ระบบคอมพิวเตอร์และสถาปัตยกรรม
(Computer System and Architecture)

บทที่ 4
ชุดคำสั่ง
(Instruction Sets)

ชุดคำสั่ง (Instruction Sets)

- โปรเซสเซอร์ใช้คำสั่งได้ตามคำสั่งที่ถูกบรรจุลงใน ALU
- คำสั่งหลาย ๆ คำสั่งที่บรรจุลงใน ALU นี้เรียกว่าชุดคำสั่งของเครื่อง (Machine Instructions Sets)
- ชุดคำสั่งนี้ทำให้ผู้ออกแบบระบบคอมพิวเตอร์และโปรแกรมเมอร์อยู่ในขอบเขต เดียวกัน
- ผู้ออกแบบระบบมองว่าชุดคำสั่งประกอบด้วยฟังก์ชันพื้นฐานสำหรับการทำงานของคอมพิวเตอร์
- อาจกล่าวได้ว่าการสร้างโปรเซสเซอร์ก็คือการสร้างชุดคำสั่งนั่นเอง
- โปรแกรมเมอร์จะต้องพิจารณาถึงโครงสร้างของรีจิสเตอร์และหน่วยความจำ ประเภทข้อมูลในเครื่อง และฟังก์ชันของ ALU เพื่อเขียนโปรแกรมสั่งให้ซีพียูทำงาน

คุณสมบัติพื้นฐานของคำสั่ง

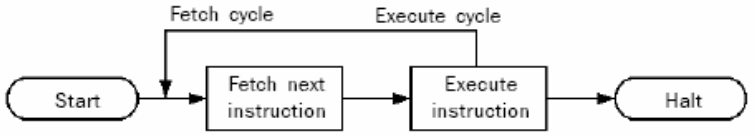
ชุดคำสั่ง (Instruction Sets) หมายถึงชุดของคำสั่งที่ Processor Executeเพื่อดำเนินการตามที่โปรแกรมเมอร์ต้องการ อาจเรียกชุดคำสั่งว่า “คำสั่งเครื่อง” (machine instructions) หรือ “คำสั่งคอมพิวเตอร์” (computer instructions) ก็ได้ ในแต่ละชุดคำสั่งอาจจะมีคำสั่งที่หลากหลายประกอบอยู่ เช่น คำสั่งสำหรับการบวก ซีพียูจะต้องมีคำสั่งในการโหลดข้อมูลจากรีจิสเตอร์ลงหน่วยความจำ แล้วเรียกใช้คำสั่งสำหรับการบวก หลังจากนั้นจะมีคำสั่งเพื่อเก็บค่าผลลัพธ์กลับรีจิสเตอร์อีกครั้ง

คุณสมบัติพื้นฐานของคำสั่ง

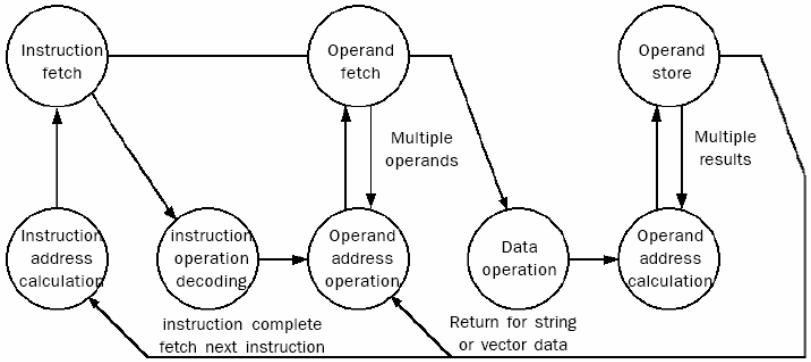
- ชุดคำสั่งของแต่ละโปรเซสเซอร์จะมีความแตกต่างกัน
- สิ่งที่แตกต่างกันอาจจะเป็นขนาดของคำสั่ง ประเภทของ opration ประเภทของ operand หรือประเภทของผลลัพธ์ก็ได้
- ชุดคำสั่งที่แตกต่างกันนี้อาจจะเกิดจากโครงสร้างภาษาชั้นสูงที่โปรแกรมเมอร์ใช้งาน เช่น ภาษา C Pascal หรือ Ada เป็นต้น
- โปรแกรมภาษาชั้นสูงเหล่านี้จะถูกคอมไพล์ (compile) ด้วยคอมไพเลอร์หรือตัวแปรภาษานั้นให้เป็นภาษาเครื่องเพื่อทำงานต่อไป
- ต้องมีการ compile ใหม่ให้ตรงกับโปรเซสเซอร์ที่ใช้งาน
- การคอมไพล์ใหม่เป็นการเปลี่ยนภาษาชั้นสูงให้เป็นภาษาเครื่องตามชุดคำสั่งของซีพียูนั่นเอง

