บทที่ 7
ชุดของคำสั่ง

~~~~~~

ชุดของคำสั่ง (Instruction Sets) หมายถึง ชุดของคำสั่งภาษาเครื่อง (Machine Instruction Sets) เนื่องจากภาษาที่โปรแกรมเมอร์ใช้เขียนโปรแกรมเป็นภาษาระดับสูงรวมถึงภาษาแอสแซมบลี (Assembly Language) ด้วยนั้นเครื่องคอมพิวเตอร์ไม่สามารถทำความเข้าใจได้ จึงต้องใช้ตัวแปลภาษาระดับสูงให้เป็นภาษาเครื่องจึงจะสามารถเอ็กซีคิวต์ (Execute) ได้ โดยที่ภาษาเครื่องนั้นประกอบด้วยตัวเลขเพียงอย่างเดียว

7.1 คุณสมบัติพื้นฐานของคำสั่ง

ชุดคำสั่ง (Instruction Sets) หมายถึง ชุดของคำสั่งที่โพรเซสเซอร์เอ็กซีคิวต์เพื่อดำเนินการตามที่โปรแกรมเมอร์ต้องการ อาจเรียกชุดคำสั่งว่า ภาษาเครื่อง (Machine Languages) หรือคำสั่งคอมพิวเตอร์ (Computer Instructions)

ในแต่ละชุดคำสั่งอาจประกอบด้วยคำสั่งที่หลากหลาย เช่น คำสั่งสำหรับการบวก โดยซีพียูจะต้องมีคำสั่งในการโหลดข้อมูลจากรีจิสเตอร์ลงในหน่วยความจำ แล้วจึงเรียกใช้คำสั่งสำหรับการบวก ต่อจากนั้นจะมีคำสั่งเพื่อเก็บค่าผลลัพธ์กลับไปเก็บไว้ในรีจิสเตอร์อีกครั้ง

ชุดคำสั่งของแต่ละโพรเซสเซอร์จะมีความแตกต่างกัน ได้แก่ ขนาดของคำสั่ง ประเภทของโอเปอเรชั่น ประเภทของโอเปอแรนด์ หรือประเภทของผลลัพธ์ นอกจากนี้ยังอาจเกิดจากโครงสร้างของภาษาชั้นสูงที่โปรแกรมเมอร์ใช้งาน เช่น ภาษา C ภาษา Pascal หรือภาษา Ada โดยโปรแกรมภาษาชั้นสูงที่โปรแกรมเมอร์เขียนขึ้นจะถูกคอมไพล์ (Compile) ด้วยคอมไพเลอร์ (Complier) หรือตัวแปลภาษานั้นๆ ให้เป็นภาษาเครื่องเพื่อทำงานตามคำสั่งต่อไป ซึ่งการคอมไพล์ต้องให้ตรงกับรุ่นของโพรเซสเซอร์ที่ใช้งานด้วย

7.2 วงรอบคำสั่ง

ภายในเครื่องคอมพิวเตอร์นั้นส่วนที่เป็น Instruction set จะเป็นส่วนที่เชื่อมโยงระหว่างผู้ออกแบบเครื่อง (Design) กับผู้ใช้โปรแกรมหรือผู้เขียนโปรแกรม (User) ผู้ออกแบบเครื่องจะทำหน้าที่ออกแบบว่าคอมพิวเตอร์เครื่องนั้นจะต้องใช้ทรานซิสเตอร์กี่ล้านตัว ใช้ชิปจำนวนเท่าไร แต่ในส่วนของ
~~~~~~

ผู้ใช้นั้นจะทราบว่าเครื่องคอมพิวเตอร์นั้นสามารถใช้งานชุดคำสั่งใดได้บ้าง เพื่อให้ผู้ใช้นำไปเป็นข้อมูลในการเขียนโปรแกรมระบบปฏิบัติการ เขียนโปรแกรมคอมไพเลอร์ เขียนแอพพลิเคชั่นโปรแกรมต่าง ๆ นอกจากนี้ผู้ออกแบบเครื่องจะออกแบบเรื่องของวงจร Adder, Multiplier, Divisor โดยอ้างอิงคำสั่งจากชุดของคำสั่ง และผู้ใช้ก็จะนำมาพิจารณาว่าต้องเขียนโปรแกรมแบบใดที่จะสามารถใช้ประโยชน์จากชุดของคำสั่งได้เต็มที่โดยไม่ต้องนำมาแปลอีก

ตัวอย่าง ผู้ออกแบบเครื่องออกแบบให้คอมพิวเตอร์เครื่องหนึ่งมีรีจิสเตอร์ 20 ตัว เป็นหน่วยความจำแบบ 16 บิต หรือ 32 บิต มี location เริ่มต้นจาก address ใดถึง address ใด ในส่วนของผู้ใช้ก็จะนำข้อมูลเหล่านี้ไปเขียนโปรแกรมระบบปฏิบัติการ เขียนโปรแกรมคอมไพเลอร์ และเขียนโปรแกรมจัดการฐานข้อมูล แล้วต่อเนื่องเป็นโปรแกรมประยุกต์ (Application Program) สำหรับงานต่าง ๆ ต่อไป

เมื่อเวลาคำสั่งใด ๆ จะเริ่มทำงานเครื่องคอมพิวเตอร์จะทำ Instruction Cycle ก่อนเป็นอันดับแรกซึ่งได้แก่

