# หน่วยที่ 3 ชุดคำสั่งและการเขียนโปรแกรม

---

อดิศักดิ์ ชิณะวงศ์

เอกสารประกอบการเรียนวิชาไมโครคอนโทรลเลอร์

เผยแพร่ที่ www.Adisak51.com

## 1. รหัสคำสั่งของไอซี MCS-51

รหัสคำสั่ง (Instruction Set) ของไอซี MCS-51 มีทั้งหมด 256 คำสั่งอยู่ในรูปแบบของเลขฐานสอง ประกอบด้วยคำสั่งขนาด 1 ไบต์จำนวน 139 คำสั่ง ขนาด 2 ไบต์จำนวน 92 คำสั่ง และขนาด 3 ไบต์ จำนวน 29 คำสั่ง มีรายละเอียดดังต่อไปนี้

**Rn** คือรีจิสเตอร์ R0 - R7 มีตำแหน่งแอดเดรสอยู่ที่ 00H - 1FH เมื่อไอซี MCS-51 ถูกรีเซตระบบ จะใช้งานรีจิสเตอร์แบงค์ 0 ตำแหน่งแอดเดรส 00H - 07H ซึ่งสามารถเรียกตำแหน่งแอดเดรส 00H-07H หรือชื่อรีจิสเตอร์ R0-R7 ได้โดยตรง

**@ RI** ใช้ได้กับรีจิสเตอร์ R0 หรือ R1 เท่านั้น การใช้งานให้นำข้อมูลในตำแหน่งของรีจิสเตอร์ R0 หรือ R1 เป็นแอดเดรสของหน่วยความจำข้อมูล หลังจากนั้นให้นำข้อมูลในตำแหน่งแอดเดรสนั้นไปใช้งาน

**#Data** คือข้อมูลค่าคงที่ขนาด 8 บิต เป็นโอเปอร์แรนด์ของคำสั่ง ถ้ามีตัวอักษรA - F เลขฐานสิบหก อยู่หลักหน้าให้เพิ่มเลข 0 ที่หลักหน้าด้วย เช่น MOV P1, #**0**F0H หากไม่เพิ่มโปรแกรมแอสเซมเบลอร์ จะฟ้องเป็นข้อผิดพลาด

**#Data16** คือข้อมูลค่าคงที่ขนาด 16 บิต หรือขนาด 2 ไบต์ (0000- FFFFH) เป็นโอเปอร์แรนด์ของคำสั่ง เช่น MOV DPTR, #0F000H

**Direct** คือค่าที่อยู่ในหน่วยความจำข้อมูล 256 ไบต์ ขณะใช้งานสามารถระบุตำแหน่งแอดเดรสในหน่วยความจำได้โดยตรง (00H – FFH)

**Rel** ค่าความสัมพัทธ์ที่มีค่าอยู่ในช่วง –127 ไบต์ ถึง +128 ไบต์ โดยแอดเดรสที่นับไปด้านหน้าใช้ค่าบวก และแอดเดรสนับถอยหลังใช้ค่าลบ

**Addr11** คือค่าแอดเดรสขนาด 11 บิต อ้างอิงหน่วยความจำแบบสัมพัทธ์ โดยตำแหน่งที่อ้างอิงอยู่ในช่วง 0 ถึง 2 กิโลไบต์ นับจากตำแหน่งที่ชี้โดยโปรแกรมเคาน์เตอร์

**Addr16** คือการอ้างอิงหน่วยความจำโดยใช้ค่าแอดเดรสแบบสัมบูรณ์ ขนาด 16 บิต สามารถอ้างอิงหน่วยความจำได้ถึง 64 กิโลไบต์

**Bit** คือการอ้างอิงข้อมูลที่เข้าถึงแบบบิตในหน่วยความจำข้อมูลภายใน หรือรีจิสเตอร์ใช้งานพิเศษ

**Bytes** คือจำนวนเนื้อที่ของหน่วยความจำโปรแกรม ที่ใช้เก็บคำสั่งให้กับไอซี MCS-51 โดยนำไปถอดรหัส และปฏิบัติตาม

**Cycle** คือรอบของการปฏิบัติตามคำสั่ง โดย 1 แมชชีนไซเคิลจะใช้จำนวน 12 สัญญาณพัลส์

## คำสั่งประมวลผลทางคณิตศาสตร์ (Arithmetic Operations)

| รูปแบบคำสั่ง | | ความหมาย | จำนวนไบต์ | แมชชีนไซเคิล |
|---|---|---|---|---|
| ADD | A,Rn | บวกค่าข้อมูลในรีจิสเตอร์ Rn กับข้อมูลในรีจิสเตอร์ A | 1 | 1 |
| ADD | A,direct | บวกค่าข้อมูลในหน่วยความจำ direct กับข้อมูลในรีจิสเตอร์ A | 2 | 1 |
| ADD | A,@RI | บวกค่าในหน่วยความจำที่อยู่ในรีจิสเตอร์A กับค่าข้อมูลที่ถูกชี้โดย Ri | 1 | 1 |
| ADD | A,#data | บวกค่าคงที่ 8 บิตกับข้อมูลในรีจิสเตอร์ A | 2 | 1 |
| ADCC | A,Rn | บวกค่าข้อมูลในรีจิสเตอร์ Rn กับข้อมูลในรีจิสเตอร์ A พร้อมแฟลกทด | 1 | 1 |
| ADDC | A,direct | บวกค่าข้อมูลในหน่วยความจำ direct กับข้อมูลในรีจิสเตอร์ A พร้อมแฟลกทด | 2 | 1 |
| ADDC | A,@Ri | บวกค่าในหน่วยความจำที่อยู่ในรีจิสเตอร์A กับค่าข้อมูลที่ถูกชี้ โดย Ri พร้อมแฟลกทด | 1 | 1 |
| ADDC | A,#data | บวกค่าคงที่ 8 บิตกับข้อมูลในรีจิสเตอร์ A พร้อมแฟลกทด | 2 | 1 |
| SUBB | A,Rn | ลบค่าข้อมูลในรีจิสเตอร์ Rn กับข้อมูลในรีจิสเตอร์ A พร้อมแฟลกยืม | 1 | 1 |
| SUBB | A,direct | ลบค่าข้อมูลในหน่วยความจำ direct กับข้อมูลในรีจิสเตอร์ A พร้อมแฟลกยืม | 2 | 1 |
| SUBB | A,@Ri | ลบค่าในหน่วยความจำที่อยู่ในรีจิสเตอร์A กับค่าข้อมูลที่ถูกชี้โดย Ri พร้อมแฟลกยืม | 1 | 1 |
| SUBB | A,#data | ลบค่าคงที่ 8 บิตกับข้อมูลในรีจิสเตอร์ A พร้อมแฟลกยืม | 2 | 1 |
| INC | A | เพิ่มค่าข้อมูลในรีจิสเตอร์ A = A+1 | 1 | 1 |
| INC | Rn | เพิ่มค่าข้อมุลในรีจิสเตอร์ Rn = Rn+1 | 1 | 1 |
| INC | direct | เพิ่มค่าข้อมูลในหน่วยความจำ direct | 2 | 1 |
| INC | @Ri | เพิ่มค่าข้อมูลในหน่วยความจำที่ชี้แอดเดรสโดยข้อมูลใน Ri | 1 | 1 |
| DEC | A | ลดค่าข้อมูลในรีจิสเตอร์ A = A-1 | 1 | 1 |
| DEC | Rn | ลดค่าข้อมุลในรีจิสเตอร์ Rn = Rn -1 | 1 | 1 |
| DEC | direct | ลดค่าข้อมูลในหน่วยความจำ direct | 2 | 1 |
| DEC | @Ri | ลดค่าข้อมูลในหน่วยความจำที่ชี้แอดเดรสโดยข้อมูลใน Ri | 1 | 1 |
| INC | DPTR | เพิ่มค่าข้อมูลในรีจิสเตอร์ DPTR = DPTR+1 | 1 | 2 |
| MUL | AB | คูณข้อมูลในรีจิสเตอร์ A กับข้อมูลในรีจิสเตอร์ B แล้วเก็บค่าข้อมูลไบต์สูงในรีจิสเตอร์ A ไบต์ต่ำไว้ในรีจิสเตอร์ B | 1 | 4 |
| DIV | AB | หารข้อมูลในรีจิสเตอร์ A กับข้อมูลในรีจิสเตอร์ B แล้วเก็บผลหารไว้ในรีจิสเตอร์ A เศษเก็บไว้ในรีจิสเตอร์ B | 1 | 4 |
| DA | A | ปรับค่าข้อมูลในรีจิสเตอร์ A ให้เป็นเลข BCD | 1 | 1 |

## คำสั่งทางตรรกะ (Logical Operations)

| รูปแบบคำสั่ง | | ความหมาย | จำนวนไบต์ | แมชชีนไซเคิล |
|---|---|---|---|---|
| ANL | A,Rn | AND ค่าข้อมูลในรีจิสเตอร์ Rn กับค่าข้อมูลรีจิสเตอร์ใน A | 1 | 1 |
| ANL | A,direct | AND ค่าข้อมูลในหน่วยความจำ direct กับค่าข้อมูลรีจิสเตอร์ใน A | 2 | 1 |
| ANL | A,@Ri | AND ค่าข้อมูลที่ชี้แอดเดรสโดย Ri กับค่าข้อมูลรีจิสเตอร์ใน A | 1 | 1 |
| ANL | A,#data | AND ค่าคงที่ 8 บิตกับค่าข้อมูลรีจิสเตอร์ใน A | 2 | 1 |
| ANL | direct,A | AND ค่าข้อมูลรีจิสเตอร์ใน A กับข้อมูลในหน่วยความจำ direct | 2 | 1 |
| ANL | direct,#data | AND ค่าคงที่ 8 บิตกับหน่วยความจำ direct | 3 | 2 |
| ORL | A,Rn | OR ค่าข้อมูลในรีจิสเตอร์ Rn กับค่าข้อมูลรีจิสเตอร์ใน A | 1 | 1 |
| ORL | A,direct | OR ค่าข้อมูลในหน่วยความจำ direct กับค่าข้อมูลรีจิสเตอร์ใน A | 2 | 1 |
| ORL | A,@Ri | OR ค่าข้อมูลที่ชี้แอดเดรสโดย Ri กับค่าข้อมูลรีจิสเตอร์ใน A | 1 | 1 |
| ORL | A,#data | OR ค่าคงที่ 8 บิตกับค่าข้อมูลรีจิสเตอร์ใน A | 2 | 1 |
| ORL | direct,A | OR ค่าข้อมูลรีจิสเตอร์ใน A กับหน่วยความจำ direct | 2 | 1 |
| ORL | direct,#data | OR ค่าคงที่ 8 บิตกับหน่วยความจำ direct | 3 | 2 |
| XRL | A, Rn | EX-OR ค่าข้อมูลในรีจิสเตอร์ Rn กับค่าข้อมูลรีจิสเตอร์ใน A | 1 | 1 |
| XRL | A,direct | EX-OR ค่าข้อมูลในหน่วยความจำ direct กับค่าข้อมูลรีจิสเตอร์ใน A | 2 | 1 |
| XRL | A,@Ri | EX-OR ค่าข้อมูลที่ชี้แอดเดรสโดย Ri กับค่าข้อมูลรีจิสเตอร์ใน A | 1 | 1 |
| XRL | A,#data | EX-OR ค่าคงที่ 8 บิตกับค่ามูลรีจิสเตอร์ใน A | 2 | 1 |
| XRL | direct,A | EX-OR ค่าข้อมูลรีจิสเตอร์ใน A กับหน่วยความจำ direct | 2 | 1 |
| XRL | direct,#data | EX-ORค่าคงที่ 8 บิตกับหน่วยความจำ direct | 3 | 2 |
| CLR | A | ทำค่าข้อมูลรีจิสเตอร์ใน A ให้เป็นศูนย์ | 1 | 1 |
| CPL | A | กลับค่าบิตข้อมูลในรีจิสเตอร์ A เป็นตรงข้ามทุกบิต | 1 | 1 |
| RL | A | หมุนบิตข้อมูลในรีจิสเตอร์ A ไปทางซ้าย 1 บิต | 1 | 1 |
| RLC | A | หมุนบิตข้อมูลในรีจิสเตอร์ A ไปทางซ้าย 1 บิต และบิต 0 เป็นค่าบิต ที่อยู่ในแฟลก ทด | 1 | 1 |
| RR | A | หมุนบิตข้อมูลในรีจิสเตอร์ A ไปทางขวา 1 บิต และบิต 0 เป็นค่าจากบิต7 | 1 | 1 |
| RRC | A | หมุนบิตข้อมูลในรีจิสเตอร์ A ไปทางขวา 1 บิต และค่าจากบิต 0 นำไปเก็บในแฟลก ทดและบิตที่อยู่ในแฟลกทดเดิมจะย้ายมาเก็บที่บิต 7 | 1 | 1 |
| SWAP | | สลับค่าข้อมูลสี่บิตบนกับสี่บิตล่างภายในรีจิสเตอร์ A | 1 | 1 |

