หน่วยที่ 6 การรับส่งข้อมูลกับอุปกรณ์เชื่อมต่อภายนอก

อดิศักดิ์ ชิณะวงศ์

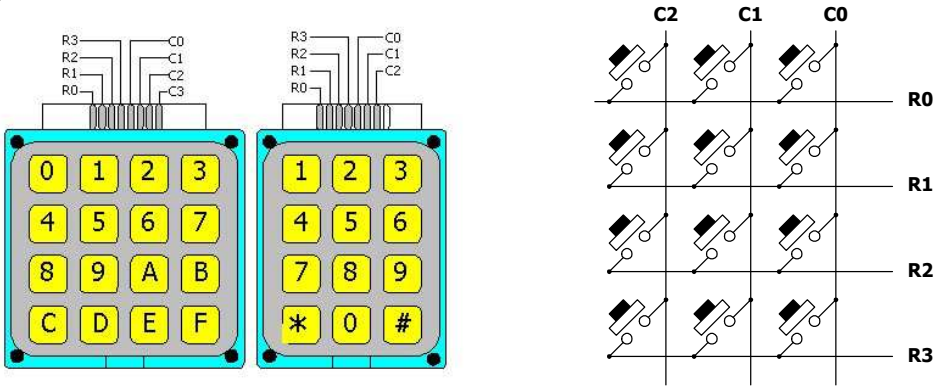
เอกสารประกอบการเรียนวิชาไมโครคอนโทรลเลอร์

เผยแพร่ที่ www.Adisak51.com

1. การเชื่อมต่อกับคีย์บอร์ดแบบเมตริกซ์

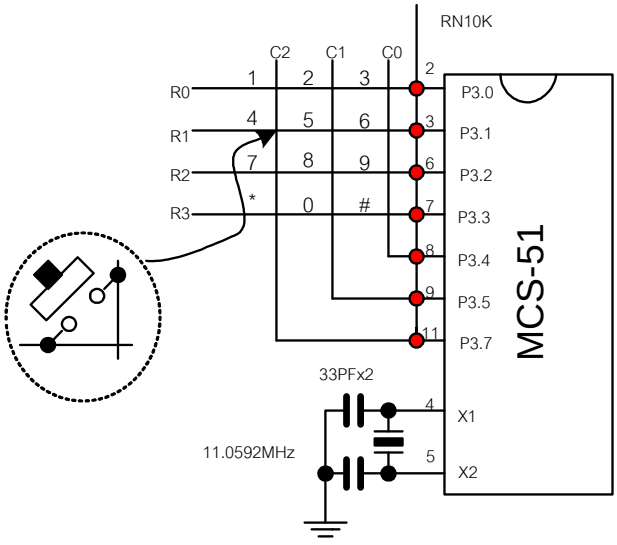
1.1 การต่อคีย์แบบเมตริกซ์

การให้ไอซี MCS-51 ทำงานเป็นอินพุตพอร์ต เพื่อรับค่าข้อมูลจากคีย์สวิตช์ เมื่อมีความต้องการใช้งานสวิตช์ ที่มีจำนวนมากกว่าขาของพอร์ตที่มีอยู่ ดังนั้นจึงต้องใช้วิธีการต่อคีย์สวิตช์ให้เป็นแบบเมตริกซ์ โดยสามารถเพิ่มจำนวนคีย์สวิตซ์ได้เท่ากับจำนวนของ แถวคูณด้วยจำนวนคอลัมน์ (Row × Column) แสดงดังภาพที่ 6.1



ภาพที่ 6.1 คีย์สวิตช์แบบเมตริกซ์ขนาด 4×4 และ 3×4

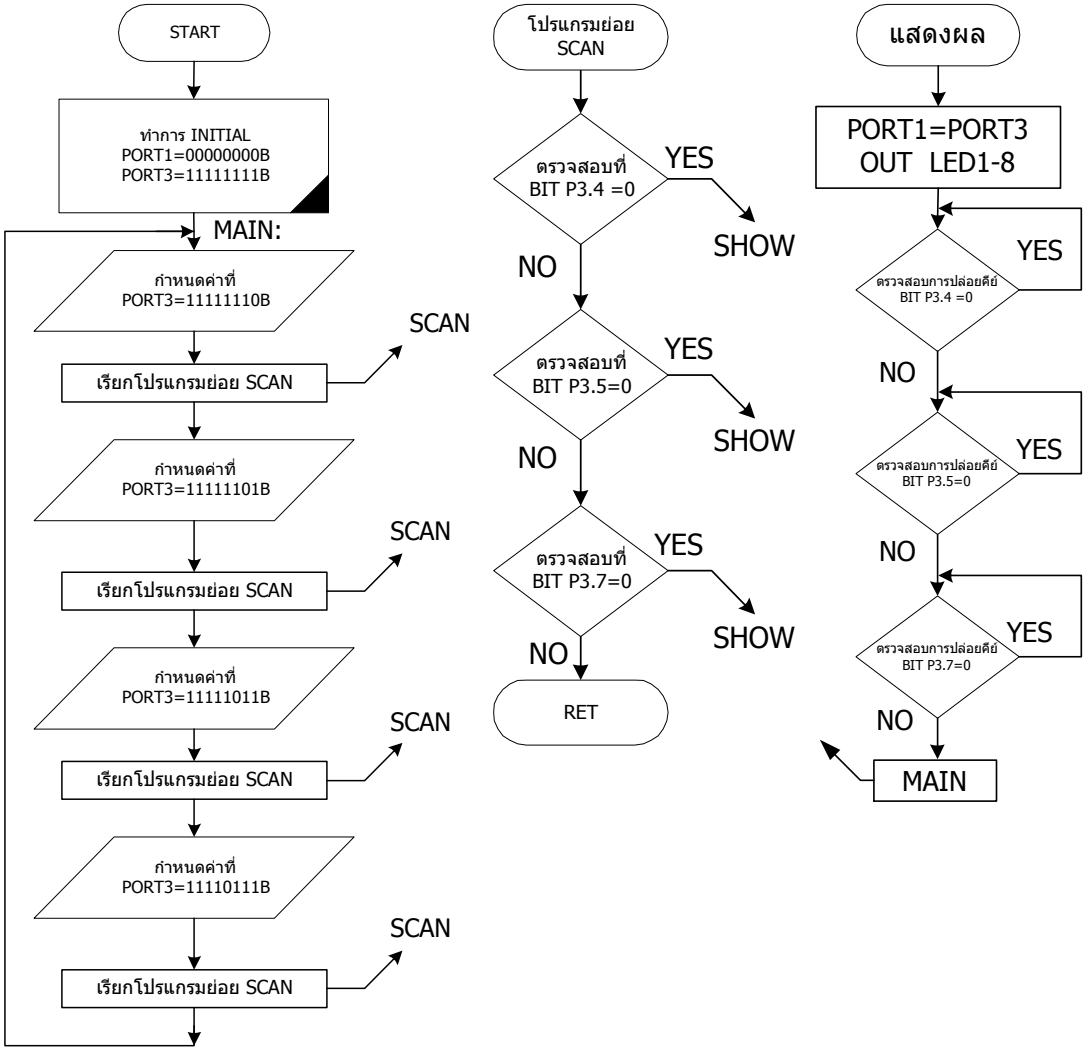
การกำหนดให้พอร์ต P3.0, P3.1, P3.2, P3.3 เป็นพอร์ตเอาต์พุตโดยให้ ผลัดกันส่งข้อมูลสถานะลอจิก “0” ออกทีละบิต โดยกำหนดเป็น 0111, 1011, 1101 และ 1110 ตามลำดับ แล้วรับข้อมูลเข้ามาที่พอร์ต P3.4, P3.5, P3.7 เพื่อตรวจสอบสถานะของคีย์ที่ถูกกดบ้าง และในทุกๆครั้งที่ผลัดกันส่งข้อมูลเป็นสถานะลอจิก “0” พอร์ตเอาต์พุต P3.0, P3.1, P3.2, P3.3 นำสถานะลอจิก ที่ส่งออกไป ในขณะนั้นมารวมกับข้อมูลที่รับเข้ามาทางอินพุตพอร์ต P3.4, P3.5, P3.7 ขณะคีย์ถูกกด ซึ่งเป็นค่าคีย์โค๊ดของแต่ละคีย์ การต่อวงจรกับ Row และ Column แสดงดังภาพที่ 6.2



ภาพที่ 6.2 แสดงการต่อวงจรแบบเมตริกซ์กับไอซี MCS-51

1.2 การเขียนโปรแกรมคีย์เมตริกซ์

ตัวอย่างที่ 1 กำหนดให้เขียนโปรแกรมรับค่ารหัสตำแหน่งคีย์ ของสวิตช์แต่ละตัวที่ต่อแบบเมตริกซ์ขนาด 3×4 โดยกำหนดให้พอร์ต P3.0, P3.1, P3.2, P3.3 เป็นเอาต์พุต แล้วรับข้อมูลเข้ามาที่พอร์ต P3.4, P3.5, P3.7 เพื่อตรวจสอบว่ามีคีย์ใดถูกกดบ้าง หลังจากนั้นจึงนำไป แสดงผลที่พอร์ต P1 ซึ่งเป็นค่ารหัสของคีย์ แต่ละคีย์ ขั้นตอนการเขียนโปรแกรมแสดงเป็นผังงานแสดงดังภาพที่ 6.3



ภาพที่ 6.3 แสดงผังงานของโปรแกรมแสดงค่าคีย์โค๊ด

```
; ************************************************************
; **      OUT:    P3.0, P3.1, P3.2, P3.3  IN: P3.4, P3.5, P3.7          **
; **              P1 Display KEY CODE                                   **
; ************************************************************
```

```
COLUMN0     BIT     P3.4    ; กำหนดค่า COLUMN0 แทน P3.4
COLUMN1     BIT     P3.5    ; กำหนดค่า COLUMN1 แทน P3.5
COLUMN2     BIT     P3.7    ; กำหนดค่า COLUMN2 แทน P3.7
        ORG     0000H       ; แอดเดรสเริ่มต้นของโปรแกรม
        MOV     P1,#00H     ; กำหนดข้อมูลพอร์ต P1 มีค่าเท่ากับ 00000000B ให้LED ดับทุกดวง
        MOV     P3,#0FFH    ; กำหนดให้พอร์ต P3 ทุกบิตเป็นลอจิก "1"
; ***************************************************************************
; **      โปรแกรมในการส่งข้อมูลโดยให้เป็นลอจิก “0” ที่ P3.0, P3.1, P3.2, P3.3 ทีละบิต **
; ***************************************************************************
MAIN:   MOV     P3,#11111110B       ;ให้บิต P3.0 เป็น "0" เป็นแถวแรก R0
        ACALL   SCAN                ; เรียกโปรแกรมย่อย SCAN เพื่อตรวจสอบการกดคีย์
        MOV     P3,#11111101B       ;ให้เลื่อนบิต P3.1 เป็น 0 เป็นแถวที่สอง R1
        ACALL   SCAN                ; เรียกโปรแกรมย่อย SCAN เพื่อตรวจสอบการกดคีย์
        MOV     P3,#11111011B       ;ให้เลื่อนบิต P3.2 เป็น 0 เป็นแถวที่สาม R2
        ACALL   SCAN                ; เรียกโปรแกรมย่อย SCAN เพื่อตรวจสอบการกดคีย์
        MOV     P3,#11110111B       ;ให้เลื่อนบิต P3.3 เป็น 0 เป็นแถวที่สี่ R3
        ACALL   SCAN                ; เรียกโปรแกรมย่อย SCAN เพื่อตรวจสอบการกดคีย์
        SJMP    MAIN                ; วนกลับไปทำที่เลเบล MAIN ใหม่
; ***************************************************************************************
; **  โปรแกรมย่อยการตรวจสอบการกดที่คีย์สวิตช์ โดยให้ข้อมูลเป็นลอจิก “0” ในบิตที่ถูกกด      **
; ***************************************************************************************
SCAN:   JNB     COLUMN0, SHOW   ; ถ้า COLUMN0 เป็นลอจิก "0" กระโดดไปทำที่เลเบล SHOW
        JNB     COLUMN1, SHOW   ; ถ้า COLUMN1 เป็นลอจิก "0" กระโดดไปทำที่เลเบล SHOW
        JNB     COLUMN2, SHOW   ; ถ้า COLUMN2 เป็นลอจิก "0" กระโดดไปทำที่เลเบล SHOW
        RET                     ; ออกจากโปรแกรมย่อย
; ***************************************************************************
; **      โปรแกรมย่อยนำข้อมูล P3 แสดงผลที่ P1 และตรวจการปล่อยคีย์ ที่สถานะลอจิก “0”  ***
; ***************************************************************************
SHOW:   MOV     P1, P3          ; นำ P3 มาแสดงผลที่ พอร์ต P1 แสดงเป็นค่า KEY CODE
        ORL     P1,#01000000B ; บิต P1.6 เป็น “1” ตลอด
        JNB     COLUMN0, $    ; ถ้า P3.4 เป็นลอจิก “0” ให้วนบรรทัดเดิมเพื่อตรวจสอบการปล่อยคีย์
        JNB     COLUMN1, $    ; ถ้า P3.5 เป็นลอจิก “0” ให้ทำวนบรรทัดเดิมเพื่อตรวจสอบการปล่อยคีย์
```

```
JNB     COLUMN2, $   ; ถ้า P3.7 เป็นลอจิก “0” ให้ทำวนบรรทัดเดิมเพื่อตรวจสอบการปล่อยคีย์
MOV     SP,#07H      ; กำหนดข้อมูลในตัวชี้สแตกไปที่แอดเดรส 07H ของหน่วยความจำข้อมูล
SJMP    MAIN         ; ถ้าคีย์ถูกปล่อยหมดให้วนไปทำใหม่ที่เลเบล MAIN
END
```

ขณะเรียกใช้โปรแกรมย่อยด้วยคำสั่ง ACALL SCAN ค่าของโปรแกรมเคาน์เตอร์ (PC) ถูกเก็บไว้ในสแตก แต่ในโปรแกรมย่อย SCAN หากกดคีย์สวิตช์ มีเงื่อนไขให้กระโดดไปทำที่เลเบล SHOW หลังจากทำงานตามคำสั่งเสร็จสิ้น กระโดดไปทำคำสั่งใหม่ที่เลเบล MAIN ดังนั้นจึงไม่ได้ทำคำสั่ง RET เพื่อคืนค่าในสแตก ทำให้สแตกถูกใช้งานไปเป็นจำนวน 2 ไบต์ ดังนั้นจึงเพิ่มคำสั่ง MOV SP,#07H เพื่อเป็นการกำหนดค่าตำแหน่งแอดเดรสให้กับรีจิสเตอร์ SP ก่อนกลับไปทำที่เลเบล MAIN

คำสั่ง ORL P1, #01000000B เป็นการกำหนดให้บิต P1.6 มีสถานะเป็น “1” ตลอด หากใช้ไอซีเบอร์ AT89CX051 พอร์ต P3 บิต P3.6 เปลี่ยนสถานะลอจิกตามค่าของ P1.0 และ P1.1 ที่เป็นอินพุตของวงจรเปรียบเทียบ จากโปรแกรมสามารถสรุปได้เป็นค่าคีย์โค๊ดดังตารางที่ 6.1

ตารางที่ 6.1 ค่าคีย์โค๊ดจากโปรแกรมตัวอย่างที่ 1

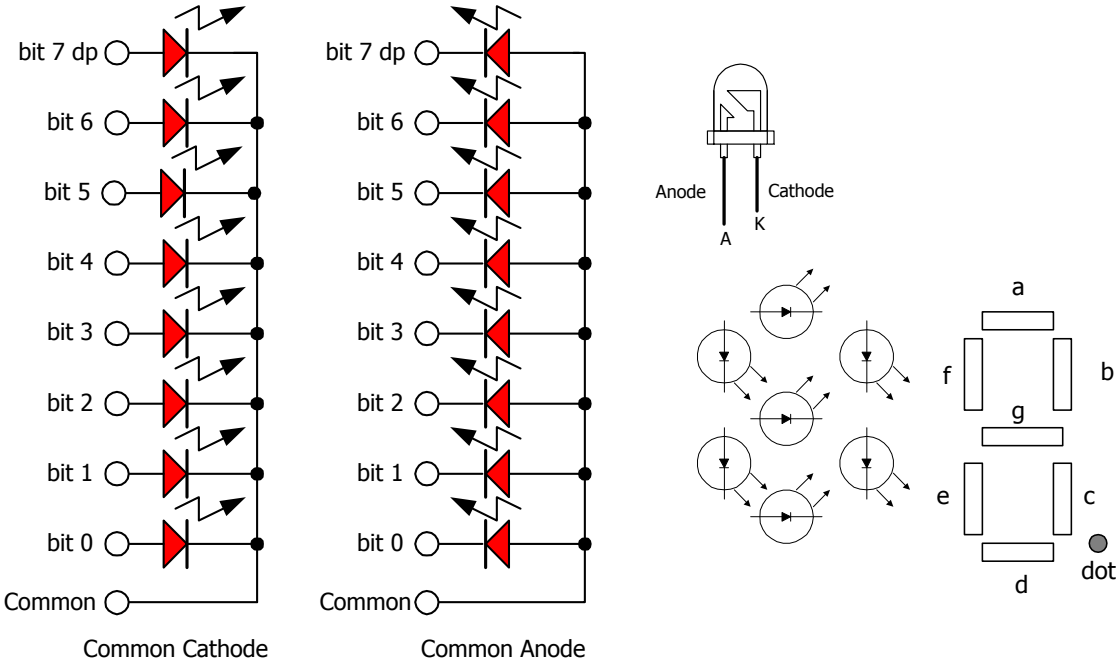
KEY	ROW & COLUMN	P3.7 C2	P3.6 NO	P3.5 C1	P3.4 C0	P3.3 R3	P3.2 R2	P3.1 R1	P3.0 R0	BINARY	HEX
1	R0C2	0	1	1	1	1	1	1	0	01111110B	7EH
2	R0C1	1	1	0	1	1	1	1	0	11011110B	DEH
3	R0C0	1	1	1	0	1	1	1	0	11101110B	EEH
4	R1C2	0	1	1	1	1	1	0	1	01111101B	7DH
5	R1C1	1	1	0	1	1	1	0	1	11011101B	DDH
6	R1C0	1	1	1	0	1	1	0	1	11101101B	EDH
7	R2C2	0	1	1	1	1	0	1	1	01111011B	7BH
8	R2C1	1	1	0	1	1	0	1	1	11011011B	DBH
9	R2C0	1	1	1	0	1	0	1	1	11101011B	EBH
*	R3C2	0	1	1	1	0	1	1	1	01110111B	77H
0	R3C1	1	1	0	1	0	1	1	1	11010111B	D7H
#	R3C0	1	1	1	0	0	1	1	1	11100111B	E7H

2. การเชื่อมต่อกับแอลอีดี 7 ส่วน

2.1 แอลอีดี 7 ส่วน (LED 7 Segments)

หากต้องการให้แอลอีดีติดสว่าง ต้องให้ขาแอโนดเป็นลอจิก "1” ขาแคโถดเป็นลอจิก “0” ทำนองเดียวกันหากนำมาต่อรวมกัน 7 ตัวเรียกว่า แอลอีดี 7 ส่วน มีการต่อ 2 แบบคือ แบบแอโนดร่วม (Common Anode) เป็นการนำเอาขาแอโนดทั้งหมดมารวมกันเป็นจุดร่วม (Common) ส่วนขาที่เหลือเป็น

อินพุต แบบที่สองคือแบบแคโถดร่วม (Common Cathode) เป็นการนำเอาขาแคโถดทั้งหมด มาต่อรวมกันเป็นจุดร่วม แสดงดังภาพที่ 6.4



ภาพที่ 6.4 แสดงการต่อแอลอีดี 7 ส่วนแบบแอโนดร่วมและแบบแคโถดร่วม

หากนำพอร์ต P1ต่อแอลอีดีของแต่ละตัวโดยเรียงค่าตามตารางที่ 6.2 กำหนดการแสดงผลที่แอลอีดีให้เป็นตัวเลข หากเป็นแบบแอโนดร่วมต้องกำหนดให้สว่างด้วยลอจิก “0” และดับด้วย ลอจิก “1” แต่ถ้าเป็นแบบแคโถดร่วมกำหนดให้สว่าง ด้วยลอจิก “1” และดับด้วยลอจิก “0” หลังจากได้ทุกตัวเลข นำค่ามาแปลงเป็นเลขฐานสิบหก สรุปได้ดังตารางที่ 6.3 (P1.7 แสดงผลเป็นจุด และกำหนดให้ดับตลอด)

ตารางที่ 6.2 กำหนดการต่อแอลอีดีเพื่อแสดงผลที่พอร์ต P1

D7 (bit 7)	D6(bit 6)	D5(bit 5)	D4(bit 4)	D3(bit 3)	D2(bit 2)	D1(bit 1)	D0(bit 0)
dot	g	f	e	d	c	b	a

ในภาพที่ 6.5 แสดงวิธีการกำหนดสถานะลอจิกแต่ละบิตเพื่อให้แสดงผลที่แอลอีดี 7 ส่วนแบบแคโถด และแบบแอโนดร่วม

Common Cathode									
bit7 dot	bit6 g	bit5 f	bit4 e	bit3 d	bit2 c	bit1 b	bit0 a	HEX	แสดงผล
0	0	1	1	1	1	1	1	3FH	
0	0	0	0	0	1	1	0	06H	
0	1	0	1	1	0	1	1	5BH	
0	1	0	0	1	1	1	1	4FH	
0	1	1	0	0	1	1	0	66H	
0	1	1	0	1	1	0	1	6DH	
0	1	1	1	1	1	0	1	7DH	
0	0	0	0	0	1	1	1	07H	
0	1	1	1	1	1	1	1	7FH	
0	1	0	1	1	1	1	1	6FH	

