ภาคผนวก 1/2: ภาษาแอสแซมบลี้ สำหรับ MCS-51 Microcontroller

การเรียนรู้เพื่อใช้งานไมโครคอนโทรลเลอร์ สิ่งที่สำคัญในลำดับต่อมาจากที่ทำความเข้าใจถึงโครงสร้างทางฮาร์ดแวร์แล้วนั่นคือ การเขียนโปรแกรมเพื่อกำหนดให้ไมโครคอนโทรลเลอร์ทำงานข้อมูลของโปรแกรมที่ไมโครคอนโทรลเลอร์ต้องการจะอยู่ในรูปของรหัสเลขฐานสิบหกหรือที่เรียกกันว่าภาษาเครื่อง หรือ แมชีนโค้ด (Machine Code) แต่เนื่องจากการเขียนโปรแกรมในลักษณะที่เป็นภาษาเครื่องนี้ ผู้เขียนโปรแกรมต้องทำการเปิดตารางรหัสคำสั่งซึ่งเป็นเรื่องที่ยุ่งยากและทำให้การตรวจสอบโปรแกรมที่เขียนขึ้นกระทำได้ยากจึงใช้การเขียนโปรแกรมด้วยภาษาแอสแซมเบลอร์(Assembler) ทำการแปลภาษาแอสแซมบลีที่เขียนขึ้นนั้นเป็นภาษาเครื่องแล้วเขียนลงในหน่วยความจำโปรแกรมของไมโครคอนโทรลเลอร์ต่อไป

1. โครงสร้างของโปรแกรมภาษาแอสแซมบลี

ประกอบด้วย 4 ส่วนหลักคือ

1. ลาเบล (Label) ใช้ในการอ้างถึงบรรทัดใดบรรทัดหนึ่งของโปรแกรมที่ทำการเขียนขึ้น
2. รหัสนีโมนิก (Mnemonic) เป็นส่วนแสดงคำสั่งของไมโครคอนโทรลเลอร์ที่ต้องการให้กระทำ
3. โอเปอร์แรนด์ (Operand) เป็นส่วนที่แสดงถึงตัวกระทำหรือถูกกระทำและข้อมูลที่ใช้ในการกระทำตามคำสั่งที่กำหนดโดยรหัสนีโมนิกก่อนหน้านี้
4. คอมเมนต์ (Comment) เป็นส่วนที่ผู้เขียนโปรแกรมเขียนขึ้นเพื่อใช้ในการอธิบายคำสั่งที่กระทำ หรือผลของการกระทำคำสั่งในบรรทัดหรือโปรแกรมย่อยนั้นๆ ทั้งนี้เพื่อช่วยให้ผู้เขียนสามารถตรวจสอบโปรแกรมที่เขียนขึ้นได้ง่ายรวมถึงเป็นประโยชน์ต่อผู้ที่นำโปรแกรมนั้นมาศึกษาใหม่อีกด้วย

2. ชุดคำสั่งของไมโครคอนโทรลเลอร์ MCS-51

ไมโครคอนโทรลเลอร์MCS-51 ประกอบด้วยคำสั่งทั้งหมดจำนวนมากซึ่งนำมาแสดงไว้ในตารางของชุดคำสั่งต่างๆ ซึ่งสามารถจะจัดกลุ่มคำสั่งเหล่านี้ตามลักษณะและหน้าที่การทำงานที่คล้ายคลึงกัน เพื่อความสะดวกต่อการศึกษาทำความเข้าใจและใช้งาน ดังนี้

2.1. กลุ่มการถ่ายเทข้อมูล คือ กลุ่มคำสั่งในการโอนย้ายข้อมูล ทำหน้าที่ในโอนย้ายข้อมูลระหว่างรีจิสเตอร์หรือหน่วยความจำภายในแรม โดยมีรายละเอียดดังนี้ ชุดคำสั่งในการถ่ายเทแรม ภายในซึ่งเวลาที่ใช้ในหนึ่งคำสั่งนั้น จะเป็นเวลาเมื่อขณะที่ความถี่ในการทำงานของหน่วยประมวลผลกลาง ที่ความถี่ 12 เมกะเฮิรตซ์ และรายละเอียดของแต่ละคำสั่งมีดังนี้ MOV :จะทำงานในลักษณะเป็นการถ่ายเทข้อมูลที่มีขนาดเป็นไบต์ หรือ บิตก็ได้ จากแหล่งกำเนิดเข้าสู่ตัวรับข้อมูลในฟิลด์โอเปอร์แรนด์ PUSH:จะทำงานโดยเพิ่มค่ารีจิสเตอร์ SP ก่อนแล้วจึงทำการถ่ายเทข้อมูล 1 ไบต์จากแหล่งกำเนิดไปบริเวณสแต็กตามตำแหน่งที่รีจิสเตอร์ SP กำหนด POP:การถ่ายเทข้อมูลขนาด 1 ไบต์จากบริเวณตำแหน่งที่รีจิสเตอร์ SP กำหนดไปยังรีจิสเตอร์ที่โอเปอร์แรนด์ กำหนดและหลังจากนั้นรีจิสเตอร์ SP จะลดค่าลง XCH: คำสั่งแลกเปลี่ยนไบต์ระหว่างแหล่งกำเนิดโอเปอร์แรนด์กับรีจิสเตอร์ AXCHD คำสั่งในการแลกเปลี่ยนขนาดนิบเบิลทางอันดับต่ำของแหล่งกำเนิดโอเปอร์แรนด์กับนิบเบิลอันดับต่ำลงของแอกคิวมูเลเตอร์

MOV A, Rn	จำนวนไบต์: 1 การทำงาน: นำข้อมูลของรีจีสเตอร์ R0-R7 มาเก็บไว้ในแอกคิวมูเลเตอร์
MOV A, direct	จำนวนไบต์: 2 การทำงาน: นำข้อมูลของหน่วยความจำข้อมูลภายในมาเก็บไว้ในแอกคิวมูเลเตอร์
MOV A, #data	จำนวนไบต์: 2 การทำงาน: นำข้อมูลมาเก็บไว้ที่แอกคิวมูเลเตอร์