## คำสั่งเคลื่อนย้ายข้อมูลของหน่วยความจำภายในไอซี (Data Transfer)

| รูปแบบคำสั่ง | | ความหมาย | จำนวนไบต์ | แมชชีนไซเคิล |
|---|---|---|---|---|
| MOV | A,Rn | ย้ายค่าข้อมูลในรีจิสเตอร์ Rn ไปเป็นข้อมูลของรีจิสเตอร์ A | 1 | 1 |
| MOV | A,direct | ย้ายข้อมูลจากหน่วยความจำ direct ไปเป็นค่าข้อมูลของรีจิสเตอร์A | 2 | 1 |
| MOV | A,@Ri | ย้ายข้อมูลจากหน่วยความจำที่ชี้แอดเดรสโดย Ri ไปเป็นข้อมูลในรีจิสเตอร์ A | 1 | 1 |
| MOV | A,#data | ย้ายค่าคงที่ 8 บิตไปเก็บเป็นข้อมูลในรีจิสเตอร์ A | 2 | 1 |
| MOV | Rn,A | ย้ายข้อมูลจากในรีจิสเตอร์ A ไปเป็นข้อมูลในรีจิสเตอร์ Rn | 1 | 1 |
| MOV | Rn,direct | ย้ายข้อมูลจากหน่วยความจำ direct ไป Rn | 2 | 2 |
| MOV | direct,A | ย้ายข้อมูลในรีจิสเตอร์ A ไปยังหน่วยความจำ direct | 2 | 1 |
| MOV | direct,Rn | ย้ายข้อมูลในรีจิสเตอร์ Rn ไปยังหน่วยความจำ direct | 2 | 2 |
| MOV | direct,direct | ย้ายข้อมูลระหว่างหน่วยความจำภายใน | 3 | 2 |
| MOV | direct,@Ri | ย้ายข้อมูลจากหน่วยความจำที่ชี้แอดเดรสโดย Ri ไปยังหน่วยความจำ Direct | 2 | 2 |
| MOV | direct,#data | ย้ายค่าคงที่ 8 บิตไปยังหน่วยความจำ direct | 3 | 2 |
| MOV | @Ri,A | ย้ายข้อมูลในรีจิสเตอร์ A ไปยังหน่วยความจำที่ชี้แอดเดรสโดย Ri | 1 | 1 |
| MOV | @Ri,direct | ย้ายข้อมูลจากหน่วยความจำ direct ไปยังหน่วยความจำที่ชี้แอดเดรสโดย Ri | 2 | 2 |
| MOV | @Ri,#data | ย้ายค่าคงที่ 8 บิตไปยังหน่วยความจำที่ชี้แอดเดรสโดย Ri | 2 | 1 |

## คำสั่งเคลื่อนย้ายข้อมูลของหน่วยความจำภายนอกไอซี (Data Transfer)

| รูปแบบคำสั่ง | | ความหมาย | จำนวนไบต์ | แมชชีนไซเคิล |
|---|---|---|---|---|
| MOV | DPTR,#data | ย้ายค่าคงที่ 16 บิตไปยัง DPTR | 3 | 2 |
| MOVC | A,@A+DPTR | ย้ายข้อมูลจากหน่วยความจำโปรแกรมภายนอกตำแหน่งที่ได้รับการกำหนดด้วยค่าของ A+ DPTR ไปยัง A | 1 | 2 |
| MOVC | A,@A+PC | ย้ายข้อมูลจากหน่วยความจำโปรแกรมภายนอกตำแหน่งที่ได้รับการกำหนดด้วยค่าของ A+ PC ไปยัง A | 1 | 2 |
| MOVX | A,@Ri | ย้ายข้อมูลจากหน่วยความจำที่เก็บอยู่ใน Ri ไปยัง A | 1 | 2 |
| MOVX | A,@DPTR | ย้ายข้อมูลจากหน่วยความจำที่เก็บอยู่ใน DPTR ไปยัง A | 1 | 2 |
| MOVX | @Ri,A | ย้ายข้อมูลที่เก็บอยู่ใน A ไปยังหน่วยความจำที่เก็บอยู่ใน Ri | 1 | 2 |
| MOVX | @DPTR,A | ย้ายข้อมูลที่เก็บอยู่ใน A ไปยังหน่วยความจำที่เก็บอยู่ใน DPTR | 1 | 2 |

## คำสั่งเคลื่อนย้ายข้อมูลของหน่วยความจำภายนอกไอซี (ต่อ)

| รูปแบบคำสั่ง | | ความหมาย | จำนวนไบต์ | แมชชีนไซเคิล |
|---|---|---|---|---|
| PUSH | direct | ย้ายข้อมูลจากหน่วยความจำ direct ไปเก็บยัง stack | 2 | 2 |
| POP | direct | ย้ายข้อมูลจาก stack ไปยังหน่วยความจำ direct | 2 | 2 |
| XCH | A,Rn | แลกเปลี่ยนข้อมูลระหว่าง A กับ Rn | 1 | 1 |
| XCH | A,direct | แลกเปลี่ยนข้อมูลระหว่างหน่วยความจำ direct กับ A | 2 | 1 |
| XCH | A,@Ri | แลกเปลี่ยนข้อมูลระหว่างหน่วยความจำที่เก็บอยู่ใน Ri กับ A | 1 | 1 |
| XCHD | A,@Ri | แลกเปลี่ยนข้อมูลสี่บิตล่างจากหน่วยความจำที่เก็บอยู่ใน Ri กับ A | 1 | 1 |

## คำสั่งจัดข้อมูลแบบบิต (Boolean Variable Manipulation)

| รูปแบบคำสั่ง | | ความหมาย | จำนวนไบต์ | แมชชีนไซเคิล |
|---|---|---|---|---|
| CLR | C | ทำให้ค่าแฟลกทดเป็น 0 | 1 | 1 |
| CLR | bit | กำหนดให้บิตเป็น 0 | 2 | 1 |
| SETB | C | กำหนดค่าในแฟลกทดให้เป็น 1 | 1 | 1 |
| SETB | bit | ให้บิตที่กำหนดเป็น 1 | 2 | 1 |
| CPL | C | กลับค่าบิตทดให้เป็นตรงกับข้าม | 1 | 1 |
| CPL | bit | กลับค่าบิตให้เป็นตรงกับข้าม | 2 | 1 |
| ANL | C,bit | AND ค่า bit กับแฟลกทด | 2 | 2 |
| ANL | C,/bit | AND ค่าตรงข้ามของ bit กับแฟลกทด | 2 | 2 |
| ORL | C,bit | ORLค่า bit กับแฟลกทด | 2 | 2 |
| ORL | C,/bit | ORL ค่าตรงข้ามของ bit กับแฟลกทด | 2 | 2 |
| MOV | C,bit | ย้ายค่า bit มายังแฟลกทด | 2 | 1 |
| MOV | bit,C | ย้ายค่าบิตทด มายัง bit | 2 | 2 |
| JC | rel | กระโดด ถ้าค่าแฟลก ทดเป็น 1 | 2 | 2 |
| JNC | rel | กระโดด ถ้าค่าแฟลก ทดเป็น 0 | 2 | 2 |
| JB | bit,rel | กระโดด ถ้าค่า bit เป็น 1 | 3 | 2 |
| JNB | bit,rel | กระโดด ถ้าค่า bit เป็น 0 | 3 | 2 |
| JBC | bit,rel | กระโดด ถ้าค่า bit เป็น 1 และเปลี่ยนค่า bit เป็น 0 | 3 | 2 |
| ACALL | addr 11 | ไปทำโปรแกรมย่อยตำแหน่งแอดเดรส 11 บิต | 2 | 2 |
| LCALL | addr 16 | ไปทำโปรแกรมย่อยตำแหน่งแอดเดรส 16 บิต | 3 | 2 |
| RET | | คำสั่งสิ้นสุดการทำงานโปรแกรมย่อย | 1 | 2 |
| RETI | | คำสั่งสิ้นสุดการทำงานโปรแกรมย่อยบริการอินเตอร์รัพท์ | 1 | 2 |

## คำสั่งควบคุมการทำงานโปรแกรม (Program and Machine Control)

| รูปแบบคำสั่ง | ความหมาย | จำนวนไบต์ | แมชชีนไซเคิล |
|---|---|---|---|
| AJMP addr 11 | กระโดดไปยังตำแหน่งจากค่าแอดเดรส 11 บิต | 2 | 2 |
| LJMP addr 16 | กระโดดไปยังตำแหน่งจากค่าแอดเดรส 16 บิต | 3 | 2 |
| SJMP rel | กระโดดไปยังตำแหน่งที่สัมพัทธ์กับตำแหน่งปัจจุบัน | 2 | 2 |
| JMP @A+DPTR | กระโดดไปยังตำแหน่งที่สัมพัทธ์กับค่าข้อมูลใน รีจิสเตอร์ A บวกกับค่าใน DPTR | 1 | 2 |
| JZ rel | กระโดดไปยังตำแหน่งที่สัมพัทธ์กับตำแหน่งปัจจุบัน ถ้าหากค่าข้อมูลในรีจิสเตอร์ A เป็น 0 | 2 | 2 |
| JNZ rel | กระโดดไปยังตำแหน่งที่สัมพัทธ์กับตำแหน่งปัจจุบัน ถ้าหากค่าข้อมูลในรีจิสเตอร์ A ไม่เป็น 0 | 2 | 2 |
| CJNE A,direct, rel | เปรียบเทียบค่าข้อมูลใน A กับหน่วยความจำ direct และ กระโดดไปยังตำแหน่งที่สัมพัทธ์กับตำแหน่งปัจจุบัน ถ้าค่าไม่เท่ากัน | 3 | 2 |
| CJNE A,#data,rel | เปรียบเทียบค่าข้อมูลใน A กับค่าคงที่ และกระโดดไปยัง ตำแหน่งที่สัมพัทธ์กับตำแหน่งปัจจุบัน ถ้าค่าไม่เท่ากัน | 3 | 2 |
| CJNE Rn,#data,rel | เปรียบเทียบค่าข้อมูลใน Rn กับค่าคงที่ และกระโดดไปยัง ตำแหน่งที่สัมพัทธ์กับตำแหน่งปัจจุบัน ถ้าค่าไม่เท่ากัน | 3 | 2 |
| CJNE @Ri,#data,rel | เปรียบเทียบค่าในหน่วยความจำที่เก็บใน Ri กับ ค่าคงที่และ กระโดดไปยังตำแหน่งที่สัมพัทธ์กับ ตำแหน่งปัจจุบัน ถ้าค่าไม่เท่ากัน | 3 | 2 |
| DJNZ Rn,rel | ลดค่าข้อมูลใน Rn 1 ค่า(Rn = Rn-1) และกระโดด ไปยังตำแหน่งที่สัมพัทธ์กับตำแหน่งปัจจุบัน ถ้าค่าไม่เป็น 0 | 2 | 2 |
| DJNZ direct,rel | ลดค่าในหน่วยความจำ direct 1 ค่า (direct = direct -1) กระโดดไปตำแหน่งที่สัมพัทธ์กับตำแหน่งปัจจุบัน ถ้าค่าไม่เป็น 0 | 3 | 2 |
| Nop | ไม่มีการทำงานใดๆเกิดขึ้น | 1 | 1 |