Common Anode									
bit7 dot	bit6 g	bit5 f	bit4 e	bit3 d	bit2 c	bit1 b	bit0 a	HEX	แสดงผล
1	1	0	0	0	0	0	0	C0H	
1	1	1	1	1	0	0	1	F9H	
1	0	1	0	0	1	0	0	A4H	
1	0	1	1	0	0	0	0	B0H	
1	0	0	1	1	0	0	1	99H	
1	0	0	1	0	0	1	0	92H	
1	0	0	1	0	0	1	0	82H	
1	1	1	1	1	0	0	0	F8H	
1	0	0	0	0	0	0	0	80H	
1	0	0	1	0	0	0	0	90H	

ภาพที่ 6.5 แสดงผลแอลอีดี 7 ส่วนแบบแคโถดร่วมและแบบแอโนดร่วมของเลข 0-9

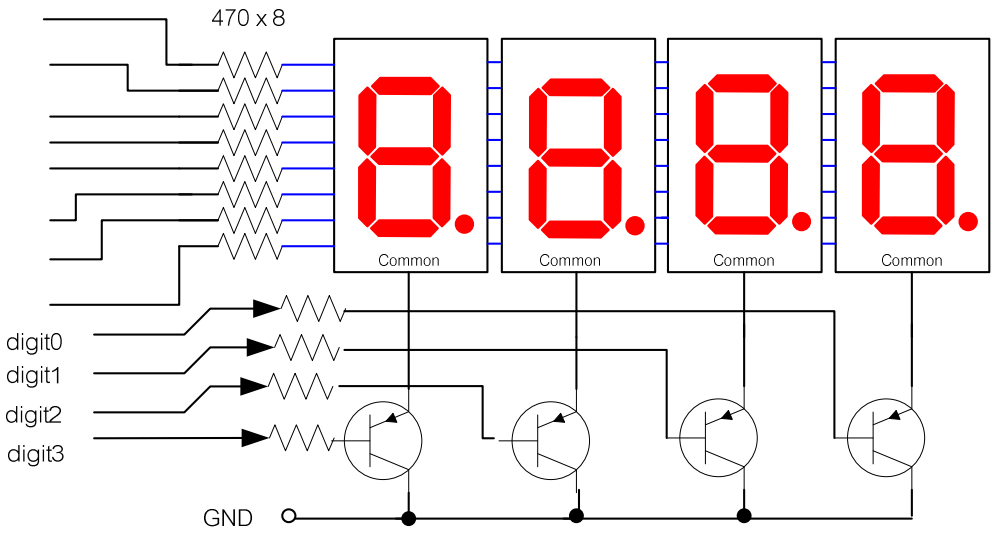
ตารางที่ 6.3 แสดงผลของการแปลงเป็นเลขฐาน 16 แบบแคโถดร่วม และแบบ แอโนดร่วม

ตัวเลขที่แสดงผล	0	1	2	3	4	5	6	7	8	9
Common Anode	C0	F9	A4	B0	99	92	82	F8	80	90
Common Cathode	3F	06	5B	4F	66	6D	7D	07	7F	6F

2.2 แอลอีดี 7 ส่วนแบบหลายหลัก

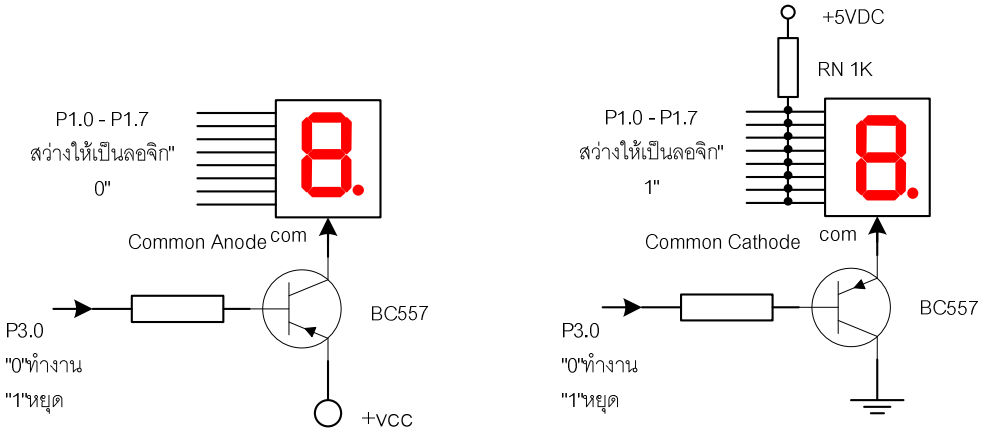
การใช้งานไอซี MCS-51 เพื่อแสดงผลที่แอลอีดี 7 ส่วนเป็นเอาต์พุต หากต่อรวมกันโดยตรงหลายๆหลัก เมื่อแอลอีดีติดสว่างทุกเซกเมนต์พร้อมกัน ต้องใช้กระแสไฟที่จ่ายให้แอลอีดีมากขึ้น ดังนั้นหากต้องการแสดงผลด้วยแอลอีดี 7 ส่วนหลายตัวพร้อมกัน จึงใช้การแสดงผลแบบมัลติเพล็ก (Multiplexed Display) โดยต่อขาของแอลอีดี 7 ส่วนในแต่ละเซกเมนต์ ขนานกับเซกเมนต์เดียวกันของแอลอีดี 7 ส่วนตัวอื่นๆ ทุกตัว แสดงดังภาพที่ 6.6

การแสดงผลแบบมัลติเพล็กโดยให้แอลอีดี 7 ส่วนติดทีละหลัก ควบคุมที่ขาคอมมอน แต่ละตัวให้ทำงาน ต้องใช้ความเร็วในการสลับหลักโดยสามารถมองเห็นการดับของแอลอีดี 7 ส่วนแต่ละหลักได้ทัน ทำให้มองเห็นเหมือนกับว่า แสดงผลทุกหลักพร้อมๆ กัน ซึ่งการใช้งานในลักษณะมัลติเพล็ก มีข้อดีหลายประการ เช่น แอลอีดี 7 ส่วนติดได้เพียงทีหนึ่งหลักดังนั้นจึงใช้แหล่งจ่ายไฟให้แอลอีดี 7 ส่วนเพียงแค่หลักเดียว ถึงแม้มี จำนวนหลายหลักสว่างพร้อมๆ กันและ ตัวต้านทานที่ทำหน้าที่จำกัดกระแสให้แต่ละเซกเมนต์ใช้ทั้งหมด 8 ตัว (รวมทั้งที่แสดงผลเป็นจุด Dot) ดังนั้นวิธีนี้จึงทำให้ประหยัดอุปกรณ์ที่นำมาต่อร่วม และปริมาณของกระแสไฟที่ต้องใช้ในระบบทั้งหมด



ภาพที่ 6.6 แสดงวงจรการใช้งานในลักษณะการแสดงผลแบบมัลติเพล็ก

วิธีการต่อและใช้งานแอลอีดี 7 ส่วนทั้งสองแบบแสดงดังภาพที่ 6.7 โดยแบบคอมมอนแคโถดควรต่อตัวต้านทานค่าประมาณ 1 กิโลโอห์ม ระหว่างตำแหน่งของแต่ละบิต เพื่อขับแอลอีดี 7 ส่วนในแต่ละเซกเมนต์ กับแหล่งจ่ายไฟ +VCC เนื่องจากความต้านทานภายในของพอร์ตมีค่ามากทำให้กระแสที่ขับให้แอลอีดีสว่างไม่เพียงพอ

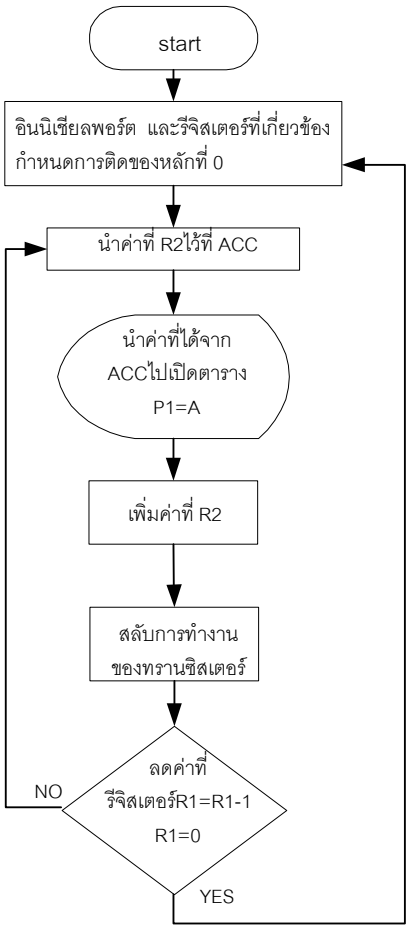


ภาพที่ 6.7 แสดงการใช้ทรานซิสเตอร์แบบ PNP กำหนดสถานะที่ขาคอมมอน

2.3 การเขียนโปรแกรมเชื่อมต่อกับแอลอีดี 7 ส่วน

ตัวอย่างที่ 2 เขียนโปรแกรมสลับหลัก การแสดงผลแอลอีดี 7 ส่วน โดยใช้ทรานซิสเตอร์ชนิด PNP ทำหน้าที่เป็นสวิตช์ ในการมัลติเพล็ก โดยกำหนดสถานะของลอจิกให้กับขาของพอร์ตที่ต่อกับขาเบสทรานซิสเตอร์ เพื่อควบคุมการปิด เปิดขาคอมมอนของแอลอีดี 7 ส่วน กำหนดสถานะของพอร์ตเป็นลอจิก “0” หรือใช้คำสั่ง CLR Bit จะทำให้ทรานซิสเตอร์นำกระแสได้ ส่งผลให้แอลอีดี 7 ส่วน หลักนั้นๆ ทำงาน ผังงานของโปรแกรมแสดงดังภาพที่ 6.8

```
; ******************************************************
; **       ON-OFF TRANSISTER DIGIT1 P3.1, DIGIT0 P3.0       **
; ******************************************************
DIGIT1 BIT     P3.1          ; ให้บิต P3.1 ควบคุมที่ขาเบสของทรานซิสเตอร์หลักที่ 1
DIGIT0 BIT     P3.0          ; ให้บิต P3.0 ควบคุมที่ขาเบสของทรานซิสเตอร์หลักที่ 0
       ORG     0000H         ; เริ่มต้นโปรแกรมที่แอดเดรส 0000H
       SETB    DIGIT1        ; ให้ขาเบสของทรานซิสเตอร์หลักที่ 1 เป็นลอจิก “1” หยุดนำกระแส
       CLR     DIGIT0        ; ให้ขาเบสของทรานซิสเตอร์หลักที่ 0 เป็นลอจิก “0” นำกระแส
MAIN:  MOV     P1,#00H       ; กำหนดให้พอร์ต P1=0000000B
       CLR     A             ; เคลียร์ค่าข้อมูลในรีจิสเตอร์ ACC=00000000B
       MOV     R2,#00H       ; กำหนดให้ข้อมูลในรีจิสเตอร์ R2=00H
```

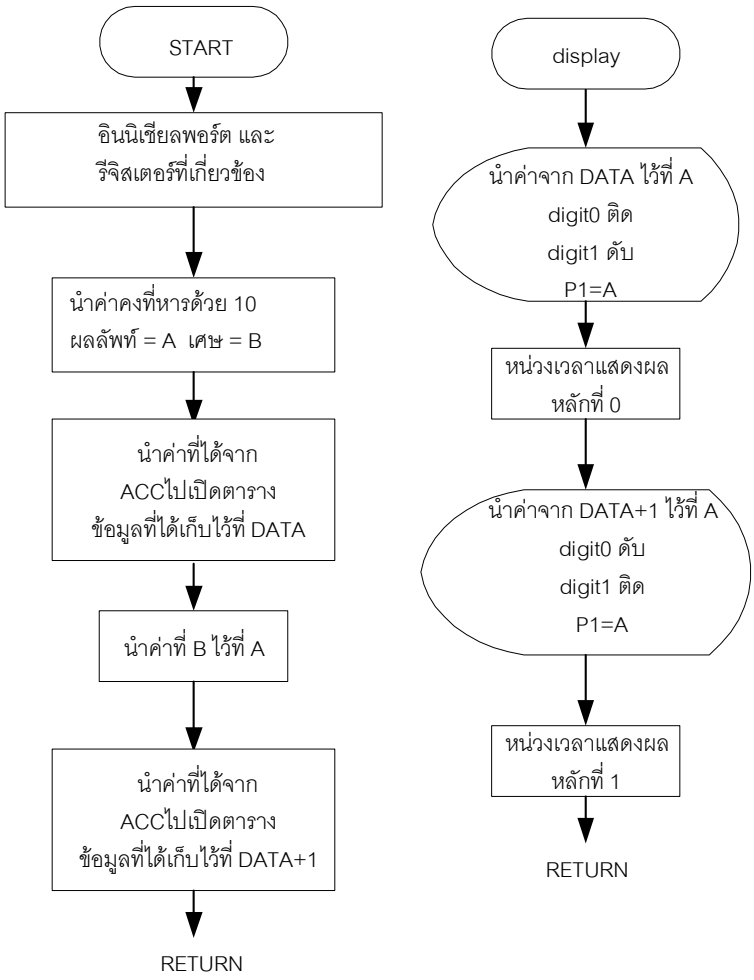


ภาพที่ 6.8 แสดงผังงานของโปรแกรม

```
        MOV   R1,#0AH       ; ให้รีจิสเตอร์ R1 = 0AH เพื่อเป็นตัวนับรอบของการแสดงค่าตัวเลข 0-9
        MOV   DPTR,#TABLE
LOOP:   MOV   A,R2
        MOVC  A,@A+DPTR
        MOV   P1,A          ; นำค่าข้อมูลที่รีจิสเตอร์ A ออกพอร์ต P1
        ACALL DELAY         ; เรียกโปรแกรมย่อยหน่วงเวลา
        INC   R2            ; เพิ่มค่าข้อมูลที่รีจิสเตอร์ R2
        CPL   DIGIT0        ; กลับค่าสถานะลอจิกที่ขาเบสของทรานซิสเตอร์
        CPL   DIGIT1        ; กลับค่าสถานะลอจิกที่ขาเบสของทรานซิสเตอร์
        DJNZ  R1,LOOP       ; ลดค่า R1 ตรวจสอบ R1 = 0 หรือไม่ ถ้าไม่ให้กระโดดไป LOOP
        SJMP  MAIN          ; กระโดดไปทำใหม่ที่เลเบล MAIN
        ;*** โปรแกรมย่อยการหน่วงเวลาเพื่อแสดงผลที่ตัวเลข 7 ส่วน 0-9  ***
DELAY:  MOV   R5,#09H
```

```
DELAY2:   MOV   R4,#0FFH
DELAY1:   MOV   R3,#0FFH
          DJNZ  R3,$
          DJNZ  R4, DELAY1
          DJNZ  R5, DELAY2
          RET
TABLE:    DB    3FH, 06H, 5BH, 4FH, 66H
          DB    6DH, 7DH, 07H, 7FH, 6FH
          END
```

ตัวอย่างที่ 3 เขียนโปรแกรมแสดงผลที่แอลอีดี 7 ส่วนขนาด 2 หลักโดยกำหนดค่าคงที่เป็นเลข 79 เก็บไว้ในหน่วยความจำ หลังจากนั้นให้แสดงผลเป็นตัวเลขพร้อมกัน



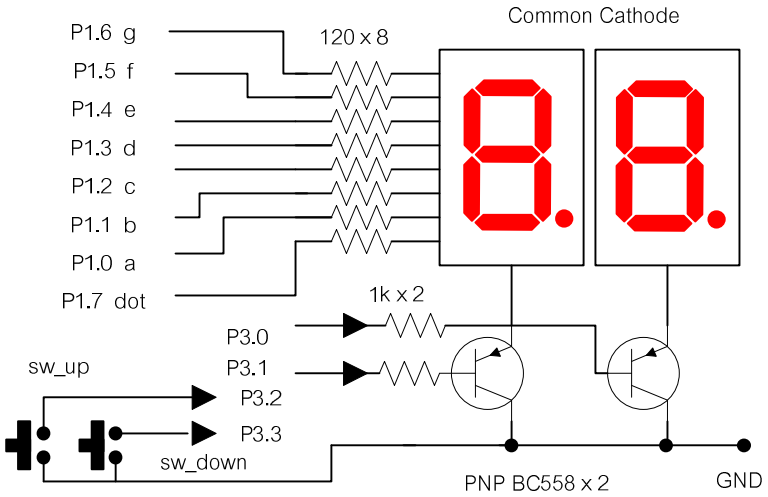
ภาพที่ 6.9 แสดงผังงานของโปรแกรมในตัวอย่างที่ 3

คำสั่ง DIV AB ใช้หารข้อมูลในรีจิสเตอร์ A ด้วยค่าข้อมูลในรีจิสเตอร์ B ผลลัพธ์จากการหารเก็บไว้ที่รีจิสเตอร์ A และเศษเก็บไว้ในรีจิสเตอร์ B ถ้ากำหนดค่าในรีจิสเตอร์ A = 79_{10} รีจิสเตอร์ B = 1010B หลังจากใช้คำสั่ง DIV AB ผลลัพธ์ข้อมูลที่รีจิสเตอร์ A = 07H และที่รีจิสเตอร์ B = 09H เมื่อนำไปเขียนโปรแกรม ต้องเพิ่มคำสั่งจากตัวอย่างที่ 2 โดยการหาค่าหลักหน่วย และหลักสิบ ที่ได้จากการหาร แล้วนำผลลัพธ์ไปเปิดตารางข้อมูล เพื่อให้แสดงผลที่แอลอีดี 7 ส่วน วิธีการดังกล่าว เขียนเป็นผังงานได้ดังภาพที่ 6.9

```
        ; ********************************************
        ; **      7 SEMENT 2 DIGIT Multiplexed Display        **
        ; ********************************************
     DIGIT1  BIT    P3.1          ; ให้บิต P3.1 ควบคุมที่ขาเบสของทรานซิสเตอร์หลักที่ 1
     DIGIT0  BIT    P3.0          ; ให้บิต P3.0 ควบคุมที่ขาเบสของทรานซิสเตอร์หลักที่ 0
COUNT        EQU    20H           ; กำหนดชื่อ COUNT อยู่ตำแหน่งหน่วยความจำแอดเดรส 20 H
DATA:        DS     2             ; จองเนื้อที่ในหน่วยความจำขนาด 2 ไบต์ชื่อ DATA, DATA+1
             ORG    0000H         ; เริ่มต้นโปรแกรมที่แอดเดรส 0000H
             MOV    P3,#0FFH      ; กำหนดให้ P3=11111111B ทรานซิสเตอร์หยุดทำงานทั้ง 2 ตัว
             MOV    COUNT, #79    ; กำหนดค่าคงที่เลข 79 ไว้ในหน่วยความจำ
LOOP:        ACALL DATA_DIS       ; เรียกโปรแกรมย่อยทำการหาร เพื่อแสดงผลเป็นเลข 2 หลัก
             ACALL DISPLAY        ; เรียกโปรแกรมย่อยแสดงผลทั้งสองหลักโดยสลับการทำงาน
             SJMP   LOOP          ; กระโดดกลับไปทำเริ่มต้นใหม่ที่ LOOP
DATA_DIS:    MOV    DPTR,#TABLE
             MOV    B,#10
             MOV    A,COUNT
             DIV    AB
             MOVC   A,@A+DPTR
             MOV    DATA,A        ; นำค่าที่ได้ Reg. A เก็บไว้ที่ หน่วยความจำ DATA ที่จองไว้
             MOV    A,B           ; นำค่าที่ รีจิสเตอร์ B ไปเก็บไว้ที่ รีจิสเตอร์ A
             MOVC   A,@A+DPTR     ;
             MOV    DATA+1,A      ; นำค่าที่ได้จาก Reg. A เก็บไว้หน่วยความจำตำแหน่ง DATA+1
             RET                  ; ออกจากโปรแกรมย่อย
DISPLAY:     MOV    A, DATA+1     ; นำค่าของตำแหน่ง DATA+1 เก็บไว้ที่ รีจิสเตอร์ A
             MOV    P1,A          ; นำค่าที่รีจิสเตอร์ A ออกพอร์ต P1
             CLR    DIGIT1        ; ให้ DIGIT1 หรือ P3.1 เป็น 0 ทรานซิสเตอร์ PNP นำกระแส
             ACALL DELAY_1        ; เรียกโปรแกรมย่อยหน่วงเวลา
```

```
        SETB    DIGIT1          ; ให้ DIGIT1 เป็น 1 ทรานซิสเตอร์ PNP หยุดนำกระแส
        MOV     A, DATA         ; นำค่าที่ตำแหน่ง DATA เก็บไว้ที่ รีจิสเตอร์ A
        MOV     P1,A            ; นำค่าที่รีจิสเตอร์ A ออกพอร์ต P1
        CLR     DIGIT0          ; ให้ DIGIT0 หรือ P3.0 เป็น 0 ทรานซิสเตอร์ PNP นำกระแส
        ACALL DELAY_1           ; เรียกโปรแกรมย่อยหน่วงเวลา
        SETB    DIGIT0          ; ให้ DIGIT0 เป็น 1 ทรานซิสเตอร์ PNP หยุดนำกระแส
        RET                     ; ออกจากโปรแกรมย่อย
    ;*** โปรแกรมย่อยการหน่วงเวลาเพื่อแสดงผลที่ตัวเลข 7 ส่วน 0-9   ***
DELAY_1:MOV     R1,#200         ; กำหนดให้ค่าที่ รีจิสเตอร์ R1 มีค่าคงที่ 200
        DJNZ    R1,$
        RET                     ; ออกจากโปรแกรมย่อยหน่วงเวลา
TABLE:  DB      3FH, 06H, 5BH, 4FH, 66H   ; ข้อมูลที่ต้องการแสดงผลโดยใช้วิธีการเปิดตาราง
        DB      6DH, 7DH, 07H, 7FH, 6FH
        END
```

