หน่วยที่ 10 การประยุกต์ใช้งานไอซีไมโครคอนโทรลเลอร์

อดิศักดิ์ ชิณะวงศ์

เอกสารประกอบการเรียนวิชาไมโครคอนโทรลเลอร์

เผยแพร่ที่ www.Adisak51.com

1. อุปกรณ์แสดงผลแบบ LCD

LCD (Liquid Crystal Display) เป็นอุปกรณ์แสดงผลที่ใช้กระแสไฟต่ำ แสดงผลได้ทั้งตัวอักษร ตัวเลข และรูปภาพโดยมีวงจรควบคุมการแสดงผลต่ออยู่ภายใน ช่วยในการใช้งานได้ง่ายขึ้น LCD แบ่งออกเป็น 2 ประเภทใหญ่ ๆ คือ แบบ Dot Matrix (Text) แสดงผลเป็นตัวอักษร และจำนวนบรรทัดแตกต่างกันไปตามรุ่นที่เลือกใช้ ส่วนแบบ Graphic สามารถแสดงผล แบบ Bit-Map สร้างภาพต่าง ๆ ได้ตามต้องการ

1.1 ส่วนประกอบของ LCD

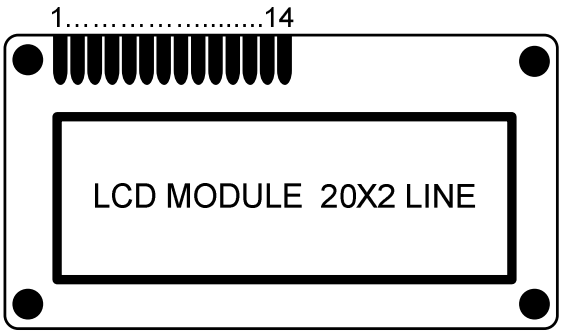
ในโมดูล LCD มีส่วนประกอบหลัก 3 ส่วนดังนี้

1.1.1 ตัวเลขแสดงผล (Display) ภายในเป็นผลึกเหลวสามารถแสดงผลให้เห็นได้โดยอาศัยแสงจากภายนอก ดังนั้นจึงต้องมีมุมในการมองข้อมูลแสดงผลที่ LCD

1.1.2 ตัวควบคุม (Controller) เป็นตัวรับข้อมูลจากอุปกรณ์ภายนอกนำมาควบคุมการทำงานของโมดูล LCD เช่น ลบจอภาพ แสดงตัวอักษรหรือลบเคอร์เซอร์ เป็นต้น ตัวควบคุมจะใช้ ไอซีเฉพาะ เช่น เบอร์ HD44780 จะทำหน้าที่ควบคุม LCD แบบอักขระ ส่วน HD61830 จะควบคุม LCD แบบกราฟิก

1.1.3 ตัวขับ (Driver) เป็นตัวรับสัญญาณจากตัวควบคุม ทำให้มีการแสดงผลข้อมูลตามที่กำหนด ได้แก่ เบอร์ HD44100 และ MSN5259

โมดูล LCD แบบ Dot Matrix ขนาด 20 ตัวอักษร 2 บรรทัด แสดงดังภาพที่ 10.1 ควบคุมโดยการส่งข้อมูลขนาด 8 บิต มีรายละเอียดหน้าที่ของขาสัญญาณ ความสัมพันธ์ในการทำงานของขาสัญญาณแสดงดังตารางที่ 10.1



ภาพที่ 10.1 การจัดขา LCD แบบ Dot Matrix ขนาด 20 ตัวอักษร 2 บรรทัด

ตารางที่ 10.1 รายละเอียดหน้าที่ของขาสัญญาณต่างๆ ของ LCD

ขาที่	สัญญาณ	หน้าที่
1	VSS	GROUND (0V)
2	VCC	+5V Power Supply
3	VEE	LCD Drive Voltage Input
4	RS	“1” Data “0” Instruction
5	R/W	“1” Read “0” Write
6	E	Enable Signal
7	DB0	Data Bus Bit 0
8	DB1	Data Bus Bit 1
9	DB2	Data Bus Bit 2
10	DB3	Data Bus Bit 3
11	DB4	Data Bus Bit 4
12	DB5	Data Bus Bit 5
13	DB6	Data Bus Bit 6
14	DB7	Data Bus Bit 7

RS	R/W	E	การทำงาน
0	0		เขียนคำสั่ง
0	1		อ่านสถานะของโมดูล LCD
1	0		เขียนข้อมูล
1	1		อ่านข้อมูล

1.2 หน้าที่และการใช้งานของขา LCD

ขา 1, 2 ต่อเข้ากับแหล่งจ่ายไฟ +5 โวลต์

ขา 3 ใช้สำหรับปรับความเข้มของ LCD

ขา 4 (RS) เป็นขาสัญญาณอินพุตสำหรับกำหนดว่าข้อมูลที่รับเข้ามาจาก DB0-DB7 เป็นข้อมูล (DATA) หรือเป็นชุดคำสั่ง (Instruction) โดย RS มีลอจิก “1” เป็นข้อมูลสำหรับการแสดงผล RS มีลอจิก “0” เป็นชุดคำสั่งสำหรับควบคุมการทำงานของตัว LCD

ขา 5 (R/W) เป็นขาสัญญาณสำหรับกำหนดการอ่านหรือเขียนข้อมูล ให้กับตัว LCD โดย R/W มีลอจิก “1” หมายถึงตัวควบคุมจากภายนอกต้องการ ข้อมูลที่ถูกแสดงสภาวะของตัว LCD จากขาสัญญาณ DB0-DB7 (LCD สามารถแสดงสถานะของตัวเองให้ตัวควบคุมภายนอกได้) ถ้า R/W มีลอจิก “0” หมายถึงตัวควบคุมจากภายนอกต้องการส่งข้อมูลให้กับ LCD ผ่านทางขาสัญญาณ DB0-DB7 ซึ่งข้อมูลที่ส่งให้ LCD ขึ้นอยู่กับขา RS ด้วยว่าเป็นชุดคำสั่งหรือเป็นข้อมูลสำหรับการแสดงผล

ขา 6 (E) เป็นขาสัญญาณอินพุต กำหนดให้ LCD ตอบสนองกับตัวควบคุมจากภายนอก

ขา 7-14 (DB0-DB7) เป็นขาสัญญาณรับ หรือส่งข้อมูลระหว่าง LCD กับตัวควบคุมภายนอก

1.3 คำสั่งควบคุมการใช้งานของ LCD

การเขียนหรืออ่านข้อมูลกับ LCD เป็นการกำหนดให้ LCD ทำตามเงื่อนไขต่างๆ โดยจะมีชุดคำสั่งให้ LCD แสดงข้อความหรือรูปภาพตามกำหนดได้ดังนี้

1.3.1 คำสั่งเคลียร์ตัวแสดงผล (Clear Display)

เป็นคำสั่งลบข้อมูลต่างๆ มีคำสั่งเป็น 01H โดยการเขียนข้อมูลที่อยู่ใน DDRAM ให้เป็น

ข้อความที่ว่างเปล่า ทำให้บนจอ LCD ถูกลบหายไป และส่งให้ CURSOR ไปอยู่ที่ตำแหน่งแรกของจอ LCD (ตำแหน่งบนซ้ายสุด)

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	0	0	0	1

1.3.2 คำสั่ง Return Home

เป็นคำสั่งส่ง CURSOR ไปตำแหน่งซ้ายสุด ต้องกำหนดให้ DB1 เป็น “1” เป็นคำสั่งกำหนดให้ DDRAM ADDRESS เป็น Ø คือไปอยู่ตำแหน่งแรกสุด และส่งให้ CURSOR ไปอยู่ที่ตำแหน่งซ้ายสุดของจอ LCD โดยข้อมูลบนจอแสดงผลไม่เปลี่ยนแปลง

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	0	0	1	X

1.3.3 คำสั่งเลือกโหมดการป้อนข้อมูล (Entry Mode Set)

เป็นคำสั่งกำหนดตำแหน่งของ CURSOR ในขณะที่ LCD แสดงข้อความ โดยกำหนดสถานะของบิต I/D และ S ก่อนที่มีการเขียนข้อมูลใน DDRAM และต้องไม่ใช้คำสั่ง CLEAR DISPLAY

I/D = “0” ทิศทางของ CURSOR และ ตำแหน่งของ DDRAM เป็นแบบเพิ่มขึ้น

I/D = “1” ทิศทางของ CURSOR และตำแหน่งของ DDROM เป็นแบบลดลง

S = “0” CURSOR จะเลื่อนไปตามทิศทางของการกำหนดค่า I/D

S = “1” CURSOR จะอยู่กับที่ แต่ตัวอักษรจะเลื่อนตามทิศทางของการกำหนด I/D

เช่นคำสั่ง 06H เกิดข้อมูลใหม่ เคอร์เซอร์เลื่อนไปทางขวามือ แอดเดรส DDRAM เพิ่มขึ้น

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	0	1	I/D	S

1.3.4 คำสั่งควบคุมการแสดงผล (Display ON /OFF)

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	1	D	C	B

D = “0” ปิดจอแสดงผล D = “1” เปิดจอแสดงผล

C = “0” ไม่แสดงเคอร์เซอร์ C = “1” เคอร์เซอร์เป็นขีดใต้ตัวอักษร

B = “0” เคอร์เซอร์ไม่มีการพระพริบ B = “1” เคอร์เซอร์กระพริบเป็นรูปสี่เหลี่ยม

เช่นคำสั่ง 0CH กำหนดให้เปิดจอแสดงผล ไม่แสดงเคอร์เซอร์

คำสั่ง 0FH กำหนดให้เปิดจอแสดงผล แสดงเคอร์เซอร์ ให้เคอร์เซอร์กระพริบ

1.3.5 คำสั่งควบคุมการเลื่อนเคอร์เซอร์ และข้อมูลตัวอักษร (Display Shift)

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	1	S/C	R/L	X	X

S/C	R/L	ลักษณะการเลื่อน	ข้อมูลคำสั่ง
0	0	เลื่อนเคอร์เซอร์ไปทางซ้าย 1 ตำแหน่ง	10H - 13H
0	1	เลื่อนเคอร์เซอร์ไปทางขวา 1 ตำแหน่ง	14H - 17H
1	0	เลื่อนตัวอักษรใหม่ไปทางซ้าย 1 ตำแหน่ง	18H – 1BH
1	1	เลื่อนตัวอักษรใหม่ไปทางขวา 1 ตำแหน่ง	1CH – 1FH

1.3.6 คำสั่งกำหนดการใช้งาน (Function Set)

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	1	DL	N	F	X	X

DL = “0” LCD ติดต่อกับตัวควบคุมภายนอกแบบ 4 บิต

DL = “1” LCD ติดต่อกับตัวควบคุมภายนอกแบบ 8 บิต

N = “0” กำหนดจำนวนบรรทัดแสดงผล 1 บรรทัด

N = “1” กำหนดจำนวนบรรทัดแสดงผล 2 บรรทัด

F = “0” แสดงอักษรแบบ 5X7 Dots

F = “1” แสดงอักษรแบบ 5X10 Dots

เช่นคำสั่ง 38H กำหนดให้ทำงาน 8 บิต แสดงผล 2 บรรทัด ความละเอียด 5x7

1.3.7 คำสั่งเลือกแอดเดรสของ CGRAM

ก่อนการอ่านหรือเขียนข้อมูลให้ CGRAM ต้องกำหนดแอดเดรสให้บิตที่ 7 เป็น “0” บิตที่ 6 เป็น “1” ส่วน 6 บิตที่เหลือแทนด้วยค่าแอดเดรสของ CGRAM และแอดเดรสของ CGRAM จะอยู่ระหว่าง 00-3FH

1.3.8 คำสั่งเลือกแอดเดรสของ DD RAM

ใช้ในการเลือกแอดเดรสของ DDRAM ก่อนทำการอ่านหรือเขียนข้อมูล บิตที่ 7 ต้องเป็น “1” และข้อมูลอีก 7 บิตที่เหลือเป็นค่าแอดเดรสของ DDRAM ซึ่งอยู่ระหว่าง 8CH-0FFH จำนวนแอดเดรสยังขึ้นอยู่กับการกำหนดสถานะที่บิต N ของคำสั่ง Function Set หากบิต N เป็น “0” แอดเดรสของ DDRAM อยู่ระหว่าง 80H-0CH และถ้าบิต N เป็น “1” แอดเดรสของ DDRAM จะมี 2 ช่วงคือ 8CH-87H และ 0CH-0C7H

1.3.9 คำสั่งอ่านค่าแฟลก BUSY และแอดเดรส

เป็นคำสั่งในการใช้อ่านแฟลก BUSY (BF) เป็นตัวบอกสถานการณ์ควบคุม LCD ว่าพร้อมรับข้อมูลอยู่หรือไม่

BF = “0” LCD พร้อมรับข้อมูลคำสั่ง

BF = “1” LCD ยังไม่พร้อมรับข้อมูลคำสั่ง

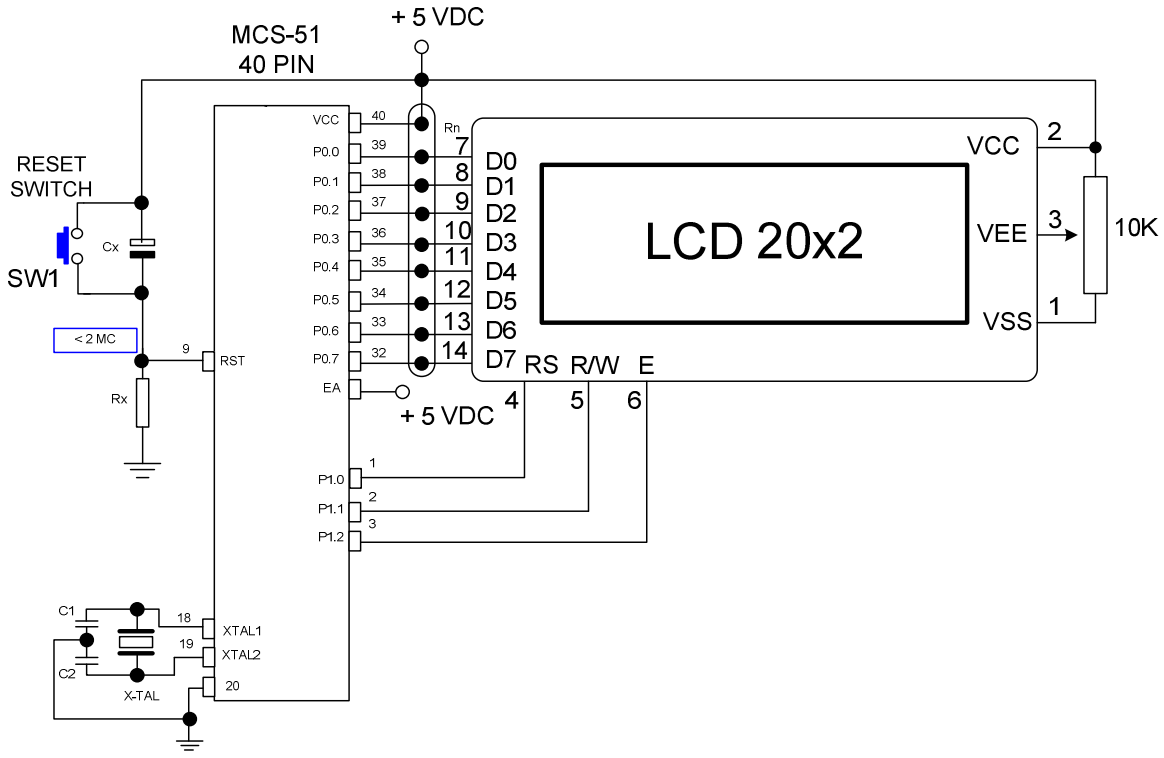
เมื่อต้องการอ่านแฟลกต้องกำหนดให้ขา R/W เป็น “1” RS = “0” ยังใช้เป็นคำสั่งสั่งอ่านข้อมูลแอดเดรสของ CGRAM และ DDRAM ด้วย โดยบิตที่ 0 - บิตที่ 6 เป็นค่าข้อมูลของแอดเดรสที่ต้องการอ่าน

RS	R/W
0	1

DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
BF	A	A	A	A	A	A	A

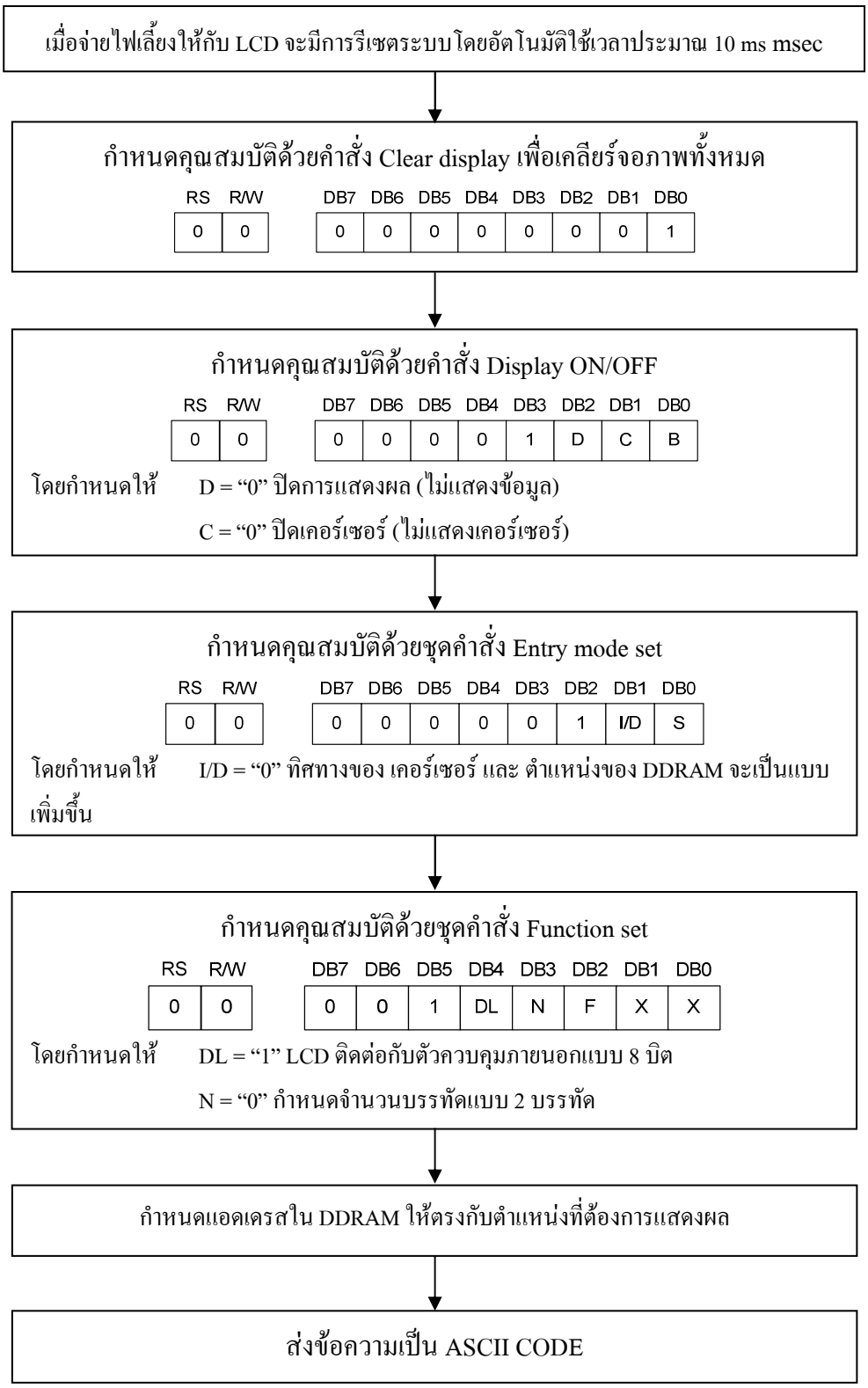
1.4 การใช้งาน LCD กับไอซี MCS-51

การต่อ LCD ใช้งานร่วมกับไอซี MCS-51 วงจรแสดงดังภาพที่ 10.2



ภาพที่ 10.2 การต่อ LCD กับไอซี MCS-51

ลำดับโปรแกรมคำสั่งติดต่อกับ LCD เขียนเป็นผังงานแสดงดังภาพที่ 10.3 ดังต่อไปนี้



ภาพที่ 10.3 แสดงลำดับคำสั่งการติดต่อกับ LCD

1.5 การเขียนโปรแกรมใช้งาน LCD

ตัวอย่างที่ 1 การกำหนดค่าการควบคุม LCD พื้นฐาน โดยกำหนดให้แถวที่ 1 หลักที่ 1 แสดงตัวอักษร “H” แถวที่ 2 หลักที่ 1 แสดง ตัวอักษร “I”

```
                ORG     0000H
RS              BIT             P1.0
EN              BIT             P1.2
R_W             BIT             P1.1
LCD_DATA        EQU             P0
; ************ LCD Initialize **************
INIL_CD:        MOV     A, #01H         ; เคลียร์การแสดงผล
                ACALL WR_INT            ; เรียกโปรแกรมย่อยเขียนคำสั่ง
                ACALL DELAY             ; หน่วงเวลาให้ LCDทำงาน
                MOV     A, #0EH         ; LCD แสดงผล เคอร์เซอร์เป็นขีดใต้ตัวอักษร และไม่กระพริบ
                ACALL WR_INT            ; เรียกโปรแกรมย่อยเขียนคำสั่ง
                ACALL DELAY
                MOV     A, #06H         ; (I/D = 1, S=0) เกิดข้อมูลใหม่เคอร์เซอร์เลื่อนไปทางขวา
                ACALL WR_INT            ; เรียกโปรแกรมย่อยเขียนคำสั่ง
                ACALL DELAY
                MOV     A, #38H         ; ติดต่อ LCD 8 บิต 1 บรรทัด แสดงผล 5x7 จุด (DL =1, N=1, F=0)
                ACALL WR_INT            ; เรียกโปรแกรมย่อยเขียนคำสั่ง
                ACALL DELAY
ROW1:           MOV     A, #08H         ; กำหนดแอดเดรส DD RAM แถวที่ 1 หลักที่ 1
                ACALL WR_INT            ;
                ACALL DELAY
                MOV     A, #48H         ; โหลดค่า ASCII Code “H”
                ACALL WR_DATA           ; เขียนข้อมูลลงใน DD RAM ที่แอดเดรส 08H
                ACALL   DELAY
ROW2:           MOV      A, #0C0H       ; กำหนดแอดเดรส DD RAM แถวที่ 2 หลักที่ 1
                ACALL   WR_INT          ; เรียกโปรแกรมย่อยเขียนคำสั่ง
                ACALL   DELAY
                MOV     A, #49H         ; โหลดค่า ASCII Code “I” กำหนดตำแหน่งแรก
```

```
                ACALL   WR_DATA       ; เขียนข้อมูลลงใน DD RAM ที่แอดเดรส 80H
                ACALL   DELAY
                AJMP    $
; ************ โปรแกรมย่อยเขียนคำสั่งที่ พอร์ต A **************
WR_INT:         CLR     R_W           ; กำหนดเป็นการเขียนข้อมูลให้กับ LCD
                SETB    EN            ; ให้ EN เป็นสถานะลอจิกสูง
                CLR     RS            ; RS = "0" เขียนคำสั่ง
                ACALL   DELAY         ; หน่วงเวลาให้ LCD ทำงาน
                MOV     LCD_DATA, A ; ส่งคำสั่งไปยัง LCD ที่ขา DB0 - DB7
                CLR     EN            ; EN เปลี่ยน "1" → "0" กำหนดการเขียน LCD
                ACALL   DELAY         ; หน่วงเวลาให้ LCD ทำงาน
                SETB    EN            ; EN เปลี่ยน "0" → "1"
                ACALL   DELAY         ; หน่วงเวลา
                RET
; ******* โปรแกรมย่อยเขียนข้อมูลลงใน DD RAM *********
WR_DATA:        CLR     R_W
                SETB    EN
                SETB    RS            ; RS = "1" เขียนข้อมูล
                ACALL   DELAY
                MOV     LCD_DATA, A ; ส่งคำสั่งไปยัง LCD
                CLR     EN            ; กำหนดใช้งาน LCD
                ACALL   DELAY
                SETB    EN
                RET
DELAY:          MOV     R0, #0AH      ; 10 ms
DELAY1:         MOV     R1, #0E6H     ; 1 ms per loop
DELAY2:         NOP
                NOP
                DJNZ    R1, DELAY2
\               DJNZ    R0, DELAY1
                RET
```

ตัวอย่างที่ 2 เขียนโปรแกรมแสดงผลที่ LCD ขนาด 20 x 2 ตัวอักษร โดยแสดงข้อความแบบ 2 บรรทัด ในบรรทัดแรกให้แสดงข้อความ MICROCONTROLLER และบรรทัดที่ 2 ให้แสดงข้อความ MCS-51

```
                ORG 0000H
RS              BIT             P1.0
R_W             BIT             P1.1
EN              BIT             P1.2
LCD_DATA        EQU             P0
; ************ LCD Initialize **************
INI_LCD:        MOV     A, #01H         ; <----------------------------------------------------<
                ACALL WR_INT            ;                       เคลียร์การแสดงผล
                ACALL DELAY             ; <----------------------------------------------------<
                MOV     A, #0EH         ; <------------------------------------------------------------<
                ACALL WR_INT    ; แสดงผล เคอร์เซอร์เป็นขีดใต้ตัวอักษร และไม่กระพริบ (I/D = 1,C=1,B=0)
                ACALL DELAY             ; <------------------------------------------------------------<
                MOV     A, #06H         ; <----------------------------------------------------<
                ACALL WR_INT            ; ** กำหนดลักษณะการแสดงผล (I/D = 1, S=0) **
                ACALL DELAY             ; <----------------------------------------------------<
                 MOV    A, #38H         ; <------------------------------------------------------------<
                ACALL WR_INT            ; เลือกติดต่อ LCD แบบ 8 บิต แสดงผล 5x7 จุด (DL =1, N=1, F=0)
                ACALL DELAY             ; <------------------------------------------------------------<
                MOV     DPTR, #TABLE
LCD_DIS:        MOV     A, #08H         ; กำหนดแอดเดรส DD RAM (แถวที่ 1, หลักที่ 1)
                ACALL WR_INT            ;
                MOV     R6, #20
                MOV     R7, 00
ROW1:           MOV     A, R7
                MOVC  A,@A+DPTR
                ACALL WR_DATA
                ACALL   DELAY
                INC     R7
```

```
                DJNZ    R6, ROW1
                MOV     A, #0C0H        ; กำหนดแอดเดรส DD RAM (แถวที่ 2 , หลักที่ 1)
                ACALL WR_INT
                MOV     R6, #20
ROW2:           MOV     A, R7
                MOVC  A,@A+DPTR
                ACALL WR_DATA
                ACALL   DELAY
                INC     R7
                DJNZ    R6, ROW2
                AJMP    $
; ************ โปรแกรมย่อยเขียนคำสั่งที่ พอร์ต A **************
WR_INT:         CLR     R_W
                SETB    EN
                CLR     RS              ; RS = “0” เขียนคำสั่ง
                ACALL DELAY
                MOV     LCD_DATA, A ; ส่งคำสั่งไปยัง LCD
                CLR     EN              ; กำหนดใช้งาน LCD
                ACALL DELAY
                SETB    EN
                ACALL DELAY
                RET
; ******* โปรแกรมย่อยเขียนข้อมูลลงใน DD RAM *********
WR_DATA:        CLR     R_W
                SETB    EN
                SETB    RS              ; RS = “1” เขียนข้อมูล
                ACALL DELAY
                MOV     LCD_DATA, A ; ส่งคำสั่งไปยัง LCD
                CLR     EN              ; กำหนดใช้งาน LCD
                ACALL DELAY
                SETB    EN
```

```
                RET
; ******* DELAY TIME 10 ms *********
DELAY:          MOV   R0, #15H        ; 10 ms
DELAY1:         MOV   R1, #0E6H       ; 1 ms per loop
DELAY2:         NOP
                NOP
                DJNZ  R1, DELAY2
\               DJNZ  R0, DELAY1
                RET
TABLE:          DB    "  MICROCONTROLLER  "
                DB    "        MCS-51        "
                END
```