## 2. แอดเดรสซิ่งโหมด

คำสั่งที่เข้าถึงข้อมูลในหน่วยความจำตำแหน่งต่างๆ หรือการอ้างแอดเดรสของหน่วยความจำ เรียกว่าแอดเดรสซิ่งโหมด (Addressing Mode) แบ่งได้ดังนี้

**2.1 การอ้างแอดเดรสของหน่วยความจำแบบทันทีทันใด (Immediate Addressing Mode)**

เป็นการนำค่าคงที่ไปเก็บไว้เป็นข้อมูล ของตำแหน่งในหน่วยความจำ หรือรีจิสเตอร์นั้นๆ เครื่องหมายที่ต้องระบุไว้นำหน้าของค่าคงที่คือ “#”

**ตัวอย่าง** MOV A, #0FH ; นำค่าคงที่ 0F เก็บไว้ที่รีจีสเตอร์A (A = 00001111B)

MOV P1, #0FFH ; นำค่าคงที่ FF เก็บไว้ที่พอร์ต P1 (P1 = 11111111B)

MOV R1, #0FH ; นำค่าคงที่ 0F เก็บไว้ที่รีจีสเตอร์ R1 (R1 = 00001111B)

MOV 25H, #7FH ; นำค่าคงที่ 7F เก็บไว้หน่วยความจำตำแหน่งแอดเดรส 25H

**2.2 การอ้างแอดเดรสของหน่วยความจำแบบโดยตรง (Direct Addressing Mode)**

เป็นการอ้างถึงข้อมูลโดยระบุตำแหน่งแอดเดรสของหน่วยความจำโดยตรง

**ตัวอย่าง** MOV 0E0H, 20H ; นำข้อมูลในหน่วยความจำตำแหน่งแอดเดรส 20H ไปเก็บที่ตำแหน่งแอดเดรส E0H ซึ่งเป็นของรีจีสเตอร์ A ในรีจิสเตอร์ใช้งานพิเศษ ดังนั้นถ้าหากใช้คำสั่ง MOV A, 20H โดยได้ผลลัพธ์เช่นเดียวกัน แต่คำสั่ง MOV 0E0H, 20H ใช้เนื้อที่หน่วยความจำขนาด 3 ไบต์ ใช้เวลาทำคำสั่ง 3 แมชชีนไซเคิล แต่คำสั่ง MOV A, 20H ใช้เนื้อที่หน่วยความจำ 2 ไบต์ และใช้เวลา 1 แมชชีนไซเคิล

**ตัวอย่าง** MOV 30H, 01H ; นำข้อมูลในหน่วยความจำตำแหน่งแอดเดรส 01H ไปเก็บไว้ในหน่วยความจำตำแหน่งแอดเดรส 30H

**2.3 การอ้างแอดเดรสของหน่วยความจำแบบรีจิสเตอร์ (Register Addressing Mode)**

เป็นการอ้างถึงข้อมูลโดยระบุตำแหน่งแอดเดรสของหน่วยความจำที่เป็นรีจิสเตอร์ R0-R7 แต่คำสั่งการโอนย้ายข้อมูลระหว่างรีจิสเตอร์กับรีจิสเตอร์ จะไม่มีในคำสั่งของไอซี MCS-51 ดังนั้น จึงต้องใช้ค่าตำแหน่งแอดเดรสแทน ซึ่งเป็นการอ้างแอดเดรสของหน่วยความจำแบบ Direct

**ตัวอย่าง** MOV A, R1 ; นำข้อมูลในรีจิสเตอร์ R1 ไปเก็บไว้ที่รีจิสเตอร์ A

MOV 01H, 07H ; เป็นการนำข้อมูลในรีจิสเตอร์ R7 ไปเก็บไว้ที่รีจิสเตอร์ R1

**2.4 การอ้างแอดเดรสของหน่วยความจำแบบโดยอ้อม (Indirect Addressing Mode)**

เป็นการอ้างถึงข้อมูลแบบทางอ้อมใช้วิธีระบุตำแหน่งของแอดเดรสผ่านรีจิสเตอร์ R0 และ R1 ต้องใช้เครื่องหมาย @ นำหน้า R0 หรือ R1

**ตัวอย่าง** กำหนดให้ 20H = 76H

MOV R0,#20H ; เป็นการนำค่าคงที่ 20H เก็บไว้ที่รีจีสเตอร์ R0 (R0 = 20H)

MOV A,@R0 ; เป็นการนำค่าข้อมูลในตำแหน่งแอดเดรสที่ถูกชี้โดย R0

ไปเก็บที่รีจิสเตอร์ A ข้อมูลในรีจิสเตอร์ R0 มีค่า 20H และใน 20H มีค่า 76 H ดังนั้นรีจิสเตอร์ A จึงมีค่าเท่ากับ 76H

**2.5 การอ้างแอดเดรสของหน่วยความจำแบบบิต (Bit Addressing Mode)**

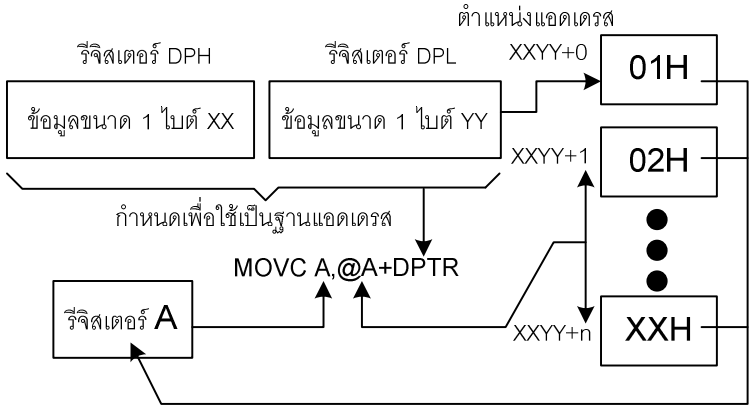
เป็นการอ้างตำแหน่งหน่วยความจำที่เข้าถึงได้ในระดับบิต

**ตัวอย่าง** SETB 25H ; หมายถึงการเซตบิตของตำแหน่งในหน่วยความจำที่เข้าแบบบิตตำแหน่ง 25H หรือตำแหน่งแอดเดรส 24H บิตที่ 5 อาจใช้คำสั่งเป็น SETB 24H.5 หมายถึงตำแหน่งบิต 25H

**ตัวอย่าง** MOV C, 20H ; หมายถึงการโอนย้ายข้อมูลในตำแหน่งหน่วยความจำแบบบิตตำแหน่งบิต 20H หรือ 24H.0 ไปที่แฟลกตัวทด (Carry Flag) ซึ่งแฟลกตัวทดมีเพียงบิตเดียวอยู่ตำแหน่งบิต ที่ 7 ของรีจิสเตอร์ PSW ตำแหน่งหน่วยความจำแอดเดรสที่ 20H เป็นหน่วยความจำขนาด 8 บิตไม่สามารถนำไปเก็บไว้ที่แฟลกตัวทดซึ่งมีขนาดเพียง 1 บิตได้ ดังนั้นในคำสั่งนี้ความหมาย ของ 20H คือตำแหน่งของบิตที่ 20H ในหน่วยความจำตั้งแต่แอดเดรส 20H-30H โดยแบ่งออกเป็นตำแหน่งบิตที่ 00H-FFH ตำแหน่งบิต 20H คือตำแหน่งของหน่วยความจำแอดเดรสที่ 24H.0

**2.6 การอ้างแอดเดรสของหน่วยความจำแบบฐานแอดเดรส (Index Addressing Mode)**

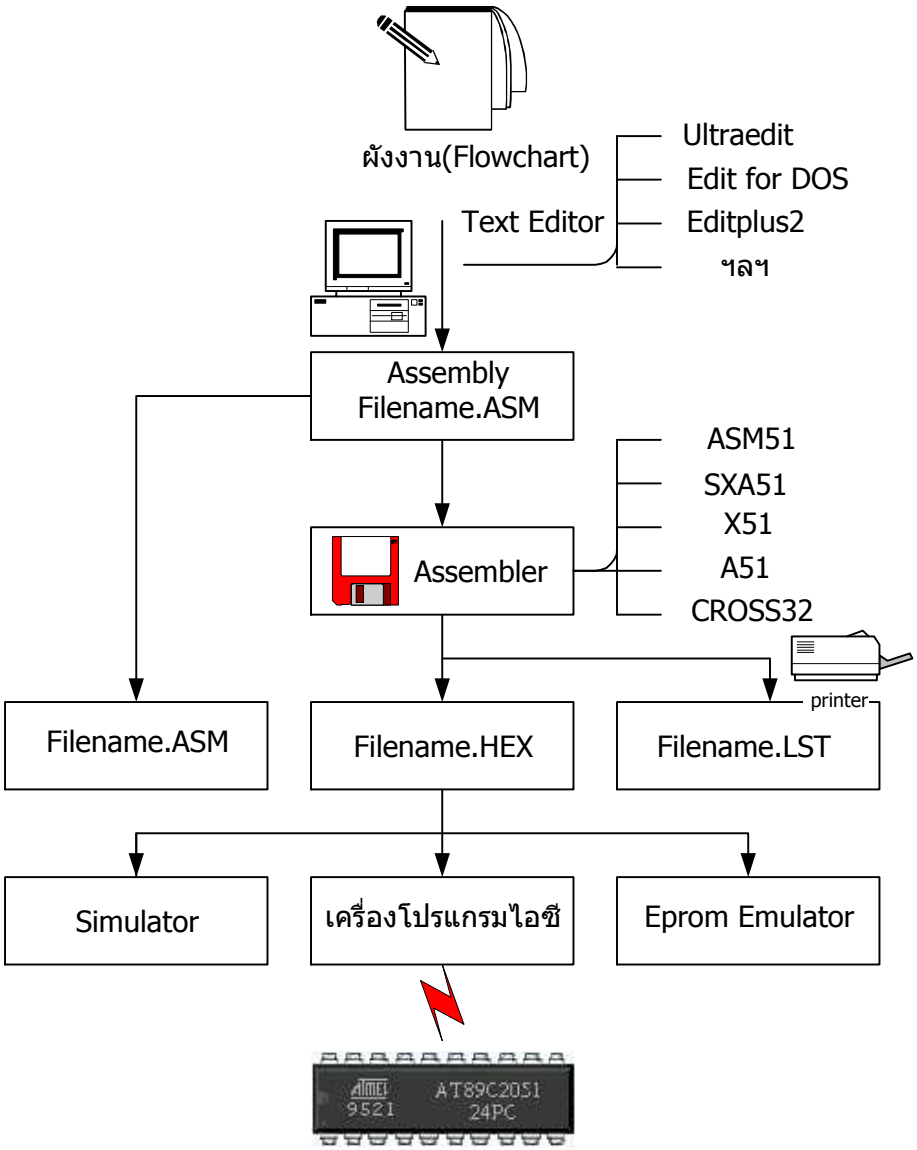
เป็นคำสั่งเคลื่อนย้ายข้อมูลผ่านรีจิสเตอร์ที่กำหนดให้เป็นฐานแอดเดรส หรืออินเด็กซ์ ซึ่งใช้งานเพียงสองคำสั่งคือ MOVC A,@A+DPTR และ MOVC A,@A+PC ทั้งสองคำสั่ง จะใช้กับหน่วยความจำแบบโปรแกรมหรือ Code Memory ( MOV**C** : Code Memory) ใช้รีจิสเตอร์ DPH และรีจิสเตอร์ DPL รวมกันให้เป็นรีจิสเตอร์ขนาด 16 บิต เรียกว่า DPTR ส่วนรีจิสเตอร์ PC หรือ โปรแกรมเคาน์เตอร์ เป็นรีจิสเตอร์ขนาด 16 บิต ทำหน้าที่ชี้ตำแหน่งแอดเดรสในการทำคำสั่ง ของซีพียู หากทำการ รีเซตไอซี MCS-51 จะมีค่าอยู่ที่แอดเดรส 0000H ในโหมดอินเด็กซ์ใช้สำหรับการอ่านค่าข้อมูลแบบตาราง (Lookup Table)โดยใช้งานรีจิสเตอร์ DPTR หรือรีจิสเตอร์โปรแกรมเคาน์เตอร์ เป็นฐานแอดเดรสตำแหน่งเริ่มต้นของตารางแสดงดังภาพที่ 3.1