MOV A, @Rn	จำนวนไบต์: 1 การทำงาน: นำข้อมูลจากแอดเดรสของ หน่วยความจำข้อมูลภายใน ที่กำหนดไว้ในรีจีสเตอร์ R0 หรือ R1 มาเก็บไว้ในแอกคิวมูเลเตอร์
MOV Rn, A	จำนวนไบต์: 1 การทำงาน: นำข้อมูลจากแอกคิวมูเลเตอร์ไปเก็บไว้ในรีจีสเตอร์ R0-R7
MOV Rn, direct	จำนวน ไบต์: 2 การทำงาน: นำข้อมูลจากแอดเดรสของหน่วย ความจำข้อมูลภายใน ที่กำหนดมาเก็บไว้ ในรีจีสเตอร์ R0-R7
MOV Rn, #data	จำนวนไบต์: 2 การทำงาน: นำข้อมูลมาเก็บไว้ที่รีจีสเตอร์ R0-R7
MOV direct, A	จำนวนไบต์: 2 การทำงาน: นำข้อมูลจากแอกคิวมูเลเตอร์มาเก็บไว้ที่หน่วยความจำข้อมูลภายใน
MOV direct, Rn	จำนวนไบต์: 2 การทำงาน: นำข้อมูลที่อยู่ภายในรีจีสเตอร์ R0-R7 มาเก็บไว้ที่หน่วยความจำข้อมูลภายใน
MOV direct, direct	จำนวนไบต์: 3 การทำงาน: นำข้อมูลจากหน่วยความจำข้อมูล ภายในแอดเดรสหนึ่ง มาเก็บไว้ที่ หน่วย ความจำข้อมูล ภายในอีกแอดเดรสหนึ่ง
MOV direct, #data	จำนวนไบต์: 3 การทำงาน: นำข้อมูลจากแอกคิวมูเลเตอร์มาเก็บไว้ที่หน่วยความจำข้อมูลภายใน
MOV direct, @Rn	จำนวนไบต์: 2 การทำงาน: นำข้อมูลจากแอดเดรสของหน่วยความจำข้อมูลภายใน ที่กำหนดไว้ ในรีจีสเตอร์ R0 หรือ R1 มาเก็บไว้ในหน่วยความจำ ข้อมูลภายในที่กำหนด
MOV @Rn, A	จำนวนไบต์: 1 การทำงาน: นำข้อมูลจาก แอกคิวมูเลเตอร์ มาเก็บไว้ที่หน่วยความจำข้อมูลภายในที่กำหนด โดยค่า ของรีจีสเตอร์ R0 หรือ R1
MOV @Rn, direct	จำนวนไบต์: 2 การทำงาน: นำข้อมูลจากหน่วยความจำข้อมูล ภายในแอดเดรสหนึ่ง มาเก็บไว้ที่ หน่วย ความจำข้อมูลภายในอีกแอดเดรสหนึ่ง ที่กำหนดโดยค่าของ R0 หรือ R1
MOV @Rn, #data	จำนวนไบต์: 3 การทำงาน: นำข้อมูลไปเก็บไว้ที่หน่วยความจำข้อมูลที่กำหนดโดยค่าของ R0 หรือ R1
MOV C, bit	จำนวนไบต์: 2 การทำงาน: นำข้อมูลในระดับบิต จากหน่วยความจำภายใน มาเก็บไว้ในแฟลกทดซึ่งอยู่ในรีจีสเตอร์ PSW
MOV bit, C	จำนวนไบต์: 2 การทำงาน: นำข้อมูลในแฟลกทด ไปเก็บไว้ที่หน่วยความจำข้อมูลภายใน ที่สามารถเข้าถึง ระดับบิตได้
MOVX A, @Rn	จำนวนไบต์: 1 การทำงาน: นำข้อมูลจากแอดเดรสของหน่วยความจำข้อมูลภายนอก ที่กำหนดไว้ในรีจีสเตอร์ R0 หรือ R1 มาเก็บไว้ในแอกคิวมูเลเตอร์
MOVX A, @DPTR	จำนวนไบต์: 1 การทำงาน: นำข้อมูลจากแอดเดรสของหน่วยความจำข้อมูลภายนอก ที่กำหนดไว้ในรีจีสเตอร์ DPTR มาเก็บไว้ในแอกคิวมูเลเตอร์
MOVX @Rn, A	จำนวนไบต์: 1 การทำงาน: นำข้อมูลจากแอกคิวมูเลเตอร์ ไปเก็บไว้ที่แอดเดรสของหน่วยความจำข้อมูล ภายนอก ที่กำหนดไว้ในรีจีสเตอร์ R0 หรือ R1

MOVX @DPTR, A	จำนวนไบต์: 1 การทำงาน: นำข้อมูลจากแอกคิวมูเลเตอร์ มาเก็บไว้ในหน่วยความจำข้อมูลภายนอกที่ แอดเดรส ซึ่งกำหนดไว้ในรีจีสเตอร์ DPTR
MOVC A, @A+DPTR	จำนวนไบต์: 1 การทำงาน: นำข้อมูลจากหย่วยความจำข้อมูลภายนอก ในตำแหน่งแอดเดรสที่ได้รับการกำหนดด้วยค่าของรีจีสเตอร์ A รวมกับค่าในรีจีสเตอร์ PC
MOVC A, @A+PC	จำนวนไบต์: การทำงาน: นำข้อมูลจากหน่วยความจำโปรแกรมภายนอก ในตำแหน่งแอดเดรสที่ได้รับการกำหนดด้วยค่าของรีจีสเตอร์ A รวมกับค่าในรีจีสเตอร์ PC
XCH A, Rn	จำนวนไบต์: 1 การทำงาน: แลกเปลี่ยนข้อมูลระหว่างรีจีสเตอร์ A กับข้อมูลภายในรีจีสเตอร์ R0-R7
XCH A, direct	จำนวนไบต์: 2 การทำงาน: แลกเปลี่ยนข้อมูลระหว่างรีจีสเตอร์ A กับหน่วยความจำข้อมูลภายใน
XCH A, @Rn	จำนวนไบต์: 1 การทำงาน: แลกเปลี่ยนข้อมูลระหว่าง A กับข้อมูลภายในแอดเดรสที่ถูกชี้โดย R0 หรือ R1
XCHD A, @Rn	จำนวนไบต์: 1 การทำงาน: แลกเปลี่ยนข้อมูล บิต 0-3 ของรีจีสเตอร์ A กับข้อมูล บิต 3-0 ภายในแอดเดรสของหน่วยความจำที่ชี้โดยรีจีสเตอร์ R0 หรือ R1
PUSH direct	จำนวนไบต์: 2 การทำงาน: เพิ่มค่าของรีจีสเตอร์ตัวชี้สแต็ก (SP) ไปหนึ่งตำแหน่ง จากนั้นนำค่าของข้อมูลในหน่วยความจำที่กำหนดไปเก็บไว้ในแอดเดรสที่ชี้โดย SP
POP direct	จำนวนไบต์: 2 การทำงาน: นำข้อมูลในแอดเดรสที่ถูกชี้โดย SP กลับคืนหน่วยความจำในแอดเดรสที่กำหนดไว้ แล้วลดค่าของ SP ไปหนึ่งตำแหน่ง