Fetch Instruction → Decode Instruction → Execute Instruction

1. ก่อน fetch จะต้องมีการคำนวณก่อนว่าคำสั่งอยู่ที่ไหน นั่นคือการทำการคำนวณหาแอดเดรสของชุดคำสั่ง (Instruction Address Calculation) ซึ่งส่วนใหญ่คำสั่งจะถูกเก็บไว้ในรีจิสเตอร์พีซี (register PC)
2. Fetch คำสั่งเข้าไปไว้ใน instruction register
3. Decode ว่าคำสั่งนั้นทำการดำเนินการ (operation) อะไรบ้าง เช่น บวก ลบ คูณ หาร
4. ถ้าต้องการหาข้อมูลว่าอยู่ที่ตำแหน่งใดก็ต้องทำการคำนวณแอดเดรสของตัวดำเนินการ (Operation Address Calculation) เพื่อหาว่าค่าที่ต้องการนำมาทำ operation นั้นอยู่ที่ตำแหน่งใดในหน่วยความจำ หรืออยู่ในรีจิสเตอร์ตัวใด
5. คำนวณจนได้ operand address แล้วจึงทำการ fetch operand
6. เมื่อได้ operand มาแล้วก็สามารถทำการ operate (execute คำสั่ง) และได้ผลลัพธ์ออกมา
7. ผลลัพธ์ที่ได้จะถูกเก็บไว้ที่ใด อาจเก็บไว้ที่รีจิสเตอร์หรือหน่วยความจำ ซึ่งไม่จำเป็นต้องเก็บที่ตำแหน่งเดิม ดังนั้นจึงต้องมี result operand address calculation เพื่อคำนวณหาตำแหน่งที่จะเก็บผลลัพธ์ว่าอยู่ที่ใด เมื่อคำนวณได้แล้วจึงนำผลลัพธ์ไปเก็บไว้ที่ตำแหน่งนั้น
8. กลับมาทำ fetch cycle อีกครั้ง ซึ่งต้องมีการอินเทอร์รัพท์ว่ายอมให้ทำในตอนนี้ (enable) หรือยังไม่พร้อมให้ทำ (disable)

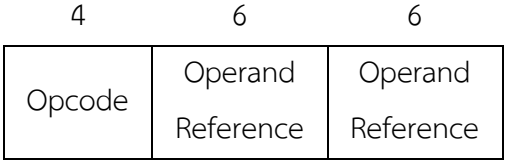
ในส่วนของ Machine Language จะต้องมีส่วนต่าง ๆ ดังนี้

1. Operand Code (opcode) มีหน้าที่กำหนด operation ที่จะต้องทำ เช่น ADD, I/O ซึ่งรูปแบบของคำสั่งจะเป็นตัวเลขฐานสอง
2. Source Operand Reference บอกว่ามีตัวแปร (operand) ใดที่จะมากระทำกับ opcode
3. Result Operand Reference บอกว่า ผลลัพธ์ของ operand จะเก็บไว้ที่ใด

4. Next Instruction Address จะบอกกับซีพียูว่าจะต้องไป fetch คำสั่งถัดไปที่ไหน ซึ่ง next instruction จะถูกเก็บไว้ที่หน่วยความจำหลัก หรือเก็บไว้ที่หน่วยความจำสำรอง (disk) ในกรณีที่ใช้หน่วยความจำเสมือน ซึ่งมักจะเป็นตำแหน่งที่ติดกับคำสั่งก่อนหน้ามัน เนื่องจากการเขียนโปรแกรมของโปรแกรมเมอร์มักจะเขียนแบบเป็นลำดับคำสั่งอยู่แล้ว ดังนั้นกลุ่มของคำสั่งที่อยู่ติดกันจะถูก fetch มารอไว้ที่หน่วยความจำหลักด้วย

7.3 รูปแบบของคำสั่ง

ภายในคอมพิวเตอร์นั้นแต่ละคำสั่งจะถูกแทนด้วยลำดับของบิต (ตัวเลขฐานสอง) และแต่ละคำสั่งจะถูกแบ่งออกเป็นฟิลด์ เรียกว่า เป็นรูปแบบของคำสั่ง (Instruction Format) มีรูปแบบเป็นดังนี้



Opcode จะแสดงด้วยตัวย่อ เรียกว่า mnemonics ที่มีหน้าที่กำหนด operation ต่าง ๆ เช่น

ADD	หมายถึง	บวก
SUB	หมายถึง	ลบ
MPY	หมายถึง	คูณ
DIV	หมายถึง	หาร
LOAD	หมายถึง	การดึงข้อมูลจากหน่วยความจำ
STOR	หมายถึง	การเก็บข้อมูลลงหน่วยความจำ

ส่วน operand มักจะแสดงด้วยสัญลักษณ์ เช่น ADD R, Y หมายถึง บวกค่าที่อยู่ในหน่วยความจำตำแหน่งที่กำหนดด้วย Y กับค่าใน register R แล้วเก็บผลลัพธ์ไว้ที่ register R

7.4 ชนิดของคำสั่ง

คอมพิวเตอร์จะต้องมีชุดคำสั่งที่ยอมให้ผู้ใช้ทำงานกับข้อมูลได้ตามต้องการ ตัวอย่างคำสั่งดังนี้

X = X + Y

คำสั่งนี้จะสั่งให้คอมพิวเตอร์บวกค่าที่เก็บไว้ใน Y เข้ากับค่าที่เก็บไว้ใน X แล้วเก็บค่าผลลัพธ์ไว้ที่ X ในแง่ของภาษาเครื่องจะต้องมีการกำหนดตำแหน่งของโอเปอแรนด์ X และ Y ไว้ในหน่วยความจำก่อนหน้านี้แล้วใน

ตอนกำหนดตัวแปร (define) ดังนั้นสมมติว่า X และ Y ถูกเก็บไว้ที่ตำแหน่ง 513 และ 514 ตามลำดับ คอมพิวเตอร์จะมีการทำงานดังนี้คือ

1. โหลดค่าจากหน่วยความจำตำแหน่งที่ 513 เข้าไปไว้ในรีจิสเตอร์
2. แอดค่าจากหน่วยความจำตำแหน่งที่ 514 เข้าไปไว้ในรีจิสเตอร์
3. เก็บค่าในรีจิสเตอร์ไว้ในหน่วยความจำตำแหน่งที่ 513

โปรแกรมที่เขียนด้วยภาษาระดับสูงจะต้องถูกแปลให้เป็นคำสั่งในรูปแบบของภาษาเครื่องโดยตัวแปลภาษา (complier) ในระดับที่ซีพียูสามารถเอ็กซีคิวต์ได้ ดังนั้นชุดคำสั่งภาษาเครื่องของซีพียูแต่ละรุ่นจะต้องมีอยู่เพียงพอสำหรับการแปลคำสั่งต่าง ๆ ของภาษาระดับสูง ซึ่งถ้าหากซีพียูมีชุดคำสั่งน้อยก็จะต้องใช้หลาย ๆ คำสั่งย่อยหรือหลายการกระทำ (operation) ในการที่จะปฏิบัติการตามคำสั่งภาษาระดับสูงเพียงคำสั่งเดียว โดยปกติคอมพิวเตอร์จะต้องมีชุดคำสั่งที่ยอมให้ผู้ใช้งานทำงานกับข้อมูลได้ตามต้องการ ซึ่งสามารถแบ่งชุดของคำสั่งได้เป็น

