลด์และอื่น ๆ
  - **Register** : จำนวนรีจิสเตอร์ที่คำสั่งสามารถอ้างอิงและใช้ประโยชน์ได้
  - **Addressing** : การกำหนดโหมดของแอดเดรสสำหรับโอเปอแรนด์

ประเภทของโอเปอเรชัน

- จำนวน Opcode แต่ละเครื่องมีหลากหลายกันไป แต่อย่างไรก็ตาม ประเภทโอเปอเรชันพื้นฐานที่พบในแต่ละเครื่องจะเหมือนกัน ซึ่งสามารถแบ่งกลุ่มได้ ดังนี้
  - โอเปอเรชันทางด้านการถ่ายโอนข้อมูล (Data transfer)
  - โอเปอเรชันทางด้านคณิตศาสตร์ (Arithmetic)
  - โอเปอเรชันทางด้านตรรกะ (Logical)
  - โอเปอเรชันทางด้านการแปลงค่า (Conversion)
  - โอเปอเรชันทางด้านอุปกรณ์อินพุต/เอาท์พุต (I/O)
  - โอเปอเรชันทางด้านการควบคุมระบบ (System control)

ประเภทของโอเปอเรชัน

ประเภทโอเปอเรชัน	ชื่อโอเปอเรชัน	คำอธิบาย
การถ่ายโอนข้อมูล	Move	ถ่ายโอนข้อมูลจากต้นทางไปปลายทาง
	Store	ถ่ายโอนข้อมูลจากโปรเซสเซอร์ไปหน่วยความจำ
	Load	ถ่ายโอนข้อมูลจากหน่วยความจำไปโปรเซสเซอร์
	Exchange	แลกเปลี่ยนข้อมูลระหว่างต้นทางกับปลายทาง
	Clear	ถ่ายโอนค่า 0 ไปยังปลายทาง
	Set	ถ่ายโอนค่า 1 ไปปลายทาง
	Push	ถ่ายโอนค่าจากต้นทางขึ้นไปบนสุดของ Stack
	Pop	ถ่ายโอนค่าจากบนสุดของ Stack ไปปลายทาง

ประเภทของโอเปอเรชัน

ประเภทโอเปอเรชัน	ชื่อโอเปอเรชัน	คำอธิบาย
คณิตศาสตร์	Add	หาผลบวกของโอเปอแรนด์ 2 ตัว
	Add with Carry	หาผลบวกของโอเปอแรนด์ 2 ตัวและบิตตัวทด
	Subtract	หาผลลบของโอเปอแรนด์ 2 ตัว
	Multiply	หาผลคูณของโอเปอแรนด์ 2 ตัว
	Divide	หาผลหารของโอเปอแรนด์ 2 ตัว
	Absolute	เปลี่ยนค่าโอเปอแรนด์เป็นค่าสมบูรณ์
	Negate	เปลี่ยนเครื่องหมายโอเปอแรนด์เป็นค่าตรงข้าม
	Increment	บวก 1 เข้ากับโอเปอแรนด์
	Decrement	ลบ 1 ออกจากโอเปอแรนด์

ประเภทของโอเปอเรชัน

ประเภทโอเปอเรชัน	ชื่อโอเปอเรชัน	คำอธิบาย
ตรรกะ	AND	กระทำโอเปอเรชันทางตรรกะบิตต่อบิต
	OR	
	NOT	
	Exclusive-OR	
	Shift	เลื่อนบิตของโอเปอแรนด์ (ซ้ายหรือขวา) 1 บิต
	Rotate	หมุนบิตของโอเปอแรนด์ (ซ้ายหรือขวา) 1 บิต โดยบิตสุดท้ายวนกลับเป็นค่าบิตแรก

ประเภทของโอเปอเรชัน

ประเภทโอเปอเรชัน	ชื่อโอเปอเรชัน	คำอธิบาย
การควบคุม	Jump (branch)	กระโดดไปโดยไม่มีเงื่อนไข ; โหลด PC ไปยังแอดเดรสที่กำหนด
	Jump condition	กระโดดไปถ้าตรงเงื่อนไข ; อาจจะโหลด PC ไปยังแอดเดรสที่กำหนด หรือ อาจจะไม่ได้ทำอะไรเลย
	Jump to subroutine	วางข้อมูลโปรแกรมคอนโทรลในตำแหน่งที่แน่นอน ; กระโดดไปตำแหน่งที่กำหนด
	Return	เปลี่ยนข้อมูล PC และรีจิสเตอร์อื่นจากตำแหน่งที่แน่นอน
	Execute	ดึงโอเปอแรนด์จากตำแหน่งที่กำหนดแล้วเอ็กซิคิวต์คำสั่ง; ไม่มีการเปลี่ยนแปลง PC
	Skip	เพิ่มค่า PC เพื่อข้ามไปยังคำสั่งต่อไป

ประเภทของโอเปอเรชัน

ประเภทโอเปอเรชัน	ชื่อโอเปอเรชัน	คำอธิบาย
การควบคุม	Skip condition	ข้ามไปถ้าตรงเงื่อนไข ; อาจจะข้ามไป หรืออาจจะไม่ได้ทำอะไรเลย
	Test	ตรวจสอบเงื่อนไข; เซ็ตค่าแฟ็กตามผลที่เกิดขึ้น
	Compare	เปรียบเทียบโอเปอแรนด์ 2 ตัวหรือมากกว่า; เซ็ตค่าแฟ็กตามผลที่เกิดขึ้น
	Set control variable	ควบคุมสำหรับป้องกัน,แก้ไขอินเทอร์รัพท์,ควบคุมเวลาหรืออื่น ๆ
	Halt	หยุดการเอ็กซิคิวต์โปรแกรม
	Wait(hold)	หยุดการเอ็กซิคิวต์โปรแกรม; ตรวจสอบเงื่อนไขไปเรื่อยจนกว่าเงื่อนไขจะเป็นจริง
	No operation	ไม่มีโอเปอเรชัน แต่ยังคงมีการเอ็กซิคิวซ์โปรแกรมต่อไป

ประเภทของโอเปอเรชัน

ประเภทโอเปอเรชัน	ชื่อโอเปอเรชัน	คำอธิบาย
อุปกรณ์ I/O	Input(read)	ถ่ายโอนข้อมูลจากพอร์ตของอุปกรณ์ I/O หรือ device ไปยังปลายทาง(เช่น จากหน่วยความจำหลักไปรีจิสเตอร์ของโปรเซสเซอร์)
	Output(write)	ถ่ายโอนข้อมูลจากต้นทางที่กำหนดไปยังพอร์ตอุปกรณ์ I/O หรือ device
	Start I/O	ถ่ายโอนคำสั่งจากโปรเซสเซอร์ของอุปกรณ์ I/O ไปยังโอเปอเรชันเริ่มต้นของ I/O
	Test I/O	ถ่ายโอนข้อมูลสถานะจากระบบของอุปรกรณ์ I/O ไปยังปลายทางที่กำหนด
การแปลงค่า	Translate	แปลงค่าในส่วนของหน่วยความจำตามตารางที่สัมพันธ์กัน
	Convert	แปลงค่าจากรูปแบบหนึ่งไปรูปแบบหนึ่ง (เช่น แปลงเลขฐานสิบ (Decimal) เป็นเลขฐานสอง (Binary))

โอเปอเรชันทางด้านการโอนถ่ายข้อมูล

- โอเปอเรชันทางด้านการโอนถ่ายข้อมูล จัดเป็นประเภทของคำสั่งที่เป็นพื้นฐานที่สุด ซึ่งจะต้องมีการกำหนด ดังนี้
  - ตำแหน่งของโอเปอแรนด์ต้นทางกับโอเปอแรนด์ปลายทาง แต่ละตำแหน่งอาจจะเป็นหน่วยความจำ รีจิสเตอร์ หรือส่วนบนสุดของ stack ก็ได้
  - ขนาดความยาวของข้อมูลที่จะถ่ายโอนนั้นจะต้องกำหนดให้แน่นอน
  - กำหนดโหมดแอดเดรส (Addressing mode) ของแต่ละโอเปอแรนด์

โอเปอเรชันทางด้านการโอนถ่ายข้อมูล

- ถ้าพิจารณาการทำงานของ CPU การถ่ายโอนข้อมูลเป็นประเภทที่ธรรมดาที่สุด ถ้าทั้งต้นทางและปลายทางเป็นรีจิสเตอร์ CPU ก็แค่ย้ายข้อมูลจากรีจิสเตอร์หนึ่งไปรีจิสเตอร์อีกตัวหนึ่งเท่านั้น ซึ่งเป็นการทำงานภายใน CPU แต่ถ้าโอเปอแรนด์ใดหรือทั้งสองโอเปอแรนด์อยู่ในหน่วยความจำแล้ว CPU จะต้องกระทำสิ่งใดสิ่งหนึ่ง หรือทุกสิ่งดังต่อไปนี้
  - คำนวณแอดเดรสของหน่วยความจำตามการกำหนดโหมดของแอดเดรส
  - ถ้าแอดเดรสอ้างถึงหน่วยความจำเสมือน จะแปลงจากแอดเดรสเสมือนเป็นแอดเดรสจริง
  - อาจกำหนดแอดเดรสในแคช
  - ถ้าไม่กำหนด ให้ออกคำสั่งไปยังโมดูลของหน่วยความจำ

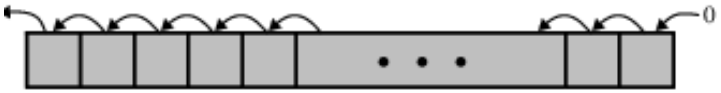
โอเปอเรชันทางด้านตรรกะ

- โอเปอเรชัน **Arithmetic shift**
  - จะกระทำกับข้อมูลเหมือนจำนวนเต็มที่มีเครื่องหมาย และไม่เลื่อนบิตเครื่องหมาย สำหรับการเลื่อนทางขวาแบบนี้ บิตเครื่องหมายจะถูกทำสำเนาแทรกเข้าไปในตำแหน่งบิตที่ถูกเลื่อนออกไป แต่สำหรับการเลื่อนทางซ้ายจะเลื่อนบิตไปทางซ้ายเหมือน logical left shift แต่ไม่ต้องเลื่อนบิตเครื่องหมาย ให้คงบิตนั้นไว้

โอเปอเรชันทางด้านตรรกะ

- **Rotate หรือการหมุน**
  - เป็นการเลื่อนแบบวนรอบเป็นโอเปอเรชันที่เก็บค่าทุกบิตไว้ โดยการหมุน
  - ถ้าเป็นการหมุนขวา (Right Rotate) บิตขวาสุดจะหมุนไปเป็นบิตซ้ายสุด
  - ถ้าเป็นการหมุนซ้าย (Left Rotate) บิตที่อยู่ซ้ายสุดจะหมุนไปเป็นบิตขวาสุด
  - (ทั้งสองกรณีจะต้องมีการตรวจสอบเพื่อให้คงสภาพในกรณีที่บิตซ้ายสุดเป็นเป็นเครื่องหมาย (sign bit) ด้วย

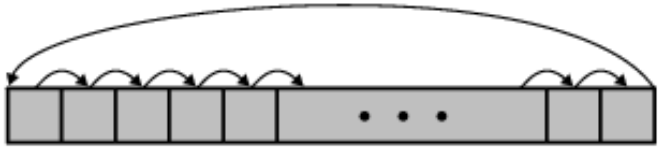
(a) Logical right shift



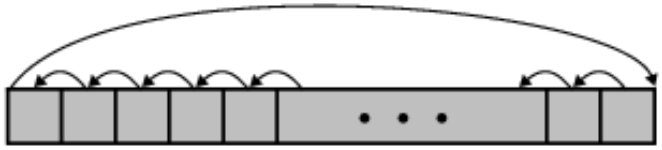
(b) Logical left shift

(c) Arithmetic right shift

(d) Arithmetic left shift



(e) Right rotate



(f) Left rotate

โอเปอเรชันทางด้านตรรกะ

ค่าเริ่มต้น	โอเปอเรชัน	ผลลัพธ์
10100110	Logical right shift (3 บิต)	00010100
10100110	Logical left shift (3 บิต)	00110000
10100110	Arithmetic right shift (3 บิต)	11110100
10100110	Arithmetic left shift (3 บิต)	10110000
10100110	Right rotate (3 บิต)	11010100
10100110	Left rotate (3 บิต)	00110101

โอเปอเรชันการควบคุม

- คำสั่งในกลุ่มต่าง ๆ ที่กล่าวมาแล้วเป็นคำสั่งที่มีการโหลดคำสั่งต่อไปไว้ในหน่วยความจำเรียบร้อยแล้ว คือคำสั่งจะเรียงลำดับคำสั่งก่อนหลังไว้แล้ว
- อย่างไรก็ตามยังมีกลุ่มคำสั่งที่เปลี่ยนลำดับการเอ็กซิคิวต์คำสั่ง เมื่อซีพียูเอ็กซิคิวต์คำสั่งเหล่านี้ จะมีผลทำให้เกิดการปรับปรุงค่าในริจิสเตอร์ PC ซึ่งเป็นที่เก็บตำแหน่งที่อยู่ของคำสั่งที่จะได้รับการเอ็กซิคิวต์ในลำดับถัดไป ยกตัวอย่างเช่น คำสั่ง
  - branch
  - skip
  - procedure

คำสั่ง branch

- หรือบางครั้งเรียกว่า jump
- ใช้สำหรับการสั่งให้ประมวลผลคำสั่งในลำดับต่อไป เกิดขึ้นที่ตำแหน่งที่ระบุ ผ่านพารามิเตอร์ที่เป็นส่วนประกอบของคำสั่ง ส่วนมากคำสั่งประเภทนี้จะอยู่ในประเภทที่เรียกว่า conditional branch ซึ่งจะมีการเปรียบเทียบเงื่อนไขก่อน
- ถ้าเกิดผลตามเงื่อนไขจึงกระโดดไปแอ็ดเดรสตามที่กำหนด (อัปเดท PC ให้เท่ากับแอ็ดเดรสที่กำหนดในโอเปอแรนด์) แต่ถ้าไม่ตรงเงื่อนไขก็จะเพิ่ม PC ตามปกติเพื่อเอ็กซิคิวต์คำสั่งถัดไป

คำสั่ง branch

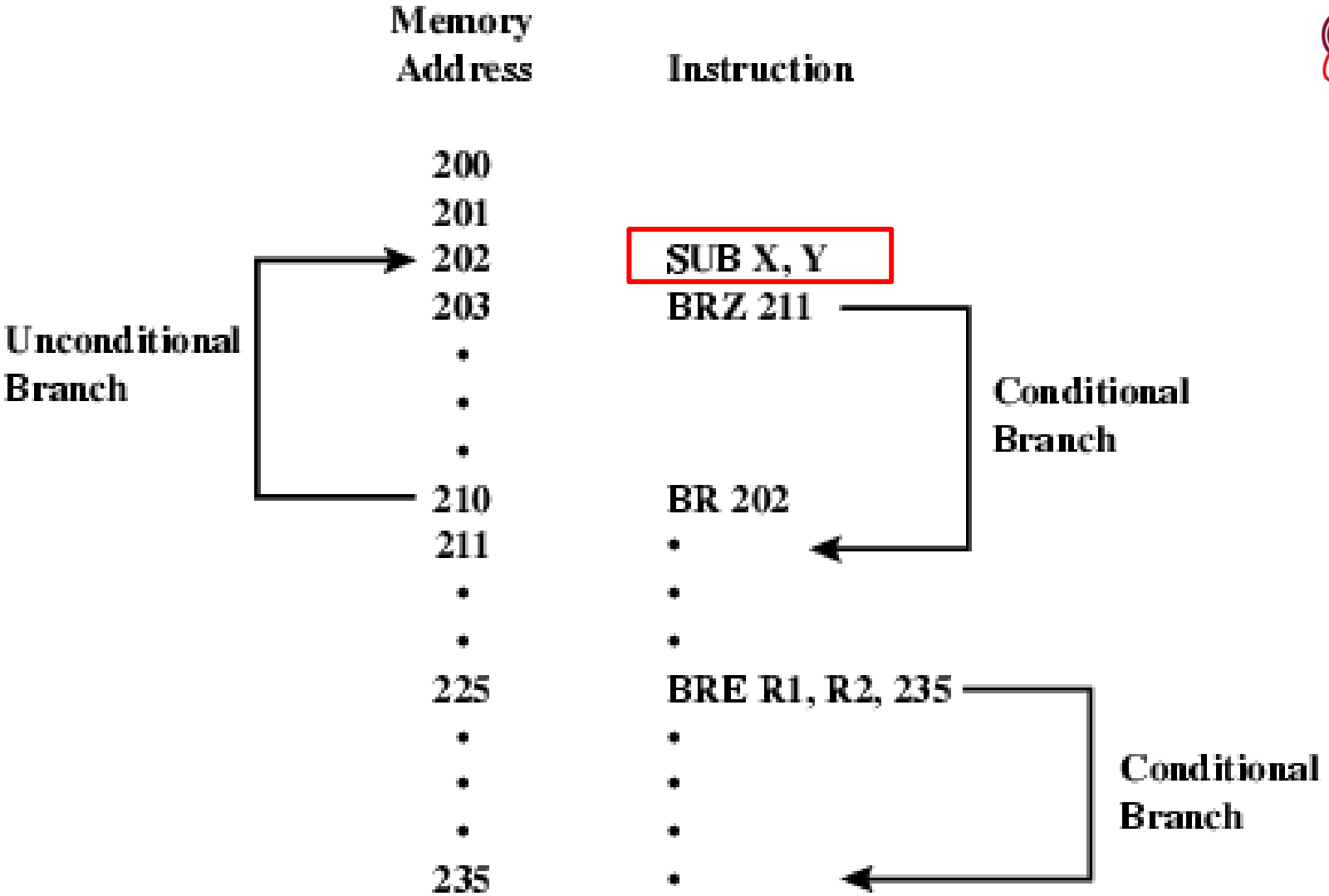
- มี 2 วิธี ในการสร้างเงื่อนไขเพื่อตรวจสอบคำสั่ง conditional branch
- กรณีแรก กำหนดรหัสขนาด 1 บิต (condition code) หรือหลายบิตก็ได้เพื่อรองรับผลลัพธ์ที่เกิดขึ้น เช่นในบางเครื่องคำสั่งสำหรับ 4 กรณีอาจเป็น
  - BRP X กระโดดไปตำแหน่ง X ถ้าผลลัพธ์การคำนวณเป็นบวก
  - BRN X กระโดดไปตำแหน่ง X ถ้าผลลัพธ์การคำนวณเป็นลบ
  - BRZ X กระโดดไปตำแหน่ง X ถ้าผลลัพธ์การคำนวณเป็นศูนย์
  - BRO X กระโดดไปตำแหน่ง X ถ้าผลลัพธ์การคำนวณเกิด overflow

คำสั่ง branch

- ในกรณีที่สอง เกิดขึ้นกับคำสั่งชนิดที่มีการอ้างอิงถึงตำแหน่งที่อยู่ 3 แอ็ดเดรส (three-address instruction) เพื่อทำการเปรียบเทียบและกำหนดตำแหน่งที่กระโดดไปในคำสั่งเดียวกันเลย เช่น

BRE R1, R2, X

- กระโดดไปตำแหน่ง X เมื่อข้อมูลใน R1 = R2
  - การกระโดดสามารถกระโดดไปข้างหน้า (ไปคำสั่งแอดเดรสสูงกว่า) หรือ forward
  - หรือกระโดดย้อนกลับ (แอดเดรสต่ำกว่า) หรือ backward
  - ในตัวอย่างแสดงถึงการกระโดดโดยไม่มีเงื่อนไขและแบบมีเงื่อนไขที่สามารถเกิดการวนรอบคำสั่งได้ คำสั่งในตำแหน่ง 202 ถึง 210 จะถูกเอ็กซิคิวซ์ซ้ำ ๆ จนกว่าผลลัพธ์ของ X-Y เป็น 0