**ภาพที่ 3.1** แสดงการอ้างแอดเดรสของหน่วยความจำแบบฐานแอดเดรส

## 3. โครงสร้างโปรแกรมภาษาแอสเซมบลี

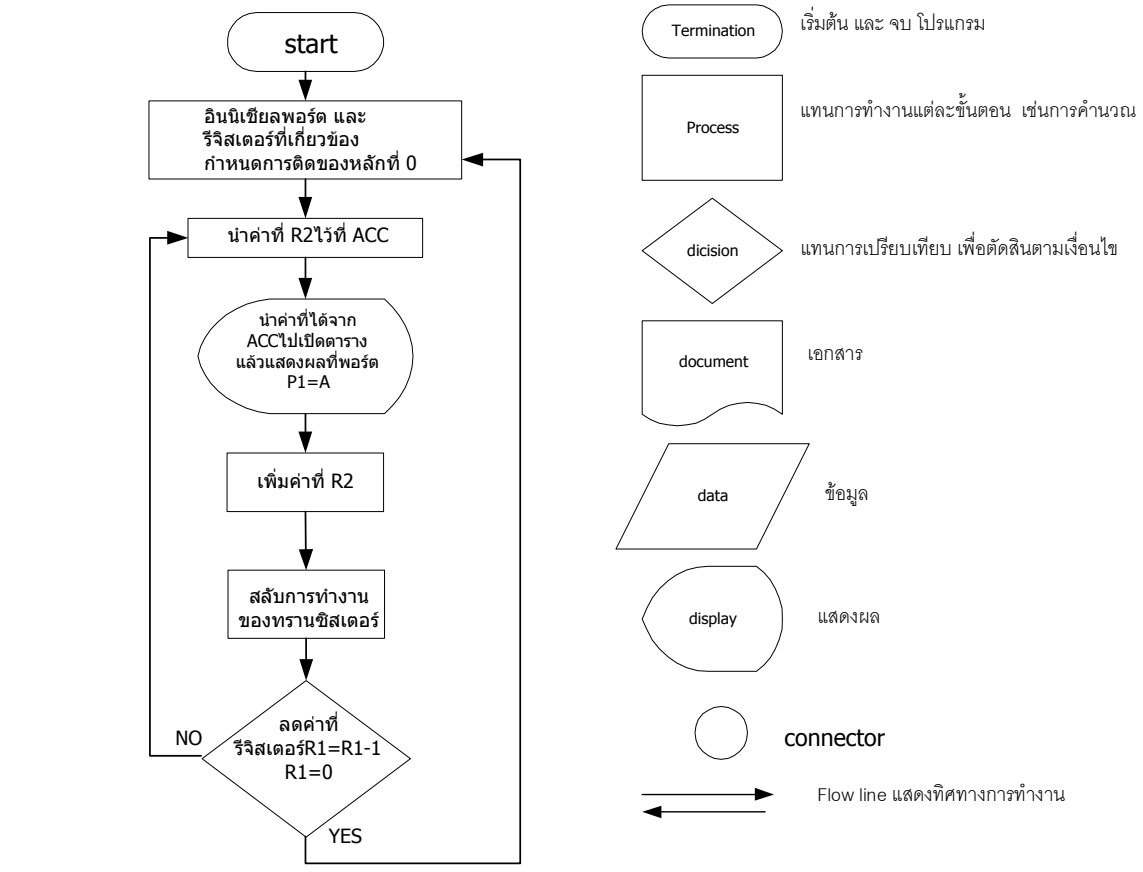
โปรแกรมจะถูกเก็บไว้ในหน่วยความจำ เป็นเลขไบนารี่ที่เรียกว่า **ภาษาเครื่อง** (Machine Language) เป็นภาษาที่ติดต่อให้คอมพิวเตอร์เข้าใจได้ การเปลี่ยนภาษาเครื่องให้เป็นเลขฐานสิบหก (HEX) เช่น คำสั่งขนาด 8 บิต 11101011B (B-ไบนารี่) เขียนได้เป็น EBH (H-ฐานสิบหก) ซึ่งทำให้เข้าใจได้ยากดังนั้น จึงใช้สัญลักษณ์ (Symbols) เรียกว่า **นีโมนิค** (Mnemonics) เพื่อแทนความหมายของคำสั่ง เช่น MOV A, #67H (ให้นำข้อมูลค่าคงที่ 67H เก็บไว้เป็นข้อมูลของรีจิสเตอร์ A) โปรแกรมที่เขียนด้วยรหัสนีโมนิค เรียกว่า **ภาษาแอสเซมบลี** (Assembly) และไอซีไมโครคอนโทรลเลอร์ จะสามารถทำงานตามโปรแกรมที่เขียนด้วยภาษาแอสเซมบลีได้ ต้องเปลี่ยนให้เป็นภาษาเครื่อง เรียกว่า **การแอสเซมเบลอร์** (Assembler) โดยมีขั้นตอนแสดงดังภาพที่ 3.2 โครงสร้างของโปรแกรมภาษาแอสเซมบลีประกอบด้วย



**ภาพที่ 3.2** แสดงขั้นตอนการเขียนโปรแกรม เพื่อนำข้อมูลบันทึกลงไอซี

### 3.1 ผังงาน (Flowchart)

ใช้แสดงลักษณะ และอัลกอริทึมของโปรแกรมในรูปแบบผังกราฟที่เหมาะสม เป็นการลำดับงานแบบขั้นตอน สามารถทำความเข้าใจความต้องการของผู้เขียนโปรแกรมสำหรับผู้อื่นได้ดี ทำให้ไม่เกิดการซ้ำซ้อนในการเขียนโปรแกรม รูปแบบของผังงาน และตัวอย่างผังงานแสดงดังภาพที่ 3.3




**ภาพที่ 3.3** แสดงรูปแบบของผังงาน

### 3.2 สเตทเมนท์ (Statement)

ประกอบด้วย 4 ส่วน (Field) คือ Label, Op-code, Operand และ Comment รูปแบบของโปรแกรมภาษาแอสเซมบลี การเขียนแต่ละคำสั่งเรียกว่า สเตทเมนท์ (Statement) เช่น

| Label | Op-code & | Operand | Comment |
|---|---|---|---|
| | ORG | 0000H | |
| LOOP1: | MOV | A,#0FFH | ; REG A= 11111111B |
| | MOV | P1,A | |
| | ACALL | DELAY | |

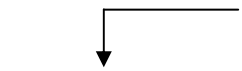
XX: เครื่องหมาย COLON

; เครื่องหมาย Semicolon
Comment

```
MOV     P1,#00000000B
ACALL   DELAY
SJMP    LOOP
```

3.2.1 เลเบล (Label) ต้องขึ้นต้นด้วยตัวอักษร **"A - Z"** หรือเครื่องหมาย **"_", "."** ตัวอักษรจาก A ถึง Z ตัวเลขจาก **0 ถึง 9** เลเบลเขียนจำนวนตัวอักษร เมื่อรวมทั้งบรรทัดต้องไม่เกิน 255 ตัวอักษร การเขียนเลเบลจะต้องมีเครื่องหมาย **" : "** (Colon) ตามหลังเลเบลด้วยทุกครั้ง เพื่ออ้างตำแหน่งในบรรทัดของโปรแกรม เช่น LOOP:

3.2.2 ออปโค๊ด (Op-code) เป็นส่วนของนีโมนิค แสดงจุดมุ่งหมายของการกระทำแต่ละคำสั่ง เช่น MOV ADD SUB ในช่องห่างระหว่าง Op-Code และโอเปอร์แรนด์จะต้องมีช่องว่างคั่นอยู่ อย่างน้อย 1 ช่อง เช่น




**MOV A, R1**

3.2.3 โอเปอร์แรนด์ (Operand) แสดงถึงส่วนที่ถูกกระทำ ประกอบด้วย 2 ส่วนที่มีการคั่นด้วยเครื่องหมาย "," (จุลภาค: Comma) หรือการแสดงถึงตำแหน่งที่จะกระทำ

3.2.4 หมายเหตุ (Comment) เป็นการเขียนคำอธิบายให้แต่ละบรรทัด หรือหมายเหตุของโปรแกรมในส่วนต่างๆ ต้องใช้เครื่องหมาย **";"** (Semi - Colon) นำหน้าทุกครั้ง เพราะโปรแกรมแอสเซมเบลอร์จะไม่แปล หากพบเครื่องหมาย **";"** ในส่วนของแรกของบรรทัด

### 3.3 แอสเซมเบลอร์ไดเร็กทีฟ (Assembler Directives)

แอสเซมเบลอร์ไดเร็กทีฟ หรือคำสั่งเทียม (Pseudo Instruction) ใช้กำหนดสัญลักษณ์ การแปลความหมาย การจองพื้นที่ของหน่วยความจำทั้งแบบไบต์ และแบบบิต ใช้บอกจุดเริ่มต้นของโปรแกรม และจุดสิ้นสุดของโปรแกรม โดยเป็นคำสั่งของโปรแกรมแอสเซมเบลอร์แต่ละตัว จะมีลักษณะคำสั่งที่คล้ายๆกัน ซึ่งอธิบายหัวข้อที่ใช้งานดังนี้

**ORG** (Program Counter Origin) เป็นไดเร็กทีพที่ให้ผู้ใช้กำหนดค่าแอดเดรสเริ่มต้นในโปรแกรมเคาน์เตอร์

**ตัวอย่าง** ORG 0000H ; ระบุให้ไมโครคอนโทรลเลอร์เริ่มต้น ทำคำสั่งที่ตำแหน่งแอดเดรส 0000H ของหน่วยความจำโปรแกรม

**END** (End Of Source Program) เป็นไดเร็กทีพที่บอกให้โปรแกรมแอสเซมเบลอร์ รู้ว่าถึงจุดสิ้นสุดของโปรแกรมที่เขียนแล้ว

**EQU** (Equate Label) เป็นไดเร็กทีพที่ใช้กำหนดค่าให้กับสัญลักษณ์ แทนตำแหน่งแอดเดรส หรือค่าคงที่ของข้อมูล หรือการกำหนดค่าอินทีเจอร์ ให้มีค่าเท่ากับ เลเบล

**ตัวอย่าง** SEC EQU 23H ; กำหนดให้ SEC มีค่าเท่ากับข้อมูลในตำแหน่งหน่วยความจำแอดเดรสที่ 23H และเมื่อนำไปใช้งานถ้าเรากำหนดชื่อ SEC ในส่วนของโปรแกรมก็จะหมายถึงแอดเดรสที่ 23H นั้นเอง

**BIT** (Symbol Name Bit Expression) เป็นไดเร็กทีพที่ใช้กำหนดชื่อแทนค่าแอดเดรสตำแหน่งหน่วยความจำที่เข้าแบบบิต ในกรณีที่มีค่าอยู่หน่วยความจำตำแหน่งแอดเดรส 20H – 2FH (20H.0 – 2FH.7 หรือตำแหน่งของบิตที่ 00-FFH) ซึ่งหมายถึงตำแหน่งที่อยู่พื้นที่ของแรมภายในสามารถเข้าแบบบิตได้ แต่ถ้ามีค่าอยู่ระหว่าง 80H-FFH จะหมายถึงตำแหน่งบิตใน รีจิสเตอร์ใช้งานพิเศษที่สามารถเข้าถึงแบบบิตได้