2.2. กลุ่มคำสั่งทางคณิตศาสตร์ เช่น การบวก ลบ คูณ และหารข้อมูลภายในตัว รีจิสเตอร์ต่างๆ ช่วงเวลาการทำงาน ของแต่ละคำสั่งนั้นจะกำหนดที่ความถี่ของสัญญาณนาฬิกาที่ 12 เมกะเฮิรตซ์ คำสั่งทางคณิตศาสตร์ส่วนใหญ่ใช้เวลา 1 ms ยกเว้นคำสั่ง INC DPTR ซึ่งใช้เวลา 2 ms โดยที่คำสั่งการคูณและหารใช้เวลา 4 ms โดยมีรายละเอียดการใช้คำสั่งมีดังนี้ INC:เป็นการบวกหนึ่งกับโอเปอร์แรนด์และใส่ค่าใหม่กลับเข้าที่ตัวโอเปอร์แรนด์นั้นๆ DEC:เป็นการลบออกจากตัวเลขที่อยู่ในแหล่งกำเนิดโอเปอร์แรนด์ และนำผลลัพธ์ที่ได้มาเก็บไว้ที่ตัวโอเปอร์แรนด์นั้น ADD:เป็นการบวกในแอกคิวมูเลเตอร์เข้ากับค่าในแหล่งกำเนิดโอเปอร์แรนด์ ADDC:เป็นการบวกค่าต่างๆ ในแอกคิวมูเลเตอร์เข้ากับค่าในแหล่งกำเนิดโอเปอร์-แรนด์และบวกกับบิตทดด้วย SUBB:เป็นการนำเลขที่แหล่งกำเนิดโอเปอร์แรนด์ ลบออกจากตัวเลขใน A และนำค่าบิตตัวทดมาลบออกอีกและผลลัพธ์ที่ได้นำมาใส่ลงในแอกคิวมูเลเตอร์ A MUL:เป็นการคูณแบบไม่คิดตัวเครื่องหมายของตัวเลขที่อยู่ใน แอกคิวมูเลเตอร์กับเลขใน รีจิสเตอร์ B แล้วได้ผลลัพธ์ 2 ไบต์ นำมาเก็บไว้ที่ AB โดย A จะรับอันดับต่ำส่วน B จะรับอันดับสูง DIV:เป็นคำสั่งในการหารแบบไม่คิดเครื่องหมายที่อยู่ในแอกคิวมูเลเตอร์แล้วหารตัวเลขในรีจิสเตอร์ B แล้วนำผลลัพธ์ไปเก็บในแอกคิวมูเลเตอร์และเศษของการหารตัวเลข จะเก็บไว้ในรีจิสเตอร์ B DA:สำหรับการบวกกันทางตัวเลข BCD เป็นการปรับค่ารวม ซึ่งเป็นผลมาจากการบวกกันทางไบนารี่ของระบบตัวเลข BCD ขนาด 2 หลักสองจำนวน การปรับค่าตัวเลขผลรวมด้วยการใช้คำสั่ง DA จะได้ผลลัพธ์กลับมาที่แอกคิวมูเลเตอร์

ADD A, direct	จำนวนไบต์: 2 การทำงาน: ทำการบวกค่าในแอกคิวมูเลเตอร์ เข้ากับข้อมูลในหน่วยความจำข้อมูลภายใน แล้วนำผลลัพธ์ไปเก็บไว้ในแอกคิวมูเลเตอร์