1. Data processing : คำสั่งทางคณิตศาสตร์และตรรกะ เป็นคำสั่งในการประมวลผลข้อมูล เช่น การคำนวณและเปรียบเทียบข้อมูล
2. Data storage : คำสั่งจัดการหน่วยความจำ เป็นคำสั่งที่กระทำกับหน่วยความจำ
3. Data movement : คำสั่งจัดการอินพุต/เอาต์พุต เป็นคำสั่งที่เกี่ยวกับอินพุต/เอาต์พุต
4. Control : คำสั่งตรวจสอบเงื่อนไขและกระโดดไปทำงาน เป็นคำสั่ง Test & Branch

7.5 จำนวนของแอดเดรสในคำสั่ง

โดยทั่วไปการกล่าวถึงสถาปัตยกรรมคอมพิวเตอร์จะกล่าวถึงจำนวนของแอดเดรส ซึ่งปัจจุบันมีความสำคัญน้อยกว่าในเรื่องของการออกแบบซีพียู จำนวนของแอดเดรสของระบบมีผลต่อวงรอบการทำงานของคำสั่งเครื่อง ยิ่งคอมพิวเตอร์มีจำนวนแอดเดรสมากก็จะยิ่งทำให้วงรอบการทำงานน้อยลงทำให้คอมพิวเตอร์ทำงานได้เร็วขึ้น

การกำหนดแอดเดรสหรือตำแหน่งในหน่วยความจำของคำสั่งชุดหนึ่งอย่างน้อยควรจะมี 1 หรือ 2 โอเปอแรนด์ คือ 1 แอดเดรสสำหรับโอเปอแรนด์ผลลัพธ์ อีก 1 แอดเดรสสำหรับคำสั่งถัดไปที่จะต้องทำ ซึ่งตามปกติแล้วซีพียูส่วนมากจะมีคำสั่งที่เป็น 1 หรือ 2 หรือ 3 แอดเดรส ส่วนแอดเดรสที่เป็น instruction จะรู้จักกันดีในชื่อของ program counter (PC)

<u>ตัวอย่างที่ 7.1</u> ประโยคคำสั่ง y = (a-b) / (c+d*e)

1 Address		2 Address		3 Address	
Instruction	Comment	Instruction	Comment	Instruction	Comment
LOAD A	AC ← D	MOVE Y,A	Y ← A	SUB Y,A,B	Y ← A-B
MPY E	AC ← AC*B	SUB Y,A	Y ← Y-B	MPY T,D,E	T ← D*E
ADD C	AC ← AC+C	MOVE T,D	T ← D	ADD T,T,C	T ← T+C
STOR Y	Y ← AC	MPY T,E	T ← T*E	DIV Y,Y,T	Y ← Y/T
LOAD A	AC ← A	ADD T,C	T ← T-C		
SUB B	AC ← AC-B	DIV Y,T	Y ← Y/T		
DIV Y	AC ← AC/Y				
STOR	Y ← AC				

ตัวอย่างที่ 7.2

0 Address (Zero Address)	
คำสั่ง	สิ่งที่กระทำ
PUSH A	ย้าย A ไปอยู่บนสุดของสแต็ก
PUSH B	ย้าย B ไปอยู่บนสุดของสแต็ก
MULTIPLY	ลบ A,B ออกจากสแต็ก และแทนค่าด้วย A*B
PUSH C	ย้าย C ไปอยู่บนสุดของสแต็ก
PUSH C	ย้ายก๊อปปี้ที่สองของ C ไปอยู่บนสุดของสแต็ก
MULTIPLY	ลบ C,C ออกจากสแต็ก และแทนค่าด้วย C*C
ADD	ลบ C*C, A*B ออกจากสแต็ก และแทนค่าด้วยผลบวกของ (C*C)+(A*B)

1 Address (One Address)	
คำสั่ง	สิ่งที่กระทำ
LOAD A	ย้าย A ไปยังแอคคิวมูเลเตอร์ AC
MULTIPLY B	AC := AC*B
STORE T	ย้าย AC ไปหน่วยความจำตำแหน่ง T
LOAD C	ย้าย C ไปยังแอคคิวมูเลเตอร์ AC
MULTIPLY C	AC := AC*C
ADD T	AC := AC+T
STORE X	เก็บค่าผลลัพธ์ในหน่วยความจำตำแหน่ง X

2 Address (Two Address)	
คำสั่ง	สิ่งที่กระทำ
MOVE T,A	T := A
MULTIPLY T,B	T := T*B
MOVE X,C	X := C
LOAD C	ย้าย C ไปยังแอคคิวมูเลเตอร์ AC
MULTIPLY X,C	X := X*C
LOAD X,T	X := X+T

3 Address (Three Address)	
คำสั่ง	สิ่งที่กระทำ
MULTIPLY T,A,B	T := A*B
MULTIPLY X,C,C	X := C*C
ADD X,X,T	X := X+T

ในการเขียนโปรแกรมนั้นโปรแกรมเมอร์ต้องการเขียนโปรแกรมอย่างอิสระ สามารถกำหนดตัวแปรต่างๆ ได้ตามต้องการ แต่ในแง่การทำงานของซีพียูแล้วนั้นยิ่งมีการอ้างถึงแอดเดรสในหน่วยความจำน้อยเท่าไรก็จะยิ่งทำให้ซีพียูทำงานได้เร็วขึ้นเท่านั้น เพราะไม่ต้องเสียเวลาในการคำนวณหาตำแหน่งของโอเปอแรนด์ที่ใช้ ไม่ต้องเสียเวลาในการถ่ายโอนข้อมูล (transfer) จากหน่วยความจำ