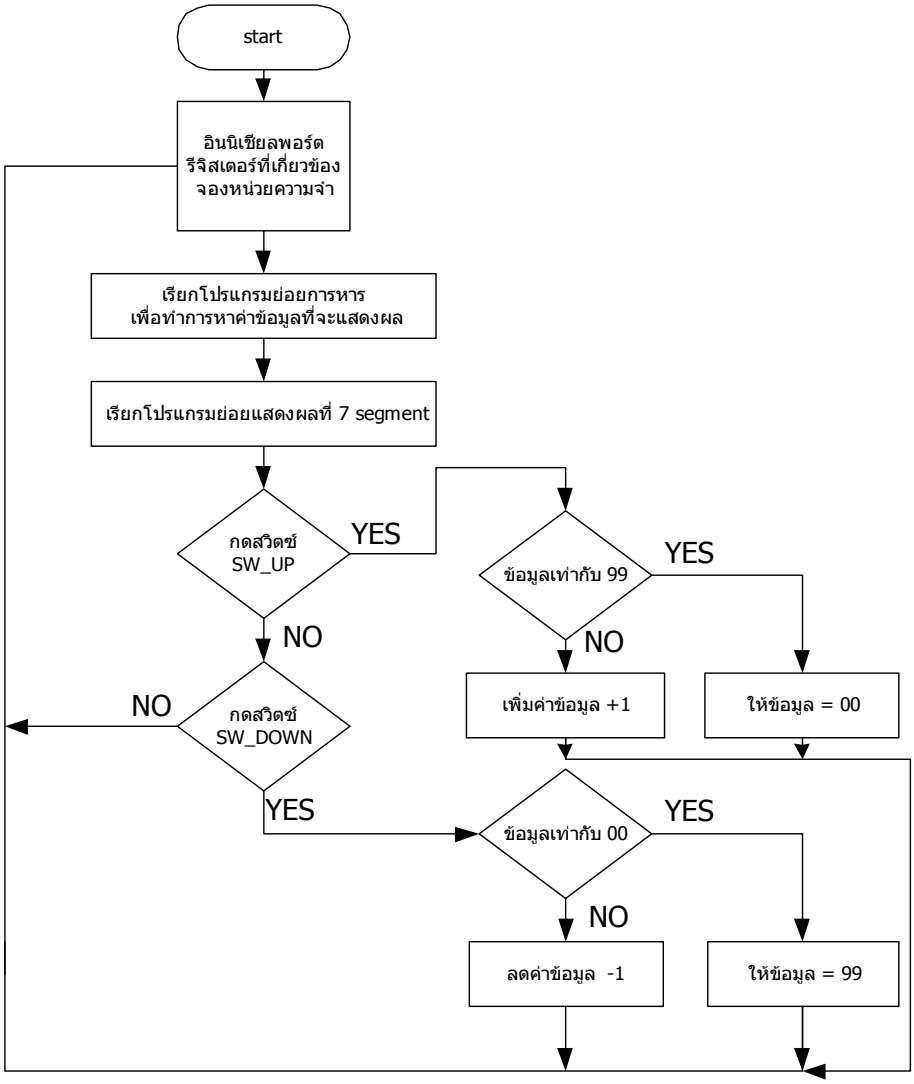
ตัวอย่างที่ 4 กำหนดเงื่อนไขให้รับข้อมูลอินพุต จากสวิตซ์แบบกดติดปล่อยที่พอร์ต P3.2 และ P3.3 ต่อวงจรแสดงดังภาพที่ 6.10 ถ้ามีการกดสวิตซ์พอร์ต P3.2 หนึ่งครั้งโปรแกรมทำคำสั่งในการเพิ่มค่าของข้อมูลที่แสดงผลครั้งละ1 ค่า และหากกดสวิตซ์พอร์ต P3.3 เป็นการลดค่าของข้อมูลที่แสดงผลครั้งละ 1 ค่า เขียนเป็นผังงานแสดงดังภาพที่ 6.11



ภาพที่ 6.10 วงจรการต่อแอลอีดี 7 ส่วน 2 หลัก

```
        ;***********************************************
        ; ** PROGRAM   COUNTS UP & COUNTS DOWN       **
        ; **      7 SEMENT 2 DIGIT      PORT 1 OUTPUT       **
        ;***********************************************
SW_UP       BIT     P3.2     ; กำหนดชื่อ SW_UP มีค่าเท่ากับ P3.2
SW_DOWN     BIT     P3.3     ; กำหนดชื่อ SW_DOWN มีค่าเท่ากับ P3.3
DIGIT0      BIT     P3.0     ; กำหนดชื่อ DIGIT0 มีค่าเท่ากับ P3.0
DIGIT1      BIT     P3.1     ; กำหนดชื่อ DIGIT1 มีค่าเท่ากับ P3.1
COUNT       EQU     20H      ; กำหนด COUNT ให้อยู่หน่วยความจำข้อมูล (RAM) แอดเดรส 20 H
DATA:       DS      2        ; จองเนื้อที่ในหน่วยความจำข้อมูล (RAM ภายใน) ขนาด 2 ไบต์
            ORG     0000H            ; เริ่มต้นโปรแกรมที่แอดเดรสแรก 0000H
START:      SETB    SW_UP            ; กำหนดให้เป็นพอร์ตอินพุตจะต้องเซตบิตให้เป็นลอจิก " 1 "
            SETB    SW_DOWN          ; กำหนดให้เป็นอินพุตต้องเซตให้เป็น ลอจิก "1 "
            MOV     COUNT,#00H       ; เคลียร์ค่าข้อมูลใน COUNT = 00000000b
            MOV     P1,#00H          ; เคลียร์ค่าข้อมูลในพอร์ต P1 = 00000000b
            MOV     R7,#00H          ; เคลียร์ค่าข้อมูลในรีจิสเตอร์ R7 = 00000000b
            SETB    DIGIT0           ; ให้ทรานซิสเตอร์หยุดนำกระแส Digit0 ดับ
            SETB    DIGIT1           ; ให้ทรานซิสเตอร์หยุดนำกระแส Digit1 ดับ
MAIN:       ACALL DATA_DIS           ; เรียกโปรแกรมย่อยหาค่าข้อมูล นำไปเก็บไว้ที่ DATA+1
            ACALL DISPLAY            ; เรียกโปรแกรมย่อยแสดงผลที่ แอลอีดี 7 ส่วน
            JNB     SW_UP,UP         ; ตรวจสอบ SW_UP ถ้ามีการกดคีย์ให้กระโดดไปทำที่เลเบล UP
            JNB     SW_DOWN, DOWN
            SJMP    MAIN             ; ถ้าไม่มีการกดคีย์ใดๆกระโดดกลับไปทำที่ MAIN ใหม่
        ; **      ทำการเพิ่มค่าข้อมูลที่ COUNT (Address 20 H )      **
UP:         ACALL DELAY_D            ; เรียกโปรแกรมย่อยหน่วงเวลา
            JB      SW_UP, MAIN      ; ตรวจสอบสถานะคีย์อีกครั้ง ถ้าไม่กดกลับไปที่เลเบล MAIN
            MOV     R7, COUNT        ; นำค่าข้อมูลจากแอดเดรส COUNT มาเก็บไว้ที่รีจิสเตอร์ R7
            CJNE    R7,#63H,INC_COUNT   ;ข้อมูล R7 ≠ 63H หรือ  99D ไปที่ INC_COUNT
            MOV     COUNT,#00H       ;ให้ข้อมูลใน COUNT = 00H
            JNB     SW_UP,$          ; ตรวจสอบการปล่อยคีย์ เช็คสถานะลอจิกที่สวิตซ์
            SJMP    MAIN             ; กระโดดกลับไปที่ MAIN
INC_COUNT:  INC     COUNT            ; เพิ่มค่าข้อมูลที่ COUNT = COUNT+1
```

```
        JNB     SW_UP,$         ; ตรวจสอบการปล่อยคีย์ถ้าไม่ปล่อยให้วนทำที่บรรทัดเดิม
        SJMP    MAIN            ; กระโดดกลับไปที่ MAIN
```



ภาพที่ 6.11 แสดงผังงานของโปรแกรมแอลอีดี 7 ส่วน 2 หลัก

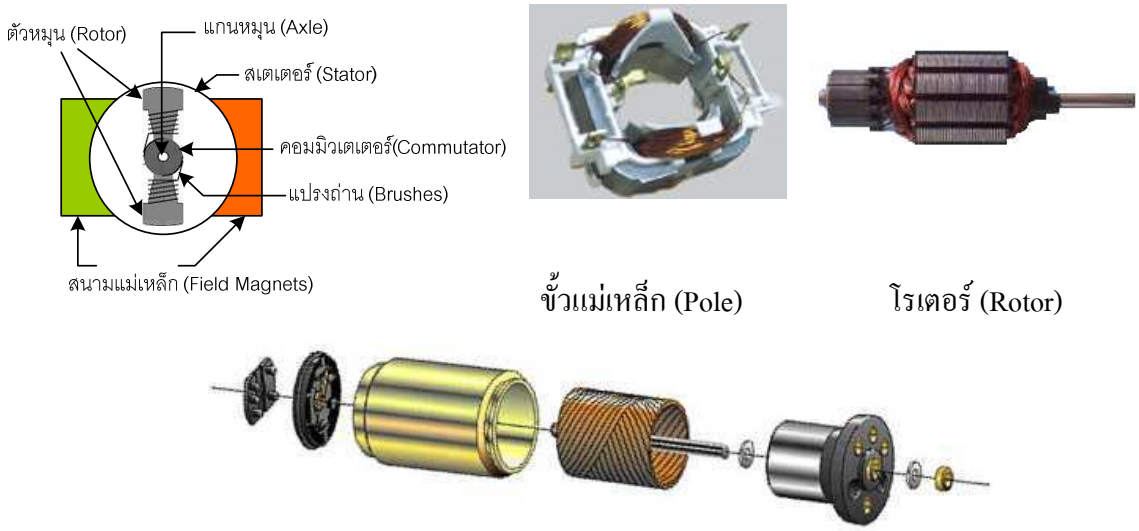
```
        ; **    ทำการลดค่าข้อมูลที่ COUNT (ADDR 20 H)        **
DOWN:   ACALL DELAY_D           ; เรียกโปรแกรมหน่วงเวลา
        JB      SW_DOWN, MAIN   ; ตรวจสอบการกดคีย์อีกครั้งว่ามีการกดจริงหรือไม่
        MOV     R7, COUNT       ; ข้อมูลจากแอดเดรส COUNT มาเก็บไว้ที่รีจิสเตอร์ R7
        CJNE    R7,#00,DEC_COUNT ; ถ้าค่าข้อมูลที่รีจิสเตอร์ R7 ≠ 00 ไป DEC_COUNT
        MOV     COUNT,#63H      ; ให้ COUNT = 99D ถ้าเป็นตามเงื่อนไขของคำสั่ง
        JNB     SW_DOWN, $      ; ตรวจสอบการปล่อยคีย์
        SJMP    MAIN            ; กระโดดกลับไปที่ MAIN
```

```
DEC_COUNT:  DEC    COUNT              ; เพิ่มค่าข้อมูลที่ COUNT = COUNT-1
            JNB    SW_DOWN,$          ; ตรวจสอบการปล่อยคีย์
            SJMP   MAIN               ; กระโดดกลับไปที่ MAIN
      ; **   โปรแกรมย่อยแปลงค่าข้อมูลที่ COUNT (ADDR 20 H )  **
DATA_DIS:   MOV    DPTR,#TABLE  ; กำหนดค่าฐานแอดเดรสที่ TABLE
            MOV    B,#10        ; ให้ข้อมูลในรีจีสเตอร์ B = 10 เพื่อจะนำไปหารกับ รีจิสเตอร์ A
            MOV    A, COUNT     ; นำค่าคงที่แอดเดรส COUNT มาเก็บไว้ที่ รีจิสเตอร์ A
            DIV    AB           ; นำค่าในรีจิสเตอร์ B ไปหารค่าในรีจิสเตอร์ A
            MOVC   A,@A+DPTR    ; นำค่าข้อมูลจาก Reg. A ไปรวมกับค่าที่เป็นฐานแอดเดรส
            MOV    DATA+1,A     ; นำค่าที่ได้จากรีจิสเตอร์ A ไปเก็บไว้ตำแหน่ง DATA ที่จองไว้
            MOV    A,B          ; นำค่าในรีจิสเตอร์ B ไปเก็บไว้ในรีจิสเตอร์ A
            MOVC   A,@A+DPTR    ; นำค่าข้อมูลจาก Reg. A ไปรวมกับค่าที่เป็นฐานแอดเดรส
            MOV    DATA,A       ; นำรีจิสเตอร์ A เก็บไว้ตำแหน่ง DATA+1
            RET
      ; **   โปรแกรมย่อยแสดงผลค่าข้อมูลที่   DATA, DATA+1          **
DISPLAY:    MOV    A, DATA+1    ; นำค่าข้อมูลตำแหน่ง DATA+1 เก็บไว้ที่ รีจิสเตอร์ A
            MOV    P1,A         ; นำค่าข้อมูลที่รีจิสเตอร์ A ออกพอร์ต P1
            CLR    DIGIT1       ; ให้ DIGIT1 เป็นลอจิก " 0 " ทรานซิสเตอร์ นำกระแส
            ACALL  DELAY_SEG    ; เรียกโปรแกรมย่อยหน่วงเวลา
            SETB   DIGIT1       ; ให้ DIGIT1 เป็น ลอจิก "1" ทรานซิสเตอร์ PNP หยุดนำกระแส
            MOV    A, DATA      ; นำข้อมูลตำแหน่ง DATA เก็บไว้ที่ รีจิสเตอร์ A
            MOV    P1,A         ; นำค่าข้อมูลที่รีจิสเตอร์ A ออกพอร์ต P1
            CLR    DIGIT0       ; ให้ DIGIT0 เป็นลอจิก "0" ทรานซิสเตอร์ PNP นำกระแส
            ACALL  DELAY_SEG    ; เรียกโปรแกรมย่อยหน่วงเวลา
            SETB   DIGIT0       ; ให้ DIGIT0 เป็นลอจิก "1" ทรานซิสเตอร์ PNP หยุดนำกระแส
            RET
      ;*** โปรแกรมย่อยการหน่วงเวลาเพื่อแสดงผลที่ตัวเลข 7 ส่วน 0-9   ***
DELAY_SEG:  MOV    R1,#200      ; กำหนดให้ค่าข้อมูลที่ รีจิสเตอร์ R1 มีค่าคงที่เท่ากับ 200D
            DJNZ   R1,$         ;
            RET                 ; ออกจากโปรแกรมย่อยหน่วงเวลา
      ;*** โปรแกรมย่อยการหน่วงเวลาเพื่อแก้ปัญหาจากสวิตซ์    ***
DELAY_D:    MOV    R2,#0FFH     ; กำหนดให้ค่าข้อมูลที่ รีจิสเตอร์ R2 มีค่าคงที่เท่ากับ FFH
```

```
        DJNZ    R2,$
        RET                     ; ออกจากโปรแกรมย่อยหน่วงเวลา
TABLE:  DB      3FH, 06H, 5BH, 4FH, 66H   ; ข้อมูลที่ต้องการแสดงผลโดยใช้วิธีการเปิดตาราง
        DB      6DH, 7DH, 07H, 7FH, 6FH
        END
```

3. การเชื่อมต่อกับมอเตอร์กระแสตรง

มอเตอร์กระแสตรง (DC MOTOR) มีการทำงานโดยผ่านกระแสให้กับขดลวดในสนามแม่เหล็กทำให้เกิดแรงแม่เหล็ก ส่วนของแรงนี้ขึ้นอยู่กับกระแส และกำลังของสนามแม่เหล็ก



ภาพที่ 6.12 แสดงโครงสร้างทั่วไปของมอเตอร์กระแสตรง

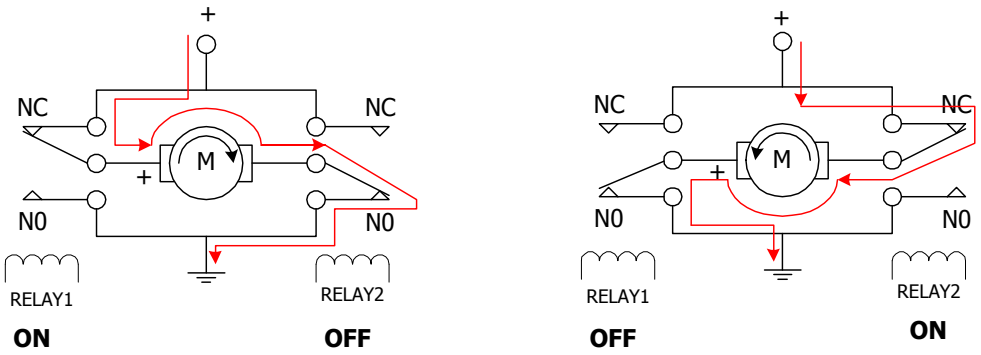
(แหล่งอ้างอิง http://www.solarbotics.net/starting/200111_dcmotor/200111_dcmotor2.ht)

ทางเดินของฟลั๊กซ์แม่เหล็ก และสนามแม่เหล็กเกิดจากแท่งแม่เหล็กเฟอร์ไรต์ 2 ชิ้น ขึ้นรูปเป็นแบบโค้งยึดติดกับตัวถังได้พอดี เพื่อให้เส้นแรงแม่เหล็กวิ่งเข้าสู่ใจกลางของมอเตอร์ได้ แสดงดังภาพที่ 6.12 ดังนั้นความเข้มของแม่เหล็กขึ้นอยู่กับขนาดความหนาของแม่เหล็กด้วย ซึ่งส่งผลให้ฟลั๊กซ์แม่เหล็ก วิ่งไปบนตัวถังโลหะ ทำให้กระแสไฟฟ้าในขดลวดที่พันกับทุ่นโรเตอร์ (Rotor) เกิดสนามแม่เหล็กไฟฟ้า และต้านกับสนาม แม่เหล็กถาวร จึงเกิดเป็นแรงบิดเพื่อหมุนทุ่นโรเตอร์ ให้ไปในทิศทางเดียวกันกับทิศทางของสนามแม่เหล็กที่มีแรงมากกว่า กระแสไหลผ่านไปยังทุ่นโรเตอร์ โดยผ่านแปรงถ่าน ซึ่งสัมผัสกับแหวนตัวนำในทุ่นโรเตอร์ และแหวนคอมมิวเตเตอร์ (Commutator) ซึ่งถูกแบ่งออกเป็น 3 เซกเมนต์เพื่อทำหน้าที่นำกระแสเข้าขดลวดนั้นเอง

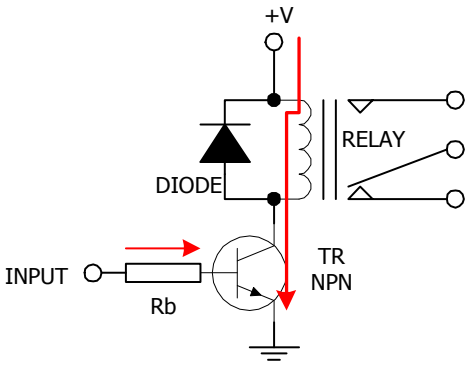
3.1 วงจรขับมอเตอร์ และกลับทิศทางของมอเตอร์กระแสตรง

การใช้ไอซี MCS-51 ควบคุมการหมุน และทิศทางของมอเตอร์กระแสตรง วงจรในส่วนทำหน้าที่ขับมอเตอร์ (Driver) และวงจรกลับทิศทางของมอเตอร์ ใช้รีเลย์ต่อวงจรเป็นสวิตช์ เพื่อกำหนดขั้วไฟฟ้ากระแสตรง หรือใช้อุปกรณ์สารกึ่งตัวนำเป็นวงจรขับกำลังเช่น ทรานซิสเตอร์ มอสเฟต ฯลฯ

การใช้รีเลย์ควบคุมทิศทางการหมุนของมอเตอร์ แสดงดังภาพที่ 6.13 โดยสลับการทำงานของรีเลย์ 2 ตัว ซึ่งทำหน้าที่กลับทิศทางของขั้วไฟฟ้าที่ป้อนให้กับมอเตอร์ ตัวอย่างเช่นให้รีเลย์ตัวที่ 1 ทำงาน (ON) และรีเลย์ตัวที่ 2 หยุดทำงาน (OFF) ทำให้มอเตอร์หมุนไปทางซ้าย และในทำนองเดียวกันถ้าหากรีเลย์ตัวที่ 1 หยุดทำงาน (OFF) และรีเลย์ตัวที่ 2 ทำงาน (ON) ทำให้มอเตอร์หมุนไปทางขวา

