บทที่ 3
การเขียนโปรแกรมภาษาแอสแซมบลี

การที่จะให้ MCS – 51 ทำงานได้จะต้องให้คำสั่งแก่มัน การนำรหัสคำสั่งหลาย ๆ รหัสคำสั่งมาเรียงกันให้ MCS – 51 ทำงานเรียกว่าโปรแกรมภาษาแอสแซมบลี ซึ่งภาษานี้จะเป็นภาษาที่อยู่ระหว่างภาษาเครื่องกับภาษาระดับสูง โดยภาษาระดับสูงเช่น Pascal หรือ C จะใช้ชุดคำสั่งที่ใกล้เคียงกับภาษาที่มนุษย์เข้าใจได้ง่าย ส่วนภาษาเครื่องจะเป็นภาษาที่ใช้เลขฐานสองซึ่งคอมพิวเตอร์จะเข้าใจสำหรับ CPU แต่ละเบอร์ โปรแกรมภาษาเครื่องจะเป็นชุดจำนวนไบต์ของเลขฐานสองซึ่งขึ้นกับชุดคำสั่งที่จะให้คอมพิวเตอร์ทำงาน

ภาษาแอสแซมบลีจะแทนรหัสภาษาเครื่องเลขฐานสองด้วยรหัสที่เราเข้าใจง่ายคือ นีโมนิก "Mnemonics" ถ้าหากให้ CPU บวกเลขจะต้องป้อนรหัสคำสั่งเป็นภาษาเครื่องว่า "10110011" ซึ่งถ้าเขียนเป็นภาษาแอสแซมบลีอาจแทนด้วย "ADD" การเขียนโปรแกรมภาษาแอสแซมบลีจะเขียนเป็นรหัสนีโมนิกทั้งหมดเพื่อให้เข้าใจได้ง่าย แต่ถ้าจะให้ CPU เข้าใจจะต้องแปลงรหัสนีโมนิกเหล่านี้ให้เป็นเลขฐานสองขั้นตอนนี้เรียกว่า Assembler ซึ่งอาจแปลงได้โดยการเปิดตารางรหัสคำสั่งของ CPU ที่กำลังศึกษาอยู่ หรือแปลงโดยใช้โปรแกรม Assembler ในปัจจุบันการเขียนโปรแกรมภาษาแอสแซมบลีจะเขียนบนเครื่องคอมพิวเตอร์แล้วใช้โปรแกรม Assembler ทั้งสิ้น นอกจากนี้การเขียนโปรแกรมบนเครื่องคอมพิวเตอร์ยังมีคำสั่งพิเศษต่าง ๆ ที่ช่วยในการเขียนโปรแกรมอีกด้วย

3.1 รูปแบบของภาษาแอสแซมบลี[1]

โปรแกรมภาษาแอสแซมบลีประกอบด้วยส่วนต่าง ๆ ดังต่อไปนี้

1. Machine Instructions
2. Assembler Directives
3. Assembler Controls
4. Comments

Machine Instructions คือชุดคำสั่งภาษาเครื่องโดยแทนด้วยรหัสนีโมนิก เช่น MOV , ANL เป็นต้น Assembler Directives หรือคำสั่งเทียมเป็นคำสั่งที่กำหนดขึ้นในภาษาแอสแซมบลีโดยจะไม่ถูกเปลี่ยนแปลงเป็นรหัสภาษาเครื่อง มีไว้สำหรับกำหนดโครงสร้างโปรแกรม, สัญลักษณ์, ข้อมูลต่าง ๆ เช่น ORG,EQU เป็นต้น Assembler Controls เป็นชุดคำสั่งในการควบคุมของตัวแอสแซมเบอร์ Comments เป็นการเขียนคำอธิบายเข้าไปในโปรแกรม

ในการเขียนโปรแกรมในภาษาแอสแซมบลีในแต่ละบรรทัดมีส่วนประกอบที่สำคัญ 4 ส่วนคือ Label, Mnemonics, Operand, Comment โดยรูปแบบทั่วไปจะเป็นดังนี้

→ [Label :] Mnemonic [Operand] [,Operand] [; Comment]

ในโปรแกรมมักจะเริ่ม Label ที่คอลัมน์ที่ 1

Label Field

ในภาษาแอสแซมบลี Lable จะเป็นตัวแทนตำแหน่งของชุดคำสั่งหรือข้อมูลที่ตามหลัง Label อยู่ในคำสั่งกระโดดหรือคำสั่งต่าง ๆ ที่ต้องการอ้างตำแหน่ง สามารถแทนค่าตำแหน่งด้วย Label ได้เลย เช่น SJMP เป็นการกระโดดไปที่ตำแหน่งที่แทนด้วย Label SKIP

