หน่วยที่ 4 กลุ่มคำสั่งและการใช้งาน

อดิศักดิ์ ชิณะวงศ์

เอกสารประกอบการเรียนวิชาไมโครคอนโทรลเลอร์

เผยแพร่ที่ www.Adisak51.com

1. กลุ่มคำสั่งเคลื่อนย้ายข้อมูล

กลุ่มคำสั่งเคลื่อนย้ายข้อมูล (Data Transfer Instruction) แบ่งออกเป็นตามประเภทของตำแหน่งหน่วยความจำที่ใช้ดังนี้

1.1 กลุ่มคำสั่งเคลื่อนย้ายข้อมูลระหว่างหน่วยความจำภายในแรม

เป็นกลุ่มคำสั่งเคลื่อนย้ายข้อมูลระหว่างหน่วยความจำภายใน กับรีจิสเตอร์ต่างๆ หรือระหว่างหน่วยความจำด้วยกัน มีดังต่อไปนี้

MOV A,Rn	เคลื่อนย้ายค่าข้อมูลในรีจิสเตอร์ Rn ไปเป็นข้อมูลของรีจิสเตอร์ A
MOV A, direct	เคลื่อนย้ายข้อมูลจากหน่วยความจำ direct ไปเป็นข้อมูลของรีจิสเตอร์ A
MOV A,@Ri	เคลื่อนย้ายข้อมูลจากหน่วยความจำชี้โดย Ri เป็นข้อมูลในรีจิสเตอร์ A
MOV A,#data	เคลื่อนย้ายค่าคงที่ 8 บิตไปเก็บเป็นข้อมูลในรีจิสเตอร์ A
MOV Rn,A	เคลื่อนย้ายข้อมูลจากในรีจิสเตอร์ A ไปเป็นข้อมูลในรีจิสเตอร์ Rn
MOV Rn,direct	เคลื่อนย้ายข้อมูลจากหน่วยความจำ direct ไป Rn
MOV direct,A	เคลื่อนย้ายข้อมูลในรีจิสเตอร์ A ไปยังหน่วยความจำ direct
MOV direct,Rn	เคลื่อนย้ายข้อมูลในรีจิสเตอร์ Rn ไปยังหน่วยความจำ direct
MOV direct,direct	เคลื่อนย้ายข้อมูลระหว่างหน่วยความจำภายใน
MOV direct,@Ri	เคลื่อนย้ายข้อมูลจากหน่วยความจำที่ชี้แอดเดรสโดย Ri ไปที่ direct
MOV direct,#data	เคลื่อนย้ายค่าคงที่ 8 บิตไปยังหน่วยความจำ direct
MOV @Ri,A	เคลื่อนย้ายข้อมูลในรีจิสเตอร์ A ไปหน่วยความจำที่ชี้แอดเดรสโดย Ri
MOV @Ri,direct	เคลื่อนย้ายข้อมูลหน่วยความจำ direct ไปหน่วยความจำที่ชี้โดย Ri
MOV @Ri,#data	เคลื่อนย้ายค่าคงที่ 8 บิตไปยังหน่วยความจำที่ชี้แอดเดรสโดย Ri

กำหนดให้ R1 = 23H กำหนดหน่วยความจำข้อมูลภายในตำแหน่งแอดเดรสที่ 23H = 1DH

คำสั่ง	MOV A, Rn	A ← (Rn)
ตัวอย่าง	MOV A, R1	ผลที่ได้รีจิสเตอร์A = 23H
คำสั่ง	MOV A, direct	A ← (direct)
ตัวอย่าง	MOV A, 01H	ผลที่ได้รีจิสเตอร์A = 23H
คำสั่ง	MOV A,# data	A ← #data
ตัวอย่าง	MOV A, #40H	ผลที่ได้รีจิสเตอร์A = 40H
คำสั่ง	MOV A,@ Ri	(A) ← ((Ri))
ตัวอย่าง	MOV R1,#23H	
	MOV A,@R1	ผลที่ได้รีจิสเตอร์A = 1DH

กำหนดให้รีจิสเตอร์ A = 06H ตำแหน่งหน่วยความจำข้อมูลแอดเดรสที่ 06H = 23H

คำสั่ง MOV Rn, A Rn ← (A)

ตัวอย่าง MOV R1, A ผลที่ได้รีจิสเตอร์ R1 = 06H

คำสั่ง MOV Rn, direct Rn← (direct)

ตัวอย่าง MOV R1, 06H ผลที่ได้รีจิสเตอร์ R1 = 23H

คำสั่ง MOV Rn,# data (R1)← #data

ตัวอย่าง MOV R1, #40H ผลที่ได้รีจิสเตอร์ = 40H

1.2 กลุ่มคำสั่งเคลื่อนย้ายข้อมูลระหว่างหน่วยความจำภายใน และภายนอกไอซี (Data Transfer)

ไอซี MCS-51 สามารถติดต่อกับหน่วยความจำข้อมูลภายนอก และหน่วยความจำโปรแกรมภายนอก ได้ถึง 64 กิโลไบต์โดยใช้ขาแอดเดรสจำนวน 16 เส้น (0000H –FFFFH) ไม่สามารถทำโดยตรงได้ แต่ใช้รีจิสเตอร์ DPTR หรือรีจิสเตอร์ PC ที่มีขนาด 16 บิตใช้งานร่วมกับรีจิสเตอร์ A เพื่อทำการเคลื่อนย้ายโดยทางอ้อม แบ่งกลุ่มคำสั่งได้ดังนี้

1.2.1 กลุ่มคำสั่งการเคลื่อนย้ายข้อมูลภายในกับหน่วยความจำข้อมูลภายนอก

MOVX A,@Ri ย้ายข้อมูลจากหน่วยความจำที่เก็บอยู่ใน Ri ไปยัง A

MOVX @Ri,A ย้ายข้อมูลที่เก็บอยู่ใน A ไปยังหน่วยความจำที่เก็บอยู่ใน Ri

MOVX A,@DPTR ย้ายข้อมูลจากหน่วยความจำที่เก็บอยู่ใน DPTR ไปยัง A

MOVX @DPTR,A ย้ายข้อมูลที่อยู่ใน A ไปยังหน่วยความจำที่เก็บอยู่ใน DPTR

กำหนดให้รีจิสเตอร์ A = 06H ตำแหน่งหน่วยความจำข้อมูลแอดเดรสที่ 30H = 23H

คำสั่ง MOVX A,@Ri MOVX (A) ← ((Ri)) ใช้ได้กับริจิสเตอร์ R0, R1

ตัวอย่าง MOV R1, #30H ผลที่ได้ R1 = 30H

MOVX A,@R1 ผลที่ได้รีจิสเตอร์ A มีค่าเท่ากับ 23H

คำสั่ง MOVX @Ri, A ((Ri)) ← MOVX (A)

ตัวอย่าง MOV R1, #30H ผลที่ได้ R1 = 30H

MOVX @R1, A ข้อมูลในแอดเดรส 30H มีค่าเท่ากับรีจิสเตอร์ A

คำสั่ง MOVX A,@DPTR MOVX (A) ← ((DPTR))

ตัวอย่าง MOV DPTR, #8000H ผลที่ได้รีจิสเตอร์ DPTR = 8000H

MOVX A,@DPTR นำข้อมูลที่หน่วยความจำข้อมูลภายนอกแอดเดรสที่ 8000H นำไปเก็บไว้ที่รีจิสเตอร์ A

คำสั่ง MOVX @DPTR,A MOVX((DPTR)) ← (A)

ตัวอย่าง MOV DPTR, #8000H ผลที่ได้รีจิสเตอร์ DPTR = 8000H

MOVX @DPTR, A นำค่าในรีจิสเตอร์ A เก็บไว้ที่หน่วยความจำข้อมูลภายนอกตำแหน่งแอดเดรสที่ 8000H

1.2.2 กลุ่มคำสั่งเคลื่อนย้ายข้อมูลกับหน่วยความจำโปรแกรมภายนอก

MOVC A,@A+DPTR ย้ายข้อมูลจากหน่วยความจำข้อมูลที่สำพัทธ์กับ DPTR ไป A

MOVC A,@A+PC ย้ายข้อมูลจากหน่วยความจำข้อมูลที่สำพัทธ์กับ PC ไปยัง A

คำสั่ง MOVC A,@A+ADPTR MOVX (A) ← ((A) + (DPTR))

ตัวอย่าง MOV DPTR, #8000H ผลที่ได้ DPTR = 8000H

MOVX A, #04H กำหนดค่าในรีจิสเตอร์ A

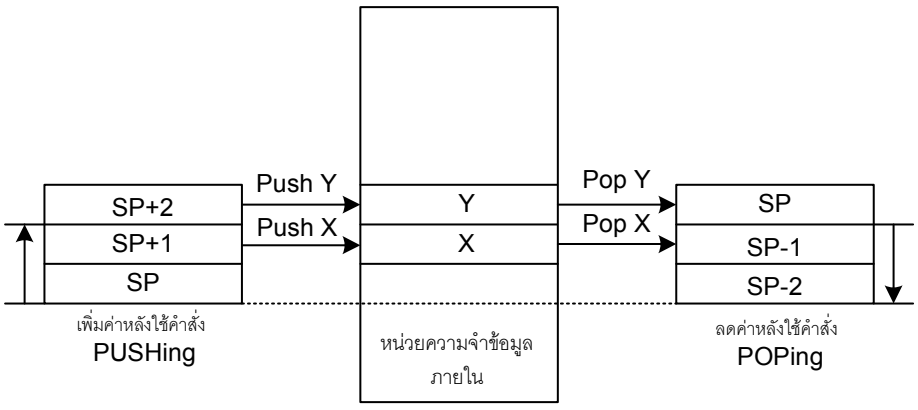
MOVX A, @A+DPTR นำข้อมูลจากหน่วยความจำโปรแกรมภายนอกตำแหน่งที่ A รวมกับค่าใน DPTR ไปเก็บไว้ที่รีจิสเตอร์ A รีจิสเตอร์ DPTR = 8004H

คำสั่ง MOVC A,@A+PC MOVX (A) ← ((A) + (PC)) นำข้อมูลจากหน่วยความจำโปรแกรมภายนอกตำแหน่งที่โปรแกรมเคาน์เตอร์ชี้รวมกับค่าในรีจิสเตอร์ A ไปเก็บไว้ที่รีจิสเตอร์ A รีจิสเตอร์ PC เก็บค่าแอดเดรสที่ไอซี MCS-51 ทำปัจจุบันบวกอีกหนึ่งตำแหน่ง PC = PC+1