2. บัสแบบ I^2C

2.1 บัสอนุกรมแบบ I^2C

I^2C BUS ย่อมาจาก Inter-Integrated Circuit Bus หมายถึง เป็นการสื่อสารอนุกรม แบบซิงโครนัส (Synchronous) เพื่อใช้ติดต่อสื่อสารระหว่างไมโครคอนโทรลเลอร์ กับอุปกรณ์ภายนอก ซึ่งถูกพัฒนาขึ้นโดยบริษัท Philips Semiconductors เพื่อต้องการให้ไอซีหรือโมดูลสามารถติดต่อ สั่งงาน และควบคุมโดยใช้สายสัญญาณเพียง 2 เส้น คือ สายรับส่งข้อมูล และ สายสัญญาณนาฬิกากำหนดจังหวะการทำงาน ส่วนการต่อใช้งานร่วมกันของอุปกรณ์บนบัสจะต่อสายข้อมูล และสายสัญญาณนาฬิกา ของอุปกรณ์แต่ละตัวขนานกันไป การกำหนดแอดเดรสของแต่ละตัว จะใช้รหัสข้อมูล และการกำหนดสภาวะลอจิกเพื่อขอแอดเดรสของอุปกรณ์แต่ละตัว สายข้อมูลบนบัส I^2C เป็นสายข้อมูลอนุกรมหรือ SDA (Serial Data Line) ส่วนสายสัญญาณนาฬิกา จะเป็นแบบอนุกรมหรือ SCL (Serial Clock Line)

อุปกรณ์ใช้เชื่อมต่อบนบัสแบบ I^2C เช่นไอซีขยายพอร์ตอินพุตเอาต์พุต (I/O Expander) ไอซีแปลงสัญญาณอนาล็อกเป็นดิจิตอล (ADC) และแปลงสัญญาณดิจิตอลเป็นอนาล็อก (DAC) ไอซีขับโมดูล LCD ไอซีเรียวไทม์คล็อก (RTC) หน่วยความจำอีอีพรอม และไมโครคอนโทรลเลอร์

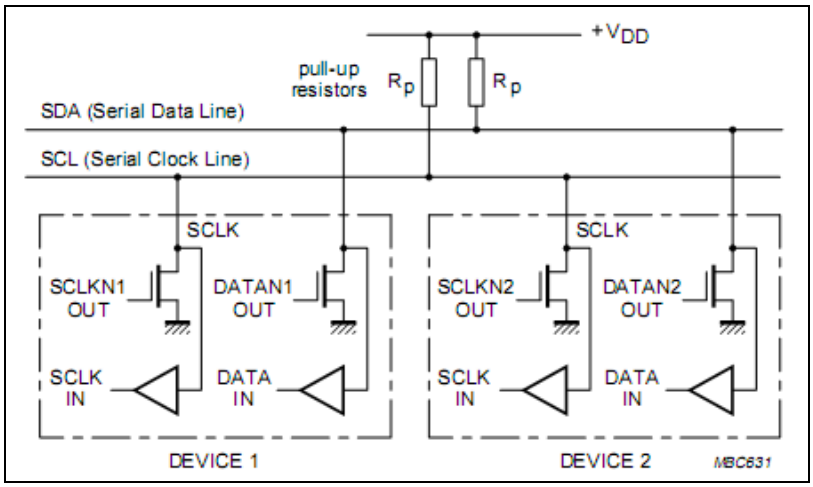
2.2 คุณสมบัติทั่วไปของบัส I^2C

สาย SDA และ SCL เป็นสายสัญญาณ 2 ทิศทาง (Bi-Directional Line) ต่ออยู่กับแหล่งจ่ายไฟบวกโดยมีการต่อตัวต้านทานพูลอัปกับแรงดัน +5โวลต์ เมื่อบัสเป็นอิสระโดยไม่มีการติดต่อใช้งาน ทั้ง SDA และ SCL จะมีสถานะลอจิกสูง (High Impedance) และยังช่วยในการป้องกันสัญญาณรบกวนที่อาจมี

เข้ามาในสายสัญญาณทั้งสอง วงจรเอาต์พุต ของอุปกรณ์ที่ต่ออยู่บนบัส I^2C ต้องมีลักษณะเป็นวงจรเดรนเปิด (Open-Drain) หรือคอลเล็กเตอร์เปิด (Open-Collector) แสดงดังภาพที่ 10.4

อัตราการถ่ายเทข้อมูลบนบัส I^2C สูงถึง 100 กิโลบิตต่อวินาทีในโหมดมาตรฐาน (Standard Mode) และสูงถึง 400 กิโลบิตต่อวินาทีในโหมดความเร็วสูง (Fast Mode) อุปกรณ์ที่ต่อร่วมอยู่บนบัส I^2C จะต้อง มีค่าความจุไฟฟ้ารวมที่เกิดขึ้นระหว่างสาย SDA และ SCL ไม่เกิน 400 pF การเข้าถึงอุปกรณ์บนบัส I^2C ใช้ข้อมูลสำหรับการเข้าถึง 2 ค่าคือ 7 บิต (7-Bit Addressing) หรือ 10 บิต (10- Bit Addressing)

บัส I^2C คือ สามารถเชื่อมต่ออุปกรณ์ที่ใช้ไฟเลี้ยงไม่เท่ากันให้สามารถติดต่อสื่อสารกันได้ โดยอุปกรณ์บนบัส I^2C ตัวหนึ่งอาจใช้ไฟเลี้ยง +5โวลต์ในขณะที่อีกตัวหนึ่งใช้ไฟเลี้ยง +12โวลต์ การต่อสาย SDA และ SCL ของอุปกรณ์แต่ละตัวเข้าด้วยกัน และต่อตัวต้านทานพูลอัป (Rp) เข้ากับแรงดันไฟ +5โวลต์



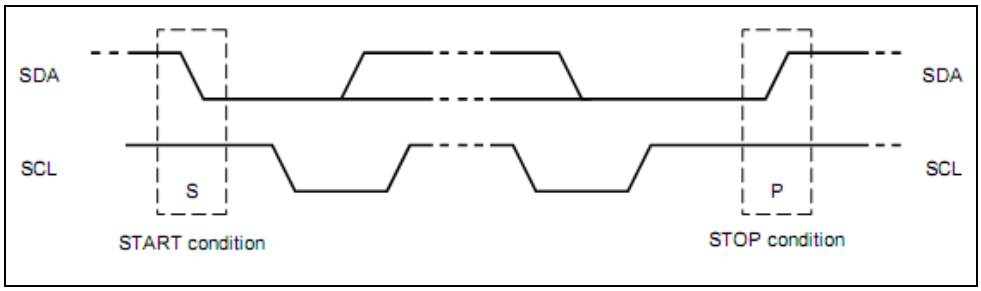
ภาพที่ 10.4 การต่ออุปกรณ์เข้ากับ I^2C บัส

(แหล่งอ้างอิง http://i2c2p.twibright.com/spec/i2c.pdf)

บัส I^2C เป็นการกำหนดรูปแบบของการติดต่อ หรือโปรโตคอล (Protocols) เพื่อกำหนดการติดต่อ การรับ และการส่งของอุปกรณ์แต่ละตัว

2.3 สภาวะที่เกิดขึ้นบนบัส I^2C

กำหนดชื่อเรียกของอุปกรณ์ โดยอุปกรณ์ทำหน้าที่กำเนิดสัญญาณนาฬิกาการติดต่อบนบัส I^2C เรียกว่า มาสเตอร์ (Master) สามารถเป็นได้ทั้งตัวรับและตัวส่ง อุปกรณ์ที่ถูกระบุแอดเดรสโดยมาสเตอร์หรือต่อพ่วงเข้าไปบนบัส I^2C เรียกว่า สเลฟ (Slave) อุปกรณ์ที่เป็นตัวสร้างสัญญาณการส่งข้อมูลเรียกว่า ตัวส่ง (Transmitter) อุปกรณ์ที่รับข้อมูล เรียกว่า ตัวรับ (Receiver) อุปกรณ์บนบัส I^2C สามารถเป็นได้ทั้งตัวรับ และส่ง อุปกรณ์บางตัวทำหน้าที่เป็นตัวรับเพียงอย่างเดียว แต่ไม่มีอุปกรณ์ใดบนบัส I^2C ทำหน้าที่เป็นตัวส่งเพียงอย่างเดียว การสื่อสารแบบ I^2C มีการทำงาน 5 สภาวะ



ภาพที่ 10.5 แสดงสภาวะ เริ่มต้นถ่ายเทข้อมูล และหยุดการถ่ายเทข้อมูล

2.3.1 บัสว่าง (Bus Not Busy) สภาวะนี้เกิดขึ้นเมื่อลอจิกบนสาย SDA และ SCL เป็นลอจิก “1” ทั้งคู่

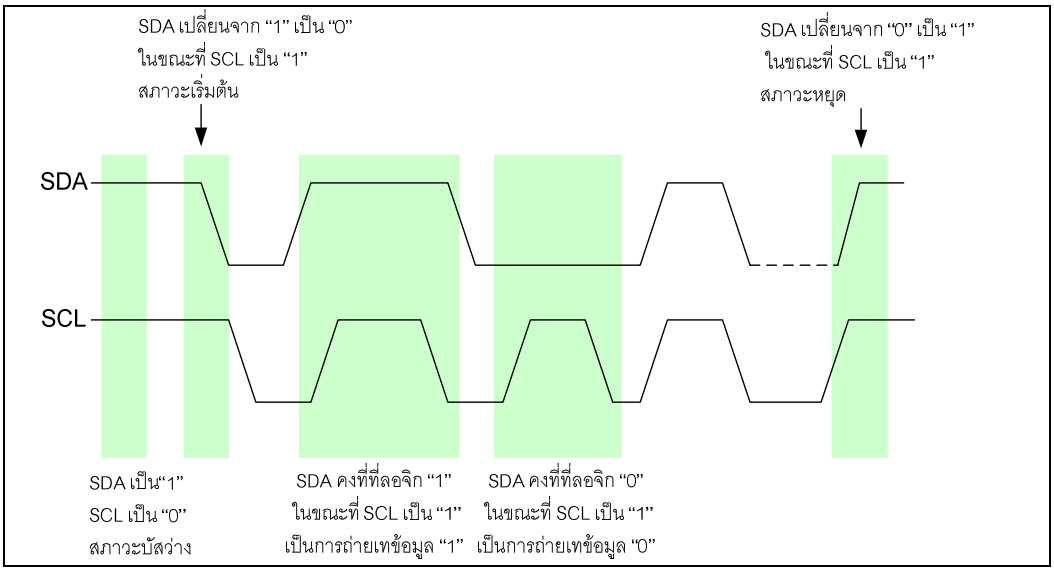
2.3.2 เริ่มต้นการถ่ายเทข้อมูล (Start Data Transfer) เกิดขึ้นเมื่อสาย SDA มีการเปลี่ยนแปลงระดับลอจิก “1” ไป “0” ในขณะที่สาย SCL มีสถานะลอจิก“1” เรียกว่า สภาวะเริ่มต้น (START) แสดงดังภาพที่ 10.5

2.3.3 หยุดการถ่ายเทข้อมูล (Stop Data Transfer) เกิดขึ้นเมื่อสาย SDA มีการเปลี่ยนแปลงระดับลอจิก “0”ไป “1”ในขณะที่สาย SCL มีสถานะลอจิกสูง เรียกสภาวะนี้ว่า สภาวะหยุด (STOP) แสดงดังภาพที่ 10.5

2.3.4 ข้อมูลดำรงอยู่บนบัส (Data Valid) สภาวะนี้เกิดขึ้นถัดจากสภาวะเริ่มต้น โดยสถานะลอจิกที่เกิดขึ้นบนสาย SDA คือข้อมูลที่ทำการถ่ายเท เมื่อสาย SCL เป็นลอจิกสูง สถานะที่สาย SDA ต้องคงที่ เพื่อให้อุปกรณ์ตอบรับรู้ข้อมูลในจังหวะนั้นว่า เป็น “0” หรือ “1” ข้อมูลอาจเกิดการเปลี่ยนแปลงได้ในขณะที่สาย SCL เป็นลอจิกต่ำ แต่เมื่อต้องการให้เกิดการถ่ายเทข้อมูลอย่างสมบูรณ์ สถานะลอจิกที่ขา SDA ต้องคงที่ตลอดช่วงเวลาที่สาย SCL มีสถานะลอจิกสูง หากมีการเปลี่ยนแปลงสถานะลอจิกในขณะที่สาย SCL มีลอจิกสูงอยู่นั้น อุปกรณ์มาสเตอร์ที่ควบคุมการถ่ายเทข้อมูล จะแปลความหมายเป็นสภาวะหยุดหรือสภาวะเริ่มต้นได้ ทำให้ข้อมูลที่ถ่ายเทเกิดความผิดพลาด

2.3.5 การตอบรับ (Acknowledge) เกิดขึ้นหลังจากถ่ายเทข้อมูลจากตัวส่งมายังตัวรับเกิดขึ้นอย่างสมบูรณ์ โดยตัวส่งทำการส่งข้อมูลมา 1 บิต เรียกว่า บิตตอบรับ (Acknowledge Bit) มีสถานะเป็นลอจิกสูง หลังจากส่งข้อมูลจนครบ ส่วนอุปกรณ์มาสเตอร์จะทำการส่งสัญญาณการตอบรับ ซึ่งสัมพันธ์กับสัญญาณนาฬิกาเพื่อตอบสนองบิตตอบรับที่ส่งมาจากตัวส่ง ทางด้านตัวรับจะส่งบิตตอบรับ ที่มีสถานะลอจิกต่ำลงบนบัส อุปกรณ์สเลฟที่ถูกอ้างถึงในการติดต่อหรือกำลังติดต่ออยู่ในขณะนั้นจะกำหนดบิตตอบรับเพื่อตอบสนองให้ทราบว่าได้รับข้อมูลในแต่ละไบต์เรียบร้อยแล้ว

จากการทำงานทั้ง 5 สภาวะของการติดต่อสื่อสารข้อมูล สามารถแสดงเป็นไดอะแกรมเวลาที่แสดงถึงสภาวะต่างๆ บนบัส I^2C แสดงดังภาพที่ 10.6

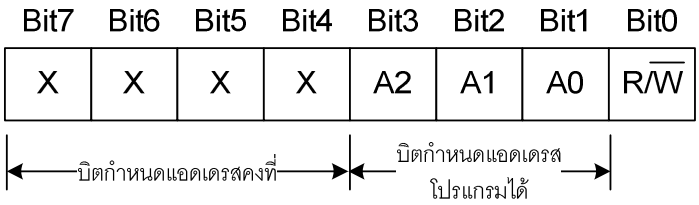


ภาพที่ 10.6 แสดงไดอะแกรมเวลาที่แสดงถึงสภาวะต่างๆ บนบัส I^2C

2.4 การทำงานบนบัส I^2C

การอ้างถึงอุปกรณ์บนบัส I^2C สามารถอ้างถึงแบบ 7 บิตหรือ 10 บิตได้ โดยกรณีที่มีอุปกรณ์ต่อบนบัสไม่มากจะใช้การอ้างถึงแบบ 7 บิต แต่ถ้ามีอุปกรณ์ต่อบนบัสมากกว่า 127 แอดเดรส ต้องมีการอ้างถึงแบบ 10 บิต หลังจากที่ติดต่ออุปกรณ์แต่ละตัวแล้ว จะเริ่มต้นการถ่ายเทข้อมูลต่อไป

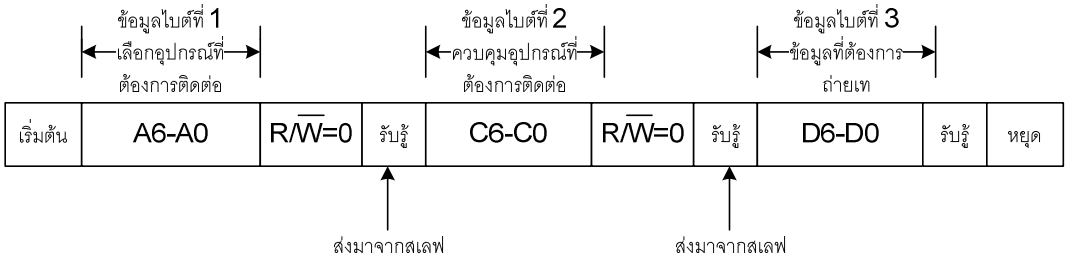
การอ้างถึงแบบ 7 บิต (7-Bit Addressing) ข้อมูลไบต์แรกที่เกิดขึ้น หลังสภาวะเริ่มต้นคือข้อมูลอ้างถึงอุปกรณ์ที่ต้องการติดต่อ หรือ ข้อมูลกำหนดแอดเดรส โดยมีรูปแบบแสดงดังภาพที่ 10.7 ใน 7 บิตบนรวมทั้งบิต MBS จะเป็นข้อมูลแอดเดรสของอุปกรณ์สเลฟที่ต้องการติดต่อ โดยแบ่งเป็น บิตกำหนดแอดเดรสคงที่ (Fixed Address Bit) จำนวน 4 บิต ซึ่งข้อมูลนี้อุปกรณ์แต่ละตัวจะถูกกำหนดมาจากผู้ผลิตไม่สามารถเปลี่ยนแปลงแก้ไขได้ ถัดมาอีก 3 บิตเป็นบิตที่กำหนดแอดเดรสโดยสามารถโปรแกรมได้ (Programmable Address Bit) โดยผู้ใช้งานต้องกำหนดสถานะลอจิกให้แก่ขา A0-A2 ของอุปกรณ์ที่มีการเชื่อมต่อแบบบัส I^2C ส่วนในบิต LSB เป็นบิตที่ใช้กำหนดการอ่านหรือเขียนข้อมูลกับอุปกรณ์สเลฟ ตัวนั้นๆ หากบิต LSB เป็น "0" หมายถึงต้องเขียนข้อมูลไปยังอุปกรณ์นั้นถ้าเป็น "1" จะเป็นการอ่านข้อมูลจากอุปกรณ์สเลฟ



ภาพที่ 10.7 แสดงการอ้างแอดเดรสขนาด 7 บิต

ข้อมูลในไบต์ต่อมาคือ ข้อมูลควบคุม (Control Byte) ซึ่งในอุปกรณ์แต่ละตัว จะมีการกำหนดข้อมูลควบคุมที่แตกต่างกันออกไป เช่น ไอซีขยายพอร์ตมีข้อมูลควบคุมกำหนดให้แต่ละบิตเป็นอินพุต หรือเป็นเอาต์พุต หรือไอซี ADC/DAC ต้องการข้อมูลให้ทำงานเป็นวงจร ADC หรือ DAC เป็นต้น

ข้อมูลในไบต์ต่อมาคือ ข้อมูลที่ทำการถ่ายเทจริง (Data) หลังจากมีการถ่ายเทข้อมูลในแต่ละไบต์ อุปกรณ์สเลฟที่ได้รับการติดต่อ ต้องส่งสัญญาณรับรู้ตอบกลับมาด้วยทุกครั้ง เพื่อให้กระบวนการถ่ายเทข้อมูลสามารถดำเนินต่อไป แสดงได้ดังภาพที่ 10.8



ภาพที่ 10.8 แสดงการติดต่อแบบ I^2C บัส ใช้ถ่ายเทข้อมูลเมื่ออ้างแอดเดรสขนาด 7 บิต

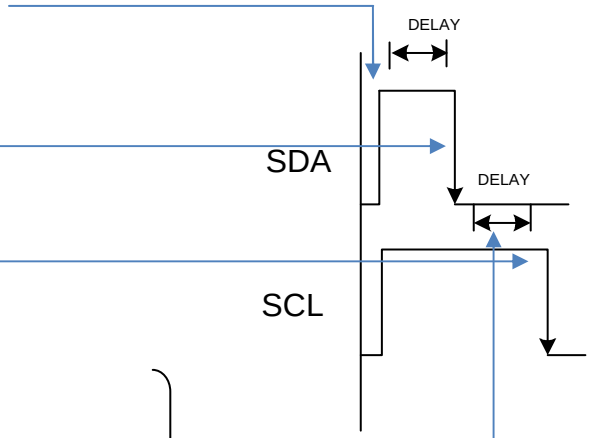
การอ้างถึงแบบ 10 บิต (10-Bit Addressing) ใช้รูปแบบข้อมูลอนุกรมแบบ 7 บิต โดยในข้อมูลไบต์แรกหลังจากเกิดสภาวะเริ่มต้น ต้องกำหนดให้ 5 บิต บนมีข้อมูลเป็น 11110 ส่วนอีก 2 บิตถัดมาเป็นบิตแอดเดรส ของอุปกรณ์ที่ต้องการติดต่อในบิต LSB ของข้อมูลไบต์แรกยังคงเป็นการกำหนดว่าต้องการอ่านหรือเขียนข้อมูลกับอุปกรณ์ สเลฟตัวที่ต้องการติดต่อด้วย ข้อมูลไบต์ต่อมาเป็นข้อมูลแอดเดรส ใบไบต์ที่สอง ของอุปกรณ์ที่ต้องการติดต่อด้วย ข้อมูลไบต์ถัดไปจึงเป็นข้อมูลควบคุม ข้อมูลหลังจากนั้น เป็นข้อมูลจริงที่ใช้ในการติดต่อ

2.5 โปรแกรมสร้างสัญญาณติดต่อกับบัส I^2C

การสร้างสภาวะมาตรฐานของบัส I^2C อันประกอบด้วย สภาวะเริ่มต้น, สภาวะสิ้นสุดการส่งข้อมูล, สภาวะหยุด, สัญญาณนาฬิกาบนขา SCL

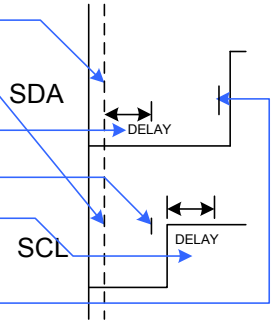
2.5.1 การสร้างสภาวะเริ่มต้นเมื่อต้องการติดต่อกับบัส I^2C จะต้องทำให้บัสว่างโดยกำหนดที่ขา SCL และ ขา SDA มีลอจิกเป็น “1” ทั้งคู่ จากนั้นทำให้ขา SDA มีลอจิก “0” โดยที่ขา SCL ยังคงเป็นลอจิก “1” อยู่ ในช่วงเวลาหนึ่ง และกำหนดให้ขา SCL มีลอจิกเป็น “0” ทำให้ SCL และ SDA มีลอจิกเป็น “0” ทั้งคู่พร้อมติดต่อ สามารถเขียนเป็นโปรแกรมเพื่อสร้างสัญญาณดังนี้

```
I2C_START_:  SETB   SCL
             SETB   SDA
             ACALL I2C_DELAY
             CLR    SDA
             ACALL I2C_DELAY
             CLR    SCL
             RET
I2C_DELAY:   MOV    R6, #0CH
I2C_DELAY1:  NOP
             NOP
             DJNZ   R6, #IC2_DELAY1
             RET
```



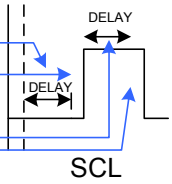
2.5.2 การสร้างสภาวะหยุด เมื่อจะหยุดส่งข้อมูล ต้องส่งสภาวะหยุดออกไป โดยกำหนดให้ขา SCL และ SDA เป็นลอจิก “0” ทั้งคู่ ต่อมาจึงกำหนดให้ขา SCL มีลอจิกเป็น “1” โดย SDA ยังมีลอจิกเป็น “0” จากนั้น จึงทำให้ขา SDA มีลอจิกเป็น “1” ซึ่งจะทำให้ระบบบัสกลับเข้าสู่บัสว่างอีกครั้ง พร้อมที่ รับหรือส่งข้อมูลต่อไป

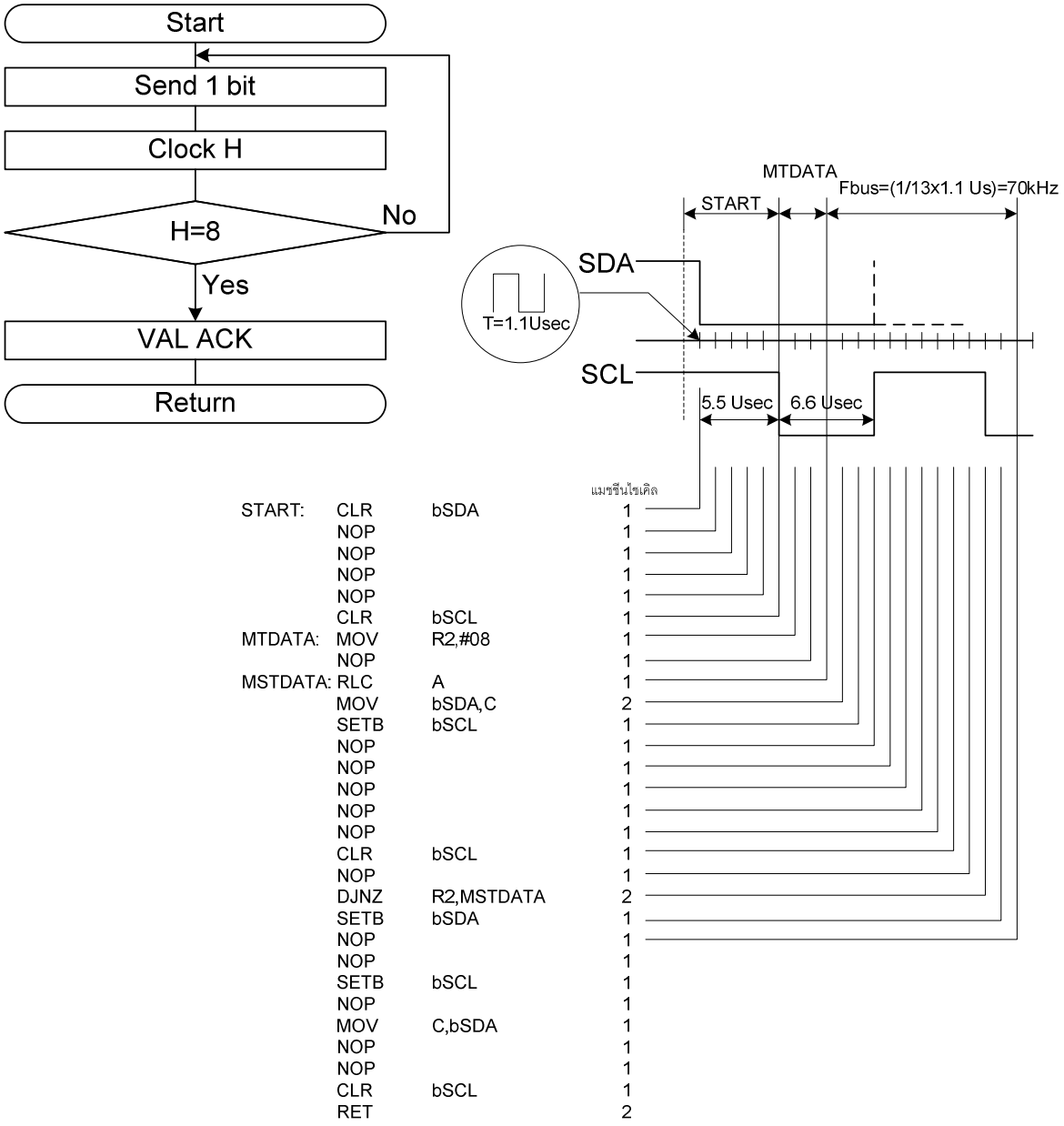
```
I2C_STOP:    CLR    SDA
             CLR    SCL
             ACALL I2C_DELAY
             SETB   SCL
             ACALL I2C_DELAY
             SETB   SDA
             RET
```



2.5.3 การสร้างโปรแกรมย่อยกำเนิดสัญญาณนาฬิกา

```
I2C_STOP:    ACALL I2C_DELAY
             SETB   SCL
             ACALL I2C_DELAY
             CLR    SCL
             RET
```