โดยทั่วไปแล้ว Label Field แทนได้ สองความหมายคือ เป็น "Label" แทนค่าตำแหน่งและเป็น "Symbol" ซึ่งจะแทนค่าสัญลักษณ์ ถ้าหากจะใช้เป็น Label จะตามด้วยเครื่องหมาย " : " (Colon) แต่ถ้าจะเป็น Symbol จะตามด้วยคำสั่งเทียม เช่น EQU , SEGMENT , BIT , DATA เป็นต้น พิจารณาโปรแกรมต่อไปนี้จะใช้ PAR แทน Symbol และ START แทน Label

```
PAR    EQU 500          ; "PAR" เป็นสัญลักษณ์แทนด้วยค่า 500
START : MOV , #oFFH     ; "START" เป็นlabelจะมีความหมายแทน
                                ตำแหน่งที่เก็บคำสั่ง MOV
```

การตั้งชื่อให้ Label Field โดยมากจะเริ่มด้วยตัวอักษร สำหรับรายละเอียดต่าง ๆ สามารถศึกษาได้จากคู่มือของตัว Assembler โดยตรง

Mnemonic Field

ส่วน Mnemonic Field จะเป็นชุดคำสั่งหรือคำสั่งเทียมก็ได้ ซึ่งจะเขียนตามหลัง Lable ตัวอย่างของชุดคำสั่งได้แก่ ADD, MOV, DIV, หรือ INC ตัวอย่างของคำสั่งเทียมได้แก่ ORG ,EQU หรือ DB

Operand Field

ในการเขียน Mnemonic เป็นตัวบอกว่าจะให้ Mnemonic ทำอะไร เช่น MOV จะเป็นตัวบอกว่าให้ย้ายข้อมูล แต่ยังไม่มีตำแหน่งที่จะย้ายว่าจะย้ายจากไหนไปไหน ส่วนที่จะบอกว่าจะให้ทำกับอะไรเรียกว่า Operand ซึ่งจะเขียนตามหลังรหัส Mnemonic ใน Operand Field นี้จะประกอบด้วย Address หรือ Data ที่จะใช้กับชุดคำสั่ง ซึ่งอาจจะเป็น Label ที่แทนตำแหน่งของข้อมูลหรือ Symbol ที่แทนค่าข้อมูลด้วยก็ได้คำสั่งส่วนใหญ่จะมี Operand Field ตามหลังแต่บางคำสั่งจะไม่มี เช่น คำสั่ง RET

Comment Field

ส่วนนี้เป็นส่วนที่เขียนขึ้นในภาษาแอสแซมบลี เพื่อเป็นคำอธิบายส่วนต่าง ๆ ในโปรแกรม โดยการเขียนจะใช้เครื่องหมาย " ; " (Semicolon) นำหน้าส่วน Comment นี้ ตัวแอสแซมเบอร์จะไม่แปลเป็นภาษาเครื่อง

Special Assembler Symbols

ในภาษาแอสแซมบลีของ MCS – 51 การอ้างตำแหน่งของรีจีสเตอร์ต่าง ๆ สามารถเขียนเป็นตัวอักษรได้โดยตรง เช่น A,Ro ถึง R7, DPTR, PC, C และ AB ส่วนสัญลักษณ์ $ (Dollar Sign) จะแทนตำแหน่งที่รีจีสเตอร์ PC ชี้อยู่ โดยมากมักจะใช้คำสั่งกระโดด ตัวอย่าง

```
SETB C          ; เซตบิต C
INC DPTR        ; เพิ่มค่า DPTR
JNB T1 , $      ; กระโดดไปตำแหน่งเดิมโดยการตรวจบิต T1
```

คำสั่งในบรรทัดสุดท้ายจะใช้เครื่องหมาย $ ซึ่งจะมีค่าเท่ากับคำสั่งต่อไปนี้

```
HERE : JNB T1 , HERE
```

Indirect Address

การกำหนดตำแหน่งของข้อมูลทางอ้อมจะใช้เครื่องหมาย "@" (AT) หน้ารีจีสเตอร์ โดยรีจีสเตอร์ที่ใช้ได้แก่ Ro, R1, DPTR, PC ตัวอย่างเช่น

```
ADD   A , @Ro
MOV   A , @ A+PC
```

Immediate Data

การกำหนดค่าโดยตรงในส่วนของ Operand Field ในภาษาแอสแซมบลีจะใช้เครื่องหมาย "#" (Pound Sign) พิจารณาตัวอย่างต่อไปนี้

```
CONSTANT        EQU   100
                MOV   A, #0FEH
                ORL   40H, #CONSTANT
```

การกำหนดค่าโดยตรงถ้าหากกำหนดค่าขนาด 16 บิตให้กับรีจีสเตอร์ขนาด 8 บิต ผลที่ได้จะนำค่า Low – byte ไปเก็บในรีจีสเตอร์ โดย High – byte จะกำหนดได้เพียง 2 ค่าเท่านั้นคือ 00H และ FFH ถ้าเป็นค่าอื่นจะเกิดข้อผิดพลาดขึ้น เช่น

```
MOV   A , #oFFooH
MOV   A , #ooFFH
```

แต่ถ้าเขียนในลักษณะสองบรรทัดต่อไปนี้ จะเกิดข้อผิดพลาดขึ้น เนื่องจากใน High – byte กำหนดค่าผิด

```
MOV   A, #oFEooH
MOV   A, #o1FFH
```