**ตัวอย่าง** SW_TEST BIT P3.0 ; กำหนดให้ BIT P3.0 ในแอดเดรส B0H.0 ของรีจิสเตอร์ใช้งานพิเศษมีชื่อแทนตำแหน่งเป็น SW_TEST

**DB** (Define Byte) เป็นไดเร็กทีพที่ยอมให้ผู้ใช้กำหนดค่าของข้อมูลในหน่วยความจำตามต้องการ โดยจะใส่ค่าคงที่ขนาดเป็นไบต์ลงในหน่วยความจำโปรแกรม สามารถกำหนดเป็น ตัวเลข สูตรคณิตศาสตร์ สัญลักษณ์ หรือรหัสแอสกี้ (ASCII)

**ตัวอย่าง** DATA: DB “MICROCONTROLLER”, CR, LF ; ASCII String

DISP: DB 0F0H, 3FH, 44H, 02H ; Hexadecimal

**DS** (Define Storage) เป็นไดเร็กทีพที่ยอมให้ผู้ใช้จองเนื้อที่ส่วนหนึ่งภายในหน่วยความจำข้อมูล

**ตัวอย่าง** BUFFER: DS 5 ; จองเนื้อที่ในหน่วยความจำแรมขนาด 5 ไบต์

## 4. โปรแกรมแอสเซมเบลอร์

แอสเซมเบลอร์ (Assembler Program) คือการแปลงรหัสข้อมูล ของโปรแกรมที่ถูกเขียนขึ้นโดยภาษาแอสเซมบลีให้เป็นภาษาเครื่อง เพื่อบันทึกข้อมูลคำสั่งให้กับไอซีไมโครคอนโทรลเลอร์ โปรแกรมทีใช้ในการแอสเซมเบลอร์ เช่น SXA51, CROSS32 ASM51 และ RAD51

### 4.1 โปรแกรมแอสเซมเบลอร์ SXA51

SXA51 เป็นโปรแกรมแอสเซมเบลอร์ ใช้สำหรับไอซีไมโครคอนโทรลเลอร์ตระกูล MCS-51 โดยเขียนโปรแกรมต้นฉบับ (Source Program) ด้วยโปรแกรมเท็กซ์เอดิเตอร์ (Text Editor Program) บันทึกเป็นไฟล์แอสเซมบลี (Assembly File) หลังจากนั้นโปรแกรม SXA51 จะทำการแปลงให้เป็นเลขฐานสิบหกสามารถนำไปบันทึกให้กับไอซีหน่วยความจำโปรแกรมได้ ตัวอย่างการใช้งานโปรแกรม **SXA51** มีดังนี้

**C:\ > SXA51 - <Option> File name. ASM <CR> - <OPTION>** แสดงตัวเลือกดังรายการ

-L = ให้สร้างไฟล์แสดงรายละเอียด Address Op-code โดยจะได้ไฟล์นามสกุล .LST

-N = ไม่สร้างไฟล์เอาต์พุตใดๆ เพื่อความรวดเร็ว

-C = สร้าง Symbol Cross-Reference ไว้ที่ท้ายของโปรแกรมเพื่อความสะดวกของการหาตำแหน่งของเลเบล (LABLE)

-D = แสดง Process ขณะที่กำลังทำการแอสเซมบลี โดยโปรแกรมจะแสดงเลขบรรทัดที่กำลังแปลอยู่

ถ้าไม่ใช้ทางเลือกใดโปรแกรม SXA51 จะสร้างแต่ไฟล์เลขฐานสิบหก ในกรณีเช่นนี้ถ้าหากมีรหัสผิดพลาด (Error Code) จะแสดงออกมาทางจอภาพ แต่ถ้าเลือกทางเลือก - L เมื่อมีรหัสผิดพลาดจะถูกสร้างขึ้นใน Listing file และใช้โปรแกรมเท็กซ์เอดิเตอร์ตรวจสอบรายละเอียดของข้อผิดพลาดได้

**4.2 โปรแกรมแอสเซมเบลอร์ Cross32**

โปรแกรม Cross32 เป็นโปรแกรมครอส - แอสเซมเบลอร์ (Cross - Assembler) โดยจะทำการตรวจสอบสองครั้ง (Two Pass) และมีทางเลือกแบบตรวจสอบครั้งที่สาม (Third Pass) ถ้าหากตรวจสอบพบข้อผิดพลาดในโปรแกรมต้นแบบ โปรแกรม Cross32 สามารถใช้ได้กับซีพียูในหลายตระกูล ดังนั้น การใช้งานโปรแกรมจึงต้องมีการระบุเบอร์ของไอซีโดยคำสั่ง CPU Directive ไว้ในส่วนหัวของโปรแกรม เช่น CPU “8051.TBL” เป็นการระบุให้โปรแกรม Cross32 ใช้ตารางคำสั่งของ CPU เบอร์8051 และต้องกำหนดรูปแบบของไฟล์แสดงรหัสภาษาเครื่อง โดยคำสั่ง HOF Directive เพื่อให้โปรแกรมครอส 32 ต้องวางไฟล์ตามมาตรฐานใด เช่น HOF “INT8”เป็นการกำหนดให้วางไฟล์ในรูปแบบตามมาตรฐานของบริษัทอินเทล เป็นต้น

การเขียนโปรแกรมจะใช้เทกซ์ เอดิเตอร์ สร้างโปรแกรมต้นแบบ โดยพิมพ์ที่ส่วนหัวของโปรแกรมที่มีรายละเอียดดังนี้

```
CPU “8051.TBL”          ; ระบุตารางคำสั่งของไมโครคอนโทรลเลอร์เบอร์ 8051
HOF “INT8”              ; กำหนดรูปแบบของไฟล์แสดงรหัสมาตรฐานอินเทล
ORG 0000H               ; ระบุตำแหน่งแอดเดรสในหน่วยความจำเพื่อที่จะให้
                        ; เริ่มต้นการทำงานของโปรแกรม
    .
    .                   ; ส่วนของโปรแกรมที่เป็นคำสั่ง (แอสเซมบลี)
    .
END                     ; จบโปรแกรม
```

รูปแบบของคำสั่งที่ใช้งานเป็นดังนี้

**C32 Filename.asm [-L LISTFILE] [-H HEXFILE)**

โดยส่วนที่อยู่ภายใน [ ] เป็นทางเลือก จะใช้หรือไม่ใช้ก็ได้

Filename.asm หมายถึง โปรแกรมต้นแบบ (Source Program) ที่เขียนขึ้นโดยมีนามสกุลเป็นจุด ASM (Assemble)

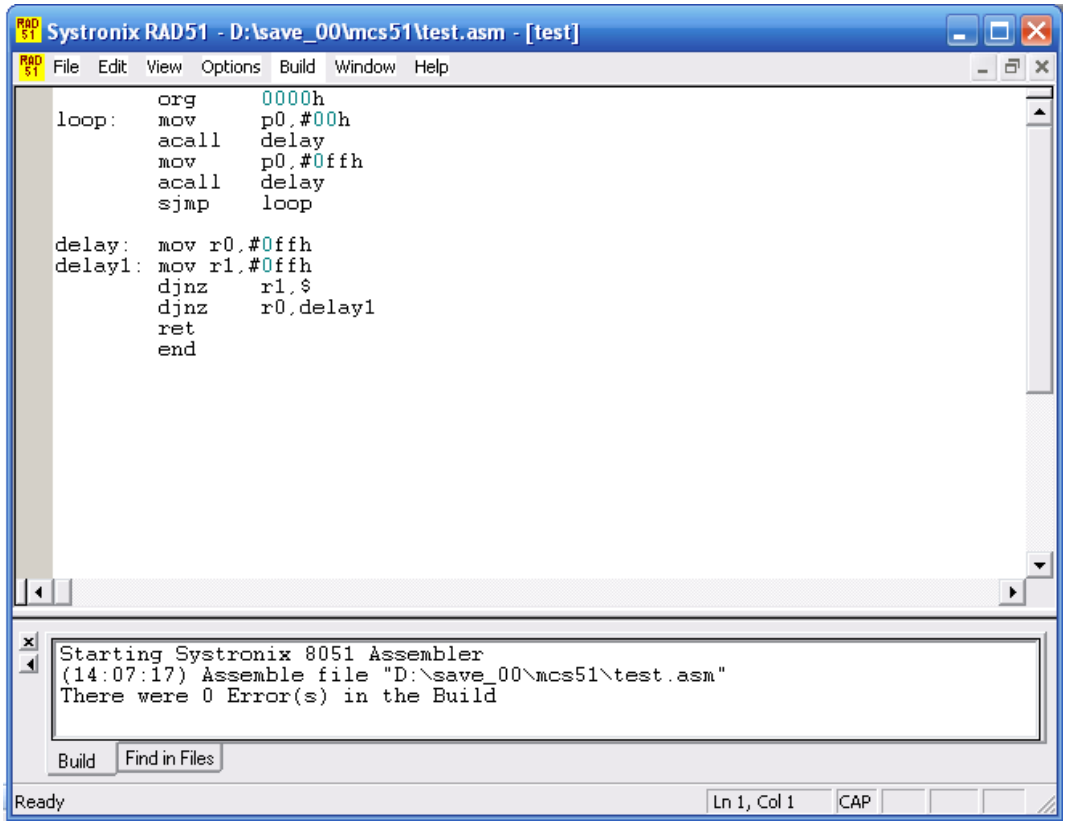
- L List File หมายถึง ให้ Cross32 สร้างไฟล์แสดงรายละเอียดการแอสเซมเบลอร์ (Assemble Listing File) โดยใช้ชื่อไฟล์ที่ตามหลังมา (ในที่นี้คือ List File)

- H Hex File หมายถึง ให้ Cross32 สร้างไฟล์รหัสภาษาเครื่อง (Machine Code) ออกมาในไฟล์ที่ ตามหลังมา (ในที่นี้คือ hex file)

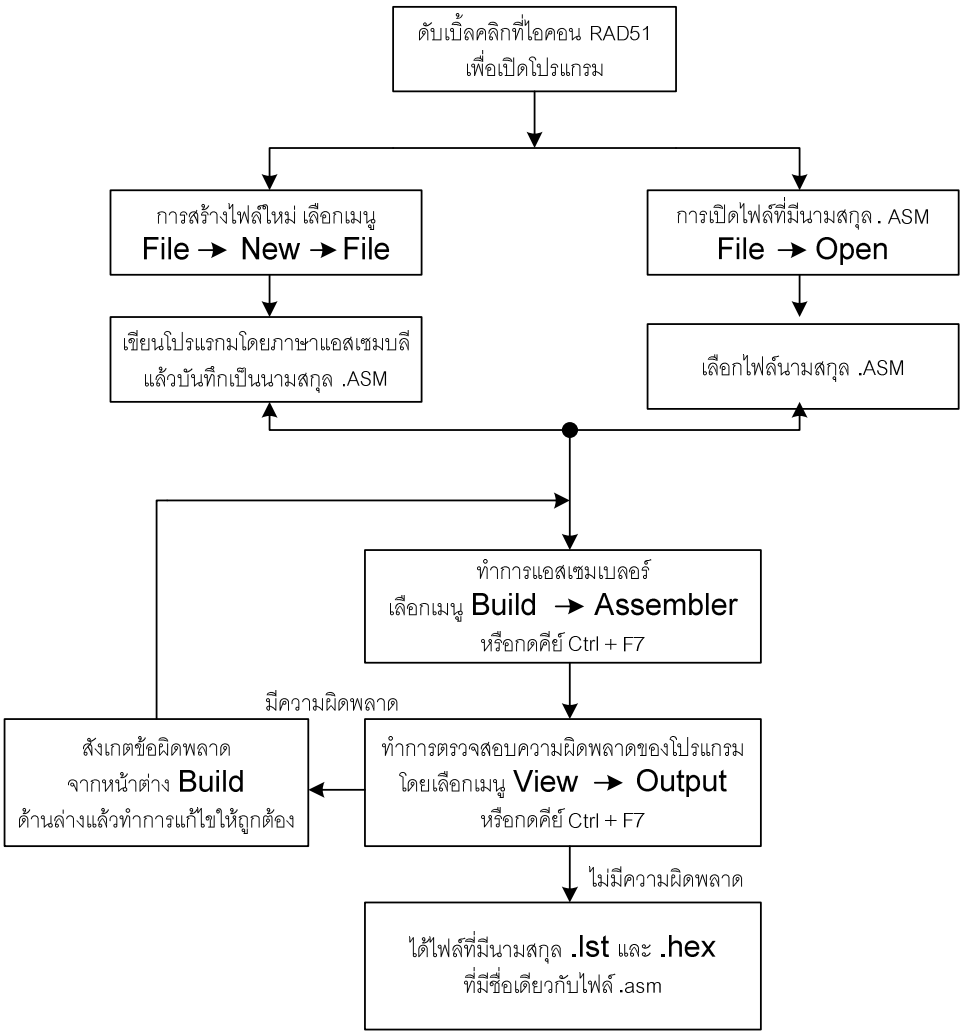
ถ้าไม่ใช้ทางเลือก - L หรือ - H โปรแกรม Cross32จะไม่สร้างไฟล์เหล่านี้ ขึ้นมา ในกรณีเช่นนี้ รหัสที่แสดงการผิดพลาด (Error Code) จะแสดงออกมาจอภาพ (ถ้ามี) แต่ถ้าเลือกทางเลือก - L แล้ว รหัสผิดพลาดจะถูกสร้างขึ้นใน List File ด้วย