วงรอบคำสั่ง

- **การทำงานของคอมพิวเตอร์คือการที่ Processor Execute คำสั่งในโปรแกรมตามลำดับเรื่อยไปตั้งแต่ต้นจนจบ**
- **รูปแบบคำสั่งที่ง่ายที่สุดจะมี 2 ขั้นตอนคือการที่โปรเซสเซอร์อ่านหรือเฟ็ตช์คำสั่ง (fetches) จากหน่วยความจำครั้งละ 1 คำสั่ง หลังจากนั้นจะ execute ตามคำสั่งนั้น**



วงรอบคำสั่ง



Fetch Diagram

วงรอบคำสั่ง

- Instruction Address Calculation (IAC)
- Instruction Fetch (IF)
- Instruction Operation Decoding (IOD)
- Operand Address Calculation (OAC)
- Operation Fetch (OF)
- Data Operation (DO)
- Operand Store (OS)

ส่วนประกอบคำสั่งเครื่อง

- **Operation code** : กำหนด Operation ที่จะกระทำ (เช่น ADD, I/O) operation ถูกกำหนดด้วยเลขฐานสองที่เรียกว่า operation code หรือ opcode
- **Source operand reference** : กำหนดส่วนอ้างอิงของฯ operand ที่ใส่เข้ามาสำหรับ opration
- **Result operand reference** : อ้างอิงถึงผลลัพธ์จาก Operation
- **Next instruction reference** : บอกซีพียูว่าจะไปอ่านคำสั่งต่อไปได้จากไหนหลังจาก executeคำสั่งนี้เสร็จสมบูรณ์แล้ว

ประเภทคำสั่ง

คอมพิวเตอร์จะต้องมีชุดคำสั่งที่ยอมให้ผู้ใช้งานทำงานกับข้อมูลได้ตามต้องการ

- **Data processing** : คำสั่งทางคณิตศาสตร์และตรรกะ
- **Data storage** : คำสั่งจัดการหน่วยความจำ
- **Data movement** : คำสั่งจัดการอินพุต/เอาต์พุต
- **Control** : คำสั่งตรวจสอบเงื่อนไขและกระโดดไปทำงาน

จำนวน Address

- โดยทั่วไปการกล่าวถึงสถาปัตยกรรมคอมพิวเตอร์จะกล่าวถึงจำนวน Address ปัจจุบันมีความสำคัญน้อยกว่าการออกแบบ CPU
- จำนวน Address ของระบบมีผลต่อวงรอบการทำงานของคำสั่งเครื่อง
- ยิ่งมีจำนวน Address มากก็จะยิ่งทำให้วงรอบการทำงานน้อยลง ทำให้ทำงานได้เร็วขึ้น

จำนวน Address

คำสั่ง	สิ่งที่กระทำ
LOAD A	ย้าย A ไปยังแอ็กคิวมูเลเตอร์ AC
MUTIPLY B	AC := AC x B
STORE T	ย้าย AC ไปหน่วยความจำตำแหน่ง T
LOAD C	ย้าย C ไปยังแอ็กคิวมูเลเตอร์ AC
MULTIPLY C	AC := AC x C
ADD T	AC := AC + T
STORE X	เก็บค่าผลลัพธ์ในหน่วยความจำตำแหน่ง X