ถ้าหากใช้สัญลักษณ์เป็นเลขฐานสิบ ซึ่งมักมีค่าระหว่าง -256 ถึง +256 ถ้าหากเขียนในลักษณะสองบรรทัดต่อไปนี้ค่าที่ได้จะมีผลเท่ากัน

```
MOV   A, #-256
MOV   A, #oFFoH
```

Data Address

คำสั่งในการเข้าถึงข้อมูลในหน่วยความจำ จะใช้วิธีเข้าถึงโดยตรงได้กับหน่วยความจำภายในชิพตำแหน่ง ooH ถึง 7FH สำหรับรีจีสเตอร์ฟังก์ชั่นพิเศษจะอยู่ในตำแหน่ง 80H ถึง oFFH ในการเขียนโปรแกรมสามารถอ้างตำแหน่งโดยใช้ชื่อรีจีสเตอร์โดยตรงได้

```
MOV   A, 45H
MOV   A, SBUF        ;มีผลเท่ากับการอ่านจากตำแหน่ง 99H
```

Bit Address

การเข้าถึงข้อมูลระดับบิตใน MCS – 51 จะอยู่ในไบต์ที่ 20H ถึง 2FH โดยตำแหน่งของบิตคือตำแหน่งของบิตที่ ooH ถึง 7FH และในรีจีสเตอร์ในฟังก์ชั่นพิเศษบางตัวก็สามารถจะเข้าถึงข้อมูลระดับบิตได้ดังที่กล่าวไว้แล้วในตอนต้น ในการเขียนโปรแกรมภาษาแอสแซมบลี ถ้าเป็นคำสั่งที่

กระทำกับบิตสามารถใส่ตำแหน่งของบิตได้ทันที และถ้าเป็นการอ้างตำแหน่งของบิตหรือใช้ชื่อบิตของรีจีสเตอร์นั้นเลย ตัวอย่างเช่น

```
SETB    oE7H        ; เซตบิตตำแหน่งที่ oE7H
SETB    ACC.7       ; เซตบิตที่ 7 ในรีจีสเตอร์ A
JNB     TI,$        ; ใช้เครื่องหมาย TI แทนตำแหน่งบิต
JNB     99H , $
```

Code Address

เราสามารถใช้รหัสตำแหน่ง (Code Address) ซึ่งสร้างจาก Label แทนในคำสั่ง JUMP และคำสั่ง CALLS ต่าง ๆ ได้

Generic Jumps and Calls

การเขียนโปรแกรมภาษาแอสแซมบลีแล้วใช้โปรแกรมแอสแซมเบอร์แปลงเป็นรหัสคำสั่งนั้นมีข้อดีอีกอย่างหนึ่ง คือการใช้คำสั่งกระโดดและคำสั่งเรียกโปรแกรมย่อยสามารถใช้ได้โดยง่ายโดยรูปแบบคำสั่งที่ใช้คือ JMP และ CALL โดย "JMP" สามารถแทนคำสั่ง SJMP, AJMP หรือ LJMP ได้ สำหรับคำสั่ง "CALL" สามารถใช้แทน ACALL หรือ LCALL ได้ โดยตัวแอสแซมเบอร์จะแปลงคำสั่ง JMP และ CALL ให้เป็นคำสั่งที่เหมาะสมเอง โดยพิจารณาจากระยะทางที่ PC จะกระโดดไป

พิจารณาโปรแกรมดังรูปที่ 3.1 จากโปรแกรมจะเริ่มที่ตำแหน่ง 1234H คำสั่งกระโดดคำสั่งแรกจะเริ่มที่บรรทัดที่ 3 ซึ่งระยะทางที่จะกระโดดไปจะเห็นว่าอยู่ระหว่าง -128 ถึง +127 ดังนั้นตัวแอสแซมเบอร์จะแปลง JMP เป็น SJMP ซึ่งดูได้จากรหัส OBJ (รหัส 80 เป็นรหัสของ SJMP) ในบรรทัดที่ 4 จะกำหนด ORG เพิ่มอีก 200 ตำแหน่ง ห่างจากตำแหน่งที่อ้างด้วย Label START ในบรรทัดที่ 5 จะมีคำสั่ง JMP ซึ่งตำแหน่งที่จะกระโดดไปมีค่าเกิน -128 ถึง +127 แต่ไม่เกิน 2k ตำแหน่ง ตัวแอสแซมเบอร์จะแปลงให้เป็น AJMP

LOC	OBJ	LINE	SOURCE		
1234		1		ORG	1234
1234	04	2	START:	INC	A
1235	80FD	3		JMP	START
12FC		4		ORG	START + 200
12FC	4134	5		JMP	START
12FE	021301	6		JMP	FINISH
1301	04	7	FINISH:	INC	A
		8		END	