Memory Address
Instruction
200
201
202
SUB X, Y
203
BRZ 211
Unconditional Branch
Conditional Branch
210
BR 202
211
225
BRE R1, R2, 235
Conditional Branch
235

คำสั่ง skip

- คำสั่งในกลุ่มการควบคุมอีกคำสั่งหนึ่งคือ skip ซึ่งคำสั่งนี้จะมีแอดเดรสที่แน่นอน โดยปกติแล้วคำสั่ง skip จะบอกว่ามีหนึ่งคำสั่งที่ถูกข้ามไป ดังนั้นแอดเดรสที่เห็นนี้จะเท่ากับแอดเดรสของคำสั่งต่อไปบวกกับความยาวคำสั่ง เนื่องจากคำสั่ง skip ไม่ต้องการแอดเดรสปลายทาง

คำสั่ง skip

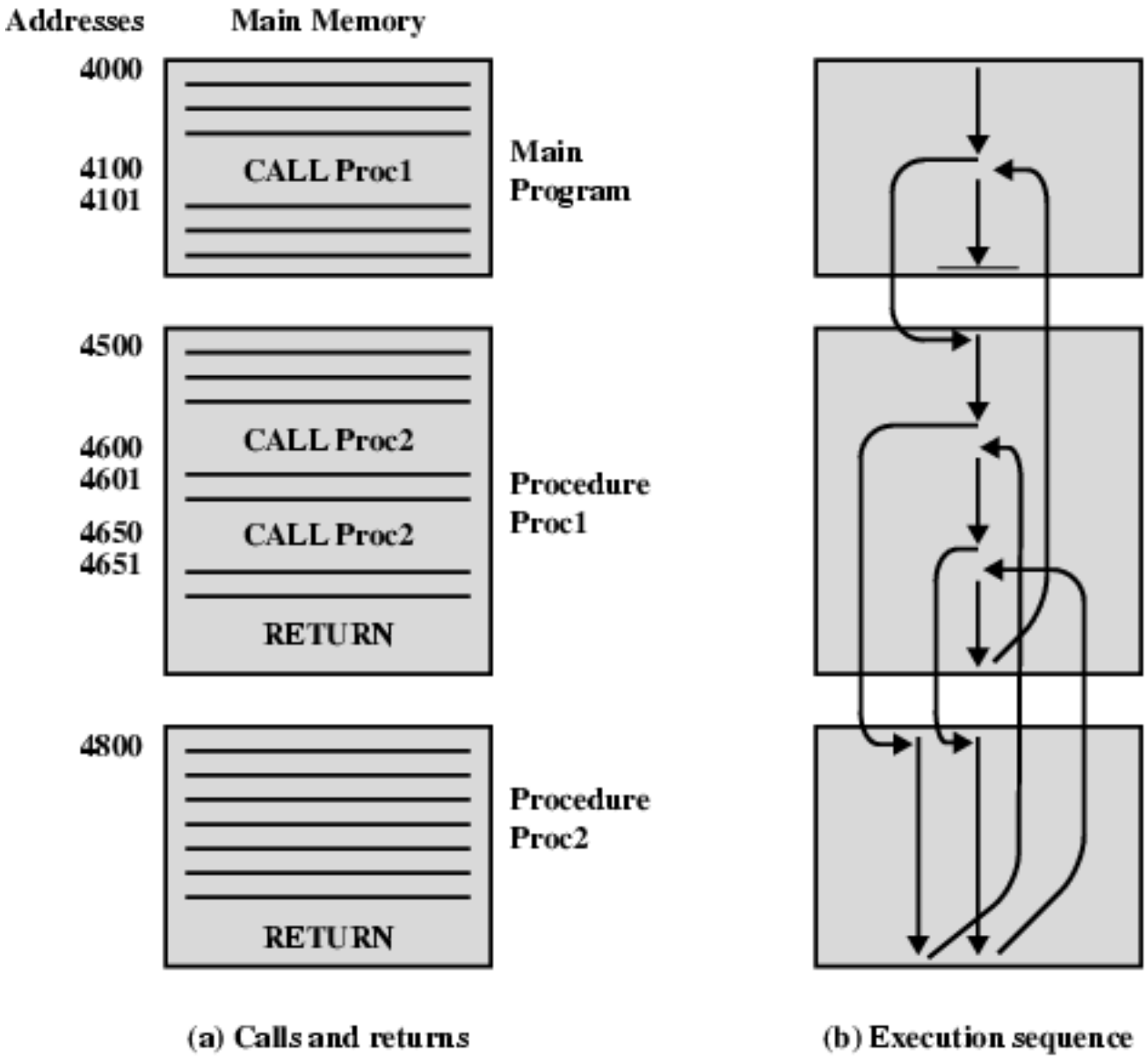
- increment-and-skip-if-zero (ISZ)

```
401      ........
 ..
 ..
409      ISZ  R1
410      BR   401
411      ........
```

  - ในตัวอย่างนี้มีการใช้คำสั่ง 2 คำสั่งที่ทำให้เกิดการวนรอบ
  - รีจิสเตอร์ R1 ถูกกำหนดค่าให้เป็นเลขจำนวนลบของจำนวนรอบที่ต้องการวนซ้ำ
  - เมื่อทำงานมาจนถึงตำแหน่ง 408 ค่าของรีจิสเตอร์ R1 จะถูกเพิ่มขึ้นอีก "1"
    - ถ้า R1 ยังมีค่าเป็นจำนวนลบอยู่ โปรแกรมจะวนย้อนกลับไปประมวลผลที่ตำแหน่ง 401
    - เมื่อ R1 มีค่าเป็น "0" คำสั่งตำแหน่ง 410 จะถูกข้ามจึงไม่ได้ย้อนกลับไปที่ตำแหน่ง 401 อีก

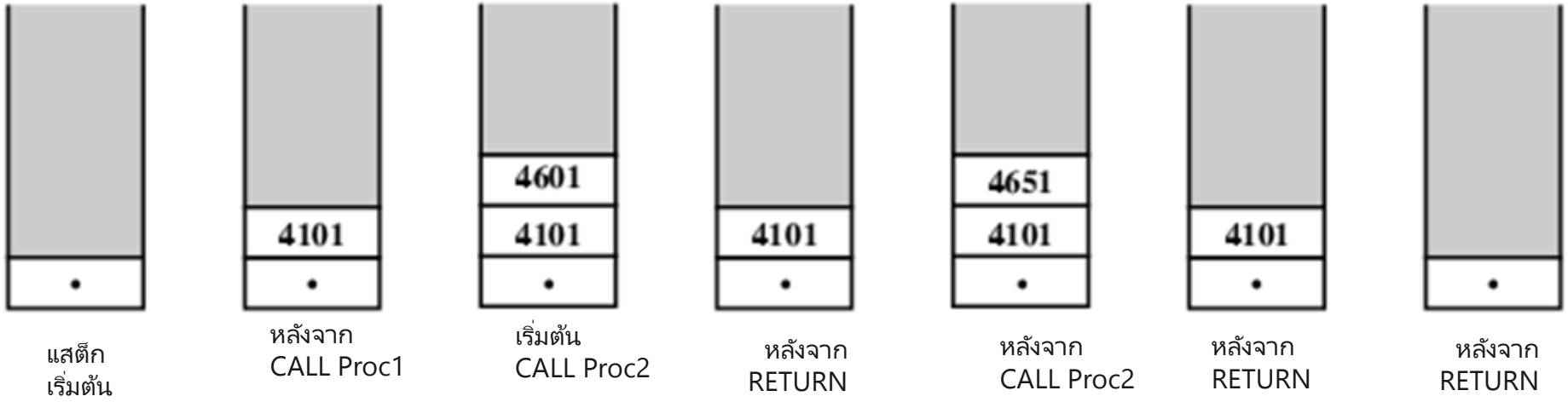
คำสั่ง procedure call

- procedure คือส่วนย่อย ๆ ของโปรแกรมคอมพิวเตอร์ที่รวมอยู่ในโปรแกรมใหญ่
  - ประโยชน์ของ procedure คือ สามารถเรียกใช้ส่วนย่อยของโปรแกรมได้หลายครั้งโดยไม่ต้องเขียนใหม่ทำให้เกิดประสิทธิภาพสูงในการจัดเก็บข้อมูล และยังช่วยให้การเขียนโปรแกรมขนาดใหญ่ทำได้ง่ายโดยการแบ่งเป็นส่วนย่อย ๆ แล้วเขียนโปรแกรมให้เสร็จเรียบร้อยในแต่ละส่วน แล้วค่อยนำมารวมกันภายหลัง
- กลไกการทำงานของ procedure จะมีคำสั่งพื้นฐาน 2 คำสั่ง คือ
  - คำสั่ง **call** เพื่อกระโดดจากตำแหน่งปัจจุบันไปยัง procedure ที่ต้องการ
  - คำสั่ง **return** เพื่อกลับจาก procedure มายังตำแหน่งเดิมที่เรียก procedure


Addresses
Main Memory
4000
4100
4101
CALL Proc1
Main
Program
4500
4600
4601
CALL Proc2
4650
4651
CALL Proc2
RETURN
Procedure
Proc1
4800
RETURN
Procedure
Proc2
(a) Calls and returns
(b) Execution sequence

คำสั่ง procedure call

- เนื่องจากสามารถเรียก procedure ได้จากหลายจุดทำให้ซีพียูต้องมีวิธีการเก็บแอดเดรสเพื่อกลับมายังตำแหน่งเดิมได้ (return address) สามารถเก็บไว้ได้ 3 แห่งคือ รีจิสเตอร์, ส่วนเริ่มต้นของ procedure ที่ถูกเรียกและสแต็ก



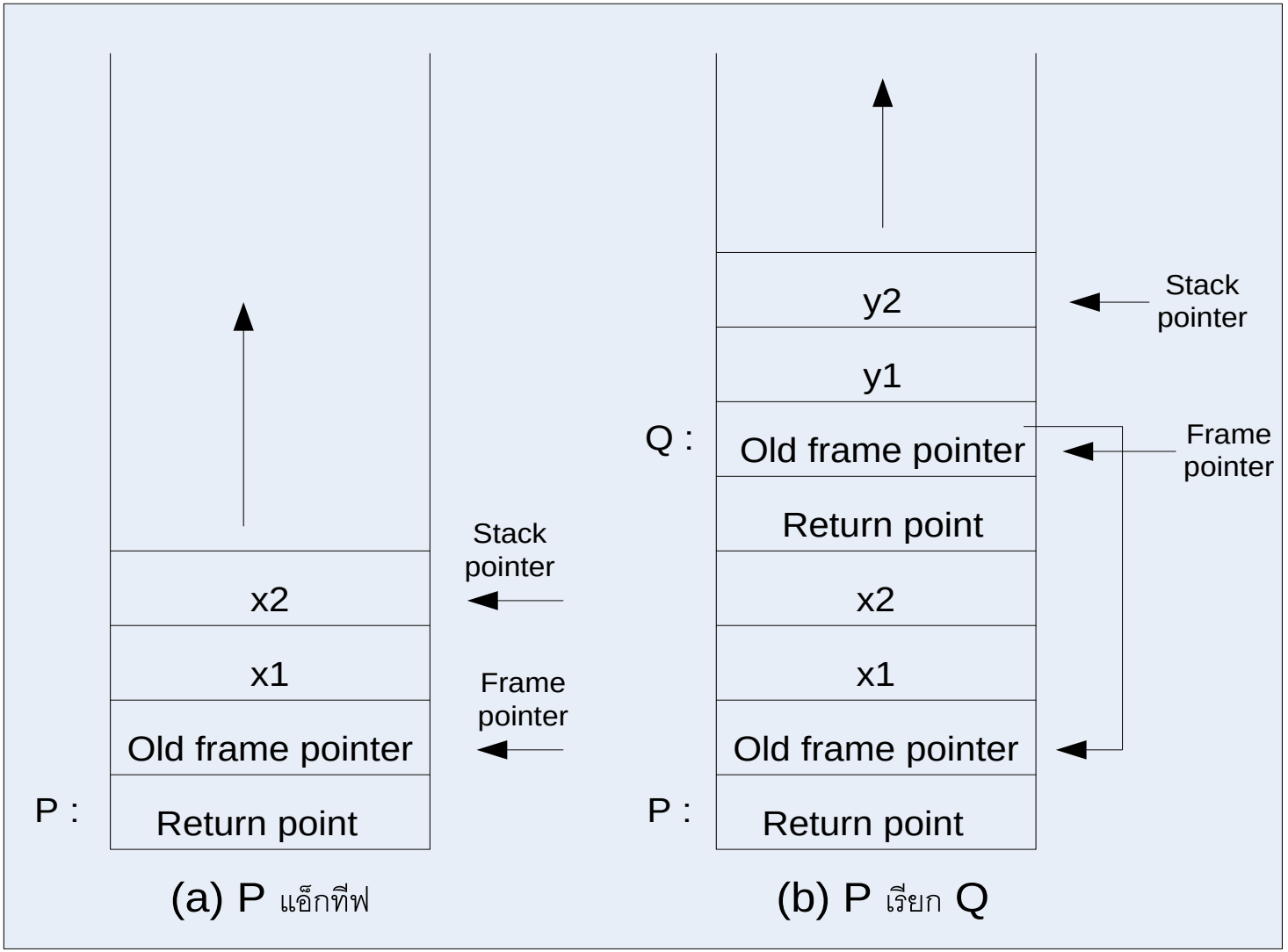
แสต็กเริ่มต้น | หลังจาก CALL Proc1 | เริ่มต้น CALL Proc2 | หลังจาก RETURN | หลังจาก CALL Proc2 | หลังจาก RETURN | หลังจาก RETURN

คำสั่ง procedure call

- นอกจากการเก็บค่าแอดเดรสเพื่อกลับไปทำงานที่เดิมแล้ว การเรียก procedure จะต้องมีการผ่านค่าพารามิเตอร์เข้าไปด้วย
- วิธีการส่งผ่านพารามิเตอร์ที่ยืดหยุ่นที่สุดคือการใช้สแต็ก เมื่อโปรเซสเซอร์มีการเรียก procedure ไม่เพียงเก็บค่าแอดเดรสเดิมแล้ว มันยังเก็บค่าพารามิเตอร์เพื่อส่งผ่านไปยัง procedure ที่ถูกเรียกแล้ว procedure นั้นยังสามารถใช้พารามิเตอร์จากสแต็กได้อีกด้วย
- นอกจากนั้นค่าพารามิเตอร์ที่ส่งกลับก็ยังเก็บบนสแต็กได้ด้วย ชุดของพารามิเตอร์ทั้งหมดที่รวมทั้งแอดเดรสเดิม จะถูกเก็บสำหรับการอ้างอิงใน procedure ที่อ้างอิงถึงสแต็กเฟรม

คำสั่ง procedure call

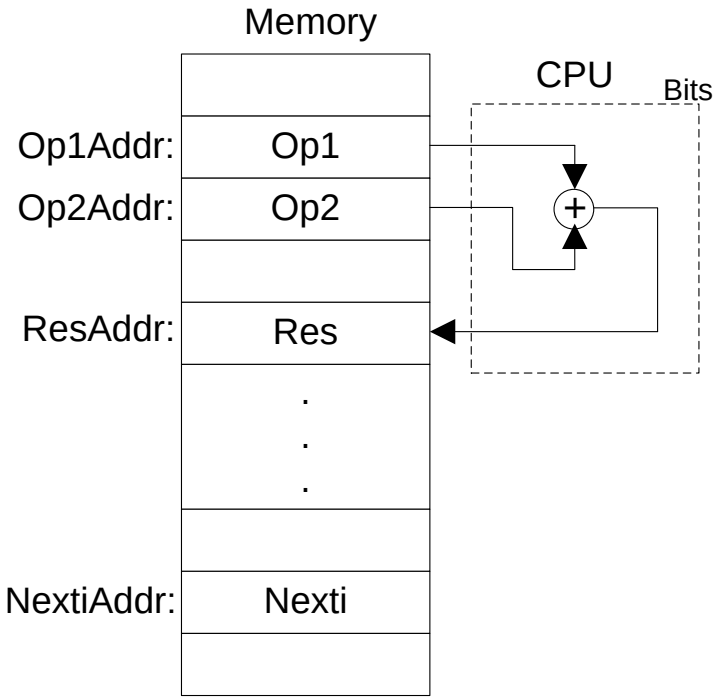
- ตัวอย่างการส่งผ่านค่าพารามิเตอร์จะเป็นดัง Slide หน้าถัดไปซึ่งในตัวอย่างมี procedure ที่ชื่อ P และ Q
  - ใน P มีการประกาศตัวแปร x1 และ x2 แบบโลคอล
  - ส่วน Q ที่สามารถเรียกได้จาก P มีการประกาศตัวแปร y1 และ y2 แบบโลคอลเช่นเดียวกัน
- ในรูปนี้ ที่จุด return point ในแต่ละ procedure เป็นไอเท็มแรกที่ถูกเก็บในสแต็กเฟรมที่สัมพันธ์กัน ถัดไปจะเก็บพอยเตอร์ของจุดเริ่มต้นเฟรมเดิม ซึ่งเป็นสิ่งจำเป็นถ้าจำนวนหรือความยาวของพารามิเตอร์ที่เก็บบนสแต็กมีค่าไม่แน่นอน



Stack Frame เมื่อใช้ใน procedure P และ Q

รหัสคำสั่งแบบ 4-Address Instruction

add, Res, Op1, Op2, Nexti (Res ← Op1 + Op2)



Instruction Format (4-Address Instruction)

8	24	24	24	24
add	ResAddr	Op1Addr	Op2Addr	NextiAddr
Which operation	Where to put result	Where to find operands		Where to find next instruction

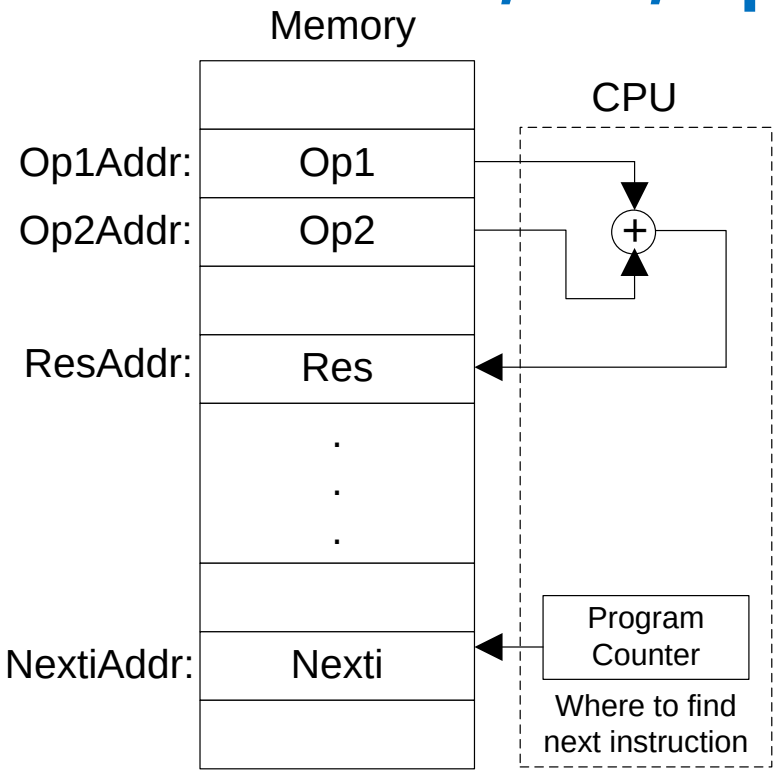
- add เป็น Opcode (ขนาด 1 ไบต์) CPU จะตีความว่าเป็นคำสั่งบวก (จะทำให้ CPU ทำงานเป็นวงจรบวกเลข)
- Res (ขนาด 3 ไบต์) หลังจากดำเนินการแล้วนำผลลัพธ์ไปเก็บที่แอดเดรสใด (ResAddr)
- Op1 , Op2 ตัวดำเนินการนำผลบวกมาจากที่ใด (Op1Add, Op2Add)
- Nexts คำสั่งถัดไปเก็บอยู่ที่แอดเดรสใด (NextiAddr)