1.3 กลุ่มคำสั่งเคลื่อนย้ายข้อมูลกับสแตก

ไอซี MCS-51 เริ่มต้นทำงานที่โปรแกรมหลัก ส่วนสแตกมีตำแหน่งเมื่อไอซี MCS-51 ถูกรีเซตจะอยู่ที่แอดเดรส 07H เมื่อเรียกใช้โปรแกรมย่อย หรือกระโดดไปทำงานในแอดเดรสอื่นไอซี MCS-51 จัดเก็บค่าโปรแกรมเคาน์เตอร์ หรือข้อมูลที่ต้องการไว้ในสแตก เรียกว่าการ PUSH โดยเริ่มเก็บค่าไว้ที่ SP+1 ของหน่วยความจำข้อมูลภายใน เมื่อไอซีต้องการกลับจากโปรแกรมย่อยเข้าสู่โปรแกรมหลักต้องคืนค่าของโปรแกรมเคาน์เตอร์ ที่ในสแตกโดยการ POP โดยจะลดค่าลง SP-1 แสดงดังภาพที่ 4.1

PUSH	direct	ย้ายข้อมูลจากหน่วยความจำ direct ไปเก็บยัง Stack
POP	direct	ย้ายข้อมูลจาก Stack ไปยังหน่วยความจำ direct



ภาพที่ 4.1 การทำงานของคำสั่ง PUSH และ POP

1.4 กลุ่มคำสั่งแลกเปลี่ยนข้อมูล

เป็นการสลับค่าของข้อมูลระหว่างต้นทางกับปลายทาง มีรายละเอียดดังนี้

XCH	A,Rn	แลกเปลี่ยนข้อมูลระหว่าง A กับ Rn
XCH	A,direct	แลกเปลี่ยนข้อมูลระหว่างหน่วยความจำ Direct กับ A
XCH	A,@Ri	แลกเปลี่ยนข้อมูลระหว่างหน่วยความจำที่เก็บอยู่ใน Ri กับ A
XCHD	A,@Ri	แลกเปลี่ยนข้อมูลสี่บิตล่างจากหน่วยความจำที่เก็บใน Ri กับ A

2. กลุ่มคำสั่งทางคณิตศาสตร์

กลุ่มคำสั่งทางคณิตศาสตร์(Arithmetic Instructions)ในไอซี MCS-51 ประกอบด้วยกลุ่มคำสั่งที่กระทำทางคณิตศาสตร์ดังต่อไปนี้

2.1 กลุ่มคำสั่งการบวก

เป็นคำสั่งบวกค่าข้อมูลโดยรีจิสเตอร์ A ทำหน้าที่เก็บค่าผลลัพธ์ปลายทางไว้ คำสั่งการบวกข้อมูลมีผลต่อ แฟลกในรีจิสเตอร์ PSW

การบวกเลขแบบไม่คิดเครื่องหมาย คือเลขที่มีค่าบวกทั้งหมดไม่มีค่าลบ เช่น 00H-FFH ในเลขฐานสิบหก เท่ากับ 000-256 แบบเลขฐานสิบ ส่วนแบบคิดเครื่องหมายใช้บิตสูงสุด (MSB) เป็นบิตแสดงเครื่องหมายเช่นเลข 00H-FFH ในเลขฐานสิบหก เท่ากับ -128 ถึง +127 อย่างละ 128 ค่า โดยการตรวจสอบค่าที่บิต D7 ถ้า D7 = 1 เป็นค่าลบ D7= 0 เป็นค่าบวก การบวกเลขแบบไม่คิดเครื่องหมายหากผลลัพธ์เกินค่า 255 ไม่สามารถบอกข้อมูลที่บวกได้ ดังนั้นจึงต้องมีบิต เพื่อบอกสถานะของตัวทด ตัวอย่างเช่น

	120 +		01111000 +	
	60		00111100	ผลลัพธ์ไม่เกิน 255
=	180	=	10110100	Carry Flag เคลียร์

---

	120 +		01111000 +	
	180		10110100	ผลลัพธ์เกิน 255
=	300	=	100101100	Carry Flag เซต

การบวกเลขแบบคิดเครื่องหมาย หากเลขที่นำมาบวกมีเครื่องหมายเดียวกัน หรือมีเครื่องหมายต่างกัน ผลลัพธ์ที่ได้แทนด้วยเลขฐานสอง ในกรณีนี้บิตนับเกิน(Overflow Flag) ถูกเคลียร์เสมอ แต่บิตทด (Carry Flag) อาจถูกเซตหรือเคลียร์ ขึ้นอยู่กับค่าของเลขที่นำมาบวกกันดังตัวอย่าง

120 + 0111 1000 +

-40 1101 1000 carry flag เซต

= 80 = 1 0101 0000 Overflow flag เคลียร์

2.1.1 คำสั่งการบวกแบบไม่รวมตัวทด

กำหนดให้ค่าข้อมูลในแอดเดรส และรีจิสเตอร์ต่างๆดังนี้ Reg A = 67H, Rn =R7=32H Direct = 30H, Ri = R0 =30H, Data = 20H, Adders 30H = 08H

คำสั่ง ADD A,Rn บวกค่าข้อมูลในรีจิสเตอร์ Rn กับข้อมูลในรีจิสเตอร์ A

ตัวอย่าง ADD A,R7 67H + 32H รีจิสเตอร์ A มีค่าเท่ากับ 99H

คำสั่ง ADD A,direct บวกค่าข้อมูลในหน่วยความจำ direct กับรีจิสเตอร์ A

ตัวอย่าง ADD A,30H 67H + 08H รีจิสเตอร์ A มีค่าเท่ากับ 6FH

คำสั่ง ADD A,@RI บวกค่าหน่วยความจำในรีจิสเตอร์A กับค่าที่ถูกชี้โดย Ri

ตัวอย่าง ADD A,@R0 67H + 08H รีจิสเตอร์ A มีค่าเท่ากับ 6FH

คำสั่ง ADD A,#data บวกค่าคงที่ 8 บิตกับข้อมูลในรีจิสเตอร์ A

ตัวอย่าง ADD A,#20H 67H + 20H รีจิสเตอร์ A มีค่าเท่ากับ 87H

2.1.2 คำสั่งการบวกแบบรวมตัวทด

กำหนดให้ค่าข้อมูลในแอดเดรส และรีจิสเตอร์ต่างๆดังนี้ Reg A = 08H, Rn =R7=03H Direct = 30H, Ri = R0 =30H, Data = AFH, Adders 30H = 62H, C=1

คำสั่ง ADDC A,Rn บวกค่ารีจิสเตอร์ Rn กับข้อมูลใน Reg A พร้อมแฟลกทด

คำสั่ง ADDC A,@RI บวกค่าในรีจิสเตอร์A กับค่าที่ถูกชี้โดย Ri พร้อมแฟลกทด

ตัวอย่าง ADDC A,@R0 08H + 62H รีจิสเตอร์ A มีค่าเท่ากับ 6BH

ตัวอย่าง ADDC A, R7 08H + 03H รีจิสเตอร์ A มีค่าเท่ากับ 0CH

คำสั่ง ADDC A,direct บวกค่าในหน่วยความจำ direct กับค่าในรีจิสเตอร์ A พร้อมแฟลกทด

ตัวอย่าง ADDC A,30H 08H + 62H รีจิสเตอร์ A มีค่าเท่ากับ 6BH

คำสั่ง ADDC A, data บวกค่าคงที่ 8 บิตกับค่าในรีจิสเตอร์ A พร้อมแฟลกทด

ตัวอย่าง ADDC A, #80H 08H + 62H รีจิสเตอร์ A มีค่าเท่ากับ 89H

2.2 กลุ่มคำสั่งการลบ

เป็นคำสั่งการลบค่าข้อมูล การลบแบบคิดตัวยืม ถ้ากำหนดค่าข้อมูลในแอดเดรส และรีจิสเตอร์ต่างๆดังนี้ Reg A = 20H, Rn =R7=02H, Direct = 30H, Ri = R0 =30H, Data = AFH, Adders 30H = 10H

คำสั่ง SUBB A,Rn ลบค่าข้อมูลใน Rn กับข้อมูลในรีจิสเตอร์ A พร้อมแฟลกตัวยืม

ตัวอย่าง SUBB A,R7 กำหนด **C = 1** 20H (A) - 02H -1(C) รีจิสเตอร์ A เท่ากับ 1DH

ตัวอย่าง SUBB A,R7 กำหนด **C** = 0 20H (A) - 02H -0(C) รีจิสเตอร์ A เท่ากับ 1EH

คำสั่ง SUBB A,direct ลบข้อมูลใน direct กับข้อมูลในรีจิสเตอร์ Aพร้อมแฟลก

ตัวอย่าง SUBB A,30H กำหนด **C** = 1 20H (A) - 10H -1(C) รีจิสเตอร์ A เท่ากับ 0FH

ตัวอย่าง SUBB A,30H กำหนด **C** = 0 20H (A) - 10H -0(C) รีจิสเตอร์ A เท่ากับ 10H

คำสั่ง SUBB A,@RI ลบค่าในรีจิสเตอร์A กับค่าที่ถูกชี้โดย Ri พร้อมแฟลกตัวยืม

ตัวอย่าง SUBB A,@R0 กำหนด **C** = 1 20H (A) - 10H -1(C) รีจิสเตอร์ A เท่ากับ 0FH

ตัวอย่าง SUBB A,@R0 กำหนด **C** = 0 20H (A) - 10H -0(C) รีจิสเตอร์ A เท่ากับ 10H

คำสั่ง SUBB A,#data ลบค่าคงที่ 8 บิตกับข้อมูลในรีจิสเตอร์ A พร้อมแฟลกตัวยืม

ตัวอย่าง SUBB A,#08H กำหนด **C** = 1 20H (A) - 08H -1(C) รีจิสเตอร์ A เท่ากับ 17H

