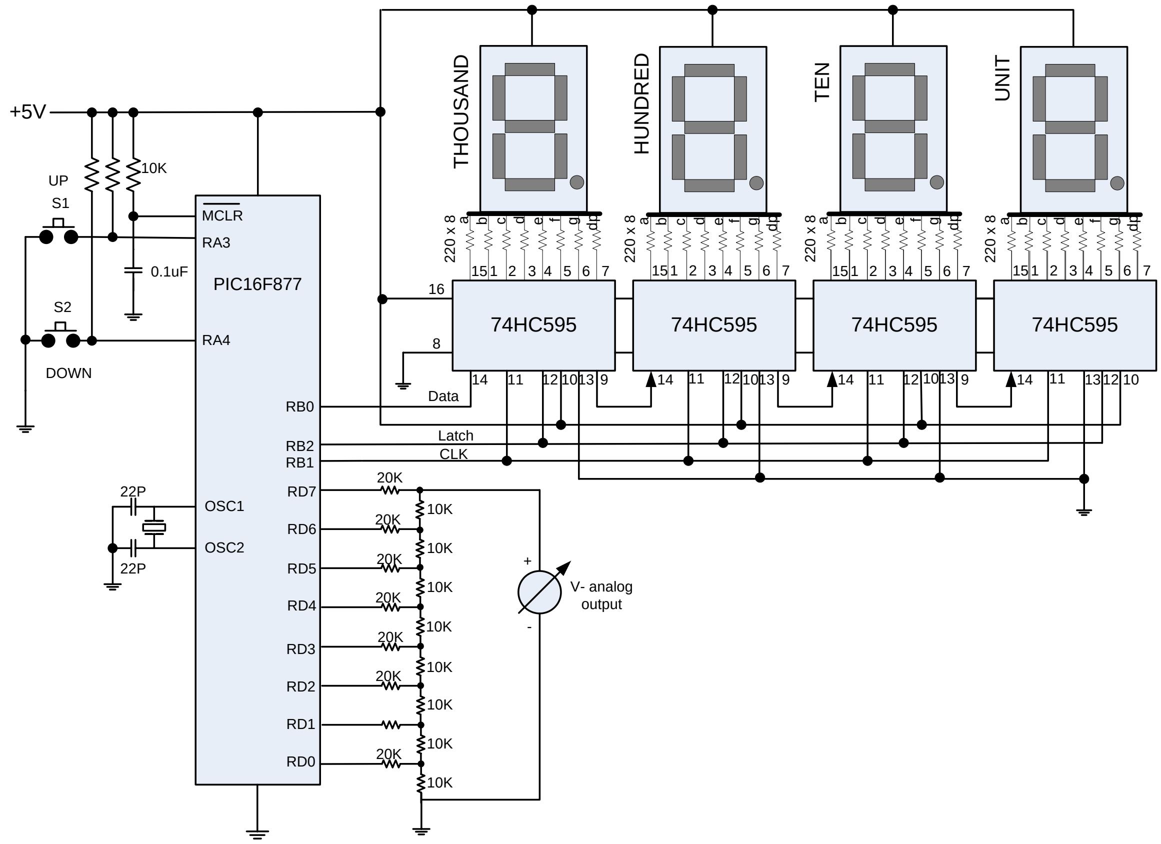
## *Experiment 22* การเขียนโปรแกรมแปลงสัญญาณดิจิตอล เป็นอนาล็อก (D/A)

จุดประสงค์ เพื่อศึกษาการเขียนโปรแกรมเพื่อแปลงสัญญาณดิจิตอลเป็นอนาล็อก (D/A)ขนาด 8 บิท โดยใช้วงจร R-2R Ladder Network

### ทฤษฎีพื้นฐาน

การแปลงสัญญาณดิจิตอล เป็นอนาล็อก มีความจำเป็นที่จะต้องสร้างวงจรอยู่ภายนอกตัวไมโคร - คอนโทรลเลอร์ เนื่องจากผู้ผลิตชิพ MCU ส่วนมากไม่ได้สร้างบรรจุไว้เนื่องจากวงจร D/A ใช้อุปกรณ์ประเภท Passive ได้แก่ตัวต้านทาน เป็นส่วนมาก ซึ่งจะทำให้ต้องใช้แผ่นชิพที่มีขนาดใหญ่ขึ้น และมีความร้อนสะสมในตัวชิพมาก การประยุกต์แปลงสัญญาณ D/A ส่วนมากเพื่อสร้างสัญญาณควบคุมแบบลิเนียร์ เช่น มิเตอร์แบบมีเข็ม สร้างสัญญาณรูปคลื่นแบบต่าง ๆ เป็นต้น