ภาพที่ 10.9 การส่งข้อมูลในแต่ละบิต กำหนดโดยโปรแกรมคำสั่ง

การส่งข้อมูลในแต่ละบิต กำหนดได้จากโปรแกรมคำสั่งแสดงดังภาพที่ 10.9 โดยเขียนข้อมูลไปยังอุปกรณ์สเลฟ ข้อมูลที่ส่ง จะส่งไปทางขา SDA โดยจะกำหนดที่รีจิสเตอร์ A แล้วทำการส่งออกไปยังแฟลกทด โดยการใช้คำสั่งหมุนข้อมูล (RLCA) เพื่อถ่ายเทไปยังขา SDA ต่อไป

การส่งข้อมูลลอจิก “0” ดำเนินการตามขั้นตอนดังนี้

ขั้นตอนที่ 1 ให้ขา SDA เป็น “0” สำหรับการส่งข้อมูลลอจิก “0”

ขั้นตอนที่ 2 ให้ขา SCL “1” สำหรับการป้อนสัญญาณนาฬิกา

ขั้นตอนที่ 3 ในขณะที่ขา SDA ยังคงเป็น “0” ให้ขา SCL กลับมาเป็นสถานะลอจิก “0” เหมือนเดิม

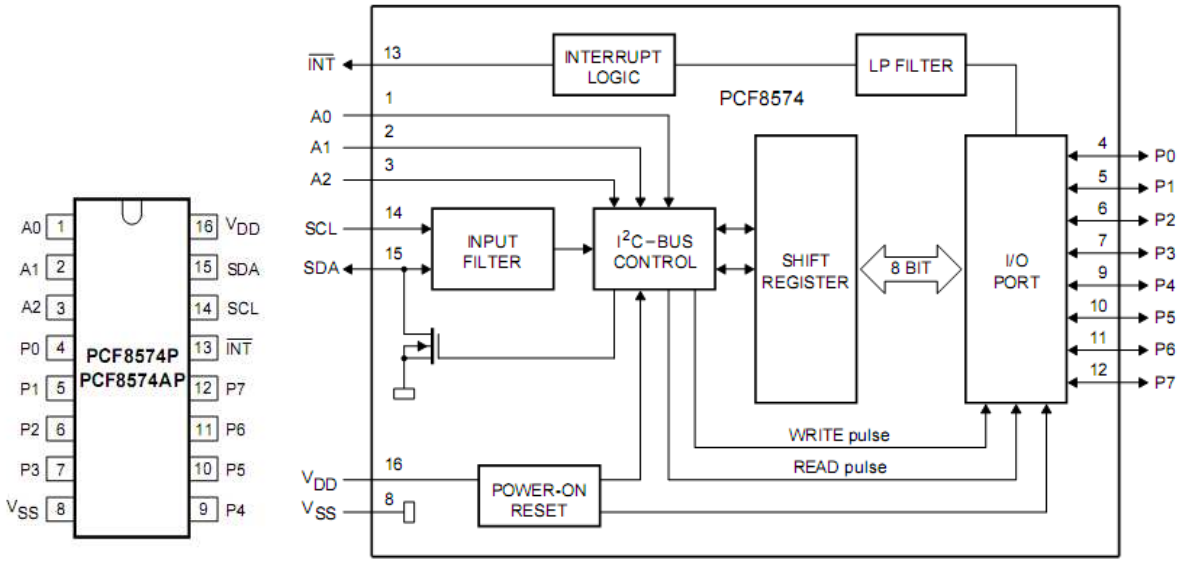
การส่งข้อมูลลอจิก “1” ดำเนินการตามขั้นตอนดังนี้

ขั้นตอนที่ 1 ทำให้ขา SDA มีลอจิกเป็น “1” สำหรับการส่งข้อมูลลอจิก “1”

ขั้นตอนที่ 2 ให้ขา SCL เป็น “1” สำหรับการส่งสัญญาณนาฬิกา ในขณะที่ขา SDA ยังคงเป็น “1”อยู่

ขั้นตอนที่ 3 จากนั้นทำให้ขา SCL กลับมาเป็นสถานะเป็นลอจิก “0” เหมือนเดิม

2.6 ไอซีขยายพอร์ต PCF8574A



ภาพที่ 10.10 แสดงการจัดขา และบล็อกไดอะแกรมของไอซีขยายพอร์ต PCF8574A

2.6.1 ไอซีขยายพอร์ต PCF8574A มีคุณสมบัติดังนี้

1) ทำงานที่ระดับแรงดันตั้งแต่ 2.5V ถึง 6V
2) กินกระแสในสภาวะสแตนด์บายต่ำเพียง 10 μA
3) เอาต์พุตแลตช์ค่าได้สามารถจ่ายกระแสได้สูงเพื่อขับ LED ได้โดยตรง
4) กำหนดตำแหน่งแอดเดรสได้ 8 ค่าทางฮาร์ดแวร์ ที่ขา A0-A2 ต่อใช้งานได้ถึง 8 ตัว

การจัดขาและบล็อกไดอะแกรมของไอซี PCF8574A แสดงในภาพที่ 10.10 ขาของ PCF8574A สามารถกำหนดให้เป็นพอร์ตอินพุต หรือเอาต์พุตได้โดยอิสระ ไม่ต้องใช้คำสั่งควบคุมเพื่อเลือกเป็นเอาต์พุตหรืออินพุต เมื่อจ่ายไฟให้กับ PCF8574A ครั้งแรก ขาพอร์ตทั้ง 8 ขาจะมีลอจิกเป็น “1” เป็นการจ่ายกระแส

จากแหล่งจ่ายกระแสคงที่ภายในตัวไอซี ทำให้มีกระแสในขณะลอจิก “1” นี้เพียง 100 μA เท่านั้น ในกรณีต้องการให้มีการจ่ายกระแสสูงๆ ต้องต่อตัวต้านทานพลูอัปไว้ที่ขาพอร์ตด้วย

เมื่อต้องการให้ขาพอร์ตทำหน้าที่เป็นอินพุตต้องส่งสัญญาณให้มีลอจิก “1” ก่อนเมื่อขาอินพุตได้รับสัญญาณจากภายนอกป้อนเข้ามา ไอซีPCF8574A จะสร้างสัญญาณอินเตอร์รัปต์ (INT) ป้อนให้ไมโครคอนโทรลเลอร์ หรือคอมพิวเตอร์รับรู้แทนการตรวจสอบขาอินพุตตลอดเวลา สัญญาณอินเตอร์รัปต์ถูกรีเซตเมื่อมีการอ่านค่าข้อมูลหรือการเปลี่ยนค่าของอินพุตไปสู่ค่าเดิม

การติดต่อกับไอซี PCF8574A สามารถกำหนดแอดเดรสดังนี้

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0	1	1	1	A2	A1	A0	$R/\overline{W}$

บิต A0, A1, A2 ใช้ระบุ PCF8574A ในกรณีต่อ PCF8574A มากกว่า 1 ตัวโดยค่าของ A0-A2 มีความแตกต่างกันในแต่ละตัวสามารถกำหนดได้ทางฮาร์ดแวร์ โดยต่อขา A0-A2 เข้ากับไฟเลี้ยง +5โวลต์เพื่อกำหนดให้เป็นลอจิก “1” หรือกราวด์เพื่อกำหนดเป็นลอจิก “0” ส่วนบิต R/W ใช้กำหนดการอ่านหรือเขียนข้อมูลกับไอซี PCF8574A

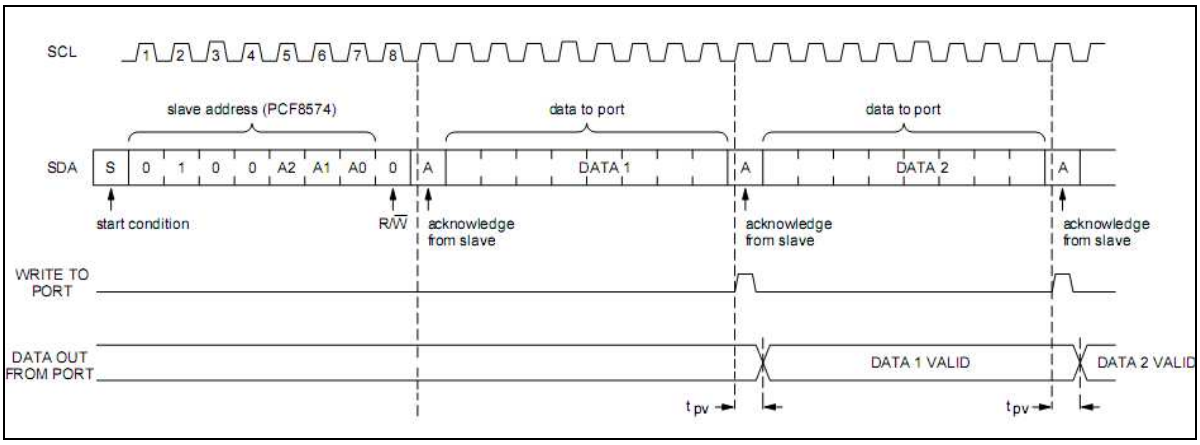
ตัวอย่าง ถ้ากำหนดขา A0-A2 ลงกราวด์ทั้งหมด และต้องการอ่านข้อมูลจาก PCF8574A ข้อมูลต้องกำหนดแอดเดรส คือ 01110001 B เป็นต้น

ไอซีขยายพอร์ต PCF8574 มีการกำหนดแอดเดรสแตกต่างกับ PCF8574A แต่ฟังก์ชั่นการทำงานเหมือนกันทุกประการ โดยข้อมูลกำหนดแอดเดรสของ PCF8574 สามารถต่อพ่วงไอซีอนุกรม PCF8574x ได้สูงสุดถึง 16 ตัว มีรายละเอียดดังนี้

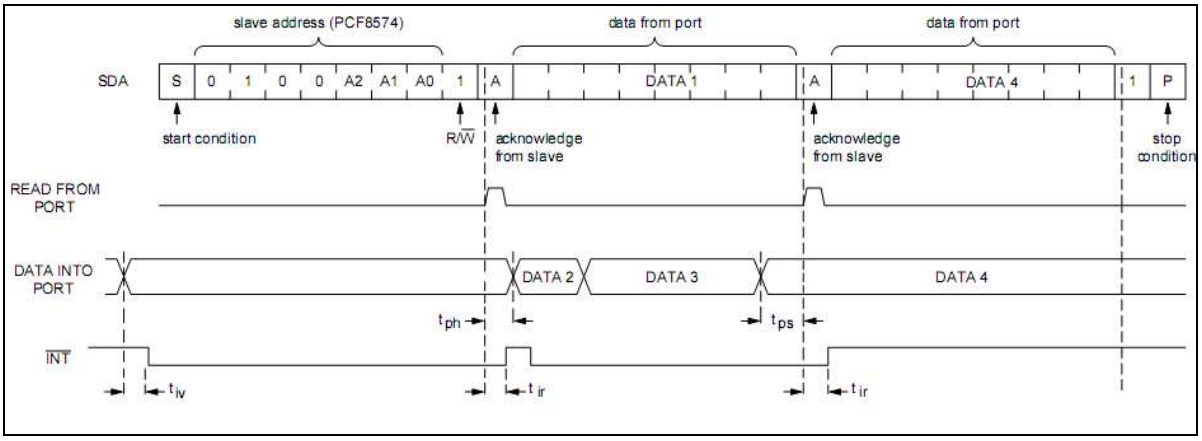
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0	1	0	0	A2	A1	A0	$R/\overline{W}$

2.6.2 การส่งหรือเขียนข้อมูลไปยัง PCF8574A แสดงในภาพที่ 10.11 มีขั้นตอนดังนี้

1) กำหนดแอดเดรสของ PCF8574A
2) เรียกโปรแกรมย่อยการติดต่ออุปกรณ์สเลฟ
3) รอรับการตอบกลับจาก PCF8574A
4) ส่งข้อมูลไปยัง PCF8574A
5) เรียกโปรแกรมย่อยสภาวะหยุด



ภาพที่ 10.11 แสดงเขียนข้อมูลไอซีขยายพอร์ต PCF8574A



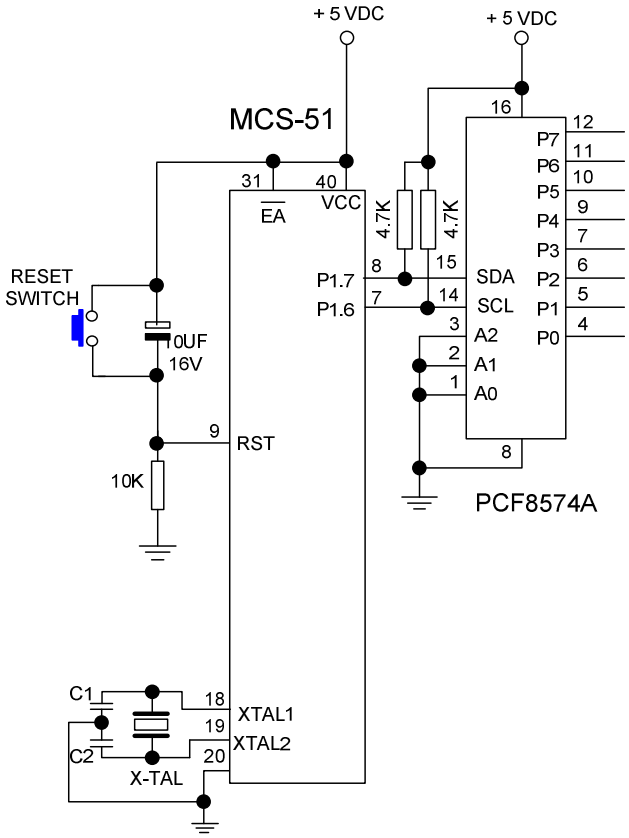
ภาพที่ 10.12 แสดงอ่านข้อมูลจากไอซีขยายพอร์ต PCF8574A

2.6.3 การอ่านข้อมูลจาก PCF8574A แสดงในภาพที่ 10.12 มีขั้นตอนดังนี้

1) กำหนดแอดเดรสของ PCF8574A
2) เรียกโปรแกรมย่อยการติดต่ออุปกรณ์สเลฟ
3) รอรับการตอบกลับจาก PCF8574A
4) อ่านข้อมูลจาก PCF8574A โดยใช้โปรแกรมย่อยการอ่านข้อมูลจากอุปกรณ์สเลฟ
5) เรียกโปรแกรมย่อยสภาวะหยุด

2.6.4 การเขียนโปรแกรมใช้งานไอซี PCF8574A

การติดต่อระหว่างไอซี MCS-51 กับไอซี PCF8574A โดยกำหนดให้ขาหนึ่งเป็น SDA อีกขาหนึ่งเป็น SCL และต่อตัวต้านทานค่าประมาณ 4.7K พลูอัฟที่ขาพอร์ตทั้งสองขาแสดงดังภาพที่ 10.13



ภาพที่ 10.13 การติดต่อระหว่างไอซี MCS-51 กับไอซี PCF8574A

ตัวอย่างที่ 3 เขียนโปรแกรมส่งค่าให้เอาต์พุตไอซี PCF8574A โดยกำหนดค่าข้อมูล F0H การต่อวงจรแสดงดังภาพที่ 10.13

```
SDA         BIT     P1.7
SCL         BIT     P1.6
SW_READ     BIT     P1.0
INKEY       BIT     P3.5
FLAG        EQU     2FH
I2C_ACK     BIT     FLAG.0
DIO_ID      EQU     01110000B
I2C_ADDR    EQU     029H
I2C_DATA    EQU     028H
INPORT      EQU     040H
OUTPORT     EQU     041H
```

```
;******** MAIN PROGRAM **********
                ORG     0000H
MAIN:           MOV     OUPPORT, #0F0H
                ACALL   DIO_WR
                SJMP    MAIN
;******** I2C DIO WRITE **********
DIO_WR:         MOV     I2C_ADDR, # DIO_ID
                ACALL   I2C_SLAVE
                MOV     I2C_DATA, OUTPORT
                ACALL   I2C_DATA_WR
                ACALL   I2C_STOP
                RET
;******* I2C STOP CONDITION *******
I2C_STOP:       CLR     SDA
                ACALL   I2C_DELAY
                SETB    SCL
                ACALL   I2C_DELAY
                SETB    SDA
                RET
;******* I2C    SLAVE CONNECTS *******
;****        I/P: I2C_ADDR                ****
;****    O/P FLAG:       I2C_ACK         ****
I2C_SLAVE:      PUSH    ACC
                SETB    I2C_ACK
                MOV     A, I2C_ADDR
                ACALL   I2C_START
                MOV     R5, #8
I2C_SLAVE_1:    RLC     A
                MOV     SDA,C
                ACALL   I2C_CLK
                DJNZ    R5, I2C_SLAVE_1
```

เป็นโปรแกรมหลักกำหนดให้ส่งข้อมูล F0H ออกไปที่พอร์ตของไอซี PCF8574A และเรียกโปรแกรมย่อยการเขียนข้อมูลออกไป

เป็นกระบวนการส่งข้อมูลกำหนดขั้นตอนดังนี้
1. ส่งข้อมูลกำหนดแอดเดรสติดต่อตัวรับในโปรแกรมย่อย I2C_SLAVE
2. นำข้อมูลที่ต้องการส่งไปแสดงผล (ข้อมูลจริง) ในโปรแกรมย่อย I2C_DATA_WR
3. อยู่ในสภาวะการหยุดส่งโดยโปรแกรมย่อย I2C_STOP

เป็นกระบวนการส่งข้อมูลกำหนดตำแหน่งของตัวรับ
1. นำข้อมูลที่ต้องการส่งไว้ในรีจิสเตอร์ A
2. เรียกโปรแกรมย่อย I2C_START เข้าสู่สภาวะเริ่มต้น
3. กำหนดจำนวนการส่งไว้ที่รีจิสเตอร์ R5 = 8 ไบต์
4. หมุนข้อมูลผ่านแฟลกตัวทดที่ละไบต์ส่งออกที่ SDA
5. เรียกโปรแกรมย่อยกำเนิดสัญญาณนาฬิกาเพื่อส่งที่ละไบต์

```
                SETB    SDA
                ACALL   I2C_DELAY
                SETB    SCL
                ACALL   I2C_DELAY
                JB      SDA, I2C_SLAVE_2
                CLR     I2C_ACK
I2C_SLAVE_2:    CLR     SCL
                POP     ACC
                RET
;********* I2C DATA WRITE ***********
;****        I/P: I2C_DATA              ****
I2C_DATA_WR:    PUSH    ACC
                SETB    I2C_ACK
                MOV     A, I2C_DATA
                MOV     R5,#8
I2C_DATA_WR_1:  RLC     A
                MOV     SDA, C
                ACALL   I2C_CLK
                DJNZ    R5, I2C_DATA_WR_1
                SETB    SDA
                CALL    I2C_DELAY
                SETB    SCL
                ACALL   I2C_DELAY
                JB      SDA, I2C_DATA_WR_2
                CLR     I2C_ACK
I2C_DATA_WR_2:  CLR     SCL
                POP     ACC
                RET
;********* I2C ACKNOWLEDGE **********
I2C_ACK_BIT:    CLR     SDA
                ACALL   I2C_DELAY
                ACALL   I2C_CLK
```

เป็นการตรวจสอบสัญญาณการตอบรับจากอุปกรณ์ที่ติดต่อด้วย โดยบิต I2C_ACK เป็น “1” เมื่อไม่มีการตอบรับ และเป็น “0” เมื่อมีการตอบรับ

ส่งข้อมูลที่ต้องการส่งไปแสดงผล (ข้อมูลจริง)
1. นำข้อมูลที่ต้องการส่งไว้ในรีจิสเตอร์ A
2. กำหนดจำนวนการส่งไว้ที่รีจิสเตอร์ R5 = 8 ไบต์
3. หมุนข้อมูลผ่านแฟลกตัวทดที่ละไบต์ส่งออกที่ SDA
4. เรียกโปรแกรมย่อยกำเนิดสัญญาณนาฬิกาเพื่อส่งที่ละไบต์
5. ตรวจสอบการส่งข้อมูลครบ 8 ไบต์หรือยัง

เป็นการตรวจสอบสัญญาณการตอบรับจากอุปกรณ์ที่ติดต่อด้วย โดยบิต I2C_ACK เป็น “1” เมื่อไม่มีการตอบรับ และเป็น “0” เมื่อมีการตอบรับ

```
                SETB    SDA
                RET
;******* I2C START CONDITION *******
I2C_START:      SETB    SCL
                SETB    SDA
                ACALL   I2C_DELAY
                CLR     SDA
                ACALL   I2C_DELAY
                CLR     SCL
                RET
;******* I2C CLOCK *******
I2C_CLK:        ACALL   I2C_DELAY
                SETB    SCL
                ACALL   I2C_DELAY
                CLR     SCL
                RET
;******* I2C NOT ACKNOWLEDGE *******
I2C_NACK_BIT:SETB  SDA
                ACALL   I2C_DELAY
                ACALL   I2C_CLK
                RET
;******* I2C DELAY *******
I2C_DELAY:      MOV     R6, #0CH
I2C_DELAY1:     NOP
                DJNZ    R6, I2C_DELAY1
                RET
                END
```

ตัวอย่างที่ 4 จากโปรแกรมตัวอย่างที่ 3 เขียนโปรแกรมอ่านค่าจากไอซี PCF8574A กำหนดให้แสดงผลที่พอร์ต P2 ของไอซี MCS-51 และอ่านค่าทุกครั้งที่กดสวิตซ์ P1.0 โดยมีโปรแกรมเพิ่มเติมดังนี้

```
INPORT          EQU     040H
OUTPORT         EQU     041H
```

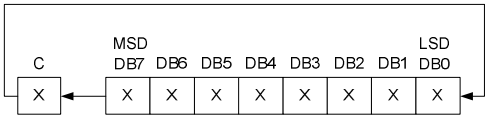
```
                ORG     0000H
                SETB    SW_READ
MAIN:           ACALL   DIO_RD
                MOV     A, INPORT
                MOV     P2, A
WAIT:           JB      SW_READ, WAIT
                SJMP    MAIN
;********* I2C DIO READ***********
DIO_RD:         MOV     I2C_ADDR, # DIO_ID+1
                ACALL   I2C_SLAVE
                ACALL   I2C_DATA_RD
                MOV     INPORT, I2C_DATA
                ACALL   I2C_NACK_BIT
                ACALL   I2C_STOP
                RET
;********* I2C DATA READ ***********
;****        O/P: I2C_DATA          ****
I2C_DATA_RD:    PUSH    ACC
                CLR     A
                MOV     R5, #8
I2C_DATA_RD_1:  ACALL   I2C_DELAY
                SETB    SCL
                ACALL   I2C_DELAY
                MOV     C, SDA
                RLC     A
                CLR     SCL
                DJNZ    R5, I2C_DATA_RD_1
                MOV     I2C_DATA, A
                POP     ACC
                RET
```

เป็นกระบวนการอ่านค่าข้อมูล กำหนดขั้นตอนดังนี้

1. ส่งข้อมูลกำหนดแอดเดรสติดต่อตัวรับ+1 ในโปรแกรมย่อย I2C_SLAVE
(บิต0 เลือก เขียนหรืออ่านจากไอซี +1 คืออ่านค่า)
2. เรียกโปรแกรมย่อยอ่านค่าจากไอซี PCF8574A
3. นำข้อมูลที่อ่านได้เก็บไว้ใน INPORT
4. เรียกโปรแกรมย่อยให้ตัวส่งหยุดทำการส่งข้อมูล
5. อยู่ในสภาวะการหยุดส่งโดยโปรแกรมย่อย I2C_STOP

การอ่านข้อมูล

1. กำหนดจำนวนการรับไว้ที่จิสเตอร์ R5 = 8 ไบต์
2. รับข้อมูลที่ SDA โดยหมุนข้อมูลผ่านแฟลกตัวทดที่ละไบต์เก็บไว้ในรีจิสเตอร์ A
5. ตรวจสอบการรับข้อมูลโดยลดค่า R5 เมื่อครบ 8 ไบต์ นำข้อมูลเก็บไว้ที่ I2C_DATA



ตัวอย่างที่ 5 จากโปรแกรมตัวอย่างที่ 3 ให้เขียนโปรแกรมอ่านค่าจากไอซี PCF8574A กำหนดให้แสดงผลที่พอร์ต 2 ของไอซี MCS-51 โดยจะรับค่าเมื่อมีสัญญาณจากขาอินเตอร์รัปต์ของไอซี PCF8574A ที่ต่อกับ P3.5 ของไอซี MCS-51 โดยมีโปรแกรมเพิ่มเติมจากตัวอย่างที่ 3 ดังนี้

```
INPORT      EQU   040H
OUTPORT     EQU   041H
            ORG   0000H
;********* MAIN PROGRAM ***********
            MOV   30H, #00H
MAIN:       MOV   OUTPORT, #0FFH
            ACALL DIO_WR
LOOP:       MOV   P2, 30H
            JB    INKEY, LOOP
            ACALL DIO_RD
            MOV   30H, INPORT
            SJMP  LOOP
;********* I2C DIO READ***********
DIO_RD:     MOV   I2C_ADDR, # DIO_ID+1
            ACALL I2C_SLAVE
            ACALL I2C_DATA_RD
            ACALL I2C_NACK_BIT
            MOV   INPORT, I2C_DATA
            ACALL I2C_STOP
            RET
;********* I2C DATA READ ***********
;****      O/P: I2C_DATA          ****
I2C_DATA_RD:      PUSH  ACC
                  CLR   A
                  MOV   R5, #8
I2C_DATA_RD_1:    ACALL I2C_DELAY
                  SETB  SCL
                  ACALL I2C_DELAY
```

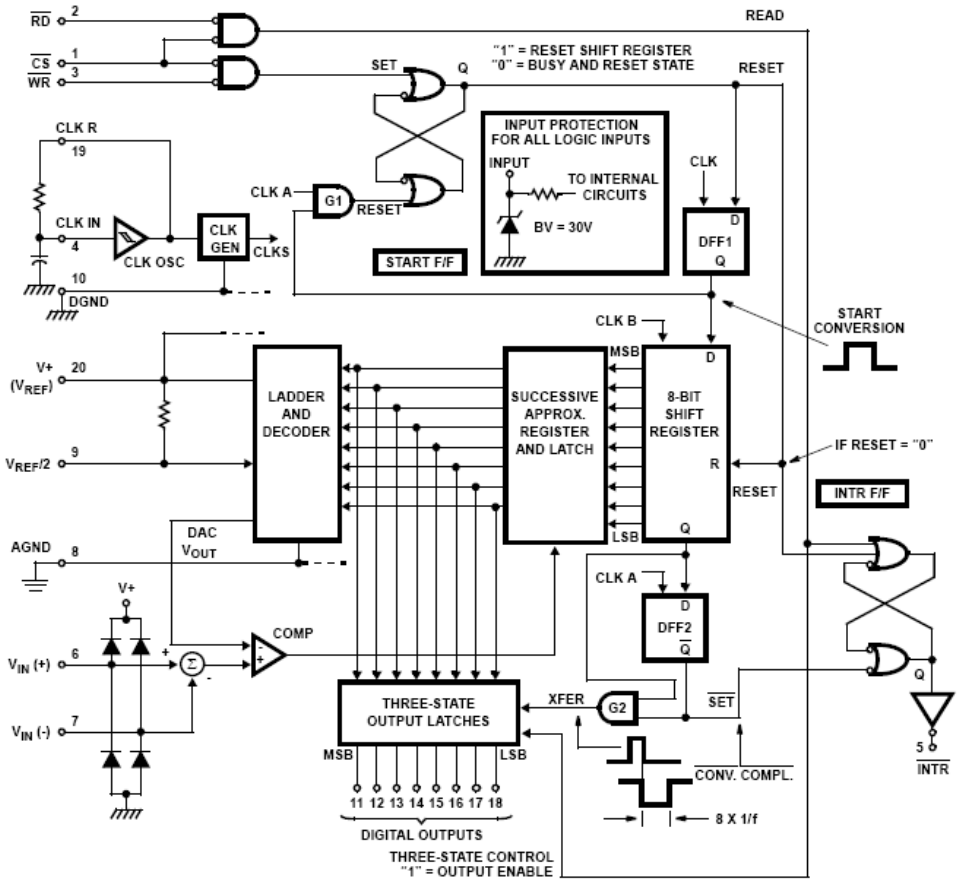
1. ส่งข้อมูลให้กับไอซี PCF8574A ให้เป็น “1” ทุกบิต (FFH) ในกรณีเป็นการอ่านค่า (อินพุต)
2. ตรวจสอบการรับข้อมูลจากขา INT ของไอซี PCF8574A ที่ขา INKEY ของไอซี MCS-51
3. เรียกโปรแกรมอ่านค่าข้อมูล

```
MOV    C, SDA
RLC    A
CLR    SCL
DJNZ   R5, I2C_DATA_RD_1
MOV    I2C_DATA, A
POP    ACC
RET
```