7.6 ชนิดของตัวปฏิบัติการ

พื้นฐานของโอเปอเรชั่นสามารถแบ่งเป็นกลุ่มได้ดังนี้คือ

1. โอเปอเรชั่นทางด้านการถ่ายโอนข้อมูล (Data Transfer) คือ การเคลื่อนย้ายข้อมูลจากที่หนึ่งไปยังอีกที่หนึ่ง เช่น จากรีจิสเตอร์ไปยังรีจิสเตอร์ หรือจากรีจิสเตอร์ไปยังหน่วยความจำ หรือจากหน่วยความจำไปยังรีจิสเตอร์ โดยถ้ามีตัวแปรหรือโอเปอแรนด์อยู่ในหน่วยความจำซีพียูจะต้อง

- คำนวณหาที่อยู่ในหน่วยความจำ โดยขึ้นอยู่กับวิธีการกำหนดที่อยู่ของตัวแปรนั้น ซึ่งอาจกำหนดในแบบ direct หรือแบบ relative
- ถ้าที่อยู่นั้นอ้างถึงหน่วยความจำเสมือน จะต้องมีการแปลงแอดเดรสเสมือนไปเป็นแอดเดรสจริงเสียก่อน
- ถ้าแอดเดรสนั้นอยู่ในแคช ก็จะสามารถใช้ข้อมูลจากในแคชได้เลย
- ถ้าแอดเดรสนั้นไม่อยู่ในแคช ก็จะไปหาข้อมูลจากหน่วยความจำ

2. โอเปอเรชั่นทางด้านคณิตศาสตร์ (Arithmetic) นอกจากคำสั่งในการบวก ลบ คูณ หาร แล้วยังมีคำสั่งอื่น ๆ อีกเช่น absolute, negative, increment, decrement

3. โอเปอเรชั่นทางด้านตรรกะ (Logical) คือ การกระทำทางตรรกศาสตร์ที่เรียกว่า บูลีนโอเปอเรชั่น (Boolean Operation) เช่น

P	Q	NOT P	NOT Q	P AND Q	P OR Q	P XOR Q	P=Q
0	0	1	1	0	0	0	1
0	1	1	0	0	0	1	0
1	0	0	1	0	1	1	0
1	1	0	0	1	1	0	1

7.7 การควบคุมการโยกย้าย

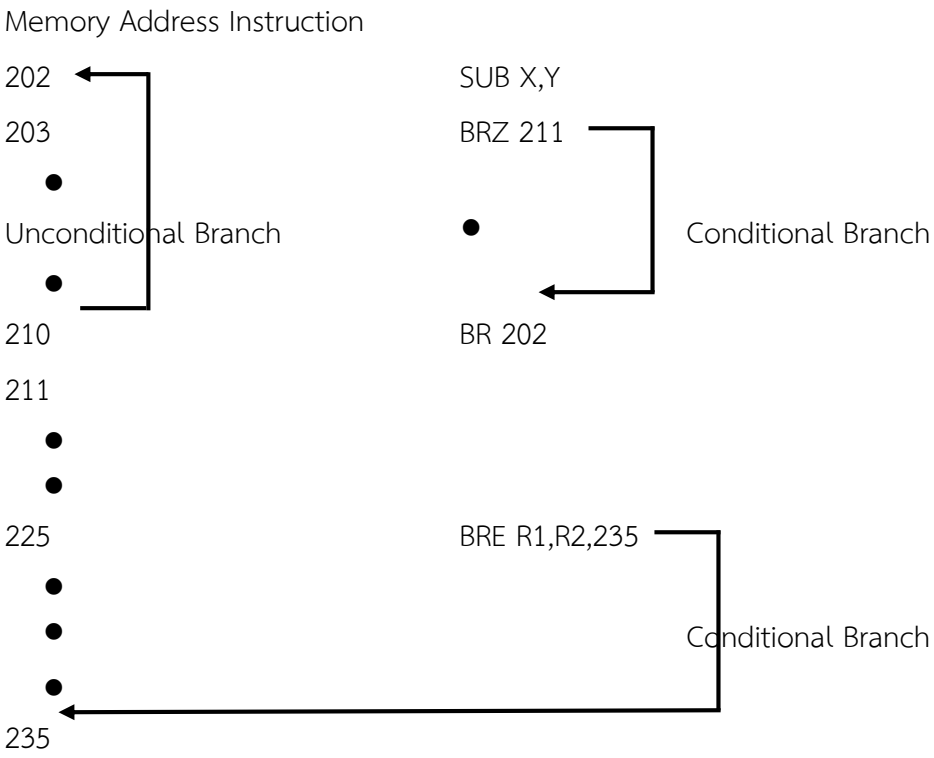
ตามปกติแล้วคำสั่งถัดไปที่จะถูกกระทำนั้น จะอยู่ในหน่วยความจำตำแหน่งที่ถัดจากตำแหน่งของหน่วยความจำปัจจุบัน เนื่องจากเวลาเราดึง (fetch) คำสั่งมานั้น เราจะดึงมาครั้งละ 1 แถว (line) ซึ่งจะมีคำสั่งในกลุ่มเดียวกันติดมาด้วย จึงทำให้การ execute คำสั่งของซีพียูทำได้รวดเร็วมากขึ้น แต่ในการเขียนโปรแกรมนั้นโปรแกรมเมอร์อาจต้องการเปลี่ยนแปลงลำดับการทำงานจากปกติที่เป็นลำดับไปยังคำสั่งอื่นที่อยู่นอกลำดับ โปรแกรมเมอร์ทราบว่าคำสั่งถัดไปที่จะถูกดึงนั้นจะถูกระบุด้วย program counter (PC) ซึ่งซีพียูเป็นตัวเพิ่มค่าใน PC ทีละ 1 ดังนั้นเมื่อเกิดเหตุการณ์ที่ต้องการจะข้ามไปยังตำแหน่งอื่น ๆ นอกเหนือจากที่กำหนดไว้ใน PC ก็จะต้องระบุให้ซีพียูทำการอัพเดตตำแหน่งนั้นลงไปที่ PC ด้วยเพื่อให้ซีพียูสามารถที่จะรู้ตำแหน่งของการดึงคำสั่งได้ มีสาเหตุต่อไปนี้ที่ทำให้เกิดการอัพเดตตำแหน่งใน PC

1. การ Branch (Branch Instruction or Jump Instruction)