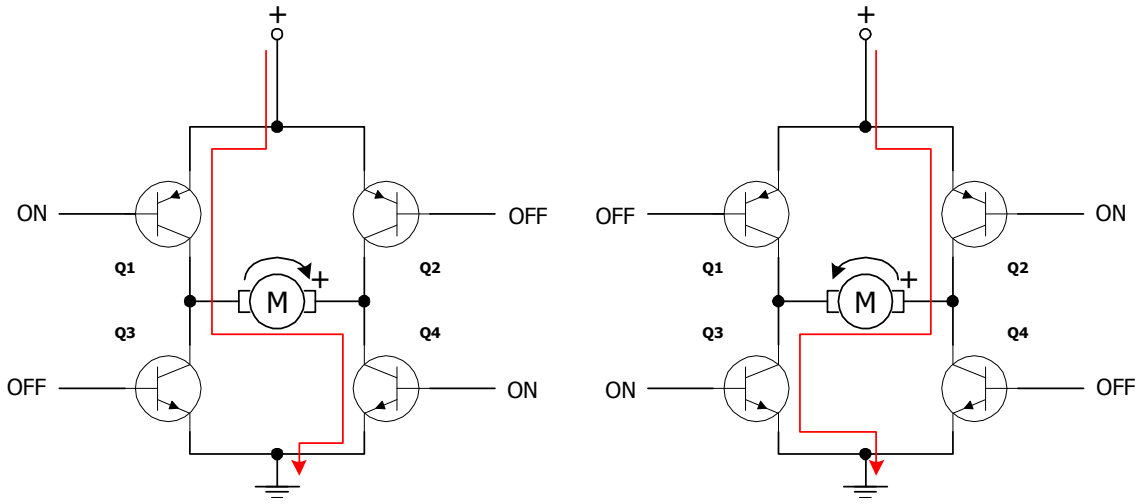


ภาพที่ 6.13 แสดงการกลับทิศทางของมอเตอร์กระแสตรงโดยใช้รีเลย์



ภาพที่ 6.14 แสดงการใช้ทรานซิสเตอร์เพื่อขับรีเลย์ให้ทำงาน

เอาต์พุตของไอซี MCS-51 ไม่สามารถจ่ายกระแสไฟที่ขดลวดของรีเลย์โดยตรงได้ ดังนั้นจึงต้องใช้ทรานซิสเตอร์ขยายกระแสให้เพียงพอให้กับขดลวดของรีเลย์ ส่วนไดโอดนำมาต่อไว้สำหรับป้องกันแรงดันย้อนกลับ เกิดจากการเหนี่ยวนำของสนามแม่เหล็กในขณะเกิดการยุบตัว ซึ่งอาจทำให้ทรานซิสเตอร์เสียหายได้ วงจรขับรีเลย์แสดงดังภาพที่ 6.14

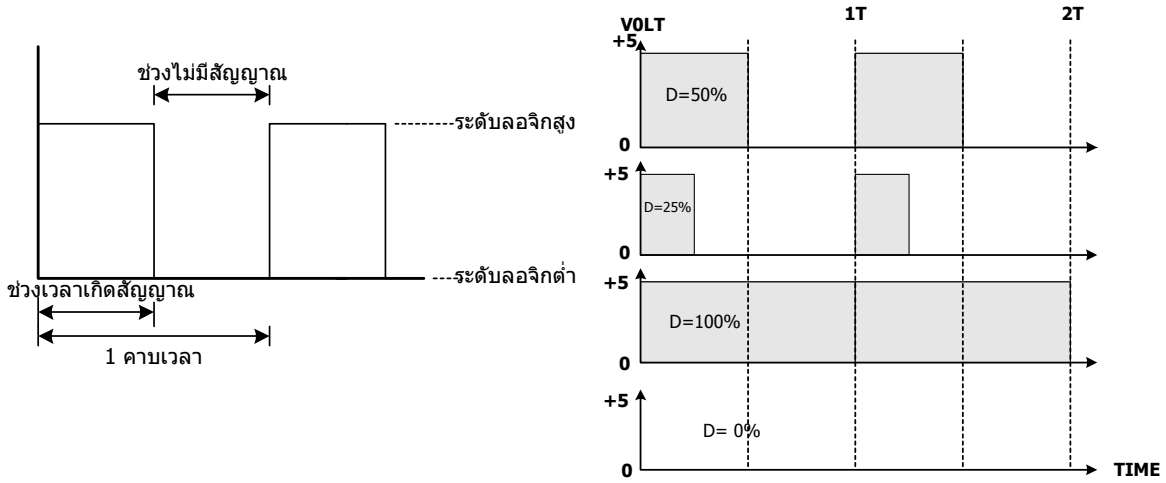


ภาพที่ 6.15 แสดงการใช้ทรานซิสเตอร์เป็นวงจรขับ และกำหนดทิศทางของมอเตอร์กระแสตรง

วงจรลิเนียร์บริดจ์แอมป์ ประกอบด้วยทรานซิสเตอร์กำลัง 4 ตัวทำหน้าที่ขับ และควบคุมทิศทางการหมุนของมอเตอร์ หากกำหนดให้ทรานซิสเตอร์ Q1 และ Q4 อยู่ในสภาวะทำงาน กระแสไฟฟ้าไหลผ่านทรานซิสเตอร์จากซ้ายไปขวา โดยผ่านมอเตอร์กระแสตรงทำให้มอเตอร์หมุนไปทางขวา ในทำนองเดียวกันหากทำให้ทรานซิสเตอร์ Q2 และ Q3 อยู่ในสภาวะทำงาน (Active) กระแสไฟฟ้าไหลจากทางขวาไปทางซ้ายซึ่งส่งผลให้มอเตอร์กลับทิศทางการหมุนจากทางขวาไปทางซ้าย แสดงดังภาพที่ 6.15

3.2 การควบคุมความเร็วของมอเตอร์กระแสตรง

การควบคุมความเร็วของมอเตอร์กระแสตรงมีหลายวิธี เช่นการควบคุมโดยใช้ตัวต้านทานปรับค่าต่ออนุกรมกับมอเตอร์ การควบคุมโดยการเปลี่ยนค่าระดับแรงดันที่ป้อนให้กับมอเตอร์ โดยสามารถควบคุมความเร็วของมอเตอร์ให้คงที่ได้ แต่ที่ความเร็วต่ำแรงบิดต่ำไปด้วย

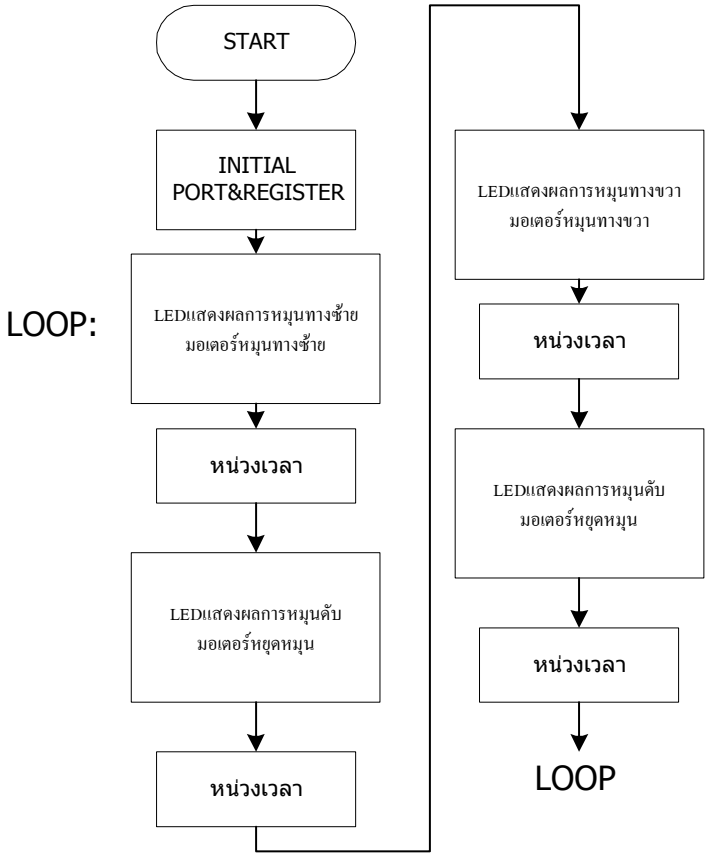


ภาพที่ 6.16 แสดงความกว้างของพัลส์ขนาดต่างๆ และค่าดิวตี้ไซเคิล ของช่วงพัลส์ที่มีความถี่คงที่

ดังนั้นจึงมีวิธีการควบคุมโดยการจ่ายกระแสไฟให้กับมอเตอร์เป็นช่วงๆ โดยอาศัยกระแสไฟที่ป้อนให้กับมอเตอร์ เป็นค่าเฉลี่ยที่เกิดขึ้นในแต่ละช่วงเรียกว่าวิธีการมอดูเลชั่นทางความกว้างของพัลส์ PWM (Pulse Width Modulation) แสดงดังภาพที่ 6.16

3.3 การเขียนโปรแกรมควบคุมทิศทางมอเตอร์กระแสตรง

ตัวอย่างที่ 5 เขียนโปรแกรมควบคุมทิศทางมอเตอร์ให้หมุนไปทางซ้าย และทางขวาโดยสลับการหมุน และในขณะสลับทิศทางการหมุน ต้องหยุดป้อนกระแสไฟฟ้าผ่านมอเตอร์ในระยะเวลาหนึ่ง หลังจากนั้นจึงกลับทิศทางการหมุนของมอเตอร์ได้ เพื่อลดแรงเฉื่อยของมอเตอร์ในทิศทางที่กลับกัน โดยไม่ให้กลับในทันทีทันใด ซึ่งอาจส่งผลต่อระบบกลไกที่ต่อร่วมอยู่ภายนอก ผังงานของโปรแกรมแสดงดังภาพที่ 6.17



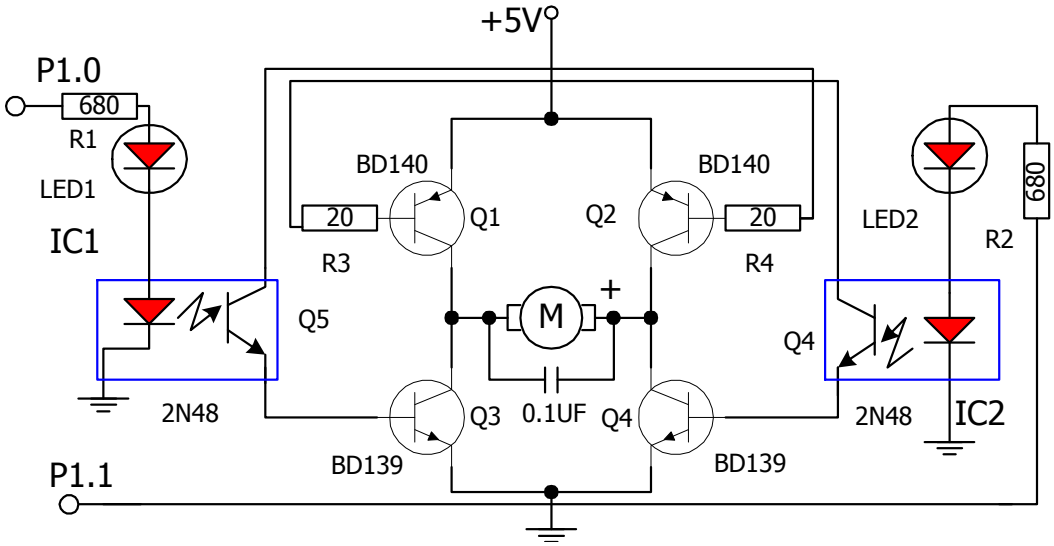
ภาพที่ 6.17 แสดงผังงานของโปรแกรมควบคุมทิศทางมอเตอร์กระแสตรง

```
;****************************************
;*** PROGRAM DC MOTOR DRIVER         ***
;****************************************
MOTOR_LEFT      BIT     P1.0    ; กำหนดค่าของพอร์ตโดยให้บิตที่ P1.0  ชื่อ MOTOR_LEFT
OTOR_RIGHT      BIT     P1.1    ; กำหนดค่าของพอร์ตโดยให้บิตที่ P1.1  ชื่อ MOTOR_RIGHT
```

```
        ORG    0000H
LOOP:  MOV    P1,#00H          ; เคลียร์ค่าข้อมูลในพอร์ต P1ให้มีค่า = 00H ทุกบิต
        ;*** เลือกทิศทางการหมุนไปทางซ้าย ****
LEFT:  SETB   LED_LEFT         ; กำหนดให้ LED_LEFT แสดงผลเป็นการเลือกทิศทางหมุนไปทางซ้าย
        SETB   MOTOR_LEFT ; กำหนดให้บิตที่ P1.0 เป็นลอจิก “1”ส่งผลให้มอเตอร์หมุนไปทางซ้าย
        ACALL DELAY            ; เรียกโปรแกรมย่อยหน่วงเวลาครั้งที่ 1
        ACALL DELAY            ; เรียกโปรแกรมย่อยหน่วงเวลาครั้งที่ 2
        CLR    MOTOR_LEFT ; กำหนดให้บิตที่ P1.0 เป็นลอจิก  “ 0 ”ส่งผลให้มอเตอร์หยุดหมุน
        ACALL DELAY            ; ทำการหน่วงเวลาให้มอเตอร์ลดแรงเฉื่อยลงก่อนกลับทิศทางการหมุน
        ;****  เป็นการเลือกทิศทางหมุนไปทางขวา ***
RIGHT:SETB   LED_RIGHT              ; ให้ LED_LEFT แสดงผลเป็นการเลือกทิศทางหมุนไปทางขวา
        SETB   MOTOR_RIGHT          ; ให้บิตที่ P1.1 เป็นลอจิก “1”ส่งผลให้มอเตอร์หมุนไปทางขวา
        ACALL DELAY                 ; เรียกโปรแกรมย่อยหน่วงเวลาครั้งที่ 1
        ACALL DELAY                 ; เรียกโปรแกรมย่อยหน่วงเวลาครั้งที่ 2
        CLR    MOTOR_RIGHT          ; ให้บิตที่ P1.1 เป็นลอจิก “0”ส่งผลให้มอเตอร์หยุดหมุน
        ACALL DELAY                 ; ให้มอเตอร์ลดแรงเฉื่อยลงก่อนกลับทิศทางการหมุน
        SJMP   LOOP                 ; วนกลับไปทำที่เลเบล LOOP ใหม่
        ; โปรแกรมย่อยหน่วงเวลาเพื่อลดแรงเฉื่อย
        DELAY:      MOV    R0,#0FH
        DELAY1:     MOV    R1,#0FFH
        DELAY2:     MOV    R2,#0FFH
                    DJNZ   R2,$
                    DJNZ   R1, DELAY2
                    DJNZ   R0, DELAY1
                    RET
                    END
```

ข้อควรระวังในการต่อวงจรขับมอเตอร์ ในกรณีใช้อุปกรณ์สารกึ่งตัวนำถ้าหากมีสถานะลอจิกเป็น “1” ทั้งคู่ อาจส่งผลให้อุปกรณ์ในวงจรเสียหายได้

3.4 วงจรควบคุมมอเตอร์กระแสตรงโดยใช้ทรานซิสเตอร์



ภาพที่ 6.18 แสดงวงจรควบคุมมอเตอร์กระแสตรงด้วยทรานซิสเตอร์

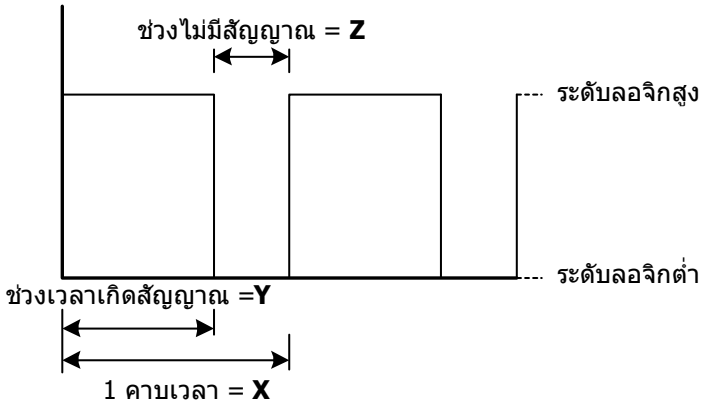
จากวงจรภาพที่ 6.18 ประกอบด้วยทรานซิสเตอร์จำนวน 6 ตัว ทำหน้าที่สลับกันทำงานครั้งละ 3 ตัว หากป้อนสถานะลอจิก “0” ที่พอร์ต P1.1 ให้มีสถานะลอจิก “1” ที่พอร์ต P1.0 จะทำให้แอลอีดีภายใน IC1 สว่างส่งผลให้ Q5 ที่อยู่ในไอซี 1 นำกระแส ทำให้ Q2 และQ3นำกระแสไปด้วย (Q2 เป็นทรานซิสเตอร์แบบ PNP โดยขา B ของ Q2 ได้ต่อกับขา C ของ Q5 ซึ่งให้สัญญาณตรงข้ามกับขา B จึงทำให้ Q2 นำกระแสได้) ดังนั้นกระแสจะไหลจากขา E ของ Q2 ผ่านไปยังมอเตอร์ ผ่านขา Cของ Q3 และผ่านไปยังขา E ของ Q3 ในที่สุดส่งผลให้มอเตอร์มีทิศทางการหมุนไปทางซ้าย ในทำนองเดียวกันหากกำหนดให้ลอจิกที่ P1.0 เป็นสถานะลอจิก “0” และให้มีสถานะลอจิก “1”ที่ P1.1 ทำให้แอลอีดีภายใน IC2 สว่าง ส่งผลให้ Q6 ที่อยู่ในไอซี 2 นำกระแส ทำให้ Q1และ Q4 นำกระแสไปด้วย ดังนั้นมอเตอร์จึงมีทิศทางการหมุนไปทางขวา ทีควรระวังในการใช้งาน

3.5 การมอดูเลชั่นทางความกว้างของพัลส์ (Pulse Width Modulation)

การมอดูเลชั่นทางความกว้างของพัลส์เป็นการปรับเปลี่ยนที่สัดส่วน และความกว้างของสัญญาณพัลส์ โดยความถี่ของสัญญาณพัลส์ไม่มีการเปลี่ยนแปลง หรือเป็นการเปลี่ยนแปลงที่ค่าของดิวตี้ไซเกิล (Duty Cycle) ซึ่งค่าของดิวตี้ไซเคิล คือช่วงความกว้างของพัลส์ที่มีสถานะลอจิกสูง โดยคิดสัดส่วนเป็นเปอร์เซ็นต์จากความกว้างของพัลส์ทั้งหมด ตัวอย่างเช่น ถ้าหากค่าดิวตี้ไซเคิลมีค่าเท่ากับ 50% หมายถึง ใน 1 รูป สัญญาณพัลส์มีช่วงของสัญญาณสถานะลอจิกสูงอยู่ครึ่งหนึ่ง และสถานะลอจิกต่ำอยู่อีกครึ่งหนึ่ง และในทำนองเดียวกันถ้าหากค่าดิวตี้ไซเคิลมีค่ามาก ค่าความกว้างของพัลส์ที่เป็นสถานะลอจิกสูง มีความ

กว้างมากขึ้น หากค่าดิวตี้ไซเคิลมีค่าเท่ากับ 100% หมายความว่าไม่มีสถานะลอจิกต่ำ ค่าดิวตี้ไซเคิลหาได้จากค่าความสัมพันธ์ดังนี้

ค่าดิวตี้ไซเคิล = (ช่วงของสัญญาณพัลส์ ÷ คาบเวลาทั้งหมดของสัญญาณ) × 100%

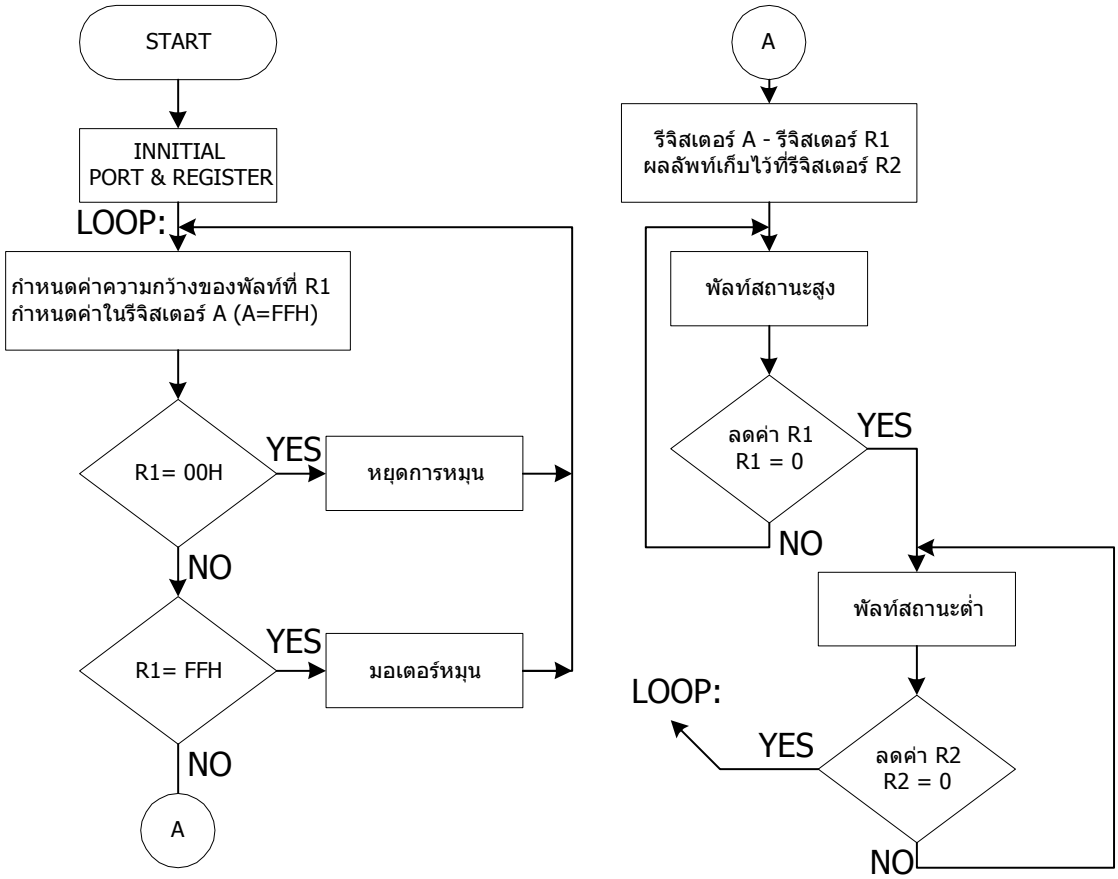


ภาพที่ 6.19 แสดงความกว้างของพัลส์ในช่วงต่างๆโดยการสมมุติค่าตัวแปรแทน

การเขียนโปรแกรมควบคุมความเร็วของมอเตอร์กระแสตรง โดยวิธีมอดูเลชั่นทางความกว้างของพัลส์ พิจารณาในภาพที่ 6.19 แสดงความกว้างของพัลส์ในช่วงต่างๆ โดยกำหนดค่าความกว้างของพัลส์ให้เป็นค่าคงที่ และเก็บไว้เป็นข้อมูลในรีจิสเตอร์ หรือในหน่วยความจำตำแหน่งหนึ่งไว้ ช่วงของความกว้างทั้งหมดใน 1 คาบเวลา (Duty Cycle =100%) กำหนดให้เท่ากับ FFH หรือทำการแบ่งออกเป็น 256 ส่วน เพื่อสะดวกในการกำหนดค่าของรีจิสเตอร์ขนาด 8 บิต และแทนค่าทั้งหมดด้วยตัวแปร **X** ในช่วง (FF) ของเวลาที่เกิดสัญญาณเป็นลอจิกสูง หรือการกำหนดค่าดิวตี้ไซเคิลในรีจิสเตอร์ใดๆก่อน 1 ค่า ดังนั้น เพื่อกำหนดค่าความกว้างของพัลส์ ต้องแทนด้วยตัวแปร **Y** หากนำเอาช่วงความกว้างทั้งหมดใน 1คาบเวลาลบออกจากค่าความกว้างที่กำหนดไว้ จึงมีค่าเท่ากับ X - Y = Z ซึ่งผลลบให้เป็นตัวแปร **Z** คือค่าความกว้างของพัลส์ที่ไม่มีสัญญาณหรือสถานะลอจิกต่ำนั้นเอง