- Mnemonics Code) มาใช้แทนรหัสคำสั่งได้ดังนี้ Add, Res, Op1, Op2, Nexti
- 1 คำสั่งใช้หน่วยความจำจำนวน (4 x 3) + 1 = 13 ไบต์

รหัสคำสั่งแบบ 3-Address Instruction

- เป็นการอ้างแอดแดรสแบบ 3 ตำแหน่งโดยเพิ่ม Program Counter (PC) เพื่อใช้ชี้แอดแดรสที่เก็บคำสั่งถัดไป โดยใช้ข้อมูลจำนวน 24 บิต

add, Res, Op1, Op2 (Res ← Op1 + Op2)



Instruction Format (3-Address Instruction)

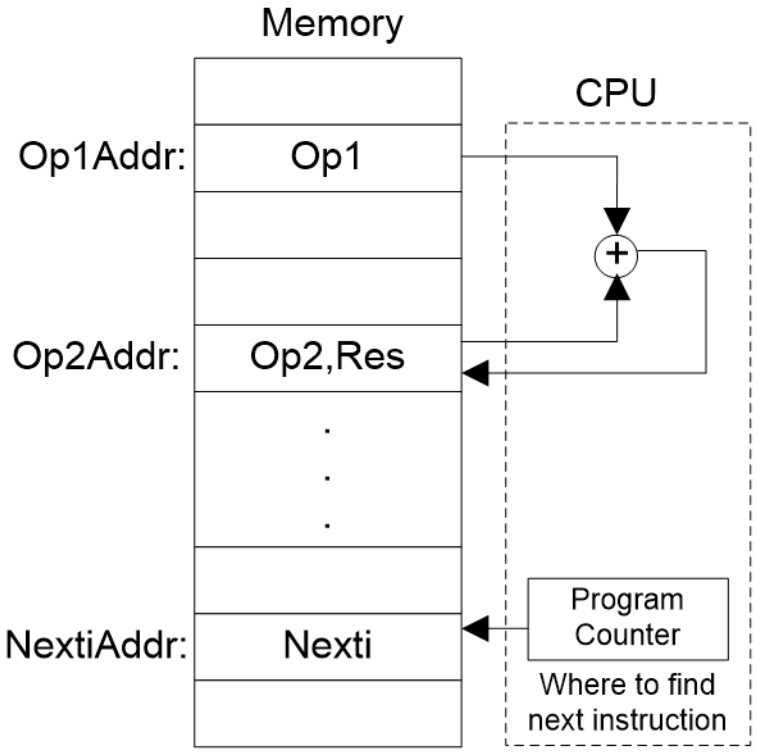
Bits	8	24	24	24
	add	ResAddr	Op1Addr	Op2Addr
	Which operation	Where to put result	Where to find operands	

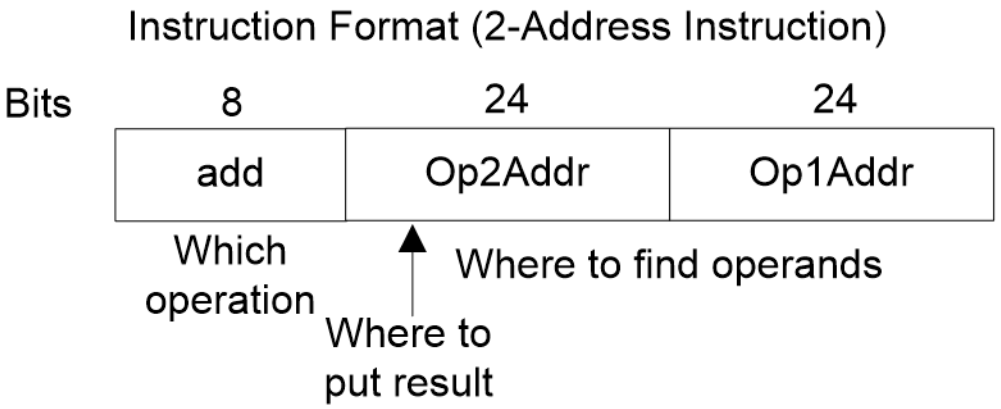
1 คำสั่งใช้หน่วยความจำจำนวน (3 x 3) + 1 = 10 ไบต์

รหัสคำสั่งแบบ 2-Address Instruction

- การอ้างแอดแดรสแบบ 2 ตำแหน่ง ซึ่งพบได้ในเครื่องคอมพิวเตอร์แบบ ISA

add, Op2, Op1 (Res ← Op2 + Op1)

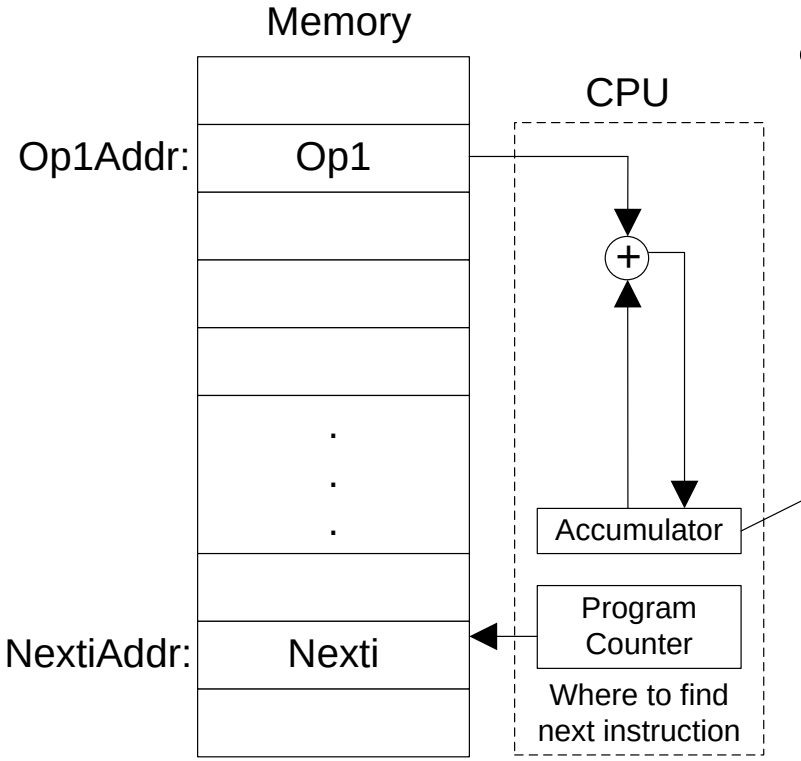




1 คำสั่งใช้หน่วยความจำ จำนวน (2 x 3) + 1 = 7 ไบต์

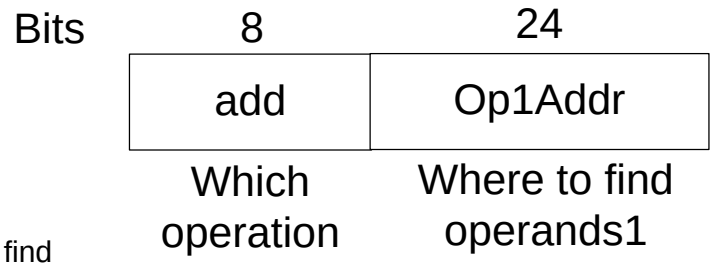
รหัสคำสั่งแบบ 1-Address Instruction

- เป็นการอ้างแอดแดรสแบบ 1 ตำแหน่ง (Single Address Instruction) ภายในไมโครโปรเซสเซอร์จะมีการเพิ่มรีจิสเตอร์เข้าไปอีก 1 ตัวสำหรับจัดเก็บข้อมูลเรียกว่า Accumulator)



add, Op1 (Acc ← Acc + Op1)

Instruction Format (1-Address Instruction)



1 คำสั่งใช้หน่วยความจำ จำนวน 4ไบต์

การเขียนคำสั่ง

- ถ้าหาก Opcode ขนาด 8 บิตจะออกแบบคำสั่งได้ทั้งหมด 256 คำสั่ง และถ้าหากไมโครโปรเซสเซอร์สามารถอ้างหน่วยความจำได้ไม่มาก จำนวนบิตของการอ้างอิงตัวถูกดำเนินการอาจมีน้อยลง
- สำหรับรหัสคำสั่งต่างๆ ที่ไมโครโปรเซสเซอร์เข้าใจจะเป็นรหัสเลขฐานสอง แต่ในการใช้งานจริงจะใช้รหัสช่วยจำ (Memonics) หรือการใช้สัญลักษณ์ (Symbolic Representation) เพื่อให้เข้าใจได้มากขึ้นได้แก่

การเขียนคำสั่ง

รหัสช่วยจำ	ความหมาย	
ADD	Addition	(บวก)
SUB	Subtraction	(ลบ)
MPY	Multiply	(คูณ)
DIV	Divide	(หาร)
LOAD	Load	Data from Memory (โหลดข้อมูลจากหน่วยความจำ)
STOR	Store	Data to Memory (เก็บข้อมูลลงหน่วยความจำ)

ตัวอย่างการเขียนคำสั่ง

- ถ้าหากเขียนคำสั่งเป็น ADD A, B หมายความว่า
  - คำสั่งนี้เป็นการอ้างแอดเดรสแบบ 2 ตำแหน่ง โดยทำการบวกค่าที่เก็บอยู่ในแอดเดรส B กับค่าที่เก็บอยู่ในแอดเดรส A แล้วนำผลลัพธ์ที่ได้ไปเก็บในแอดเดรส A
- ถ้ามีไมโครโปรเซสเซอร์ที่อ้างแอดเดรสตำแหน่งเดียว จะเขียนคำสั่งคูณค่าที่อยู่ในหน่วยความจำ D และหน่วยความจำ C ได้ดังนี้

```
LOAD  D;    นำค่าจากหน่วยความจำ D มาเก็บใน AC
MPY   C;    คูณค่าข้อมูลใน AC กับค่าในหน่วยความจำ C
```

ตัวอย่างการเขียนคำสั่ง

- ถ้าหากเปรียบเทียบไมโครโปรเซสเซอร์ที่มีการอ้างแอดเดรสแบบ 1,2 และ 3 ตำแหน่งโดยต้องการเขียนคำสั่งให้ประมวลผล Y = (A – B) / (C + D x E) จะเขียนคำสั่งได้ดังนี้ (หน่วยความจำชั่วคราว T)

$$Y = \frac{A - B}{C + (D \times E)}$$

Instruction		Comment
SUB	Y, A, B	Y ← A – B
MPY	T, D, E	T ← D × E
ADD	T, T, C	T ← T + C
DIV	Y, Y, T	Y ← Y ÷ T

(a) Three-address instructions

Instruction		Comment
MOVE	Y, A	Y ← A
SUB	Y, B	Y ← Y – B
MOVE	T, D	T ← D
MPY	T, E	T ← T × E
ADD	T, C	T ← T + C
DIV	Y, T	Y ← Y ÷ T

(b) Two-address instructions

Instruction		Comment
LOAD	D	AC ← D
MPY	E	AC ← AC × E
ADD	C	AC ← AC + C
STOR	Y	Y ← AC
LOAD	A	AC ← A
SUB	B	AC ← AC – B
DIV	Y	AC ← AC ÷ Y
STOR	Y	Y ← AC

(c) One-address instructions

รูปแบบคำสั่งและโหมดของแอดเดรส

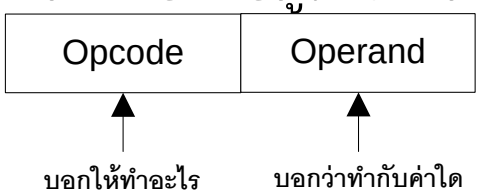
- รูปแบบของรหัสคำสั่งส่วนมากจะมีมากกว่า 1 รูปแบบ เมื่อมีการทำคำสั่งรหัสคำสั่งจะถูกอ่านเข้ามาเก็บในรีจิสเตอร์คำสั่ง (Instruction Register) จากนั้นคำสั่งจะถูกตีความแล้วทำตามคำสั่งนั้น
  - โดยรหัสคำสั่งที่ใช้กันมากในปัจจุบันจะประกอบด้วย 2 ส่วนคือ
  - ส่วนที่เป็นรหัสดำเนินการ (Opcode) หรือ “อ้อปโค้ด” ที่บอกให้คำสั่งทำอะไร
  - ส่วนอ้างอิงตัวถูกดำเนินการ (Operand Reference) หรือเรียกว่า “โอเปอร์แรนด์” เป็นตัวที่บอกให้ทำกับข้อมูลที่เก็บอยู่ที่ใด
- ไมโครโปรเซสเซอร์ที่ต่างรุ่นกันจะมีรหัสคำสั่งที่แตกต่างกัน และอาจมีความยาวของรหัสคำสั่งแตกต่างกัน โดยความยาวของรหัสคำสั่งจะมีผลต่อขนาดของหน่วยความจำด้วย
- การเขียนคำสั่งเช่น MOV Oper1, Oper2 เป็นการใช้คำสั่งโอนย้ายข้อมูลจากโอเปอร์แรนด์ตัวที่ 2 ไปยังโอเปอร์เรนด์ตัวที่ 1

รูปแบบคำสั่งและโหมดของแอดเดรส

- การอ้างแอดเดรสจะเป็นโอเปอร์แรนด์ที่อยู่ในรูปแบบของคำสั่ง ซึ่งมักจะเป็นข้อมูลขนาดเล็ก แต่ข้อมูลนี้จะต้องติดต่อกับหน่วยความจำขนาดใหญ่ได้
- การอ้างแอดเดรสมีหลายรูปแบบดังนี้
  - การกำหนดแอดเดรสแบบให้ค่าตรง (Immediate Addressing Mode)
  - การกำหนดแอดเดรสโดยใช้รีจิสเตอร์ (Register Addressing Mode)
  - การกำหนดแอดเดรสโดยตรง (Direct Addressing Mode)
  - การกำหนดแอดเดรสผ่านรีจิสเตอร์โดยอ้อม (Register Indirect Addressing Mode)
  - การกำหนดแอดเดรสแบบแทนที่ (Indexed Addressing Mode)
  - การกำหนดแอดเดรสแบบสัมพันธ์ (Relative Addressing Mode)
  - การกำหนดแอดเดรสโดยใช้สแตก (Stack Addressing Mode)

การกำหนดแอดเดรสแบบให้ค่าตรง

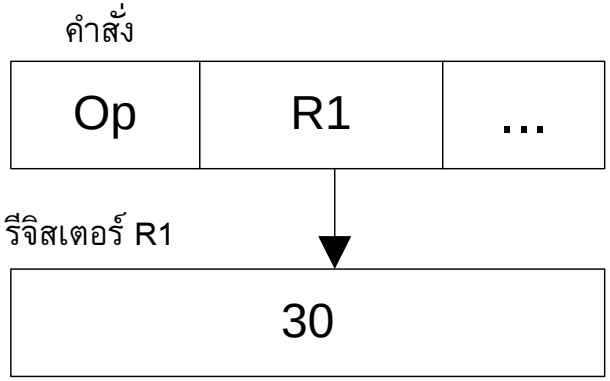
- เป็นวิธีการอ้างแอดเดรสแบบง่ายที่สุดและมีการใช้งานบ่อยที่สุด
- ค่าของข้อมูลจะอยู่รวมกับรหัสคำสั่ง
- การอ้างแอดเดรสแบบนี้จะไม่มีการอ้างอิงหน่วยความจำและทำงานได้เร็ว
- ขนาดของคำสั่งขึ้นอยู่กับขนาดของข้อมูล โดยมีรูปแบบคำสั่งดังนี้



- ถ้าออปโค้ดมีขนาด 1 ไบต์และข้อมูลที่ต้องการประมวลผลมีขนาด 1 ไบต์จะทำให้รหัสคำสั่งมีขนาด 2 ไบต์
- แต่หากข้อมูลที่ต้องการประมวลผลมีขนาด 2 ไบต์ก็จะทำให้รหัสคำสั่งมีขนาด 3 ไบต์ด้วยเช่น
  - MOV AL, 12H ; เป็นการนำค่า 12 ฐานสิบหกไปเก็บในรีจิสเตอร์ AL
  - MOV AX, 1234H ; เป็นการนำค่า 1234 ฐานสิบหกไปเก็บในรีจิสเตอร์ AX
  - LOAD X, #1000 ; เป็นการโหลดข้อมูล 1000 ไว้ที่ตัวแปร X

การกำหนดแอดเดรสโดยใช้รีจิสเตอร์

- เป็นวิธีการติดต่อกับข้อมูลที่อยู่ในรีจิสเตอร์ ไม่มีการติดต่ออยู่กับตำแหน่งของหน่วยความจำตำแหน่งใด
- รหัสคำสั่งส่วนที่เป็นโอเปอร์แรนด์จะประกอบด้วยบิตข้อมูลที่อ้างไปที่รีจิสเตอร์ที่ต้องการติดต่อ
- รหัสคำสั่งจะมีขนาดเล็ก เนื่องจากโอเปอร์แรนด์ที่ต้องการติดต่อมีขนาด 3-5 บิตก็จะสามารถแทนค่ารีจิสเตอร์ได้ถึง 8-32 ตัว และยังทำงานได้เร็ว เนื่องจากไม่ต้องติดต่อกับหน่วยความจำ

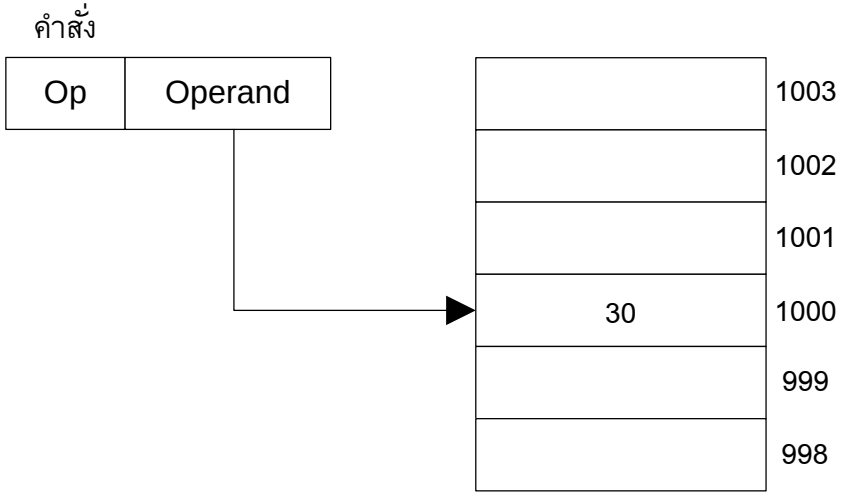


LOAD X,(R1) ; เป็นการโหลดค่าจากรีจิสเตอร์ R1 ไว้ที่ตัวแปร X (ถ้ารีจิสเตอร์ R1 มีค่า 30 ดังนั้น X จะมีค่าเท่ากับ 30)

MOV AX, BX ; เป็นการนำเอาข้อมูลที่อยู่ในรีจิสเตอร์ BX มาเก็บใน AX

การกำหนดแอดเดรสโดยตรง

- เป็นวิธีการอ้างแอดเดรสโดยตรง หรือ เป็นวิธีการอ้างตำแหน่งบนหน่วยความจำโดยตรง (Memory Direct Addressing)
- ไบต์แรกเป็นออปโค้ด ไบต์ถัดไปเป็นแอดเดรสของหน่วยความจำ
- การอ้างแบบนี้ในปัจจุบันไม่ค่อยนิยมใช้ ถูกใช้ในคอมพิวเตอร์รุ่นแรกๆ เนื่องจากใช้เวลานานในการประมวลผล
  - เช่น LOAD X, [1000]; เป็นการโหลดข้อมูลที่แอดเดรส 1000 ไว้ที่ตัวแปร X (ถ้าที่แอดเดรส 1000 มีค่า 30 ดังนั้น X จะมีค่าเท่ากับ 30)