เป็นการ branch แบบมีเงื่อนไข และเมื่อเกิดการ branch (เงื่อนไขเป็นจริง) จะมีการอัพเดต PC ด้วยตำแหน่งที่ระบุไว้ในคำสั่ง branch ถ้าเงื่อนไขไม่เป็นจริง (คือไม่เกิดการ branch) ก็จะไปทำคำสั่งที่อยู่ในลำดับถัดไปของ PC เช่น

```
BRP X          branch to location X if result is positive
BRN X          branch to location X if result is negative
BRZ X          branch to location X if result is zero
BRO X          branch to location X if overflow occurs
BRE R1,R2,X    branch to location X if content of R1=content of R2
```

ตัวอย่างที่ 7.3 คำสั่ง branch



2. การ Skip

เป็นคำสั่งที่ไม่ได้บอกหรือระบุว่าให้ไปทำที่ตำแหน่งใด แต่จะบอกว่าให้ข้ามคำสั่งปัจจุบันนี้ไปถ้าไม่ตรงตามเงื่อนไข

ตัวอย่างที่ 7.4

```
301
●
●
●
309     ISZ R1
310     BR 301
311
```

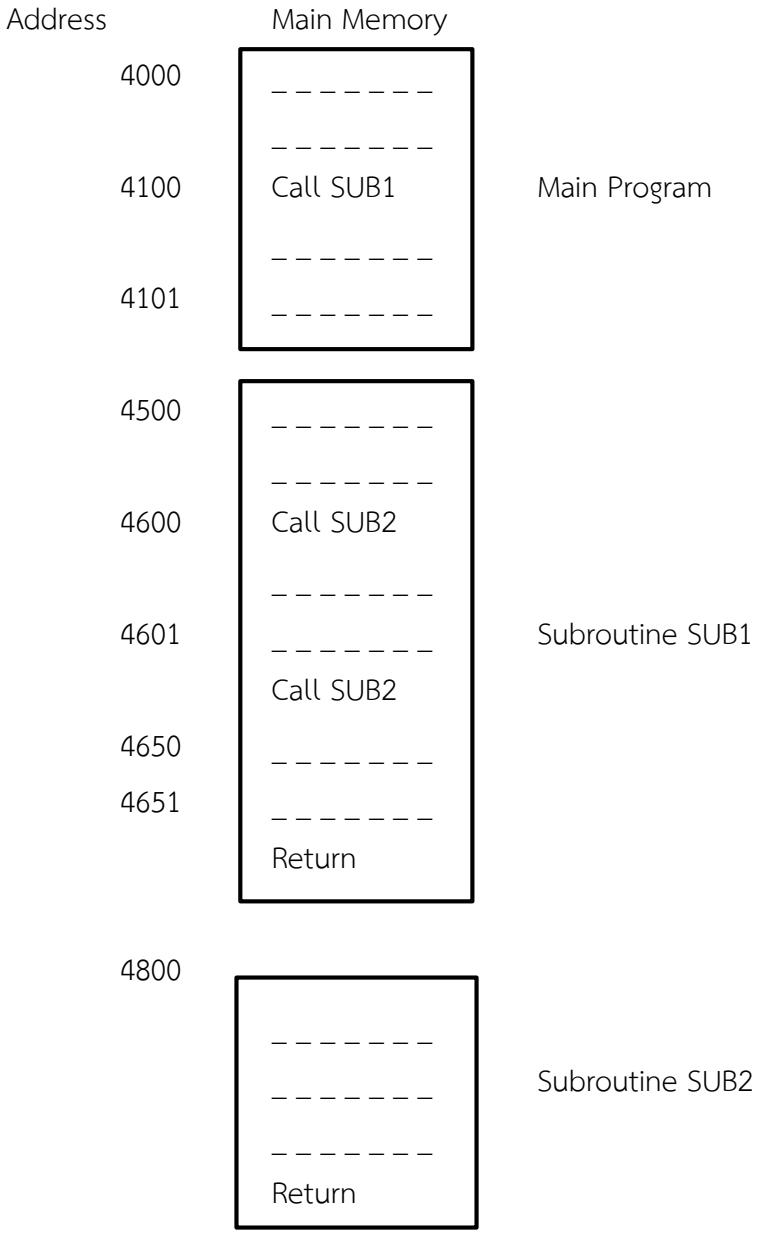
เป็นการควบคุมการทำงานที่ซ้ำ ๆ กัน เช่น การวนลูป โดยที่มีการกำหนดคำสั่งนี้ไว้ที่ท้ายของกลุ่มคำสั่งที่ต้องการทำซ้ำ

เงื่อนไข ISZ R1 หมายถึง increment and skip if R1 = zero (ค่า R1 ตอนเริ่มต้นถูกเซตให้เป็น negative) คำสั่ง ISZ จะเพิ่มค่า R1 ทีละ 1 และทดสอบเงื่อนไขว่าถ้า R1 = 0 จะ skip ข้ามคำสั่ง BR ไปทำคำสั่งถัดไป (ตำแหน่งที่ 311) แต่ถ้า R1 ≠ 0 ก็จะ branch กลับไปทำคำสั่งที่ 301

3. การใช้ Subroutine Call

เป็นโปรแกรมย่อยที่ถูกเรียกโดยโปรแกรมใหญ่หรือโปรแกรมหลักอีกต่อหนึ่ง โดยใช้คำสั่ง call หรือ perform โดยเมื่อโปรแกรมหลักเรียกใช้คำสั่ง call ซีพียูจะอัพเดต PC ด้วยตำแหน่งเริ่มต้นที่หน่วยความจำของ subroutine และเมื่อเอ็กซีคิวต์ subroutine จนกระทั่งเจอคำสั่ง return หรือ exit จึงจะกลับมาทำคำสั่งปกติของโปรแกรมหลักต่อไป

<u>ตัวอย่างที่ 7.5</u>



ข้อกำหนดในการ Call Subroutine

1. Subroutine สามารถถูกเรียกได้จากตำแหน่งใดก็ได้ของโปรแกรม
2. Subroutine สามารถถูกเรียกจากภายใน subroutine อื่น ๆ หรือตัวมันเองได้ เรียกว่า nesting subroutine
3. แต่ละ subroutine ที่ถูกเรียกจะสัมพันธ์กับคำสั่ง return ที่จะบอกว่าสิ้นสุด subroutine นั้น ๆ แล้ว