### 4.3 โปรแกรมแอสเซมเบลอร์ RAD51

RAD51 (Rapid Application Development Environment and 8051 Cross Assembler) เป็นโปรแกรมที่รวมเท็กซ์เอดิเตอร์ และโปรแกรมแอสเซมเบลอร์ไว้ด้วยกัน ทำให้สะดวก ในการเขียนโปรแกรม และทำการแอสเซมเบลอร์ได้ทันที โดยไม่ต้องมีโปรแกรมอื่นเข้ามาช่วย แสดงดังภาพที่ 3.4 ส่วนการใช้งานโปรแกรม RAD51 อธิบายได้เป็นบล็อกไดอะแกรม แสดงดังภาพที่ 3.5




**ภาพที่ 3.4** แสดงหน้าต่างของโปรแกรม RAD51



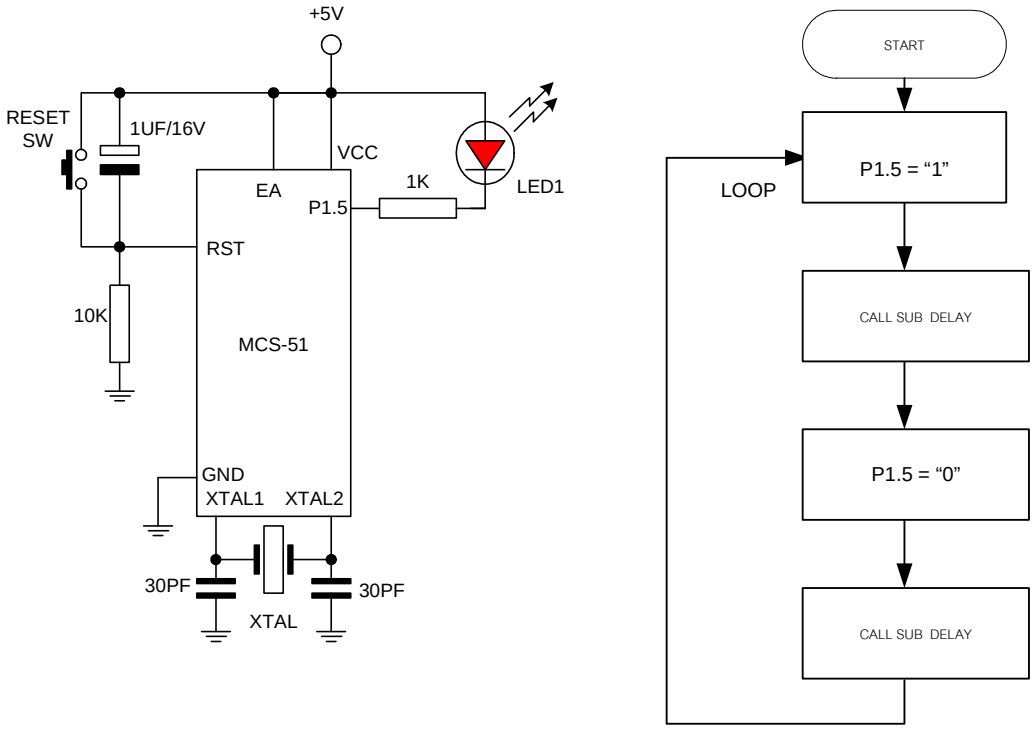

**ภาพที่ 3.5** แสดงบล็อกไดอะแกรมการใช้งานโปรแกรม RAD51

## 5. การเขียนโปรแกรมภาษาแอสเซมบลี

โปรแกรมใช้งานประเภทเท็กซ์เอดิเตอร์ใช้สำหรับพิมพ์ข้อมูลที่เป็นโปรแกรมต้นฉบับ สามารถเลือกใช้โปรแกรม Word Processor เช่น Notepad; Edit Plus 2, Edit For Dos หรือ Ultraedit-32, RAD51 ส่วนโปรแกรมแอสเซมเบลอร์ ของไอซีตระกูล MCS-51 เช่น SXA51, ASM51, CROSS32 และ RAD51

### 5.1 แนวคิดในการเขียนโปรแกรม

เขียนผังงานของโปรแกรมส่งข้อมูลแบบบิต กำหนดให้ LED1 ติดสว่าง และดับสลับกันไป จากวงจรแสดงดังภาพที่ 3.6 วิธีการต่อแอลอีดีให้ขาแคโถดต่อกับพอร์ต P1.5 อาโนดต่อกับแหล่งจ่ายไฟ กำหนดค่าไดเร็คทีฟเป็น LED_TEST ถ้าให้สถานะลอจิก“0” ที่ขา P1.5 ทำให้แอลอีดีติดสว่าง และถ้าให้ลอจิก “1” ทำให้แอลอีดีดับ




**ภาพที่ 3.6** แสดงผังงาน และการต่อวงจร

### 5.2 การเขียนโปรแกรมแอสเซมบลี

สร้างโฟลเดอร์ชื่อ LAB51 หลังจากนั้นนำโปรแกรม SXA51.EXE ไปเก็บไว้ในโฟลเดอร์ และให้เขียนโปรแกรมต้นฉบับ โดยใช้โปรแกรมเท็กซ์เอดิเตอร์

เมื่อเข้าสู่โปรแกรมให้พิมพ์คำสั่งข้างล่าง หลังจากนั้นให้บันทึกเป็นชื่อไฟล์ TB1.ASM (พิมพ์จุด, ตัวอักษร A, ตัวอักษร S และตัวอักษร M หมายถึง Assembly)

```
;***********************************************
;***   Program TB1.ASM Output P1.5 Assembler: SXA51
;***********************************************
        LED_TEST    BIT     P1.5    ; กำหนดชื่อให้กับบิตP1.5 มีค่าเท่ากับ LED_TEST
            ORG     0000H           ; เริ่มต้นโปรแกรมที่แอดเดรส 0000H
LOOP:       SETB    P1.5            ; กำหนดพอร์ต 1 บิตที่ 5 (P1.5) เป็นสถานะ "1"
            ACALL   DELAY           ; เรียกโปรแกรมย่อยชื่อ DELAY
            CLR     P1.5            ; กำหนดพอร์ต 1 บิตที่ 5 (P1.5) เป็นสถานะ "0"
            ACALL   DELAY           ; เรียกโปรแกรมย่อยชื่อ DELAY
            SJMP    LOOP            ; กระโดดไปเริ่มต้นที่ เลเบลชื่อ LOOP
```

```
;***********************************************
;*** Sub Program Delay Time โปรแกรมย่อยหน่วงเวลา  ***
;***********************************************
DELAY:    MOV    R0,#0FFH        ;R0 = FF = 255
DELAY1:   MOV    R1,#0FFH        ;R1 = FF = 255
          DJNZ   R1,$
          DJNZ   R0, DELAY1
          RET                    ; ออกจากโปรแกรมย่อยหลังจากปฏิบัติตามคำสั่งเสร็จสิ้น
          END                    ; จบโปรแกรม
```

**5.3 การแปลงไฟล์โดยใช้โปรแกรมแอสเซมเบลอร์**

การใช้โปรแกรมแอสเซมเบลอร์ ให้เปิดดูชื่อไฟล์ในโฟลเดอร์ที่สร้างไว้โดยพิมพ์ C:\LAB51>dir/w จะต้องมีชื่อไฟล์ที่ บันทึกชื่อ TB1.ASM (หากใช้โปรแกรมเท็กซ์เอดิเตอร์ที่มีโปรแกรมแอสเซมเบลอร์ในตัวให้ ปฏิบัติตามการใช้งานของโปรแกรม)

```
C:\LAB51>dir/w
Volume in drive C is
Volume Serial Number is 046B-19FD
Directory of C:\LAB51
[.]             [..]            TB1.ASM             SXA51.EXE

                2 file (s)      33,403 bytes
                2 dir (s)       660,189,184 bytes free
```

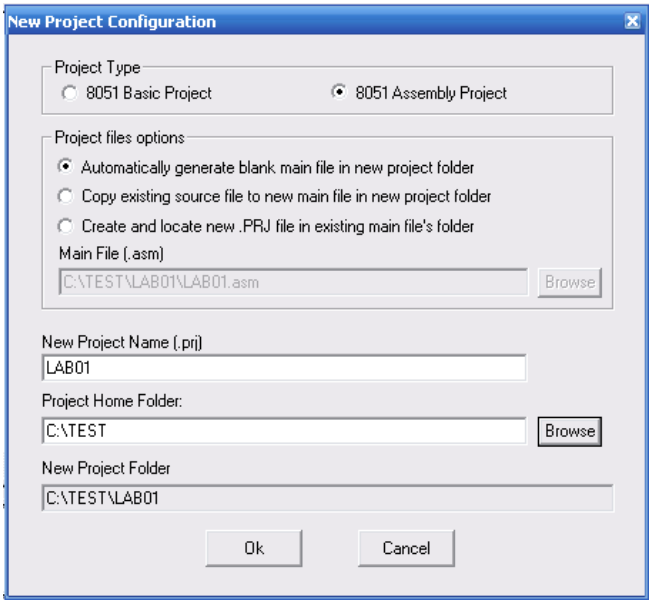
หลังจากนั้นให้ใช้โปรแกรมแอสเซมเบลอร์ SXA51 แปลงไฟล์โดยมีรูปแบบการพิมพ์ดังนี้

```
C:\>SXA51 –L TB1.ASM กด Enter
C:\LAB51>sxa51 –L tb1.asm
8051 Cross-Assembler (1.3) Copyright (c) 1987, 1989
Binary Technology, Inc. Meriden, MH

No errors detected
Object file size:  19   bytes
Program entry address:  0000 (Hex)
```

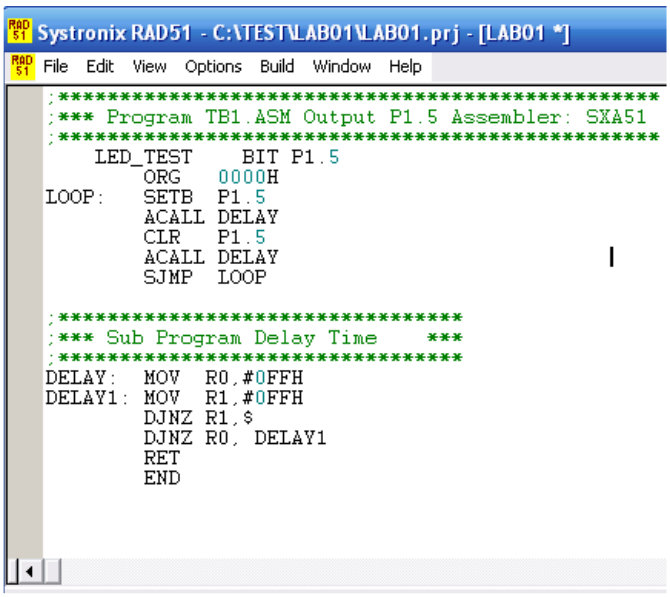
### 5.4 การเขียนโปรแกรมโดยใช้โปรแกรม RAD51