ตัวอย่างที่ 6 เขียนโปรแกรมจึงกำหนดค่าช่วงความกว้างทั้งหมดใน 1 คาบเวลาไว้ที่รีจิสเตอร์ A (A= FFH) และค่าความกว้างของพัลส์ในระดับลอจิกสูงเป็นข้อมูลในรีจิสเตอร์ R1 (R1= ระบุค่าได้เองตามต้องการ 01H ถึง FEH) และนำค่าข้อมูลในรีจิสเตอร์ A ลบออกด้วยค่าข้อมูลในรีจิสเตอร์ R1 ผลลัพธ์ที่ได้เป็นค่าของความกว้างของช่วงที่ไม่มีสัญญาณพัลส์หรือระดับลอจิกต่ำ แต่ก่อนที่นำค่าข้อมูลใน R1 ไปลบกับค่าข้อมูลในรีจิสเตอร์ A ต้องทำการตรวจสอบข้อมูลในรีจิสเตอร์ R1 เสียก่อน หากมีข้อมูลเป็นค่า FFH ซึ่งเท่ากับค่าดิวตี้ไซเคิล 100 % ให้มอเตอร์หมุนได้ความเร็วสูงสุดโดยไม่ต้องกำหนดค่าใดๆ และในทำนองเดียวกันหากข้อมูลมีค่าเป็น 00H ซึ่งเท่ากับค่าดิวตี้ไซเคิล 0 % ให้มอเตอร์หยุดหมุน ตรวจสอบข้อมูลในรีจิสเตอร์ R1 หากมีข้อมูลเป็นค่า FFH เท่ากับค่าดิวตี้ไซเคิล 100 % ทำให้มอเตอร์หมุนได้ความเร็วสูงสุด

โดยไม่ต้องกำหนดค่าใดๆ และในทำนองเดียวกันหากข้อมูลมีค่าเป็น 00H ซึ่งเท่ากับค่าดิวตี้ไซเคิล 0 % ทำให้มอเตอร์หยุดหมุน ขันตอนการเขียนผังงานแสดงดังภาพที่ 6.20

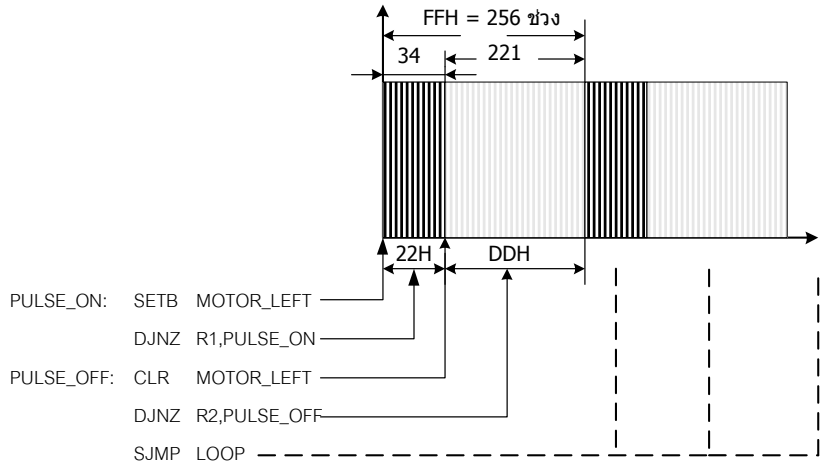


ภาพที่ 6.20 แสดงผังงานควบคุมความเร็วมอเตอร์แบบมอดูเลชั่นทางความกว้างของพัลส์

```
;*********************************************
;***        PWM   DC MOTOR CONTROL             ***
;*********************************************
MOTOR_LEFT      BIT     P1.0    ; กำหนดค่าของพอร์ตให้บิตที่ P1.0 ชื่อ MOTOR_LEFT
MOTOR_RIGHT     BIT     P1.1    ; กำหนดค่าของพอร์ตให้บิตที่ P1.1 ชื่อ MOTOR_RIGHT
                ORG     0000H           ; เริ่มต้นโปรแกรมที่แอดเดส 0000H
LOOP:           MOV     P1,#00H         ; เคลียร์ค่าในพอร์ต P1ให้มีค่า = 00H ทุกบิต
                MOV     R1, #22H        ; กำหนดค่าความกว้างของพัลท์สถานะลอจิกสูง (DUTY 50%
                                        ; R1=80H VA=2.45V: LOW SPEED R1 = 22H VA = 0.6 V)
                MOV     A,#0FFH         ; กำหนดค่าช่วงความกว้างของพัลท์ทั้งหมดใน 1 คาบเวลา
```

```
            CJNE   R1,#00H,PULSE; นำค่าที่กำหนดจากรีจิสเตอร์ R1 ตรวจสอบค่าต่ำสุด
            SJMP   STOP         ; หากเท่ากับ 00H ให้ไปหยุดมอเตอร์ที่เลเบล STOP
            CJNE   R1, #0FFH, PULSE
            SJMP   START        ; หากเท่ากับ FFH ให้ไปหมุนมอเตอร์ที่เลเบล START
PULSE:      CLR    C            ; คำสั่ง SUBB ใช้ Carry Flag ลบกับรีจิสเตอร์ A
            SUBB   A, R1        ; นำค่าในรีจิสเตอร์ R1 กับค่าแฟลก C ลบค่าในรีจิสเตอร์ A
            MOV    R2, A        ; นำผลลัพธ์ที่ได้จากรีจิสเตอร์ A เก็บไว้ที่รีจิสเตอร์ R2
PULSE_ON:   SETB   MOTOR_LEFT
            DJNZ   R1, PULSE_ON ; ช่วงเวลาในการหมุนค่ารีจิสเตอร์ R1 ลดค่าลงทีละ 1 ค่า
PULSE_OFF:  CLR    MOTOR_LEFT
            DJNZ   R2, PULSE_OFF; ช่วงเวลาหยุดหมุนกำหนดรีจิสเตอร์ R2 ลดค่าทีละ 1 ค่า
            SJMP   LOOP         ; กลับไปทำที่เลเบล LOOP
START:      SETB   MOTOR_LEFT ; มอเตอร์หมุนรอบสูงสุด
            SJMP   LOOP         ; กลับไปทำที่เลเบล LOOP
STOP:       CLR    MOTOR_LEFT ; มอเตอร์หยุดหมุน
            SJMP   LOOP         ; กลับไปทำที่เลเบล LOOP
            END
```

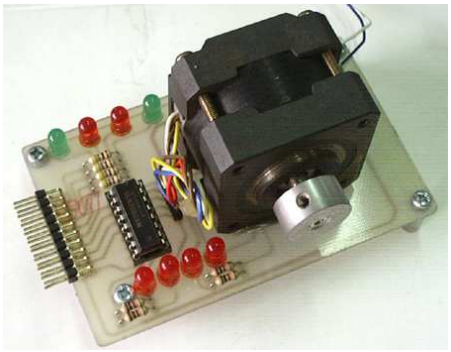
วิธีการเขียนโปรแกรมกำหนดค่าช่วงความกว้างทั้งหมดใน 1 คาบเวลา สรุปคำสั่งเป็นรูปสัญญาณแสดงดังภาพที่ 6.21



ภาพที่ 6.21 แสดงคำสั่งในการกำหนดช่วงเวลาในการเกิดลอจิกสถานะสูง และต่ำ

4. การเชื่อมต่อกับสเต็ปเปอร์มอเตอร์

สเต็ปเปอร์มอเตอร์ (Stepper Motor) เป็นอุปกรณ์ที่สามารถใช้ไอซี MCS-51 ควบคุมได้สะดวก เหมาะสำหรับใช้ในงานควบคุมการหมุนโดยต้องการตำแหน่ง และทิศทางที่แน่นอนแสดงดังภาพที่ 6.22 การทำงานของสเต็ปเปอร์มอเตอร์ขับเคลื่อนทีละขั้นๆ ละ (Step) 0.9, 1.8, 5, 7.5, 15 หรือ 50 องศา ขึ้นอยู่กับคุณสมบัติของ สเต็ปเปอร์ตัวนั้นๆ สเต็ปเปอร์มอเตอร์แตกต่างจากมอเตอร์กระแสตรงทั่วไป (DC Motor) โดยการทำงานของมอเตอร์กระแสตรงหมุนไปแบบต่อเนื่อง ไม่สามารถหมุนเป็นแบบสเต็ป ๆ ได้ดังนั้นการนำไปกำหนดตำแหน่งจึงควบคุมได้ยากกว่า การความคุมตำแหน่งของสเต็ปเปอร์มอเตอร์ส่วนใหญ่ใช้ระบบดิจิตอล เช่น เครื่องพรินเตอร์ พล็อตเตอร์ (X-Y Plotter) ดิสก์ไดร์ฟ (Disk Drive) ฯลฯ ข้อดีของสเต็ปเปอร์มอเตอร์เมื่อเปรียบกับมอเตอร์กระแสตรง สเต็ปเปอร์มอเตอร์จะควบคุมได้โดยไม่ต้องอาศัยตัวตรวจจับการหมุน ไม่ต้องใช้แปรงถ่านจึงไม่มีปัญหาด้านการสึกหรอ และการสปาร์คที่ทำให้เกิดสัญญาณรบกวน



ภาพที่ 6.22 สเต็ปเปอร์มอเตอร์

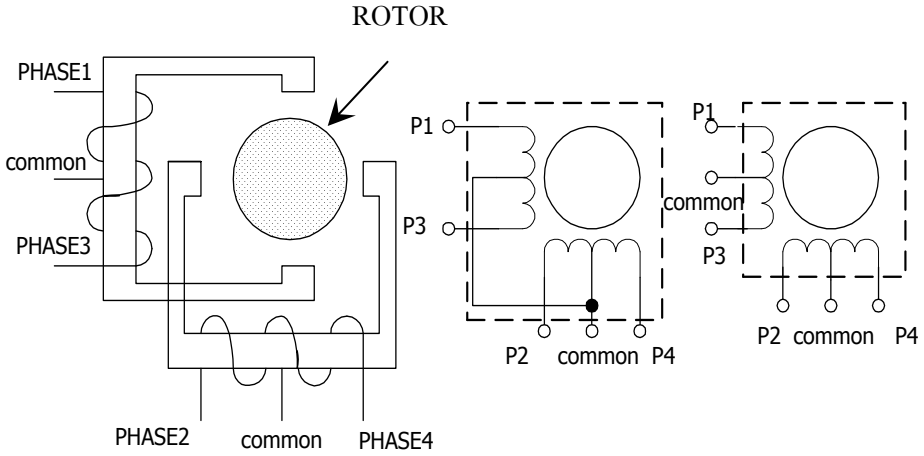
4.1 ชนิดของสเต็ปเปอร์มอเตอร์

สเต็ปเปอร์มอเตอร์แบ่งตามลักษณะสายที่ใช้ต่อกับวงจรขับ โดยแบ่งออกได้ 2 แบบคือ

4.1.1 แบบไบโพลา (Bipolar) ขดลวดที่สเตเตอร์แต่ละชุดไม่มีจุดร่วม การต่อเข้ากับวงจรขับใช้ปลายทั้งสองด้านของขดลวดแต่ละชุด การทำให้เกิดขั้วแม่เหล็กที่สเตเตอร์ โดยการจ่ายกระแสไฟ จากปลายด้านหนึ่งไปยังปลายอีกด้านหนึ่งของขดลวด และการเปลี่ยนขั้วแม่เหล็กที่สเตเตอร์ชุดเดียวกันนี้ทำโดยสลับทิศทางการไหลของกระแสไฟฟ้า ดังนั้นวงจรขับที่ใช้จึงจำเป็นต้องสามารถกลับทิศทางการไหลของกระแสได้ กรณีเป็นมอเตอร์ 2 เฟสมีสายที่ใช้ต่อขับกับวงจร 4 เส้น

4.1.2 แบบยูนิโพลา (Uni-polar) ขดลวดที่สเตเตอร์แต่ละชุดมีจุดร่วม การต่อเข้ากับวงจรขับใช้ปลายของขดลวดแต่ละด้านต่อเข้ากับวงจรขับ และใช้จุดร่วมต่อเข้ากับขั้วบวกของแหล่งจ่ายไฟ การทำให้เกิดขั้วแม่เหล็กที่สเตเตอร์ทำได้โดยการจ่ายกระแสไฟให้ไหลจากจุดร่วมลงกราวด์มาครบวงจรที่ปลายด้านหนึ่งของขดลวด การเปลี่ยนขั้วแม่เหล็กที่สเตเตอร์ชุดเดียวกันนี้ได้ โดยเปลี่ยนการจ่ายกระแสไฟฟ้าจากขด

หนึ่งไปยังอีกขดหนึ่งของขดลวดที่พันอยู่บนสเตเตอร์ชุดเดียวกัน ดังนั้นวงจรขับจึงเป็นวงจรสวิทช์เพื่อทำให้จ่ายกระแสไฟที่ผ่านขดลวดครบวงจรเท่านั้น กรณีเป็นมอเตอร์ 2 เฟสจึงมีสายที่ใช้ต่อขับกับวงจร 5 หรือ 6 เส้น โครงสร้างของสเต็ปเปอร์มอเตอร์แบบนี้แสดงดังภาพที่ 6.23 มีส่วนประกอบที่สำคัญ 2 ส่วน คือ ส่วนที่ทำการหมุน (Rotor) เป็นแม่เหล็กถาวรหรืออื่นๆ และส่วนที่อยู่กับที่ (Stator) เป็นขดลวดที่พันไว้จำนวนหลายๆขด



ภาพที่ 6.23 สเต็ปเปอร์มอเตอร์ 4 เฟส แบบยูนิโพล่าร์

4.2 วิธีการขับสเต็ปเปอร์มอเตอร์ให้หมุนโดยการกระตุ้นเฟส

ในการควบคุมสเต็ปเปอร์มอเตอร์เพื่อให้ทำการหมุน มีวิธีการควบคุมกระแสไฟที่จ่ายให้กับขดลวด สเตเตอร์ในแต่ละเฟสของสเต็ปเปอร์มอเตอร์ อย่างเป็นลำดับที่แน่นอน โดยถ้าหากต้องการให้กระแสไหลในเฟสใดๆ ทำให้สถานะของเฟสนั้นๆเป็นสถานะลอจิก “1” และในการกระตุ้นเฟสของสเต็ปเปอร์ มีอยู่ 2 แบบ โดยแบบแรกเป็นการกระตุ้นเฟส แบบฟูลสเต็ปมอเตอร์ (Full Step Motor) สามารถแบ่งออกเป็น 2 วิธีดังนี้

4.2.1 การกระตุ้นเฟสแบบฟูลสเต็ป 1 เฟส (Single-Phase Driver) หรือแบบเวฟ เป็นการป้อนกระแสไฟให้กับขดลวด ของสเต็ปเปอร์มอเตอร์ทีละขด แสดงดังภาพที่ 6.24 ก โดยป้อนกระแสเรียงตามลำดับกันไป ดังนั้นกระแส ที่ไหลในขดลวดทำการไหลในทิศทางเดียวกันทุกขด ลักษณะเช่นนี้จึงทำให้แรงขับของ สเต็ปเปอร์มอเตอร์มีน้อย

4.2.2 การกระตุ้นเฟสแบบฟูลสเต็ป 2 เฟส (Two-Phase Driver) แสดงดังภาพที่ 6.24 ข เป็นการป้อนกระแสให้กับขดลวด 2 ขด ของสเต็ปเปอร์มอเตอร์พร้อมๆกันไป และกระตุ้นเรียงถัดกันไป เช่นเดียวกับแบบหนึ่งเฟส ดังนั้นการกระตุ้นแบบนี้จึงต้องใช้กำลังไฟมากขึ้น และทำให้มีแรงบิดของมอเตอร์มากกว่าการกระตุ้นแบบ 1 เฟส

แบบที่สองการกระตุ้นเฟส แบบฮาลฟ์สเต็ป (Half Step Motor) หรือ One-Two Phase Driver คือการกระตุ้นเฟสแบบฟลูสเต็ป 1 เฟส และ 2 เฟส เรียงลำดับกันไป แสดงดังภาพที่ 6.24 ค แรงบิดที่ได้จากการกระตุ้นเฟสแบบนี้มีเพิ่มมากขึ้น เพราะช่วงของสเต็ปมีระยะสั้นลง และต้องมีการกระตุ้นที่เฟสถึง 2 ครั้ง จึงได้ระยะของสเต็ปเท่ากับการกระตุ้นเพียงครั้งเดียวของแบบฟูลสเต็ป 2 แบบแรก ความละเอียดการหมุนตำแหน่งองศาต่อสเต็ปเป็นสองเท่าของแบบแรก ความถูกต้องของตำแหน่งที่กำหนดจึงมีมากขึ้น

สเต็ปที่	เฟสที่1	เฟสที่2	เฟสที่3	เฟสที่4
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1

ก. แบบฟูลสเต็ป 1 เฟส

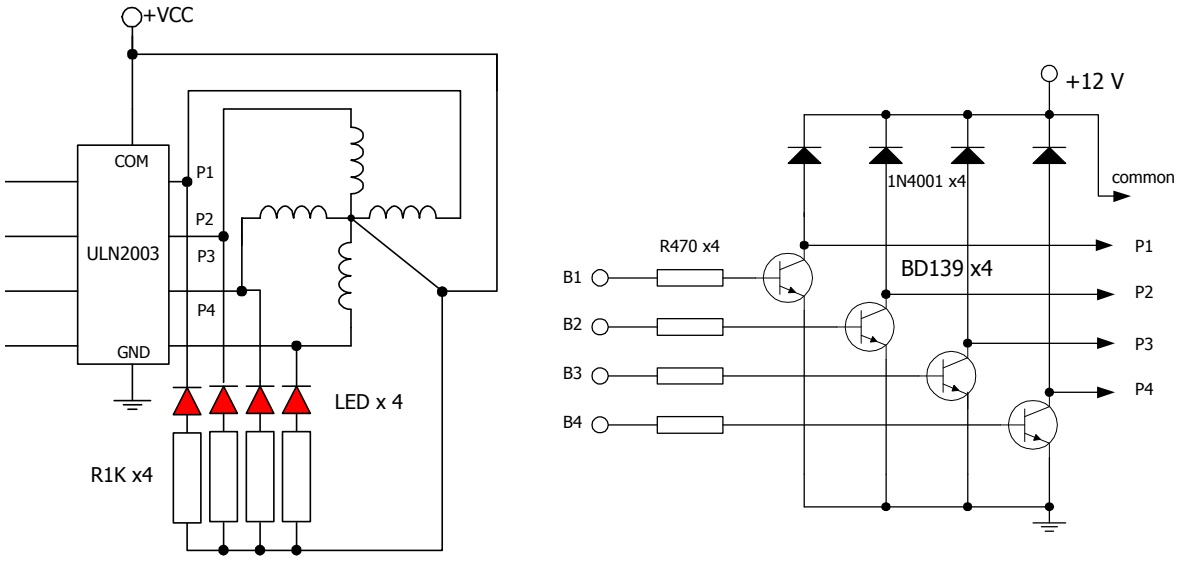
สเต็ปที่	เฟสที่1	เฟสที่2	เฟสที่3	เฟสที่4
1	1	1	0	0
2	0	1	1	0
3	0	0	1	1
4	1	0	0	1

ข. แบบฟูลสเต็ป 2 เฟส

สเต็ปที่	เฟสที่1	เฟสที่2	เฟสที่3	เฟสที่4
1	1	0	0	0
2	1	1	0	0
3	0	1	0	0
4	0	1	1	0
5	0	0	1	0
6	0	0	1	1
7	0	0	0	1
8	1	0	0	1