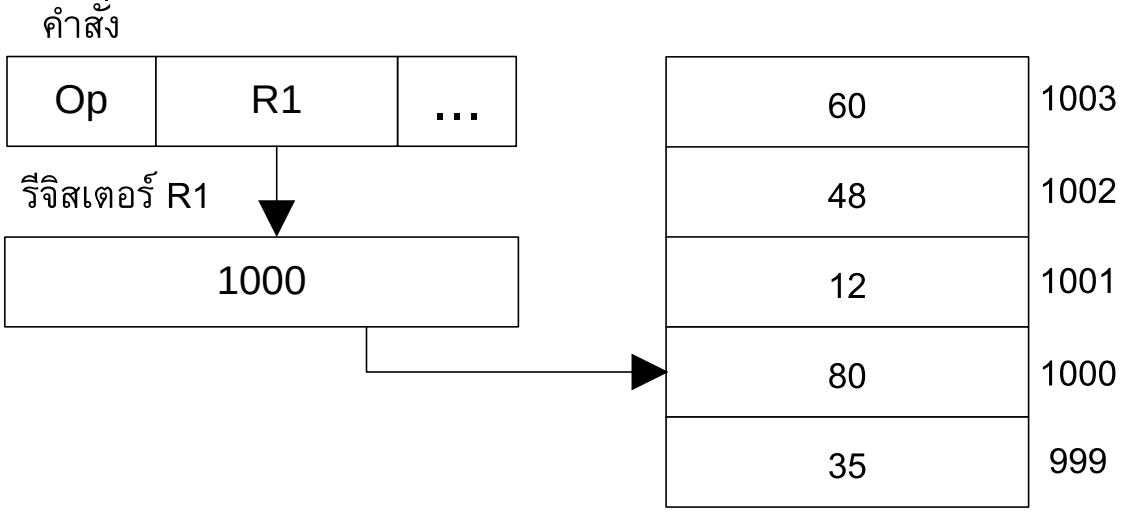


ตัวอย่าง

MOV DL, [2400]; อ่านข้อมูลจากแอดเดรส 2400 มาเก็บในรีจิสเตอร์ DL

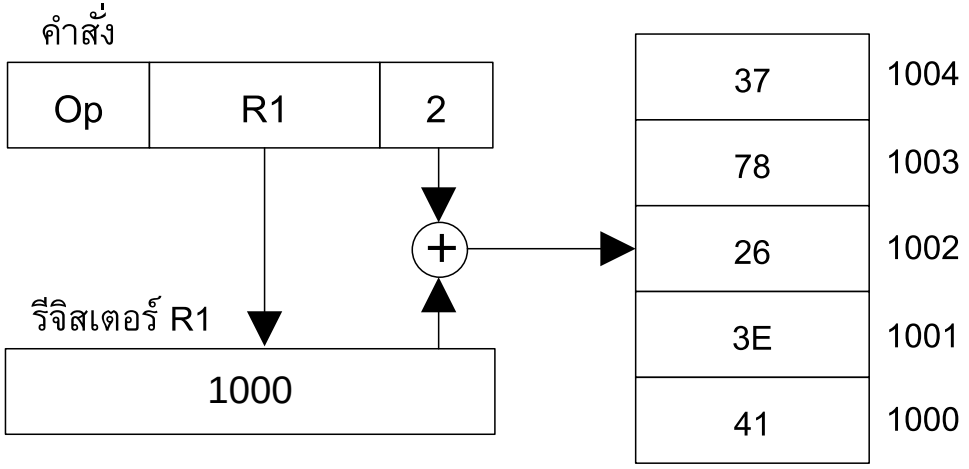
การกำหนดแอดเดรสผ่านรีจิสเตอร์โดยอ้อม

- เป็นการอ้างตำแหน่งโดยอ้อมโดยใช้รีจิสเตอร์ คล้ายกับการอ้างตำแหน่งโดยตรง แต่แทนที่จะใช้โอเปอร์แรนด์ในการอ้างตำแหน่งหน่วยความจำก็จะใช้รีจิสเตอร์เป็นตัวชี้ตำแหน่งแทน
- ในรหัสคำสั่งส่วนแรกจะเป็นออปโค้ด ส่วนต่อมาเป็นหมายเลขรีจิสเตอร์ที่เก็บตำแหน่งหน่วยความจำที่ต้องการติดต่อ
  - เช่น LOAD X, [R1]; โหลดข้อมูลจากแอดเดรสที่เก็บอยู่ในรีจิสเตอร์ R1 มาเก็บในตัวแปร X ถ้า R1 เก็บค่า 1000 จะทำให้ค่าในตัวแปร X มีค่าเท่ากับ 80



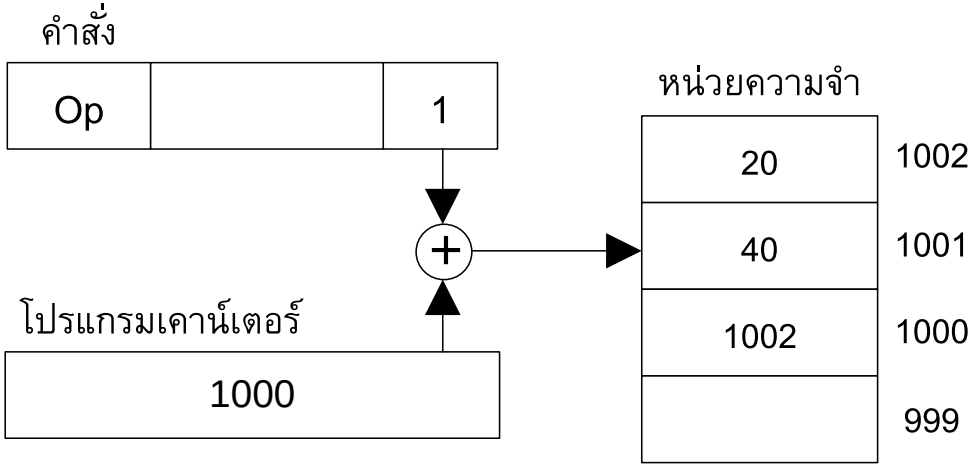
การกำหนดแอดเดรสแบบแทนที่

- การอ้างแอดเดรสแบบนี้ จะนำการอ้างแอดเดรสแบบโดยตรงและแบบโดยอ้อมผ่านรีจิสเตอร์มาประยุกต์ใช้ร่วมกัน
- รหัสคำสั่งจะแบ่งออกเป็น 3 ส่วน ส่วนแรกเป็นอ้อปโค้ด ส่วนที่สองเป็นรีจิสเตอร์ที่เก็บค่าตำแหน่งแอดเดรส ส่วนที่สามจะเป็นค่าคงที่ซึ่งระบุตำแหน่งของข้อมูลที่อยู่ห่างจากรีจิสเตอร์ที่กำหนดแอดเดรส โดยมีรูปแบบคำสั่งดังนี้
  - LOAD X, [R1] + Constant เป็นการโหลดข้อมูลจากแอดเดรสบนหน่วยความจำที่เกิดจากค่าในรีจิสเตอร์บวกกับค่าคงที่ไว้ที่ตัวแปร X (ถ้าในรีจิสเตอร์มีค่า 1000, ค่าคงที่เท่ากับ 2 และค่าที่แอดเดรส 1002 มีค่า 26 ดังนั้น X จะมีค่าเท่ากับ 26



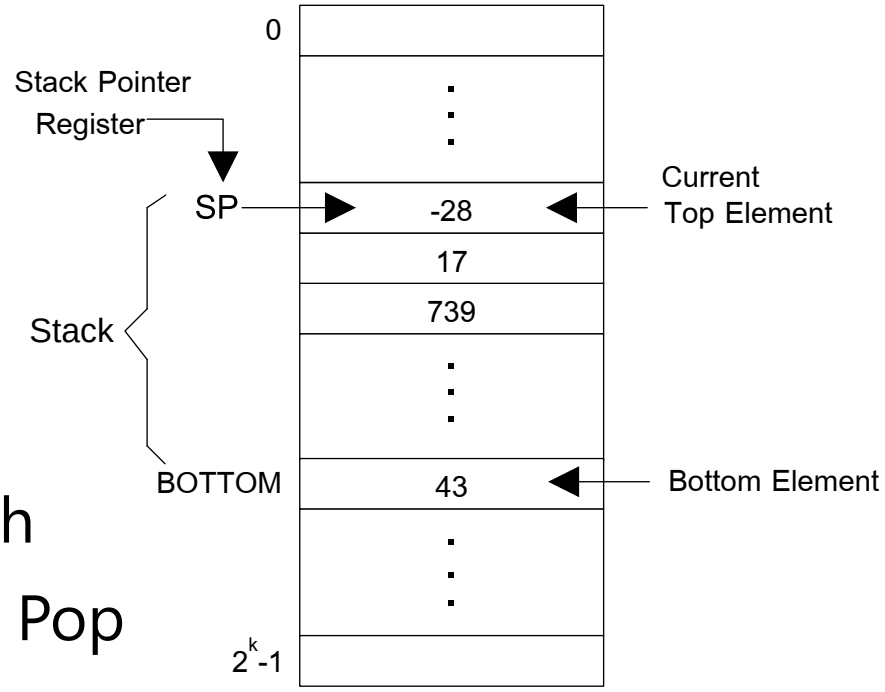
การกำหนดแอดเดรสแบบสัมพันธ์

- การกำหนดแอดเดรสแบบนี้คล้ายกับการกำหนดแอดเดรสแบบแทนที่ เพียงแต่เป็นการกำหนดโดยอ้างอิงกับโปรแกรมเคาน์เตอร์ (Program Counter: PC)
- ดังนั้นค่าคงที่จะบวกกับค่าในโปรแกรมเคาน์เตอร์แล้วได้ค่าแอดเดรสที่แท้จริง ซึ่งค่าแอดเดรสที่แท้จริงนี้เป็นแอดเดรสในหน่วยความจำ
- ดังนั้นค่าที่อยู่ในแอดเดรสดังกล่าวจึงเป็นค่าที่นำมาใช้งาน รูปแบบคำสั่งจะเป็นดังนี้
  - LOAD X, PC+Constant เป็นการโหลดข้อมูลจากแอดเดรสบนหน่วยความจำที่เกิดจากค่าในโปรแกรมเคาน์เตอร์บวกกับค่าคงที่ไว้ที่ตัวแปร X (ถ้าในโปรแกรมเคาน์เตอร์มีค่า 1000, ค่าคงที่เท่ากับ 1 และค่าที่แอดเดรส 1001 มีค่า 40 ดังนั้น X จะมีค่าเท่ากับ 40

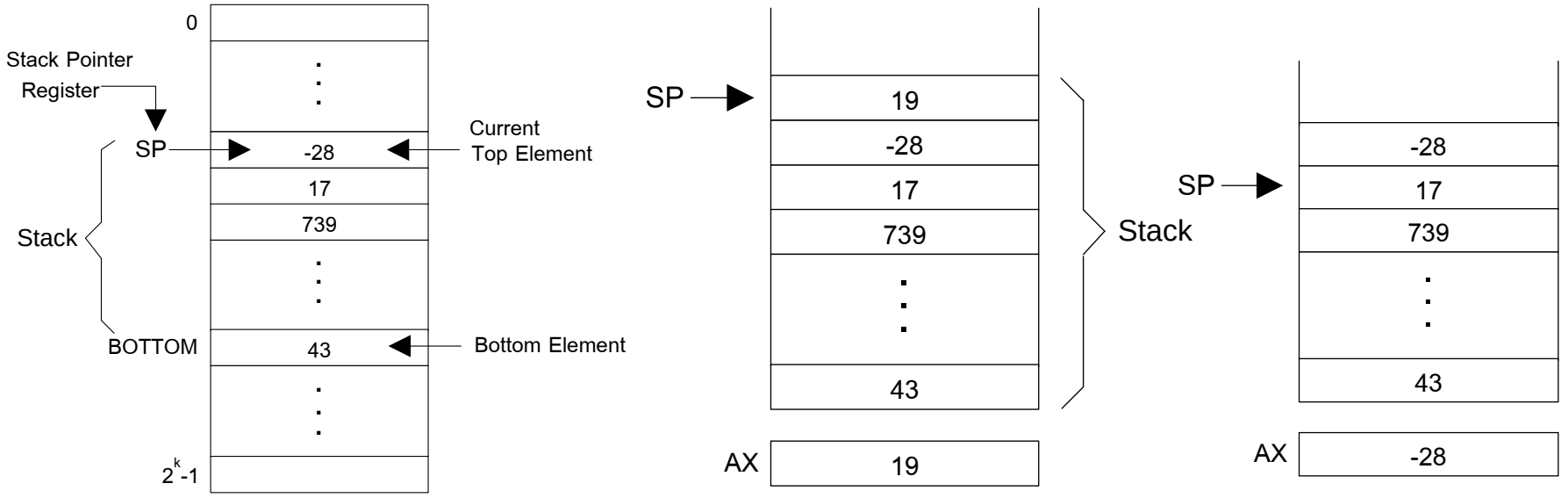


การกำหนดแอดเดรสโดยใช้สแตก

- สแตกเป็นหน่วยความจำเฉพาะ ที่ถูกอ้างอิงโดยรีจิสเตอร์ชี้สแตกหรือสแตกพอยเตอร์ (Stack Pointer) หรือเรียกว่า รีจิสเตอร์ SP
- เป็นการเก็บข้อมูลแบบอาร์เรย์ (Array) ซ้อนๆ กันอยู่เมื่อมีการใช้งาน
- โดยข้อมูลจะถูกเขียนลงบนส่วนบนของหน่วยความจำก่อน จากนั้นสแตกพอยเตอร์ จะชี้ไปยังแอดเดรสตำแหน่งถัดไป
- การนำข้อมูลไปใช้จะนำจากตำแหน่งที่พอยเตอร์ชี้เสมอ ดังรูป
- การเขียนข้อมูลลงสแตกเรียกว่า Push
- การอ่านข้อมูลออกจากสแตกเรียกว่า Pop



การกำหนดแอดเดรสโดยใช้สแตก



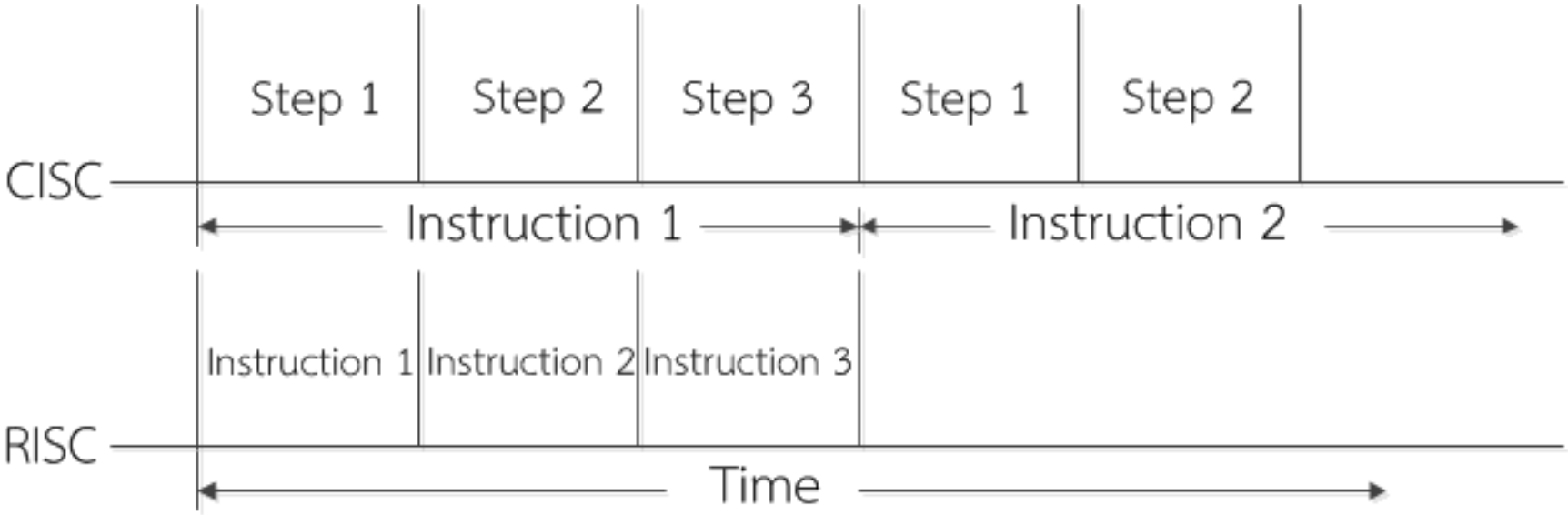
ตัวอย่างสแตกในหน่วยความจำ หลังจากทำคำสั่ง Push AX หลังจากทำคำสั่ง Pop AX

- หลังจากทำคำสั่ง Push AX จะนำเอาข้อมูล 19 เขียนทับทางด้านบน โดยข้อมูลที่พุชลงไปใหม่จะถูกเขียนในตำแหน่งที่รีจิสเตอร์ SP ชี้อยู่ ซึ่งเป็นหน่วยความจำที่อยู่ด้านบน
- แต่หากมีข้อมูลในสแตกแล้วมีการป็อปข้อมูลออกมา จะทำให้ข้อมูลที่อยู่บนสุดถูกอ่านมาเก็บในรีจิสเตอร์ จากนั้นรีจิสเตอร์ SP จะชี้ในตำแหน่งถัดลงมา

สถาปัตยกรรมแบบ CISC และ RISC

- สถาปัตยกรรมของไมโครโปรเซสเซอร์ หากแบ่งสถาปัตยกรรมตามชุดคำสั่งสามารถแบ่งออกได้เป็น 2 กลุ่ม คือสถาปัตยกรรมแบบ CISC (Complex Instruction Set Computer) และสถาปัตยกรรมแบบ RISC (Reduced Instruction Set Computer)
  - ไมโครโปรเซสเซอร์รุ่นเก่าๆ จะเป็นสถาปัตยกรรมแบบ CISC ทั้งสิ้น แต่ในปัจจุบันจะแยกความแตกต่างระหว่างสถาปัตยกรรมของไมโครโปรเซสเซอร์ทั้ง 2 แบบได้ยาก เนื่องจากชุดคำสั่งทั้งสองถูกออกแบบให้มีลักษณะใกล้เคียงกัน
  - ในการทำคำสั่งแบบ CISC จะมีกระบวนการย่อยๆ ประกอบอยู่ภายใน แต่ละคำสั่งจะมีขั้นตอนการทำงานที่ไม่เท่ากันได้และมีขนาดของรหัสคำสั่งที่ไม่เท่ากันได้
  - ส่วนคำสั่งแบบ RISC ทุกคำสั่งของไมโครโปรเซสเซอร์จะมีความยาวของรหัสคำสั่งเท่ากันทั้งหมด

สถาปัตยกรรมแบบ CISC และ RISC



สถาปัตยกรรมแบบ CISC

- สถาปัตยกรรมแบบ CISC ออกแบบให้ไมโครโปรเซสเซอร์มีชุดคำสั่งเป็นจำนวนมาก บางคำสั่งสามารถทำงานที่ซับซ้อนได้และบางคำสั่งสามารถอ้างแอดเดรสได้หลายแบบ เพื่อให้นักโปรแกรมสามารถเขียนโปรแกรมได้ง่าย
  - การอ้างแอดเดรสได้หลายแบบจะทำให้คำสั่งทำงานได้หลากหลาย แต่ก็ทำให้คำสั่งของไมโครโปรเซสเซอร์มีความยาวที่แตกต่างกันได้ ทำให้โครงสร้างภายในของไมโครโปรเซสเซอร์มีจำเป็นต้องมีรีจิสเตอร์เป็นจำนวนมาก
  - การเพิ่มจำนวนรีจิสเตอร์จะทำให้โครงสร้างภายในมีความหนาแน่นของวงจรซับซ้อนยิ่งขึ้น
  - คำสั่งของไมโครโปรเซสเซอร์ 80x86 โดยทั่วๆ ไปจะมีความยาวอยู่ระหว่าง 2-6 ไบต์ ส่วนคำสั่งที่ทำงานยากๆ จะใช้หน่วยความจำถึง 13 ไบต์
  - เวลาที่ใช้แต่ละคำสั่งจะใช้เวลามาก จึงมีการแก้ปัญหาโดยนำการทำงานแบบไปป์ไลน์ (Pipelining) มาใช้ แต่ก็เกิดปัญหาใหม่ เมื่อมีการทำคำสั่งบางคำสั่งที่มีความยาวไม่เท่ากัน