ADD A, Rn	จำนวนไบต์: 1 การทำงาน: ทำการบวกค่าในแอกคิวมูเลเตอร์เข้ากับข้อมูลในรีจีสเตอร์ R0-R7 ขนาด 8 บิต แล้วนำผลลัพไปเก็บไว้ในแอกคิวมูเลเตอร์
ADD A, @Rn	จำนวนไบต์: 1 การทำงาน: ทำการบวกค่าในแอกคิวมูเลเตอร์เข้ากับข้อมูล 8 บิต ในแอดเดรสของหน่วยความจำที่ถูกชี้โดย R0 หรือ R1 แล้วนำผลลัพธ์ไปเก็บไว้ในแอกคิวมูเลเตอร์
SUBB A, #data	จำนวนไบต์: 2 การทำงาน: ทำการลบค่าในแอกคิวมูเลเตอร์ด้วยค่าของแฟลกทด (C) แล้วลบด้วยข้อมูล data ขนาด 8 บิต นำผลลัพธ์ไปเก็บไว้ในแอกคิวมูเลเตอร์
SUBB A, direct	จำนวนไบต์: 2 การทำงาน: ทำการลบค่าในแอกคิวมูเลเตอร์ด้วยค่าของแฟลกทด แล้วลบด้วยข้อมูล ในหน่วยความจำ ข้อมูลภายใน นำผลลัพธ์ไปเก็บไว้ในแอกคิวมูเลเตอร์
SUBB A, Rn	จำนวนไบต์: 1 การทำงาน: ทำการลบค่าในแอกคิวมูเลเตอร์ด้วยค่าของแฟลกทด แล้วลบด้วยข้อมูลในรีจีสเตอร์ R0-R7 ขนาด 8 บิต นำผลลัพธ์ไปเก็บไว้ในแอกคิวมูเลเตอร์
SUBB A, @Rn	จำนวนไบต์: 1 การทำงาน: ทำการลบค่าในแอกคิวมูเลเตอร์ด้วยค่าของแฟลกทด แล้วลบด้วยข้อมูล ในหน่วยความจำ ที่ถูกชี้โดย R0 หรือ R1 นำผลลัพธ์ไปเก็บไว้ในแอกคิวมูเลเตอร์
MUL AB	จำนวนไบต์: 1 การทำงาน: ทำการคูณค่าในแอกคิวมูเลเตอร์ด้วยค่าในรีจีสเตอร์ B นำผลคูณไบต์ล่างเก็บไว้ใน แอกคิวมูเลเตอร์ และผลคูณไบต์บนเก็บไว้ในรีจีสเตอร์ B
DIV AB	จำนวนไบต์: 1 การทำงาน: ทำการหารค่าในแอกคิวมูเลเตอร์ด้วยค่าในรีจีสเตอร์ B นำผลหารไบต์บนเก็บไว้ใน แอกคิวมูเลเตอร์ และเศษการหารไบต์ล่างเก็บไว้ในรีจีสเตอร์ B
INC A	จำนวนไบต์: 1 การทำงาน: ทำการเพิ่มค่าในแอกคิวมูเลเตอร์ขึ้นหนึ่งค่า แล้วนำค่าที่เพิ่มขึ้นไปเก็บไว้ใน แอกคิวมูเลเตอร์
INC direct	จำนวนไบต์: 2 การทำงาน: ทำการเพิ่มค่าของข้อมูลในหน่วยความจำข้อมูลภายในขึ้นหนึ่งค่า
INC Rn	จำนวนไบต์: 1 การทำงาน: ทำการเพิ่มค่าของข้อมูลในรีจีสเตอร์ R0-R7 ขึ้นหนึ่งค่า
INC @Rn	จำนวนไบต์: 1 การทำงาน: ทำการเพิ่มค่าของข้อมูลในหน่วยความจำข้อมูลภายในที่ถูกชี้ โดยรีจีสเตอร์ R0 หรือ R1 ขึ้นหนึ่งค่า
INC DPTR	จำนวนไบต์: 1 การทำงาน: ทำการเพิ่มค่าของข้อมูลในรีจีสเตอร์ DPTR ขึ้นหนึ่งค่า
DEC A	จำนวนไบต์: 1 การทำงาน: ทำการลดค่าในแอกคิวมูเลเตอร์ ลงหนึ่งค่า แล้วนำค่าที่ลดลงนี้ ไปเก็บไว้ใน รีจีสเตอร์ A
DEC direct	จำนวนไบต์: 2 การทำงาน: ทำการลดค่าของข้อมูลในหน่วยความจำข้อมูลภายในลงหนึ่งค่า
DEC Rn	จำนวนไบต์: 1 การทำงาน: ทำการลดค่าของข้อมูลในหน่วยความจำข้อมูลภายในลงหนึ่งค่า
DEC @Rn	จำนวนไบต์: 1 การทำงาน: ทำการลดค่าของข้อมูลในหน่วยความจำข้อมูลภายในที่ถูกชี้ โดยรีจีสเตอร์ R0 หรือ R1 ลงหนึ่งค่า

DA A	จำนวนไบต์: 1 การทำงาน: คำสั่งนี้ใช้ปรับค่าข้อมูลในรีจิสเตอร์ A ภายหลังการบวกเลขที่ใช้รหัส BCD(Binary Code Decimal) โดยคำสั่งนี้จะทำตามหลังคำสั่งบวก ADD หรือ ADDC ในกรณีที่เลข นำมาบวกเป็นเลขรหัส BCD ทั้งนี้เพื่อให้ผลลัพธ์ที่ได้จาก การบวกถูกเปลี่ยนกลับเป็นค่าเลขรหัส BCD ด้วย โดยการทำงานของคำสั่งจะตรวจสอบค่าในรีจิสเตอร์ A ภายหลังกระทำคำสั่งบวก

2.3 กลุ่มคำสั่งทางตรรกศาสตร์หรือ แบบลอจิก ทำหน้าที่เกี่ยวกับการประมวลผลแบบ ลอจิกต่างๆ เช่น การ AND OR หรือ EX-OR ระหว่างข้อมูลในรีจิสเตอร์ A นั่นเอง โดยมีการใช้คำสั่งดังนี้ CPL:เป็นการใช้คำสั่งกลับค่าหรือคอมพลีเมนต์ ข้อมูลในแอกคิวมูเลเตอร์จะไม่มีผลใดๆ ต่อค่าของแฟลก หรือการอ้างถึงตำแหน่งแอดเดรสนั้นตามบิตนั้นๆ RL, RLC, RR, RRC, SWAP:ทั้ง 5 คำสั่งนี้เป็นคำสั่งในการทำงานการวนบิตบนตัวของแอกคิวมูเลเตอร์ซึ่ง RL เป็นการวนบิตทางขวา, RLC เป็นการทำการวนทางซ้ายผ่านบิตทด, RRC เป็นการวนขวาผ่านบิตทด และ SWAP เป็นการวนซ้ายสี่ครั้ง ANL:เป็นการ ADD กันทางตรรกศาสตร์ ระหว่างแหล่งกำเนิดสองโอเปอร์แรนด์ ซึ่งจะสั่งให้ทำงานในรูปแบบของตรรกศาสตร์ทางข้อมูลขนาดเป็นไบต์หรือบิต