3. การแปลงสัญญาณอนาล็อกเป็นสัญญาณดิจิตอล

เป็นการแปลงสัญญาณอนาล็อก (A/D: Analog To Digital Converter) ที่มีการเปลี่ยนแปลงอย่างต่อเนื่อง เช่นระดับของแรงดันไฟฟ้า หรือปริมาณของกระแสไฟฟ้า ให้กลายเป็นสัญญาณดิจิตอลที่อยู่ในรูปแบบของเลขฐานสองคือ “0” กับ “1” ซึ่งเป็นสัญญาณที่ไม่ขึ้นอยู่กับเวลา

3.1 ไอซีแปลงสัญญาณอนาล็อกเป็นสัญญาณดิจิตอล ADC 0804



ภาพที่ 10.14 วงจรภายใน และการจัดวางขาของไอซีเบอร์ ADC 0804

ไอซีที่ทำหน้าที่เป็นวงจรแปลง A/D มีหลายชนิดด้วยกัน เช่น Dual Slope Type, SAR (Successive Approximation Register Type), Flash Type และชนิด Tracking type ไอซีเบอร์ ADC 0804 ทำหน้าที่เป็นวงจรแปลง A/D แบบ **SAR** (Successive Approximation Register Type) มีความเร็วประมาณ 100 μS มีวงจรภายใน แสดงดังภาพที่ 10.14 หน้าที่ของแต่ละขาของไอซีเบอร์ADC 0804มีรายละเอียดดังนี้

ขา 1 คือขา CS รับสัญญาณการเลือกทำงาน

ขา 2 คือขา RD (Read) ขนาดควบคุมการอ่านข้อมูลใน ADC0804 โดยสัญญาณควบคุมมาจากไมโครคอนโทรลเลอร์

ขา 3 คือขา WR (Write) ขาควบคุมการเขียนข้อมูลลงใน ADC0804 โดยสัญญาณควบคุมมาจากไมโครคอนโทรลเลอร์

ขา 4 คือขา CLK in รับสัญญาณนาฬิกาจากภายนอก

ขา 5 คือขา INTR (Interrupt) รับสัญญาณอินเตอร์รัพต์มาจากไมโครคอนโทรลเลอร์

ขา 6 คือขา VIN (+) เป็นอินพุตรับแรงดันบวก

ขา 7 คือขา VIN (-) เป็นอินพุตรับแรงดันลบ (O V)

ขา 8 คือขา A GND จุดดินของวงจรอนาล็อกภายใน ADC0804

ขา 9 คือขา Vref / 2 เป็นอินพุตรับแรงดันอ้างอิง

ขา 10 คือขา D GND จุดดินของวงจรดิจิตอลภายใน ADC0804

ขา 11 –18 คือขา Digital output ขา 11 คือ D7 (MSB) และขา 18 คือขา Do (LSB)

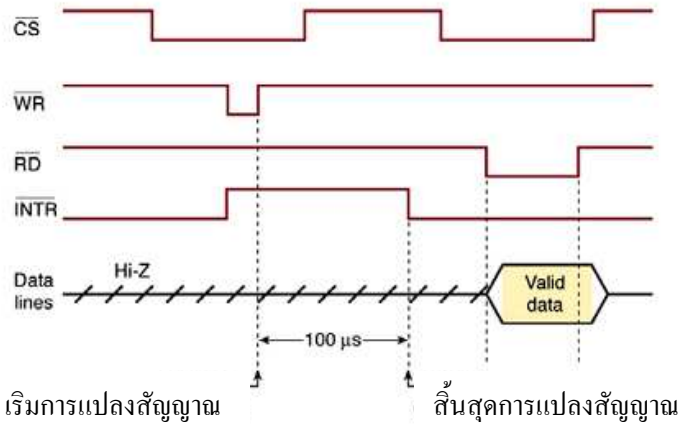
ขา 19 คือขา CLK out เป็นอินพุตสำหรับต่อตัวต้านทานภายนอกสำหรับสัญญาณนาฬิกา

ขา 20 คือขา VCC (or Ref) เป็นขาจ่ายกำลังไฟฟ้า +5 VDC

ไอซี ADC0804 มีโครงสร้างของวงจรดิจิตอลภายในเป็นแบบซีมอส มีความเร็วในการแปลงสัญญาณแต่ละรอบเท่ากับ 100 μS โครงสร้างภายในเป็นแบบ **Successive Approximation** หรือแบบประมาณค่าต่อเนื่อง และรับแรงดันอนาล็อกอินพุตได้ในย่าน 0 ถึง +5 โวลต์ ด้านเอาต์พุต 8 บิต ดิจิตอลมีลอจิก 3 สถานะเป็นบัฟเฟอร์ทำให้ต่อเข้ากับบัสข้อมูลของระบบ ไอซี MCS-51ได้โดยตรง ไอซี ADC0804 มีค่าความแยกชัดต่อบิต19.6 mV (เมื่อ VCC = +5 โวลต์ ดังนั้น Steps = 5V /255 = 19.6 mV) ความถี่ของสัญญาณที่เหมาะสมคือ f = 1/1.1RC ค่าในคู่มือกำหนด R = 10 KΩ และ C = 150 pF, R และ C ต่อขา CLK out และ CLK in เมื่อแทนค่า R = 10 KΩ C = 150 pF จะได้ความถี่ของสัญญาณภายในเท่ากับ 606 kHz แต่ถ้าใช้ สัญญาณนาฬิกาภายนอก ต้องต่อเข้าที่ขา CLK In และเปิดวงจรขา CLK Out ที่ความถี่ 606 kHz การต่อจุดกราวด์ของไอซี ADC0804 ควรแยกกันระหว่างจุดกราวด์ของดิจิตอล และจุดกราวด์ของอนาล็อก เนื่องจากในระบบดิจิตอลมีสัญญาณรบกวน มากกว่า อาจทำให้การทำงานไม่สมบูรณ์ได้

การแปลงสัญญาณเริ่มจากการทำงานของวงจร A/D โดยให้ขา $\overline{WR}$ เป็น “0” หลังจากนั้นไอซีจะเริ่มแปลงสัญญาณ โดยความเร็วในการแปลงสัญญาณ ขึ้นกับชนิดของ A/D แต่ละเบอร์ เช่นเบอร์

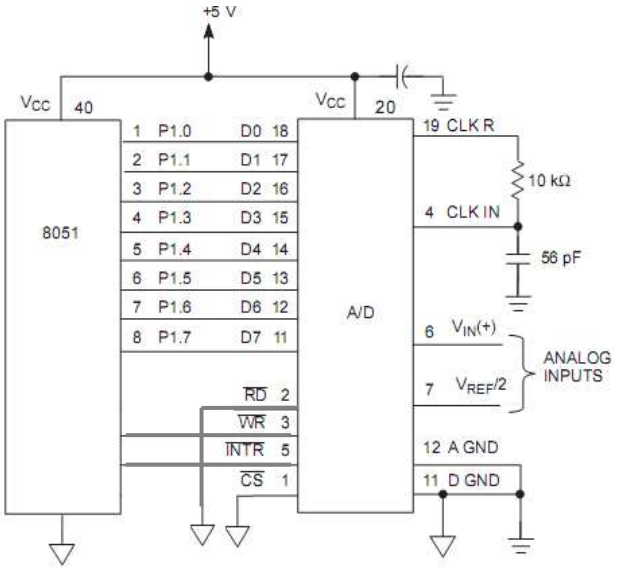
ADC0804 มีค่า Conversion Time ประมาณ 100 µS เมื่อแปลงสัญญาณเสร็จ จะส่งสัญญาณออกไปที่ขา INTR เป็น “0” สัญญาณ A/D บางเบอร์ จะเขียนบอกเป็น EOC (End of Conversion) ดังนั้น การเขียนโปรแกรมจึงต้องเขียนพอร์ต A/D ก่อน (เพื่อทำให้ $\overline{WR}$ และ $\overline{CS}$ ของ A/D แอคทีฟ) หลังจากนั้นจึงคอยให้สถานะของขา INTR เป็น “0” จึงอ่านข้อมูลไปเก็บไว้ และสิ้นสุดการทำงาน 1 รอบ ยังสามารถใช้วิธีการอ่านค่าโดยเริ่มให้ A/D ทำงาน และให้วนลูปรอจนกว่าเวลาครบ 100 µS จึงอ่านค่าข้อมูลจาก A/D ได้ ไดอะแกรมเวลาของสัญญาณ $\overline{CS}$ $\overline{WR}$ $\overline{RD}$ และ INTR ใน ADC0804 แสดงได้ดังภาพที่ 10.15



ภาพที่ 10.15 แสดงไดอะแกรมเวลาของขาสัญญาณของไอซี ADC0804

3.2 การต่อไอซี MCS-51 เข้ากับไอซีเบอร์ ADC0804

การต่อใช้งานไอซี MCS-51 กับไอซี ADC0804 แสดงดังภาพที่ 10.16



ภาพที่ 10.16 แสดงการต่อวงจรไอซี ADC0804 กับไอซี MCS-51

ขาข้อมูล DB0-DB7 ของ A/D ต่อเข้า P1 ซึ่งเป็นดาต้าบัส ขา $\overline{CS}$ ต่อเข้า I/O Decoder หรือสามารถต่อที่ขา $\overline{INT0}$ และ $\overline{INT1}$ การเขียนโปรแกรมกำหนดดังนี้

3.3 การเขียนโปรแกรม

ตัวอย่างที่ 6 กำหนดให้เขียนโปรแกรมควบคุมให้ ADC 0804 แปลงสัญญาณ หลังจากนั้นให้เก็บค่าไว้ในรีจิสเตอร์ A กำหนดให้ไอซี MCS-51 ทำงานที่ 12 MHz เขียนโปรแกรมได้ดังนี้

คำนวณโปรแกรมหน่วงเวลา เพื่อให้ได้ค่า 100 µSec ไอซี MCS-51 ทำงานที่ 12 MHz

ดังนั้นใน 1 แมชชีนไซเคิล = 1 µS

```
;**** Subroutine for Delay 100 µS *****
DELAY:   MOV R0, #xxH      [1]
LOOP:    DJNZ R0, LOOP     [2]
         NOP               [1]
         RET               [2]
```

สูตร = 1 + R0 (2) + 1+ 2 = 4 +R0 (2)

100 = 4 +R0 (2)

R0 = $96/2 = 48_{10} = 30H$

เขียนเป็นโปรแกรมคำสั่งได้ดังนี้

```
        ORG   0000H
WR      BIT   P0.1
INT     BIT   P3.1
MAIN:   CLR   WR          ; WR มีสถานะลอจิกเป็น “0”
        CALL  DELAY       ; หน่วงเวลา 100 µSec
        JB    INT,$       ; เป็น “1” ให้วนคอย
        MOV   A,P1
        SETB  WR
        SJMP  MAIN
;**** DELAY 100 µS  X-TAL 12 MHz ****
DELAY:  MOV R0, #30H
LOOP:   DJNZ R0, LOOP
        NOP
        RET
        END
```

4. การประยุกต์ใช้งานไอซี MCS-51 โดยภาษาซี

ภาษาซี เป็นภาษาคอมพิวเตอร์ ที่ใช้เขียนโปรแกรมคอมพิวเตอร์ ที่มีประสิทธิภาพสูง มีผู้ผลิตหลายบริษัทได้ผลิต**โปรแกรมแปลภาษา** (**C**ompiler) ใช้เปลี่ยนชุดคำสั่งที่เขียนในรูปแบบของภาษาซีไปเป็นภาษาเครื่องเพื่อสั่งให้คอมพิวเตอร์ทำงานตามต้องการได้ และแต่ละบริษัทอาจมีในหลายชื่อหลายรุ่น ซึ่งแต่ละรุ่นอาจมีความแตกต่างกันไปบ้าง แต่ส่วนใหญ่จะเป็นไปตามมาตรฐานของสถาบัน **ANSI** (American Standard National Institute) ซึ่งได้กำหนดมาตรฐานของภาษาซีไว้เรียกว่า ANSI C

ภาษา C51 เป็นภาษาที่ใช้เขียนโปรแกรมสำหรับสั่งงานควบคุมไมโครคอนโทรลเลอร์ MCS-51 ซึ่งเป็นภาษาระดับสูงที่มีรูปแบบโครงสร้างและไวยากรณ์เหมือนกับภาษาซี แต่จะเพิ่มคำสั่งและฟังก์ชั่นสำหรับติดต่อกับไอซี MCS-51 ภาษา C51 จะเป็นไปตามมาตรฐานของสถาบัน ANSI

4.1 โครงสร้างของโปรแกรมที่เขียนด้วยภาษาซี

โปรแกรมที่เขียนด้วยภาษาซีจะมีโครงสร้างดังนี้

```
#include <library>   หรือ header file
```

ส่วนประกาศ (Declaration part) ส่วนนี้ใช้ประกาศตัวแปร ค่าคงที่ ชนิดของข้อมูลแบบโกลบอล (global)

```
main() ฟังก์ชันหลักของโปรแกรม
{
        คำสั่งต่าง ๆ ค่าคงที่ ฯลฯ
}
ฟังก์ชันย่อย (sub function)
{
      คำสั่งต่าง ๆ ค่าคงที่ ฯลฯ
}
```

ไฟล์ส่วนหัว(Header file) ประกอบด้วย #include ซึ่งจัดเป็นฟรีโพรเซสเซอร์ ไดเร็กทีฟ (Preprocessor Directives) เมื่อมีการแปลภาษา (Compiler) โปรแกรมยังไม่ต้องการ Source code แต่จะเรียกโพรเซสเซอร์ไดเร็กทีฟ เพื่อตรวจสอบว่ามีการเรียกใช้ฟังก์ชันใดบ้างมีอยู่ในไฟล์ส่วนหัวที่เรียกใช้หรือไม่ ไฟล์ส่วนหัวจะมีส่วนขยายเป็น .h จะเป็นที่เก็บรวบรวมฟังก์ชันในไลบารี่ (Library) โดยที่แต่ละไฟล์ส่วนหัวจะจัดเก็บฟังก์ชันต่าง ๆ ไว้ แตกต่างกันไป เช่น stdiol.h จะเกี่ยวฟังก์ชันต่าง ๆที่เกี่ยวกับ I/O ต่าง ๆ โดยฟรีโพรเซสเซอร์ ไดเร็กทีฟจะเริ่มต้นด้วย # และไม่ต้องจบด้วย เครื่องหมาย ; เพราะไม่ใช่คำสั่งหรือฟังก์ชัน โดยไดเร็กทีฟ มีหลายตัว เช่น include , define โดยการเขียนโปรแกรมบางครั้งอาจต้องเรียกไฟล์ส่วนหัวหลายไฟล์เพราะมีการเรียกใช้ฟังก์ชันที่อยู่ในไลบารี่ที่เก็บไว้โดยไฟล์ส่วนหัวต่างกัน

ในการประกาศ ชนิด ค่าของตัวแปร ค่าคงที่เป็นแบบโกลบอล จะถูกเรียกไปใช้ได้ในทุก ๆ ฟังก์ชันของโปรแกรม ในส่วนนี้ยังเป็นที่ใช้ประกาศว่ามีฟังก์ชันย่อยใดบ้างในโปรแกรม

main () เป็นฟังก์ชันหลักของโปรแกรม ซึ่งต้องมีอย่างน้อย 1 ฟังก์ชัน การทำงานของฟังก์ชันนี้นอกจากใช้คำสั่ และใช้ฟังก์ชันต่างๆ จากไลบารี่ (Library) แล้วยังสามารถเรียกใช้ฟังก์ชันย่อย ต่าง ๆ ได้

ฟังก์ชันย่อย (Sub Function) เป็นฟังก์ชันที่มีการสร้างขึ้นเพื่อให้โปรแกรมเพื่อให้โปรแกรมสามารถทำงานได้ตามต้องการ

ในการเขียนโปรแกรมภาษาซีมีโครงสร้างพื้นฐานดังนี้

1) ฟังก์ชัน main() จะไม่ส่งค่าให้ฟังก์ชันอื่น ๆ และไม่ต้องรับค่า ใช้คำสั่ง void main(void) โดย void หมายถึงไม่ส่งค่ากลับ ส่วน(void) หมายถึงไม่มีอาร์กิวเมนต์ (Argument) คือไม่ต้องรับค่า

2) ขอบเขตของฟังก์ชัน จะเริ่มต้นด้วยเครื่องหมายปีกกาเปิด ({) และสิ้นสุดขอบเขตด้วยเครื่องหมายปีกกาปิด (}) ดังนั้นการเขียนฟังก์ชันใดต้องเริ่มต้นด้วย { และจบด้วย } ในทำนองเดียวกันเมื่อนำคำสั่งหลาย ๆ คำสั่งมาประกอบเป็นชุดเดียวกันจะเริ่มด้วย { และจบด้วย } เช่นเดียวกัน

3) คำสั่งแต่ละคำสั่งจะต้องจบด้วยเครื่องหมาย; (Semicolon)

4) ชื่อฟังก์ชันและคำสั่งต่าง ๆ ในภาษาซีจะต้องใช้ตัวอักษรตัวเล็ก (Lowercase)

5) ชื่อตัวแปรจะใช้ตัวอักษรตัวเล็กหรือตัวอักษรตัวใหญ่ (Uppercase) ก็ได้ และภาษาซีจะถือว่าตัวอักษรตัวเล็กตัวใหญ่ต่างกัน (Case Sensitive) ตัวแปรในฟังก์ชันต่างกันสามารถใช้ชื่อเหมือนได้แต่ในฟังก์ชันเดียวกันจะมีตัวแปรชื่อเหมือนกันไม่ได้

ตัวอย่างโปรแกรมภาษาซี

```
/* ตัวอย่างโปรแกรมภาษาซี */   หรือ // ตัวอย่างภาษาซี
#include <stdio.h>                    /* บรรทัดที่1*/
main()                                // บรรทัดที่ 2
{                                     /* บรรทัดที่3 */
Printf ("Microcontroller Mcs-51 \n");  // บรรทัดที่4
 }                                    // บรรทัดที่ 5
```

บรรทัดที่ 1 **#include <stdio.h>** เป็นการเรียกใช้แฟ้มที่บรรจุความหมายรูปแบบของฟังก์ชันที่จำเป็นต้องใช้ในที่นี้คือ printf () อยู่ในแฟ้มนี้

บรรทัดที่ 2 ไม่มีชนิดของฟังก์ชันนำหน้าชื่อ **main** และ ไม่มีอาร์กิวเมนต์ในวงเล็บ หรือ อาจเขียนเป็น void main(void) หมายถึงไม่มีการส่งค่าจากฟังก์ชันนี้และไม่มีการรับค่าที่ส่งมา

บรรทัดที่ 3 { แสดงถึงฟังก์ชัน main () มีจุดเริ่มตั้งแต่ตำแหน่งนี้

บรรทัดที่ 4 ให้พิมพ์คำว่า Microcontroller Mcs-51 ที่จอภาพแล้วให้ขึ้นบรรทัดใหม่ (\n)

บรรทัดที่ 5 } แสดงว่าสิ้นสุดขอบเขตของฟังก์ชัน main()

เครื่องหมาย /* ..... */ ใช้สำหรับเขียนหมายเหตุในโปรแกรม โดยเมื่อมีการแปลภาษา (Compiler) จะไม่ดำเนินการกับข้อความที่อยู่ระหว่างเครื่องหมาย /* */ ในกรณีที่หมายเหตุอยู่ในบรรทัดเดียว อาจนำหน้าด้วย // บอกให้รู้ว่าสิ่งที่อยู่ต่อจาก // ตัวแปลภาษาจะไม่ดำเนินการกับข้อความนั้น

4.2 องค์ประกอบพื้นฐานของภาษาซี

องค์ประกอบของภาษาซี จะประกอบด้วย อักขระ(Characters) ค่าคงที่ (Constants) ตัวแปร (Variables) ตัวดำเนินการ (Operators) และนิพจน์ (Expressions)

4.2.1 อักขระ(Characters) แบ่งออกได้ดังนี้

1) ตัวเลข (Digits) คือ ตัวเลข 0 ถึง 9 และเลขฐานสิบหก คือ A B C D E และ F

2) ตัวอักษร(Letters) คือตัวอักษรในภาษาอังกฤษ เป็นตัวพิมพ์ใหญ่และตัวพิมพ์เล็ก คือ A-Z และ a-z

3) อักขระพิเศษ(Special Characters) คือ **! * + " < # (= | > %) ~ : / ^ - [; ? , & _] ' . Space** ในภาษาซี ถือว่า เครื่องหมายขีดเส้นใต้เป็นตัวอักษร ตัวหนึ่งอักขระต่าง ๆ นี้จะใช้เป็นค่าคงที่ ตัวแปร ตัวดำเนินการ

4.2.2 ค่าคงที่ (Constants) เป็นค่าที่มีค่าไม่เปลี่ยนแปลง แบ่งออกได้ดังนี้

1) ค่าคงที่ประเภทเลขจำนวนเต็ม (Integer Constant) คือเลขจำนวนเต็ม ที่อยู่ระหว่าง **32768 ถึง 32767** เช่น -25 หรือ 0 หรือ 236 ค่าเหล่านี้แต่ละค่าใช้หน่วยความจำในการเก็บ 2 ไบต์ ค่าเหล่านี้เขียนในรูปเลขฐานสิบ ฐานแปด(เขียนโดยใช้เลขศูนย์นำหน้า เช่น 045) เลขฐานสิบหก(เขียนโดย ใช้เลขศูนย์และx นำหน้า เช่น 0x28 0X2AF)

2) ค่าคงที่ชนิดตัวเลขทศนิยม(Floating Point Constants) เป็นตัวเลขทศนิยม เช่น 1.0 1.6 E+09 ค่านี้ใช้หน่วยความจำในการเก็บค่าละ 4 ไบต์ และมีค่าอยู่ในช่วง **1.2 E -38 ถึง 3.4E+38** โดย 3 ไบต์แรกเก็บค่าตัวทศนิยม ส่วนไบต์สุดท้ายเก็บเลขยกกำลัง

3) ค่าคงที่ชนิดตัวเลขทศนิยมความละเอียดสองเท่า (Double Floating Point) หรือเรียกว่า Double เก็บจำนวน **2.2E-308** ถึง **1.8E+308** เท่านั้น ใช้หน่วยความจำ 8 ไบต์ 7 ไบต์แรกเก็บเลขทศนิยม ไบต์สุดท้ายเก็บเลขยกกำลัง

4) ค่าคงที่ชนิดตัวอักขระ(Single Character Constant)สามารถเก็บตัวอักขระ 1 ตัวโดยใช้เครื่องหมาย ' และ ' ล้อม 1 ตัวอักขระใช้ 1 ไบต์ เช่น 'E' 'X'

5) ค่าคงที่ชนิดข้อความ (String Constant) ใช้เก็บข้อความ ความยาวตั้งแต่ 1 ตัวอักขระเก็บในรูปอาร์เรย์ แต่ละตัวใช้หน่วยความจำ 1 ไบต์ เรียงต่อกันโดยไบต์สุดท้ายต้องเก็บ \0 (Null Character) เพื่อบอกว่าจบข้อความแล้ว เช่น "Microcontroller" ใช้หน่วยความจำ 16 ไบต์

4.2.3 ชนิดของตัวแปร (Variables)

เป็นชื่อที่ตั้งเพื่อใช้อ้างอิงถึงข้อมูลต่าง ๆ โดยตัวแปรจะมีการใช้เนื้อที่ในหน่วยความจำในปริมาณที่ต่างกันขึ้นกับชนิดของข้อมูล และข้อมูลพื้นฐานในภาษาซี มี 5 ชนิด คือ อักขระ (Char) จำนวนเต็ม (Int) จำนวนจริง (Float) จำนวนจริงละเอียด 2 เท่า (Double) ไม่ให้ค่าใด ๆ (Void) นอกจากนี้เพื่อความสามารถในการใช้งานจึงมีการเพิ่มชนิดของตัวแปรขึ้นมาดังตารางที่ 10.2

ตารางที่ 10.2 แสดงชนิดของตัวแปร

คำประกาศชนิดของตัวแปร	เครื่องหมาย	จำนวนไบต์ที่ใช้	ค่าที่เป็นไปได้
char	คิดเครื่องหมาย	1	-128 ถึง 128
int	คิดเครื่องหมาย	2	-32768 ถึง 32767
short	คิดเครื่องหมาย	2	-32768 ถึง 32767
long	คิดเครื่องหมาย	4	-2147483648 ถึง 2147483647
unsigned char	ไม่คิดเครื่องหมาย	1	0 ถึง 255
unsigned int	ไม่คิดเครื่องหมาย	2	0 ถึง 65535
unsigned short	ไม่คิดเครื่องหมาย	2	0 ถึง 65535
unsigned long	ไม่คิดเครื่องหมาย	4	0 ถึง 4294967295
float	คิดเครื่องหมาย	4	3.4E-38 ถึง 3.4E+38
double	คิดเครื่องหมาย	8	1.7E-308 ถึง 1.7E+308
long double	คิดเครื่องหมาย	10	3.4E-4932 ถึง 1.1E+4932

4.2.4 การตั้งชื่อตัวแปร ในภาษาซียังมีข้อกำหนดในการตั้งชื่อของตัวแปรดังนี้

1) ต้องขึ้นต้นด้วยตัวอักษร A-Z หรือ a-z หรือเครื่องหมาย _ (Underscore) เท่านั้น ความยาว ไม่เกิน 31 ตัว

2) ภายในชื่อตัวแปรสามารถใช้ตัวอักษร A-Z หรือ a-z หรือตัวเลข 0-9 เครื่องหมาย_ (Underscore)

3) ภายในชื่อห้ามเว้นช่องว่าง หรือใช้สัญลักษณ์นอกเหนือจากข้อ 2

4) ตัวอักษรเล็กหรือใหญ่มีความหมายแตกต่างกัน

5) ชื่อตัวแปรควรสื่อความหมายของตัวแปรเพื่อป้องกันความสับสนของการพิจารณาโปรแกรม

6) ห้ามตั้งชื่อซ้ำกับคำสงวน (Reserved Word) ในภาษาซี ซึ่งมี 33 คำ ดังนี้

auto default float register type struct

break	do	for	return	union	while
case	double	goto	short	unsigned	static
char	else	if	signed	void	switch
const	enum	int	sizeof	volatile	long
continue	extern				

4.2.5 การประกาศตัวแปร

ในการใช้งานตัวแปรต้องมีการประกาศชนิดและชื่อของตัวแปรนั้นก่อน การประกาศตัวแปรใช้รูปแบบ คือ **ชนิดของตัวแปร ชื่อตัวแปร** โดยถ้ามีตัวแปรชนิดเดียวอาจประกาศพร้อมกันโดยใช้เครื่องหมายคอมมา “,” คั่นระหว่างชื่อของตัวแปร ถ้ามีการกำหนดค่าให้ใช้เครื่องหมาย = และใช้เครื่องหมายแสดงการจบคำสั่งเมื่อสิ้นสุดคำสั่ง

ตัวอย่าง

```
char name, day = 'S' , surname[20] = "MCS51";
int  x=5 ,y,z[100];      float  a=5.00 ,b,c;      double  k=1.234567, m ;
```

4.3 ตัวดำเนินการ (Operator)

การดำเนินการในการเขียนโปรแกรมภาษาซีมีอยู่ 3 ประเภท คือการคำนวณทางคณิตศาสตร์ การดำเนินการทางตรรกศาสตร์ และการเปรียบเทียบ ซึ่งการดำเนินการแต่ละประเภทจะมีเครื่องหมายที่ต้องใช้เพื่อเขียนคำสั่งสำหรับการดำเนินการประเภทนั้น ๆ ดังรายละเอียด