ตัวอย่าง SUBB A,#08H กำหนด **C** = 0 20H (A) - 08H -0(C) รีจิสเตอร์ A เท่ากับ18H

2.3 กลุ่มคำสั่งการเพิ่มค่า 1 ค่า

เป็นคำสั่งที่ใช้เพิ่มค่าข้อมูลในหน่วยความจำที่กำหนดจำนวน 1 ค่า หรือข้อมูล +1 กำหนดให้ Reg A = 67H, Rn =R7=32H, Direct = 30H, Ri = R0 =30H, Data = AFH, Adders 30H = 98H

INC A เพิ่มค่าข้อมูลในรีจิสเตอร์ A = A+1 ดังนั้น Reg A มีค่าเท่ากับ 68H

INC Rn เพิ่มค่าข้อมูลในรีจิสเตอร์ Rn = RN+1 ดังนั้น Reg R7 มีค่าเท่ากับ 68H

INC direct เพิ่มค่าข้อมูลในหน่วยความจำ direct 1ค่า ดังนั้น 30H มีค่าเท่ากับ 31H

INC @ RI เพิ่มค่าข้อมูลในหน่วยความจำที่ชี้แอดเดรสโดยข้อมูลใน Ri จากข้อมูลที่กำหนด Reg R0 มีค่าเท่ากับ 30H ดังนั้น ที่ตำแหน่งแอดเดรส 30H มีข้อมูล 99H

INC DPTR เพิ่มค่าข้อมูลในรีจิสเตอร์ DPTR = DPTR+1 ดังนั้น DPTR มีค่าเพิ่มอีก 1 ค่า

2.4 กลุ่มคำสั่งการลดค่า 1 ค่า

เป็นคำสั่งที่ใช้ลดค่าข้อมูลในหน่วยความจำที่กำหนด 1 ค่า หรือข้อมูล -1 ถ้ากำหนดให้ Reg A = 67H, Rn=R7=32H, Direct = 30H, Ri = R0 =30H, Data=AFH, 30H = 98H

DEC A ลดค่าข้อมูลในรีจิสเตอร์ A = A-1 ดังนั้น Reg A มีค่าเท่ากับ 68H

DEC Rn ลดค่าข้อมูลในรีจิสเตอร์ Rn = RN-1 ดังนั้น Reg A มีค่าเท่ากับ 68H

DEC direct ลดค่าข้อมูลในหน่วยความจำ direct ดังนั้นข้อมูลที่ 30H มีค่าเท่ากับ 31H

DEC @Ri ลดค่าข้อมูลในหน่วยความจำที่ชี้แอดเดรสโดยข้อมูลใน Ri จากข้อมูลที่กำหนด Reg R0 มีค่าเท่ากับ 30H ดังนั้น ที่ตำแหน่งแอดเดรส 30H มีข้อมูล 99H

2.5 คำสั่งการคูณ

MUL AB เป็นคำสั่งคูณข้อมูลในรีจิสเตอร์ A กับข้อมูลในรีจิสเตอร์ B แล้วเก็บค่าข้อมูลไบต์ล่างไว้ในรีจิสเตอร์ A และข้อมูลไบต์บนไว้ในรีจิสเตอร์ B ถ้าข้อมูลเกินค่า FFH แฟลก OV ถูกเซต

ตัวอย่าง กำหนดให้รีจิสเตอร์ A =25H, B=30H

หลังจากใช้คำสั่ง **MUL AB** ได้ผลลัพธ์เท่ากับ 6F0H ข้อมูลในรีจิสเตอร์ A=F0H ข้อมูลในรีจีสเตอร์ B = 06H บิต OV ถูกเซตเป็น “1”

2.6 คำสั่งการหาร

DIV AB เป็นคำสั่งในการหารค่าข้อมูลในรีจิสเตอร์ A ด้วยค่าข้อมูลที่รีจิสเตอร์ B แล้วนำผลลัพธ์จากการหารเก็บไว้ที่รีจิสเตอร์ A ส่วนเศษของการหารเก็บไว้ที่รีจิสเตอร์ B

ตัวอย่าง กำหนดให้รีจิสเตอร์ ACC = 79_{10} และกำหนดค่าในรีจิสเตอร์ B = 10_{10}

หลังจากใช้คำสั่ง DIV AB ได้ค่าข้อมูลในรีจิสเตอร์ ACC = 07H และในรีจิสเตอร์ B = 09H

2.7 คำสั่งการปรับค่าเป็นเลขฐานสิบ

DA A (Decimal-Adjust Accumulator for Addition) คือการปรับค่าจากเลข Binary ให้เป็นเลข BCD หลังจากใช้คำสั่ง ADD หรือ ADDC (รหัส BCD เขียนได้ตั้งแต่ 0000 – 1001) การปรับค่าเป็น BCD ถูกเรียกใช้โดยคำสั่ง DA A และทำที่รีจิสเตอร์ A เท่านั้นมีขั้นตอนดังนี้

ตรวจสอบ 4 บิตต่ำ ถ้ามากกว่า 9 หรือแฟลก AC ถูกเซตเป็น “1” ให้บวกค่าใน รีจิสเตอร์ A ด้วยค่า 06H

ตรวจสอบ 4 บิตสูง ถ้ามากกว่า 9 หรือแฟลก C ถูกเซตเป็น “1” ให้บวกค่าใน รีจิสเตอร์ A ด้วยค่า 60H

ตัวอย่าง

ตัวตั้ง		0011	1001	เขียนเป็นเลขฐานสิบหกได้ 39h
ตัวบวก	+	0001	0001	เขียนเป็นเลขฐานสิบหกได้ 11h
ผลบวกเก็บไว้ในรีจิสเตอร์ A =		0100	1010	เขียนเป็นเลขฐานสิบหกได้ 4Ah
คำสั่ง DA A ตามเงื่อนไข +06h	+	0000	0110	เขียนเป็นเลขฐานสิบหกได้ 06h
ผลบวกหลังคำสั่ง DA A A =		0101	0000	เขียนเป็นเลขฐานสิบหกได้ 50h

3. กลุ่มคำสั่งทางตรรกะ

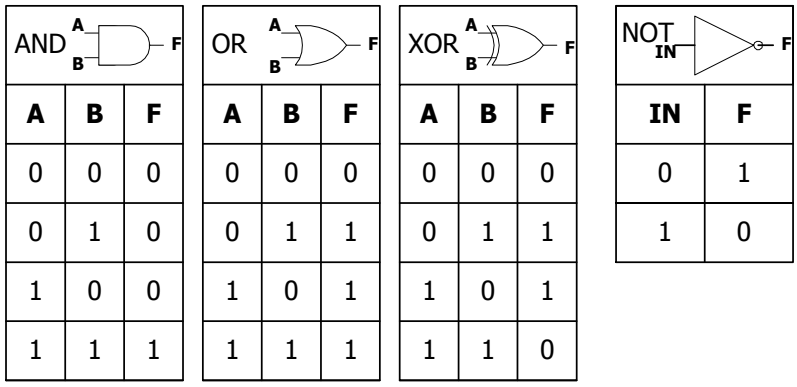
คำสั่งทางตรรกะ (Logical Instruction) เป็นกลุ่มคำสั่งที่ทำหน้าที่เกี่ยวกับการ AND, OR, XOR NOT แสดงตารางความจริงได้ แสดงดังภาพที่ 4.2 รวมถึงการหมุนข้อมูล โดยกระทำทางลอจิกครั้งละ 8 บิต และทำบิตต่อบิตที่ตรงกัน

AND บิตที่กระทำลอจิกแอนด์ (AND) ต้องมีลอจิก”1” ทั้งคู่ จึงได้ผลลัพธ์จากการแอนด์เป็นลอจิก “1” นอกเหนือจากนี้ให้ผลลัพธ์จากการแอนด์เป็นลอจิก “0”

OR บิตที่กระทำลอจิกออร์ (OR) ต้องมีลอจิกเป็น”0” ทั้งคู่ จึงได้ผลลัพธ์จากการออร์เป็นลอจิก “0” นอกเหนือจากนี้ให้ผลลัพธ์จากการออร์ เป็นลอจิก “1”

XOR บิตที่กระทำลอจิกเอ็กคูซีฟออร์ (Exclusive - OR) ต้องมีลอจิกที่เหมือนกัน จึงทำให้ได้ผลลัพธ์จากการ XORเป็นลอจิก “0” นอกเหนือจากนี้ให้ผลลัพธ์จากการ XOR เป็นลอจิก “1”

NOT ผลลัพธ์ที่ได้เป็นการกลับสถานะของบิตนั้นๆ ให้เป็นตรงกันข้าม CPL (Complement)



A	B	F
0	0	0
0	1	0
1	0	0
1	1	1

A	B	F
0	0	0
0	1	1
1	0	1
1	1	1

A	B	F
0	0	0
0	1	1
1	0	1
1	1	0

IN	F
0	1
1	0

ภาพที่ 4.2 เกตแบบ AND, OR, XOR และ NOT

3.1 กลุ่มคำสั่งการ AND

ANL A,Rn AND ค่าข้อมูลในรีจิสเตอร์ Rn กับค่าข้อมูลรีจิสเตอร์ใน A

ANL A,direct AND ค่าข้อมูลในหน่วยความจำ direct กับค่าข้อมูลรีจิสเตอร์ใน A

ANL A,@Ri AND ค่าข้อมูลที่ชี้แอดเดรสโดย Ri กับค่าข้อมูลรีจิสเตอร์ใน A

ANL A,#data AND ค่าคงที่ 8 บิตกับค่าข้อมูลรีจิสเตอร์ A

ANL direct,A AND ค่าข้อมูลรีจิสเตอร์ใน A กับข้อมูลในหน่วยความจำ direct

ANL direct,#data AND ค่าคงที่ 8 บิตกับหน่วยความจำ direct

ตัวอย่าง ก่อนกระทำสั่ง A = 78H

```
ANL     A, #0FH     ; ทำการแอนด์ค่าคงที่ 0FH กับค่าข้อมูล 78H ในรีจิสเตอร์ A
```

หลังกระทำคำสั่ง A = 08H เป็นการเคลียร์ข้อมูล 4 บิตบน ให้เป็นลอจิก “0” หรือ การ กรองข้อมูล 4 บิตล่าง

ค่าข้อมูลในรีจิสเตอร์ A ก่อนทำคำสั่ง **78H**	0	1	1	1	1	0	0	0
ค่าคงที่ **0FH**	0	0	0	0	1	1	1	1
ผลลัพธ์เก็บไว้ที่รีจิสเตอร์ A **08H**	0	0	0	0	1	0	0	0