ANL A, #data	จำนวนไบต์: 2 การทำงาน: ทำการแอนด์ค่าของข้อมูลในแอกคิวมูเลเตอร์ กับข้อมูล data ขนาด 8 บิต แล้วนำผลลัพธ์ไปเก็บไว้ในแอกคิวมูเลเตอร์
ANL A, direct	จำนวนไบต์: 2 การทำงาน: ทำการแอนด์ค่าของข้อมูลในหน่วยความจำข้อมูลภายในกับค่าของรีจีสเตอร์ A นำผลลัพธ์ไปเก็บไว้ในรีจีสเตอร์ A
ANL A, Rn	จำนวนไบต์: 1 การทำงาน: ทำการแอนด์ค่าของข้อมูลในรีจีสเตอร์ R0-R7 กับค่าของรีจีสเตอร์ A นำผลลัพธ์ไปเก็บไว้ในรีจีสเตอร์ A
ANL A, @Rn	จำนวนไบต์: 1 การทำงาน: ทำการแอนด์ค่าของข้อมูลในหน่วยความจำข้อมูลภายในที่ถูกชี้ โดยรีจีสเตอร์ R0 หรือ R1 กับค่าในรีจีสเตอร์ A นำผลลัพธ์ไปเก็บไว้ในรีจีสเตอร์ A
ANL direct, A	จำนวนไบต์: 2 การทำงาน: ทำการแอนด์ค่าของข้อมูล ในหน่วยความจำข้อมูลภายในกับค่าของรีจีสเตอร์ A นำผลลัพธ์ไปเก็บไว้ในหน่วยความจำข้อมูลภายใน
ANL direct, #data	จำนวนไบต์: 3 การทำงาน: ทำการแอนด์ค่าของข้อมูล ในหน่วยความจำข้อมูลภายใน กับข้อมูล data ขนาด 8 บิต แล้วนำผลลัพธ์ไปเก็บไว้ในหน่วยความจำข้อมูลภายใน
ANL C, bit	จำนวนไบต์: 2 การทำงาน: ทำการแอนด์ค่าของข้อมูลในแฟลกทด กับค่าของข้อมูลในระดับบิตของรีจีสเตอร์ แล้วนำผลลัพธ์ไปเก็บไว้ในแฟลกทด
ANL C, /bit	จำนวนไบต์: 2 การทำงาน: ทำการแอนด์ค่าของข้อมูลในแฟลกทด กับค่าคอมพลีเมนต์ ของข้อมูลในระดับบิต ของรีจีสเตอร์ โดยข้อมูลของรีจีสเตอร์ไม่มีการเปลี่ยนแปลงจากนั้นนำผลลัพธ์ไปเก็บไว้ใน แฟลกทด (ค่าคอมพลีเมนต์ คือค่าที่ตรงข้ามกับค่าของข้อมูล)
ORL A, #data	จำนวนไบต์: 2 การทำงาน: ทำการออร์ค่าในแอกคิวมูเลเตอร์ กับข้อมูล data ขนาด 8 บิต นำผลลัพธ์ ไปเก็บไว้ในแอกคิวมูเลเตอร์

ORL A, direct	จำนวนไบต์: 2 การทำงาน: ทำการออร์ค่าของข้อมูลในหน่วยความจำข้อมูลภายใน กับค่าของรีจีสเตอร์ A นำผลลัพธ์ไปเก็บไว้ในรีจีสเตอร์ A
ORL A, Rn	จำนวนไบต์: 1 การทำงาน: ทำการออร์ค่าของข้อมูลในรีจีสเตอร์ R0-R7 กับค่าของรีจีสเตอร์ A นำผลลัพธ์ไปเก็บไว้ในรีจีสเตอร์ A
ORL A, @Rn	จำนวนไบต์: 1 การทำงาน: ทำการออร์ค่าของข้อมูลในหน่วยความจำข้อมูลภายใน ที่ถูกชี้โดยรีจีสเตอร์ R0 หรือ R1 กับค่าในรีจีสเตอร์ A นำผลลัพธ์ไปเก็บไว้ในรีจีสเตอร์ A
ORL direct, A	จำนวนไบต์: 2 การทำงาน: ทำการออร์ค่าของข้อมูลในหน่วยความจำข้อมูลภายใน กับค่าของรีจีสเตอร์ A นำผลลัพธ์ไปเก็บไว้ในหน่วยความจำข้อมูลภายใน
ORL direct, #data	จำนวนไบต์: 3 การทำงาน: ทำการออร์ค่าของข้อมูลในหน่วยความจำข้อมูลภายใน กับข้อมูล data ขนาด 8 บิต นำผลลัพธ์ไปเก็บไว้ในหน่วยความจำข้อมูลภายใน
ORL C, bit	จำนวนไบต์: 2 การทำงาน: ทำการออร์ค่าของข้อมูลในแฟลกทด กับค่าของข้อมูลในระดับบิตของรีจีสเตอร์ แล้วนำผลลัพธ์ไปเก็บไว้ในแฟลกทด
ORL C, /bit	จำนวนไบต์: 2 การทำงาน: ทำการออร์ค่าของข้อมูลในแฟลกทด กับค่าคอมพลีเมนต์ของข้อมูล ในระดับบิต ของรีจีสเตอร์ โดยข้อมูลของรีจีสเตอร์ไม่มีการเปลี่ยนแปลงจากนั้นนำผลลัพธ์ไปเก็บไว้ในแฟลกทด
XRL A, #data	จำนวนไบต์: 2 การทำงาน: ทำการเอ็กคลูซีฟ-ออร์ค่าในแอกคิวมูเลเตอร์กับข้อมูล data ขนาด 8 บิต นำผลลัพธ์ไปเก็บไว้ในแอกคิวมูเลเตอร์
XRL A, direct	จำนวนไบต์: 2 การทำงาน: ทำการเอ็กคลูซีฟ-ออร์ค่าในหน่วยความจำข้อมูลภายใน กับค่าของรีจีสเตอร์ A นำผลลัพธ์ไปเก็บไว้ในรีจีสเตอร์ A
XRL A, Rn	จำนวนไบต์: 1 การทำงาน: ทำการเอ็กคลูซีฟ-ออร์ค่าของข้อมูลในรีจีสเตอร์ R0-R7 กับค่าของรีจีสเตอร์ A นำผลลัพธ์ไปเก็บไว้ในรีจีสเตอร์ A
XRL A, @Rn	จำนวนไบต์: 1 การทำงาน: ทำการเอ็กคลูซีฟ-ออร์ค่าของข้อมูล ในหน่วยความจำข้อมูลภายใน ที่ถูกชี้โดย รีจีสเตอร์ R0 หรือ R1 กับค่าในรีจีสเตอร์ A นำผลลัพธ์ไปเก็บไว้ในรีจีสเตอร์ A
XRL direct, A	จำนวนไบต์: 2 การทำงาน: ทำการเอ็กคลูซีฟ-ออร์ค่าของข้อมูล ในหน่วยความจำข้อมูลภายในกับค่าของ รีจีสเตอร์ A นำผลลัพธ์ไปเก็บไว้ในหน่วยความจำข้อมูลภายใน
XRL direct, #data	จำนวนไบต์: 3 การทำงาน: ทำการเอ็กคลูซีฟ-ออร์ค่าของข้อมูล ในหน่วยความจำข้อมูลภายในกับข้อมูล data ขนาด 8 บิต นำผลลัพธ์ไปเก็บไว้ในหน่วยความจำข้อมูลภายใน
CLR A	จำนวนไบต์: 1 การทำงาน: ทำการเคลียร์ค่าของรีจีสเตอร์ A ให้เท่ากับ 00H
CPL A	จำนวนไบต์: 1 การทำงาน:ทำการกลับสถานะของข้อมูลในรีจีสเตอร์ A ให้มีค่าตรงข้าม