รูปที่ 3.1 ตัวอย่างโปรแกรมภาษาแอสแซมบลี

3.2 Assemble – time Expression Evalution

ในที่นี้จะกล่าวการกำหนดค่าคงที่และตัวแปรต่าง ๆ

Number Bases

การกำหนดค่าตัวเลขหรือค่าคงที่ต่าง ๆ ให้กับตัวแปรในการเขียนภาษาแอสแซมบลีนั้นถ้าเป็นเลขไบนารีจะลงท้ายด้วยตัว "B" ถ้าเป็นเลขฐานสิบหกจะใส่ตัวเลข 0 นำหน้าและลงท้ายด้วยตัว "H" ถ้าเป็นเลขฐานสิบจะลงท้ายด้วยอักษร "D" ถ้าเป็นเลขฐานแปดจะลงท้ายด้วย O ตัวอย่างเช่น

```
MOV     A , #1111B
MOV     A , #oFH
MOV     A , #15D
```

Character Strings

การกำหนดตัวอักษรจะใช้ 1 หรือ 2 ตัว โดยการกำหนดจะใช้เครื่องหมาย (') ปิดตัวอักษร ตัวแปรภาษาแอสแซมเบอร์จะแปลงตัวอักษรเป็นรหัส ASCII ดังตัวอย่างต่อไปนี้

```
CJNE    A , # 'Q' , AGAIN
SUBB    A , # 'Q'
```

```
MOV     DPTR , # 'AB'
MOV     DPTR , #4124H
```

Arithmetic Operators

ตัวดำเนินการทางคณิตศาสตร์ ที่ใช้ในภาษาแอสแซมบลีมีดังนี้

ตัวดำเนินการ	ความหมาย
+	บวก
-	ลบ
*	คูณ
/	หาร
MOD	หารแบบ module (เอาเศษที่ได้จากการหาร)

ตารางที่ 3.1 ตัวดำเนินการทางคณิตศาสตร์

การเขียนโปรแกรมทางคณิตศาสตร์ของสองบรรทัดต่อไปนี้ให้ผลเหมือนกัน โดยบรรทัดแรกจะบวกค่า 10 ในฐานสิบหกมาเก็บไว้ในรีจีสเตอร์ A ซึ่งมีค่าเท่ากับการเก็บค่า 1AH ลงในรีจีสเตอร์ A โดยตรง

```
MOV     A , #10 + 10H
MOV     A , # 1AH
```

การเขียนโปรแกรมดังสองบรรทัดต่อไปนี้ให้ผลเช่นเดียวกันโดยบรรทัดแรกจะเอาเศษที่ได้จากการหารคือ 4 ใส่ในรีจีสเตอร์ A

```
MOV     A , #25  MOD 7
MOV     A , #4
```

Logical Operators

ตัวดำเนินการทางลอจิกมีดังนี้

ตัวดำเนินการ	ความหมาย
OR	การกระทำ OR
AND	การกระทำ AND
XOR	การกระทำ Exclusive
NOT	การกระทำ Complement

ตารางที่ 3.2 ตัวดำเนินการทางลอจิก

การเขียนโปรแกรม เราสามารถใส่ตัวดำเนินการเข้าไปในคำสั่งได้เลย ดังตัวอย่างต่อไปนี้

```
MOV          A , '#9' AND oFH
```

ตัวอย่างต่อไปนี้จะแสดงให้เห็นว่าตัวดำเนินการ NOT สามารถใส่เข้าไปในตัว Operand ได้ทันทีซึ่งคำสั่ง MOV ทั้งสามคำสั่งผลที่ได้จะเหมือนกัน

```
          THREE        EUQ   3
MINUS_THREE            EQU   -3
                       MOV   A , # (NOT  THREE) +1
                       MOV   A , #MINUS_THREE
                       MOV   A , #11111101B
```

Special Operators

ตัวดำเนินการพิเศษมีดังนี้

ตัวดำเนินการ	ความหมาย
SHR	เลื่อนไปทางขวา
SHL	เลื่อนไปทางซ้าย
HIGH	ค่าไบต์สูง
LOW	ค่าไบต์ต่ำ
( )	ให้ทำค่านี้ก่อน

ตารางที่ 3.3 ตัวดำเนินการพิเศษ

ตัวอย่างเช่น

```
MOV     A , #8 SHL 1
MOV     A , #10H
```

ตัวอย่างจะเป็นคำสั่งเลื่อนค่า 8 บิตไปหนึ่งครั้งแล้วเก็บไว้ในรีจีสเตอร์ A ค่า 8 มีเลขไบนารีเป็น 00001000 เมื่อถูกเลื่อนไปทางซ้ายหนึ่งครั้งจะมีค่าเป็น 00010000 ซึ่งผลลัพธ์จะเท่ากับ 10H ซึ่งจะทำให้คำสั่งทั้งสองมีผลเท่ากัน

ตัวอย่างเช่น

```
MOV     A ,#HIGH 1234H
MOV     A , #12H
```

จากตัวอย่างคำสั่งแรกจะเป็นการนำค่าไบต์สูงของ 1234H มาใส่ในรีจีสเตอร์ A ซึ่งค่าไบต์สูงคือ 12H ทำให้คำสั่งทั้งสองมีผลเท่ากัน