ตัวอย่าง ก่อนกระทำคำสั่ง A=78H

```
ANL   A, #0F0H    ;  ทำการแอนด์ค่า F0H  กับค่า 78H  ในรีจิสเตอร์ A  หลังกระทำคำสั่ง
```

A=70H เป็นการเคลียร์ข้อมูล 4 บิตล่างให้เป็นลอจิก “0” หรือการกรองข้อมูล 4 บิตบน

3.2 กลุ่มคำสั่งการ ORL

ORL A,Rn OR ค่าข้อมูลในรีจิสเตอร์ Rn กับค่าข้อมูลรีจิสเตอร์ใน A

ORL A,direct OR ค่าข้อมูลในหน่วยความจำ direct กับค่าข้อมูลรีจิสเตอร์ใน A

ORL A,@Ri OR ค่าข้อมูลที่ชี้แอดเดรสโดย Ri กับค่าข้อมูลรีจิสเตอร์ใน A

ORL A,#data OR ค่าคงที่ 8 บิตกับค่าข้อมูลรีจิสเตอร์ใน A

ORL direct,A OR ค่าข้อมูลรีจิสเตอร์ใน A กับหน่วยความจำ direct

ORL direct,#data OR ค่าคงที่ 8 บิตกับหน่วยความจำ direct

ตัวอย่าง ก่อนกระทำคำสั่งให้รีจิสเตอร์ A มีค่าข้อมูลเท่ากับ 0BH

ORL A, #0DFH ; ทำการออร์ค่า DFH กับค่า 0BH ในจิสเตอร์ A หลังกระทำคำสั่ง A = DFH เป็นการเซตข้อมูล 4 บิตล่างให้เป็น “1” ผลลัพธ์เก็บไว้ในรีจิสเตอร์ A

ค่าข้อมูลในรีจิสเตอร์ A ก่อนทำคำสั่ง **0BH**	0	0	0	0	1	0	1	1
ค่าคงที่ **DFH**	1	0	1	1	1	1	1	1
ผลลัพธ์เก็บไว้ที่รีจิสเตอร์ A **DFH**	1	0	1	1	1	1	1	1

3.3 กลุ่มคำสั่งการ XRL

XRL A,Rn ค่าข้อมูลในรีจิสเตอร์ Rn กับค่าข้อมูลรีจิสเตอร์ใน A

XRL A,direct EX-OR ค่าข้อมูลในหน่วยความจำ direct กับค่าข้อมูลรีจิสเตอร์ใน A

XRL A,@Ri EX-OR ค่าข้อมูลที่ชี้แอดเดรสโดย Ri กับค่าข้อมูลรีจิสเตอร์ใน A

XRL A,#data EX-OR ค่าคงที่ 8 บิตกับค่ามูลรีจิสเตอร์ใน A

XRL direct,A EX-OR ค่าข้อมูลรีจิสเตอร์ใน A กับหน่วยความจำ direct

XRL direct,#data EX-ORค่าคงที่ 8 บิตกับหน่วยความจำ direct

ตัวอย่าง ก่อนกระทำคำสั่ง A มีค่าข้อมูลเท่ากับ 8FH

XRL A, #0DFH ; ทำการเอกคูซีฟออร์ค่า DFH กับค่า 8FH ในจิสเตอร์ A

หลังกระทำคำสั่ง A = 50H เพราะบิตที่เหมือนกัน เป็นลอจิก “0” และต่างกันเป็น “1” ผลลัพธ์เก็บไว้ในรีจิสเตอร์ A

ค่าข้อมูลในรีจิสเตอร์ A ก่อนทำคำสั่ง **8FH**	1	0	0	0	1	1	1	1
ค่าคงที่ **DFH**	1	1	0	1	1	1	1	1
ผลลัพธ์เก็บไว้ที่รีจิสเตอร์ A **50H**	0	1	0	1	0	0	0	0

3.4 กลุ่มคำสั่งการ CPL

CPL A กลับค่าบิตข้อมูลในรีจิสเตอร์ A เป็นตรงข้ามทุกบิต

CPL (Complement) ทำการกลับสถานะข้อมูลในบิตที่กำหนด หรือในรีจิสเตอร์ A

ตัวอย่าง ก่อนทำคำสั่งสถานะของพอร์ต P1.5 = “1” รีจิสเตอร์ A มีค่าเท่ากับ 01010101B และ CARRY FLAG =”0”

CPL P1.5 (Complement bit) หลังทำคำสั่งพอร์ต P1.5 มีสถานะลอจิก = “0”

CPL A (Complement Accumulator)

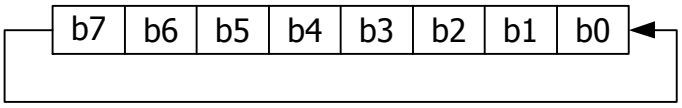
หลังทำคำสั่งรีจิสเตอร์ A มีข้อมูลเท่ากับ = 10101010B

CPL C (Complement Carry Flag) หลังทำคำสั่ง Carry Flag มีสถานะลอจิก “1”

3.5 กลุ่มคำสั่งการหมุนข้อมูล

การหมุนข้อมูลในคำสั่ง (Rotate) จะใช้สัญลักษณ์เป็นตัวอักษร “R” (Rotate) และถ้าการหมุนไปทางซ้ายใช้สัญลักษณ์เป็นตัวอักษร “L” (LEFT) ถ้าหมุนไปทางขวาใช้สัญลักษณ์ “R” (RIGHT) ส่วนตัวอักษร “C” (Carry Flag) เป็นการกระทำร่วมกับแฟลกตัวทด (Carry Flag) และกระทำที่รีจิสเตอร์ A เท่านั้น มีด้วยกัน 4 คำสั่งด้วยกันคือ

RL A (Rotate Accumulator Left)



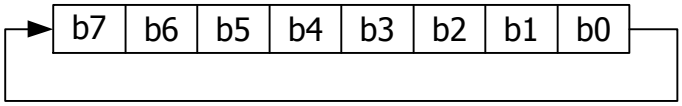
หมุนข้อมูลในแต่ละบิตของรีจิสเตอร์ A วนไปทางซ้าย และบิตที่ 7 หมุนวนกลับมายังบิตที่ 0

ตัวอย่าง ก่อนกระทำคำสั่ง A=0111 1101B หรือ 7DH

หลังกระทำคำสั่ง1 A=1111 1010B หรือ FCH

หลังกระทำคำสั่ง2 A=1111 0101B หรือ F5H

RR A (Rotate Accumulator Right)



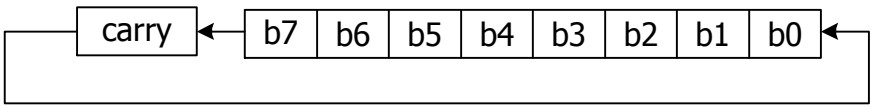
หมุนข้อมูลในแต่ละบิตของรีจิสเตอร์ A วนมาทางขวา และบิตที่ 0 วนกลับมาแทนบิตที่ 7

ตัวอย่าง ก่อนกระทำคำสั่ง A=0111 1110B หรือ 7EH

หลังกระทำคำสั่ง1 A=0011 1111B หรือ 2FH

หลังกระทำคำสั่ง2 A=1001 1111B หรือ 9FH

RLC A (Rotate Accumulator Left Through The Carry Flag)



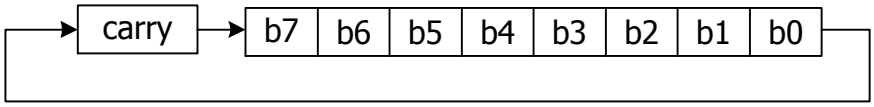
หมุนข้อมูลในแต่ละบิตของรีจิสเตอร์ A วนทางซ้ายผ่านแฟลกทด โดยบิตที่ 7 หมุนไปยังแฟลกทด และข้อมูลของแฟลกทดเดิมหมุนเข้ามาในบิตที่ 0

ตัวอย่าง ก่อนกระทำคำสั่ง A= 0100 0000B หรือ 00H และ C=1

หลังกระทำคำสั่ง1 A= 1000 0001B หรือ 01H และ C=0

หลังกระทำคำสั่ง2 A= 0000 0010B หรือ 02H และ C=1

RRC A (Rotate Accumulator Right Through The Carry Flag)



หมุนข้อมูลในแต่ละบิตของรีจิสเตอร์ A ให้วนทางขวาผ่านแฟลกทด โดยบิต 0 จะหมุนไปยังแฟลกทด และข้อมูลของแฟลกทดเดิมหมุนเข้ามาในบิต 7

ตัวอย่าง ก่อนกระทำคำสั่ง A= 0000 0010B หรือ 00H และ C=1

หลังกระทำคำสั่ง1 A= 1000 0001B หรือ 80H และ C=0

หลังกระทำคำสั่ง2 A= 0100 0000B หรือ 40H และ C=1

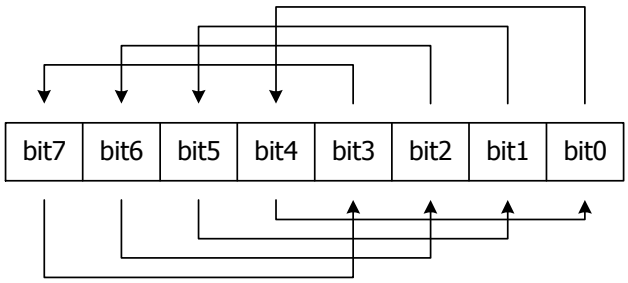
3.6 คำสั่งการเคลียร์ข้อมูล

CLR A (Clear Register A) เป็นการทำให้ค่าข้อมูลในรีจิสเตอร์ A เป็น 00000000B ใช้ได้กับรีจิสเตอร์ A เท่านั้น

3.7 คำสั่งการสลับข้อมูล

SWAP A (Swap Nibbles Within The Accumulator) คือการแลกเปลี่ยนข้อมูลระหว่างบิต 0-3 กับบิต 7-4 หรือเป็นการสลับค่าระหว่าง 4 บิตสูง กับ 4 บิตต่ำของรีจิสเตอร์ A แสดงดังภาพที่ 4.3