4.3.1 ตัวดำเนินการทางคณิตศาสตร์ (Mathematical Operators) มีเครื่องหมาย และการแสดงความหมายในภาษาซี รายละเอียดแสดงดังตารางที่ 10.3

ตารางที่ 10.3 แสดงเครื่องหมายตัวดำเนินการทางคณิตศาสตร์

สัญลักษณ์	การดำเนินการ	ตัวอย่าง
+	การบวก	2+5 ผลลัพธ์ 7
-	การลบ	7-4 ผลลัพธ์ 3
*	การคูณ	2*6 ผลลัพธ์ 12
/	การหาร	8/2 ผลลัพธ์ 4
%	การหารหาเศษ	9%4 ผลลัพธ์ 1

4.3.2 ตัวดำเนินการความสัมพันธ์หรือการเปรียบเทียบ (Relational Operators) มีเครื่องหมายและการแสดงความหมายในภาษาซี รายละเอียดแสดงดังตารางที่ 10.4

4.3.3 ตัวดำเนินการตรรกะ (Logical Operators) มีเครื่องหมาย และการแสดงความหมายในภาษาซี รายละเอียดแสดงดังตารางที่ 10.5

ตารางที่ 10.4 แสดงเครื่องหมายตัวดำเนินการความสัมพันธ์หรือการเปรียบเทียบ

สัญลักษณ์	การดำเนินการ	ตัวอย่าง
<	น้อยกว่า	2<3 ผลลัพธ์ จริง(1)
>	มากกว่า	2>3 ผลลัพธ์ เท็จ(False)(0)
<=	น้อยกว่าหรือเท่ากับ	2<= 3 ผลลัพธ์ จริง(True)
>=	มากกว่าหรือเท่ากับ	2>= 3 ผลลัพธ์ เท็จ(False)
==	เท่ากับ	4==4 ผลลัพธ์ จริง(True)
!=	ไม่เท่ากับ	2!= 2 ผลลัพธ์ เท็จ(False)

ตารางที่ 10.5 แสดงเครื่องหมายตัวดำเนินการตรรกะ

สัญลักษณ์	การดำเนินการ	ตัวอย่าง
&&	และ(AND)	(2<3)&&(3>1) ผลลัพธ์ จริง
||	น้อยกว่า(OR)	(2>3)||(4<1) ผลลัพธ์ เท็จ(False)
!	ไม่(NOT)	!(2> 3) ผลลัพธ์ จริง(True)

4.3.4 ตัวดำเนินการกำหนดค่า (Assignment Operators) รายละเอียดแสดงดังตารางที่ 10.6 เป็นการแสดงเครื่องหมาย และความหมายในภาษาซี

ตารางที่ 10.6 แสดงเครื่องหมายตัวดำเนินการกำหนดค่า

สัญลักษณ์	การดำเนินการ	ตัวอย่าง
=	กำหนดค่า (Assignment)	a=2 ความหมายคือ กำหนดให้a มีค่าเป็น2
+=	การบวก (Addition)	a+=b ความหมายคือ(a=a+b)
*=	การคูณ (Multiplication)	a*=b ความหมายคือ(a=a*b)
-=	การลบ (Subtraction)	a-=b ความหมายคือ(a=a-b)
/=	การหาร (Division)	a/=b ความหมายคือ(a=a/b)
%=	การหารหาเศษ (Remainder)	a%=b ความหมายคือ(a=a%b)
++	เพิ่มค่า (Increment)	a++ หรือ ++a ความหมายคือ a=a+1
--	ลดค่า (Decrement)	a-- หรือ --a ความหมายคือ a=a-1

4.3.5 ลำดับการดำเนินการของตัวดำเนินการ (Operator Precedence) มีเครื่องหมาย และการแสดงความหมายในภาษาซี รายละเอียดแสดงดังตารางที่ 10.7

ตารางที่ 10.7 แสดงลำดับการดำเนินการของตัวดำเนินการ

ลำดับที่	ตัวดำเนินการ	ลักษณะการทำงาน
1	**( ) [ ] . ->**	ซ้าย ไป ขวา
2	**- ~ | * &**	ขวา ไป ซ้าย
3	**++ --**	ขวา ไป ซ้าย
4	*** / %**	ซ้าย ไป ขวา
5	**+ -**	ซ้าย ไป ขวา
6	**<< >>**	ซ้าย ไป ขวา
7	**< > <= >=**	ซ้าย ไป ขวา
8	**== !=**	ซ้าย ไป ขวา
9	**&(bitwise AND)**	ซ้าย ไป ขวา
10	**^(bitwise XOR)**	ซ้าย ไป ขวา
11	**|(bitwise OR)**	ซ้าย ไป ขวา
12	**&&**	ซ้าย ไป ขวา
13	**||**	ซ้าย ไป ขวา
14	**?:**	ซ้าย ไป ขวา
15	**= += -= /= %=**	ขวา ไป ซ้าย
16	**<<= >>=**	ขวา ไป ซ้าย

โดยตัวดำเนินการที่มีลำดับน้อยกว่าจะดำเนินก่อนตัวดำเนินการที่มีลำดับสูงกว่า เช่น X = 2 + 5 * 3 จะมีลำดับการดำเนินการ คือ

ลำดับที่ 1 5 * 3 (เพราะ * มีลำดับเป็น 4 ส่วน + อยู่ลำดับ 5)

ลำดับที่ 2 2 + 15

ลำดับที่ 3 17 เป็นค่าของ X

4.4 นิพจน์ (Expression)

เกิดจากการนำค่าคงที่ หรือตัวแปร และตัวดำเนินการมาประกอบกันนิพจน์มีหลายชนิด เช่น นิพจน์ทางคณิตศาสตร์ นิพจน์ทางตรรกะ นิพจน์ทางการเปรียบเทียบ นิพจน์เกี่ยวกับเงื่อนไข นิพจน์เกี่ยวกับ

ข้อความ เช่น 2 + x * 5 เป็นตัวอย่างของนิพจน์ทางคณิตศาสตร์ (2>3)&&(5>=4) เป็นนิพจน์ทางตรรกะ และการเปรียบเทียบ !(a) เป็นนิพจน์ทางตรรกะ if (x==y) เป็นนิพจน์เงื่อนไข

ตัวอย่าง การหาผลลัพธ์ของนิพจน์

กำหนด int a = 10 , b = 12, c =3; จงหาผลลัพธ์ของ 2 * a + b % c ดำเนินการดังนี้ (2 * a) + (b %c) ผลลัพธ์ คือ 20 + 0 คือ 20

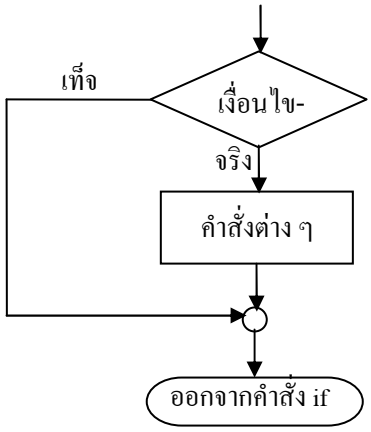
จากข้อกำหนดด้านบน จงหาผลลัพธ์ ของ (a>b)&&(c<=b) ซึ่งเราอาจพิจารณา ดังนี้ (เท็จ) && (จริง) ดังนั้น ผลลัพธ์ คือ เท็จ

4.5 คำสั่งควบคุมการทำงานของโปรแกรม

คำสั่งควบคุมการทำงานของโปรแกรม กลุ่มที่มีการทดสอบเงื่อนไขก่อนการตัดสินใจเพื่อทำงานต่อไปตามคำสั่งที่กำหนดไว้ มีหลายคำสั่ง คือ if, if else , if else if คำสั่งเหล่านี้จะมีลักษณะคล้ายกันที่ต้องมีการทดสอบเงื่อนไขเพื่อเลือกเลือกคำสั่งที่จะทำงานต่อไป

4.5.1 **คำสั่ง if**

คำสั่งนี้เป็นคำสั่งที่มีการทดสอบเงื่อนไขก่อนที่จะทำงานตามคำสั่งที่กำหนด คำสั่งของ if เขียนเป็นผังงานแสดงได้ดังภาพที่ 10.17



ภาพที่ 10.17 แสดงการทำงานของคำสั่ง if

รูปแบบของคำสั่ง if เป็น ดังนี้

```
if (expression) statement ;
หรือ
If (expression)
{
```

```
statement 1;
...
statement n;
}
```

การแสดงความหมายในคำสั่ง if เป็นเงื่อนไขที่มีค่าได้เพียง จริง หรือ เท็จ เท่านั้น ถ้ามีค่าเป็นจริงจะทำตามคำสั่งใน if จากนั้นจะออกไปทำตามคำสั่งนอกคำสั่ง if ถ้าเงื่อนไขมีค่าเป็นเท็จ จะไม่ทำตามคำสั่งใน if

ตัวอย่าง การใช้คำสั่ง if

```
#include <stdio.h>
#include <conio.h>
int xh;
main()
{
   clrscr();
   printf("Please press any key.");
   ch = getche();
   if (ch == '\r') printf("\n Enter key ( ASCII code = 13 ) was pressed .");
   printf("\n Out of if statement");
}
```

คำสั่ง if (ch == '\r') printf("\n Enter key (ASCII code = 13) was pressed ."); กำหนดให้ตรวจสอบเงื่อนในคำสั่ง คือ ถ้ามีการเคาะแป้นจากคำสั่ง ch = getche(); เป็นแป้น enter จริง แล้วให้ทำตามคำสั่งใน if คือ คำสั่ง ให้แสดงข้อความว่า Enter key (ASCII code = 13) was pressed . ออกทางหน้าจอ แล้วจึงทำตามคำสั่งนอกคำสั่ง if คือ แสดงข้อความว่า Out of if statement

4.5.2 **คำสั่ง if else**

เป็นคำสั่งที่มีการทดสอบเงื่อนไขแบบ 2 ทางเลือก ถ้าเงื่อนไขเป็นจริง ให้ทำตามคำสั่งชุด (A) ที่อยู่ใน if ถ้าเงื่อนไขเป็นเท็จ ให้ทำตามคำสั่งชุดใน else (ชุดB) การทำงานของคำสั่ง if else เขียนเป็นผังงานแสดงได้ดังภาพที่ 10.18

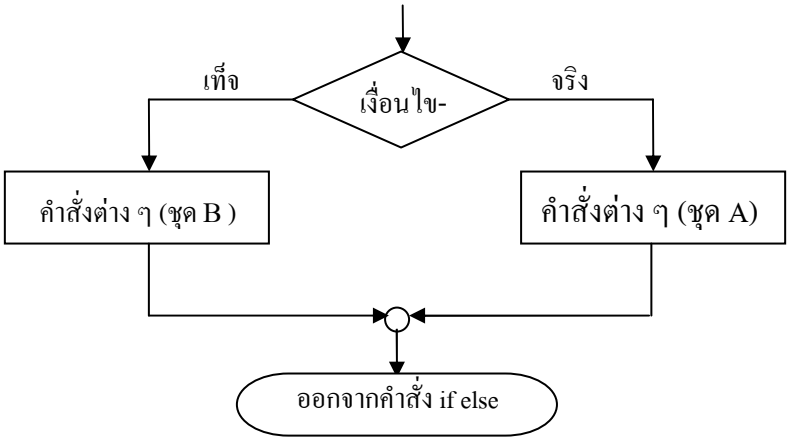
รูปแบบของคำสั่ง if else

คำสั่ง if else มีรูปแบบ ดังนี้

```
if (expression) {
statement A1;
```

```
…
statement An;
}
else      {
statement B1;
…
statement Bn;
}
```

โดยการแสดงความหมาย (Expression) ใน if มีค่าได้เพียง จริง หรือ เท็จ ถ้ามีค่าเป็นจริง โปรแกรมจะทำงานต่อไปในคำสั่งหลัง if คือ คำสั่งชุด A เสร็จแล้วออกจาก if โดยไม่ทำตามคำสั่งชุด B ถ้า Expression มีค่าเป็นเท็จ โปรแกรมจะทำตามคำสั่งหลัง else คือ คำสั่งชุด B แล้วออกไปโดยไม่ทำตามคำสั่งชุด A



ภาพที่ 10.18 แสดงการทำงานของคำสั่ง if else

ตัวอย่าง if else

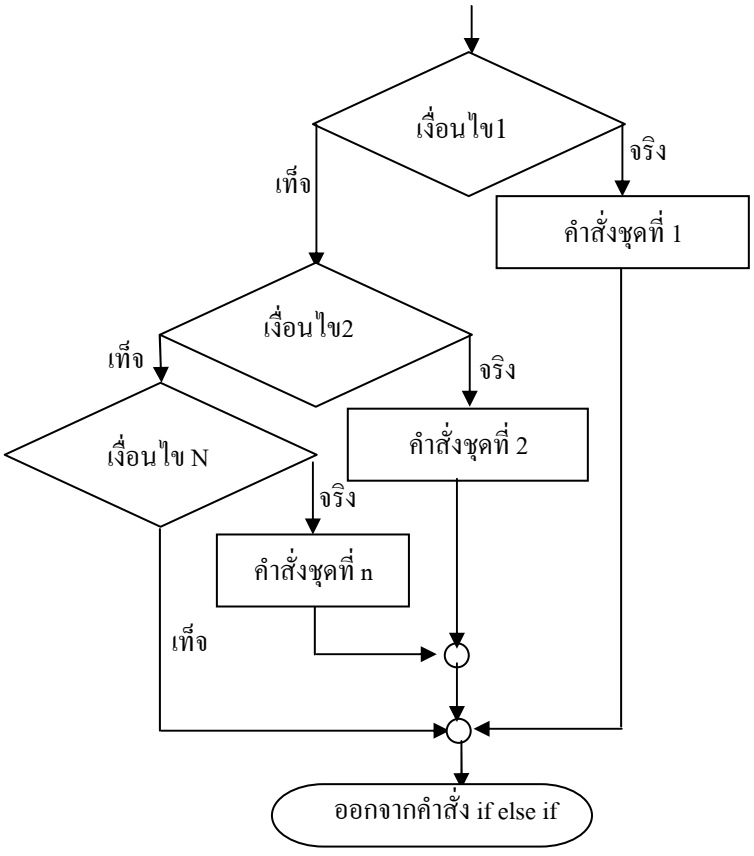
```
#include <stdio.h>
#include <conio.h>
main()
{
   int score1;
   printf("\n Please type your score  : ");
```

```
    scanf("%d",&score1);
    if (score1 >= 50) printf("You pass the examination.");
       else printf("\n You failed the examination.");
}
```

เป็นโปรแกรมรับการป้อนค่าคะแนนจากแป้นพิมพ์นำไปเก็บในตัวแปร score1 แล้วนำ score1 มาเปรียบเทียบกับ 50 ถ้ามากกว่าหรือเท่ากับ 50 จะได้ผลเป็น จริง โปรแกรมจะทำตามคำสั่งใน if คือ printf("You pass the examination."); แล้วออกจาก if โดยไม่ทำตามคำสั่งใน else แต่ถ้าเปรียบเทียบ score1 กับ 50 แล้วได้ค่าเป็นเท็จ คือ score1 น้อยกว่า 50 โปรแกรมจะทำตามคำสั่งหลัง else คือ printf("\n You failed the examination."); แล้วออกจากคำสั่งโดยไม่ทำตามคำสั่งหลัง if

4.5.3 **คำสั่ง if else if**

คำสั่งนี้มีโครงสร้าง else if เพิ่มเข้ามาในคำสั่ง else ทำให้ใช้คำสั่ง else if เพิ่มได้ตามที่ต้องการ ใช้กับการตัดสินใจที่มีทางเลือกมากกว่า 2 ทางเลือก เขียนเป็นผังงานแสดงได้ดังภาพที่ 10.19



ภาพที่ 10.19 แสดงการทำงานของคำสั่ง if else if

รูปแบบของคำสั่ง if else if

```
if (expression1)
  {
   statement A;
  }
else if (expression2)
  {
  statement B;
  }
  ...
else if (expression n)
    {
        statement N;
    }
    else
       {
           statement N + 1;
        }
```

ตัวอย่าง if else if

```
#include <stdio.h>
#include <conio.h>
main()
{
          int score1;
          clrscr();
          printf("\n Please type your score  : ");
          scanf("%d",&score1);
          if (score1 >= 80) printf("\n You get  A. \n Congratulation");
          else if (score1 >= 70) printf("\n You get B.");
               else if (score1 >= 60) printf("\n You get C.");
                    else if (score1 >= 50 ) printf("You get  D.");
```

```
else printf("\n You get E" );
}
```

โปรแกรมนี้เป็นการให้ป้อนคะแนนทางแป้นพิมพ์เพื่อหาระดับคะแนน ตามเกณฑ์และรายงานผลออกทางจอภาพ

4.6 คำสั่งการทำซ้ำหรือวนรอบ

4.6.1 **คำสั่ง for**

คำสั่ง for เป็นคำสั่งให้ทำซ้ำโดยมีเงื่อนไข ปกติใช้การเพิ่มค่าหรือลดค่าของตัวนับทำซ้ำไปเมื่อเงื่อนไขที่กำหนดเป็นจริง จนกระทั่งเมื่อเงื่อนไขเป็นเท็จให้เลิกทำ รูปแบบของคำสั่ง เป็นดังนี้

```
for (นิพจน์ที่1;นิพจน์ที่2;นิพจน์ที่3)  คำสั่ง 1 คำสั่ง
หรือ for (นิพจน์ที่1;นิพจน์ที่2;นิพจน์ที่3)
{
        คำสั่งที่ 1
        คำสั่งที่ 2
        ...
        คำสั่งสุดท้าย
}
```

นิพจน์ที่ 1 เป็นนิพจน์ที่ใช้กำหนดค่าเริ่มต้นให้กับตัวแปรที่ใช้ในการทำซ้ำ

นิพจน์ที่ 2 เป็นนิพจน์ที่ใช้เป็นเงื่อนไขมีค่าได้เพียง 1 ใน 2 ค่าเท่านั้น คือ เป็นจริง หรือเท็จ

นิพจน์ที่ 3 เป็นนิพจน์ที่กำหนดการเพิ่มหรือลดค่าของตัวแปรที่ใช้ในการทำซ้ำ

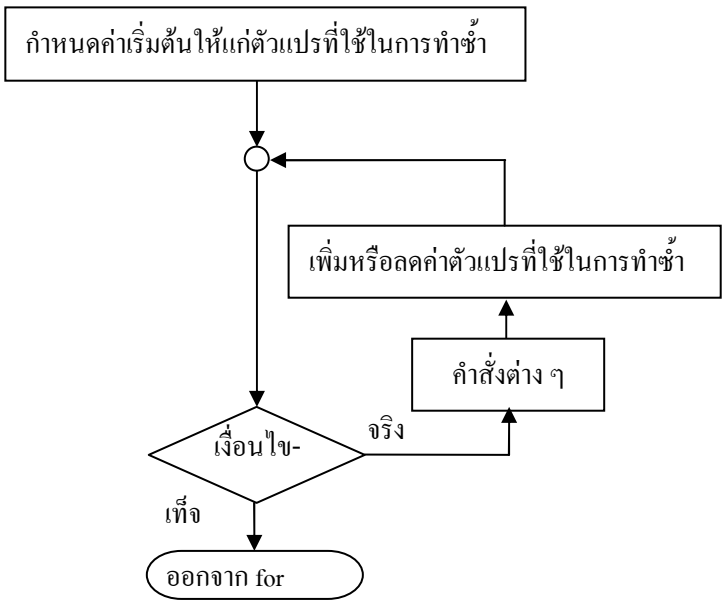
คำสั่งหลัง for ถ้ามีมากกว่า 1 คำสั่ง จะต้องอยู่ภายในเครื่องหมาย {กับ} เพื่อให้รวมเป็นคำสั่งชุดเดียว เนื่องจากในภาษาซีคำสั่งที่อยู่ในการทำซ้ำแบบ for จะต้องมีเพียง 1 คำสั่ง (คำสั่งการทำซ้ำหรือเงื่อนไขอื่นก็เป็นเช่นเดียวกัน) คำสั่ง for เขียนเป็นผังงานแสดงได้ดังภาพที่ 10.20

ตัวอย่าง การใช้คำสั่ง for บวกเลขตั้งแต่ 1 ถึง 100

```
#include<stdio.h>
main()
{
 int i,ans;
ans=0;
for(i=1;i<=100;i++){
      ans=ans+i;
```

```
}
printf("answer is %d",ans);
}
```

จากโปรแกรมทำการกำหนดให้ตัวแปร i เป็นตัวแปรนับ ส่วน ans เป็นตัวแปรที่ใช้เก็บค่าคำตอบ ในบรรทัด for นั้น ในขณะที่ในโปรแกรมพบคำสั่ง i++ หมายถึงให้ i+1 นั่นเอง ซึ่งหมายถึงใน loop นี้จะทำการเพิ่มตัวนับ i ไปทีละ 1 ค่า



ภาพที่ 10.20 แสดงการทำงานของคำสั่ง for

4.6.2 **คำสั่ง while**

while เป็นคำสั่งให้มีการทำซ้ำเป็นรอบ ๆ (Loop) ลักษณะการทำงานทำนองเดียวกับคำสั่ง for แต่ต่างกันตรงที่ไม่ทราบจำนวนรอบที่แน่นอน แต่ต้องมีเงื่อนไขที่เป็นเท็จจึงจะออกจากคำสั่ง while ได้ มิฉะนั้นจะมีปัญหาที่โปรแกรมทำงานแบบวนซ้ำแบบไม่มีที่สิ้นสุด (Endless Loop) การทำซ้ำแบบ while เขียนผังงานแสดงได้ดังภาพที่ 10.21 คำสั่ง while มีรูปแบบ ดังนี้

```
while (นิพจน์ทดสอบเงื่อนไข)  statement ;
```

หรือ

```
while (นิพจน์ทดสอบเงื่อนไข)
{
คำสั่งที่ 1;
```

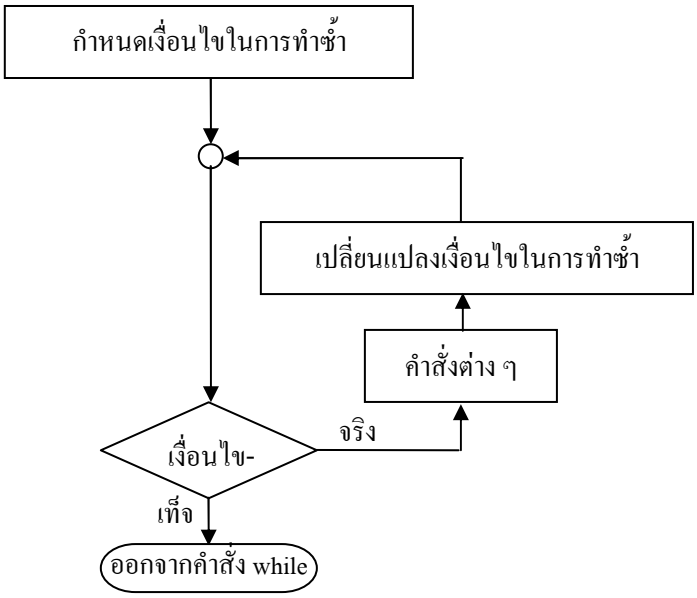
คำสั่งที่ 2;

...

คำสั่งสุดท้าย;

}

โดย while จะทำการทำซ้ำต่อไปเมื่อนิพจน์ทดสอบเงื่อนไขให้ผลลัพธ์เป็นจริง และทำต่อจนกระทั่ง ผลลัพธ์ของนิพจน์ทดสอบเงื่อนไขเป็นเท็จ



ภาพที่ 10.21 แสดงการทำงานของคำสั่ง while

ตัวอย่าง while statement

```
#include <stdio.h>
#include <conio.h>
int counter ;
char word[20] = "microcontroller";
main()
{
counter = 5;
while (counter < 11 )
{
```

```
printf("counter \t=\t%2d\tmy school is\t%s \n",counter,word);
counter++;
}
}
```

โปรแกรมนี้กำหนด เริ่มต้นให้ counter มีค่าเป็น 5 แล้ว จึงเข้าคำสั่งการทำซ้ำ แบบ while เพื่อทดสอบว่าเงื่อนไข คือ counter < 11 เป็นจริง หรือไม่ ถ้าเป็นจริง จะทำตามคำสั่ง Printf ("counter \t=\t%2d\tmy school is\t%s \n",counter,word) ; แล้วจึงทำตามคำสั่ง counter ++ ซึ่งเป็นการเพิ่มค่า ของ counter จากเดิม แล้วตรวจสอบเงื่อนไข ถ้าเป็นจริงก็ทำต่อไปจนกระทั่งนิพจน์เงื่อนไข เป็นเท็จ คือ counter ไม่น้อยกว่า จึงออกจากการทำซ้ำ ในตัวอย่างนี้มีการทำซ้ำ 6 รอบ

4.6.3 **คำสั่ง do while**

do while เป็นคำสั่งให้มีการทำซ้ำเป็นรอบ ๆ (loop) ลักษณะการทำงานคล้ายกับคำสั่ง while แต่ต่างกันตรงที่คำสั่งนี้มีการทำงานตามคำสั่งไป 1 รอบ ก่อนที่จะทดสอบเงื่อนไข ถ้าเงื่อนไขที่เป็นจริงจะทำงานต่อไป ถ้าเงื่อนไขเป็นเท็จจึงออกจากคำสั่ง do while คำสั่งนี้ต้องกำหนดให้มีโอกาสที่เงื่อนไขเป็นเท็จได้ มิฉะนั้นจะมีปัญหาที่โปรแกรมทำงานแบบวนซ้ำแบบไม่มีที่สิ้นสุด (Endless Loop) กรณีที่เงื่อนไขเป็นเท็จเพียงอย่างเดียวจะมีการทำงาน 1 รอบ ก่อนออกจากการทำซ้ำ การทำซ้ำแบบ do whileเขียนผังงานแสดงได้ดังภาพที่ 10.22 คำสั่ง do while มีรูปแบบ ดังนี้

```
do {
คำสั่งที่ 1;
คำสั่งที่ 2;
...
คำสั่งสุดท้าย;
} while (นิพจน์ทดสอบเงื่อนไข) ;
```

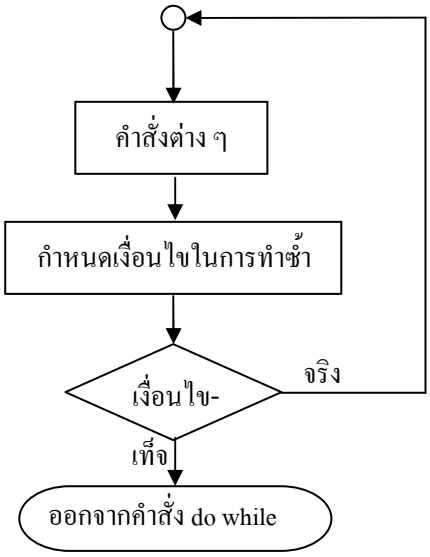
โดย do while จะทำการทำซ้ำต่อไปเมื่อนิพจน์ทดสอบเงื่อนไขให้ผลลัพธ์เป็นจริง และทำต่อจนกระทั่งผลลัพธ์ของนิพจน์ทดสอบเงื่อนไขเป็นเท็จ ก็จะออกจากการทำซ้ำ

ตัวอย่าง do while statement

```
#include <stdio.h>
#include <conio.h>
int counter ;
char word[20] = "Bodindecha";
main()
{
```

```
    clrscr();
    counter = 11;
    do                                  /* start do while */
      {
          printf("counter \t=\t%2d\tmy school is\t%s \n",counter,word);
          counter++;
      } while (counter < 11 )           /* end do while */
}                                       /* end main() */
```

โปรแกรมนี้ จะลบจอภาพ ก่อน แล้วจึงกำหนด เริ่มต้น ให้ counter มีค่าเป็น 11 แล้ว จึงเข้าสู่การทำซ้ำ แบบ do while ทำตามคำสั่ง printf ("counter \t=\t%2d\tmy school is\t%s \n",counter,word); ไปก่อนแล้วจึงทดสอบว่าเงื่อนไข คือ counter < 11 เป็นจริง หรือไม่