5.4.1 ทำการเปิดไฟล์โดยเลือกที่เมนู New เลือกที่ Project หลังจากนั้นให้กำหนดค่าเพื่อสร้างโปรเจค เลือกที่ปุ่ม Ok แสดงดังภาพที่ 3.7



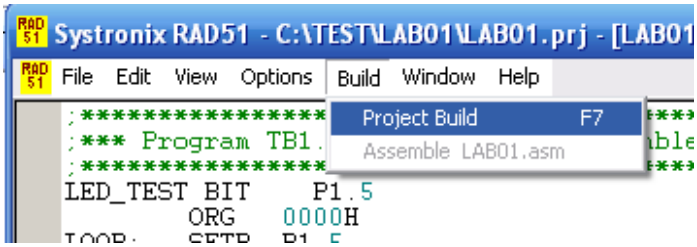

**ภาพที่ 3.7** กำหนดชื่อสร้างโปรเจค

5.4.2 ให้พิมพ์โปรแกรมภาษาแอสเซมบลี แสดงดังภาพที่ 3.8



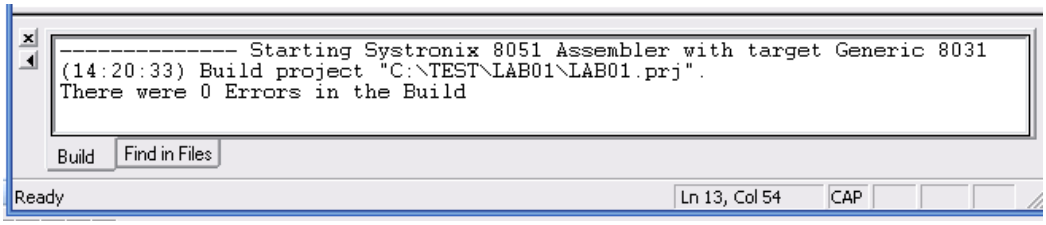

**ภาพที่ 3.8** พิมพ์โปรแกรมภาษาแอสเซมบลี

5.4.3 ให้บันทึกไฟล์โดยเลือกที่เมนู File → Save หลังจากนั้นให้เลือกที่เมนู Build เลือก Project Build แสดงดังภาพที่ 3.9




**ภาพที่ 3.9** เมนู Project Build

5.4.4 โปรแกรมแอสเซมเบลอร์ จะแปลงไฟล์จากภาษาแอสเซมบลีให้เป็นภาษาเครื่องที่หน้าต่าง Build จะมีรายละเอียด และแจ้งข้อผิดพลาดจากโปรแกรมแอสเซมเบลอร์ แสดงดังภาพที่ 3.10




**ภาพที่ 3.10** รายละเอียด และแจ้งข้อผิดพลาดจากโปรแกรมแอสเซมเบลอร์

## 5.5. รายละเอียดของไฟล์หลังการแอสเซมเบลอร์

ไฟล์หลังจากทำการแอสเซมเบลอร์จะประกอบด้วย TB1.ASM, SXA51.EXE, TB1.LST มีรายละเอียดดังต่อไปนี้

5.5.1 รายละเอียดของ Listing File

เป็นไฟล์ที่มีรายละเอียดของโปรแกรม ประกอบด้วยตำแหน่งแอดเดรสของหน่วยความจำรหัสคำสั่งภาษาเครื่อง และยังแบ่งออกเป็นคอลัมน์ (Column) ในแต่ละคอลัมน์เรียกเป็นหนึ่งฟิลด์ (Field) ในหนึ่งบรรทัดประกอบด้วย ฟิลด์สูงสุด 5 ฟิลด์ คือ Line #, Label, Op-code, Operands และ Comment มีรายละเอียดดังนี้

```
TB1.ASM
8051 Cross-Assembler (1.3)  (C) 1987, 1989 Binary Technology
     Page    1 tb1.asm
             1          ;***********************************************
```

```
                2       ;***    Program TB1.ASM Output P1.5 Assembler: SXA51
                3       ;************************************************
0095=           4                       LED_TEST    BIT     P1.5
                5               ;****************************************
                6               ;****  MAIN PROGRAM    โปรแกรมหลัก   *****
                7               ;****************************************
0000            8               ORG     0000H
0000 D295       9 LOOP:         SETB    LED_TEST
0002 110A       10              ACALL   DELAY
0004 C295       11              CLR     LED_TEST
0006 110A       12              ACALL   DELAY
0008 80F6       13              SJMP    LOOP
                14      ;*****************************************************
                15      ;****   Program Delay Time โปรแกรมย่อย  *******
                16      ;*****************************************************
000A 78FF       17 DELAY:       MOV     R0,#0FFH
000C 79FF       19 DELAY1:      MOV     R1,#0FFH
000E D9FE       20                      DJNZ    R1,$
0010 D8FA       21                      DJNZ    R0, DELAY1
0012 22         22                      RET
0000=           23                      END
    delay = 000A        delay1 = 000C       led_test = 0095         main = 0000
```

**line #** หมายถึง บรรทัดที่เท่าใดของโปรแกรมต้นแบบ

5.5.2 รายละเอียดของ Intel Hex File

เป็นรูปแบบของไฟล์มาตรฐานสามารถนำไปบันทึกลงตัวไอซีได้ ไฟล์ TB1.HEX มีรายละเอียดดังนี้

```
: 10000000D295110AC295110A80F678FF79FFD9FEC0
: 03001000D8FA22F9
: 00000001F
```

จากบรรทัดแรก

: 10000000D295110AC295110A80F678FF79FFD9FEC0 ต้นบรรทัดจะขึ้นต้นด้วย ":"

ถัดมา คือ **10** หมายถึง ข้อมูลในเรกคอร์ดนี้มีความยาว 10H (เลขฐานสิบหก) หรือ 16 ไบต์

ถัดมา คือ **0000** หมายถึง แอดเดรสเริ่มต้นในการวางข้อมูลของเรกคอร์ดนี้คือ 0000H

ถัดมา คือ **00** หมายถึง ข้อมูลที่ตามหลังมาเป็นรหัสข้อมูลที่ต้องเริ่มวางลงในหน่วยความจำ

ถัดมา คือ **D2** ถึง **FE** รหัสข้อมูลรวมทั้งสิ้น 16 ไบต์

ท้ายสุดคือ **C0** หมายถึง ค่า Check Sum ของเรกคอร์ดนี้ได้จากการบวกข้อมูลทุกตัวในเรกคอร์ดนี้เข้าด้วยกัน ยกเว้น ":" และ Check Sum ผลที่ได้นำมาทำ คอมพลีเมนท์ของสอง แล้วตัดเอาเฉพาะ 8 บิต ทางด้านต่ำ

**SUM=10+00+00+00+D2+95+11+0A+C2+95+11+0A+80+F6+78+FF+79+FF+D9+FE = 940** $_{H}$

นำค่า $940_H$ แปลงค่าเป็นเลขฐานสอง = **1001 0100 0000**$_{B}$

ให้กลับค่า (1's Complement) ของเลขฐานสอง **1001 0100 0000**$_{B}$ = **0110 1011 1111** $_{B}$ = **6BF**$_{H}$

ทำเป็นค่า 2's Complement โดยการบวกค่าอีก 1 = **0110 10111111**$_{B}$ **+1** = **0110 1100 0000**$_{B}$ = **6C0**$_{H}$

นำเอาเฉพาะค่าสองหลักสุดท้าย คือ **C0**$_{H}$ มาเป็นไบต์ Check Sum

บรรทัดที่สองเริ่มบรรทัดด้วย "**:**" ไบต์ต่อมาจะเป็นความยาวของข้อมูลทั้งหมดคือ 03 ไบต์ ซึ่งเป็นแอดเดรสด้านหน้ากับข้อมูลด้านหลัง จากนั้นก็เป็นข้อมูลอีก 03 ไบต์ **03+00+10+00+D8+FA+22 = 207**$_{H}$ และ **F9**$_{H}$ จะเป็นไบต์ที่แสดงการ Check Sum

บรรทัดที่สามเป็นการแสดงการจบข้อมูล คือ

**: 00000001FF:**

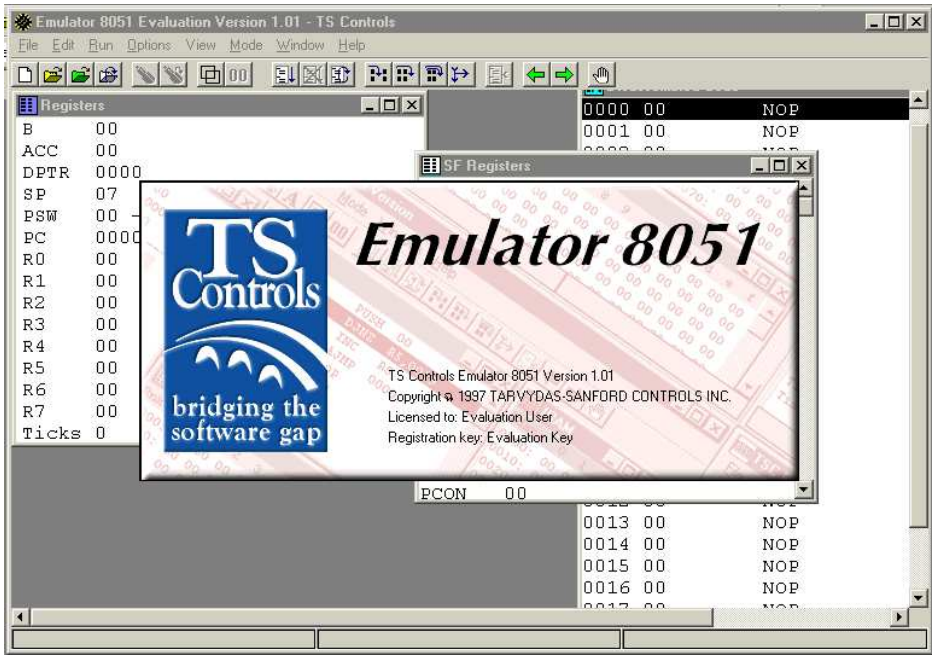
## 6. เครื่องมือในการพัฒนางาน

ส่วนสำคัญของการพัฒนาทางด้านไมโครคอนโทรลเลอร์ คือความสะดวก และรวดเร็วในการเขียนโปรแกรม เพื่อทดสอบการทำงานของชิ้นงานที่สร้างขึ้น ให้เป็นไปตามเงื่อนไขที่กำหนด ในการพัฒนาดังกล่าวต้องอาศัยเครื่องมือในการทดลอง และการแก้ไข ปรับปรุง ดังนั้นเครื่องมือที่อำนวยความสะดวกในการทดลอง จึงมีให้เลือกใช้งานหลายวิธีเช่น

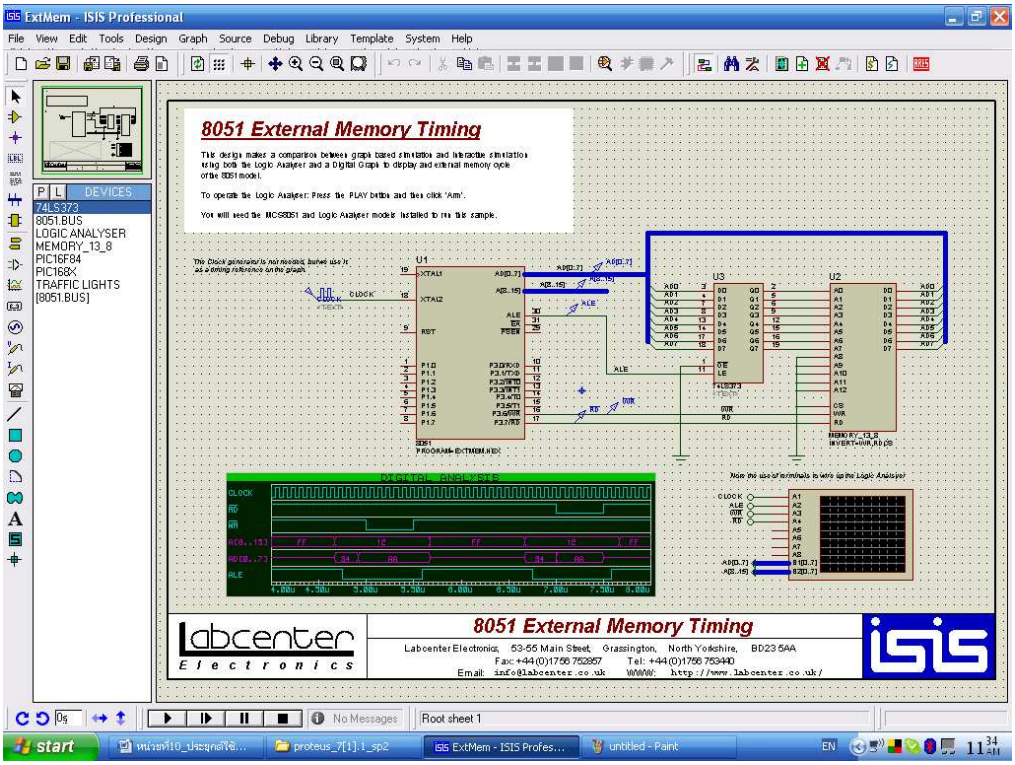
### 6.1 โปรแกรมจำลองการทำงาน (Simulator)