ตัวอย่าง SWAP A เดิมรีจิสเตอร์ A = 75H หลังการทำคำสั่ง A = 57H



ภาพที่ 4.3 การแลกเปลี่ยนข้อมูลโดยการสลับค่าระหว่าง 4 บิตสูง กับ 4 บิตต่ำ

4. กลุ่มคำสั่งจัดการแบบบูลีน

คำสั่งจัดการแบบบูลีน (Boolean Variable Manipulation) เป็นการทำในระดับบิต สามารถกระทำได้กับรีจิสเตอร์ใช้งานพิเศษบางตัว แสดงดังตารางที่ 4.1 และหน่วยความจำภายในที่เข้าถึงในระดับบิตตั้งแต่แอดเดรส 20H -2FH

ตารางที่ 4.1 แสดงตำแหน่งบิตของรีจิสเตอร์ใช้งานพิเศษ ที่เข้าถึงข้อมูลแบบบิตได้

SFR	Direct Address (HEX)	Bit Addresses (HEX) MSB LSB							
		D7	D6	D5	D4	D3	D2	D1	D0
A	E0	E7	E6	E5	E4	E3	E2	E1	E0
B	F0	F7	F6	F5	F4	F3	F2	F1	F0
IE	A8	AF	AE	AD	AC	AB	AA	A9	A8
IP	B8	BF	BE	BD	BC	BB	BA	B9	B8
P0	80	87	86	85	84	83	82	81	80
P1	90	97	96	95	94	93	92	91	90
P2	A0	A7	A6	A5	A4	A3	A2	A1	A0
P3	B0	B7	B6	B5	B4	B3	B2	B1	B0
PSW	D0	D7	D6	D5	D4	D3	D2	D1	D0
TCON	88	8F	8E	8D	8C	8B	8A	89	88
SCON	98	9F	9E	9D	9C	9B	9A	99	98

มีคำสั่งดังต่อไปนี้

CLR	C	ทำให้ค่าแฟลกทดเป็น 0
CLR	bit	กำหนดให้บิตเป็น 0 เช่น CLR P1.7
SETB	C	กำหนดค่าในแฟลกทดให้เป็น 1
SETB	bit	ให้บิตที่กำหนดเป็น 1 เช่น SETB P1.7
CPL	C	กลับค่าบิตทดให้เป็นตรงกับข้าม
CPL	bit	กลับค่าบิตให้เป็นตรงกับข้าม
ANL	C,bit	AND ค่า bit กับแฟลกทด
ANL	C,/bit	AND ค่าตรงข้ามของ bit กับแฟลกทด
ORL	C,bit	OR ค่าตรงข้ามของ bit กับแฟลกทด
ORL	C,/bit	OR ค่า bit กับแฟลกทด
MOV	C,bit	ย้ายค่า bit มายังแฟลกทด
MOV	bit,C	ย้ายค่าบิตทด มายัง bit

5. กลุ่มคำสั่งควบคุมการทำงานโปรแกรม

คำสั่งควบคุมการทำงานโปรแกรม (Program and Machine Control) เป็นกลุ่มที่ใช้ในการเปลี่ยนแปลงตำแหน่งแอดเดรสการทำงานของซีพียู เนื่องจากในบางครั้งซีพียูมีความจำเป็นต้องกระโดดไปทำงานที่แอดเดรสอื่นๆ การกระโดดแบ่งได้ดังนี้

5.1 คำสั่งกระโดดแบบไม่มีเงื่อนไข

เป็นการกระโดดไปทำงานในตำแหน่งแอดเดรสที่ระบุตามคำสั่งทันทีทันใด มีคำสั่งดังต่อไปนี้

JMP addr 11 กระโดดไปยังตำแหน่งต่างๆ โดยกำหนดได้จากค่าแอดเดรส 11 บิต หรือกระโดดได้ถึง 2 กิโลไบต์ จากที่โปรแกรมเคาน์เตอร์ทำงาน

LJMP addr 16 กระโดดไปยังตำแหน่งต่างๆ โดยกำหนดได้จากค่าแอดเดรส 16 บิต หรือกระโดดได้ถึง 64 กิโลไบต์ จากที่โปรแกรมเคาน์เตอร์ทำงาน

SJMP rel กระโดดไปตำแหน่งต่างๆ ได้แบบสัมพัทธ์ (Relative Range) ที่แอดเดรสกำหนดระยะไม่เกิน–127d +128d

JMP @A+DPTR กระโดดไปยังตำแหน่งสัมพัทธ์กับค่าข้อมูลในรีจิสเตอร์ A บวกกับค่าใน DPTR

Nop เป็นคำสั่งที่ไม่มีการทำงานใดๆเกิดขึ้นแต่ซีพียูยังทำงานปกติ

5.2 คำสั่งกระโดดแบบมีเงื่อนไข

การกระโดดแบบมีเงื่อนไข (Conditional Jumps) เป็นการกระโดดไปทำงานในตำแหน่งแอดเดรสที่ระบุไว้โดยตรวจสอบเงื่อนไขว่าเป็นจริง หรือเป็นเท็จจากการกำหนดที่คำสั่งดังต่อไปนี้

JC rel กระโดด ถ้าค่าแฟลกทดเป็น 1

JNC rel กระโดด ถ้าค่าแฟลกทดเป็น 0

JB bit,rel กระโดด ถ้าค่า bit เป็น 0 และเปลี่ยนค่า bit เป็น 1

JNB bit,rel กระโดด ถ้าค่า bit เป็น 1 และเปลี่ยนค่า bit เป็น 0

JBC bit,rel กระโดด ถ้าค่าแฟลกทดเป็น 0

JZ rel กระโดดไปตำแหน่งที่สัมพัทธ์ กับตำแหน่งปัจจุบันถ้าข้อมูล Rig A เป็น 0

JNZ rel กระโดดไปตำแหน่งที่สัมพัทธ์ กับตำแหน่งปัจจุบันถ้าข้อมูล Rig A เป็น 1

DJNZ Rn,rel ลดค่าข้อมูลใน Rn ลง 1 ค่า (Rn =Rn-1) ถ้าค่าไม่เป็น 0 ให้กระโดดไปยังตำแหน่งที่สัมพัทธ์ กับตำแหน่งปัจจุบัน

DJNZ direct,rel ลดค่าข้อมูลใน direct ลง 1 ค่า (direct = direct -1) ถ้าค่าไม่เป็น 0 ให้กระโดดไปยังตำแหน่งที่สัมพัทธ์ กับตำแหน่งปัจจุบัน

CJNE A, direct, rel

คำสั่ง CJNE (Compare and Jump If Not Equal) ทำให้ซีพียูทำการกระโดดไปยังแอดเดรสปลายทาง เมื่อค่าข้อมูลที่นำมาเปรียบเทียบ (Compare) ไม่เท่ากัน (Not Equal) แต่ถ้าข้อมูลมีค่าเท่ากันทำคำสั่งในบรรทัดถัดไป

คำสั่ง CJNE A, direct, rel นำข้อมูลในรีจิสเตอร์ A และข้อมูลในตำแหน่งหน่วยความจำที่อ้างแอดเดรสแบบ direct มาเปรียบเทียบกัน เมื่อค่าของรีจิสเตอร์ Aไม่เท่ากับค่าในหน่วยความจำข้อมูลกำหนดให้ซีพียูกระโดดไปยังแอดเดรสปลายทางตามค่าสัมพัทธ์ แต่ถ้าหากเปรียบเทียบแล้วมีค่าเท่ากันให้ทำคำสั่งในบรรทัดถัดไป

ตัวอย่าง CJNE A, 20H, LEFT ; นำค่าข้อมูลในรีจิสเตอร์ A และข้อมูลในตำแหน่งหน่วยความจำ 20H มาเปรียบเทียบกัน ถ้าค่าข้อมูลไม่เท่ากัน ซีพียูกระโดดไปทำยังตำแหน่งเลเบล LEFT แต่ถ้าหากเปรียบเทียบแล้วปรากฏว่าข้อมูลมีค่าเท่ากัน ให้ทำคำสั่งในบรรทัดถัดไป

CJNE A, direct, rel	**CJNE A, #data, rel**	**CJNE Rn, #data, rel**	**CJNE @Rn, #data, rel**
เปรียบเทียบค่าที่ Reg A กับค่าในหน่วยความจำ ถ้าไม่เท่ากันกระโดดไปที่ตำแหน่ง rel	เปรียบเทียบค่าที่ Reg A กับข้อมูลค่าคงที่ถ้าไม่เท่ากันให้กระโดดไปที่ตำแหน่ง rel	เปรียบเทียบค่าในตำแหน่งหน่วยความจำ R0-R7 กับข้อมูลค่าคงที่ ถ้าไม่เท่ากระโดดไปที่ตำแหน่ง rel	เปรียบเทียบค่าข้อมูลในตำแหน่งหน่วยความจำที่ชี้โดย R0-R1กับค่าคงที่ถ้าไม่เท่ากระโดดไปที่ตำแหน่ง rel

คำสั่ง JZ rel เป็นคำสั่งเพื่อตรวจสอบข้อมูลในรีจิสเตอร์ A หากข้อมูลในรีจิสเตอร์ A มีค่าเป็น 00H ให้กระโดดไปยังตำแหน่งที่สัมพัทธ์

คำสั่ง JNZ rel เป็นคำสั่งเพื่อตรวจสอบข้อมูลในรีจิสเตอร์ A หากข้อมูลในรีจิสเตอร์ A ไม่เป็น 00H ให้กระโดดไปยังตำแหน่งที่สัมพัทธ์กับตำแหน่งปัจจุบัน