ซีพียูจะต้องรู้จักจุดสิ้นสุดของ subroutine หรือรู้ว่าคำสั่ง return อยู่ที่ตำแหน่งใดในหน่วยความจำ โดยซีพียูจะเก็บ return address ไว้ที่ top of stack โดยเมื่อซีพียู execute คำสั่ง call จะมีการเก็บค่าของ return address ไว้บน stack โดยการ push stack และเมื่อซีพียู execute ถึงคำสั่ง return ก็จะมีการ pop top of stack ออกมาเป็น address ที่จะ execute คำสั่งต่อไป

7.8 การกำหนดแอดเดรสหน่วยความจำ (Addressing)

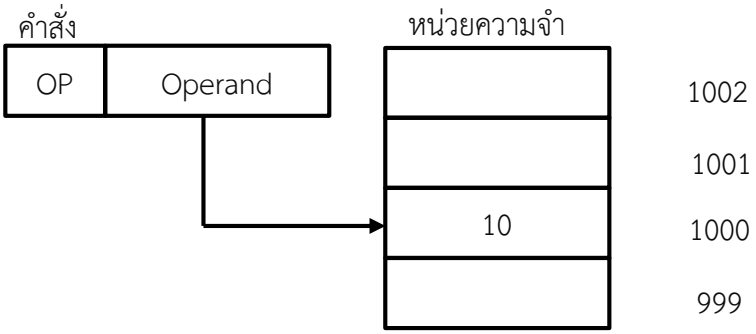
การ addressing คือ การกำหนดพื้นที่ในหน่วยความจำสำหรับคำสั่งหรือตัวแปร (Operation or Operand) ต่าง ๆ ที่เราใช้ รูปแบบหรือเทคนิคที่ใช้ในการ addressing มีหลายรูปแบบดังนี้

1. การกำหนดแอดเดรสแบบให้ค่าโดยตรง (Immediate Addressing)

   เช่น LOAD X, #1000 เป็นการโหลดข้อมูล 1000 ไว้ที่ตัวแปร X

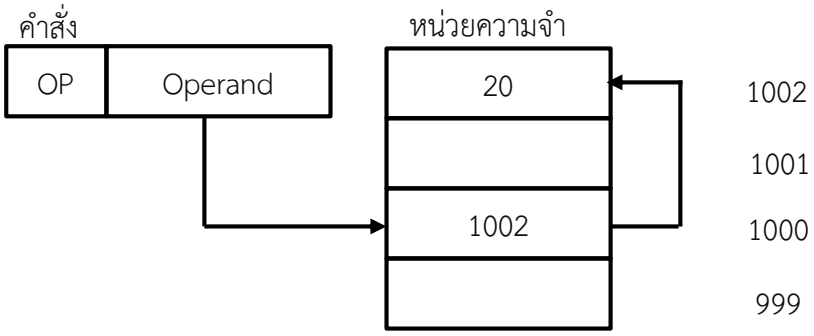
2. การกำหนดแอดเดรสโดยตรง (Direct Addressing)

   เช่น LOAD X, 1000 เป็นการโหลดข้อมูลที่แอดเดรส 1000 ไว้ที่ตัวแปร X
   (ถ้าที่แอดเดรส 1000 มีค่า 10 ดังนั้น X จะมีค่าเท่ากับ 10)



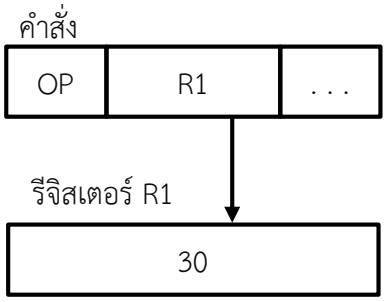
3. การกำหนดแอดเดรสทางอ้อม (Indirect Addressing)

เช่น LOAD X, 1000 เป็นการโหลดข้อมูลที่อยู่บนแอดเดรสที่เก็บอยู่ในแอดเดรส 1000 (ถ้าที่แอดเดรส 1000 มีค่า 1002 ดังนั้นข้อมูลที่แท้จริงจะอยู่ที่แอดเดรส 1002 ตามภาพคือ 20 ดังนั้น X จะมีค่าเท่ากับ 20)



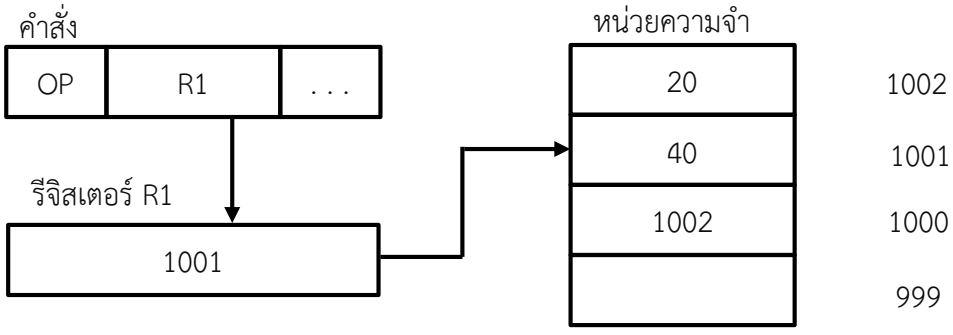
4. การกำหนดแอดเดรสผ่านรีจิสเตอร์โดยตรง (Register Direct Addressing)

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



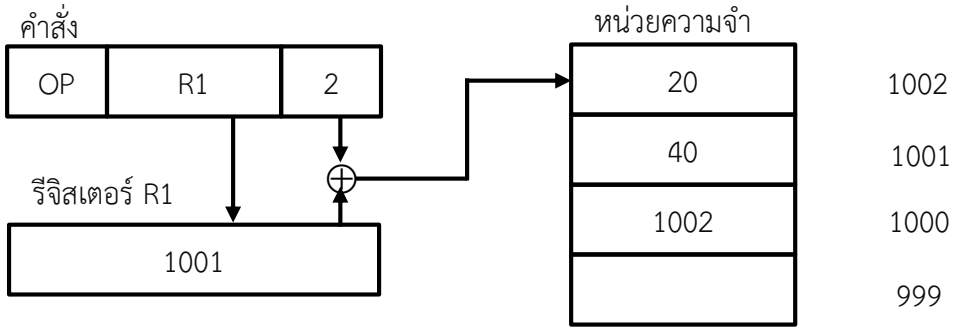
5. การกำหนดแอดเดรสผ่านรีจิสเตอร์ทางอ้อม (Register Indirect Addressing)