สถาปัตยกรรมแบบ RISC

- สถาปัตยกรรมแบบ RISC ขนาดของแต่ละคำสั่งจะมีขนาดเท่ากัน (Fixed Instruction Length) ทำให้การทำงานแบบไปป์ไลน์สามารถช่วยเพิ่มประสิทธิภาพในการทำงานของไมโครโปรเซสเซอร์ได้ดีขึ้น
  - โดยปกติจะออกแบบชุดคำสั่งให้มีความยาวเท่ากับขนาดของบัสข้อมูล เช่น ระบบบัสแบบ 32 บิต ก็จะออกแบบคำสั่งให้มีความยาว 32 บิต
  - แต่ละคำสั่งจะทำงานเพียงขั้นตอนเดียวและต้องทำงานให้เสร็จภายในสัญญาณนาฬิกา 1 ลูก (One Instruction per Cycle)
  - ไมโครโปรเซสเซอร์ที่มีคำสั่งแบบ RISC นั้น จะมีโหมดการอ้างแอดเดรสไม่มากนักและเป็นการอ้างแอดเดรสแบบง่าย โดยใช้รีจิสเตอร์เป็นหลัก
  - สำหรับการติดต่อกับหน่วยความจำภายนอกเพื่อดึงข้อมูลหรือเก็บข้อมูล จะใช้คำสั่งโหลด (Load) และสโตร์ (Store) เท่านั้น ส่วนคำสั่งอื่นๆ จะเป็นการกระทำระหว่างรีจิสเตอร์ด้วยกัน

สถาปัตยกรรมแบบ CISC และ RISC

- ไมโครโปรเซสเซอร์แบบ RISC คำสั่งมีขนาดคงที่ ทำให้จำนวนคำสั่งในไมโครโปรเซสเซอร์มีไม่กี่คำสั่ง มีคำสั่งได้จำนวนจำกัดและต้องใช้รีจิสเตอร์ในการทำงานเป็นจำนวนมาก
- ไมโครโปรเซสเซอร์แบบ CISC สามารถเพิ่มคำสั่งได้จำนวนมากขึ้น
- ในปัจจุบันไมโครโปรเซสเซอร์แบบ RISC ได้พัฒนาเกินไมโครโปรเซสเซอร์แบบ CISC ไปแล้ว (ตั้งแต่ปี ค.ศ. 1995 แทบไม่มีไมโครโปรเซสเซอร์แบบ CISC รุ่นใหม่ๆ เข้าสู่ตลาดเลย)
- ในด้านการเขียนโปรแกรมหรือการสนับสนุนของคอมไพเลอร์จะพบว่าคำสั่งแบบ CISC มีชุดคำสั่งจำนวนมาก มีคำสั่งของการทำงานที่ซับซ้อนติดมากับไมโครโปรเซสเซอร์อยู่แล้ว ทำให้การเขียนโปรแกรมทำได้ง่ายขึ้น โปรแกรมมีขนาดเล็กและใช้หน่วยความจำไม่มาก
- สำหรับการเขียนโปรแกรมด้วยภาษาระดับสูง สำหรับไมโครโปรเซสเซอร์แบบ RISC จะต้องแปลออกเป็นคำสั่งภาษาเครื่องหลายๆ คำสั่ง ตัวแปลภาษาต้องพยายามนำตัวถูกกระทำหรือโอเปอร์แรนด์ต่างๆ ที่ใช้ในการคำนวณไปเก็บในรีจิสเตอร์ให้มากที่สุดแทนที่จะเก็บไว้ในหน่วยความจำ

ไมโครโปรเซสเซอร์และการประมวลผล (Microprocessor and Process)

Chapter 5

Computer Organization and Architecture

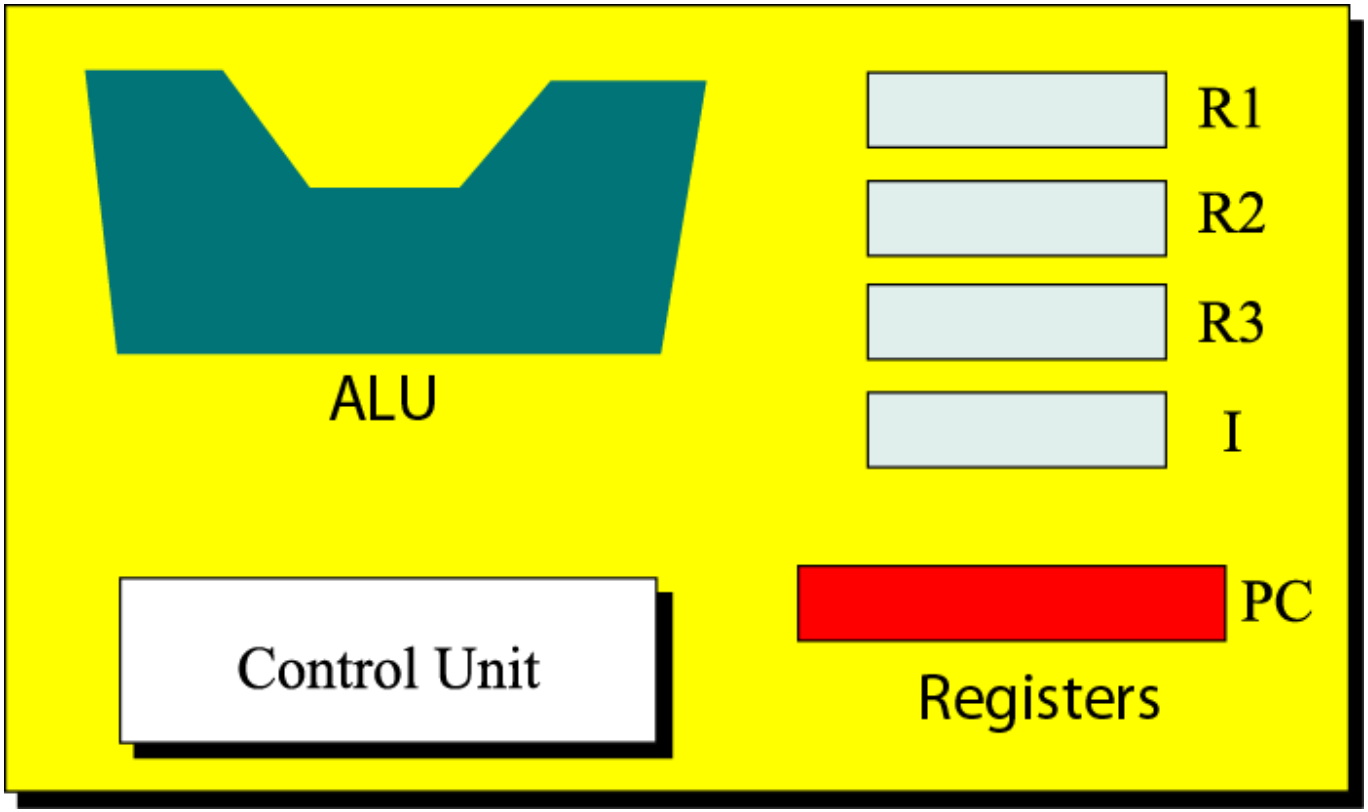
ปรับปรุงจากเอกสารของ
อ. ณัฐวุฒิ สร้อยดอกสน (NSD)

อ.เอิญ สุริยะฉาย (ENS)
ภาควิชาวิทยาการคอมพิวเตอร์และสารสนเทศ
มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าพระนครเหนือ

โครงสร้างพื้นฐานของไมโครโปรเซสเซอร์

- ALU ถือเป็นหัวใจหลักของไมโครโปรเซสเซอร์ โดยการคำนวณทั้งหมดจะเกิดขึ้นภายในหน่วยนี้
- รีจิสเตอร์เป็นอุปกรณ์จัดเก็บข้อมูลชั่วคราว
  - ขนาดของรีจิสเตอร์แต่ละรุ่นจะแตกต่างกันไปเช่น CPU ที่ประมวลผลขนาด 8 บิต ก็มักจะมีรีจิสเตอร์ขนาด 8 บิต
  - CPU ที่ประมวลผลขนาด 16 บิต ก็มักจะมีรีจิสเตอร์ขนาด 16
  - การติดต่อกับ CPU มักจะเป็นการติดต่อกับรีจิสเตอร์เสมอ
  - รีจิสเตอร์ใน CPU สามารถแบ่งออกเป็น
    - R1, R2, R3 รีจิสเตอร์สำหรับใช้งานทั่วไป
    - I สำหรับเก็บคำสั่ง
    - PC สำหรับชี้ตำแหน่งหน่วยความจำ

โครงสร้างพื้นฐานของไมโครโปรเซสเซอร์



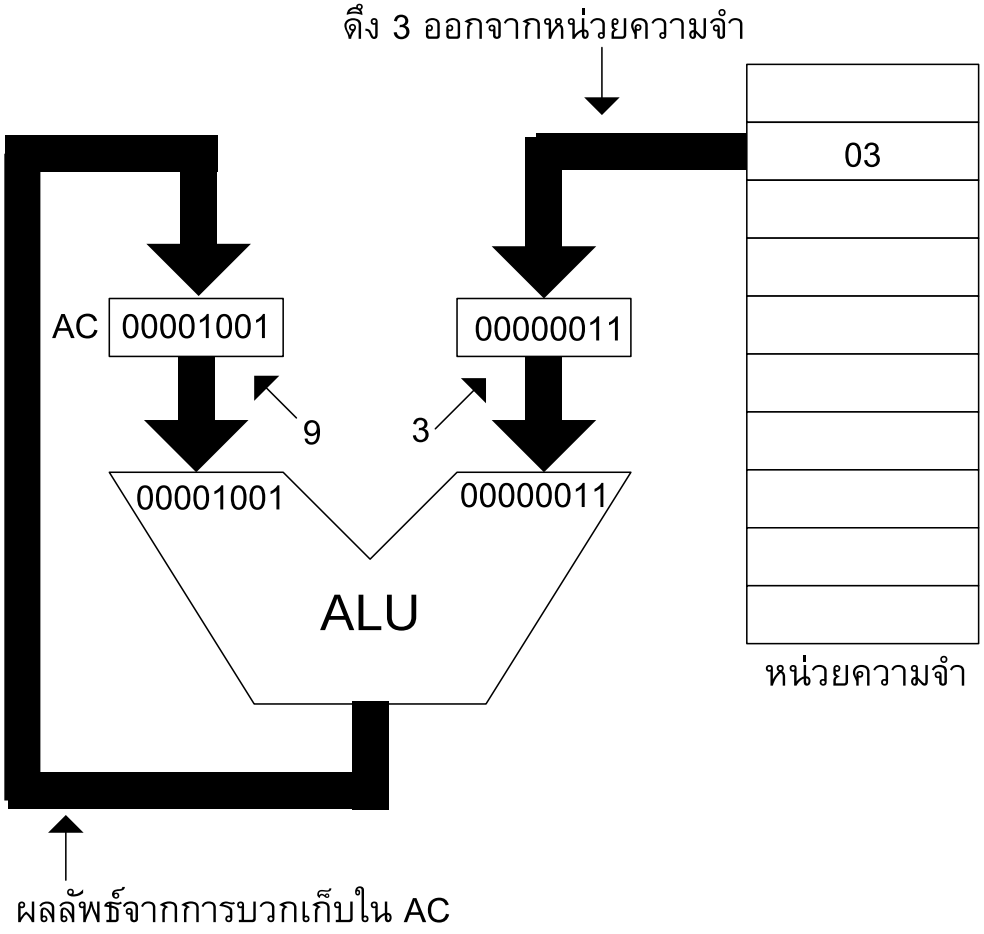
โครงสร้างพื้นฐานของไมโครโปรเซสเซอร์

- โดยทั่วไปแล้ว CPU หลายๆ รุ่นมักจะมีรีจิสเตอร์ที่ใช้ติดต่อกับ ALU ที่ชื่อว่าแอกคูมิวเลเตอร์ (Accumulator)
- เมื่อ CPU ต้องการประมวลผลทางคณิตศาสตร์ เช่นการบวกข้อมูล 2 ค่าเข้าด้วยกันก็มักจะใช้แอกคูมิวเลเตอร์เก็บข้อมูลตัวหนึ่งเช่น
- ถ้าต้องการบวกค่า 9 และ 3 CPU จะนำค่า 9 มาเก็บใน AC แล้วดึง 3 ออกจากหน่วยความจำ เพื่อส่งให้ ALU ทำการบวก โดยผลลัพธ์เก็บใน AC หรือหน่วยความต่อไป

โครงสร้างพื้นฐานของไมโครโปรเซสเซอร์

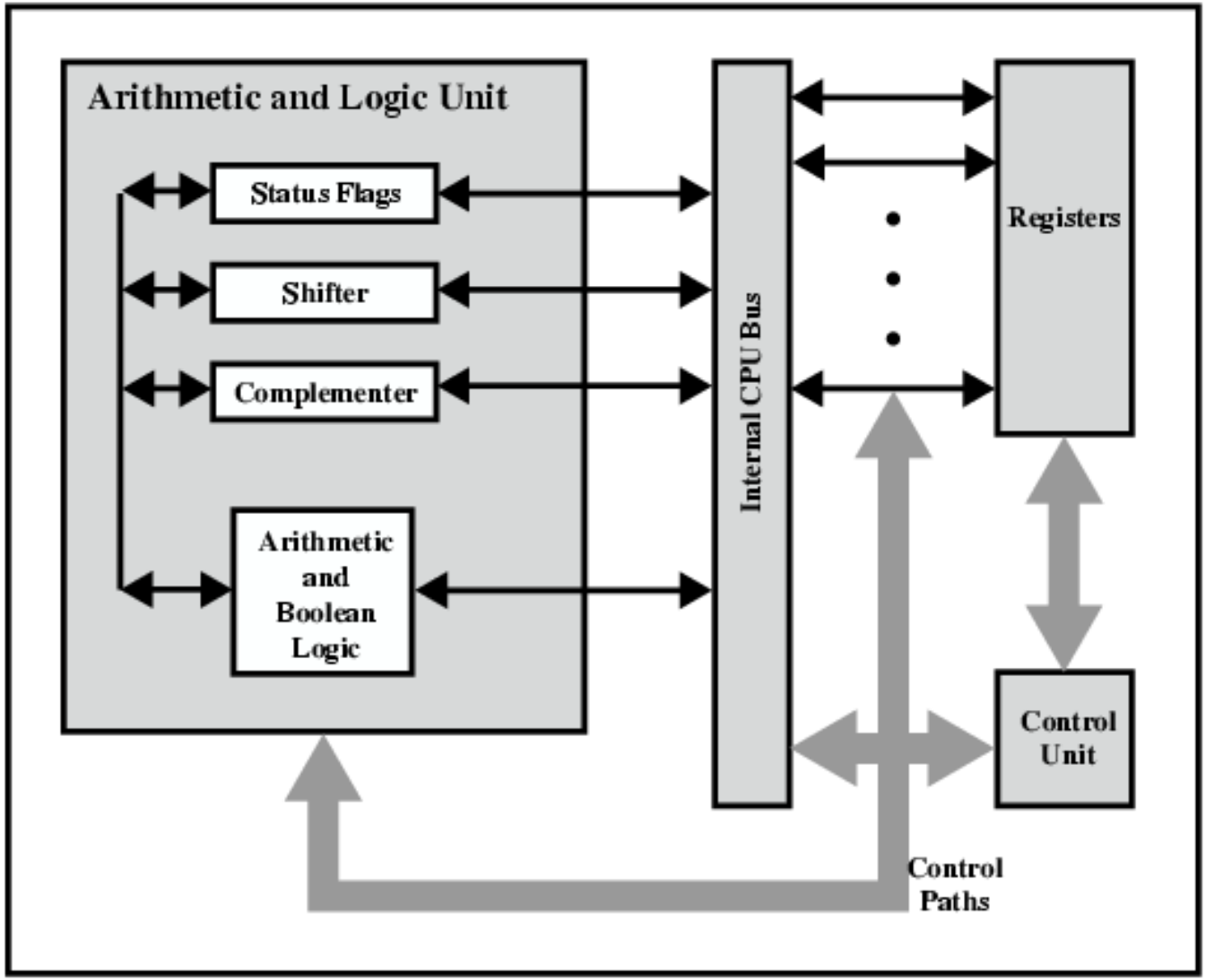
- ถ้าต้องการประมวลผลที่ซับซ้อนขึ้นหรือการคำนวณหลายขั้นตอน ก็ควรพัฒนาให้ไมโครโปรเซสเซอร์มีรีจิสเตอร์หลายๆ ตัวและมีบัส (Bus) ที่มีขนาดเท่ากับรีจิสเตอร์
- หน่วยควบคุม (Control Unit) ทำหน้าที่ควบคุมการทำงานของ CPU โดยภายในหน่วยควบคุมจะมี ROM ที่บรรจุ Microprogram ทำหน้าที่ควบคุมการทำงานย่อยๆ เช่น ถ้าต้องการบวก 3 และ 9 Microprogram จะทำให้เกิดกระบวนการเช่น
  - นำเลข 9 ไปใส่ใน AC
  - อ้างตำแหน่งหน่วยความจำที่เก็บข้อมูลตัวที่ 2 (3)
  - นำ 3 จากหน่วยความจำมาส่งให้ ALU
  - สั่งให้ ALU ทำงานเป็นวงจรบวกเลข
  - อ้างตำแหน่งหน่วยความจำในการเก็บผลลัพธ์
  - นำข้อมูลผลลัพธ์ไปเก็บยังหน่วยความจำที่อ้างถึง เป็นต้น

โครงสร้างพื้นฐานของไมโครโปรเซสเซอร์



โครงสร้างภายในของไมโครโปรเซสเซอร์

- โครงสร้างภายในของไมโครโปรเซสเซอร์จะมีความคล้ายคลึงกับโครงสร้างภายในของคอมพิวเตอร์
- ภายในตัวซีพียูจะมีบัสภายใน (Internal CPU Bus) สำหรับการถ่ายโอนข้อมูลระหว่างรีจิสเตอร์ต่างๆ
- โดย ALU จะทำงานกับข้อมูลที่อยู่ภายในตัวซีพียูเท่านั้น


Arithmetic and Logic Unit
Status Flags
Shifter
Complementer
Arithmetic and Boolean Logic
Internal CPU Bus
Registers
Control Unit
Control Paths

การจัดองค์ประกอบของรีจิสเตอร์

- ภายในตัวซีพียูจะประกอบด้วยรีจิสเตอร์ที่ทำหน้าที่เป็นหน่วยความจำเป็นจำนวนมาก ซึ่งรีจิสเตอร์ที่อยู่ในซีพียูจะทำหน้าที่ 2 ส่วนคือ
  1. **รีจิสเตอร์ที่ผู้ใช้มองเห็น** (User-visible Registers) ซึ่งรีจิสเตอร์ในส่วนนี้จะอนุญาตให้คำสั่งในโปรแกรมสามารถลดการอ้างถึงข้อมูลในหน่วยความจำหลัก โดยนำรีจิสเตอร์มาใช้งานแทน
  2. **รีจิสเตอร์สำหรับการควบคุมและรายงานสถานะการทำงาน** (Control and status Register) รีจิสเตอร์ในส่วนนี้จะถูกนำไปใช้โดยหน่วยควบคุม (Control Unit) เพื่อควบคุมการทำงานของซีพียูและถูกใช้โดยคำสั่งพิเศษของระบบปฏิบัติการในการควบคุมการประมวลผลของโปรแกรม