ค. แบบฮาลฟ์สเต็ป 2 เฟส

ภาพที่ 6.24 แสดงการกระตุ้นเฟสแบบต่างๆของสเต็ปเปอร์มอเตอร์

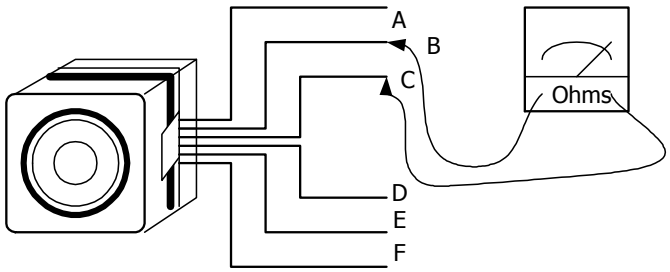


ภาพที่ 6.25 แสดงการต่อวงจรขับสเต็ปเปอร์มอเตอร์โดยใช้ไอซีสำเร็จรูป และวงจรทรานซิสเตอร์

วงจรที่ใช้ในการขับสเต็ปเปอร์มอเตอร์โดยใช้ไอซีสำเร็จรูป และวงจรจากทรานซิสเตอร์ แสดงดังภาพที่ 6.25 โดยไอซีสำเร็จรูปเบอร์ ULN2003 มีคุณสมบัติเป็นไอซีไดรเวอร์กระแสสูงแบบคอลเล็คเตอร์เปิด สามารถเลือกแรงดันได้กว้าง 5-30 โวลต์ จ่ายกระแสได้สูงถึง 500 มิลลิแอมป์ต่อขา และภายในมีไดโอดที่ป้องกันกระแสย้อนกลับภายในไอซี ส่วนแอลอีดีที่ต่อในวงจรเพื่อแสดงการกระตุ้นเฟส ของแต่ละแบบ

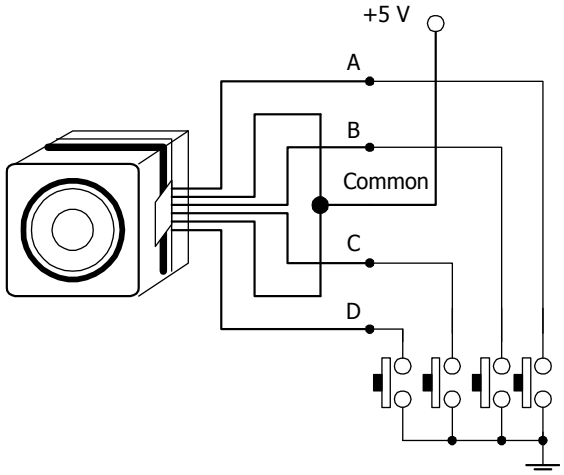
4.3 วิธีการตรวจสอบหาเฟสของขดลวดสเต็ปเปอร์มอเตอร์

ขั้นตอนที่ 1 ให้สังเกตสเต็ปเปอร์มอเตอร์ที่นำมาตรวจสอบ เป็นแบบยูนิโพล่าร์ที่มีจำนวนสาย 5 เส้นหรือ 6 เส้น



ภาพที่ 6.26 การใช้มัลติมิเตอร์วัดค่าความต้านทาน

ขั้นตอนที่ 2 ใช้มัลติมิเตอร์ตั้งค่าไว้สำหรับย่านการวัดค่าความต้านทานโดยวัดระหว่างเส้นลวดแต่ละขด กับจุดต่อร่วม แสดงดังภาพที่ 6.26 ลำดับแรกให้หาสายที่ต่อเป็นจุดร่วมก่อน โดยสังเกตค่าความต้านทานถ้าหากไม่ได้วัดระหว่าง จุดต่อร่วมกับสายแต่ละเส้น ค่าความต้านทานมีค่าเป็น 2 เท่าของการวัด ตัวอย่างเช่น ถ้าให้จุด B เป็นจุดร่วม หากวัดระหว่างที่จุด A กับจุด B มีค่า 60 Ohm แต่ถ้าวัดระหว่างที่จุด A และจุด C ซึ่งไม่ใช่จุดร่วมได้ค่าเท่ากับ 120 Ohm หากเป็นแบบที่มีสาย 6 เส้นมีจุดร่วมสองจุด เพราะมีขดลวดคนละชุดกัน และสายที่เป็นจุดร่วมส่วนใหญ่จึงมีสีเหมือนกัน ในทำนองเดียวกันหากเป็นแบบที่มีสาย 5 เส้นมีจุดร่วมเพียงจุดเดียว



ภาพที่ 6.27 แสดงการต่อวงจรเพื่อทดสอบโดยการสวิตช์เพื่อหาลำดับของเฟส

ขั้นตอนที่ 3 ถ้าสเต็ปเปอร์มอเตอร์เป็นแบบที่มีสาย 6 เส้นให้ต่อจุดร่วมเข้าด้วยกัน ได้เป็น 5 เส้น หลังจากนั้นให้ต่อวงจรตามภาพที่ 6.27 ให้ทดลองกดสวิตช์ ที่ต่อเข้ากับแต่ละจุดโดยเริ่มที่ จุด A จุด B จุด C และจุด D แล้วให้สังเกตการหมุนของสเต็ปเปอร์มอเตอร์ต่อเนื่องหรือไม่ หากมีการกระโดดข้ามสเต็ปให้ทดลองโดยเรียงลำดับการกดสวิตช์ใหม่ จนหาลำดับของสายได้ถูกต้อง มอเตอร์หมุนตามสเต็ปอย่างเป็นลำดับ

4.4 การเขียนโปรแกรมเชื่อมต่อกับสเต็ปเปอร์มอเตอร์

ตัวอย่างที่ 7 เขียนโปรแกรมขับสเต็ปเปอร์มอเตอร์โดยให้มีการกระตุ้นเฟส แบบ 1 เฟส 2 เฟส และ ครึ่งเฟสโดยใช้วิธีการส่งข้อมูลแบบไบต์

```
;************************************************
;***       PROGRAM STEPPING MOTOR CONTROL       ***
;************************************************
                ORG    0000H                ; 1 เริ่มต้นโปรแกรมที่แอดเดรส 0000H
                MOV    P1, #00000000B       ; 2 กำหนดข้อมูลให้พอร์ต P1= 00000000B
LOOP:           MOV    P1,#01H              ; 3 กำหนดให้พอร์ต P1=00000001B ในเฟสที่ 1
                ACALL DELAY                 ; 4 เรียกโปรแกรมย่อยการหน่วงเวลา
                MOV    P1, #02H             ; 5 กำหนดให้พอร์ต P1=00000010B ในเฟสที่ 2
                ACALL DELAY                 ; 6 เรียกโปรแกรมย่อยการหน่วงเวลา
                MOV    P1, #04H             ; 7 กำหนดให้พอร์ต P1=00000100B ในเฟสที่ 3
                ACALL DELAY                 ; 8 เรียกโปรแกรมย่อยการหน่วงเวลา
                MOV    P1, #08H             ; 9 กำหนดให้พอร์ต P1=00001000B ในเฟสที่ 4
                ACALL DELAY                 ; 10 เรียกโปรแกรมย่อยการหน่วงเวลา
                SJMP   LOOP                 ; 11 กระโดดไปเริ่มต้นวนรอบใหม่
;************************************************
;** โปรแกรมย่อยหน่วงเวลา (ความเร็วของสเต็ปเปอร์มอเตอร์มอเตอร์)    ***
;************************************************
DELAY:          MOV    R5, #0FH     ; 12 กำหนดให้ค่าข้อมูลที่ รีจิสเตอร์ R5 มีค่าคงที่เท่ากับ 00H
DELAY1:         MOV    R6, #0FFH    ; 13 กำหนดให้ค่าข้อมูลที่ รีจิสเตอร์ R6มีค่าคงที่เท่ากับ 0FH
                DJNZ   R6, $        ; 14
                DJNZ   R5, DELAY1   ; 15
                RET                 ; 16 ออกจากโปรแกรมย่อยหน่วงเวลา
                END                 ; 17 จบโปรแกรม
```

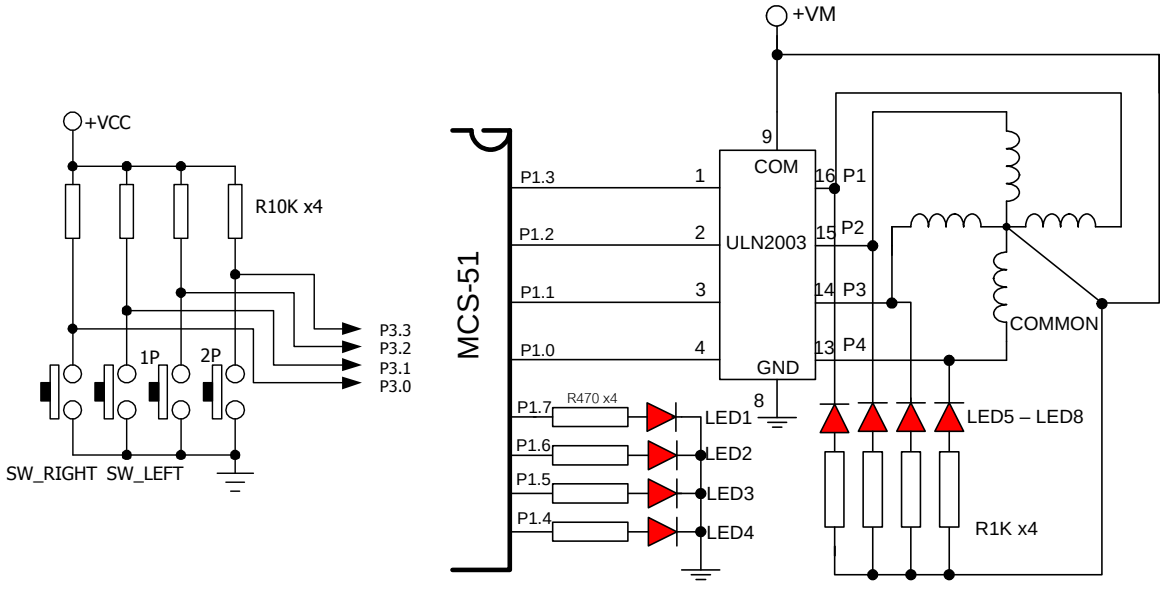
จากคำสั่งในโปรแกรมสเต็ปเปอร์มอเตอร์ถูกกระตุ้นแบบ 1 เฟส และหมุนไปตลอด ถ้าหากต้องการกระตุ้นเฟสในรูปแบบอื่นๆ สามารถแก้ไขโปรแกรมในบรรทัดที่ 3,5,7,9 โดยกำหนดค่าข้อมูลที่พอร์ต P1 ถ้าเป็นการกระตุ้นแบบ 2 เฟส ให้กำหนดข้อมูลทั้ง 4 ค่าคือ MOV P1,# 0CH ,MOV P1,# 06H MOV P1,# 03H และ MOV P1,#,09H แต่ในรูปแบบของการกระตุ้นแบบเฟสครึ่งต้องกำหนดค่าที่พอร์ต P1 อีก 4 ค่าเป็น 8 ค่าคือ 01H,03H,02H,06H,04H,0CH,08H,09H การกลับทิศทางการหมุนของสเต็ปเปอร์มอเตอร์ให้กำหนดค่าที่พอร์ต P1 โดยสลับค่าข้อมูลจากบรรทัดที่ 3 เป็น 9 , บรรทัดที่ 5 เป็น ,7 บรรทัดที่ 7 เป็น 5 และบรรทัดที่ 9 เป็น 3 การกำหนดความเร็วของสเต็ปเปอร์มอเตอร์ กำหนดค่าข้อมูลของรีจิสเตอร์ R5 และ R6 ในโปรแกรมย่อยการหน่วงเวลา ถ้าหากกำหนดให้มีการหน่วงเวลาน้อยเกินไป ทำให้สเต็ปเปอร์มอเตอร์ ไม่สามารถหมุนได้ เนื่องจากสถานะของลอจิก “1” มีช่วงเวลาสั้นเกินไปที่สามารถกระตุ้นเฟสให้โรเตอร์เคลื่อนได้

ตัวอย่างที่ 8 กำหนดให้สวิตซ์ SW1 และ SW2 ทำหน้าที่เลือกทิศทาง การหมุนของมอเตอร์ ส่วน สวิตซ์ SW3 และ SW4 กำหนดให้เลือกวิธีการกระตุ้นสเต็ปเปอร์มอเตอร์ในรูปแบบ 1 เฟส และแบบ 2 เฟสวงจรการควบคุมสเต็ปเปอร์มอเตอร์แสดงได้ดังภาพที่ 6.28 กำหนดการทำงานดังนี้

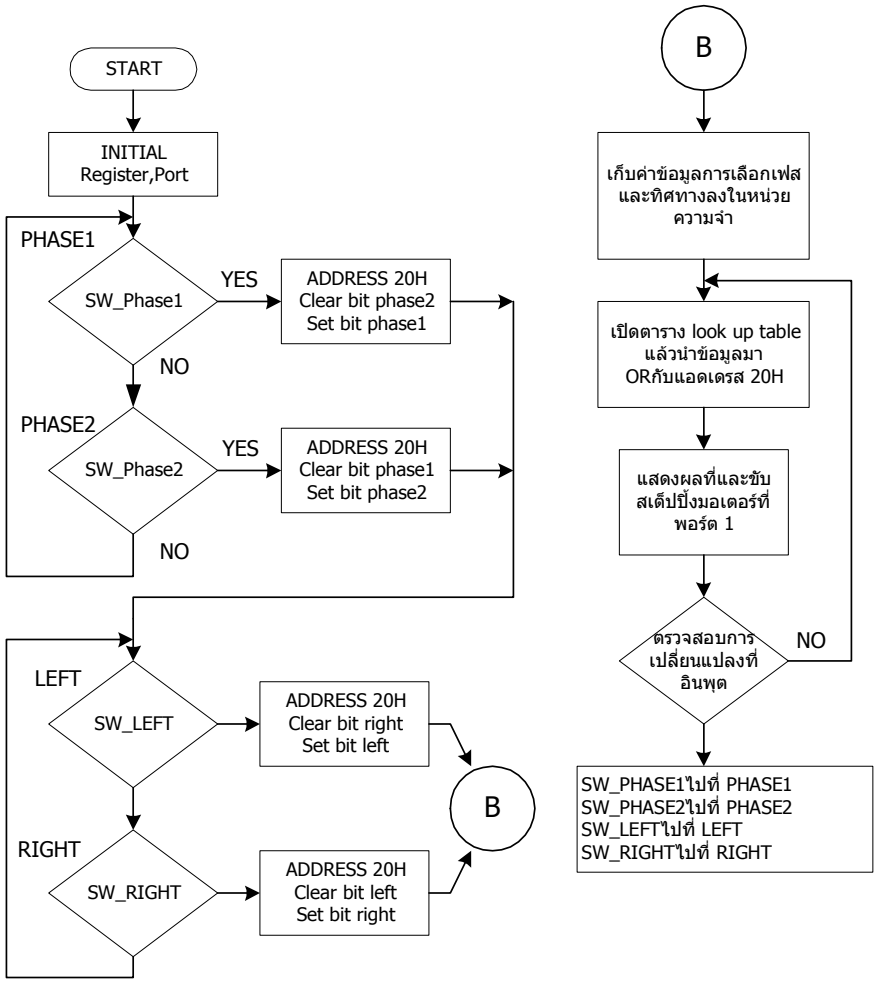
1) กำหนดค่าพอร์ตให้เป็นอินพุตโดยใช้พอร์ต P3.0 และ P3.1 เป็นการเลือกทิศทางการหมุน ส่วน P3.2 และ P3.3 เป็นการเลือกรูปแบบการกระตุ้นเฟส

2) มีการตรวจสอบการเปลี่ยนสถานะของอินพุตจากสวิตซ์ที่ P3.0, P3.1, P3.2 และ P3.3 เมื่อบิตใดบิตหนึ่งมีสถานะลอจิก “0” หรือคีย์สวิตซ์ถูกกด ให้กระโดดไปทำเลเบลที่กำหนดไว้ในคำสั่งนั้น โดยจะให้เลือกทิศทางการหมุนหรือวิธีการกระตุ้นเฟส

จากเงื่อนไขสามารถเขียนเป็นผังงานแสดงดังภาพที่ 6.29



ภาพที่ 6.28 แสดงการต่อวงจรสเต็ปเปอร์มอเตอร์ และสวิตซ์ควบคุม



ภาพที่ 6.29 แสดงผังงานโปรแกรมควบคุมสเต็ปเปอร์มอเตอร์

การเขียนโปรแกรม สามารถอธิบายเป็นส่วนการทำงานได้ดังต่อไปนี้

ส่วนที่ 1 ให้กำหนดสวิตช์เลือกทิศทาง และรูปแบบ โดยให้หน่วยความจำข้อมูลในแอดเดรสที่ 20 H ชื่อ DATA เพื่อเก็บสถานะของการกดสวิตช์ และกระตุ้นเฟส

```
SW_RIGHT    BIT    P3.0    ; กำหนดเป็นสวิตซ์เลือกสเต็ปเปอร์มอเตอร์หมุนไปทางด้านขวา
SW_LEFT     BIT    P3.1    ; กำหนดเป็นสวิตซ์เลือกสเต็ปเปอร์มอเตอร์หมุนไปทางด้านซ้าย
SW_2PHASE   BIT    P3.2    ; กำหนดเป็นสวิตซ์เลือกวิธีการกระตุ้นแบบ 2 เฟส
SW_1PHASE   BIT    P3.3    ; กำหนดเป็นสวิตซ์เลือกวิธีการกระตุ้นแบบ 1 เฟส
DATA        EQU    20H     ; กำหนดเนื้อที่ในหน่วยความจำข้อมูลที่ตำแหน่งแอดเดรส 20 H
```

เพื่อที่จะเก็บค่าจากการเลือกทิศทาง และรูปแบบของการกระตุ้น หลังจากนั้นจะนำไปรวมกับค่าข้อมูลที่ นำไปขับสเต็ปเปอร์มอเตอร์ (Lookup Table ในส่วนที่ 2)

ส่วนที่ 2 รูปแบบการกระตุ้นเฟสของสเต็ปเปอร์มอเตอร์ กำหนดข้อมูลไว้ในส่วนของโปรแกรมแบบ Lookup Table ดูได้จากตารางการกระตุ้นเฟสแบบ 1 เฟส และ 2เฟส ให้เขียนโปรแกรมกำหนดค่าที่ P1 P2 P3 P4 ไว้ส่วนท้ายของโปรแกรมดังนี้

```
PHASE1L:  DB   00001000B, 00000100B, 00000010B, 00000001B ; หมุนไปทางซ้ายแบบ 1 เฟส
PHASE1R:  DB   01H, 02H, 04H, 08H                         ; หมุนไปทางขวาแบบ 1 เฟส
PHASE2L:  DB   00001100B, 00000110B, 00000011B, 00001001B ; หมุนไปทางซ้ายแบบ 2 เฟส
PHASE2R:  DB   09H, 03H, 06H, 0CH                         ; หมุนไปทางขวาแบบ 2 เฟส
```

ส่วนที่ 3 ตรวจสอบการกดสวิตซ์ ที่ใช้เลือกรูปแบบการกระตุ้นเฟสของสเต็ปเปอร์มอเตอร์ หลังจากนั้น ทำการกำหนดบิต ในตำแหน่งของหน่วยความจำที่จองไว้ตามเงื่อนไขกำหนด

```
SCAN_PHASE: JNB   SW_1PHASE,PHASE1 ;ตรวจสอบการกดคีย์เพื่อเลือกการกระตุ้นแบบ 1 เฟส
            JNB   SW_2PHASE,PHASE2 ;ตรวจสอบการกดคีย์เพื่อเลือกการกระตุ้นแบบ 2 เฟส
            SJMP  SCAN_PHASE       ;ถ้าไม่มีการกดคีย์ใดๆให้วนไปตรวจสอบการกดคีย์ใหม่
            .
PHASE1:     CLR   06H    ; เคลียร์ข้อมูลบิต 06H ของแอดเดรสที่ 20H
            SETB  07H    ;เซตข้อมูลบิต 06H ของแอดเดรส 20H แสดงถึงการเลือกแบบ 1เฟส
            MOV   P1,20H ;นำค่าข้อมูลแอดเดรสที่ 20H แสดงผลที่ P1 เพื่อให้แอลอีดีที่กำหนดการ
                         ; แสดงเป็น 1 เฟสสว่าง
            SJMP  SCAN_DIR ; โปรแกรมย่อยการเลือกทิศทางการหมุนของสเต็ปเปอร์มอเตอร์
```

การกำหนดข้อมูลที่บิต 07H โดยเซตให้เป็น “1” ในขณะที่ทำการเลือกรูปแบบการกระตุ้นเฟสที่บิต 00H-03H ของแอดเดรส 20H จะยังไม่สนใจมีข้อมูลอะไรอยู่

1phase	2phase	Left	Right	ไม่สน	ไม่สน	ไม่สน	ไม่สน
07H	06H	05H	04H	03H	02H	01H	00H

หน่วยความจำตำแหน่ง 20 H เป็นส่วนที่สามารถเข้าแบบบิตและไบต์ได้

ส่วนที่ 4 ตรวจสอบการเลือกทิศทางการหมุนของมอเตอร์จากการเลือกที่คีย์สวิตช์

```
SCAN_DIR:  JNB   SW_1PHASE,SCAN_PHASE
           JNB   SW_2PHASE,SCAN_PHASE
           JNB   SW_LEFT,LEFT
           JNB   SW_RIGHT,RIGHT
           SJMP  SCAN_DIR
```

เป็นการตรวจสอบการกดสวิทช์ทุกตัวเพื่อตรวจสอบว่ามีการกดเพื่อเปลี่ยนแปลงข้อมูลหรือไม่ และในการตรวจสอบจะเช็คการกดเลือกเฟสก่อน

.