ภาพที่ 10.22 แสดงการทำงานของคำสั่ง do while

4.7 การใช้งานตัวแปร sbit กับ XBYTE

4.7.1 การใช้งานตัวแปร sbit

การกระทำกับข้อมูลภายในรีจิสเตอร์ของไอซี MCS-51 สามารถกระทำครั้งละ 8 บิต แต่ในการเขียนโปรแกรมที่กระทำกับข้อมูลในตำแหน่งที่เข้าได้แบบบิต ตัวแปล C51 ได้เตรียมวิธีให้ดังนี้

1) เริ่มต้นโปรแกรมต้องประกาศตัวแปรไว้พื้นที่หน่วยความจำตำแหน่งเข้าได้แบบบิต

2) ต้องทำการประกาศการใช้งานแบบบิต พร้อมกำหนดว่าจะใช้บิตที่เท่าไรจากตัวแปรที่ประกาศในข้อที่ 1

3) นำตัวแปรที่ได้จากข้อ 2 ใช้งานได้เหมือนกับตัวแปรทั่วไป

หากต้องการตรวจสอบเครื่องหมายของข้อมูลชนิดตัวอักษร (Char) ว่าเป็นบวกหรือลบ ให้ตรวจสอบข้อมูลบิตที่ 7 หากข้อมูลมีค่าเป็น 0 แสดงว่าเป็น บวก หากข้อมูลเป็น 1 แสดงว่าเป็นลบ

BDATA เป็นข้อมูลตำแหน่งของหน่วยความจำภายในที่เข้าแบบบิตตำแหน่งแอดเดรส 1FH - 20H (ในภาษาซีเขียนเลขฐานสิบหกได้เป็น 0x1F -0xFF)

ตัวอย่าง การเขียนโปรแกรมคำสั่ง sbit

```
#include <reg51.h>
bdata char test;          /*ตัวแปร test ถูกเก็บไว้ในหน่วยความจำข้อมูลส่วน BDATA*/
sbit   sign = test^7;     //กำหนดให้ sign อยู่ในตำแหน่งบิตที่ 7 ใน test
void main(void){
test = -1;        //กำหนดให้ test มีค่าเป็น ลบ
      if(sign==1){      //ตรวจสอบบิต ที่ 7 ถ้า sing มีค่าเป็น 1
        test =1    }       // จากคำสั่ง if ถ้า sing เป็น 1 ให้ตัวแปร test เป็นบวก
}
```

4.7.2 การใช้งานตัวแปร XBYTE

การติดต่อรับ ส่งข้อมูลกับอุปกรณ์ภายนอกของ MCS-51 จะทำผ่านพอร์ต ในภาษาซีกำหนดให้เราอ้างแอดเดรสของพอร์ตผ่านตัวแปรมาโครภายในโปรแกรมได้ โดยมาโครเหล่านี้ได้แก่ XBYTE ซึ่งเก็บอยู่ในไฟล์ absacc.h ดังนั้นก่อนการใช้งานต้องกำหนด #include <absacc.h> ที่ส่วนหัวของโปรแกรม การใช้งานจะต้องกำหนดตัวแปรขึ้นมาแทนพอร์ตแอดเดรสที่ต้องการ โดยมีรูปแบบดังนี้

#define variable XBYTE[adress16]

variable ตัวแปรที่แทนพอร์ตแอดเดรสโดยต้องกำหนดอยู่หลัง define

XBYTE [address16] มาโครสำหรับกำหนดแอดเดรสพอร์ตโดย address16 คือแอดเดรสขนาด 16 บิตต้องเขียนในรูปเลขฐานสิบหก

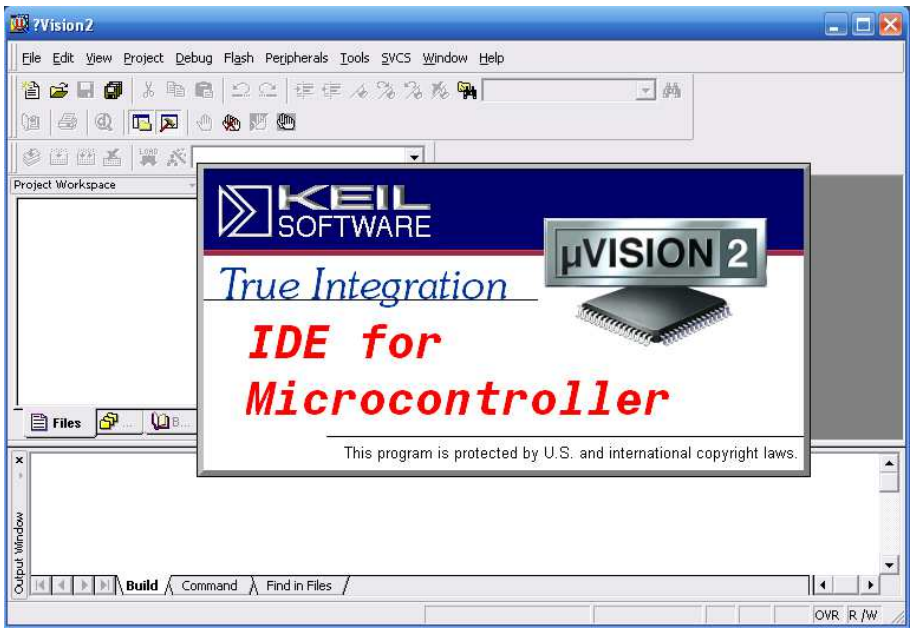
ตัวอย่าง การเขียนโปรแกรม XBYTE

```
#include<reg51.h>
#include<absacc.h>   /* อ้างมาโครภายในในไฟล์ absacc * /
#define portA XBYTE[0xA000]        //การกำหนดตัวแปรใช้แทนพอร์ตแอดเดรส
#define portb XBYTE[0xA001]        //ผ่านมาโคร XBYTE
#define portc XBYTE[0xA002]
#define portcon XBYTE[0xA003]
void delay(int count);             //ประกาศใช้งานฟังก์ชั่น delay
```

```
void main (void){                    //โปรแกรมเริ่มต้นทำงานที่นี้
      portcon =  0x81;               //กำหนดค่าควบคุมพอร์ตให้กับ portcon
      portB    =  0x02;              //ส่งค่า 0x02 ออกทาง portB
      while(1){                      //วนทำใน loop while ไม่รู้จบ
         portC  =  0x7f;             //กำหนดการส่งค่า 0x7f ให้กับ portC
        portA  =  portC;             //ส่งค่าใน portC  ให้กับ portA
         delay(10) ; }
      }
void delay(int count)                //โปรแกรมหน่วงเวลา
{
 int  i,j;
 for (i=0;i<count;i++)
 for  (j=0;j<500;j++)
 }
```

4.8 ตัวอย่างการใช้งานโปรแกรม Keil µVision 2

โปรแกรม KEIL C51 สร้างขึ้นโดยบริษัท KEIL Software ใช้เขียนโปรแกรมภาษาซี แสดงดังภาพที่ 10.23 มีแอสเซมเบลอร์ คอมไพเลอร์ ลิงค์เกอร์ และดีบักเกอร์ ไว้ในโปรแกรมเดียวกัน สามารถเขียนโปรแกรมแกรมคำสั่ง และแปลงให้เป็นภาษาเครื่อง (HEX) เพื่อบันทึกลงในตัวไอซีได้

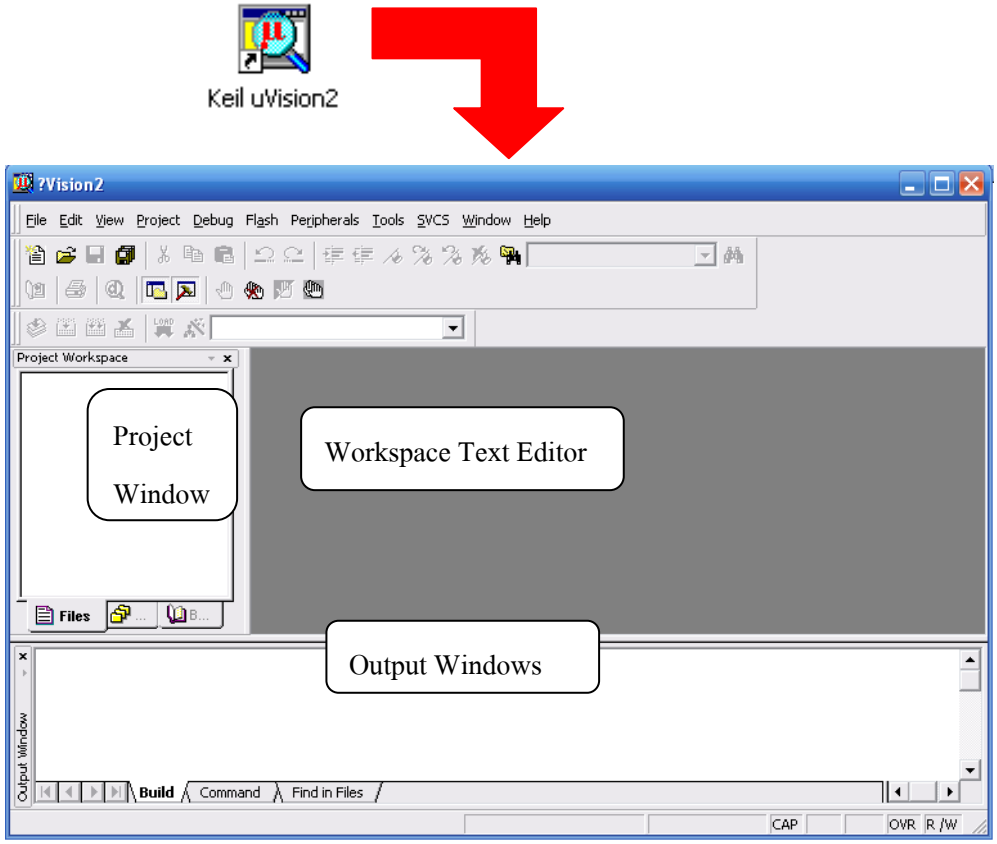


ภาพที่ 10.23 แสดงโปรแกรม KEIL µVISION 2

การเขียนโปรแกรมคำสั่งภาษาซีกับไอซี MCS-51 โดยโปรแกรม KEIL µVISION 2 มีลำดับขั้นตอนใช้งานดังนี้

ขั้นตอนที่ 1 สร้างโฟลเดอร์สำหรับเก็บไฟล์ใช้ในการเขียนโปรแกรม ตัวอย่างเช่น **"MCS51"**

ขั้นตอนที่ 2 เปิดโปรแกรม KEIL µVISION 2 โดยคลิกที่ไอคอน Keil µVision2 หรือไปที่ Start →All Program →Keil µVision 2 หน้าต่างโปรแกรมแสดงดังภาพที่ 10.24



ภาพที่ 10.24 แสดงหน้าต่างหลักของโปรแกรม KEIL µVISION 2

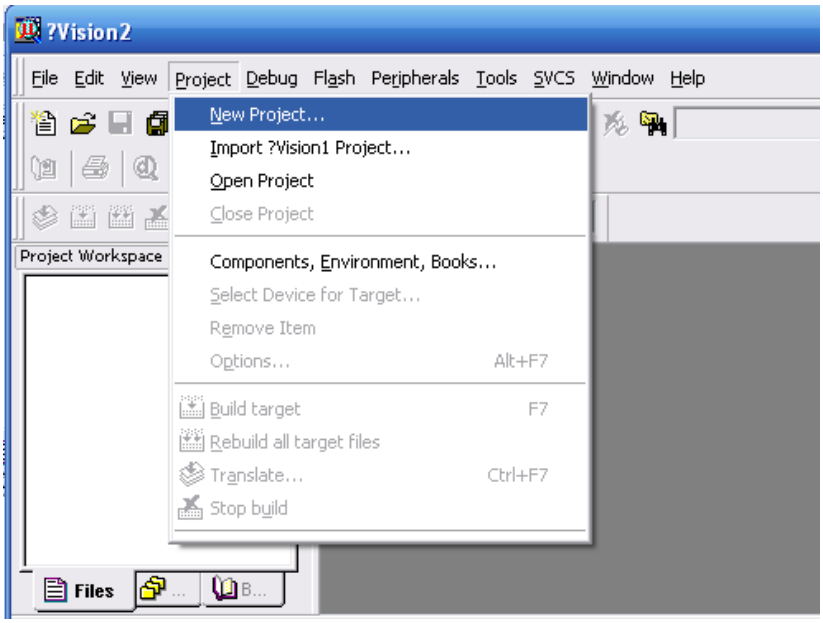
หน้าต่างของโปรแกรมประกอบด้วย

1) Project Windows สำหรับสร้างโปรแกรม และแปลคำสั่งในโปรแกรมให้อยู่ในรูป HEX ไฟล์

2) Workspace Text Editor เป็นเท็กซ์เอดิเตอร์ ใช้สำหรับเขียนโปรแกรมคำสั่ง

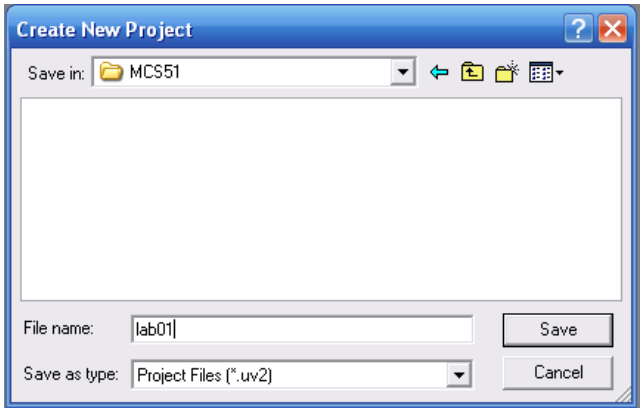
3) Output Windows เป็นตัวตรวจสอบความผิดพลาดของโปรแกรมที่เขียนขึ้น

ขั้นตอนที่ 3 ไปที่เมนู Project → New Project จะได้หน้าต่าง Create New Project แสดงดังภาพที่ 10.25



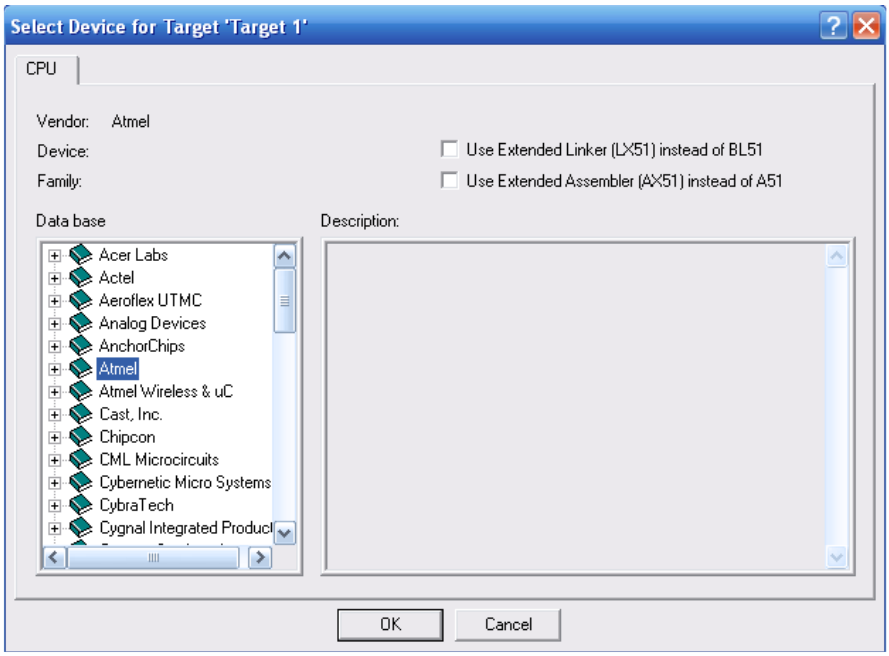
ภาพที่ 10.25 แสดงการใช้เมนูเพื่อสร้าง Project

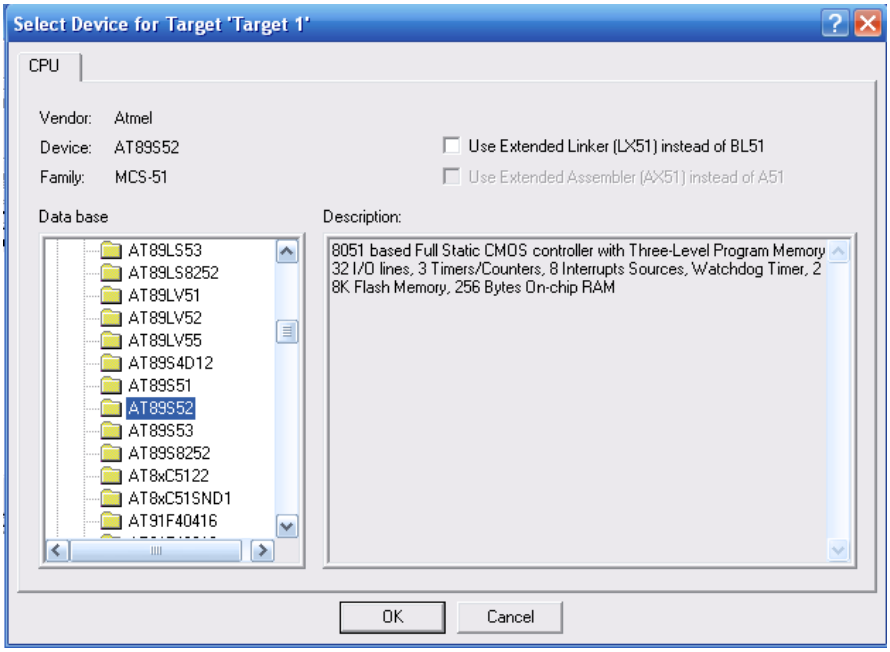
เลือกในโฟลเดอร์ MCS51 ที่สร้างขึ้นจากขั้นตอนที่ 1 จากนั้นกำหนดชื่อผังงาน หรือ Project ใหม่ ตัวอย่าง ตั้งชื่อเป็น LAB01 เลือกที่ชื่อไฟล์ เลือกที่ปุ่ม Save แสดงดังภาพที่ 10.26

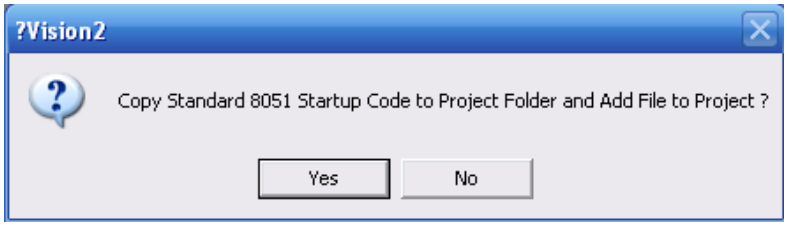


ภาพที่ 10.26 แสดงการสร้าง Project

ขั้นตอนที่ 4 เข้าสู่หน้าต่าง Select Device for Target ‘Target1’ แล้วให้เลือกบริษัท และเบอร์ของไอซีไมโครคอนโทรลเลอร์ MCS-51 ที่ใช้งาน ตัวอย่างเช่น เลือกบริษัท Atmel แล้วดับเบิลคลิก เลือกเบอร์ AT89S52 เลือกที่ปุ่ม Ok จะมีคำถามจากไดอะลอกบล็อก ว่าต้องการคัดลอกสร้างไฟล์มาตรฐานของ MCS-51 มาใช้หรือไม่ แสดงดังภาพที่ 10.27 โดยให้ตอบ Yes จะได้โฟล์เดอร์ของโปรเจคไฟล์ในช่อง File ในหน้าต่างของ Project Workspace

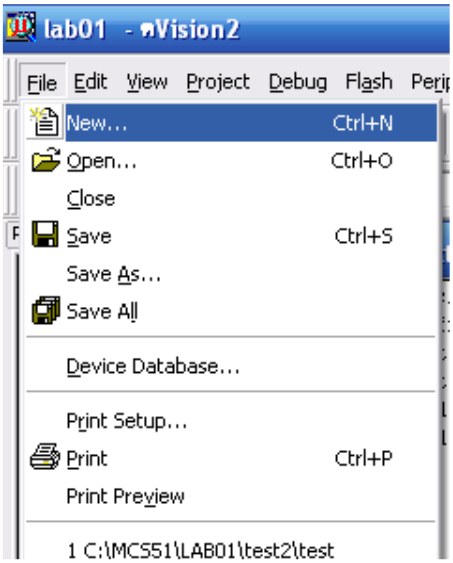


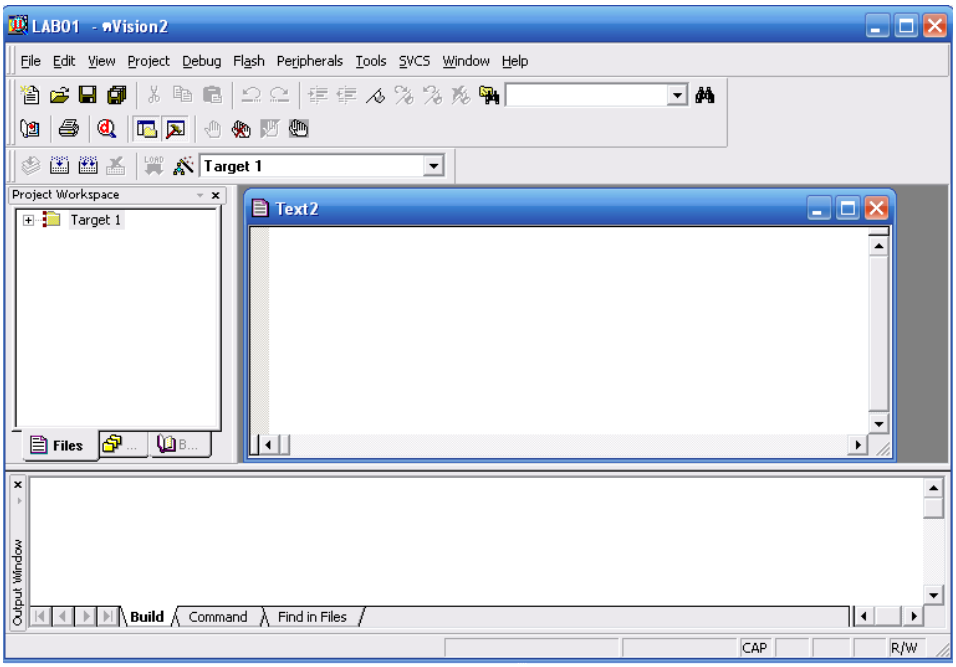




ภาพที่ 10.27 เลือกผู้ผลิต เบอร์ไอซี และคัดลอกสร้างไฟล์มาตรฐาน

ขั้นตอนที่ 5 เลือกที่เมนู File →New ได้หน้าต่างสำหรับการเขียนโค๊ดโปรแกรม แสดงดังภาพที่ 10.28





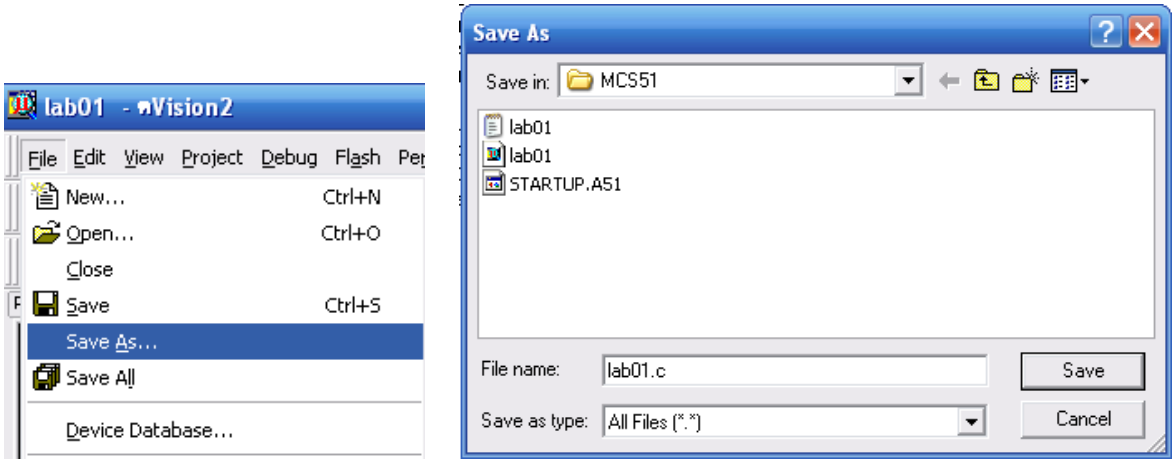
ภาพที่ 10.28 แสดงหน้าต่างสำหรับการเขียนรหัสโปรแกรมคำสั่ง

ขั้นตอนที่ 6 ทำการเขียนรหัสโปรแกรมคำสั่ง ดังนี้

```
#include <reg51.h>
#define TRUE        1
sbit P0_0 = P0^0;       // Port P0.0
```

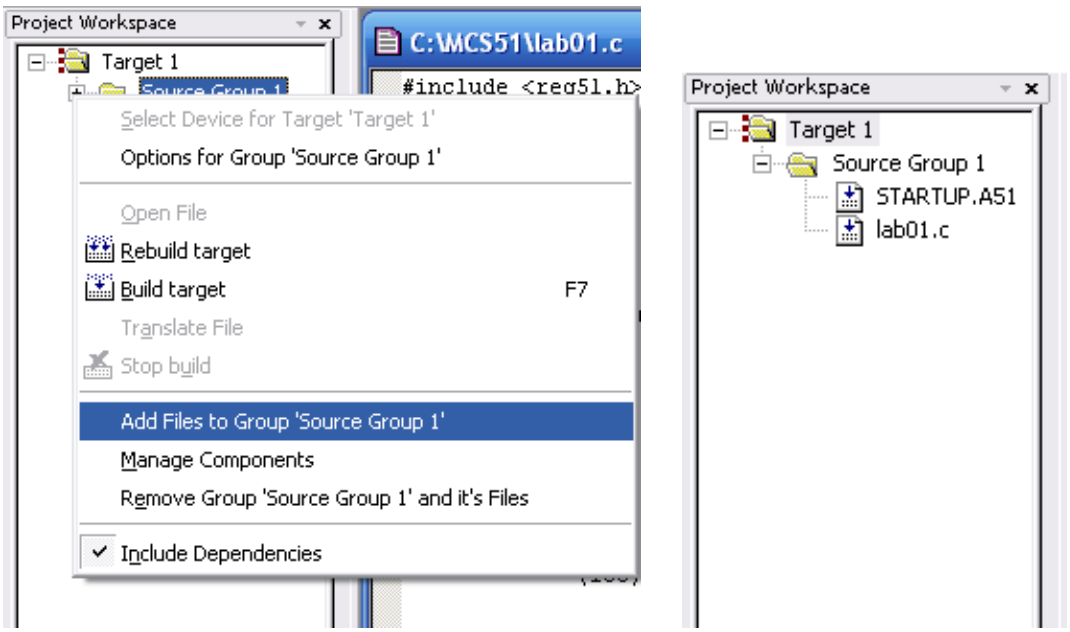
```
sbit dri_p = P1^4;
void delay(int i);
void delay(int i)        {
int j;
for(;i>0;i--)
for(j=0;j<1000;j++);
                         }
void main(void)          {
dri_p = 1;               // driver select
P0 =0x00;                // clear port P0
while(TRUE)              {
P0_0 = 1;      // high port P0.0
delay(100);    // delay
P0_0 = 0;      // low port P0.0
delay(100);    // delay
      }
      }
```

ขั้นตอนที่ 7 เลือกที่เมนู File → Save As ที่ช่อง File Name ใส่ชื่อระบุนามสกุล .C ด้วยในตัวอย่างใช้ชื่อ LAB01.C แสดงดังภาพที่ 10.29