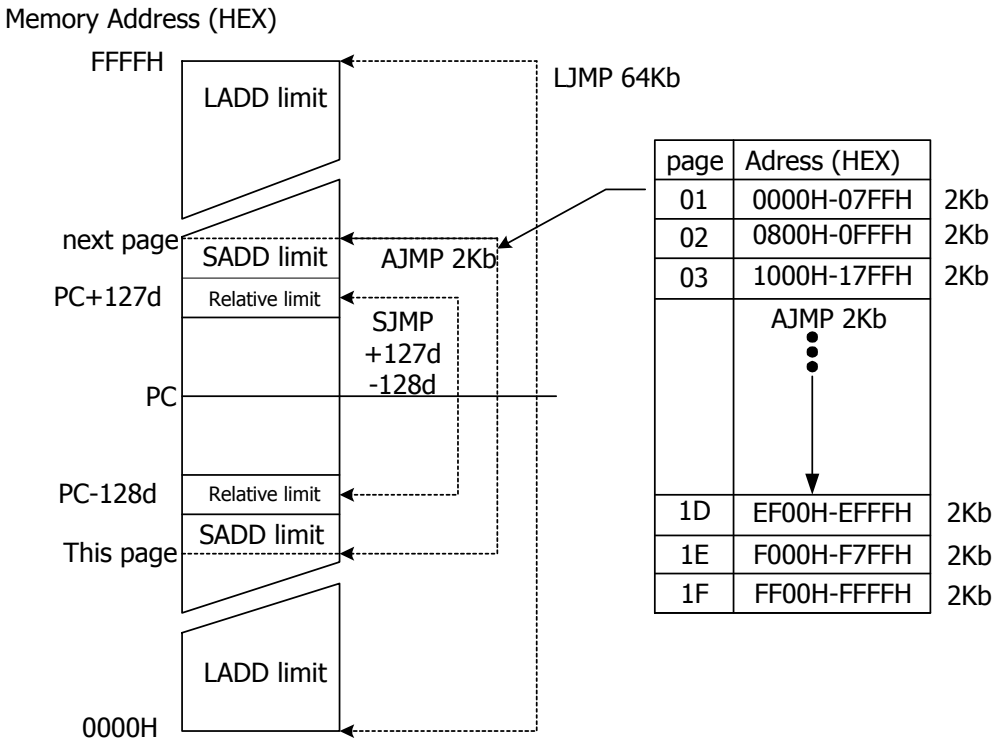
6. การกระโดด และการเรียกใช้โปรแกรมย่อย

6.1 การกระโดด

การกระโดดใช้ควบคุมการทำงานของโปรแกรม โดยเปลี่ยนตำแหน่งไปทำงานที่ต่างๆ ได้ตามต้องการ และขณะทำงานค่าของโปรแกรมเคาน์เตอร์ (PC: Program Counter) เปลี่ยนไปด้วย การกระโดดแบ่งออกเป็น 2 ลักษณะ คือการกระโดดแบบไม่มีเงื่อนไข (Unconditional Jumps) เป็นการกระโดดในตำแหน่งแอดเดรสที่ระบุตามคำสั่งทันทีทันใด และการกระโดดแบบมีเงื่อนไข (Conditional Jumps)

เป็นการกระโดดไปทำงานในตำแหน่งแอดเดรสที่ระบุ โดยตรวจสอบเงื่อนไขเป็นจริง หรือเป็นเท็จจากการกำหนดที่คำสั่ง

ในช่วงของการกระโดดเพื่อให้โปรแกรมเคาน์เตอร์ ไปทำตำแหน่งแอดเดรสที่กำหนด โดยใช้คำสั่ง สามารถแบ่งออกเป็น 3 ช่วง แสดงดังภาพที่ 4.4



ภาพที่ 4.4 แสดงแผนภาพแสดงการกระโดดทั้ง 3 ช่วง

6.1.1 การกระโดดแบบสัมพัทธ์ (Relative Range) คือการกระโดดไปทำตำแหน่งแอดเดรสที่กำหนด เป็นระยะไม่เกิน–127d +128d เช่น คำสั่ง SJMP rel (Short Jump) เป็นคำสั่งที่มีขนาด 2 ไบต์ 2 แมชชีนไซเคิล การทำงานจะกำหนดให้ซีพียูกระโดดโดยไม่มีเงื่อนไข ไปทำงานยังแอดเดรสที่กำหนดด้วยค่าสัมพัทธ์ (rel) ต้องมีขอบเขตไม่เกิน -127d ถึง + 128d แอดเดรส ระยะกระโดดเริ่มจากตำแหน่งถัดไป (ค่าบวก) หรือนับ แอดเดรสถอยหลัง (ค่าลบ) ของคำสั่งกระโดด แต่ใช้คำสั่งในการอ้างแอดเดรสขนาด 1 ไบต์ (รวมคำสั่งกระโดดเป็น 2 ไบต์) ค่าของ rel สามารถหาได้จาก ระยะห่างจากจุดที่กระโดดตำแหน่งของโปรแกรมเคาน์เตอร์ในขณะนั้นไปถึงตำแหน่งที่ต้องการแล้วนำมาลบด้วย 2 (2 มาจากเนื้อที่สองไบต์ของคำสั่ง SJMP)

6.1.2 การกระโดดแบบสัมบูรณ์ (Short Absolute Range) คือการกระโดดไปทำตำแหน่งหมายเลขแอดเดรสที่กำหนดใหม่ได้โดยตรง และอยู่ในช่วงสัมบูรณ์ของค่า 11 บิต โดยกำหนดขอบเขตเพียง 2 กิโลไบต์จากตำแหน่งแอดเดรสของคำสั่งถัดไป ซึ่งใช้หน่วยความจำในการเก็บตำแหน่ง 11 บิต เช่นคำสั่ง

AJMP addr11 เป็นคำสั่งที่มีขนาด 2 ไบต์ 2 แมชชีนไซเคิล การทำงานจะ กำหนดให้ซีพียูกระโดดโดยไม่มีเงื่อนไข ไปทำงานยังแอดเดรสที่กำหนดด้วยค่าขอบเขตสัมบูรณ์แบบใกล้ซึ่งสามารถอ้างแอดเดรสได้สูงสุด 2 กิโลไบต์ (000H-7FFH) แสดงดังภาพที่ 4.3 ระยะที่กระโดดกำหนดตำแหน่งถัดจากคำสั่งนี้ไป

6.1.3 การกระโดดแบบไกล (Long Absolute Range) สามารถระบุตำแหน่งที่กระโดดได้ตลอดช่วงความสามารถในการอ้างแอดเดรสของไอซี MCS-51 ทั้ง 16 บิต (64 Kbytes) แต่ให้คำสั่งในการอ้างแอดเดรสขนาด 2 ไบต์ (รวมคำสั่งกระโดดแล้ว เป็น 3 ไบต์) เช่นคำสั่ง LJMP addr1 เป็นคำสั่งที่มีขนาด 3 ไบต์ 2 แมชชีนไซเคิล การทำงานจะกำหนดให้กระโดดแบบไม่มีเงื่อนไข โดยไปทำงานยังแอดเดรสที่กำหนดด้วยค่าขอบเขตสัมบูรณ์แบบไกลซึ่งสามารถอ้างแอดเดรสได้ตลอดช่วงความสามารถในการอ้างแอดเดรสของ ไอซี MCS-51 ทั้ง 16 บิต

6.2 โปรแกรมเคาน์เตอร์ (Program Counter)

โปรแกรมเคาน์เตอร์ หรือ PC เป็นรีจิสเตอร์ขนาด 16 บิต ทำหน้าที่ชี้ ค่าตำแหน่งแอดเดรสของหน่วยความจำโปรแกรม โดยทำหน้าที่ให้ไอซี MCS-51 นำคำสั่งไปประมวลผล (Fetch) ในหน่วยความจำได้อย่างถูกต้อง และหลังจากไอซี MCS-51 ทำงานเสร็จสิ้น ค่าตำแหน่งในโปรแกรมเคาน์เตอร์เพิ่มขึ้นอีก 1 ค่าโดยอัตโนมัติ เพื่อเตรียมอ่านคำสั่งถัดต่อไปอีก โปรแกรมเคาน์เตอร์ นำหน้าตำแหน่งแอดเดรสในขณะที่ซีพียูทำงานอยู่ 1 ตำแหน่ง

6.3 โปรแกรมย่อย

โปรแกรมย่อย คือโปรแกรมที่เขียนไว้สำหรับให้ใช้งานในหน้าที่หนึ่งๆ ซึ่งแยกออกไปจากโปรแกรมหลัก และถูกเรียกใช้งานซ้ำหลายครั้ง ตัวอย่างเช่นโปรแกรมหน่วงเวลา ในโปรแกรมหลักหากต้องการเรียกใช้โปรแกรมหน่วงเวลาหลายครั้ง ต้องเขียนโปรแกรมหน่วงเวลา เท่ากับจำนวนครั้งที่เรียกใช้ ดังนั้นหากเขียนให้เป็นโปรแกรมย่อย ทำหน้าที่หน่วงเวลาชื่อ DELAY เก็บไว้แล้ว หากต้องการเรียกใช้ในตำแหน่งใดๆ ของโปรแกรมหลักให้เขียนคำสั่ง CALL DELAY เพื่อให้โปรแกรมหลักกระโดดไปทำที่โปรแกรมย่อยหน่วงเวลา และทำการเก็บค่าของโปรแกรมเคาน์เตอร์ไว้ใน สแตก เมื่อซีพียูปฏิบัติตามคำสั่งในโปรแกรมย่อยเสร็จ และกลับไปทำงานต่อที่โปรแกรมหลัก ต้องใช้ คำสั่ง RET โดยสแตกต้องคืนค่าให้กับโปรแกรมเคาน์เตอร์ มีคำสั่งดังต่อไปนี้

ACALL addr11 (Short Absolute Call) เป็นคำสั่งที่มีขนาด 2 ไบต์ 2 แมชชีนไซเคิล

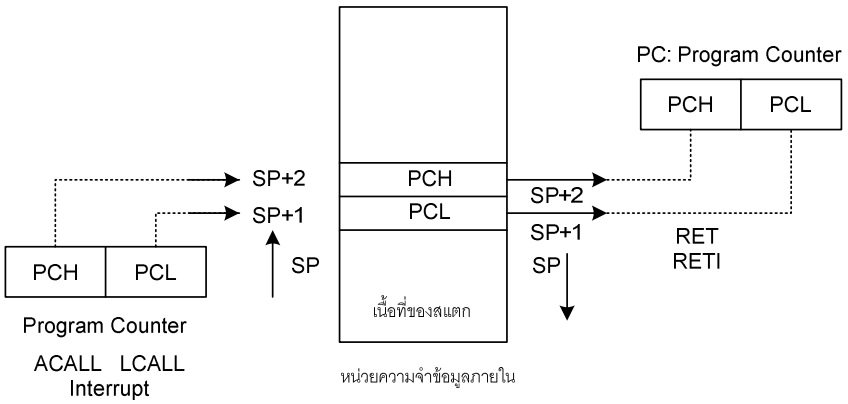
การทำงาน กำหนดให้ทำงานที่โปรแกรมย่อย โดยไม่มีเงื่อนไข มีแอดเดรสอยู่ภายในขอบเขตสัมบูรณ์แบบใกล้ อ้างแอดเดรสได้สูงสุด 2 กิโลไบต์ หรือ 11 บิต (000H-7FFH) การแบ่งเพจ (Page) เหมือนคำสั่ง AJMP คำสั่งนี้บวกค่าของโปรแกรมแกรมเคาน์เตอร์เพิ่ม 2 ค่า เพราะเป็นคำสั่งขนาด 2 ไบต์ ก่อนไปทำงานที่โปรแกรมย่อย ต้องเก็บค่าของโปรแกรมเคาน์เตอร์ไว้ที่สแตกโดยใช้เนื้อที่ 2 ไบต์ และ นำไบต์ต่ำไปเก็บไว้ก่อน ต่อมาเพิ่มค่าในสแตกขึ้นอีก 1 ค่าทำการเก็บค่าไบต์สูงไว้ ค่าของโปรแกรมเคาน์เตอร์เปลี่ยน