Relational Operators

ตัวดำเนินการแสดงความสัมพันธ์จะใช้กับ Operand สองตัวมาเปรียบเทียบกัน ซึ่งมีดังนี้

ตัวดำเนินการ		ความหมาย
=	EQ	เท่ากับ
< >	NE	ไม่เท่ากับ
<	LT	น้อยกว่า
< =	LE	น้อยกว่า หรือ เท่ากับ
>	GT	มากกว่า
> =	GE	มากกว่า หรือ เท่ากับ

ตารางที่ 3.4 ตัวดำเนินการเปรียบเทียบ

การใช้ตัวดำเนินการให้เลือกใช้อย่างใดอย่างหนึ่ง (“=” หรือ “EQ”) โดยผลลัพธ์ที่ได้จะเป็นจริงหรือเท็จถ้าเป็นจริงค่าในรีจีสเตอร์ A จะเป็น FFH ถ้าเป็นเท็จจะเป็น ooH พิจารณาตัวอย่างต่อไปนี้

```
MOV     A , #5 = 5
MOV     A , 5 NE 4
MOV     A , # 'X' LT 'Z'
MOV     A , # 'X' >= 'X'
MOV     A , #$ > 0
MOV     A , #100 GE 50
```

ผลลัพธ์ที่ได้ทุกคำสั่งจะมีค่าเท่ากับ

```
MOV     A , #oFFH
```

ตัวอย่างตัวดำเนินการต่าง ๆ และผลลัพธ์ที่ได้จะเป็นดังนี้

การกระทำ	ผลที่ได้
'B' - 'A'	0001H
8 / 3	0002H
155 MOD 2	0001H
4 * 4	0010H
8 AND 7	0000H
NOT 1	FFFEH
'A' SHL 8	4100H
LOW 65535	ooFFH
(8+1) * 2	0012H
5 EQ 4	0000H
'A' LT 'B'	FFFFH
3 < = 3	FFFFH

จากตัวอย่างในเรื่อง Timer ที่เก็บค่า -500 ลงในรีจีสเตอร์ TH1 และ TL1 จะเห็นว่ามีการใช้คำสั่ง HIGH และ LOW ดังนี้

```
COUNT    EQU   -500
         MOV   TH1, #HIGH VALUE
         MOV   TL1, #LOW VALUE
```

ตัวแอสแซมเบอร์จะแปลงค่า -500 เป็นเลขฐานสิบหก คือ FeoCH โดยค่า High คือ FEH และค่า Low คือ oCH

ถ้าหากมีการใช้ตัวดำเนินการหลาย ๆ ตัวในนิพจน์เดียวกัน จะมีลำดับการทำงานก่อนหลังดังนี้

ลำดับ	ตัวดำเนินการ
1	( )
2	HIGH LOW
3	* / MOD SHL SHR
4	+ -
5	EQ NE LT LE GT GE
6	= <> < <= > >=
7	NOT
8	OR XOR

ตารางที่ 3.5 ลำดับการทำงานของตัวดำเนินการ

ถ้าหากมีการใช้ตัวดำเนินการหลายตัวในบรรทัดเดียวกัน ผลลัพธ์ที่ได้จะแสดงได้ดังตัวอย่างต่อไปนี้

Expression		ผลลัพธ์ที่ได้
HIGH	('A' SHL 8)	0041H
HIGH	'A' SHL8	0000H
NOT	'A' – 1	FFBFH
'A' OR 'A' SHL 8		4141H

การเชื่อมต่อ 8255 กับ MCS-51

8255 เป็นไอซีพอร์ตขนานที่สามารถโปรแกรมการทำงานให้เป็นพอร์ตเอาต์พุตหรือพอร์ตอินพุตก็ได้ 8255เป็นไอซีขนาด 40 ขา ซึ่งได้รับการออกแบบให้มีสัญญาณเชื่อมโยงกับไมโครโปรเซสเซอร์เบอร์ 8080 แต่ก็เหมาะกับ MCS-51 เช่นเดียวกันไอซีเบอร์นี้เป็นไอซีที่ต่อเป็นพอร์ตให้ไมรโครโปรเซสเซอร์ได้ 3 พอร์ต

การเรียกพอร์ตของ 8255 จะเรียกว่าพอร์ต A, พอร์ต B และพอร์ต C โดยแต่ละพอร์ตจะมีขนาด 8 บิต และพอร์ต C จะแยกออกเป็นสองส่วนคือ PC0 – PC3 เรียกว่าพอร์ต C ล่าง จำนวน 4 บิต และพอร์ต C บน คือ PC4 – CP7 ที่พิเศษคือพอร์ตทุกพอร์ตเป็นได้ทั้งอินพุตและเอาต์พุต

ขาต่างๆของ 8255

D0-D7 เป็นขาข้อมูลอินพุตและเอาต์พุตที่จะต้องผ่านเข้าออกจากส่วนนี้ D0-D7 จึงเป็นส่วนที่จะต่อเข้ากับระบบบัสของไมโครคอนโทรลเลอร์ เพื่อให้ไมโครคอนโทรลเลอร์สามารถอ่านหรือเขียนข้อมูลจากพอร์ตผ่านทางบัสนี้