.

LEFT: CLR 04H ; เคลียร์ข้อมูลบิต 04H ของแอดเดรสที่ 20H (ไม่ใช่ทางขวา)

SETB 05H ; เซตบิต 05H ของแอดเดรส 20H แสดงการเลือกทิศทางหมุนไปทางซ้าย

SJMP **DIS_1LEFT** ; กระโดดไปโปรแกรมย่อย DIS_1LEFT เพื่อกำหนดค่าเริ่มต้น

; ให้กับ รีจีสเตอร์ DPTR

1phase	2phase	Left	Right	ไม่สน	ไม่สน	ไม่สน	ไม่สน
07H	06H	05H	04H	03H	02H	01H	00H

แสดงข้อมูลที่ตำแหน่งแอดเดรส 20H เลือกการกระตุ้นแบบ 1 เฟสและเลือกทิศทางการหมุนไปทางด้านซ้าย

*** ข้อสังเกต** ที่บิตของ 07 H (1 Phrase) และ 05 H (Left) จะถูกกำหนดไว้เรียบร้อยแล้ว ทำการแสดงผลที่แอลอีดีแสดงสถานะของรูปแบบการกระตุ้น และทิศทางการหมุน (LED 1 – LED 4) ส่วนที่เหลือบิต 01H – 03H จะนำไปรวมกับข้อมูลในตาราง (Lookup Table)

ส่วนที่ 5 กำหนดค่าเริ่มต้นให้กับรีจีสเตอร์ DPTR เพื่อเป็นฐานแอดเดรสให้กับตารางข้อมูล ที่มีรูปแบบของการกระตุ้นเฟส

;********** 1 PHASE LEFT**********

DIS_1LEFT: MOV A,#10100000B ;นำค่าคงที่ 10100000B เก็บไว้ที่รีจิสเตอร์ A

CJNE A,DATA,DIS_2LEFT ; เปรียบเทียบค่าข้อมูลในรีจิสเตอร์ A กับค่าข้อมูลใน

; แอดเดรสที่ 20H (DATA) ถ้าไม่เท่าให้กระโดดไปที่

; เลเบล DIS_2LEFT แต่ถ้าเท่ากันให้ทำบรรทัดถัดไป

MOV DPTR,#PHASE_1L ; นำค่าแอดเดรสของ PHASE_1L เก็บที่ Reg. DPTR

SJMP **DISP_MOTOR** ; กระโดดไปที่ DISP_MOTOR เพื่อแสดงผล และ

; ขับสเปปิ้งมอเตอร์

ส่วนที่ 6 นำข้อมูลออกไปที่พอร์ต P1 โดยการนำมาจาก Lookup Table ตำแหน่งแอดเดรส PHASE_1L จากคำสั่ง MOV A,@A+DPTR โดยการนำข้อมูลที่อยู่ในรีจิสเตอร์ A มาทำการ OR กับข้อมูลในตำแหน่งแอดเดรส DATA (ข้อมูลที่แอดเดรส 20H)

ORL A, DATA (โปรแกรมกำหนดให้ DATA EQU 20H)

;**** DISPLAY & MOTOR DRIVE ****

DISP_MOTOR:MOV R0,#04H ; กำหนด 4 สเต็ปในการแสดงผล (Counter)

MOV R1,#00H ; เริ่มต้นให้รีจิสเตอร์ R1 เท่ากับ 00H

```
        MOV   A,#00H          ; เริ่มต้นให้รีจิสเตอร์ A เท่ากับ 00H
LOOP:   MOV   A,R1            ; นำค่าข้อมูลในรีจิสเตอร์ R1 ไปเก็บไว้ที่รีจิสเตอร์ A
        MOVC  A,@A+DPTR       ; นำข้อมูลจากแอดเดรสที่กำหนดโดยเปิดตารางเก็บไว้ที่ Reg A
        ORL   A,DATA          ; นำค่าที่ได้จากตารางOR ค่าใน DATA ผลลัพธ์เก็บไว้ที่ Reg A
        MOV   P1,A            ; นำค่าจากรีจิสเตอร์ A แสดงผลที่พอร์ต P1
        ACALL DELAY           ; เรียกโปรแกรมย่อยหน่วงเวลา
        INC   R1              ; เพิ่มค่าข้อมูลในรีจิสเตอร์ R1(R1 = R1+1)
        JNB   SW_1PHASE,SCAN_PHASE    ; ตรวจสอบการเปลี่ยนแปลงที่อินพุตคีย์
        JNB   SW_2PHASE,SCAN_PHASE    ; ตรวจสอบการเปลี่ยนแปลงที่อินพุตคีย์
        JNB   SW_LEFT,LEFT            ; ตรวจสอบการเปลี่ยนแปลงที่อินพุตคีย์
        JNB   SW_RIGHT,RIGHT          ; ตรวจสอบการเปลี่ยนแปลงที่อินพุตคีย์
        DJNZ  R0,LOOP                 ; วนให้ครบ 4 สเต็ป
        SJMP  DISP_MOTOR              ; กระโดดไปทำที่เลเบล DISP_MOTOR ใหม่
```

ข้อมูล ที่ 20H	1	0	1	0	0	0	0	0
ข้อมูลที่ Reg A	0	0	0	0	1	0	0	0
ผลลัพธ์ที่ Reg A	1	0	1	0	1	0	0	0
แอลอีดีแสดงผลที่ พอร์ต P1 ในสเตป 1	1phase	-	L	-	Step1	-	-	-
แอลอีดีแสดงผลที่ พอร์ต P1 ในสเตป 2	1phase	-	L	-	-	Step2	-	-
แอลอีดีแสดงผลที่ พอร์ต P1 ในสเตป 3	1phase	-	L	-	-	-	Step3	-
แอลอีดีแสดงผลที่ พอร์ต P1 ในสเตป 4	1phase	-	L	-	-	-	-	Step4

แสดงค่าผลลัพธ์จากการกระทำทางลอจิก OR ระหว่างข้อมลที่ในแอดเดรส 20H (DATA)และค่าข้อมลในรีจิสเตอร์ A

การนำข้อมูลระหว่างแอดเดรสที่ 20H มารวมกับข้อมูลในรีจิสเตอร์ A ที่ได้จากวิธีของ Lookup Table แล้ว หลังจากนั้นผลลัพธ์ของข้อมูลจะอยู่ในรีจิสเตอร์ A ซึ่งเป็นข้อมูลที่บอกถึง ทิศทางการหมุน รูปแบบของการกระตุ้น และการกระตุ้นทีละสเต็ป ของสเต็ปเปอร์มอเตอร์ โดยจะนำไปแสดงผลที่แอลอีดี และวงจรขับสเต็ปเปอร์มอเตอร์

สามารถนำเอาโปรแกรมทั้งหมดมาเขียนต่อรวมกันได้ดังนี้

```
;************** ส่วนที่ 1 ****************
SW_RIGHT     BIT     P3.0
SW_LEFT      BIT     P3.1
SW_2PHASE    BIT     P3.2
SW_1PHASE    BIT     P3.3
DATA         EQU      20H
             ORG     0000H
             SETB    SW_1PHASE
             SETB    SW_2PHASE
             SETB    SW_LEFT
             SETB    SW_RIGHT
             MOV     P1,#00000000B
             MOV     DATA,#00000000B
              CLR    A
;********************** SELECT PHASE ****************************
;************** ส่วนที่ 3 ****************
SCAN_PHASE: JNB      SW_1PHASE, PHASE1
             JNB     SW_2PHASE, PHASE2
             SJMP    SCAN_PHASE
PHASE1:      MOV     DATA, #00H
             CLR     06H
              SETB   07H
              MOV    P1,DATA
              SJMP   SCAN_DIR
PHASE2:      MOV     DATA,#00H
             CLR     07H
             SETB    06H
             MOV     P1,DATA
             SJMP    SCAN_DIR
;***********   SELECT DIR  *************
;************** ส่วนที่ 4 ****************
SCAN_DIR:    JNB     SW_1PHASE,SCAN_PHASE
```

```
                JNB     SW_2PHASE,SCAN_PHASE
                JNB     SW_LEFT,LEFT
                JNB     SW_RIGHT,RIGHT
                SJMP    SCAN_DIR
LEFT:           CLR     04H
                SETB    05H
                MOV     P1,DATA
                SJMP    DIS_1LEFT
RIGHT:          CLR     05H
                SETB    04H
                MOV     P1,DATA
                SJMP    DIS_1RIGHT
;******* MOV  INDEX  Reg. DPTR  ******
;************** ส่วนที่ 5 ***************
;********   1 PHASE LEFT**************
DIS_1LEFT:      MOV   A,#10100000B
                CJNE   A,DATA,DIS_2LEFT
                MOV   DPTR,#PHASE1L
                SJMP   DISP_MOTOR
;********   1 PHASE  RIGHT*************
DIS_1RIGHT:    MOV   A,#10010000B
                CJNE   A,DATA,DIS_2RIGHT
                MOV   DPTR,#PHASE1R
                SJMP   DISP_MOTOR
;*********  2 PHASE  LEFT**************
DIS_2LEFT:      MOV   A,#01100000B
                CJNE   A,DATA,SCAN_PHASE
                MOV   DPTR,#PHASE2L
                SJMP   DISP_MOTOR
;********** 2 PHASE  RIGHT ************
DIS_2RIGHT:   MOV   A,#01010000B
                CJNE   A,DATA,SCAN_PHASE
```

```
                MOV    DPTR,#PHASE2R
                SJMP   DISP_MOTOR
;******** DISPLAY & MOTOR DRIVE ******
;************* ส่วนที่ 6 ***************
DISP_MOTOR: MOV    R0,#04H
                MOV    R1,#00H
                MOV    A,#00H
LOOP:           MOV    A,R1
                MOVC   A,@A+DPTR
                ORL    A,DATA
                MOV    P1,A
                ACALL  DELAY
                INC    R1
                JNB    SW_1PHASE, SCAN_PHASE
                JNB    SW_2PHASE, SCAN_PHASE
                JNB    SW_LEFT, LEFT
                JNB    SW_RIGHT, RIGHT
                DJNZ   R0, LOOP
                SJMP   DISP_MOTOR
;*********    โปรแกรมย่อยหน่วงเวลา เพื่อกำหนดความเร็วให้สเต็ปเปอร์มอเตอร์   ********
DELAY:          MOV    R5,#00H
DELAY1:         MOV    R6,#0FH
                DJNZ   R6,$
                DJNZ   R5, DELAY1
                RET
;************ ส่วนที่ 2 *****************
PHASE1L:        DB     00001000B, 00000100B, 00000010B, 00000001B
PHASE1R:        DB     01H, 02H, 04H, 08H
PHASE2L:        DB     00001100B, 00000110B, 00000011B, 00001001B
PHASE2R:        DB     09H, 03H, 06H, 0CH
                END
```

5. การประยุกต์ใช้งานการรับส่งข้อมูล กับอุปกรณ์เชื่อมต่อภายนอก

5.1 คุณสมบัติวงจรใช้งานคีย์เมตริกซ์ 3×4 หรือ 4×4

คีย์เมตริกซ์ 3 ×4 หรือ 4×4 โดยใช้ไอซี MCS-51 จะทำหน้าที่เป็นวงจรถอดรหัสคีย์แบบเมตริกซ์ มีขอบเขตดังนี้

5.1.1 ใช้งานกับคีย์แบบเมตริกซ์ขนาด 3×4, 4×4 แสดงผลเอาต์พุตเป็นเลขไบนารี

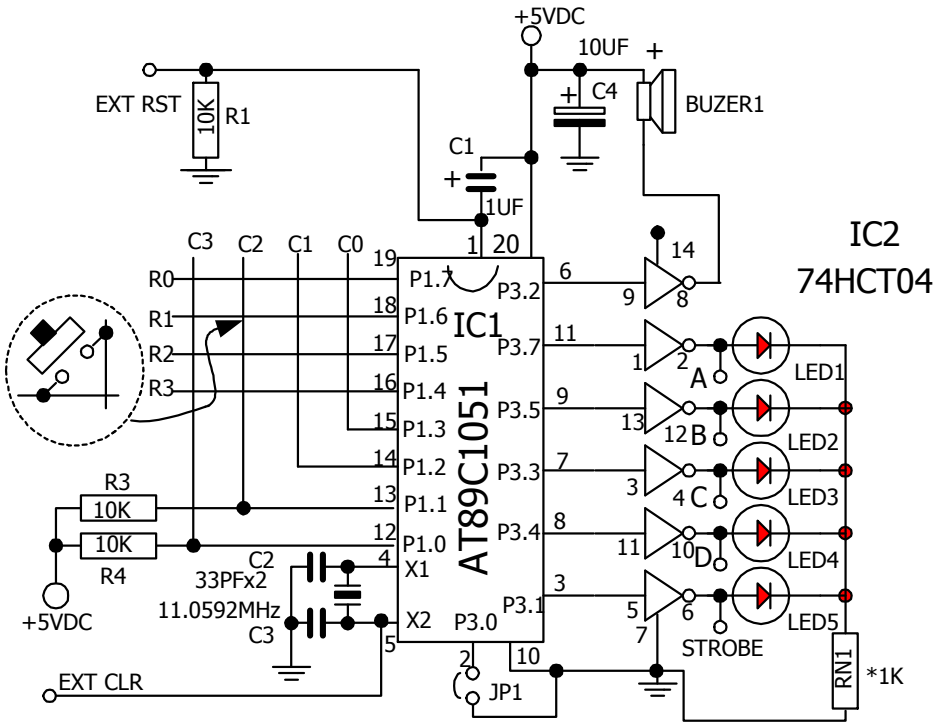
5.1.2 มีสัญญาณในการกดทุกครั้ง

5.1.3 มีสัญญาณเป็นเสียงทุกครั้งที่กดสวิตซ์

5.1.4 ใช้สัญญาณรีเซต สัญญาณนาฬิกา และแหล่งจ่ายไฟจากภายนอก

5.2 การทำงานของวงจร

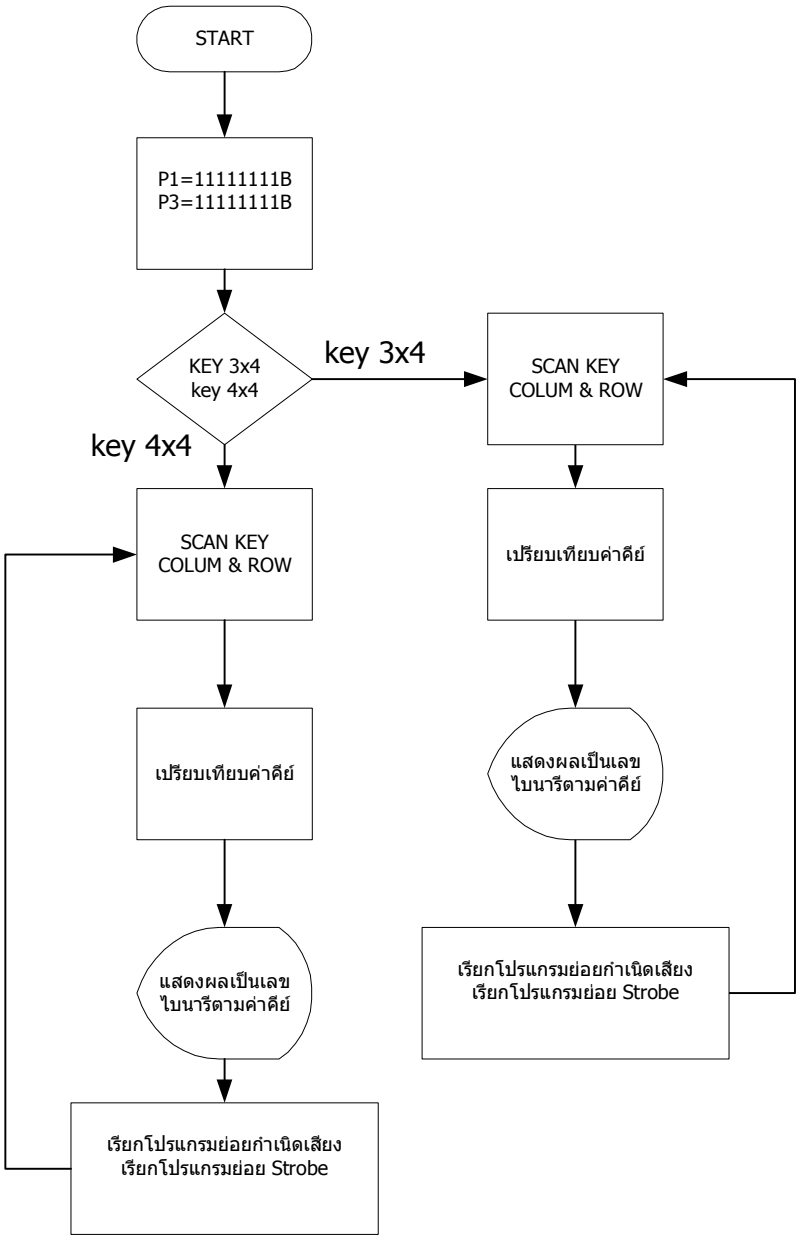
IC1 เป็นไอซีไมโครคอนโทรลเลอร์เบอร์ AT89C1051 ของ Atmel ทำหน้าที่ควบคุม และสั่งงานในส่วนของโปรแกรมทั้งหมด วงจรแสดงในภาพที่ 6.30 เริ่มจากกำหนดการสแกนคีย์ และเปรียบเทียบค่าข้อมูลที่ได้จากการถอดรหัสสัญญาณ หลังจากนั้นจะแสดงผลข้อมูลเป็นรหัสไบนารี พร้อมทั้งสร้างสัญญาณเอาต์พุตตามที่ต้องการ



ภาพที่ 6.30 วงจรการถอดรหัสคีย์บอร์ดแบบเมตริกซ์

IC2 ทำหน้าที่เป็นบัฟเฟอร์เพื่อนำไปใช้งานแสดงผลข้อมูลที่ LED1 - LED5 และขับ BUZER ส่วน R1 และ C1 เป็นวงจรรีเซตไอซี หากใช้วงจรรีเซตจากภายนอก ไม่ต้องต่อ C1, R1 ส่วน C2, C3 และ X-tal เป็นสัญญาณนาฬิกา ป้อนให้กับไอซีไมโครคอนโทรลเลอร์ หากใช้สัญญาณนาฬิกาจากภายนอกไม่ต้องต่อ C2,C3 และ X-tal ส่วนจัมเปอร์ JP1 ที่ขา P3.0 เป็นการเลือกขนาดของจำนวนคีย์ที่ใช้ ถ้าเป็นคีย์แพดสวิตซ์ 12 ตัวให้ต่อจุด JP1 (Close) และถ้าเป็นแบบสวิตซ์ 16 ตัวให้เปิดจุดต่อที่ JP1 (Open) ส่วนแหล่งจ่ายไฟในวงจรจะใช้ IC3 เบอร์ 78L05 รักษาระดับแรงดันไฟ +5 VDC ส่วน C4,C5 เป็นวงจรกรองแรงดัน ถ้าใช้แหล่งจ่ายไฟ 5 VOLT จากภายนอกก็ไม่ต้องต่อ IC3 C4 และ C5

5.3 การเขียนโปรแกรม



ภาพที่ 6.31 แสดงผังการทำงานของโปรแกรมคีย์เมตริกซ์ 3 x4 หรือ 4x4

ผังการทำงานของโปรแกรม แสดงภาพที่ 6.31 กำหนดค่าเริ่มต้นของพอร์ตและหน่วยความจำ ตรวจสอบการเลือกขนาดคีย์ 3 x4 หรือ 4x4 จากบิต P3.0 หลังจากนั้นจะกำหนดค่าแถวจาก ROW0 -ROW3 กำหนดค่าทีละหลัก แล้วรับข้อมูลอินพุตจากพอร์ต P1 นำไปเก็บไว้ที่รีจิสเตอร์ A จากนั้นนำมาเปรียบเทียบกับค่าที่กำหนดไว้ หากตรงกับค่าใด จะให้แสดงผลโดยการกำหนดค่าที่บิตของพอร์ต P3 ให้เป็นรหัสไบนารี่ โดยเอาต์พุตบิตที่ P3.1 เป็นแอลอีดีกระพริบ 1 ครั้ง และที่บิต P3.2 ต่อกับบัซเซอร์มีสัญญาณเสียงดัง 1 ครั้ง

```
;*********************************
;*****   key matrix & sound      ******
;*********************************
        OUT_A       BIT     P3.7    ; กำหนดค่าบิตที่ 0 หรือ A (D C B A) แทน P3.7
        OUT_B       BIT     P3.5    ; กำหนดค่าบิตที่ 1 หรือ B (D C B A) แทน P3.5
        OUT_C       BIT     P3.3    ; กำหนดค่าบิตที่ 2 หรือ C (D C B A) แทน P3.3
        OUT_D       BIT     P3.4    ; กำหนดค่าบิตที่ 3 หรือ D (D C B A) แทน P3.4
        BUZER       BIT     P3.2    ; กำหนด BUZER แทน P3.2
        STROBE      BIT     P3.1    ; กำหนดค่าแสดงสัญญาณการกด (STROBE)
        C0          BIT     P1.3    ; กำหนดค่า COLLOUM 0 แทน P1.3
        C1          BIT     P1.2    ; กำหนดค่า COLLOUM 1 แทน P1.2
        C2          BIT     P1.1    ; กำหนดค่า COLLOUM 2 แทน P1.1
        C3          BIT     P1.0    ; กำหนดค่า COLLOUM 3 แทน P1.0
        KEY12_16    BIT     P3.0    ; กำหนดการเลือกขนาดของคีย์ 3x4 ,4x4
```