One-address

คำสั่ง	สิ่งที่กระทำ
MOVE T, A	T := A
MULTIPLY T, B	T := T X B
MOVE X, C	X := C
MULTIPLY X, C	X := X x C
LOAD X, T	X := X + T

Two-address

จำนวน Address

คำสั่ง	สิ่งที่กระทำ
MULTIPLY T, A, B	T := A x B
MULTIPLY X, C, C	X := C x C
ADD X, X ,T	X := X + T

Three-address

คำสั่ง	สิ่งที่กระทำ
PUSH A	ย้าย A ไปอยู่บนสุดของสแต็ก
PUSH B	ย้าย B ไปอยู่บนสุดของสแต็ก
MULTIPLY	ลบ A, B ออกจากสแต็ก และแทนค่าด้วย A x B
PUSH C	ย้าย C ไปอยู่บนสุดของสแต็ก
PUSH C	ย้ายก็อปปี้ที่สองของ C ไปอยู่บนสุดของสแต็ก
MULTIPLY	ลบ C, C ออกจากสแต็ก และแทนค่าด้วย C x C
ADD	ลบ C x C, A x B ออกจากสแต็ก และแทนค่าด้วย ผลบวกของ (C x C) + (A x B)
POP X	ย้ายผลลัพธ์จากบนสุดของสแต็กไปเก็บไว้ที่ X

Zero-address

การออกแบบชุดคำสั่ง

พื้นฐานการออกแบบชุดคำสั่ง

- **Operation repertoire** : จำนวน Operation ที่มีให้เลือกใช้ รวมทั้งความซับซ้อนของ Operation ที่ควรเป็น
- **Data type** : ความหลากหลายของประเภทของข้อมูลที่ทำ Operation
- **Instruction format** : ความยาวของคำสั่ง (เป็นบิต) จำนวน address ขนาดของฟิลด์ และอื่น ๆ
- **Register** : จำนวนรีจิสเตอร์ที่คำสั่งสามารถอ้างอิงและใช้ประโยชน์ได้
- **Addressing** : การกำหนดโหมดของaddressสำหรับoperand

ประเภท Operation

พื้นฐานของ Operation สามารถแบ่งเป็นกลุ่ม ๆ ได้คือ

- Operation ทางด้านการถ่ายโอนข้อมูล (Data transfer)
- Operation ทางด้านคณิตศาสตร์ (Arithmetic)
- Operation ทางด้านตรรกะ (Logical)
- Operation ทางด้านการแปลงค่า (Conversion)
- Operation ทางด้านอุปกรณ์อินพุต/เอาต์พุต (I/O)
- Operationทางด้านการควบคุมระบบ (System control)

ประเภท Operand

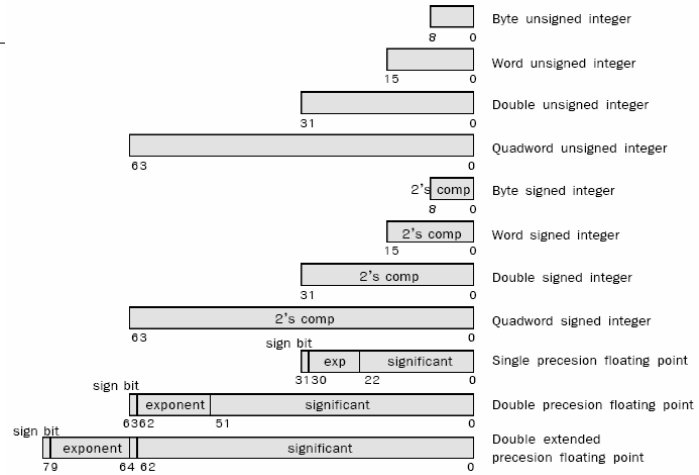
ประเภทของ Operand มีดังนี้

- address
- number
- character
- logical

ประเภทข้อมูลของ Pentium

- General
- Integer
- Ordinal
- Unpacked binary coded decimal (BCD)
- Packed BCD
- Near pointer
- Bit field
- Byte string
- Floating point