เป็นค่าของตำแหน่งแอดเดรสใหม่ ตามที่กำหนดในคำสั่ง ACALL การออกจากโปรแกรมย่อย เพื่อกลับไปโปรแกรมหลัก โดยใช้คำสั่ง RET จึงคืนค่าจากสแตกให้กับโปรแกรมเคาน์เตอร์

LCALL addr16 (Long Absolute Call) เป็นคำสั่งที่มีขนาด 3 ไบต์ 2 แมชชีนไซเคิล

การทำงาน กำหนดให้ไปทำงานที่โปรแกรมย่อย โดยไม่มีเงื่อนไข มีแอดเดรสอยู่ภายในขอบเขตสัมบูรณ์แบบไกล ซึ่งสามารถอ้างแอดเดรสได้สูงสุด ตลอดช่วงความสามารถในการอ้างแอดเดรสของไอซี MCS-51 ทั้ง 16 บิต (64 Kbytes) คำสั่งนี้บวกค่าของโปรแกรมแกรมเคาน์เตอร์ ไปอีก 3 ค่า เพราะเป็นคำสั่งขนาด 3 ไบต์ ก่อนทำงานที่โปรแกรมย่อย ต้องเก็บค่าของโปรแกรมเคาน์เตอร์ไว้ที่สแตกโดยใช้เนื้อที่ 2 ไบต์ และนำไบต์ต่ำไปเก็บไว้ก่อน หลังจากนั้นให้เพิ่มค่าในสแตกขึ้น และทำการเก็บค่าไบต์สูงในลำดับต่อไป ค่าของโปรแกรมเคาน์เตอร์ เปลี่ยนเป็นค่าของตำแหน่งแอดเดรสใหม่ตามที่กำหนดในคำสั่ง ACALL และออกจากโปรแกรมย่อยเพื่อกลับมาโปรแกรมหลัก เมื่อพบคำสั่ง RET

RET (Return from Subroutine) การทำงาน กำหนดให้ซีพียูออกจากโปรแกรมย่อย โดยให้สแตกคืนค่าแอดเดรสให้กับโปรแกรมเคาน์เตอร์ ดังนั้นค่าของสแตกถูกลดค่าลง 2 ค่า โดยคำสั่ง RET เป็นคำสั่งสุดท้ายของทุกโปรแกรมย่อย ยกเว้นโปรแกรมย่อยบริการอินเตอร์รัปต์ต้องเพิ่มอักษร I: RETI



ภาพที่ 4.5 แสดงการ PUSH และ POP ค่าโปรแกรมเคาน์เตอร์ลงในสแตก

ลำดับการทำงานเมื่อเลือกใช้โปรแกรมย่อย

1) เมื่อพบคำสั่ง CALL เช่น ACALL, LCALL หรือมีสัญญาณอินเตอร์รัปต์เข้ามา

2) ก่อนที่กระโดดไปทำในตำแหน่งโปรแกรมย่อย ต้องเก็บค่า PC ของคำสั่งถัดไปไว้ในสแตก โดยใส่ไบต์ต่ำในตำแหน่ง SP+1 และไบต์สูงในตำแหน่ง SP+2

3) ตำแหน่งของโปรแกรมย่อยจะถูกโหลดเข้าในโปรแกรมเคาน์เตอร์ PC แล้วไปทำงานในตำแหน่งนั้นจนเสร็จสิ้น

4) เมื่อพบคำสั่ง RET หรือ RETI ในบรรทัดสุดท้าย ของโปรแกรมย่อย ซีพียูจะกระโดดกลับมาที่โปรแกรม หลักโดยวิธีการ POP ค่าในสแตกเข้าที่โปรแกรมเคาน์เตอร์ PC โดยตำแหน่ง SP+2 จะโหลดเข้าที่ PCH และค่าในตำแหน่ง SP+1 จะถูกโหลดเข้าใน PCL หลังจากนั้น จะลดค่า SP ลงอีกหนึ่งเหลือ SP แสดงดังภาพที่ 4.5

7. การเขียนโปรแกรมแบบเปิดตาราง (Lookup Table)

โปรแกรมแบบ เปิดตาราง (Lookup Table) หมายถึงการใช้คำสั่งนำค่าข้อมูลจากการเก็บบันทึกที่หน่วยความจำโปรแกรม เพื่อนำกลับใช้งานในภายหลังร่วมกับวิธีการเข้าถึงข้อมูลโดยการอ้างแอดเดรสแบบอินเด็กซ์ แสดงดังตารางที่ 4.2

ตารางที่ 4.2 แสดงการทำงานของวิธีเปิดตาราง (Lookup Table)

เลเบล (Label)	แอดเดรสของ หน่วยความจำโปรแกรม (Flash) รีจิสเตอร์ DPTR (MOV DPTR,#TABLE)	ข้อมูลที่กำหนดไว้ใน หน่วยความจำแบบโปรแกรม (Flash) มีขนาด 8 บิต DB :Define Byte	รีจิสเตอร์ A =@A+DPTR @ A = 0, 1, 2, 3, 4… การอ้างแอดเดรสแบบ อินเด็กซ์
TABLE:	0500H	01H	Reg. A =@0+0500
	0501H	02H	Reg. A =@1+0500
	0502H	75H	Reg. A =@2+0500
	0503H	AFH	Reg. A =@3+0500
	0504H	CDH	Reg. A =@4+0500

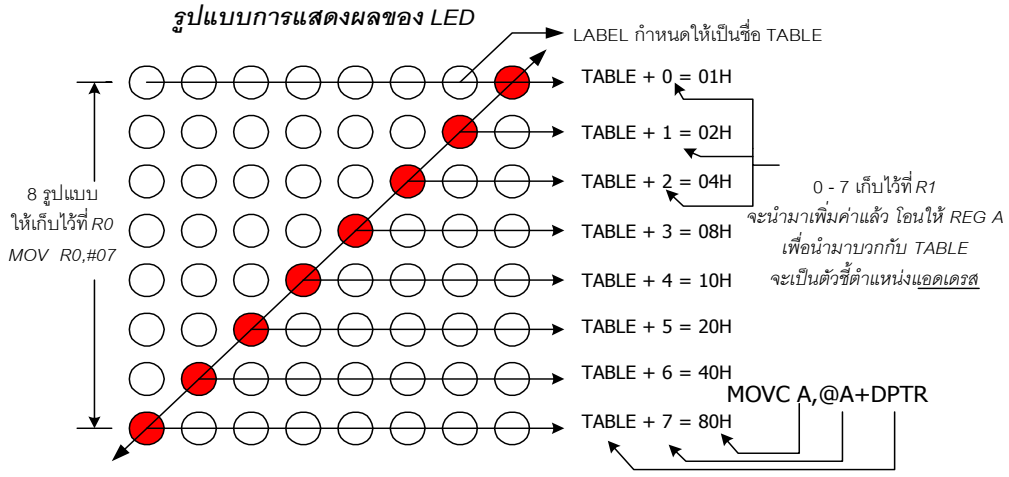
การทำงานของรีจิสเตอร์ DPTR คำสั่ง **MOV DPTR,#TABLE** หรือ **MOV DPTR,#0500H** เป็นการกำหนดค่าฐานแอดเดรสให้กับรีจิสเตอร์ DPTR ในตารางรีจิสเตอร์ DPTR มีค่าเท่ากับ 0500H รีจิสเตอร์ A ในคำสั่ง **MOVC A,@A+DPTR** ถูกใช้งาน 2 ครั้ง โดยครั้งแรกเป็นตัวบวกกับฐานแอดเดรสเพื่อชี้ตำแหน่งของข้อมูล และในครั้งที่สองเป็นตัวเก็บค่าข้อมูลที่ได้จากการเปิดตาราง ดังนั้นรีจิสเตอร์ A ถูกเปลี่ยนค่าไปสองครั้งในหนึ่งคำสั่ง ถ้าต้องการเปิดข้อมูลแบบต่อเนื่องจากตารางทีละลำดับแอดเดรสต้องกำหนดค่าที่รีจิสเตอร์ A ใหม่ทุกครั้ง หรือใช้วิธีการเพิ่มค่าข้อมูลในรีจิสเตอร์ DPTR โดยใช้คำสั่ง **INC DPTR** แต่คำสั่งลดค่าข้อมูลใน **DPTR** ไม่มีใน MCS-51 ดังนั้นจึงต้องใช้รีจิสเตอร์ Rn (R0-R7) หรือหน่วยความจำข้อมูล 1 ตำแหน่ง เพื่อเก็บค่าข้อมูลของรีจิสเตอร์ A ไว้ก่อน แล้วจึงนำรีจิสเตอร์ A ไปบวกในคำสั่ง @A+DPTR ดังตัวอย่างดังนี้

```
MOV   R0,#00H          ; 1  ให้ R0 เป็นตัวกำหนดค่า และนำข้อมูลให้กับรีจิสเตอร์ A
MOV   DPTR,#TABLE  ; 2  DPTR เป็นฐานแอดเดรส เก็บค่าตำแหน่งแอดเดรสเลเบล TABEL
```

```
LOOP:  MOV    A,R0            ; 3 นำค่าใน  R0 เป็นข้อมูลให้รีจิสเตอร์ A  ดังนั้น  A= ข้อมูลใน R0
       MOVC   A,@A+DPTR       ; 4 ข้อมูลใน  DPTR บวกกับค่าในรีจิสเตอร์ A (XXXX + YY) ผลลัพธ์
                              ; ไปเปิดข้อมูลในตำแหน่งแอดเดรสนั้น และนำข้อมูลเก็บที่รีจิสเตอร์ A
       MOV    P1,A            ; 5  นำข้อมูลที่ได้ไปแสดงผลที่พอร์ต P1
       ACALL DELAY            ; 6 เรียกโปรแกรมย่อยหน่วงเวลา
       INC    R0              ; 7 เพิ่มค่าที่ R0 หนึ่งค่าเป็นข้อมูลให้กับ A เพื่อบวกกับฐานแอดเดรส
       SJMP   LOOP            ; 8 กระโดดไป LOOP เพื่อนำค่าในรีจิสเตอร์ R0 เก็บที่ A ใหม่อีกรอบ
              .
              .
TABLE: DB   01H, 02H, 03H, 04H     ; 9 กำหนดข้อมูลขนาด 8 บิต ลงตำแหน่งแอดเดรสทีละค่าโดยเริ่ม
       DB   05H, 06H, 07H, 08H     ; จากแอดเดรสที่อยู่ของเลเบล TABLE
       DB   09H,0AH,0BH,0CH        ; ใช้เครื่องหมายคอมม่า ( ; ) แบ่งข้อมูลขนาด 8 บิต
              .
DELAY:        .

              END
```