ภาพที่ 10.29 การบันทึกไฟล์ให้เป็นภาษาซี

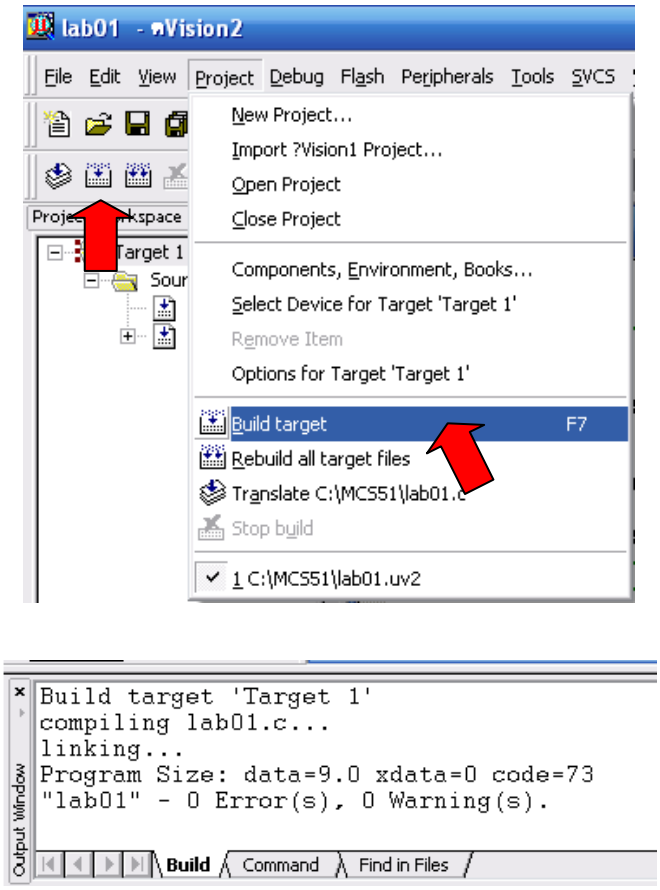
ขั้นตอนที่ 8 ใช้เมาส์ปุ่มขวาคลิกไปที่ Source Group1 ในหน้าต่าง Project Workspace ที่เมนูย่อยเลือก Add File to Group ‘Source Group1’ เลือกไฟล์ LAB01.C คลิกปุ่ม Add หลังจากนั้นให้เลือกปุ่ม Close ไฟล์ที่เพิ่มเข้า แสดงอยู่ใน Source Group1 ในหน้าต่าง Project Workspace แสดงดังภาพที่ 10.30



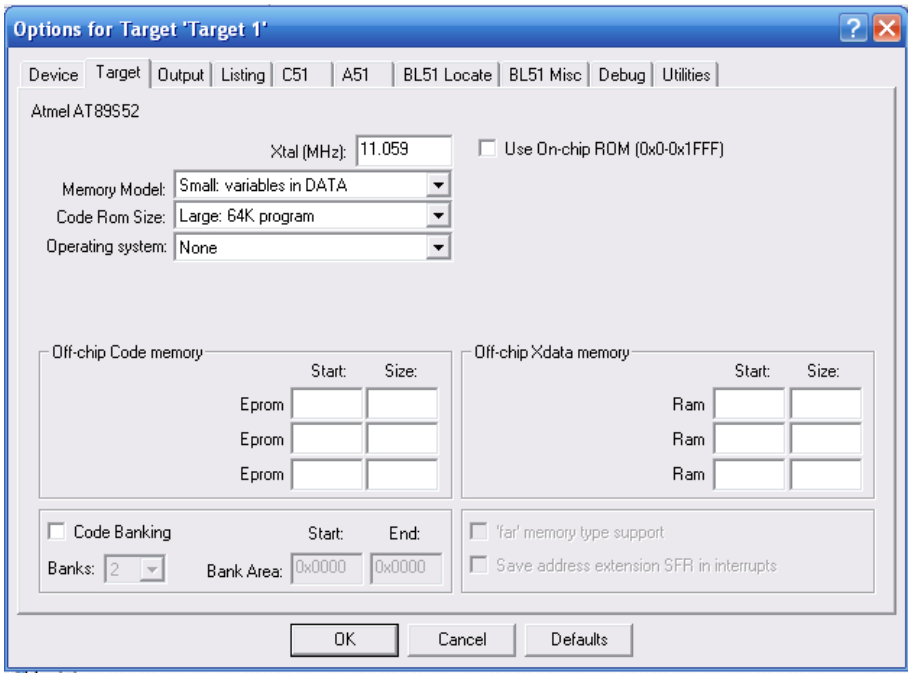
ภาพที่ 10.30 การเพิ่มไฟล์เข้าไปใน Source Group1

ขั้นตอนที่ 9 ทำการคอมไพล์โปรแกรมโดยไปที่เมนู Project → Built Target หรือกดที่คีย์ F7 ถ้าไม่พบข้อผิดพลาดใดๆ ที่ หน้าต่าง Output Windows แสดงดังภาพที่ 10.31

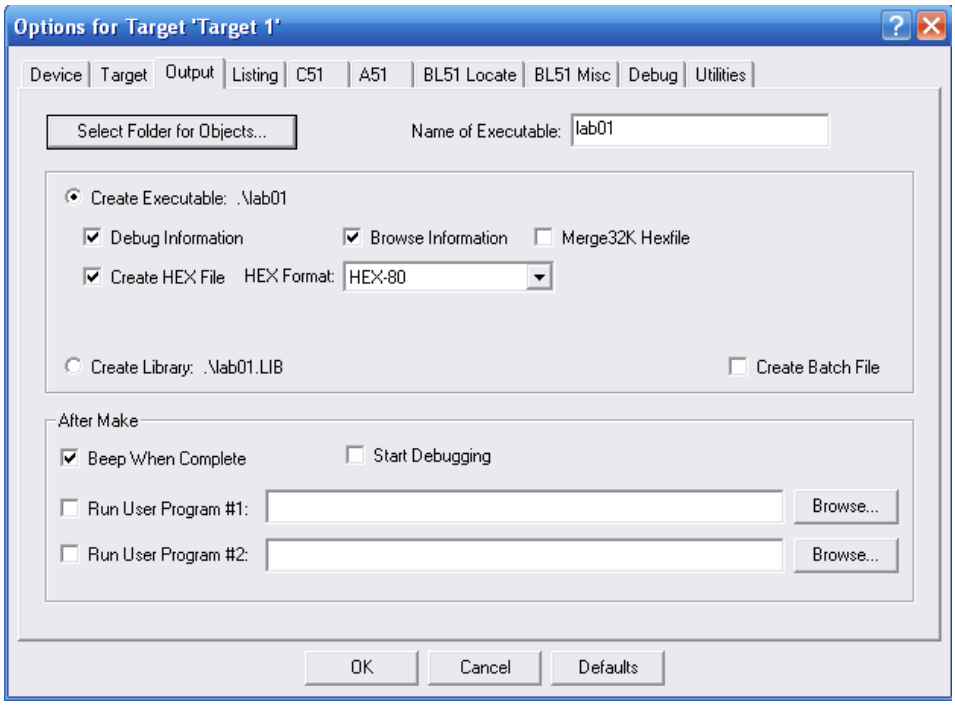
ขั้นตอนที่ 10 การสร้างไฟล์ HEX เพื่อนำไปโปรแกรมลงในตัวไอซี ให้เลือกเมนูไปที่ Project →Option for Target ‘Target1’ ที่แถบ Target ทำการกำหนด X-tal (MHz) เท่ากับ 11.059 แสดงดังภาพที่ 10.32 เลือกที่แถบ Output คลิกเลือกที่หน้าช่อง Create HEX File แสดงดังภาพที่ 10.33 แล้วกดที่ปุ่ม Ok



ภาพที่ 10.31 คอมไพล์โปรแกรมโดยเมนู Built Target

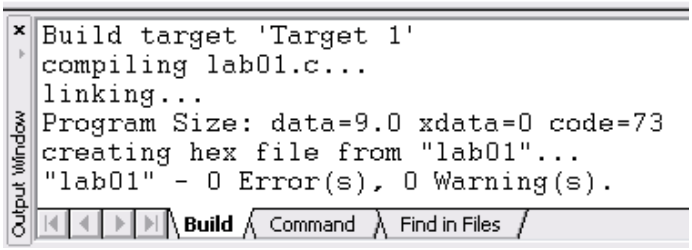


ภาพที่ 10.32 กำหนดค่าความถี่ของ X-tal ที่ใช้บนบอร์ดมีหน่วยเป็น MHz



ภาพที่ 10.33 กำหนดการสร้างไฟล์ HEX

ขั้นตอนที่ 11 ทำการคอมไพล์โปรแกรมโดยไปที่เมนู Project → Built Target อีกครั้ง ข้อความ จะสร้างไฟล์ HEX ที่หน้าต่างของวินโดว์ แสดงดังภาพที่ 10.34



ภาพที่ 10.34 สร้างไฟล์ HEX ที่ Built Target

นำไฟล์ HEX ที่ได้ไปทำการโปรแกรมลงในตัวไอซีด้วยเครื่องโปรแกรม

5. การประยุกต์ใช้งานบอร์ดไฟวิ่งควบคุมด้วยสัญญาณ DTMF

5.1 คุณสมบัติการใช้งาน

การประยุกต์การใช้งาน โดยการนำไมโครคอนโทรลเลอร์ MCS-51 ร่วมกับสัญญาณ DTMF ของโทรศัพท์ โดยใช้ควบคุมสั่งงานดังนี้

5.1.1 มีรูปแบบไฟวิ่งที่เก็บไว้ในหน่วยความจำโปรแกรมมากกว่า 80 รูปแบบ

5.1.2 ทำโปรแกรมไฟวิ่งได้หลายรูปแบบจากโทรศัพท์โดยไม่ต้องโทรออก

5.1.3 มีสวิตช์กดควบคุมระดับความเร็วของรูปแบบไฟวิ่งได้

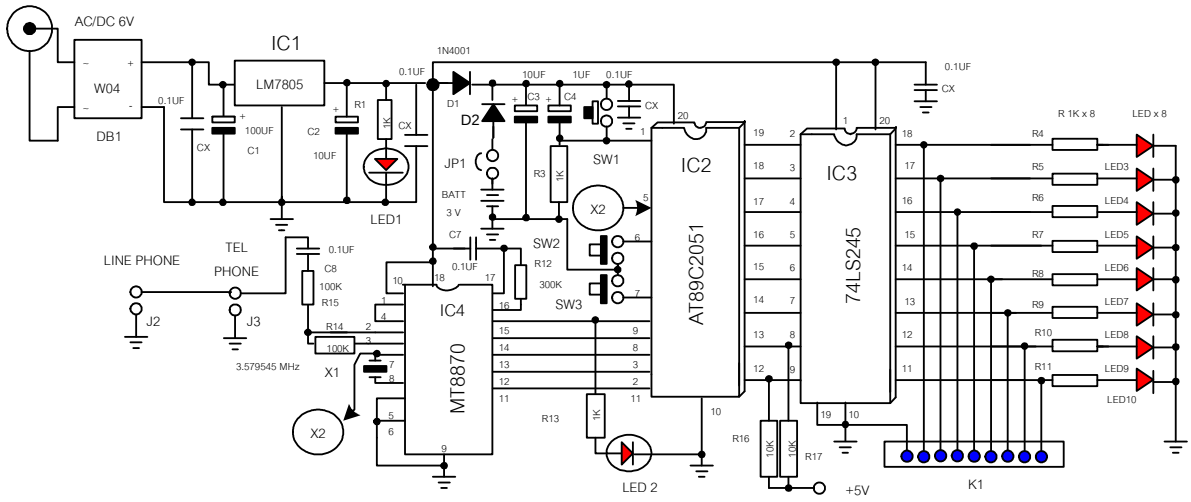
5.1.4 ใช้โซลิสเตทรีเลย์ 8 ช่องในการควบคุมการเปิดปิดหลอดไฟฟ้า

5.1.5 สั่งงานเครื่องใช้ไฟฟ้าได้ 8 ช่องอย่างอิสระ

5.2 การทำงานของวงจร

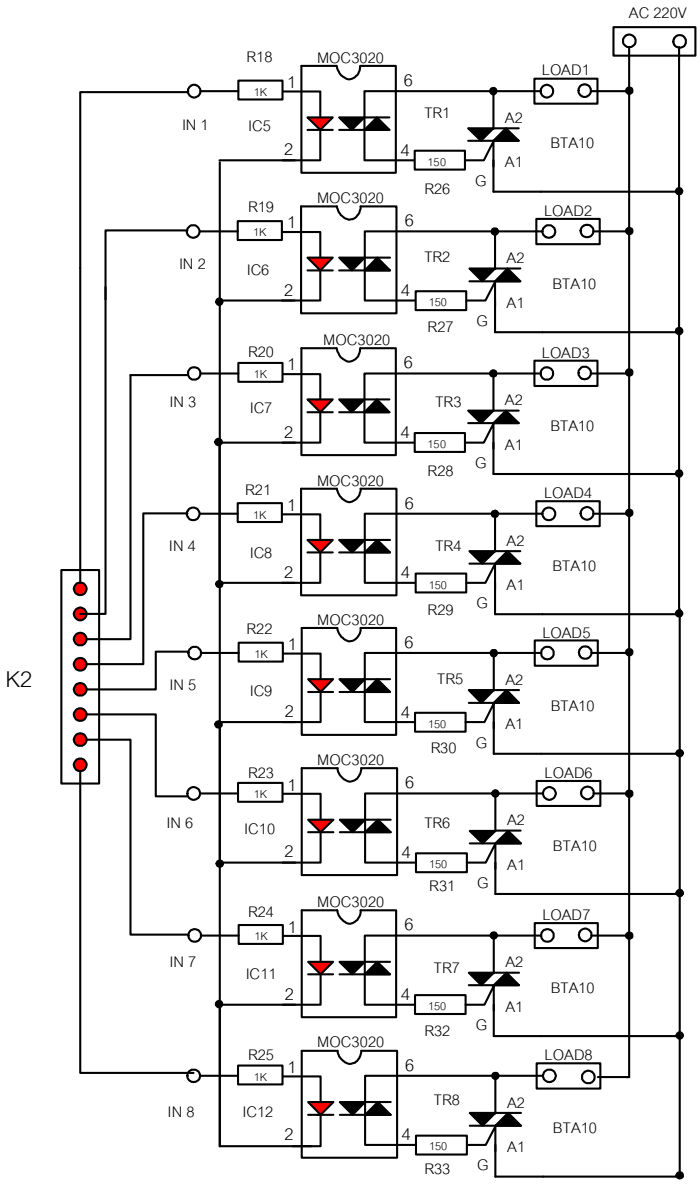
การทำงานแบ่งออกได้เป็น 2 ส่วน คือส่วนของไมโครคอนโทรลเลอร์ ทำหน้าที่ติดต่อกับสัญญาณ DTMF และควบคุมสั่งงานโดยโปรแกรม และส่วนควบคุมอุปกรณ์ไฟฟ้าภายนอกโดยใช้วงจรโซลิสเตทรีเลย์

จากภาพที่ 10.35 แสดงวงจรไมโครคอนโทรลเลอร์ การทำงานของวงจรเริ่มจาก IC4 เป็นไอซีถอดรหัสสัญญาณ DTMF จากโทรศัพท์ให้เป็นเลขไบนารี่ โดยมีขาสัญญาณที่เป็น Strobe แสดงผลที่ LED2 ทุกครั้ง ที่กดคีย์สวิตช์ โดยมี X-tal X1 กำหนดคาบเวลา ส่วน R13 และ C7 กำหนดค่าการ์ดไทม์ และ C8 R17, R16 ทำหน้าที่เป็นวงจรขยายความแตกต่างภายในตัวไอซี



ภาพที่ 10.35 วงจรถอดรหัสสัญญาณ DTMF และไอซี MCS-51

IC2 เป็นไมโครคอนโทรลเลอร์เบอร์ AT89C2051 ของ Atmel ทำหน้าที่ควบคุมจากโปรแกรมทั้งหมด จะใช้สัญญาณนาฬิกาจาก X-tal ร่วมกับ IC4 และ IC2 จะทำงานโดยเริ่มจากการแสดงผลข้อมูลของหน่วยความจำโปรแกรมภายในไอซี และทำการตรวจสอบข้อมูล ที่ได้จากการถอดรหัสสัญญาณจาก IC4 เพื่อใช้กำหนดการทำงานของโปรแกรม SW2 และ SW3 ทำหน้าที่ปรับความเร็วของรูปแบบการแสดงผล โดยที่ SW2 ทำให้การแสดงผลเร็วขึ้น และที่ SW3 จะทำให้การแสดงผลช้าลง



ภาพที่ 10.36 แสดงวงจรวงจรโซลิสเตทรีเลย์ 8 ช่อง

แหล่งจ่ายไฟของวงจรได้จาก อแดปเตอร์ ACหรือ DC 6-12 โวลต์ โดยผ่านไดโอด DB1 เป็นแบบบริดจ์เรกติไฟเออร์ เพื่อจัดขั้วของอแดปเตอร์ที่ต่อเข้ากับ IC1 เบอร์ LM7805 ให้มีแรงดันเอาต์พุตคงที่ 5 โวลต์ ป้อนเป็นไฟเลี้ยงให้กับวงจรโดยมี LED1 เป็นตัวแสดงสถานะของการทำงาน คอนเดนเซอร์ C1 C2 ทำหน้าที่กรองแรงดัน ส่วนไฟเลี้ยง IC2 นั้นจะผ่านไดโอด D1 เพื่อใช้แบตเตอร์รี่สำรองข้อมูลในขณะไม่มีไฟเลี้ยง โดยให้จำข้อมูลเดิมได้ D2 ทำหน้าที่ไม่ให้มีแรงดันย้อนกลับมาที่แบตเตอร์รี่ และ JP1 เลือกใช้แบตเตอร์รี่สำรอง หรือไม่ถ้าไม่ใช้ให้เปิดวงจรไว้ C4, R3 เป็นวงจรรีเซต ส่วน IC3 ทำหน้าที่เป็นบับเฟอร์

เพื่อป้องกัน IC2 เกิดความเสียหาย จากการจ่ายกระแสเกิน เอาต์พุตของ IC3 จะแสดงรูปแบบข้อมูลที่ LED3 – LED10 และอีกทางหนึ่งนำไปเป็นอินพุตให้กับส่วนของวงจรโซลิสเตทรีเลย์

วงจรควบคุมกับอุปกรณ์ไฟฟ้าภายนอก เป็นวงจรโซลิสเตทรีเลย์ 8 ช่อง โดย IC5 – IC12 เป็นออปโตไอโซเลเตอร์แบบไดแอค เพื่อนำไปทริกให้ขาเกทของไตรแอค TR1-TR8 และนำไปควบคุมอุปกรณ์ไฟฟ้าที่ต้องการได้แสดงดังภาพที่ 10.36

5.3 การทำงานของโปรแกรม

ในส่วนของโปรแกรมแสดงเป็นผังงานแสดงดังภาพที่ 10.37 และได้แบ่งหน้าที่การทำงานเป็น 3 ส่วนด้วยกันคือ 1 ส่วนควบคุมความเร็ว 2 ส่วนของการแสดงผลข้อมูล 3 ส่วนการตรวจสอบคีย์การกดสัญญาณ DTMF

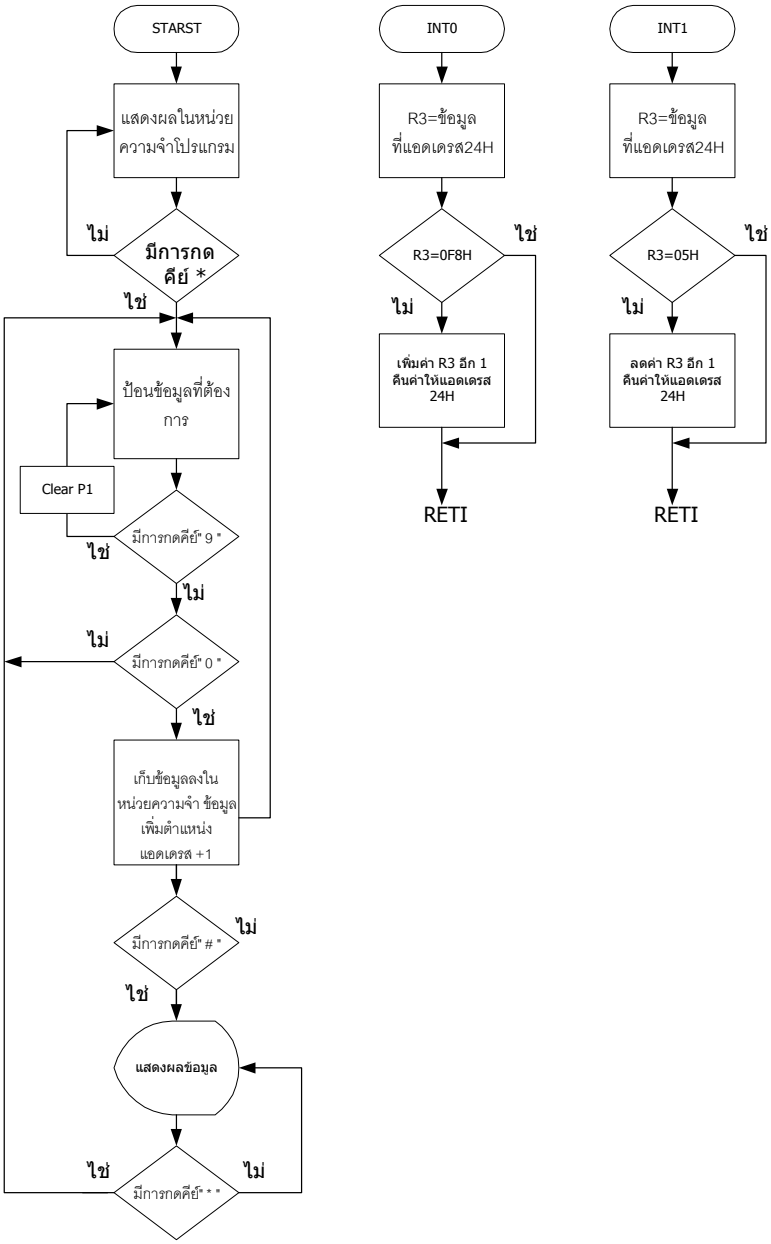
ส่วนที่ 1 จะควบคุมความเร็ว โดยใช้สัญญาณการอินเตอร์รัพท์จากภายนอก INT0 และ INT1 เป็นส่วนในการรับคีย์สวิตช์ โดยใช้รีจิสเตอร์ R3 เปรียบเทียบกับค่าที่เปลี่ยนแปลงได้สูงสุดและต่ำสุด กับหน่วยความจำแอดเดรสที่ 24H เก็บค่าที่เปลี่ยนแปลงจาก R3 เพื่อที่จะนำค่าไปใช้ในส่วนของการหน่วงเวลาที่เลเบล DEALY_SHOW

ส่วนที่ 2 เป็นการแสดงผลข้อมูลหลังจากมีไฟเลี้ยงวงจรขณะเริ่มต้น โดยจะนำค่าจากหน่วยความจำโปรแกรมเป็น Lookup Table นำข้อมูลมาแสดงผลก่อน โดยมีการตรวจสอบการกดคีย์จากโทรศัพท์โดยเช็คจากสัญญาณ Strobe ทุกครั้ง ในการแสดงในหนึ่งรูปแบบ หากพบการกดหมายเลขคีย์ “*” จะหยุดการแสดงผลที่แอลอีดีทั้งหมด ทำการเคลียร์ให้แอลอีดีดับทุกดวงแล้วไปทำโปรแกรม ในส่วนที่ 3

ส่วนที่ 3 เป็นการรับและตรวจสอบหมายเลขจากสัญญาณ DTMF ของโทรศัพท์ โดยเก็บไว้ที่หน่วยความจำตำแหน่ง 20H แล้วนำมาเปรียบเทียบกับข้อมูลที่กำหนดหมายเลข 0 - 8 ถ้าตรงค่าใด จะแสดงค่าเป็นรูปแบบที่ LED3-LED10 และถ้ากดคีย์ “0” เป็นการเก็บข้อมูลลงในหน่วยความจำซึ่งจองเนื้อที่ตั้งแต่หน่วยความจำที่ 25H เป็นต้นไปส่วนคีย์ “9” เป็นส่วนของการแก้ไขข้อมูลลงในหน่วยความจำของแต่ละแอดเดรส และคีย์ ‘#” เป็นการเริ่มให้แสดงผลรูปแบบที่เลเบล NEW_DISP และยังมีการตรวจสอบการกดคีย์เพื่อแก้ไข และทำรูปแบบใหม่

```
;*********************************************
;*****  DTMF CONTROL  DISPLAY          *****
;*********************************************
STROBE      BIT     P3.5    ; เป็นไดเร็กทีพที่ใช้กำหนดค่า Strobe แทนขา P3.5
IN_A        BIT     P3.7    ; เป็นไดเร็กทีพที่ใช้กำหนดค่า IN_A แทนขา P3.7
IN_B        BIT      P3.0   ; เป็นไดเร็กทีพที่ใช้กำหนดค่า IN_B แทนขา P3.5
IN_C        BIT     P3.1    ; เป็นไดเร็กทีพที่ใช้กำหนดค่า IN_C แทนขา P3.4
IN_D        BIT     P3.4    ; เป็นไดเร็กทีพที่ใช้กำหนดค่า IN_D แทนขา P3.1
```

```
INC_SW      BIT     P3.2    ; เป็นไดเร็กทีพที่ใช้กำหนดค่า INC_SW แทนขา P3.2
DEC_SW      BIT     P3.3    ; เป็นไดเร็กทีพที่ใช้กำหนดค่า DEC_SW แทนขา P3.3
```