เป็นโปรแกรมที่สร้างขึ้นบนคอมพิวเตอร์ เพื่อจำลองการทำงานส่วนต่างๆภายในของตัวไอซี โดยนำโปรแกรมที่เขียนเรียบร้อยแล้ว แปลงไฟล์ข้อมูลให้เป็นเลขฐานสิบหก และนำขึ้นไปเป็นข้อมูลเพื่อทดสอบการทำงานบนคอมพิวเตอร์ ซึ่งสามารถทำเป็นขั้นตอนในแต่ละคำสั่งได้ ทำให้สะดวกที่จะดูข้อมูลหรือตรวจสอบการเปลี่ยนแปลงค่าในส่วนของรีจิสเตอร์ หรือหน่วยความจำได้ แต่การใช้งานส่วนที่มีเงื่อนไขมาก เช่นอินพุต และเอาต์พุตภายนอกทำได้ค่อนข้างยุ่งยาก แต่เหมาะสำหรับผู้ที่เรียนรู้เบื้องต้น เพื่อสังเกตการทำงานของโปรแกรมได้โดยไม่ต้องต่อบอร์ดทดลองใดเลย ตัวอย่างเช่นโปรแกรม TS Controls Emulator 8051 แสดงดังภาพที่ 3.11 ดาวน์โหลดได้ที่ http://www.tscontrols.com โปรแกรม Proteusใช้

จำลองการทำงานของไมโครคอนโทรลเลอร์ ซึ่งเป็นการทำงานของฮาร์ดแวร์ สภาพแวดล้อมของโปรแกรมการใช้งาน การเขียนวงจร การวางอุปกรณ์ ดังแสดงในภาพที่ 3.12



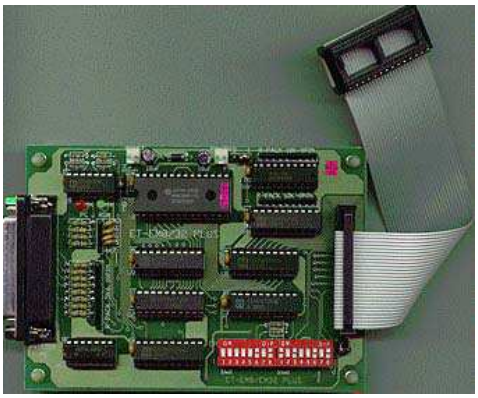

**ภาพที่ 3.11** โปรแกรม TS Controls Emulator 8051จำลองการทำงานของไอซี MCS-51

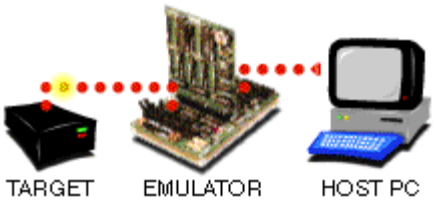


**ภาพที่ 3.12** โปรแกรม Proteus จำลองการทำงานของไอซี MCS-51

### 6.2 อีพรอมอีมูเลเตอร์ (EPROM Emulator)

โครงสร้างวงจรเป็นแบบหน่วยความจำแรมใช้การโหลดข้อมูลผ่านทางพอร์ตขนาน (LPT) หรือพอร์ตอนุกรม (COM x) ของเครื่องไมโครคอมพิวเตอร์ การใช้งานมีขาที่เชื่อมต่อกับบอร์ดทดลอง เพื่อนำข้อมูลของแรมบนอีมูเลเตอร์ ไปแทนการทำงานของหน่วยความจำโปรแกรมในตัวไอซี MCS-51 ข้อดีของการใช้เครื่องอีมูเลเตอร์ คือ การแก้ไขข้อมูลสะดวก รวดเร็ว เนื่องจากใช้หน่วยความจำแรม สามารถโปรแกรม และลบข้อมูลได้ง่าย ส่วนข้อเสียคือมีราคาแพง และมีสัญญาณรบกวน แสดงดังภาพที่ 3.13






**ภาพที่ 3.13** อีพรอมอีมูเลเตอร์

(แหล่งอ้างอิง http://www.etteam.com)

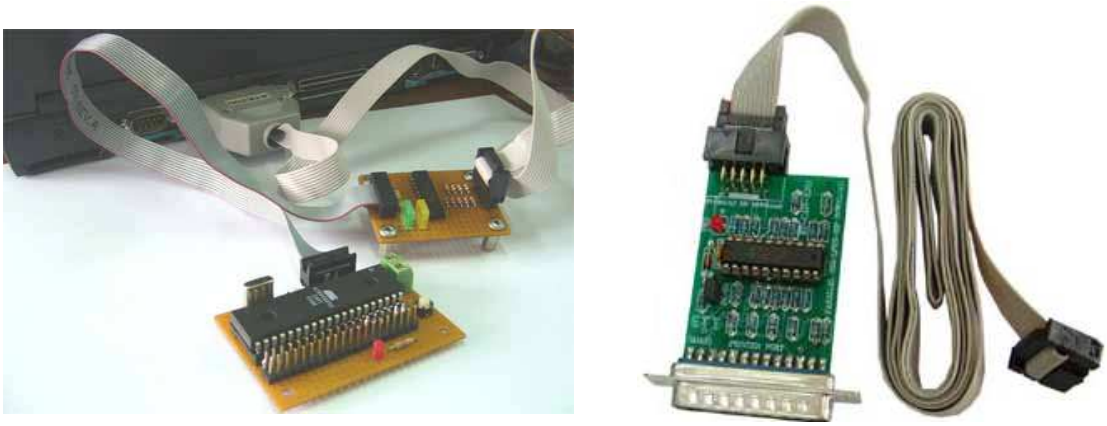
### 6.3 เครื่องโปรแกรมไอซี (IC Programmer)

เป็นการนำข้อมูลภาษาเครื่องบันทึกลงในตัวไอซีโดยเครื่องโปรแกรม แสดงดังภาพที่ 3.14 หลังจากนั้นนำไอซีที่บันทึกข้อมูลไปทำการทดสอบไอซี MCS-51 ที่มีหน่วยความจำโปรแกรมแบบแฟลช ในการโปรแกรมแต่ละครั้งเครื่องโปรแกรมจะลบ และบันทึกข้อมูลในครั้งเดียวกัน ซึ่งใช้เวลาไม่มาก ส่วนไอซี MCS-51 ที่มีหน่วยความจำแบบ EPROM อาจใช้เวลานานในการลบข้อมูลด้วยแสงอัลตร้าไวโอเลต เพื่อบันทึกข้อมูลใหม่

**ภาพที่ 3.14** เครื่องโปรแกรมไอซี MCS-51

### 6.4 เครื่องโปรแกรมแบบ ISP (In-System Programming)

สามารถโปรแกรมข้อมูลลงในหน่วยความจำโปรแกรมภายในไอซี MCS-51 โดยไม่ต้องถอดไอซีออกจากบอร์ด มีวงจรที่ไม่ซับซ้อน การแก้ไขข้อมูลทำได้สะดวก และรวดเร็ว แสดงดังภาพที่ 3.15



**ภาพที่ 3.15** บอร์ดโปรแกรมแบบ ISP

(แหล่งอ้างอิง http://www.etteam.com)

# สรุป

ชุดรหัสคำสั่งของ MCS-51 มีทั้งหมด 256 คำสั่ง อยู่ในรูปแบบของเลขฐานสอง ประกอบด้วยคำสั่งขนาด 1 ไบต์จำนวน 139 คำสั่ง คำสั่งขนาด 2 ไบต์จำนวน 92 คำสั่งและ คำสั่งขนาด 3 ไบต์จำนวน 29 คำสั่ง

แอดเดรสซิ่งโหมดคือ คำสั่งในการเข้าถึงข้อมูลในหน่วยความจำตำแหน่งต่างๆ จะสามารถแบ่งการอ้างแอดเดรสของหน่วยความจำออกเป็นแบบต่างๆ ได้ดังนี้

1) การอ้างแอดเดรสของหน่วยความจำแบบทันทีทันใด
2) การอ้างแอดเดรสของหน่วยความจำแบบโดยตรง
3) การอ้างแอดเดรสของหน่วยความจำแบบรีจิสเตอร์
4) การอ้างแอดเดรสของหน่วยความจำแบบโดยอ้อม
5) การอ้างแอดเดรสของหน่วยความจำแบบบิต
6) การอ้างแอดเดรสของหน่วยความจำแบบฐานแอดเดรส

โปรแกรมถูกเก็บไว้ในหน่วยความจำ เป็นรูปแบบของเลขไบนารี่เรียกว่า ภาษาเครื่อง เป็นภาษาที่คอมพิวเตอร์เข้าใจได้ และเปลี่ยนรูปแบบของภาษาเครื่อง ให้อยู่ในรูปแบบของเลขฐานสิบหก เช่น คำสั่งขนาด 8 บิต 11101011B เขียนได้เป็น EBH แต่เป็นการเข้าใจได้ยาก ดังนั้นจึงมีการใช้สัญลักษณ์ ที่เรียกว่า นีโมนิค เพื่อแทนความหมายของคำสั่ง เช่น MOV A, #67H โปรแกรมที่เขียนด้วยรหัส นีโมนิคเรียกว่าภาษาแอสเซมบลี และก่อนให้ไมโครคอนโทรลเลอร์ทำงานตามโปรแกรมภาษาแอสเซมบลีได้ ต้องเปลี่ยนให้เป็นภาษาเครื่องก่อน เรียกว่าการแอสเซมเบลอร์ หลังการแอสเซมเบลอร์จะได้ LISTING FILE และ HEX FILE

เครื่องมือที่ใช้ในการพัฒนาไอซีไมโครคอนโทรลเลอร์

1) โปรแกรมซิมูเลเตอร์ เป็นโปรแกรมสร้างบนคอมพิวเตอร์ เพื่อเป็นการจำลองการทำงานของ ส่วนต่างๆภายในตัวไอซี

2) อีพรอม อีมูเลเตอร์เป็นการนำเอาข้อมูลมาทำการบันทึกลงในไอซีหน่วยความจำของเครื่องอีมูเลเตอร์แทน ซึ่งจะมีโครงสร้างวงจรเป็นแบบหน่วยความจำประเภทแรม

3) เครื่องโปรแกรมตัวไอซี เป็นการนำเอาโปรแกรมที่เขียนเสร็จแล้วมาทำการแปลงไฟล์ให้เป็นข้อมูลเลขฐานสิบหก หลังจากนั้นก็จะนำไปอัดลงในตัวไอซีโดยใช้เครื่องโปรแกรม

4) เครื่องโปรแกรมแบบ ISP สามารถทำการโปรแกรมข้อมูลลงในหน่วยความจำโปรแกรมได้โดยตรง โดยไม่ต้องถอดตัวไอซีไมโครคอนโทรลเลอร์ออกจากบอร์ด