CS ขานี้เป็นขาอินพุตที่จะรับสัญญาณจากภายนอกเพื่อเลือกชิพ 8255 โดยเมื่อขานี้เป็นลอจิก "0" จะทำให้ 8255 ต่อเข้ากับระบบบัสของไมโครคอนโทรลเลอร์เพื่อให้ไมโครคอนโทรลเลอร์เขียนหรืออ่านข้อมูลจากพอร์ตนี้ได้

RD ขาสัญญาณการอ่าน เป็นสัญญาณอินพุตที่ส่งมาจาก CPU เมื่อสัญญาณนี้เป็น "0" และ CS เป็น "0"ด้วยตัว 8255 จะทำให้ตัว CPU อ่านข้อมูลจากบัสในขณะที่เป็นพอร์ตอินพุต

WR ขาสัญญาณการเขียน จะแอคทีฟเมื่อขา WR เป็น "0" สัญญาณนี้มาจาก CPU เมื่อต้องการเขียนข้อมูลลงบนพอร์ตที่กำหนด

A0-A1 ขาแอดเดรส ลอจิกทั้งสองข้างนี้ถอดรหัสออกมาได้ 4 ค่าเพื่อกำหนดค่ารีจีสเตอร์ภายในที่เชื่อมต่อเข้ากับพอร์ตอินพุตและเอาต์พุตของ 8255 เพื่อเลือกใช้พอร์ต A, B และ C

RESET ขารีเซต เป็นสัญญาณที่ส่งมาจากภายนอกเพื่อทำการรีเซต 8255 เพื่อเคลียร์สถานะต่างๆของ 8255 เมื่อ 8255 ได้รับการรีเซต มันจะกลับเข้าสู่โหมดอินพุตหรือทุกพอร์ตเป็นพอร์ตอินพุต

PA0 – PA7 เป็นสายสัญญาณที่เป็นพอร์ตของ 8255 ชื่อพอร์ต A การเลือกพอร์ตจะเลือกโดยขา A0-A1

PB0 – PB7 เป็นสายสัญญาณที่เป็นพอร์ตของ 8255 ชื่อพอร์ต B การเลือกพอร์ตจะเลือกโดยขา A0-A1

PC0 – PC7 เป็นสายสัญญาณที่เป็นพอร์ตของ 8255 การกำหนดพอร์ตนี้จะได้รับการกำหนดโดยขาแอดเดรส A0 – A1 พอร์ต C นี้แบ่งออกได้เป็นสองกลุ่มคือ PC0 – PC3 และ PC4 –PC7

ถ้าต้องการให้ 8255 ทำงานจะต้องให้ขา CS แอคทีฟ จากนั้นเลือกพอร์ตที่จะมาติดต่อโดย A0 – A1 รีจีสเตอร์แต่ละตัวใน 8255 จะได้รับการกำหนดค่าควบคู่กับสัญญาณ RD และ WR เพื่อแสดงการทำงานต่างๆดังนั้นสัญญาณต่างๆของขาควบคุมที่นำมาประกอบกันจะมีความหมายดังตารางที่ 11.1

RD	WR	A0	A1	ความหมาย
1	0	0	0	ส่งข้อมูลไปที่พอร์ต A
0	1	0	0	อ่านข้อมูลจากพอร์ต A
1	0	0	1	ส่งข้อมูลไปที่พอร์ต B
0	1	0	1	อ่านข้อมูลจากพอร์ต B
1	0	1	0	ส่งข้อมูลไปที่พอร์ต C
0	1	1	0	อ่านข้อมูลจากพอร์ต C
1	0	1	1	เขียนข้อมูลซึ่งเป็นรหัสควบคุม
0	1	1	1	ไม่ใช้

ตารางที่ 3.6 แสดงการเลือกพอร์ตของ 8255

การทำงานของไอซีเบอร์ 8255 จะทำงานได้ 3 โหมดคือ

1. Mode 0 : Basic I/O
2. Mode 1 : Strobed I/o
3. Mode 2 : Bidirctional Bus

การจะให้ 8255 ทำงานแต่ละโหมดจะเลือกโดยการโปรแกรมให้กับ 8255 คำสั่งที่โปรแกรมจะมี 8 บิตแต่ละบิตจะมีความหมายดังนี้

บิต D7 เป็นบิตแสดงรหัสคำสั่งควบคุม ถ้าบิตนี้เป็น "1" จะหมายถึงรหัสควบคุมนี้มีผลต่อการเซตโหมดต่างๆของ 8255

บิต D6 และ **บิต D5** เป็นโหมดของพอร์ต A ซึ่งมีการทำงาน 3 โหมด คือโหมด 0, 1 และ2

บิต D4 ถ้ามีค่าเท่ากับ "0" หมายถึงการกำหนดพอร์ต A เป็นเอาต์พุต ถ้ามีค่าเป็น "1" จะกำหนดให้พอร์ต A เป็นพอร์ตอินพุต