ข้อสังเกตรีจิสเตอร์ R0 ทำหน้าที่กำหนดค่าให้กับรีจิสเตอร์ A ในบรรทัดที่ 3 และรีจิสเตอร์ A ต้องเปลี่ยนค่าในบรรทัดที่ 4 จากข้อมูลในรีจิสเตอร์ R0 เป็นข้อมูลที่เปิดตารางบรรทัดที่ 9 ตัวอย่างไฟวิ่งโดยใช้วิธีการเปิดตาราง (Lookup Table) แสดงดังภาพที่ 4.6



ภาพที่ 4.6 แสดงขั้นตอนของโปรแกรมโดยใช้วิธีการเปิดตาราง (Lookup Table)

สรุป

ไอซี MCS-51 สามารถติดต่อกับหน่วยความจำข้อมูลภายนอก และหน่วยความจำโปรแกรมภายนอก ได้ถึง 64 กิโลไบต์โดยใช้ขาแอดเดรสจำนวน 16 เส้น (0000H –FFFFH) และไม่สามารถกระทำโดยตรงได้ แต่ใช้รีจิสเตอร์ DPTR หรือรีจิสเตอร์ PC ที่มีขนาด 16 บิตใช้งานร่วมกับรีจิสเตอร์ A เพื่อเคลื่อนย้าย โดยทางอ้อม ไอซี MCS-51 แบ่งกลุ่มคำสั่งได้ดังนี้

1. กลุ่มคำสั่งเคลื่อนย้ายข้อมูลแบ่งออกได้เป็น ประเภทของหน่วยความจำทีใช้ดังนี้

1.1 กลุ่มคำสั่งเคลื่อนย้ายข้อมูลระหว่างหน่วยความจำภายในแรม

1.2 กลุ่มคำสั่งเคลื่อนย้ายข้อมูลระหว่างภายใน และภายนอกไอซี (Data Transfer)

1.3 กลุ่มคำสั่งการเคลื่อนย้ายข้อมูลกับสแตก

1.4 กลุ่มคำสั่งการแลกเปลี่ยนข้อมูล

2. กลุ่มคำสั่งทางคณิตศาสตร์ (Arithmetic Instructions) ในไอซี MCS-51 ประกอบด้วยกลุ่มคำสั่งกระทำทางคณิตศาสตร์ดังต่อไปนี้

2.1 คำสั่งการบวก เป็นคำสั่งบวกค่าข้อมูลโดยรีจิสเตอร์ A ทำหน้าที่เก็บค่าผลลัพธ์ปลายทางไว้ คำสั่งการบวกข้อมูลมีผลต่อ แฟลกในรีจิสเตอร์ PSW

2.2 คำสั่งการลบ

2.3 คำสั่งการเพิ่มค่า 1 ค่า

2.4 คำสั่งการลดค่า 1 ค่า

2.5 คำสั่งการคูณข้อมูลในรีจิสเตอร์ A กับข้อมูลในรีจิสเตอร์ B แล้วเก็บค่าข้อมูลไบต์สูงในรีจิสเตอร์ A ไบต์ต่ำไว้ในรีจิสเตอร์ B

2.6 คำสั่งการหารค่าข้อมูลในรีจิสเตอร์ A ด้วยค่าข้อมูลที่รีจิสเตอร์ B แล้วนำผลลัพธ์การหารเก็บไว้ที่รีจิสเตอร์ A ส่วนเศษของการหารเก็บไว้ที่รีจิสเตอร์ B

2.7 คำสั่งการปรับค่าเป็นเลขฐานสิบเป็นการปรับค่าจากเลข Binary ให้เป็นเลข BCD

3. คำสั่งทางตรรกะ (Logical Instruction) เป็นกลุ่มคำสั่งที่ทำหน้าที่เกี่ยวกับการ AND, OR, XOR, NOT รวมถึงการหมุนข้อมูล โดยกระทำทางลอจิกครั้งละ 8 บิต และทำบิตต่อบิตที่ตรงกัน

4. คำสั่งจัดการแบบบูลีน (Boolean Variable Manipulation) เป็นการทำในระดับบิต สามารถกระทำได้กับรีจิสเตอร์ใช้งานพิเศษบางตัว และหน่วยความจำภายในที่เข้าถึงในระดับบิตตั้งแต่แอดเดรส 20H -2FH

5. คำสั่งควบคุมการทำงานโปรแกรม (Program and Machine Control) เป็นกลุ่มที่ใช้ในการเปลี่ยนแปลงตำแหน่งแอดเดรสการทำงานของซีพียู เนื่องจากในบางครั้งซีพียูมีความจำเป็นต้องกระโดดไปทำงานที่แอดเดรสอื่นๆ การกระโดดควบคุมการทำงานของโปรแกรม โดยเปลี่ยนตำแหน่งของไปทำงานที่ต่างๆตามต้องการ และขณะทำงานค่าของโปรแกรมเคาน์เตอร์ (Program Counter) ต้องเปลี่ยนไปด้วย

การกระโดดแบ่งออกเป็น 2 ลักษณะ คือการกระโดดแบบไม่มีเงื่อนไข (Unconditional Jumps) เป็นการกระโดดในตำแหน่งแอดเดรสที่ระบุตามคำสั่งทันทีทันใด และการกระโดดแบบมีเงื่อนไข (Conditional Jumps) เป็นการกระโดดไปทำงานในตำแหน่งแอดเดรสที่ระบุ โดยตรวจสอบเงื่อนไขเป็นจริง หรือเป็นเท็จจากการกำหนดที่คำสั่ง

5.1 การกระโดดแบบสัมพัทธ์ (Relative Range) เป็นการกระโดดไปทำตำแหน่งแอดเดรสที่กำหนด เป็นระยะไม่เกิน–127d +128d เช่น คำสั่ง

5.2 การกระโดดแบบสัมบูรณ์ (Short Absolute Range) เป็นการกระโดดไปทำตำแหน่งหมายเลขแอดเดรสที่กำหนดใหม่ได้โดยตรง และอยู่ในช่วงสัมบูรณ์ของค่า 11 บิต โดยกำหนดขอบเขตเพียง 2 กิโลไบต์จากตำแหน่งแอดเดรสของคำสั่งถัดไป ซึ่งใช้หน่วยความจำในการเก็บตำแหน่ง 11 บิต เช่นคำสั่ง

5.3 การกระโดดแบบไกล (Long Absolute Range) สามารถระบุตำแหน่งที่กระโดดได้ตลอดช่วงความสามารถในการอ้างแอดเดรสของไอซี MCS-51 ทั้ง 16 บิต (64 Kbytes) ให้คำสั่งในการอ้างแอดเดรสขนาด 2 ไบต์ (รวมคำสั่งกระโดดแล้ว เป็น 3 ไบต์)

โปรแกรมเคาน์เตอร์ (Program Counter) หรือ PC เป็นรีจิสเตอร์ขนาด 16 บิต ทำหน้าที่ชี้ค่าตำแหน่งแอดเดรสของหน่วยความจำโปรแกรม โดยทำหน้าที่ให้ไอซี MCS-51 นำคำสั่งไปประมวลผล (Fetch) ในหน่วยความจำได้อย่างถูกต้อง และหลังจากไอซี MCS-51 ทำงานเสร็จสิ้น ค่าตำแหน่งในโปรแกรมเคาน์เตอร์เพิ่มขึ้นอีก 1 ค่าโดยอัตโนมัติ เพื่อเตรียมอ่านคำสั่งถัดต่อไปอีก โปรแกรมเคาน์เตอร์ นำหน้าตำแหน่งแอดเดรสในขณะที่ซีพียูทำงานอยู่ 1 ตำแหน่ง

โปรแกรมย่อย คือโปรแกรมที่เขียนไว้สำหรับให้ใช้งานในหน้าที่หนึ่งๆ ซึ่งแยกออกไปจากโปรแกรมหลัก และถูกเรียกใช้งานซ้ำหลาย ครั้งในส่วนของโปรแกรมหลัก หากต้องการเรียกใช้ในตำแหน่งใดๆ ของโปรแกรมหลักให้ใช้คำสั่ง CALL เพื่อให้โปรแกรมหลักกระโดดไปทำที่โปรแกรมย่อย และทำการเก็บค่าของโปรแกรมเคาน์เตอร์ไว้ใน สแตก เมื่อซีพียูปฏิบัติตามคำสั่งในโปรแกรมย่อยเสร็จ และกลับไปทำงานต่อที่โปรแกรมหลัก ต้องใช้ คำสั่ง RET โดยสแตกต้องคืนค่าให้กับโปรแกรมเคาน์เตอร์

การเขียนโปรแกรมแบบเปิดตาราง (Lookup Table) หมายถึงการใช้คำสั่งนำค่าข้อมูลจากการเก็บบันทึกที่หน่วยความจำโปรแกรม เพื่อนำกลับใช้งานในภายหลังร่วมกับวิธีการเข้าถึงข้อมูลโดยการอ้างแอดเดรสแบบอินเด็กซ์