โครงสร้างข้อมูลตัวเลขของ Pentium



ประเภทข้อมูลของ PowerPC

- Unsigned byte
- Unsigned Halfword
- Signed Halfword
- Unsigned Word
- Signed Word
- Unsigned Doubleword
- Byte String

รูปแบบคำสั่ง

- ความยาวคำสั่ง
- การจัดวางบิต
  - จำนวนในการกำหนดโหมดของแอ็ดเดรส
  - จำนวนโอเปอแรนด์
  - รีจิสเตอร์กับหน่วยความจำ
  - จำนวนชุดของรีจิสเตอร์
  - ขอบเขตของแอ็ดเดรส
  - ความละเอียดในการกำหนดแอ็ดเดรส

การกำหนดAddressหน่วยความจำ

- Little Endian
- Big Endian

ข้อมูล คือ 89ABCDEF

Address	Data
1003	89
1002	AB
1001	CD
1000	EF

(ก)

Address	Data
1003	EF
1002	CD
1001	AB
1000	89

(ข)

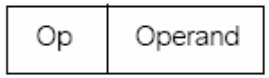
(ก) การวางไบต์แบบ Little Endian (ข) การวางไบต์แบบ Big Endian

การกำหนด Address Mode

- การกำหนด address แบบให้ค่าโดยตรง (Immediate Addressing)
- การกำหนด address โดยตรง (Direct Addressing)
- การกำหนด address ทางอ้อม (Indirect Addressing)
- การกำหนด address ผ่านรีจิสเตอร์โดยตรง (Register Direct Addressing)
- การกำหนด address ผ่านรีจิสเตอร์ทางอ้อม (Register Indirect Addressing)
- การกำหนด address แบบแทนที่ (Displacement หรือ Indexed Addressing)
- การกำหนด address แบบสัมพันธ์ (Relative Addressing)
- การใช้สแต็ก (Stack)

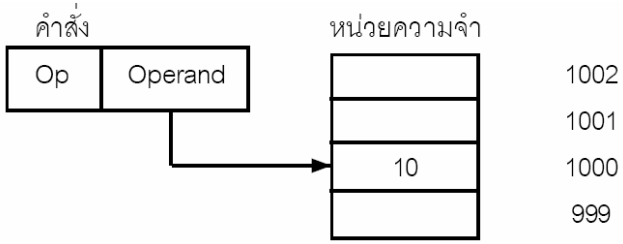
Immediate Addressing

LOAD X, #1000 เป็นการโหลดข้อมูล 1000 ไว้ที่ตัวแปร X



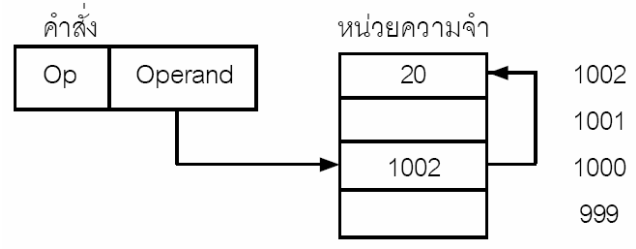
Direct Addressing

LOAD X, 1000 เป็นการโหลดข้อมูลที่ address 1000 ไว้ที่ตัวแปร X
(ถ้าที่ address 1000 มีค่า 10 ดังนั้น X จะมีค่าเท่ากับ 10)



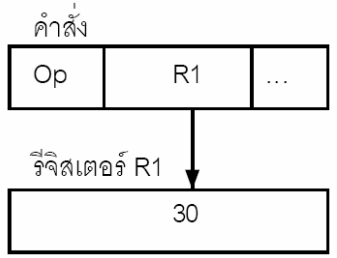
Indirect Addressing

LOAD X, (1000) เป็นการโหลดข้อมูลที่อยู่บน address ที่เก็บอยู่ใน address 1000 (ถ้าที่ address 1000 มีค่า 1002 ดังนั้นข้อมูลที่แท้จริงอยู่ที่ address 1002 ในที่นี้คือ 20 ดังนั้น X จะมีค่าเท่ากับ 20)