รีจิสเตอร์ที่ผู้ใช้มองเห็น

- รีจิสเตอร์ที่ผู้ใช้มองเห็น (User-visible Registers) หมายถึง รีจิสเตอร์ที่สามารถใช้คำสั่งภาษาเครื่องควบคุมและใช้งานผ่านการประมวลผลโดยซีพียู ซึ่งรีจิสเตอร์ในกลุ่มนี้จะแบ่งออกเป็นประเภทต่างๆ ดังนี้

1. รีจิสเตอร์เพื่อใช้งานทั่วไป
2. รีจิสเตอร์สำหรับการจัดเก็บข้อมูล
3. รีจิสเตอร์สำหรับจัดเก็บตำแหน่งที่อยู่
4. รีจิสเตอร์สำหรับเก็บเงื่อนไขการทำงาน

รีจิสเตอร์ที่ผู้ใช้มองเห็น

- **รีจิสเตอร์ที่ใช้งานทั่วไป** (General-purpose Register)
  - สามารถนำไปใช้งานได้หลายหน้าที่ตามที่ผู้พัฒนาโปรแกรมต้องการ โดยรีจิสเตอร์ในกลุ่มนี้สามารถบันทึกข้อมูลตัวถูกกระทำสำหรับรหัสดำเนินการนั้นๆ ได้
  - แต่ในรีจิสเตอร์ประเภทนี้บางส่วนถูกกำหนดมาให้ใช้งานกับเลขจำนวนจริง หรือเป็นตัวชี้ตำแหน่งข้อมูลในสแต็ก
  - ในบางครั้งรีจิสเตอร์สำหรับใช้งานทั่วไปสามารถนำมาใช้ในการกำหนดตำแหน่งที่อยู่ เช่นการอ้างถึงรีจิสเตอร์ทางอ้อม
- **รีจิสเตอร์สำหรับเก็บข้อมูล** (Data Register)
  - สามารถนำไปใช้เก็บข้อมูล ซึ่งไม่สามารถนำไปใช้ในคำสั่งการคำนวณหาค่าตำแหน่งที่อยู่

รีจิสเตอร์ที่ผู้ใช้มองเห็น

- **รีจิสเตอร์สำหรับเก็บตำแหน่งที่อยู่** (Address Register)
  - อาจเป็นรีจิสเตอร์สำหรับใช้งานทั่วไปหรือเป็นรีจิสเตอร์ที่กำหนดให้ใช้ในการอ้างอิงเกี่ยวกับตำแหน่งที่อยู่เท่านั้น
- **รีจิสเตอร์สำหรับเก็บเงื่อนไขการทำงาน** (Condition codes Register) หรือบางครั้งเรียกว่า “Flags”
  - โดยเงื่อนไขการทำงานหมายถึง บิตพิเศษกลุ่มหนึ่งที่จะถูกกำหนดค่าขึ้นมาให้สอดคล้องกับผลการประมวลผลของซีพียู เช่นการคำนวณทางคณิตศาสตร์ อาจทำให้เกิดผลลัพธ์ที่มีค่าเป็น จำนวนบวก จำนวนลบ ศูนย์หรือ Overflow
  - หรืออาจถูกกำหนดขึ้นมาเพื่อให้สอดคล้องกับเงื่อนไขที่ถูกสร้างขึ้นมาเช่น การ Branch หรือทดสอบค่าตามเงื่อนไขต่างๆ ที่ต้องการ

รีจิสเตอร์ที่ผู้ใช้มองเห็น

- ในบางครั้งการเรียกใช้งานโปรแกรมย่อย (Subroutine calls) จะทำให้เกิดการบันทึกค่าของรีจิสเตอร์ที่ผู้ใช้สามารถมองเห็นได้ทุกตัว และจะถูกนำค่าเดิมกลับมาใส่ไว้ในรีจิสเตอร์เมื่อการประมวลผลโปรแกรมย่อยสิ้นสุดลง

รีจิสเตอร์สำหรับการควบคุมและรายงานสถานะการทำงาน

- ในซีพียูมีรีจิสเตอร์เป็นจำนวนมากสำหรับการควบคุมและแสดงสถานะการทำงานของซีพียู รีจิสเตอร์กลุ่มนี้ในเครื่องคอมพิวเตอร์ส่วนใหญ่จะไม่ยินยอมให้ผู้ใช้งานมองเห็น
  - บางส่วนอาจยินยอมให้ใช้คำสั่งเครื่องในการประมวลผลได้ แต่ต้องกระทำในสถานะของระบบปฏิบัติการเท่านั้น
  - คอมพิวเตอร์แต่ละเครื่องจะมีการจัดองค์ประกอบของรีจิสเตอร์แตกต่างกันและนำไปใช้งานในลักษณะที่แตกต่างกัน แต่ส่วนใหญ่จะแบ่งออกเป็น 4 ประเภทที่มีความสำคัญต่อการประมวลผลของคำสั่งเครื่องดังนี้

รีจิสเตอร์สำหรับการควบคุมและรายงานสถานะการทำงาน

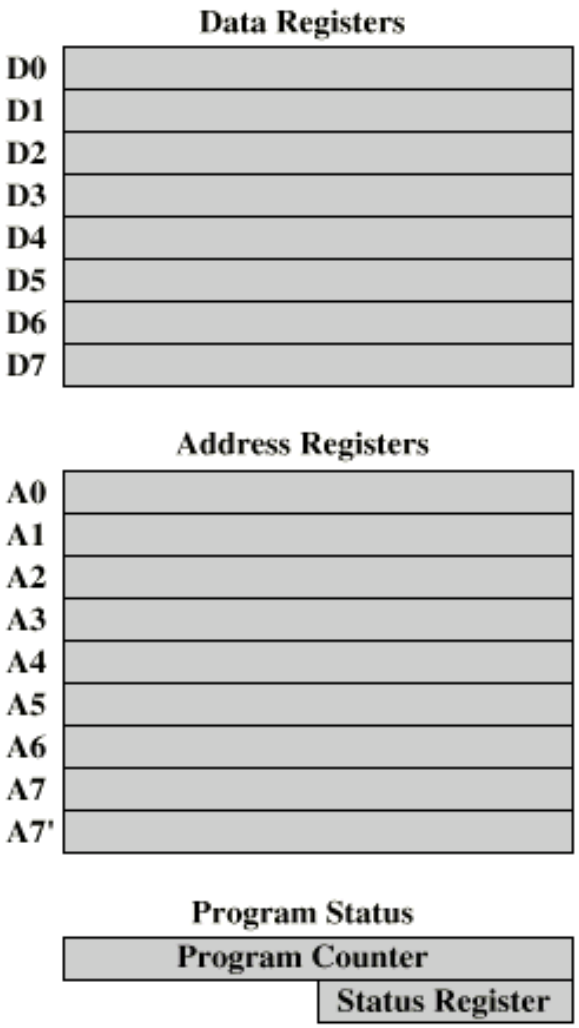
1. **Program Counter** (PC) เป็นรีจิสเตอร์ที่เก็บที่อยู่ของคำสั่งที่จะถูกประมวลผลยังลำดับถัดไป
2. **Instruction Register** (IR) เป็นรีจิสเตอร์ที่ใช้เก็บคำสั่งเครื่องที่พึ่งถูกอ่านเข้ามาในซีพียู
3. **Memory Address Register** (MAR) เป็นรีจิสเตอร์ใช้เก็บตำแหน่งที่อยู่ที่อ้างถึงในหน่วยความจำหลัก
4. **Memory Buffer Register** (MBR) เป็นรีจิสเตอร์ที่ใช้เก็บข้อมูลเพื่อเตรียมบันทึกลงในหน่วยความจำหลัก หรือใช้เก็บข้อมูลขนาด 1 Word ล่าสุดที่ถูกอ่านเข้ามาอยู่ในซีพียู

รีจิสเตอร์สำหรับการควบคุมและรายงานสถานะการทำงาน

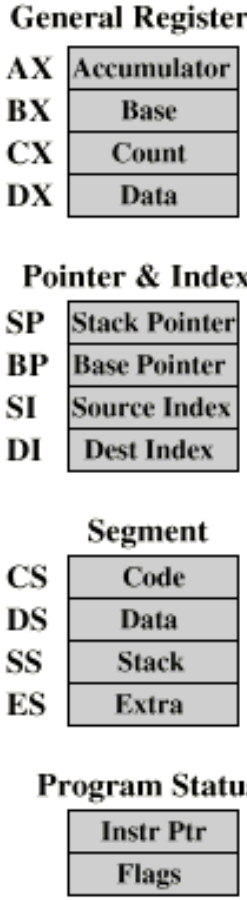
- โดยทั่วไป ซีพียูจะปรับปรุงค่าในรีจิสเตอร์ Program Counter ทันทีภายหลังที่ได้อ่านคำสั่งเข้ามา (Instruction fetch) เพื่อให้รีจิสเตอร์ PC นี้ชี้ตำแหน่งคำสั่งที่จะถูกอ่านเข้ามาในลำดับถัดไปต่อไปเสมอ โดยคำสั่งประเภท branch หรือ skip จะมีผลให้เกิดการเปลี่ยนแปลงค่าในรีจิสเตอร์นี้
- คำสั่งที่ถูกอ่านเข้ามาจะถูกนำไปจัดเก็บไว้ที่รีจิสเตอร์ IR ซึ่งจะถูกนำไปวิเคราะห์รหัสดำเนินงาน (Opcode) และกำหนดตัวค่าถูกกระทำ (Operand) ซึ่งอาจมีการแลกเปลี่ยนข้อมูลกับหน่วยความจำหลักผ่านรีจิสเตอร์ MAR และ MBR
- ในระบบคอมพิวเตอร์ที่ใช้บัส รีจิสเตอร์ MAR จะเชื่อมต่อโดยตรงกับ Address Bus และรีจิสเตอร์ MBR จะเชื่อมต่อโดยตรงกับ Data Bus

Program Status Word (PSW)

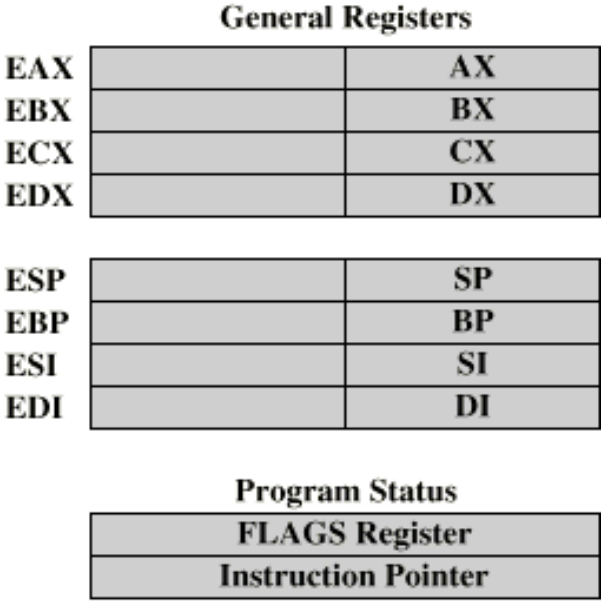
- นอกจากนี้ CPU ทุกตัวจะต้องมีรีจิสเตอร์ที่ชื่อว่า Program Status Word (PSW) ที่เก็บสถานะของข้อมูลหรือตัวแปร ณ ขณะที่ซีพียูกำลังทำงานดังต่อไปนี้
  - Sign : เก็บ sign bit ของผลลัพธ์ในการประมวลผลครั้งล่าสุด
  - Zero : จะมีค่าเป็น “1” เมื่อผลลัพธ์ของการประมวลผลเป็น 0
  - Carry : จะมีค่าเป็น “1” เมื่อ operation เป็นการบวกแล้วได้ผลลัพธ์ที่ต้องทดไปบิตข้างหน้า หรือเมื่อ operation เป็นการลบแล้วต้องมีการยืมจากบิตข้างหน้ามา
  - Equal : จะมีค่าเป็น “1” เมื่อผลของการเปรียบเทียบระหว่างตัวแปร 2 ตัวเท่ากัน
  - Overflow : ใช้ในกรณีที่มีการทำคำนวณต่าง ๆ แล้วเกิด overflow ขึ้น
  - Interrupt Enable/Disable : ใช้สำหรับกำหนดว่ายอมหรือไม่ยอมให้ interrupt
  - Supervisor : ระบุว่าจะให้ CPU execute คำสั่งในส่วนของผู้ควบคุมระบบ (supervisor) หรือ ผู้ใช้ (user) หรือเป็นการกำหนดสิทธิในการใช้ CPU นั่นเอง



(A) Motorola MC68000



(B) Intel 8086



(C) Intel 80386 – Pentium II

ขนาด 32 บิต

ตัวอย่างการจัดองค์ประกอบของรีจิสเตอร์

- Motorola MC68000: แบ่งรีจิสเตอร์ขนาด 32 บิตออกเป็น 2 กลุ่ม
  - สำหรับเก็บข้อมูลจำนวน 8 ตัว และเก็บตำแหน่งที่อยู่จำนวน 9 ตัว
  - โครงสร้างของรีจิสเตอร์ยอมให้มีการอ้างอิงข้อมูลขนาด 8,16 หรือ 32 บิต ขึ้นอยู่กับคำสั่งดำเนินการ
  - รีจิสเตอร์สำหรับเก็บตำแหน่งที่อยู่ขนาด 32 บิต โดยรีจิสเตอร์หมายเลข 7 ทั้งสองตัวจะถูกนำมาเป็นตัวชี้ตำแหน่งในสแต็ก (ตัวหนึ่งสำหรับผู้ใช้และอีกตัวหนึ่งสำหรับระบบปฏิบัติการ) ขึ้นอยู่กับสถานะการประมวลผลในขณะนั้น (เป็น user หรือ supervisor)
  - และยังมีรีจิสเตอร์ Program counter ขนาด 32 บิตและรีจิสเตอร์แสดงสถานะขนาด 16 บิต

ตัวอย่างการจัดองค์ประกอบของรีจิสเตอร์

- Intel 8086: รีจิสเตอร์ทุกตัวถือเป็นรีจิสเตอร์พิเศษ (Special-purpose register) แม้ว่ารีจิสเตอร์บางตัวจะนำมาใช้เป็นรีจิสเตอร์สำหรับใช้งานทั่วไปก็ตาม ประกอบไปด้วย
  - รีจิสเตอร์เก็บข้อมูลขนาด 16 บิต 4 ตัวซึ่งอาจจะอ้างถึงเพียง 8 บิตหรือทั้ง 16 บิตพร้อมกัน
    - รีจิสเตอร์สำหรับเก็บข้อมูลสามารถนำมาใช้งานเป็นรีจิสเตอร์สำหรับใช้งานทั่วไปได้ในบางคำสั่ง แต่คำสั่งส่วนอื่นอาจอ้างถึงรีจิสเตอร์ได้โดยตรง เช่นคำสั่งสำหรับการคูณจะเรียกใช้รีจิสเตอร์ Accumulator โดยอัตโนมัติเสมอ
  - รีจิสเตอร์สำหรับเป็นตัวชี้ (Pointer register) ขนาด 16 บิตจำนวน 4 ตัว
  - รีจิสเตอร์เซ็กเมนต์ขนาด 16 บิต 4 ตัวซึ่งมี 3 ตัวถูกนำมาใช้งานเฉพาะอย่างเพื่อชี้ไปยังตำแหน่งของเซ็กเมนต์ของคำสั่งที่กำลังประมวลผลอยู่ (Code) มีประโยชน์สำหรับคำสั่งประเภท branch, เซ็กเมนต์ที่เก็บข้อมูล (Data), และเซ็กเมนต์ที่เก็บสแต็ก (Stack)
  - รีจิสเตอร์สำหรับชี้ตำแหน่งคำสั่งเครื่องและรีจิสเตอร์ขนาด 1 บิตใช้บอกสถานะ (Flags)

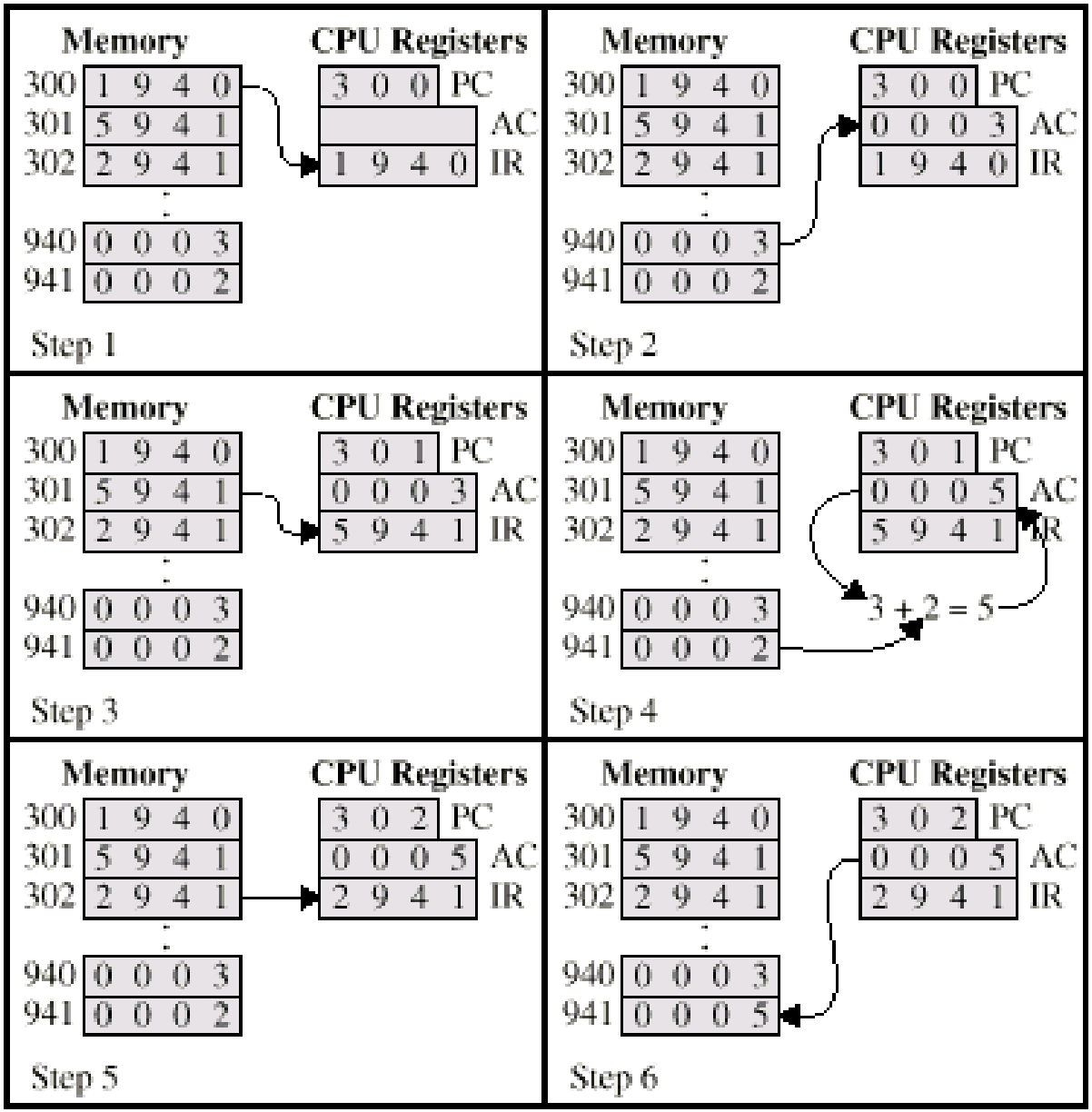
https://en.wikipedia.org/wiki/Intel_80386