> ***BIT*** Symbol Name Bit Expression
>
> เป็นไดเร็กทีพที่ใช้กำหนดค่าแอดเดรสของตำแหน่งหน่วยความจำที่เข้าแบบบิต ให้กับสัญลักษณ์ ในกรณีที่มีค่าอยู่ที่หน่วยความจำตำแหน่งแอดเดรส 20H – 2FH (20H.0 – 2FH.7 หรือตำแหน่งของบิตที่ 00-FFH)

```
                ORG     0000H
START:          MOV     P1,#0FFH
                MOV     P3, #0FFH
                MOV     A, #00H
                CALL    DELAY_D             ; หน่วงเวลาสำหรับการเริ่มต้น
                SJNB    KEY12_16, KEY12     ; เลือกขนาดของคีย์ถ้าเป็น 3x4 ให้กระโดดไปที่ KEY12
                SJMP    SCAN                ;ถ้าเป็น 4 x 4 ให้กระโดดไปที่ SCAN
KEY12:          AJMP    SCAN1       ; ระยะกระโดดจากบรรทัดเงื่อนไขไกลเกิน +127 จึงใช้ AJMP
```

SJMP KEY12_16, KEY12 เป็นการกระโดดแบบมีเงื่อนไขเพื่อตรวจสอบการเลือกใช้งานตามขนาดของคีย์แพด ที่มีการวาง ตำแหน่งของเลขคีย์ต่างกัน

```
SCAN:       NOP
;*** ::กำหนดให้ ROW0 เป็นลอจิก "0" ส่งข้อมูลออกพอร์ต P1 แล้วรับค่าจาก P1 เพื่อตรวจสอบการกดจาก COLLOUM ::
SCAN_R0C0:  MOV   P1,#01111111B      ;กำหนดให้สแกนโดย ROW 0=0 SCAN IN COLUM
            MOV   A,P1               ;นำค่าที่ได้จากอินพุตมาไว้ที่รีจิสเตอร์ A
```

;*** ::รับค่าเข้ามาเพื่อเปรียบเทียบกับคีย์โค๊ดของ คีย์ **" 0 "** :: ***

```
            CJNE  A,#01110111B,SCAN_R0C1
            SETB  OUT_A              ; ให้ที่ OUTPUT A =0
            SETB  OUT_B              ; ให้ที่ OUTPUT B =0
            SETB  OUT_C              ; ให้ที่ OUTPUT C =0
            SETB  OUT_D              ; ให้ที่ OUTPUT D =0
            CALL  BEEP               ; เรียกโปรแกรมย่อยกำเนิดเสียง
            CALL  STROBE_SIG         ; เรียกโปรแกรมย่อยกำเนิดสัญญาณ Strobe
            JNB   C0,$               ; ตรวจสอบการปล่อยคีย์ C0
            SJMP  SCAN               ; กระโดดไปเลเบล SCAN
```

;*** ::รับค่าเข้ามาเพื่อเปรียบเทียบกับคีย์โค๊ดของ คีย์ **" 1 "** :: ***

```
SCAN_R0C1:  CJNE  A,#01111011B,SCAN_R0C2
            .
            .
```

;*** ::รับค่าเข้ามาเพื่อเปรียบเทียบกับคีย์โค๊ดของ คีย์ **" 2 "** :: ***

```
SCAN_R0C2:  CJNE  A,#01111101B,SCAN_R0C3
            .
            .
```

;*** ::รับค่าเข้ามาเพื่อเปรียบเทียบกับคีย์โค๊ดของ คีย์ **“ 3 ”** :: ***

SCAN_R0C3: CJNE A,#01111110B,SCAN_R1C0

.

.

;*** ::รับค่าเข้ามาเพื่อเปรียบเทียบกับคีย์โค๊ดของ คีย์ **“ 4 ”** :: ***

SCAN_R1C0: MOV P1,#10111111B

.

.

;*** ::รับค่าเข้ามาเพื่อเปรียบเทียบกับคีย์โค๊ดของ คีย์ **“ 5 ”** :: ***

SCAN_R1C1: CJNE A,#10111011B,SCAN_R1C2

.

.

;*** ::รับค่าเข้ามาเพื่อเปรียบเทียบกับคีย์โค๊ดของ คีย์ **“ 6 ”** :: ***

SCAN_R1C2: CJNE A,#10111101B,SCAN_R1C3

.

.

;*** ::รับค่าเข้ามาเพื่อเปรียบเทียบกับคีย์โค๊ดของ คีย์ **“ 7 ”** :: ***

SCAN_R1C3: CJNE A,#10111110B,SCAN_R2C0

.

.

;*** ::กำหนดให้ ROW1 เป็นลอจิก ”0” ส่งข้อมูลออกพอร์ต P1 แล้วรับค่าจาก P1 เพื่อตรวจสอบการกดจาก COLLOUM ::

SCAN_R2C0: MOV P1,#11011111B ; เปรียบเทียบค่าที่ได้กับค่าที่กำหนดไว้คือ "8"

```
                MOV     A,P1
```

;*** ::รับค่าเข้ามาเพื่อเปรียบเทียบกับคีย์โค๊ดของ คีย์ **“ 8 ”** :: ***

```
                CJNE    A,#11010111B,SCAN_R2C1
                .
                .
```

;*** ::รับค่าเข้ามาเพื่อเปรียบเทียบกับคีย์โค๊ดของ คีย์ **“ 9 ”** :: ***

```
SCAN_R2C1:      CJNE    A,#11011011B,SCAN_R2C2
                .
                .
```

;*** ::รับค่าเข้ามาเพื่อเปรียบเทียบกับคีย์โค๊ดของ คีย์ **“ A ”** :: ***

```
SCAN_R2C2:      CJNE    A,#11011101B,SCAN_R2C3
                .
```

;*** ::รับค่าเข้ามาเพื่อเปรียบเทียบกับคีย์โค๊ดของ คีย์ **“ B ”** :: ***

```
SCAN_R2C3:      CJNE    A,#11011110B,SCAN_R3C0
                .
                .
                SJMP    END_SCAN
;*** ::กำหนดให้ ROW2 เป็นลอจิก ”0” ส่งข้อมูลออกพอร์ต P1 แล้วรับค่าจาก P1 เพื่อตรวจสอบการกดจาก COLLOUM ::
SCAN_R3C0:      MOV     P1,#11101111B           ; เปรียบเทียบค่าที่ได้กับค่าที่กำหนดไว้คือ "C"
                MOV     A,P1
```

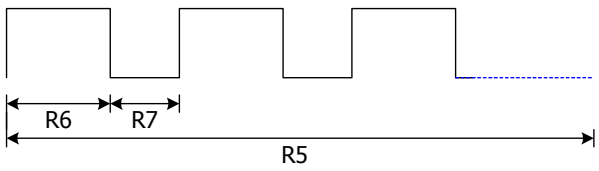
;*** ::รับค่าเข้ามาเพื่อเปรียบเทียบกับคีย์โค๊ดของ คีย์ **“ C ”** :: ***

```
                .
```

```
                .
                ;*** ::รับค่าเข้ามาเพื่อเปรียบเทียบกับคีย์โค๊ดของ คีย์ " D "  :: ***
SCAN_R3C1:      CJNE    A,#11101011B,SCAN_R3C2
                .
                .
                ;*** ::รับค่าเข้ามาเพื่อเปรียบเทียบกับคีย์โค๊ดของ คีย์ " E "  :: ***
SCAN_R3C2:      CJNE    A,#11101101B,SCAN_R3C3
                .
                .
                SJMP    END_SCAN
                ;*** ::รับค่าเข้ามาเพื่อเปรียบเทียบกับคีย์โค๊ดของ คีย์ " F "  :: ***
SCAN_R3C3:      CJNE    A,#11101110B,END_SCAN
                .
                .
END_SCAN:       AJMP    SCAN
;***********  SCAN  KEY 3 x 4        ***********
SCAN1:          NOP
SCAN1_R0C0:     MOV     P1,#01111111B
                MOV     A,P1
                CJNE    A,#01110111B,SCAN1_R0C1     ; เปรียบเทียบค่าที่ได้กับค่าที่กำหนดไว้คือ"1"
                CLR     OUT_A
                CLR     OUT_B
                SETB    OUT_C
                SETB    OUT_D
                CALL    BEEP
                CALL    STROBE_SIG
```

```
                AJMP   END_SCAN1
SCAN1_R0C1:     CJNE   A,#01111011B,SCAN1_R0C2    ; เปรียบเทียบค่าที่ได้กับค่าที่กำหนดไว้คือ "2"
                .
                .
SCAN1_R0C2:     CJNE   A,#01111101B,SCAN1_R1C0    ; เปรียบเทียบค่าที่ได้กับค่าที่กำหนดไว้คือ "3"
                .
                .
                AJMP   END_SCAN1
SCAN1_R1C0:     MOV    P1,#10111111B
                MOV    A,P1
                CJNE   A,#10110111B,SCAN1_R1C1    ; เปรียบเทียบค่าที่ได้กับค่าที่กำหนดไว้คือ "4"
                .
                .
SCAN1_R1C1:     CJNE   A,#10111011B,SCAN1_R1C2    ; เปรียบเทียบค่าที่ได้กับค่าที่กำหนดไว้คือ "5"
                .
                .
SCAN1_R1C2:     CJNE   A,#10111101B,SCAN1_R2C0    ; เปรียบเทียบค่าที่ได้กับค่าที่กำหนดไว้คือ "6"
                .
                .
SCAN1_R2C0:     MOV    P1,#11011111B
                MOV    A,P1
                CJNE   A,#11010111B,SCAN1_R2C1    ; เปรียบเทียบค่าที่ได้กับค่าที่กำหนดไว้คือ "7"
                .
                .
SCAN1_R2C1:     CJNE   A,#11011011B,SCAN1_R2C2    ; เปรียบเทียบค่าที่ได้กับค่าที่กำหนดไว้คือ "8"
                .
                .
SCAN1_R2C2:     CJNE   A,#11011101B,SCAN1_R3C0    ; เปรียบเทียบค่าที่ได้กับค่าที่กำหนดไว้คือ "9"
                .
                .
SCAN1_R3C0:     MOV    P1,#11101111B
                MOV    A,P1
```

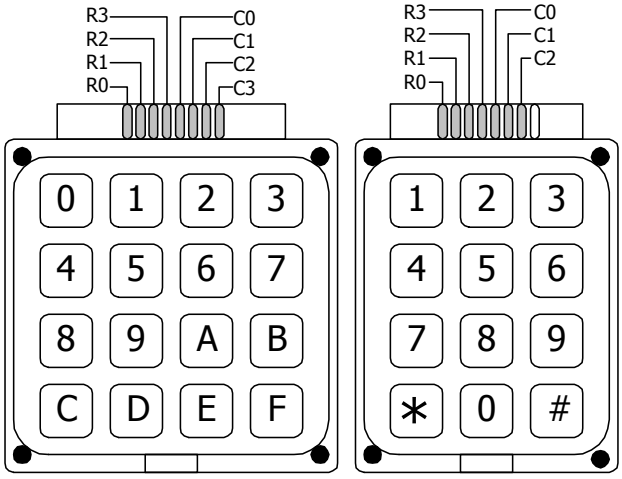
```
CJNE    A,#11100111B,SCAN1_R3C1    ; เปรียบเทียบค่าที่ได้กับค่าที่กำหนดไว้คือ "*
SCAN1_R3C1: CJNE    A,#11101011B,SCAN1_R3C2    ; เปรียบเทียบค่าที่ได้กับค่าที่กำหนดไว้คือ "0"
            .
SCAN1_R3C2: CJNE    A,#11101101B,END_SCAN1     ; เปรียบเทียบค่าที่ได้กับค่าที่กำหนดไว้คือ "#"
            .
END_SCAN1:  JNB     C0,$
            JNB     C1,$
            JNB     C2,$
            AJMP    SCAN1
            ;***  ::ผลิตสัญญาณที่บับเซอร์ 1  BEEP  :: ***
BEEP:       MOV     R5,#50H            ;กำหนดให้ค่ารีจิสเตอร์ R5 =50H
BEEP1:      MOV     R6,#20H            ;กำหนดให้ค่ารีจิสเตอร์ R6 =20H
            MOV     R7,#10H            ;กำหนดให้ค่ารีจิสเตอร์ R7 =50H
DEC_1:      DEC     R6                 ;ลดค่า R6  = R6-1
            NOP                        ; no operant
            NOP
            NOP
            DJNZ    R6,DEC_1           ;R6 = 0 (NO :R6=R6-1),(YES: JUMP DEC_1)
            MOV     R6,#20H
            CPL     BUZER
DEC_2:      DEC     R7
            NOP
            NOP
            NOP
            DJNZ    R7,DEC_2       ;R7 = 0 (NO :R7=R7-1),(YES: JUMP DEC_2)
            MOV     R7,#10H        ;กำหนดให้ค่ารีจิสเตอร์ R7 =10H
            CPL     BUZER          ; กลับค่าที่ BUZER
            DJNZ    R5,BEEP1       ;R5 = 0 (NO :R5=R5-1),(YES: JUMP BEEP1)
            RET
            ;*** ::สร้างสัญญาณให้แอลอีดี STROBE กระพริบ 1 ครั้ง  :: ***
STROBE_SIG: CLR     STROBE
            CALL    DELAY_D
```



```
                SETB    STROBE
                RET
                ;*** ::  โปรแกรมย่อยหน่วงเวลา  :: ***
DELAY_D:        MOV     R4,#02H
DELAY_1:        MOV     R3,#0A0H
Delay_2:        MOV     R2,#0FFH
                DJNZ    R2,$
                DJNZ    R3,DELAY_2
                DJNZ    R4,DELAY_1
                RET
                END
```

***** หมายเหตุ หน้าสัมผัสของคีย์แพดจะมีหน้าสัมผัสที่กว้างอาจมีปัญหาเกี่ยวกับการรบกวนจากการกดของสวิตซ์ได้ง่าย ในโปรแกรมจึงหน่วงเวลาที่ DELAY_D ไว้พอประมาณ

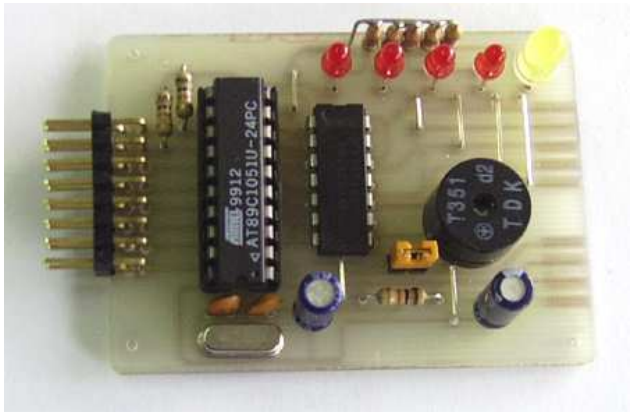
5.4 การทดสอบและใช้งาน



ภาพที่ 6.32 การใช้งานโดยต่อ C0-C3, R0-R3 ให้ตรงตำแหน่งที่ต่อเข้ากับวงจร

การเลือกใช้คีย์ดังแสดงในภาพที่ 6.32 ถ้าเลือกคีย์ขนาด 4×4 ให้ต่อ C0-C3, R0-R3 ตรงตำแหน่งเข้ากับวงจร แล้วกดตัวเลขสังเกตการแสดงผลของแอลอีดีเป็นรหัสเลขไบนารี และจะมีเสียงดังทุกครั้งที่กดคีย์สวิตซ์ พร้อมสัญญาณ Strobe ถ้าใช้คีย์โทรศัพท์ขนาด 3×4 การวางตัวอักษรจะไม่ตรงกันกับแบบคีย์ 4×4 ดังนั้น ให้ต่อ JP1 ขา 1-2 หลังจากนั้นทำการรีเซต สังเกตการทำงานแอลอีดีจะแสดงผลข้อมูลเป็นรหัส ไบนารีแบบไอซีโทรศัพท์ถอดรหัสสัญญาณ DTMF เบอร์ 8870 โดยคีย์ **“ 0 ”** จะแสดง 1010 คีย์ **“ * ”** จะแสดง 1011 และคีย์ **“ # ”** จะแสดง 1100

บอร์ดสำเร็จของวงจรควบคุมคีย์แบบเมตริกซ์แสดงดังภาพที่ 6.33 ขั้นตอนการสร้างสามารถค้นคว้าวิธีการสร้างเพิ่มเติมได้ที่ http://www.adisak51/project03.html



ภาพที่ 6.33 แสดงบอร์ดสำเร็จของวงจรควบคุมคีย์แบบเมตริกซ์

สรุป

พอร์ต คือช่องทางสำหรับโอนย้ายข้อมูลระหว่างไอซี MCS-51กับอุปกรณ์ภายนอก การกำหนดให้พอร์ตใดๆของไอซี MCS-51 เป็นอินพุต ต้องเขียนให้มีสถานะลอจิก “1” แล้วส่งไปแต่ละบิตของพอร์ตเพื่อให้เป็นอินพุต การอ่านค่าลอจิกจากพอร์ตทำได้ 2 ลักษณะคือ การอ่านค่าจากขาพอร์ตโดยตรง และการอ่านค่าจากวงจรแลตซ์ของแต่ละพอร์ต

พอร์ตเอาต์พุต สามารถส่งข้อมูลเป็นลอจิกที่ต้องการออกไปโดยตรง เมื่อเอาต์พุตพอร์ต มีสัญญาณเป็นลอจิก “0” รับกระแสเข้ามา เรียกว่ากระแสซิงค์ เมื่อให้เอาต์พุตมีสัญญาณออกเป็นลอจิก “1” มีกระแสจ่ายออกมาเรียกว่ากระแสซอร์ส

คีย์เมตริกซ์ ให้ไอซี MCS-51 เป็นอินพุตพอร์ต เพื่อทำหน้าที่รับค่าข้อมูลจากคีย์สวิตช์ หากใช้งานสวิตช์ ที่มีจำนวนมากกว่าจำนวนขาของพอร์ตไอซี MCS-51 ที่มีอยู่ จึงใช้วิธีการต่อคีย์สวิตช์ให้เป็นแบบเมตริกซ์ สามารถกำหนดให้พอร์ต P3.0, P3.1, P3.2, P3.3 เป็นพอร์ตเอาต์พุตโดยให้ ผลัดกันส่งข้อมูลที่เป็นสถานะลอจิก “0” ออกทีละบิต แล้วทำการรับข้อมูลเข้ามาที่พอร์ต P3.4, P3.5, P3.7 เพื่อตรวจสอบมีคีย์ใดถูกกดบ้าง ในทุกๆครั้งที่มีการผลัดกันส่งข้อมูล พอร์ตเอาต์พุต P3.0, P3.1, P3.2, P3.3 จึงนำสถานะลอจิกที่ส่งออกไป ในขณะนั้นมารวมกับข้อมูล ที่รับเข้ามาทางอินพุตพอร์ต P3.4, P3.5, P3.7 ขณะที่คีย์ถูกกดซึ่งเป็นค่าคีย์โค๊ดของแต่ละคีย์

แอลอีดี 7 ส่วนแบบหลายหลัก ใช้วิธีการแสดงผลแบบมัลติเพล็กโดยการต่อขาของแอลอีดี 7 ส่วนในแต่ละเซกเมนต์ขนานกับเซกเมนต์เดียวกันของแอลอีดี 7 ส่วนตัวอื่นๆ ทุกตัว การแสดงผลแบบมัลติเพล็กโดยให้แอลอีดี 7 ส่วนติดทีละหลัก ควบคุมที่ขาคอมมอนของแอลอีดี 7 ส่วน แต่ละตัวให้ทำงานแสดงผลแบบมัลติเพล็กแต่ละหลักต้องใช้ความเร็วในการสลับหลัก โดยสามารถมองเห็นการดับของแอลอีดี 7 ส่วนแต่ละหลักได้ทัน ทำให้มองเห็นเหมือนกับว่าแอลอีดี 7 ส่วนแสดงผลทุกหลักติดพร้อมๆกัน ซึ่งการใช้งานในลักษณะมัลติเพล็ก ข้อดีของวิธีนี้จึงทำให้ประหยัดอุปกรณ์ที่นำมาต่อร่วม และปริมาณของกระแสไฟที่ต้องใช้ในระบบทั้งหมด

มอเตอร์กระแสตรง ทำงานโดยวิธีการผ่านกระแสให้กับขดลวดในสนามแม่เหล็กทำให้เกิดแรงแม่เหล็ก ซึ่งขึ้นอยู่กับกระแสและกำลังของสนามแม่เหล็ก

การควบคุมความเร็วของมอเตอร์กระแสตรงโดยการจ่ายกระแสไฟให้กับมอเตอร์เป็นช่วงๆ โดยอาศัยกระแสไฟที่ป้อนให้กับมอเตอร์เป็นค่าเฉลี่ยที่เกิดขึ้นในแต่ละช่วง ซึ่งเรียกว่าวิธีการมอดูเลชั่นทางความกว้างของพัลส์ PWM

สเต็ปเปอร์มอเตอร์ เป็นอุปกรณ์ที่สามารถใช้ไอซี MCS-51ควบคุมได้สะดวก และเป็นมอเตอร์ที่เหมาะสมสำหรับใช้งานควบคุมการหมุนโดยต้องการตำแหน่ง และทิศทางที่แบ่งตามลักษณะสายที่ใช้ต่อกับวงจรขับ โดยแบ่งออกได้ 2 แบบคือ แบบไบโพลา และ แบบยูนิโพลา