บิต D3 เป็นบิตที่บอกการเซตพอร์ต C บน ถ้าเป็น "0" จะทำให้พอร์ต C บนเป็นเอาต์พุต

บิต D2 เป็นโหมดที่บอกการเซตโหมดของพอร์ต B ถ้าเป็น "0" หมายถึงเลือกพอร์ต B เป็นโหมด 0 และถ้าเป็น "1" คือการเลือกโหมด 1

บิต D1 เป็นการกำหนดอินพุตและเอาต์พุตของพอร์ต B ถ้าเป็น "0" คือพอร์ตเอาร์พุต และถ้าเป็น "1" คือพอร์ตอินพุต

บิต D0 เป็นบิตที่บอกการเซตพอร์ต C ล่างถ้าเป็น "0" จะทำให้พอร์ต C ล่างเป็นเอาต์พุต

บิตที่	กลุ่ม	ความหมาย
D0	B	พอร์ต C ล่าง 1 = Input 0 = Output
D1	B	พอร์ต B 1 = Input 0 = Output
D2	B	เลือกโหมด 1 = โหมด 0 0 = โหมด 1
D3	A	พอร์ต C บน 1 = Input 0 = Output
D4	A	พอร์ต A 1 = Input 0 = Output
D5 D6	A	เลือกโหมด 00 = โหมด 0 01 = โหมด 1 1x = โหมด 2
D7		โหมดเซตแอคทีฟ 1 = แอคทีฟ

ตารางที่ 3.7 แสดงความหมายของบิตควบคุม

การทำงานในโหมด 0 จะเป็นการทำงานแบบโหมดอินพุตและเอาต์พุตพื้นฐาน ซึ่งจะประกอบด้วยพอร์ต 8 บิต 2 พอร์ตคือ พอร์ต A และพอร์ต B พอร์ตขนาด 4 บิต 2 พอร์ตคือ พอร์ต C บนและพอร์ต C ล่าง ทุกๆพอร์ตสามารถโปรแกรมให้เป็นอินพุตหรือเอาต์พุตได้ซึ่งมีทั้งหมด 16 รูปแบบด้วยกัน รหัสควบคุมในโหมดนี้แสดงในตารางที่ 3.8

Control Word	Port A	Port C บน	Port B	Port C ล่าง
80 H	Output	Output	Output	Output
81 H	Output	Output	Output	Input
82 H	Output	Output	Input	Output
83 H	Output	Output	Input	Input
88 H	Output	Input	Output	Output
89 H	Output	Input	Output	Input
8A H	Output	Input	Input	Output
8B H	Output	Input	Input	Input
90 H	Input	Output	Output	Output
91 H	Input	Output	Output	Input
92 H	Input	Output	Input	Output
93 H	Input	Output	Input	Input
98 H	Input	Input	Output	Output
99 H	Input	Input	Output	Input
9A H	Input	Input	Input	Output
9B H	Input	Input	Input	Input

ตารางที่ 3.8 แสดงหน้าที่ของพอร์ตต่าง ๆ ตามรหัสควบคุม

การเชื่อมต่อ 8255 กับ MCS – 51

การออกแบบให้ MCS – 51 ติดต่อกับ 8255 ก็เหมือนการออกแบบให้ติดต่อกับพอร์ตทั่ว ๆ ไป คือมอง 8255 เหมือนเป็นหน่วยความจำข้อมูลภายนอก ซึ่งเป็นหน่วยความจำขนาด 4 ไบต์ คือพอร์ต A พอร์ต B พอร์ตC และพอร์ตควบคุม โดยสัญญาณที่ได้จากวงจรถอดรหัสพอร์ตจะนำมาต่อกับขา CS ส่วนขา RD, WR, A0, A1 ของ 8255 สามารถต่อโดยตรงกับ MCS –51 โดยตรงได้ ในที่นี้จะยกตัวอย่างไอซีเบอร์ 74LS138 มาถอดรหัสดังรูปที่ 3.2

จากรูปจะเห็นได้ว่าถ้า MCS – 51 ติดต่อกับหน่วยความจำในตำแหน่ง 0000H ถึง 0FFFFH ซึ่งมีทั้งหมด 4096 ตำแหน่ง จะเป็นการติดต่อกับ 8255 ตัวนี้ทั้งหมด เพราะว่าเราไม่สนใจสัญญาณ A2 – A11 แต่จริงๆแล้วจะเป็นการติดต่อกับ 8255 ได้ 4 ตำแหน่งเท่านั้น โดยการอ้างของ A0 และ A1 ถ้าเราต่อวงจรตามรูปข้างต้นพอร์ตต่าง ๆ ของ 8255 จะมีหมายเลขดังตารางที่ 3.9