RL A	จำนวนไบต์: 1 การทำงาน: ทำการหมุนข้อมูลในแต่ละบิตของรีจีสเตอร์ A วนทางซ้าย บิต 7 จะ หมุนวนมายังบิต 0
RLC A	จำนวนไบต์: 1 การทำงาน: ทำการหมุนข้อมูลในแต่ละบิตของรีจีสเตอร์ A วนทางซ้ายผ่านแฟลกทด โดยบิต 7 จะหมุนไปยังแฟลกทด และข้อมูลของแฟลกทดเดิมจะหมุนเข้ามาในบิต 0
RR A	จำนวนไบต์: 1 การทำงาน: ทำการหมุนข้อมูลในแต่ละบิตของรีจีสเตอร์ A วนทางขวา บิต 0 จะ หมุนวนมายังบิต 7
RRC A	จำนวนไบต์: 1 การทำงาน: ทำการหมุนข้อมูลในแต่ละบิตของรีจีสเตอร์ A วนทางขวาผ่านแฟลกทด โดยบิต 0 จะหมุนไปยังแฟลกทด และข้อมูลของแฟลกทดเดิมจะหมุนเข้ามาในบิต 7
SWAP A	จำนวนไบต์: 1 การทำงาน: แลกเปลี่ยนข้อมูลระหว่างบิต 0-3 กับบิต 4-7 ของรีจีสเตอร์ A

2.4 กลุ่มคำสั่งแบบบูลีนหรือแบบบิต ซึ่งเป็นความสามารถของไมโครคอนโทรลเลอร์ MCS-51 ที่จะดำเนินการประมวลผลแบบบิต แทนที่จะเป็นข้อมูลทั้งไบต์เช่นปกติ โดยมีชุดคำสั่งที่จัดการโดยตรง ทุกคำสั่งจะเข้าถึงข้อมูลโดยตรงในระดับบิต โดยมีการบิตแอดเดรสได้ตั้งแต่ 00H – 7FH ในพื้นที่ 128 บิต หน่วยความจำข้อมูลภายในและบิตแอดเดรส 80H – FFH ในบริเวณกลุ่มรีจิสเตอร์ฟังก์ชั่นพิเศษ (SFR)

ANL C, bit	จำนวนไบต์: 2 การทำงาน: ทำการแอนด์ค่าของข้อมูลในแฟลกทด กับค่าของข้อมูลในระดับบิตของรีจีสเตอร์ แล้วนำผลลัพธ์ไปเก็บไว้ในแฟลกทด
ANL C, /bit	จำนวนไบต์: 2 การทำงาน: ทำการแอนด์ค่าของข้อมูลในแฟลกทด กับค่าคอมพลีเมนต์ ของข้อมูลในระดับบิต ของรีจีสเตอร์ โดยข้อมูลของรีจีสเตอร์ไม่มีการเปลี่ยนแปลงจากนั้นนำผลลัพธ์ไปเก็บไว้ใน แฟลกทด (ค่าคอมพลีเมนต์ คือค่าที่ตรงข้ามกับค่าของข้อมูล)
ORL C, bit	จำนวนไบต์: 2 การทำงาน: ทำการออร์ค่าของข้อมูลในแฟลกทด กับค่าของข้อมูลในระดับบิตของรีจีสเตอร์ แล้วนำผลลัพธ์ไปเก็บไว้ในแฟลกทด
ORL C, /bit	จำนวนไบต์: 2 การทำงาน: ทำการออร์ค่าของข้อมูลในแฟลกทด กับค่าคอมพลีเมนต์ของข้อมูลในระดับบิต ของรีจีสเตอร์ โดยข้อมูลของรีจีสเตอร์ไม่มีการเปลี่ยนแปลง จากนั้นนำผลลัพธ์ไปเก็บไว้ในแฟลกทด
CLR C	จำนวนไบต์: 1 การทำงาน: ทำการเคลียร์ค่าของแฟลกทดให้เท่ากับ "0"
CLR bit	จำนวนไบต์: 2 การทำงาน: ทำการเคลียร์ค่าของข้อมูลในบิตที่กำหนดให้เท่ากับ "0"
CPL C	จำนวนไบต์: 1 การทำงาน: ทำการคอมพลีเมนต์ หรือกลับสถานะลอจิกของแฟลกทด
CPL bit	จำนวนไบต์: 2 การทำงาน: ทำการคอมพลีเมนต์หรือกลับสถานะลอจิกของข้อมูลในบิตที่กำหนด
SETB C	จำนวนไบต์: 1 การทำงาน: ทำการเซตค่าของแฟลกทดให้เท่ากับ "1"
SETB bit	จำนวนไบต์: 2 การทำงาน: ทำการเซตค่าของข้อมูลในบิตที่กำหนดให้เท่ากับ "1"

2.5 กลุ่มคำสั่งในการกระโดดไปยังตำแหน่งต่างๆ ภายในโปรแกรม ซึ่งจะเปลี่ยนลำดับของการประมวลผลภายในโปรแกรมไปยังส่วนต่างๆแทนที่จะดำเนินการไปเป็นลำดับ ต่อเนื่องโดยที่คำสั่ง JMP จะแบ่งเป็น 3 ลักษณะ คือ SJMP, LJMP, AJMP ซึ่งในแต่ละคำสั่ง จะมีข้อแตกต่างของการกระโดดไปยังแอดเดรสไกลสุดที่ต่างกัน คำสั่ง JMP ซึ่งเป็นแบบโมนีซิก ที่สามารถจะใช้ได้โดยมีรายละเอียดการใช้งานของคำสั่งดังต่อไปนี้ SMP:จะเป็นการกระโดดแบบการย้ายอันดับตำแหน่งของแอดเดรสตำแหน่งเดิมซึ่งจะสามารถกระโดดได้ -128 ถึง +127 ไบต์ AJMP:ลักษณะแบบนี้จะสามารถกระโดดได้ไกลสุดประมาณ 2 กิโลไบต์ ซึ่งจะใช้หน่วยความจำเพียง 2 ไบต์เท่านั้นในการกำหนด LJMP:ลักษณะแบบนี้จะสามารถกระโดดได้ไกลสุดประมาณ 64 กิโลไบต์ ซึ่งจะใช้หน่วยความจำเพียง 3 ไบต์เท่านั้นในการกำหนด JMP @A+DPTR:เป็นการควบคุมการกระโดดไปยังโปรแกรมที่ต้องการเฉพาะภายในส่วนต่างๆ