เช่น LOAD X, (R1) เป็นการโหลดข้อมูลที่อยู่ในแอดเดรสที่เก็บอยู่ในรีจิสเตอร์ R1 ไว้ที่ตัวแปร X (ถ้าที่รีจิสเตอร์ R1 เก็บค่า 1001 และที่แอดเดรส 1001 ของหน่วยความจำมีค่า 40 ดังนั้น X จะมีค่าเท่ากับ 40)



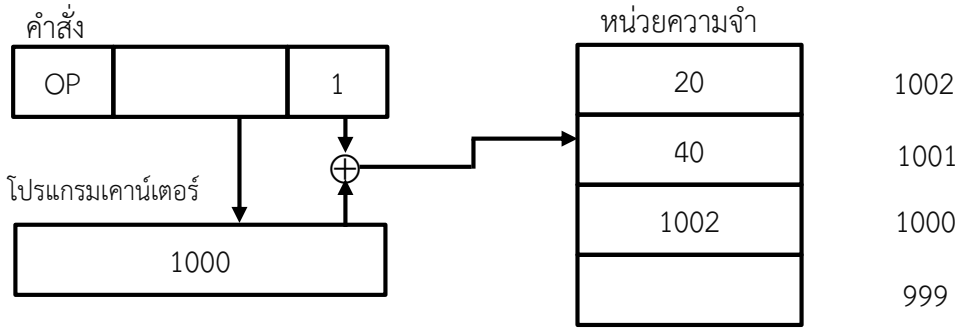
6. การกำหนดแอดเดรสแบบแทนที่ (Displacement หรือ Indexed Addressing)

เช่น LOAD X, (R1)+Constant เป็นการโหลดข้อมูลจากแอดเดรสบนหน่วยความจำที่เกิดจากค่าในรีจิสเตอร์บวกกับค่าคงที่ ไว้ที่ตัวแปร X (ถ้าที่รีจิสเตอร์มีค่า 1000, ค่าคงที่เท่ากับ 2 และค่าที่แอดเดรส 1002 มีค่าเท่ากับ 20 ดังนั้น X จะมีค่าเท่ากับ 20)



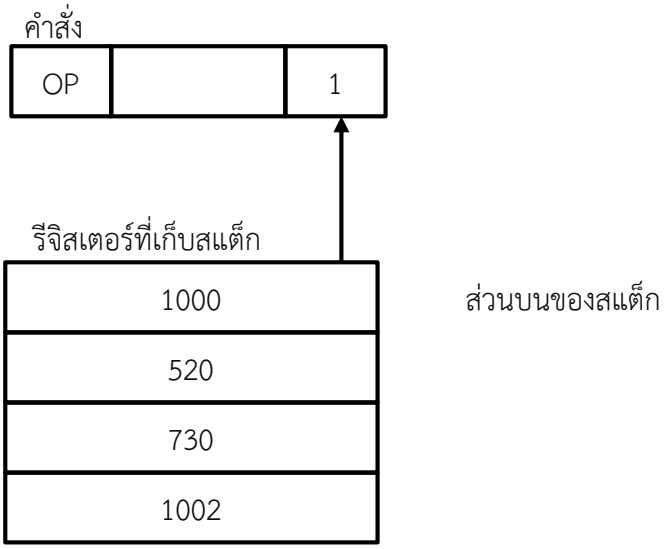
7. การกำหนดแอดเดรสแบบสัมพันธ์ (Relative Addressing)

เช่น LOAD X, PC+Constant เป็นการโหลดข้อมูลจากแอดเดรสบนหน่วยความจำที่เกิดจากค่าในโปรแกรมเคาน์เตอร์บวกกับค่าคงที่ ไว้ที่ตัวแปร X (ถ้าในโปรแกรมเคาน์เตอร์มีค่า 1000, ค่าคงที่เท่ากับ 1 และค่าที่แอดเดรส 1001 มีค่าเท่ากับ 40 ดังนั้น X จะมีค่าเท่ากับ 40)



8. การใช้สแต็ก (Stack)

เช่น LOAD X, Stack เป็นการโหลดข้อมูลที่อยู่บนสุดของสแต็ก ไว้ที่ตัวแปร X (ถ้าข้อมูลที่อยู่บนสุดของสแต็กมีค่า 1000 ดังนั้น X จะมีค่าเท่ากับ 1000)

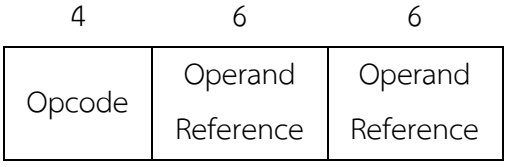


7.9 สรุป

ชุดคำสั่ง (Instruction Sets) หมายถึง ชุดของคำสั่งที่โพรเซสเซอร์เอ็กซีคิวต์เพื่อดำเนินการตามที่โปรแกรมเมอร์ต้องการ เป็นภาษาเครื่อง (Machine Languages) หรือคำสั่งคอมพิวเตอร์ (Computer Instructions)

เมื่อเวลาคำสั่งใด ๆ จะเริ่มทำงานเครื่องคอมพิวเตอร์จะทำ Instruction Cycle ก่อนเป็นอันดับแรกซึ่งได้แก่ Fetch Instruction → Decode Instruction → Execute Instruction

รูปแบบของคำสั่ง (Instruction Format) มีรูปแบบเป็นดังนี้



Opcode จะแสดงด้วยตัวย่อ เรียกว่า mnemonics ที่มีหน้าที่กำหนด operation ต่าง ๆ เช่น

ADD หมายถึง บวก

SUB หมายถึง ลบ

MPY หมายถึง คูณ

DIV หมายถึง หาร

LOAD หมายถึง การดึงข้อมูลจากหน่วยความจำ

STOR หมายถึง การเก็บข้อมูลลงหน่วยความจำ

ส่วน operand มักจะแสดงด้วยสัญลักษณ์ เช่น ADD R, Y หมายถึง บวกค่าที่อยู่ในหน่วยความจำตำแหน่งที่กำหนดด้วย Y กับค่าใน register R แล้วเก็บผลลัพธ์ไว้ที่ register R