ภาพที่ 10.37 แสดงผังงานของโปรแกรม

```
            ORG     25H             ;แอดเดรสเริ่มต้นของหน่วยความจำแรม
DATA:       DS      50H             ;จองเนื้อที่ในหน่วยความจำแรม  50H ค่า
            ORG     0000H           ;แอดเดรสเริ่มต้นของหน่วยความจำโปรแกรม
            SJMP    START           ;กระโดดไปทำที่เลเบล START
```

```
        ORG   0003H          ;อินเตอร์รัพท์ภายนอก INT0
        AJMP  INT_0          ;กระโดดไปทำที่เลเบล INT_0
        ORG   0013H          ;อินเตอร์รัพท์ภายนอก INT1
        AJMP  INT_1          ;กระโดดไปทำที่เลเบล INT_1
;**** เป็นการใช้ INTERRUPT ภายนอกเพื่อเพิ่มค่าข้อมูล และลดค่าข้อมูล
INT_0:  MOV   R3,24H         ;นำค่าข้อมูลจากหน่วยความจำแอดเดรส 20H>>R3
        CJNE  R3,#0F8H,END_INC ;เปรียบเทียบค่าสูงสุดที่จะหน่วงเวลา
        RETI                 ;ถ้าเป็นค่าสูงสุดแล้วให้กลับไปที่โปรแกรมหลัก
END_INC: INC  24H            ;เพิ่มค่าการหน่วงเวลาขึ้น 1 ค่า
        MOV   R3,24H         ;นำค่าข้อมูลที่เพิ่มค่าไปเก็บไว้ที่ R3
        RETI                 ;ออกจากโปรแกรมย่อยการอินเตอร์รัพท์
INT_1:  MOV   R3,24H         ;นำค่าจากหน่วยความจำแอดเดรส 20H>>R3
        CJNE  R3,#05H,END_DEC ;เปรียบเทียบค่าต่ำสุดที่หน่วงเวลาได้
        RETI                 ;ถ้าเป็นค่าต่ำสุดแล้วให้กลับไปที่โปรแกรมหลัก
END_DEC: DEC  24H            ;ลดค่าการหน่วงเวลาลง 1 ค่า
        MOV   R3,24H         ;นำข้อมูลที่แอดเดรส 24Hไปเก็บไว้ที่ R3
        RETI                 ;ออกจากโปรแกรมย่อยการอินเตอร์รัพท์
START:  SETB  INC_SW         ;กำหนดให้เป็น “1“ เพื่อเป็นอินพุต
        SETB  DEC_SW         ;กำหนดให้เป็น “1“ เพื่อเป็นอินพุต
        SETB  EA             ;กำหนดให้มีการอินเตอร์รัพท์ทั้งหมด
        SETB  EX0            ;รับการอินเตอร์รัพท์ภายนอก INT0
        SETB  EX1            ;รับการอินเตอร์รัพท์ภายนอก INT1
        SETB  IT0            ;เลือกทำที่ขอบขาลงของสัญญาณ
        SETB  IT1            ;เลือกทำที่ขอบขาลงของสัญญาณ
        SETB  STROBE         ;กำหนดให้เป็น “1 “ เพื่อเป็นอินพุต
        MOV   20H,#00H
        MOV   21H,#00H
        MOV   24H,#08H
        MOV   R0,#DATA
        MOV   R7,#50H        ;เคลียร์ค่าเริ่มต้นในหน่วยความจำจำนวน 50H ค่า
        MOV   A,#00H         ;เคลียร์ค่าข้อมูลในรีจิสเตอร์ A
```

```
;*** เคลียร์ข้อมูลในหน่วยความจำตั้งแต่แอดเดรสที่ 00 – 50H  ****
CLEAR:      MOV   @R0,#00H      ;กำหนดให้ R0 เก็บค่าฐานข้อมูล
            INC   R0            ;เพิ่มค่า R0
            DJNZ  R7,CLEAR      ;ครบจำนวนหรือไม่ถ้าไม่ครบวนไปเคลียร์ใหม่
;***  แสดงผลค่าข้อมูลที่กำหนดไว้ที่หน่วยความจำโปรแกรม     ****
DISPLAY:    MOV   P1,#00H               ;เคลียร์ค่าข้อมูลในพอร์ต P1
LEFT:       MOV   R1,#63D               ;กำหนดจำนวนค่าข้อมูลที่จะให้แสดงผล
            MOV   R6,#00H               ;เคลียร์ค่าข้อมูลในรีจิสเตอร์ R6
            MOV   DPTR,#TABLE           ;กำหนดค่า DPTR เพื่อให้เป็นฐานแอดเดรส
LOOP_LEFT:  MOV   A,R6                  ;นำค่าข้อมูลจากรีจิสเตอร์ R6 ไปเก็บไว้ที่รีจิสเตอร์ A
            MOVC  A,@A+DPTR             ;เปิดตารางแบบ Lookup Table
            MOV   P1,A                  ;นำค่าข้อมูลที่ได้ ออกไปแสดงผลที่พอร์ต P1
            JB    STROBE,CHEAK_INT
            ACALL DELAY_SHOW            ; เรียกโปรแกรมหน่วงเวลา
            INC   R6                    ;เพิ่มค่าข้อมูลในรีจิสเตอร์ R6 ขึ้น1ค่า
            DJNZ  R1,LOOP_LEFT          ;ถ้าข้อมูลในรีจิสเตอร์ R1ไม่เท่ากับ 0
RIGHT:      MOV   R1,#63D               ;กำหนดจำนวนค่าข้อมูลที่จะแสดงผล
            MOV   R6,#62D               ;กำหนดตำแหน่งข้อมูลที่จะให้แสดงผล
            MOV   DPTR,#TABLE           ;กำหนดค่า DPTR เพื่อเป็นฐานแอดเดรส
LOOP_RIGHT: MOV   A,R6                  ;นำค่าข้อมูลจากรีจิสเตอร์ R6 ไปเก็บไว้ที่รีจิสเตอร์ A
            MOVC  A,@A+DPTR             ;เปิดตารางแบบ Lookup Table
            MOV   P1,A                  ;นำค่าข้อมูลที่ได้ ออกไปแสดงผลที่พอร์ต P1
            JB    STROBE,CHEAK_INT      ;ไปที่ตรวจสอบที่คีย์หากมีการกดสวิตช์
            ACALL DELAY_SHOW            ;เรียกโปรแกรมหน่วงเวลา
            DEC   R6                    ;ลดค่าข้อมูลในรีจิสเตอร์ R6ลง 1 ค่า
            DJNZ  R1,LOOP_RIGHT         ;ถ้า R1ไม่เท่ากับ 0 ให้ไปวนแสดงผลใหม่
            SJMP  DISPLAY               ;วนกลับไปทำที่เลเบล DISPLAY ใหม่
;**** แสดงผลค่าข้อมูลที่ป้อนผ่านทางคีย์โทรศัพท์  ****
NEW_DISP:   MOV   R0,#DATA              ;นำค่าคงที่ตำแหน่งแอดเดรสที่จองไว้เก็บไว้ที่ R0
LOOP:       MOV   A,@R0                 ;นำข้อมูลที่แอดเดรสที่ชี้ด้วย R0 ออกที่รีจิสเตอร์ A
            CJNE  A,#00H,DISP           ;ตรวจสอบข้อมูลลำดับสุดท้ายเป็น 00 หรือไม่
```

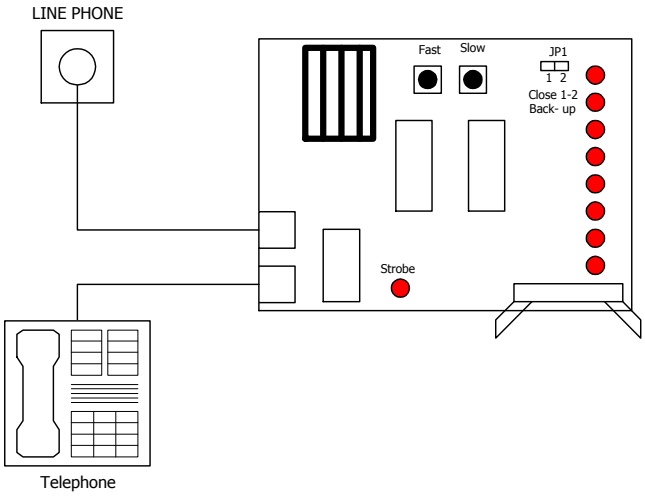
```
                SJMP    NEW_DISP            ;ถ้าข้อมูลเป็น 00 ให้วนไปเริ่มทำใหม่
DISP:           MOV     P1,A                ;ไม่เท่ากับ 00 ให้นำข้อมูลในรีจีสเตอร์ A ออกพอร์ต P1
                ACALL   DELAY_SHOW          ;เรียกโปรแกรมย่อยหน่วงเวลา
                JB      STROBE,CHEAK_EDIT;ตรวจสอบสัญญาณการกดคีย์เพื่อแก้ไขข้อมูล
                INC     R0                  ;เพิ่มค่าแอดเดรสลำดับต่อไป
                SJMP    LOOP                ;วนไปเริ่มต้นทำใหม่
CHEAK_EDIT:     ACALL   IN_KEY              ;เรียกโปรแกรมย่อยรับข้อมูล
                CJNE    A,#00001001B,A1     ;เปรียบเทียบค่าที่รับเข้ามาเท่ากับ “9”หรือไม่
                AJMP    EDIT                ;เท่ากับคีย์ “9” กระโดดไปโปรแกรมย่อยแก้ไขข้อมูล
A1:             CJNE    A,#00001011B,NEW_DISP   ;เปรียบเทียบค่าที่รับเข้ามาเท่ากับ “*”หรือไม่
                SJMP    DTMF                ;เท่ากับคีย์ “ * ’ไปเลเบล DTMF
        ;***    โปรแกรมย่อยรับค่าคีย์จากสัญญาณ DTMF    ***
CHEAK_INT:      ACALL   IN_KEY              ;เรียกโปรแกรมย่อยรับข้อมูล
                CJNE    A,#00001011B,DISPLAY    ;เปรียบเทียบค่าที่รับเข้ามาเท่ากับ “*”หรือไม่
DTMF:           MOV     P1,#00H             ;เคลียร์ค่าข้อมูลในพอร์ต P1
                MOV     P3,#0FFH            ;ให้พอร์ต P3 เป็นอินพุต
                MOV     20H,#00H            ;เคลียร์ค่าข้อมูลในแอดเดรส 20H
                MOV     21H,#00H            ;เคลียร์ค่าข้อมูลในแอดเดรส 21H
                MOV     R0,#DATA            ;เก็บค่าเริ่มต้นตำแหน่งที่จองไว้เก็บข้อมูลรีจิสเตอร์ R0
MAIN:           CALL    IN_KEY              ;เรียกโปรแกรมย่อยรับค่าข้อมูล
                CALL    CHEAK               ;เรียกโปรแกรมย่อยตรวจค่าคีย์ข้อมูล
                JNB     STROBE,$            ;ตรวจสอบการปล่อยคีย์
                SJMP    MAIN                ;กระโดดกลับไปทำที่เลเบล MAIN
IN_KEY:         MOV     C,IN_A              ;เก็บค่าสถานะที่ IN_A ไว้ที่ แฟลกตัวทด
                MOV     00H,C               ;นำค่าสถานะที่แฟลกตัวทดไปไว้ที่บิต 00H (20H.0)
                MOV     C,IN_B              ;เก็บค่าสถานะที่ IN_B ไว้ที่ แฟลกตัวทด
                MOV     01H,C               ;นำค่าสถานะที่แฟลกตัวทดไปไว้ที่บิต 01H (20H.1)
                MOV     C,IN_C              ;เก็บค่าสถานะที่ IN_C ไว้ที่ แฟลกตัวทด
                MOV     02H,C               ;นำค่าสถานะที่แฟลกตัวทดไปไว้ที่บิต 02H (20H.2)
                MOV     C,IN_D              ;เก็บค่าสถานะที่ IN_D ไว้ที่ แฟลกตัวทด
                MOV     03H,C               ;นำค่าสถานะที่แฟลกตัวทดไปไว้ที่บิต 03H (20H.3)
```

```
                MOV   A,20H
                RET                          ;ออกจากโปรแกรมย่อย
;   ****** โปรแกรมย่อยตรวจสอบค่าคีย์ และแสดงผล
CHEAK:          CJNE  A,#00000001B,DIS_02    ;ค่าที่ได้เท่ากับ “1” หรือไม่ไปเลเบล DIS_02
                CPL   08H                    ;กลับค่าสถานะของบิตที่ 08H(21H.0)
                MOV   P1,21H                 ;นำค่าข้อมูลที่ตำแหน่งแอดเดรส 21H ออกที่พอร์ต P1
                CALL  CHEAK_SW               ;เรียกโปรแกรมย่อยการปล่อยสวิตช์
                AJMP  END_RET                ;ออกโปรแกรมย่อย
DIS_02:         CJNE  A,#00000010B,DIS_03    ;ค่าที่ได้เท่ากับ “2” หรือไม่  ถ้าไม่เท่าไป DIS_03
                CPL   09H                    ;กลับค่าสถานะของบิตที่ 09H(21H.1)
                MOV   P1,21H                 ;นำค่าข้อมูลที่ตำแหน่งแอดเดรส 21H ออกที่พอร์ต P1
                CALL  CHEAK_SW               ;เรียกโปรแกรมย่อยการปล่อยสวิตช์
                AJMP  END_RET                ;ออกโปรแกรมย่อย
DIS_03:         CJNE  A,#00000011B,DIS_04    ;ค่าที่ได้เท่ากับ “3” หรือไม่ถ้าไม่ให้ไปที่เลเบล DIS_04
                .
                .
DIS_0A:         CJNE  A,#00001100B,DIS_0B    ;
                AJMP  NEW_DISP               ;กระโดดไปเลเบล NEW_DISP
DIS_0B:         CJNE  A,#00001010B,END_RET;ค่าที่ได้เท่ากับ “0” หรือไม่ถ้าไม่ออกจากโปรแกรม
                MOV   @R0,P1                 ;เก็บค่าข้อมูลที่ออก P1 ทั้งหมดไว้ที่@R0
                MOV   P1,#00H                ;เคลียร์ข้อมูลที่พอร์ต P1
                CALL  DELAY                  ;เรียกโปรแกรมย่อยหน่วงเวลา
                MOV   P1,@R0                 ;นำค่าข้อมูลที่ได้ออกพอร์ต 1 อีกครั้ง
                CALL  DELAY                  ;เรียกโปรแกรมย่อยหน่วงเวลา
                INC   R0                     ;เพิ่มตำแหน่งแอดเดรสของหน่วยความจำขึ้นอีก 1 ค่า
                MOV   P1,#00H                ;เคลียร์ข้อมูลที่พอร์ต P1
                MOV   21H,#00H               ;เคลียร์ข้อมูลที่ตำแหน่งแอดเดรส 21H
END_RET:        RET                          ;ออกจากโปรแกรมย่อย
EDIT:           MOV   A,@R0                  ;นำค่าข้อมูลที่หน่วยความจำตำแหน่งที่อ้างโดย R0 >>A
                MOV   P1,A                   ;นำค่าข้อมูลในรีจิสเตอร์ A>>>พอร์ต P1
                JB    STROBE,$               ;ตรวจสอบการปล่อยคีย์
```

```
EDIT_DIS:    ACALL IN_KEY          ;เรียกโปรแกรมย่อยรับคีย์
             ACALL CHEAK           ;เรียกโปรแกรมย่อยตรวจสอบค่าคีย์
             JNB   STROBE,$        ;ตรวจสอบการกดคีย์
             SJMP  EDIT_DIS        ;กระโดดไปที่เลเบล EDIT_DIS
CHEAK_SW:    JNB   STROBE,$        ;ตรวจสอบการกดคีย์
             CALL  DELAY           ;เรียกโปรแกรมย่อยหน่วงเวลา
             JB    STROBE,$        ;ตรวจสอบการปล่อยคีย์
             RET                   ;ออกจากโปรแกรมย่อย
        ;*** โปรแกรมย่อยหน่วงเวลาแสดงผล  *****
DELAY_SHOW:  MOV   R2,#0FH         ;กำหนดค่าคงที่ให้ R2
DELAY1:      MOV   R3,24H                ;นำค่าที่แอดเดรส 24H>>> R3
DELAY2:      MOV   R4,#0FFH              ;กำหนดค่าคงที่ให้ R4
             DJNZ  R4,$
             DJNZ  R3,DELAY2             ;ลดค่าที่ R3ถ้าไม่เท่ากับ 0 ไปทำที่ DELAY2
             DJNZ  R2,DELAY1             ;ลดค่าที่ R2ถ้าไม่เท่ากับ 0 ไปทำที่ DELAY1
             RET                         ; ออกจากโปรแกรมย่อย
DELAY:       MOV   R4,#8FH               ;กำหนดค่าคงที่ให้กับรีจิสเตอร์ R4
DELAY_1:     MOV   R5,#0FFH              ;กำหนดค่าคงที่ให้กับรีจิสเตอร์ R5
             DJNZ  R5,$                  ;ลดค่าที่ R5 ลง
             DJNZ  R4,DELAY_1            ;ลดค่าที่ R4ถ้าไม่เท่ากับ 0 ไปทำที่ DELAY_1
             RET                         ; ออกจากโปรแกรมย่อย
        ;*********************************************************************
        ;*******    ข้อมูลที่จะแสดงผล (เก็บไว้ในหน่วยความจำโปรแกรม)    *********
        ;*********************************************************************
TABLE:       DB          81H,42H,24H,18H
             .
             .
             DB          18H,81H,24H,42H
             DB          66H,18H,0FFH
             END
```

5.4 การทดสอบและใช้งาน

เสียบสายโทรศัพท์ติดต่อเข้าบอร์ดวงจรแสดงดังภาพที่ 10.38 ทดสอบการใช้งานตามลำดับดังนี้



ภาพที่ 10.38 การต่อใช้งานโดยเสียบสายโทรศัพท์เข้ากับบอร์ด

5.4.1 เมื่อเริ่มต้นเปิดเครื่องครั้งแรก แอลอีดีแสดงผล ตามรูปแบบกำหนดมาจากหน่วยความจำโปรแกรมในไอซี (Lookup Table)

5.4.2 ปรับความเร็วโดยกดที่ปุ่ม Fast หรือ Slow โดยกดที่ปุ่มเดิมหลายๆครั้งจนพอใจ ที่ปุ่มสวิตช์ Fast จะเป็นการเพิ่มความเร็วของการเปลี่ยนแปลงรูปแบบไฟวิ่ง และเมื่อกดที่ปุ่มสวิตช์ Slow จะเป็นการลดความเร็ว

5.4.3 การเปลี่ยนแปลงรูปแบบให้ยกหูโทรศัพท์ หรือใช้ในส่วนของ Speaker Phone แล้ว กดที่ปุ่มสวิตช์ หมายเลข " * " ของโทรศัพท์ ทำให้ส่วนของแอลอีดีแสดงผล ดับทุกดวง และการกดปุ่มที่โทรศัพท์ทุกครั้งทำให้แอลอีดีที่เป็นสัญญาณ Strobe แสดงผลทุกครั้ง

5.4.4 การกดที่หมายเลขสวิตช์ที่ปุ่มของโทรศัพท์ "1" – "8" ในตอนนี้สามารถสั่งงานในแต่ละช่องได้อย่างอิสระโดยการกดหนึ่งครั้งเป็นการเปิด และกดอีกครั้งเป็นการปิด หากต้องการปิดทั้งหมดพร้อมกันก็ให้กดที่ปุ่มสวิตช์ หมายเลข "9"

5.4.5 การปรับเปลี่ยนรูปแบบของไฟวิ่งให้กำหนดรูปแบบได้โดยทำเหมือนขั้นตอนที่ 4 หลังจากนั้นให้กดปุ่มสวิตช์ หมายเลข "0" แอลอีดีที่แสดงผลของรูปแบบจะกระพริบ 1 ครั้ง จากนั้นให้ป้อนรูปแบบต่อไปแล้วกดปุ่มสวิตช์ หมายเลข "0" เหมือนครั้งแรกตามทุกครั้ง จนครบรูปแบบที่กำหนดไว้ทั้งหมด และในการป้อนข้อมูลป้อนไม่ควรเกิน 80 ค่า

5.4.6 ทำการกดปุ่มจบการป้อนข้อมูลโดยการกดปุ่มสวิตช์ หมายเลข “0”ซ้ำอีก 1 ครั้งแล้วตามด้วยปุ่มสวิตช์ หมายเลข “#” แอลอีดีจะแสดงผลรูปแบบไฟวิ่งตามที่กำหนดไว้ สามารถปรับความเร็วของการวิ่งได้ดังข้อ 2

5.4.7 การปรับเปลี่ยนแก้ไขข้อมูลหลังจากที่เราป้อนข้อมูล โดยการกดปุ่ม Slow ปรับความเร็วให้ช้าลงเพื่อที่จะมองการเปลี่ยนแปลงรูปแบบได้แบบช้าๆ จนถึงตำแหน่งที่จะทำการแก้ไขให้ทำการกดที่ปุ่มสวิตช์ หมายเลข “9” แอลอีดีจะดับทุกดวง

5.4.8 ให้ทำการเปลี่ยนข้อมูลใหม่ได้ตามต้องการทำเหมือนข้อที่ 4 และข้อที่ 5 หลังจากนั้นให้กดที่ปุ่มสวิตช์ หมายเลข “#” รูปแบบจะทำการแก้ไขปรับเปลี่ยนให้ใหม่ ข้อควรระวังการแก้ไขข้อมูลทุกครั้งจะกดที่ปุ่มสวิตช์ หมายเลข “0”หลังจากแก้ไขรูปแบบใน 1 ตำแหน่งเท่านั้น หลังจากป้อนเสร็จไม่ต้องกดปุ่มสวิตช์ หมายเลข “0” ซ้ำอีกเพียงแต่กดปุ่มสวิตช์ หมายเลข “#” เท่านั้น (ต่างจากข้อที่ 6 ซึ่งเป็นการป้อนข้อมูลทั้งหมดในครั้งแรก)

5.4.9 หลังจากที่โปรแกรมเสร็จแล้วให้ถอดสายโทรศัพท์ออกได้ รูปแบบข้อมูลจะยังคงอยู่ที่บอร์ด หากต้องการแก้ไขสามารถนำสายมาต่อ แล้วทำการแก้ไขรูปแบบข้อมูลใหม่ได้

5.4.10 ในการที่ต้องการที่จะเก็บข้อมูลไว้ใช้งานหลังจากถอดแหล่งจ่ายไฟออกสามารถทำได้โดยให้เลือกจัมเปอร์ในตำแหน่ง Back-up (Close ที่ขา 1-2 ของ JP1)

5.5 การตรวจสอบ และแก้ไข

5.5.1 ถอด IC1,IC2 และ IC3 ออกจากบอร์ด

5.5.2 เสียบแจ็คอแดปเตอร์ AC หรือ DC เข้าสู่บอร์ด สังเกตที่ LED1 จะต้องสว่าง แสดงว่ามีไฟ +VCC 5 โวลต์ในวงจร ถ้า LED1 ไม่สว่างให้ตรวจสอบแรงดันที่ป้อนจากอแดปเตอร์,สวิตซ์โยก ,ขั้วของไดโอดบริดจ์ ,ขั้วของตัวเก็บประจุ , ไอซี 7805 และ ขั้วของ LED1 ตามลำดับ

5.5.3 ให้ใช้สายไฟขนาดเล็ก (สายโทรศัพท์) เสียบที่ขา 20 (+VCC)ของซ็อกเกต IC3จากนั้นให้นำปลายสายอีกข้างหนึ่งไปเสียบกับขา18 ถึง ขา11 เพื่อจนครบทุกขาสังเกตที่ LED3 –LED10 จะต้องสว่างในขณะที่สัมผัสถูกขา ถ้าไม่สว่างให้ตรวจสอบลาย PCB อาจจะลัดวงจรถึงกันได้ แต่ถ้าไม่สว่างทุกดวง ให้ทำการตรวจสอบ IC3 และ ขั้วของ LED1

5.5.4 ใส่ IC4 ลงบนซ็อกเกตไอซี ต่อสายโทรศัพท์ตามภาพที่ 10.32 หลังจากนั้นให้ทดลองกดที่สวิตซ์ใดๆ บนแป้นคีย์โทรศัพท์ LED2 จะต้องสว่างแสดงสัญญาณ Strobe ทุกครั้ง ถ้าไม่สว่างให้ตรวจสอบขั้วของ LED1 และลาย วงจรที่อาจจะเกิดการลัดวงจรได้

5.5.5 ใส่ IC2 และ IC3 ลงบนซ็อกเกตไอซี หลังจากนั้นให้กดที่สวิตซ์รีเซต สังเกตที่แอลอีดีแสดงผลจะต้องสว่างตามรูปแบบที่กำหนดไว้ ให้กดที่สวิตซ์ปรับความเร็วเพื่อทดสอบ (ถ้าหากไม่ใส่ IC4 แอลอีดีจะไม่แสดงผล เพราะไอซีไมโครคอนโทรลเลอร์ใช้ X-tal ร่วมกับ IC4)

5.5.6 หลังจากนั้นให้ทดสอบตามหัวข้อ การทดสอบและการใช้งานอีกครั้ง

บอร์ดไฟวิ่งควบคุมด้วยสัญญาณ DTMF ใช้ทำไฟวิ่งหลายรูปแบบในงานแสดงต่างๆ โดยเปลี่ยนรูปแบบได้สะดวก ยังสามารถที่นำไปดัดแปลงเพื่อควบคุมโซลีนอย์วาล์ว หรือปั้มน้ำ เพื่อทำน้ำพุ หรือนำไปใช้เปิดปิดอุปกรณ์ไฟฟ้าตามจุดต่างๆได้อย่างอิสระ และ สามารถนำไปควบคุมหัวพ่นฝอยน้ำหรือละอองหมอกสำหรับในแปลงเกษตร โดยกำหนดให้การเปลี่ยนแปลงแต่ละรูปแบบ เป็นเวลาที่ต้องการหน่วง การเปลี่ยนแปลงเวลา สามารถทำได้โดยกำหนดรูปแบบเดิมๆ หลายๆตำแหน่ง แล้วกดปุ่มปรับความเร็วของการหน่วงเวลาให้เท่ากับที่ต้องการ หากถึงเวลาที่เรากำหนดไว้ ก็ทำให้ช่องที่เรากำหนดการทำงานสามารถค้นคว้าวิธีการสร้างเพิ่มเติมได้ที่ http://www.adisak51/project08.html

สรุป

LCD เป็นอุปกรณ์แสดงผลที่ใช้กระแสไฟต่ำ แสดงผลได้ทั้งตัวอักษร ตัวเลข และรูปภาพ โดยมีวงจรควบคุมการแสดงผลต่ออยู่ภายใน LCD แบ่งออกเป็น 2 ประเภทใหญ่ ๆ คือ แบบ Dot Matrix (Text) แสดงผลเป็นตัวอักษร และจำนวนบรรทัดแตกต่างกันไปตามรุ่นที่เลือกใช้ ส่วนแบบ Graphic สามารถแสดงผล แบบ Bit-Map สร้างภาพต่าง ๆ

I^2C BUS ย่อมาจาก Inter-Integrated Circuit Bus หมายถึง เป็นการสื่อสารอนุกรม แบบซิงโครนัส เพื่อใช้ติดต่อสื่อสารระหว่างไมโครคอนโทรลเลอร์ กับอุปกรณ์ภายนอก พัฒนาขึ้นโดยบริษัท Philips Semiconductors จุดมุ่งหมายหลักคือ ต้องการให้ไอซีหรือโมดูลสามารถติดต่อ สั่งงาน และควบคุมภายใต้สัญญาณ 2 เส้น คือ สายข้อมูล และ สายสัญญาณนาฬิกา ใช้กำหนดจังหวะการทำงาน การต่อร่วมกันของอุปกรณ์บนบัส ใช้การต่อสายข้อมูล และสายสัญญาณนาฬิกาของอุปกรณ์แต่ละตัวขนานกันไป ส่วนการกำหนดแอดเดรสหรือตำแหน่งสำหรับติดต่ออุปกรณ์แต่ละตัว จะใช้รหัสข้อมูลและการกำหนดสภาวะลอจิกที่ขอแอดเดรสของอุปกรณ์แต่ละตัว สายข้อมูลบนบัส I^2C เป็นสายข้อมูลอนุกรมหรือ SDA ส่วนสายสัญญาณนาฬิกาอนุกรม หรือ SCL สาย SDA และ SCL เป็นสายสัญญาณ 2 ทิศทางต่ออยู่กับแหล่งจ่ายไฟบวกโดยมีการต่อตัวต้านทานพูลอัปกับแรงดัน +5 โวลต์ เมื่อบัสเป็นอิสระโดยไม่มีการติดต่อใช้งาน ทั้ง SDA และ SCL จะมีสถานะลอจิกสูง

บัส I^2C เป็นการกำหนดรูปแบบของการติดต่อบนหรือโปรโตคอล ที่มีการสื่อสารแบบ I^2C มีการทำงาน 5 สภาวะ 1) บัสว่าง 2) เริ่มต้นการถ่ายเทข้อมูล 3) หยุดการถ่ายเทข้อมูล 4) ข้อมูลดำรงอยู่บนบัส 5) การตอบรับ

การแปลงสัญญาณอนาล็อก (A/D: Analog to Digital Converter) ที่มีการเปลี่ยนแปลงอย่างต่อเนื่อง เช่นระดับของแรงดันไฟฟ้า หรือปริมาณของกระแสไฟฟ้า ให้กลายเป็นสัญญาณดิจิตอลที่อยู่ในรูปแบบของเลขฐานสองคือ “0” กับ “1” ซึ่งเป็นสัญญาณที่ไม่ขึ้นอยู่กับเวลา

ภาษาแอสเซมบลีเป็นภาษาที่ใช้เขียนร่วมกับไอซีไมโครคอนโทรลเลอร์ทุกตระกูล ในปัจจุบันภาษาซี มีการนำมาใช้พัฒนาโปรแกรมสำหรับไมโครคอนโทรลเลอร์ MCS-51 ภาษาซีเป็นภาษาที่มีโครงสร้าง ง่ายต่อการทำความเข้าใจ ปรับปรุง และพัฒนา เป็นภาษามาตรฐานไม่ขึ้นกับฮาร์ดแวร์ (ไมโครคอนโทรลเลอร์) มีความยืดหยุ่นในการโยกย้ายนำไปใช้กับงานไมโครคอนโทรลเลอร์ตระกูลอื่นได้ง่าย ตัวคอมไพเลอร์ และลิงเกอร์ของภาษาซี มีการพัฒนาอยู่ตลอดเวลาจนมีความสามารถสูง มีความเข้ากันได้กับภาษาแอสเซมบลีในระดับซอร์สโค้ดทั้งแบบ Inline Assembly และการลิงก์ (Link) โปรแกรมเข้าด้วยกัน ภาษาซีจึงเหมาะกับผู้ที่ไม่ต้องการศึกษาทางด้านฮาร์ดแวร์ หรือสถาปัตยกรรมภายในของไอซีไมโครคอนโทรลเลอร์มากนัก