SJMP rel	จำนวนไบต์: 2 การทำงาน: กำหนดให้ซีพียูมาทำงานยังแอดเดรสที่กำหนดด้วยค่าสัมพัทธ์(rel)
AJMP addr11	จำนวนไบต์: 2 การทำงาน: กำหนดให้ซีพียูมาทำงานยังแอดเดรสที่ระบุไว้ addr11 มีขอบเขต 2 กิโลไบต์ (000H-7FFH)
LJMP addr16	จำนวนไบต์: 3 การทำงาน: กำหนดให้ซีพียูมาทำงานยังแอดเดรสที่ระบุไว้ addr16 มีขอบเขต 64 กิโลไบต์ (0000H-FFFFH)
JMP @A+DPTR	จำนวนไบต์: 1 การทำงาน: กำหนดให้ซีพียูกระโดดไปยังแอดเดรส ของหน่วยความจำโปรแกรมในตำแหน่ง ที่ได้รับการกำหนดด้วยค่าของรีจีสเตอร์ A รวมกับค่าใน DPTR
NOP	จำนวนไบต์: 1 การทำงาน: เป็นคำสั่งที่ทำให้เกิดการเลื่อนแอดเดรสไปหนึ่งแอดเดรส
CJNE A, direct, rel	จำนวนไบต์: 2 การทำงาน: กำหนดให้ซีพียูกระโดดไปยัง แอดเดรสปลายทางตามค่าสัมพัทธ์(rel) เมื่อค่า ของรีจีสเตอร์ A ไม่เท่ากับค่าในหน่วยความจำข้อมูล
CJNE A, #data, rel	จำนวนไบต์: 2 การทำงาน: กำหนดให้ซีพียูกระโดดไปยัง แอดเดรสปลายทางตามค่าสัมพัทธ์(rel) เมื่อค่า ของรีจีสเตอร์ A ไม่เท่ากับค่าของ data
CJNE Rn, #data, rel	จำนวนไบต์: 3 การทำงาน: กำหนดให้ซีพียูกระโดดไปยัง แอดเดรสปลายทางตามค่าสัมพัทธ์(rel) เมื่อค่า ของรีจีสเตอร์ R0-R7 ไม่เท่ากับค่าของ data
CJNE @Rn, #data, rel	จำนวนไบต์: 3 การทำงาน: กำหนดให้ซีพียูกระโดดไปยัง แอดเดรสปลายทางตามค่าสัมพัทธ์(rel) เมื่อข้อ มูลในหน่วยความจำที่ชี้โดยรีจีสเตอร์ R0 หรือ R1 ไม่เท่ากับค่าของ data
DJNZ Rn, rel	จำนวนไบต์: 2 การทำงาน: กำหนดให้ซีพียูกระโดดไปยัง แอดเดรสปลายทางตามค่าสัมพัทธ์(rel) เมื่อทำ การลดค่าของรีจีสเตอร์ R0-R7 ลงหนึ่งค่า แล้วผลลัพธ์ไม่เท่ากับ "0"
DJNZ direct, rel	จำนวนไบต์: 3 การทำงาน: กำหนดให้ซีพียูกระโดดไปยัง แอดเดรสปลายทางตามค่าสัมพัทธ์(rel) เมื่อทำ การลดค่าของข้อมูลในหน่วยความจำที่กำหนดลงหนึ่งค่า แล้วผลลัพธ์ไม่เท่ากับศูนย์
ACALL addr11	จำนวนไบต์: 2 การทำงาน: กำหนดให้ซีพียูกระโดดไปยัง โปรแกรมย่อยซึ่งมีแอดเดรสอยู่ภายในขอบเขต สัมบูรณ์แบบใกล้ ซึ่งมีค่าเท่ากับ 2 กิโลไบต์ (000H-7FFH) และจะกลับมายังโปรแกรมหลักก็ต่อเมื่อพบคำสั่ง RET
LCALL addr16	จำนวนไบต์: 3 การทำงาน: กำหนดให้ซีพียูกระโดดไปยัง โปรแกรมย่อยซึ่งมีแอดเดรสอยู่ภายในขอบเขต สัมบูรณ์แบบใกล ซึ่งสามารถอ้างแอดเดรส ได้สูงสุด 64 กิโลไบต์ และจะกลับมายังโปรแกรมหลักก็ต่อเมื่อพบคำสั่ง RET