สามารถแบ่งชุดของคำสั่งได้เป็น

1. Data processing : คำสั่งทางคณิตศาสตร์และตรรกะ เป็นคำสั่งในการประมวลผลข้อมูล เช่น การคำนวณและเปรียบเทียบข้อมูล

2. Data storage : คำสั่งจัดการหน่วยความจำ เป็นคำสั่งที่กระทำกับหน่วยความจำ

3. Data movement : คำสั่งจัดการอินพุต/เอาต์พุต เป็นคำสั่งที่เกี่ยวกับอินพุต/เอาต์พุต

4. Control : คำสั่งตรวจสอบเงื่อนไขและกระโดดไปทำงาน เป็นคำสั่ง Test & Branch

การกำหนดแอดเดรสหรือตำแหน่งในหน่วยความจำของคำสั่งชุดหนึ่งอย่างน้อยควรจะมี 1 หรือ 2 โอเปอแรนด์ คือ 1 แอดเดรสสำหรับโอเปอแรนด์ผลลัพธ์ อีก 1 แอดเดรสสำหรับคำสั่งถัดไปที่จะต้องทำ ซึ่งตามปกติแล้วซีพียูส่วนมากจะมีคำสั่งที่เป็น 1 หรือ 2 หรือ 3 แอดเดรส ส่วนแอดเดรสที่เป็น instruction จะรู้จักกันดีในชื่อของ program counter (PC)

พื้นฐานของโอเปอเรชั่นสามารถแบ่งเป็นกลุ่มได้ดังนี้คือ

1. โอเปอเรชั่นทางด้านการถ่ายโอนข้อมูล (Data Transfer) คือ การเคลื่อนย้ายข้อมูลจากที่หนึ่งไปยังอีกที่หนึ่ง เช่น จากรีจิสเตอร์ไปยังรีจิสเตอร์ หรือจากรีจิสเตอร์ไปยังหน่วยความจำ หรือจากหน่วยความจำไปยังรีจิสเตอร์

2. โอเปอเรชั่นทางด้านคณิตศาสตร์ (Arithmetic) นอกจากคำสั่งในการบวก ลบ คูณ หาร แล้วยังมีคำสั่งอื่น ๆ อีกเช่น absolute, negative, increment, decrement

3. โอเปอเรชั่นทางด้านตรรกะ (Logical) คือ การกระทำทางตรรกศาสตร์ที่เรียกว่า บูลีนโอเปอเรชั่น (Boolean Operation)

สาเหตุต่อไปนี้ที่ทำให้เกิดการอัพเดตตำแหน่งใน PC (Program Counter)

1. การ Branch (Branch Instruction or Jump Instruction) เป็นการ branch แบบมีเงื่อนไข และเมื่อเกิดการ branch (เงื่อนไขเป็นจริง) จะมีการอัพเดต PC ด้วยตำแหน่งที่ระบุไว้ในคำสั่ง branch ถ้าเงื่อนไขไม่เป็นจริง (คือไม่เกิดการ branch) ก็จะไปทำคำสั่งที่อยู่ในลำดับถัดไปของ PC

2. การ Skip เป็นคำสั่งที่ไม่ได้บอกหรือระบุว่าให้ไปทำที่ตำแหน่งใด แต่จะบอกว่าให้ข้ามคำสั่งปัจจุบันนี้ไปถ้าไม่ตรงตามเงื่อนไข

3. การใช้ Subroutine Call เป็นโปรแกรมย่อยที่ถูกเรียกโดยโปรแกรมใหญ่หรือโปรแกรมหลักอีกต่อหนึ่ง โดยใช้คำสั่ง call หรือ perform โดยเมื่อโปรแกรมหลักเรียกใช้คำสั่ง call ซีพียูจะอัพเดต PC ด้วยตำแหน่งเริ่มต้นที่หน่วยความจำของ subroutine และเมื่อเอ็กซีคิวต์ subroutine จนกระทั่งเจอคำสั่ง return หรือ exit จึงจะกลับมาทำคำสั่งปกติของโปรแกรมหลักต่อไป

การ addressing คือ การกำหนดพื้นที่ในหน่วยความจำสำหรับคำสั่งหรือตัวแปร (Operation or Operand) ต่าง ๆ ที่เราใช้ รูปแบบหรือเทคนิคที่ใช้ในการ addressing มีหลายรูปแบบดังนี้

1. การกำหนดแอดเดรสแบบให้ค่าโดยตรง (Immediate Addressing)

2. การกำหนดแอดเดรสโดยตรง (Direct Addressing)

3. การกำหนดแอดเดรสทางอ้อม (Indirect Addressing)
4. การกำหนดแอดเดรสผ่านรีจิสเตอร์โดยตรง (Register Direct Addressing)
5. การกำหนดแอดเดรสผ่านรีจิสเตอร์ทางอ้อม (Register Indirect Addressing)
6. การกำหนดแอดเดรสแบบแทนที่ (Displacement หรือ Indexed Addressing)
7. การกำหนดแอดเดรสแบบสัมพันธ์ (Relative Addressing)
8. การใช้สแต็ก (Stack)

คำถามทบทวนประจำบท

~~~~~~

คำสั่ง จงตอบคำถามต่อไปนี้ โดยอธิบายให้เข้าใจ

1. จงอธิบาย Instruction Cycle ของเครื่องคอมพิวเตอร์
2. จงอธิบายส่วนการทำงานต่าง ๆ ของ Machine Language
3. จงอธิบายรูปแบบของคำสั่ง (Instruction Format)
4. ชนิดของคำสั่งแบ่งเป็นกี่ประเภท อะไรบ้าง
5. จำนวนของแอดเดรสในคำสั่ง มีผลต่อความเร็วในการทำงานของคอมพิวเตอร์อย่างไร
6. จงอธิบายพื้นฐานของโอเปอเรชั่นแต่ละกลุ่ม
7. การใช้ Subroutine Call มีลักษณะการทำงานอย่างไร
8. ยกตัวอย่าง การกำหนดแอดเดรสหน่วยความจำมา 2 ประเภท
~~~~~~