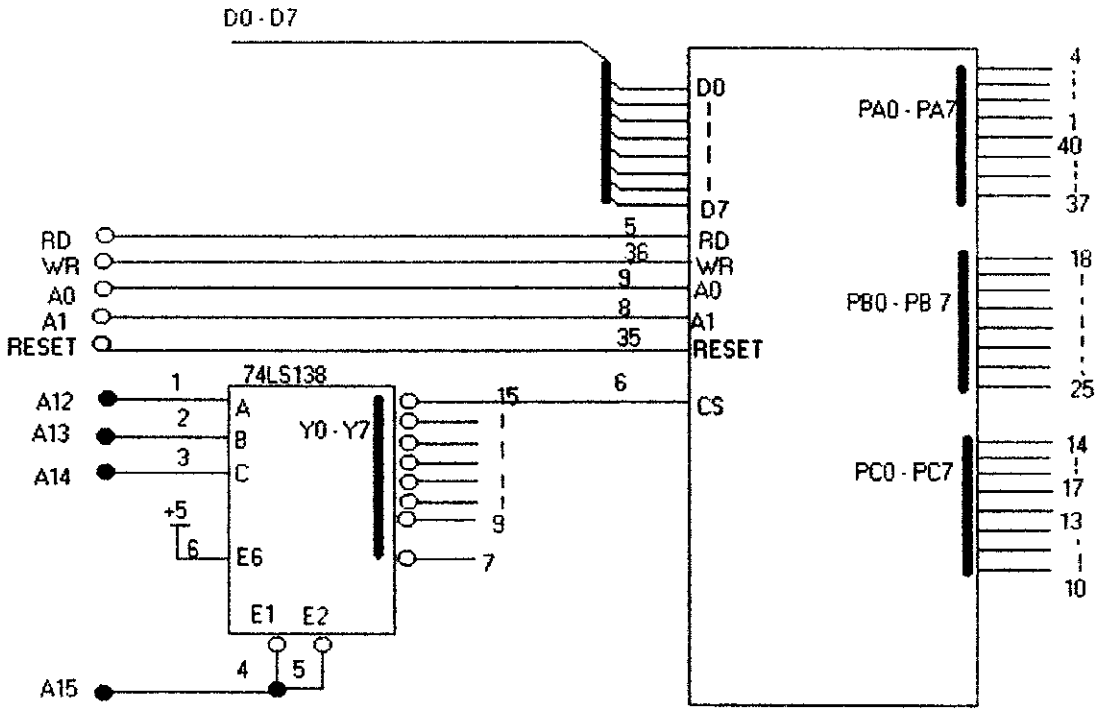
พอร์ต	A1	A0	หมายเลขพอร์ต
A	0	0	0000H
B	0	1	0001H
C	1	0	0002H
Control	1	1	0003H

ตารางที่ 3.9 แสดงหมายเลขพอร์ตของ 8255

ถ้าต้องการโปรแกรมให้ทุกพอร์ตเป็นเอาต์พุตหมด เราสามารถเขียนโปรแกรมได้ดังนี้

```
PORT_A      EQU    0000H
PORT_B      EQU    0001H
PORT_C      EQU    0002H
CON_ PORT   EQU    0003H
            MOV    DPTR,# CON_ PORT   :ส่งค่าไปที่พอร์ตควบคุมให้เป็นเอาต์พุตทั้งหมด
            MOV    A,#80H
            MOVX   @DPTR,A
            MOV    DPTR,#PORT_A
            MOVX   @DPTR,A            :ส่งค่า 1 บิตออกไปที่พอร์ต A
```

การออกแบบพอร์ตต่างๆขอให้ระลึกไว้ว่าการอ้างพอร์ตของ MCS – 51 จะมองพอร์ตเป็นหน่วยความจำภายนอก ดังนั้นการถอดรหัสพอร์ตจะต้องเหลือเนื้อที่ในหน่วยความจำข้อมูลภายนอกด้วย ซึ่งสามารถออกแบบได้หลายวิธี และ 8255 ก็สามารถมีได้หลายตัวเช่นกัน ตัวอย่างต่อไปจะเป็นการถอดรหัสสำหรับหน่วยความจำภายนอกและพอร์ตต่างๆ ที่จะมีขึ้นอาจออกแบบได้ดังรูปที่ 3.2 จากตัวอย่างเป็นการถอดรหัสโดยใช้ ไอซีเบอร์ 74LS138 ซึ่งจะมีสัญญาณจากการถอดรหัส 8 เส้น แต่ละเส้นสามารถอ้างหน่วยความจำได้ 8 kB จากตัวอย่างอาจใช้ Y0 ต่อกับหน่วยความจำ RAM ขนาดไม่เกิน 8 kB ได้ ซึ่งอ้างตำแหน่ง 0000H – 1FFFH ส่วน Y1 และ Y2 อาจต่อกับ 8255 อีก 2 ตัว (ซึ่งสามารถต่อเพิ่มได้อีกถ้าต้องการ) จาก 8255 ตัวแรกที่ต่อกับ Y1 ตำแหน่ง 2000H จะเป็นพอร์ต A ตำแหน่ง 2001H และ 2002H จะเป็นพอร์ต B และพอร์ต C ตามลำดับ ส่วนตำแหน่ง 2004H จะเป็นพอร์ตควบคุม



รูปที่ 3.2 แสดงตัวอย่างของการใช้ 74LS138 ถอดรหัส