RET	จำนวนไบต์: 1 การทำงาน: กำหนดให้ฟียูกระโดดไปยัง โปรแกรมย่อยกลับไปยังโปรแกรมหลัก เป็นคำสั่ง สุดท้ายของทุกโปรแกรมย่อย ยกเว้นโปรแกรมย่อยบริการอินเตอร์รัปต์
RETI	จำนวนไบต์: 1 การทำงาน: กำหนดให้ฟียูกระโดดออกจาก โปรแกรมย่อยบริการอินเตอร์รัปต์กลับไปยัง โปรแกรมหลัก เป็นคำสั่งสุดท้ายของโปรแกรมย่อยการบริการอินเตอร์รัปต์
JB bit, rel	จำนวนไบต์: 3 การทำงาน: กำหนดให้ซีพียูกระโดดไปยัง แอดเดรสปลายทางตามค่าสัมพัทธ์(rel) เมื่อบิต ของรีจีสเตอร์ที่ทำการตรวจสอบเกิดการเซต ใช้ได้กับรีจีสเตอร์ที่สามารถเข้าถึงได้ในระดับบิต
JBC bit, rel	จำนวนไบต์: 3 การทำงาน: กำหนดให้ซีพียูกระโดดไปยัง แอดเดรสปลายทางตามค่าสัมพัทธ์(rel) เมื่อบิต ของรีจีสเตอร์ที่ทำการตรวจสอบเกิดการเซต ใช้ได้กับรีจีสเตอร์ที่สามารถเข้าถึงได้ในระดับบิต หลังจากกระโดดแล้วจะทำการเคลียร์บิตที่ทำการตรวจสอบนั้นให้เป็น "0"
JNB bit, rel	จำนวนไบต์: 3 การทำงาน: กำหนดให้ซีพียูกระโดดไปยัง แอดเดรสปลายทางตามค่าสัมพัทธ์(rel) เมื่อบิต ของรีจีสเตอร์ที่ทำการตรวจสอบไม่เกิดการเซต หรือกระโดดเมื่อบิตที่ทำการตรวจสอบนั้นเป็น "0" ใช้ได้กับรีจีสเตอร์ที่สามารถเข้าถึงได้ในระดับบิต
JNZ rel	จำนวนไบต์: 2 การทำงาน: กำหนดให้ซีพียูกระโดดไปยัง แอดเดรสปลายทางตามค่าสัมพัทธ์(rel) เมื่อค่า ของแอกคิวมูเลเตอร์หรือรีจีสเตอร์ A ไม่เป็น "0"
JZ rel	จำนวนไบต์: 2 การทำงาน: กำหนดให้ซีพียูกระโดดไปยัง แอดเดรสปลายทางตามค่าสัมพัทธ์(rel) เมื่อค่า ของแอกคิวมูเลเตอร์หรือรีจีสเตอร์ A เป็น "0"
JNC rel	จำนวนไบต์: 2 การทำงาน: กำหนดให้ซีพียูกระโดดไปยัง แอดเดรสปลายทางตามค่าสัมพัทธ์(rel) เมื่อค่าของแฟลกทด (C) ไม่เกิดการเซตหรือเป็น "0"
JC rel	จำนวนไบต์: 2 การทำงาน: กำหนดให้ซีพียูกระโดดไปยัง แอดเดรสปลายทางตามค่าสัมพัทธ์(rel) เมื่อค่าของแฟลกทด (C) ไม่เกิดการเซตหรือเป็น "1"

3. โครงสร้างการอินเตอร์รัปต์ของไมโครคอนโทรลเลอร์ MCS-51

ไมโครคอนโทรลเลอร์ MCS-51สัญญาณที่เข้ามาทำการอินเตอร์รัปต์ MCS-51 สามารถที่จะกำหนดเลือกเพื่อยินยอม (หรืออีนาเบิล : ENABLE) และห้าม (หรือดิสเอเบิล : DISABLE) ไม่ให้มีการอินเตอร์รัปต์แต่ละประเภทได้ โดยการกำหนดบิตของข้อมูลที่เกี่ยวข้องซึ่งมักจะอยู่ภายในรีจิสเตอร์ TCON และ SCON นอกจากนี้ยังมีตำแหน่งบิตภายในรีจิสเตอร์ IE (INTERRUPT ENABLE REGISTER) ซึ่งทำหน้าที่เสมือนกับเป็นสวิตซ์หลักที่เกี่ยวข้องกับสัญญาณอินเตอร์รัปต์ทั้งหมด หากว่ากำหนดไม่ให้เกิดการอินเตอร์รัปต์แล้วการกำหนดบิตเพื่อห้ามหรือยินยอมของแต่ละอินเตอร์รัปต์ก็จะไม่มีผลใดๆเกิดขึ้น ยังแสดงให้เห็นว่าสัญญาณอินเตอร์รัปต์แต่ละประเภทยังสามารถกำหนดระดับความสำคัญ (PRIORITY) ของการอินเตอร์-รัปต์ได้สองลักษณะ คือ ระดับความสำคัญสูงหรือต่ำ (HIGH OR LOW PRIORITY) กล่าวคือขณะที่กำลังประมวลผลอยู่ภายในส่วนของโปรแกรมย่อยบริการอินเตอร์รัปต์ของสัญญาณที่มีระดับความสำคัญต่ำอยู่ ก็อาจจะถูกขัดจังหวะให้ไปประมวลผลของสัญญาณอินเตอร์รัปต์ที่มีระดับความสำคัญสูงกว่า แต่หากว่าเป็นสัญญาณอินเตอร์รัปต์ที่มีระดับความสำคัญต่ำเช่นเดียวกันแล้ว ก็ต้องรอให้เสร็จสิ้นการประมวลผลที่ดำเนินการอยู่ก่อน

4. การรีเซต

การรีเซตโดยความหมายของการรีเซตเป็นการบังคับให้มีการเริ่มต้นใหม่อีกครั้งหนึ่ง ซึ่ง มักจะกระทำโดยการกำหนดสภาวะของสัญญาณที่ขารีเซตของไอซี MCS-51 ให้เป็นระดับลอจิก ที่เหมาะสมเท่านั้น การรีเซตด้วยวิธีนี้ถือว่าเป็นการอินเตอร์รัปต์อย่างหนึ่งได้ แต่จะมีลักษณะต่างออกไปจากการอินเตอร์รัปต์ของสัญญาณนี้ได้ ซึ่งมีศัพท์เฉพาะเรียกว่า NON-MASKABLE INTERRUPT นอกจากนี้การดำเนินการของโปรแกรมก็แตกต่างออกไปด้วย โดยจะไม่มีการเก็บค่าของคำสั่งที่กำลังจะไปทำในลำดับต่อไปภายในรีจิสเตอร์ PC เมื่อมีการรีเซตเกิดขึ้นโปรแกรม จะถูกสั่งให้กระโดดไปยังแอดเดรส 0000 ทันที ซึ่งตำแหน่งนี้จะเป็นตำแหน่งเริ่มต้นของการทำงานของไมโคร-คอนโทรลเลอร์ MCS-51 เมื่อเริ่มจ่ายไฟให้กับระบบเมื่อใดก็ตามที่มีการรีเซตเกิดขึ้นค่าสภาวะต่างๆ ภายในไมโครคอนโทรลเลอร์จะถูกกำหนดกลับไปเป็นค่าเริ่มต้นใหม่อีกครั้ง