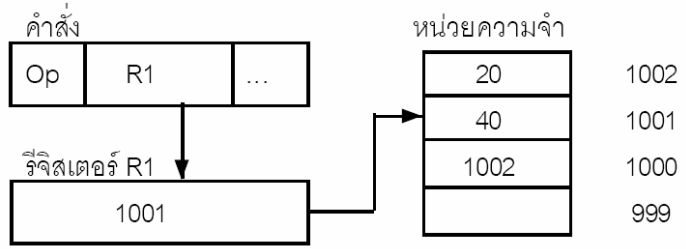
Register Direct Addressing

LOAD X, R1 เป็นการโหลดข้อมูลจากรีจิสเตอร์ R1 ไว้ที่ตัวแปร X (ถ้าที่รีจิสเตอร์ R1 มีค่า 30 ดังนั้น X จะมีค่าเท่ากับ 30)



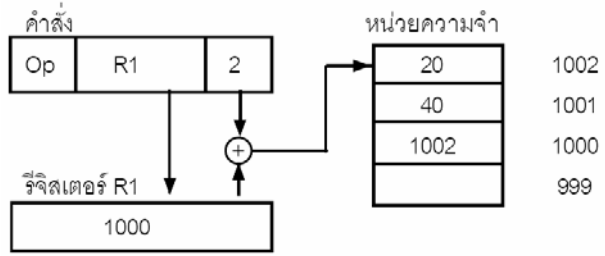
Register Indirect Addressing

LOAD X, (R1) เป็นการโหลดข้อมูลที่อยู่ใน Address ที่เก็บอยู่ในรีจิสเตอร์ R1 ไว้ที่ตัวแปร X (ถ้าที่รีจิสเตอร์ R1 เก็บค่า 1001 และที่ Address 1001 ของหน่วยความจำมีค่า 40 ดังนั้น X จะมีค่าเท่ากับ 40)



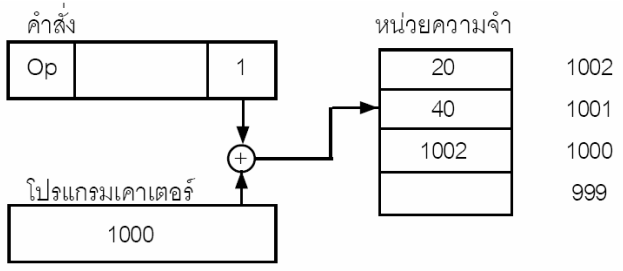
Displacement หรือ Indexed Addressing

LOAD X, (R1)+Constant เป็นการโหลดข้อมูลจากAddressบนหน่วยความจำที่เกิดจากค่าในรีจิสเตอร์บวกกับค่าคงที่ ไว้ที่ตัวแปร X (ถ้าในรีจิสเตอร์มีค่า 1000, ค่าคงที่เท่ากับ 2 และค่าที่ Address 1002 มีค่าเท่ากับ 20 ดังนั้นค่า X จะเท่ากับ 20)



Relative Addressing

LOAD X, PC+Constant เป็นการโหลดข้อมูลจากaddressบนหน่วยความจำที่เกิดจากค่าในโปรแกรมเคาเตอร์บวกกับค่าคงที่ ไว้ที่ตัวแปร X (ถ้าในโปรแกรมเคาเตอร์มีค่า 1000, ค่าคงที่เท่ากับ 1 และค่าที่ address 1001 มีค่าเท่ากับ 40 ดังนั้นค่า X จะเท่ากับ 40)



Stack

LOAD X, Stack เป็นการโหลดข้อมูลที่อยู่บนสุดของ Stack ไว้ที่ตัวแปร X (ถ้าข้อมูลที่อยู่บนสุดของ Stackมีค่า 1000 ค่า X จะมีค่าเท่ากับ 1000 นั่นเอง)