Intel 80386 registers

$^{3}_{1}$ ... $^{1}_{5}$ ... $^{0}_{7}$ ... $^{0}_{0}$ *(bit position)*			
Main registers *(8/16/32 bits)*			
EAX	AX	AL	**A**ccumulator register
EBX	BX	BL	**B**ase register
ECX	CX	CL	**C**ount register
EDX	DX	DL	**D**ata register
Index registers *(16/32 bits)*			
ESI	SI		**S**ource **I**ndex
EDI	DI		**D**estination **I**ndex
EBP	BP		**B**ase **P**ointer
ESP	SP		**S**tack **P**ointer
Program counter *(16/32 bits)*			
EIP	IP		**I**nstruction **P**ointer
Segment selectors *(16 bits)*			
	CS		**C**ode **S**egment
	DS		**D**ata **S**egment
	ES		**E**xtra **S**egment
	FS		**F** **S**egment
	GS		**G** **S**egment
	SS		**S**tack **S**egment

Status register

$^{1}_{7}$	$^{1}_{6}$	$^{1}_{5}$	$^{1}_{4}$	$^{1}_{3}$	$^{1}_{2}$	$^{1}_{1}$	$^{1}_{0}$	$^{0}_{9}$	$^{0}_{8}$	$^{0}_{7}$	$^{0}_{6}$	$^{0}_{5}$	$^{0}_{4}$	$^{0}_{3}$	$^{0}_{2}$	$^{0}_{1}$	$^{0}_{0}$	*(bit position)*
V	R	0	N	IOPL		O	D	I	T	S	Z	0	A	0	P	1	C	EFlags


Memory
CPU Registers
300 1 9 4 0
301 5 9 4 1
302 2 9 4 1
940 0 0 0 3
941 0 0 0 2
3 0 0 PC
AC
1 9 4 0 IR
Step 1
0 0 0 3 AC
Step 2
3 0 1 PC
0 0 0 3 AC
5 9 4 1 IR
Step 3
0 0 0 5 AC
3 + 2 = 5
Step 4
3 0 2 PC
2 9 4 1 IR
Step 5
Step 6

ตัวอย่างการประมวลผลโปรแกรม

- ตัวอย่างของโปรแกรมแสดงให้เห็นถึงการนำข้อมูลจากหน่วยความจำตำแหน่ง 940 มาบวกเข้ากับข้อมูลในหน่วยความจำตำแหน่งที่ 941 และเก็บผลลัพธ์ไว้ที่ตำแหน่ง 941
- การทำงานนี้ใช้คำสั่งทั้งหมด 3 คำสั่ง ทำให้เกิดการดึงข้อมูลเข้าสู่โปรเซสเซอร์ 3 ครั้งและเกิดวงรอบการทำงานขึ้น 3 ครั้งดังนี้

1. Program Counter เก็บแอดเดรส 300 ไว้ (เป็นตำแหน่งของคำสั่งแรกที่จะถูกทำงาน) คำสั่งนี้จะถูกดึงเข้ามาไว้ในรีจิสเตอร์ IR (มีค่าเป็น 1940 เลขฐานสิบหก) และ Program Counter จะปรับปรุงค่าเพิ่มขึ้นเป็น 301
2. ข้อมูล 4 บิตแรกในรีจิสเตอร์ IR (Opcode) สั่งให้ดึงข้อมูลมากจาก AC ข้อมูลอีก 12 บิตที่เหลือบอกให้ทราบถึงตำแหน่ง (คือ 940) ในหน่วยความจำที่ต้องดึงข้อมูลเข้ามาในโปรเซสเซอร์
3. คำสั่งต่อไป (5941 เลขฐานสิบหก) ถูกอ่านมาจากแอดเดรส 301 ในหน่วยความจำและ Program Counter จะปรับปรุงค่าเพิ่มขึ้นเป็น 302
4. ค่าเดิมของรีจิสเตอร์ AC และข้อมูลที่แอดเดรส 941 ถูกนำมาบวกเข้าด้วยกันและเก็บผลลัพธ์ไว้ใน AC
5. คำสั่งต่อไป (2941 เลขฐานสิบหก) ถูกอ่านมาจากแอดเดรส 302 ในหน่วยความจำและ Program Counter จะปรับปรุงค่าเพิ่มขึ้นเป็น 303
6. ข้อมูลในรีจิสเตอร์ AC จะถูกนำไปจัดเก็บไว้ที่แอดเดรส 941 ในหน่วยความจำ

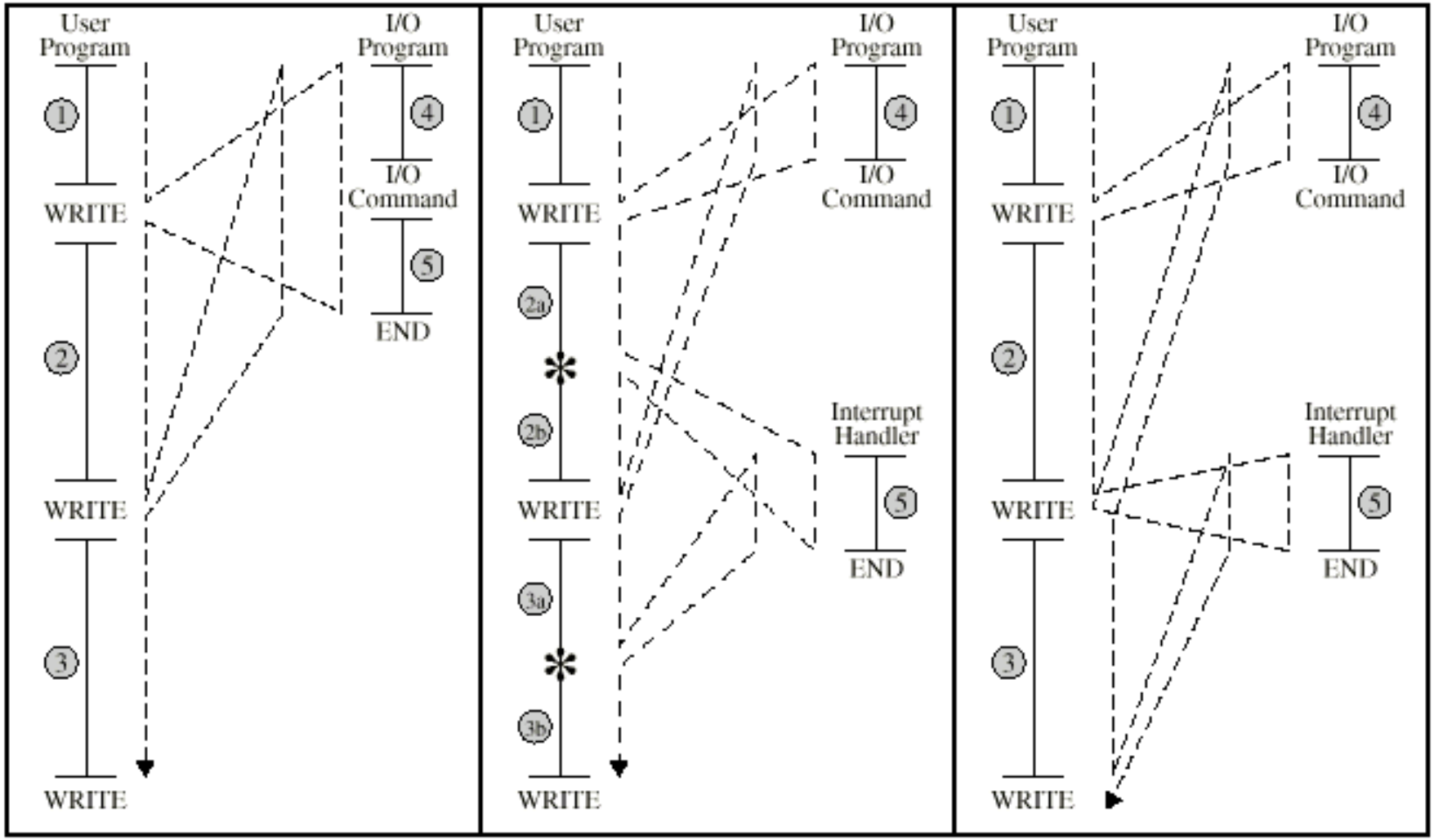
อินเทอร์รัพท์ (Interrupt)

- การที่ CPU มีการทำงานโดยติดต่อกับอุปกรณ์อื่น ๆ เช่น หน่วยความจำ อุปกรณ์ I/O ต่าง ๆ จึงสามารถจะถูกขัดจังหวะการทำงานของตัวเองได้ (interrupt) โดยแบ่งออกเป็น 4 ประเภท
  1. การ interrupt ที่เกิดจากความผิดพลาดของโปรแกรม เช่น การเกิด overflow จากการบวก การหารที่ตัวหารเป็น 0 หรือการพยายามที่จะ execute คำสั่งที่ไม่ถูกต้อง เป็นต้น
  2. การ interrupt ที่เกิดจากการตั้งเวลาของโปรเซสเซอร์เอง ในบางครั้งโปรเซสเซอร์เอง จะต้องมีการทำงานเพื่อตรวจสอบหรือทำงานบางอย่างที่จำเป็น จึงต้องมีการขัดจังหวะเพื่อทำงานเหล่านี้
  3. การ interrupt ที่เกิดจากตัวควบคุมอุปกรณ์ต่าง ๆ ซึ่งสัญญาณของการขัดจังหวะนั้นอาจเป็นการบอกว่าการทำงานเสร็จสิ้นอย่างสมบูรณ์หรือมีความผิดพลาดของการทำงานเกิดขึ้น
  4. การ interrupt ที่เกิดจากความผิดพลาดของฮาร์ดแวร์

อินเทอร์รัพท์ (Interrupt)

- การ interrupt ในบางกรณีอาจเป็นความต้องการที่จะเพิ่มประสิทธิภาพของการทำงานก็ได้ เช่น การที่อุปกรณ์ภายนอกอื่น ๆ ที่มีการทำงานช้ากว่าโปรเซสเซอร์ เช่น เครื่องพิมพ์
- เมื่อโปรเซสเซอร์โอนย้ายข้อมูลไปยังเครื่องพิมพ์โดยใช้ Instruction Cycle ตามปกติ หลังจากคำสั่ง write แต่ละครั้งโปรเซสเซอร์จะต้องหยุดชั่วขณะเพื่อรอให้เครื่องพิมพ์รับข้อมูลและพร้อมที่จะรับข้อมูลถัดไป ซึ่งในขณะที่โปรเซสเซอร์หยุดรอนี้อาจทำคำสั่งต่าง ๆ ได้เป็นร้อยหรือพันคำสั่ง
- ดังนั้นเพื่อให้โปรเซสเซอร์นั้นได้มีการทำงานอย่างคุ้มค่า จึงใช้กระบวนการ interrupt นี้มาช่วย โดยให้มีการนำเวลาที่ โปรเซสเซอร์ต้องหยุดรอเครื่องพิมพ์ หรืออุปกรณ์ I/O อื่น ๆ นี้มาทำคำสั่งอื่นๆ ทำให้การทำงานของระบบโดยรวมเร็วขึ้นด้วย

รูปแสดงการทำงานของอินเทอร์รัพท์



(a) No interrupts (b) Interrupts; short I/O wait (c) Interrupts; long I/O wait

อินเทอร์รัพท์ (Interrupt)

- รูป (a) เมื่อผู้ใช้เรียกฟังก์ชัน WRITE ระหว่างที่ทำการประมวลผลโปรแกรม (ฟังก์ชัน WRITE ทำหน้าที่เรียกใช้โปรแกรมสำหรับงานไอโอให้ทำหน้าที่บันทึกข้อมูล โดยโค้ดส่วนที่ 1,2 และ 3 คือลำดับชุดของคำสั่งที่ไม่เกี่ยวข้องกับงานไอโอ โปรแกรมทางไอโอประกอบไปด้วย 3 ส่วนคือ

1. ลำดับคำสั่ง หมายเลข 4 เป็นการเตรียมสำหรับงานไอโอที่จะเกิดขึ้น เช่น การส่งข้อมูลไปเก็บไว้ในบัฟเฟอร์
2. คำสั่งไอโอ (I/O Command) เมื่อโปรแกรมสำหรับไอโอทำงาน โปรแกรมที่เรียกใช้และซีพียูจะต้องหยุดรอจนกว่าโปรแกรมสำหรับไอโอจะทำงานเสร็จสิ้น
3. ลำดับคำสั่ง หมายเลข 5 เมื่อการทำงานของไอโอเสร็จสิ้น อาจมีการกำหนดค่า Flag ที่เกี่ยวข้องเพื่อแสดงผลการทำงานที่เกิดขึ้น เช่นสำเร็จหรือล้มเหลว

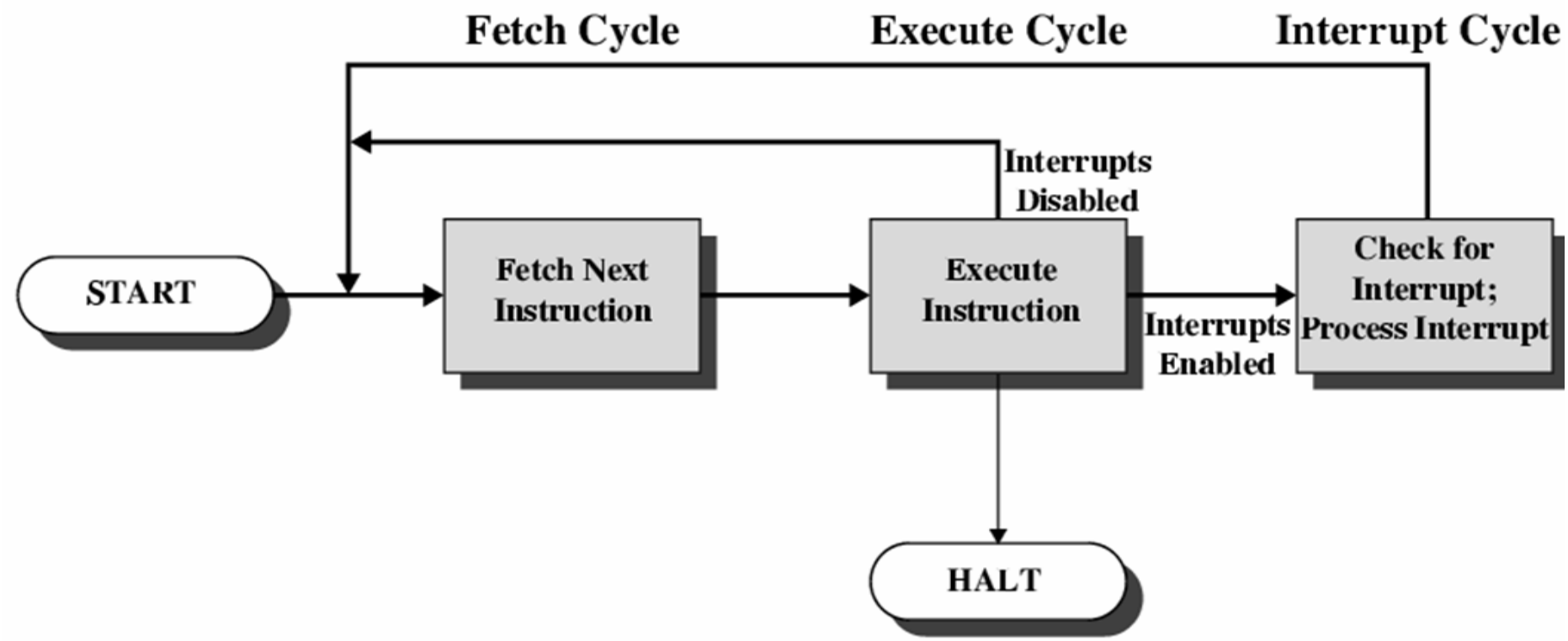
อินเทอร์รัพท์ (Interrupt)

- รูป (b) เมื่ออุปกรณ์ภายนอกพร้อมที่จะให้บริการ อุปกรณ์ไอโอนั้นจะส่งสัญญาณอินเทอร์รัพท์เพื่อเสนอความต้องการ (Interrupt request) มายังซีพียู ซีพียูจะต้องสนองด้วยการหยุดการประมวลผลโปรแกรมที่กำลังประมวลผลอยู่เป็นการชั่วคราว (Suspend) และนำโปรแกรมสำหรับอุปกรณ์ไอโอนั้นๆ (Interrupt handler) เข้ามาประมวลผลแทน หลังจากที่อุปกรณ์ไอโอได้รับการตอบสนองเสร็จสิ้นก็จะนำโปรแกรมเดิมกลับมาประมวลผล (resume) ต่อไป

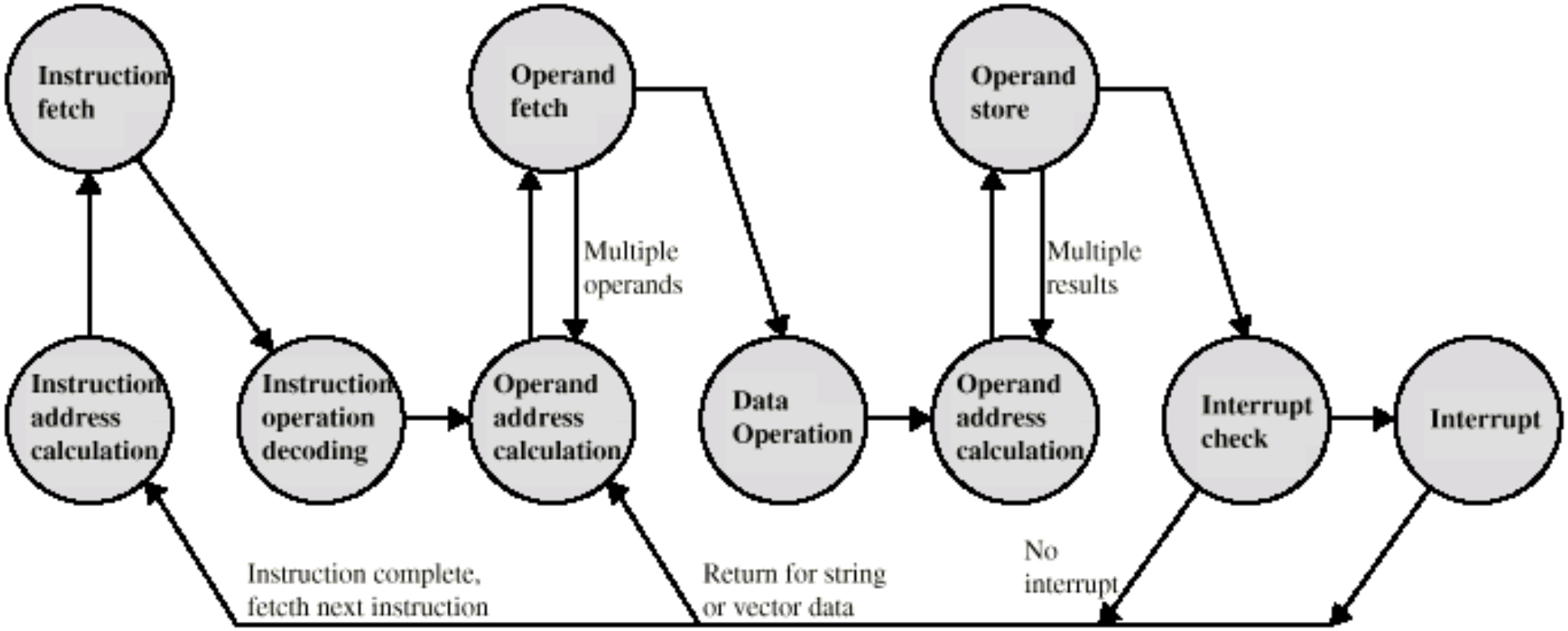
วงรอบคำสั่งของ CPU พร้อมการอินเทอร์รัพท์

- การที่ CPU ทำการประมวลผลคำสั่งต่างๆ และมีการเกิดอินเทอร์รัพท์ขึ้น สามารถแบ่งขั้นตอนการทำงานออกเป็นวงรอบคำสั่งได้ดังนี้
  - **Fetch** เป็นระยะที่ CPU ทำหน้าที่ดึง Instruction (Fetch Instruction) จากหน่วยความจำหลัก
  - **Execute** เป็นระยะที่ CPU ทำหน้าที่ตีความ Instruction (Execute Instruction) เพื่อปฏิบัติงานตาม Instruction จากนั้นจึงเตรียมทำ Instruction ต่อไป
  - **Interrupt** ถ้ายินยอมให้เกิดการ interrupt และเกิดการ interrupt ขึ้น ซีพียู จะต้องบันทึกสถานะการทำงานของโปรแกรมปัจจุบัน และให้บริการการ interrupt ที่ถูกเรียกใช้นั้น

วงรอบคำสั่งของ CPU พร้อมการอินเทอร์รัพท์



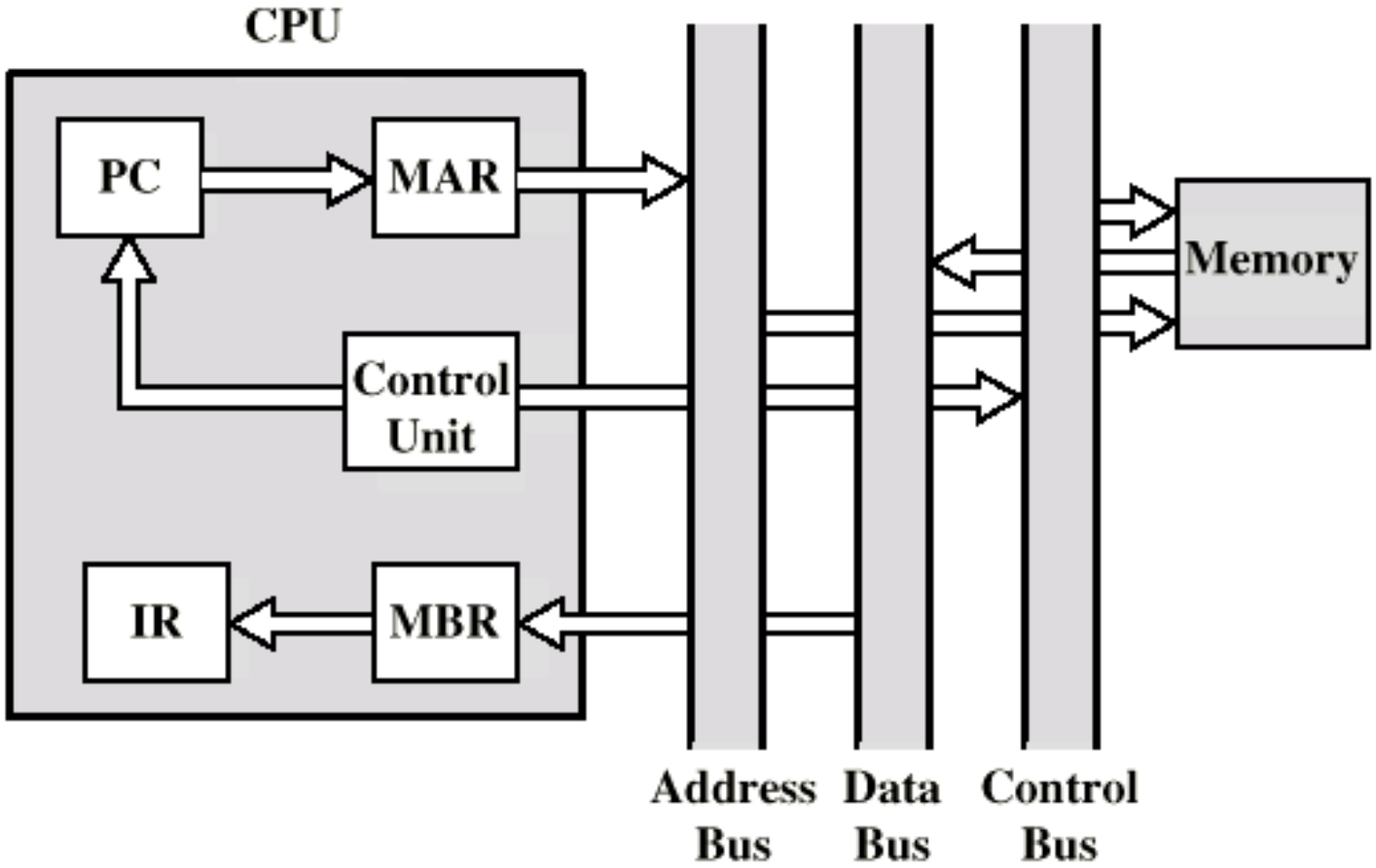
วงรอบคำสั่งของ CPU พร้อมการอินเทอร์รัพท์



การไหลของข้อมูล (Data Flow)

- **วงรอบคำสั่ง Fetch** (Fetch Cycle) คำสั่งจะถูกอ่านจากหน่วยความจำ โดย PC จะเก็บตำแหน่งของคำสั่งเครื่องที่ถูกดึงเข้ามาใน CPU ในลำดับถัดไป
  - โดยตำแหน่งคำสั่งจะถูกย้ายไปที่รีจิสเตอร์ MAR (Memory Address Register) และส่งเข้าไปที่บัสแอดเดรส (Address Bus)
  - หลังจากนั้นหน่วยควบคุม (Control Unit) จะร้องขอการอ่านข้อมูลกับหน่วยความจำ และจะนำข้อมูลที่ต้องการส่งไปบนบัสข้อมูล (Data Bus) และข้อมูลในบัสข้อมูลจะถูกส่งเข้ามาในรีจิสเตอร์ MBR (Memory Buffer Register) จากนั้นจะถูกย้ายไปที่รีจิสเตอร์ IR
  - ซึ่งเป็นขณะเดียวกับที่ค่าใน PC จะถูกเพิ่มค่าขึ้น 1 เพื่อเตรียมแอดเดรสถัดไปที่จะ ทำการอ่าน (Fetch)

Data Flow (Fetch Diagram)

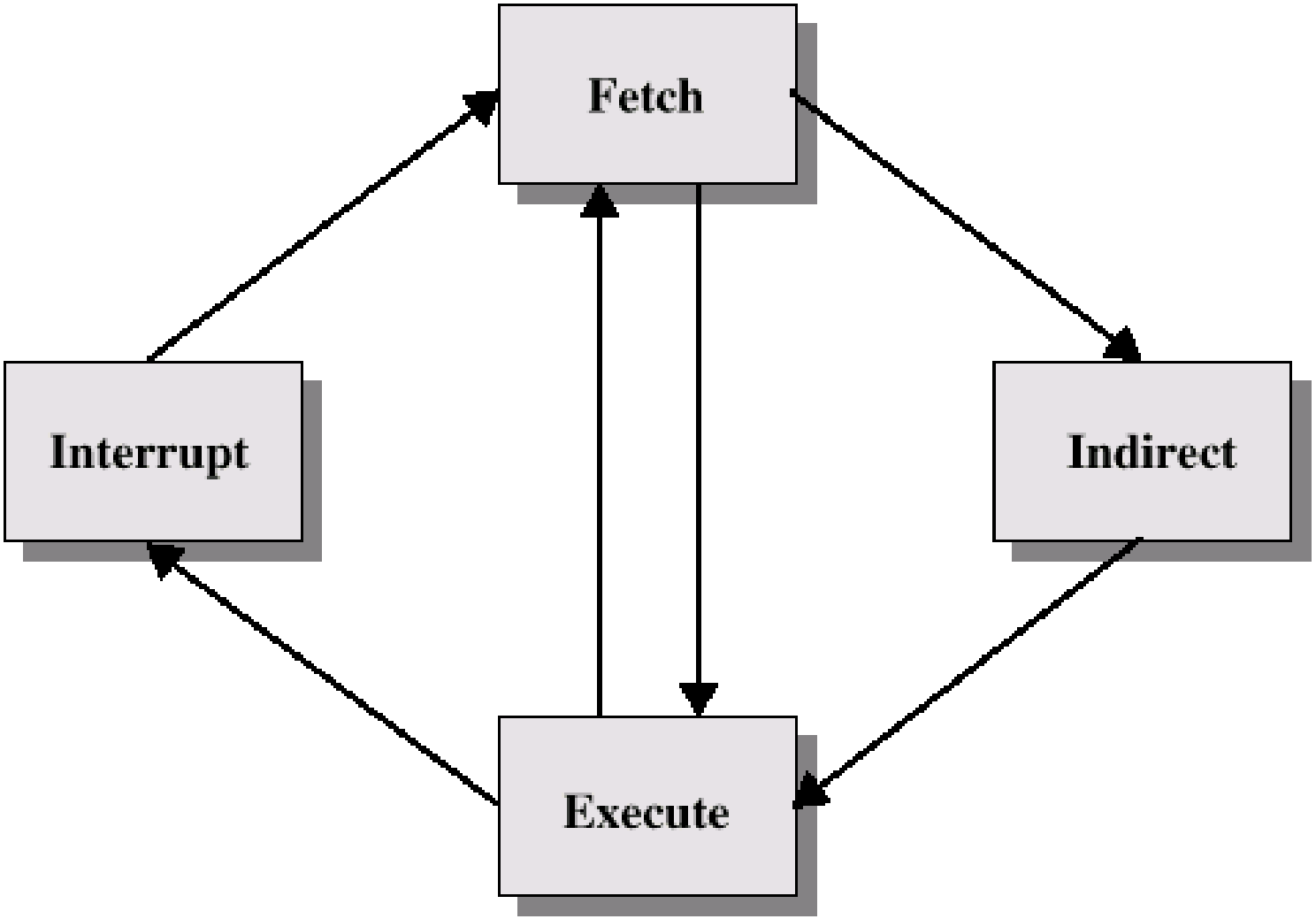


MBR = Memory buffer register
MAR = Memory address register
IR = Instruction register
PC = Program counter

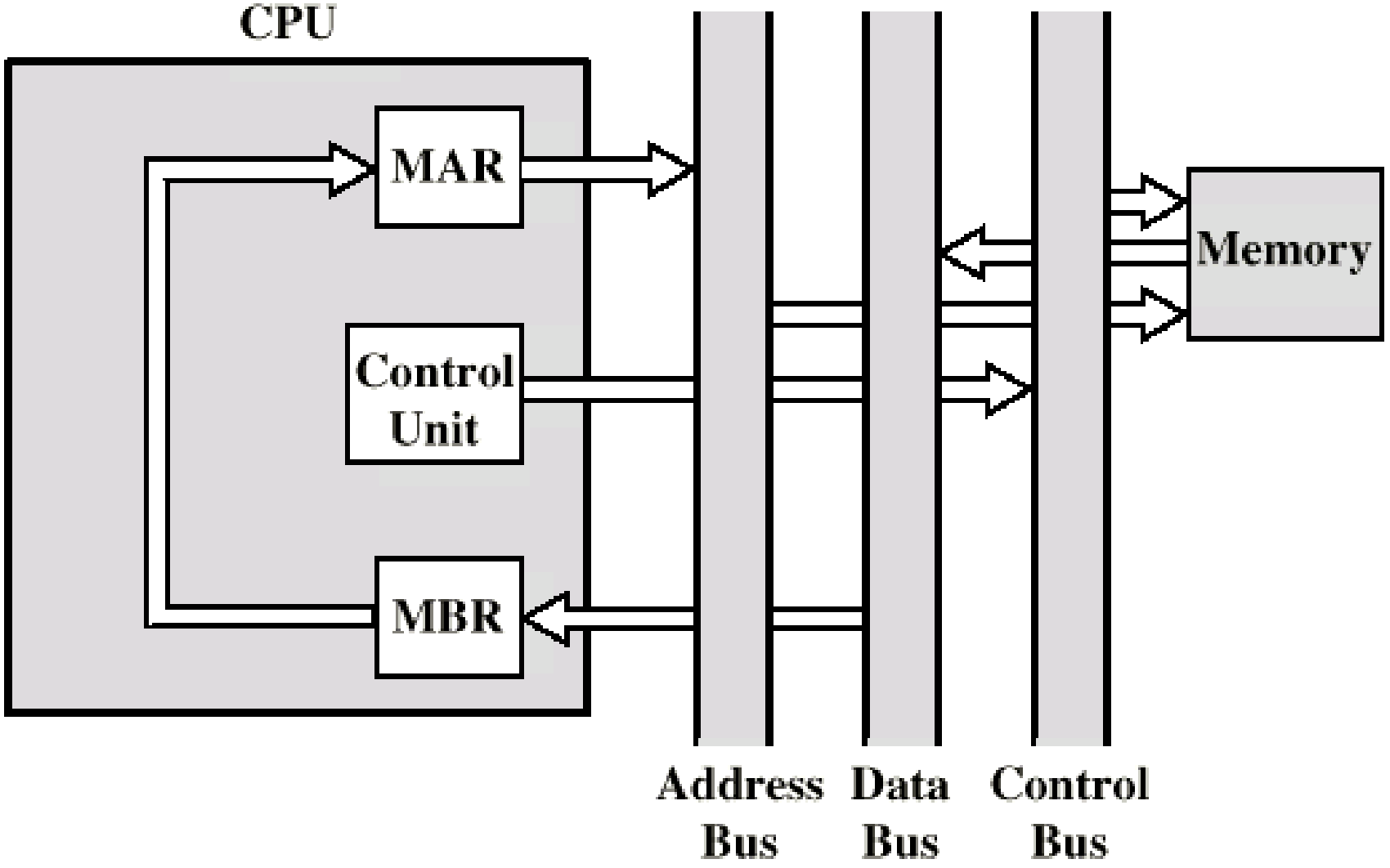
Data Flow (Indirect Addressing)

- เมื่อเสร็จสิ้นกระบวนการ Fetch หน่วยควบคุม (Control Unit) จะตรวจสอบข้อมูลหรือเนื้อความที่อยู่ในรีจิสเตอร์ IR ว่ามีการใช้ตัวแปรที่มีการอ้างถึงแบบใด (Addressing mode แบบใด)
- ถ้าหากว่ามีการอ้างถึงแบบ Indirect Addressing จะต้องมีการเรียก Indirect Cycle มาทำงาน โดยบิตขวาสุด N บิตของรีจิสเตอร์ MBR ซึ่งระบุแอดเดรสที่อ้างถึง จะถูกโอนย้ายไปที่รีจิสเตอร์ MAR จากนั้นหน่วยควบคุมก็จะร้องขอหน่วยความจำเพื่อขอข้อมูลและนำแอดเดรสของ operand นั้นมาไว้เก็บไว้ที่รีจิสเตอร์ MBR
- ภายหลังจากวงรอบการประมวลผลอาจมีอินเทอร์รัพท์เกิดขึ้น ซึ่งจะได้รับการตอบสนองในวงรอบอินเทอร์รัพท์ (Interrupt cycle) ก่อนที่จะกลับมาทำในขั้นตอนการ Fetch ต่อไป

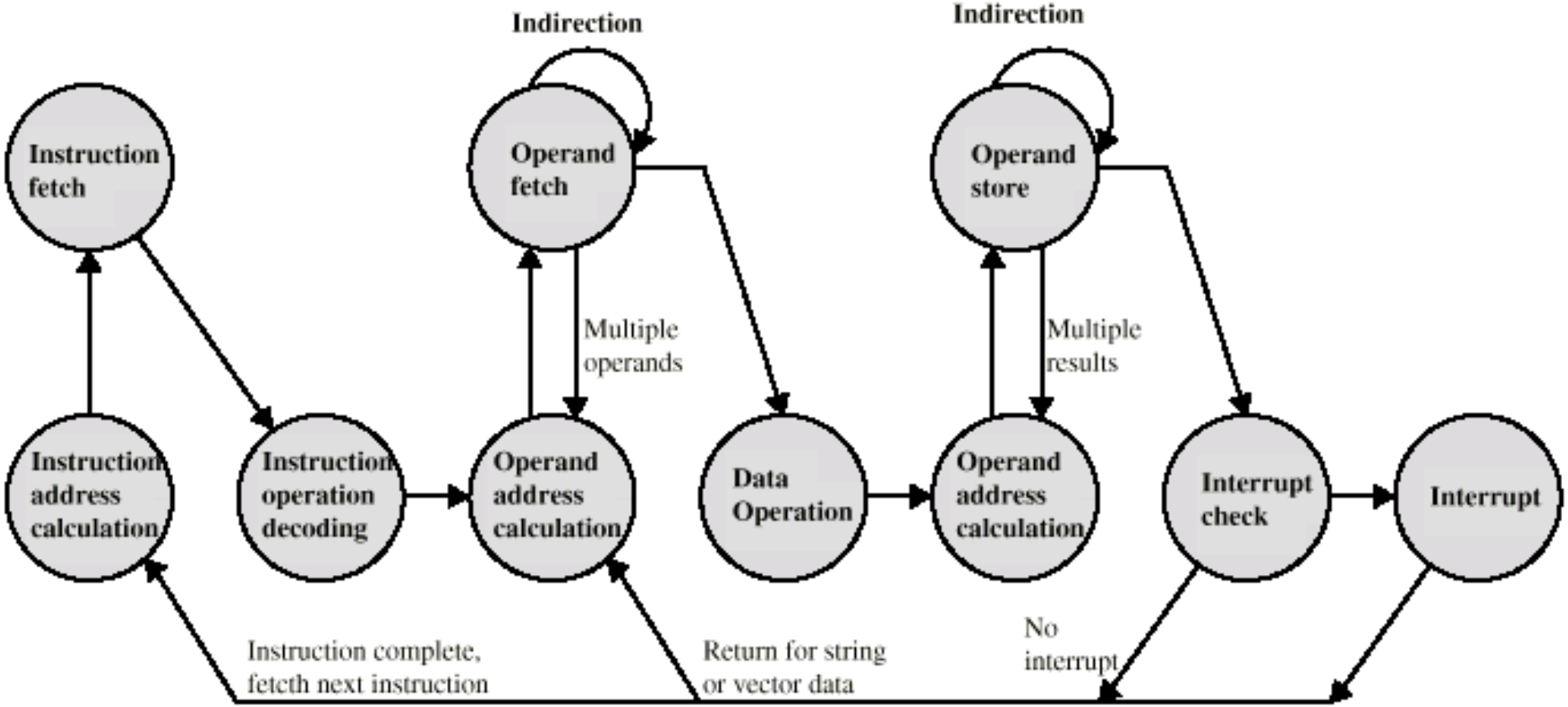
Instruction Cycle with Indirect



Data Flow (Indirect Diagram)



วงรอบคำสั่งของ CPU กับ Interrupt



Data Flow (Interrupt Cycle)

- กรณีวงรอบอินเทอร์รัพท์ (Interrupt Cycle) ค่าที่เก็บอยู่ในรีจิสเตอร์ PC จะต้องถูกบันทึกไว้ เพื่อที่ซีพียูสามารถย้อนกลับมาทำงานเดิมได้หลังจากที่ให้บริการอินเทอร์รัพท์เสร็จสิ้น ดังนั้นค่าในรีจิสเตอร์ PC จะถูกโอนย้ายไปเก็บไว้ที่รีจิสเตอร์ MBR เพื่อที่จะเขียนลงไปในหน่วยความจำหลัก
- โดยพื้นที่ในหน่วยความจำส่วนหนึ่งจะถูกสงวนไว้เป็นพิเศษสำหรับการทำงานนี้ (อาจเป็นตำแหน่งของตัวชี้ตำแหน่งในสแต็ก จากนั้นรีจิสเตอร์ PC จะถูกใส่ค่าด้วยตำแหน่งของโปรแกรมอินเทอร์รัพท์ ทำให้วงรอบการ Fetch ข้อมูลถัดไปจะเป็นการนำคำสั่งในโปรแกรมอินเทอร์รัพท์ขึ้นมาทำงาน

Data Flow (Interrupt Diagram)