วงจรแปลงสัญญาณ D/A มีหลายแบบ แต่ที่นิยมใช้กันมากมี 2 แบบ ได้แก่ ใช้ไอซี D/A และวงจร R-2R Ladder network สำหรับการทดลองตามใบงานนี้จะใช้วงจร R-2R Ladder network เนื่องจากมีราคาถูก เป็นวงจรที่ง่ายไม่ซับซ้อน และอุปกรณ์เป็นตัวต้านทานเพียง 2 ค่า ที่สามารถหาซื้อได้ง่าย




รูปที่ 1 แสดงวงจรการแปลงสัญญาณดิจิตอลเป็นอนาล็อกโดยใช้วงจร R-2R Ladder สำหรับทดลองตามใบงานที่ 22

การทำงานของโปรแกรม ตามวงจรรูปที่ 1

เมื่อเริ่มเปิดไฟเข้าวงจร โปรแกรมจะเริ่มนับจาก 0 ไปถึง 255 ทุกค่าที่นับจะส่งออกไปที่พอร์ท D และจะค้างอยู่ที่ 255 ถ้าต้องการให้นับลง กด S2 ต้องการให้นับขึ้น กด S1 ค่าดิจิตอลจะแสดงด้วยตัวเลข LED 7 - Segment ขนาด 3 หลัก

โปรแกรมคำสั่ง

```
INCLUDE"MODEDEFS.BAS"
DEFINE SHIFT_PAUSEUS 100
s1        var   porta.3
s2        var   porta.4
dat       var   portb.0
clk       var   portb.1
latch     var   portb.2
stat      var   bit
unit      var   byte
ten       var   byte
hun       var   byte
thou      var   byte
digit     var   byte
num       var   byte
b0        var   byte
b1        var   byte
    TRISA     =   %111111
    TRISD     =   %0000000
    ADCON1  = 7
    stat        = 1
    thou        = 255
    num         = 0
'
begin:
      IF  (S1 =  0)  and  (S2 = 1) THEN stat = 1
      if   (S1 =  1)  and  (S2 = 0) THEN stat = 0
      pause 50
      if stat = 1  then
         num  =  num + 1
         gosub  display
         portd = num
         if num = 255 then num = 254
    else
         num  =  num - 1
         gosub  display
         portd = num
         if num = 0 then num = 1
    endif
    pause 80
    GOTO  begin
    END
'------------------ End  of  program ---------------
display:
    digit  =  num dig 0
    lookup digit,[$c0,$f9,$a4,$b0,$99,_
               $92,$82,$f8,$80,$90],unit
    digit  =  num dig 1
    lookup digit,[$c0,$f9,$a4,$b0,$99,_
               $92,$82,$f8,$80,$90],ten
    digit  =  num dig 2
    lookup digit,[$c0,$f9,$a4,$b0,$99,_
               $92,$82,$f8,$80,$90],hun
    high latch
    pauseus 10
    ShiftOut dat,clk,1,[unit,ten,hun,thou]
```

ต่อ

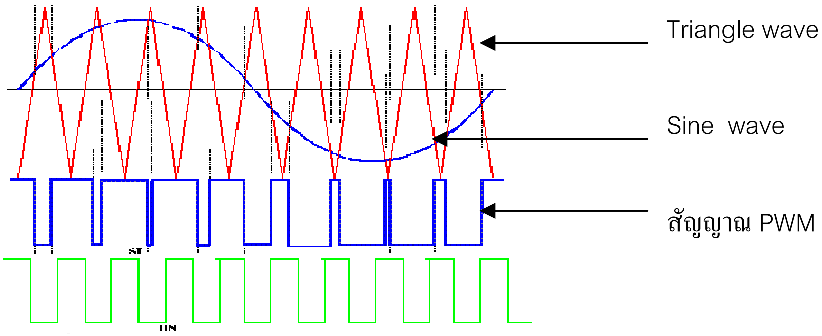
```
Pauseus 10
Low latch
Pauseus 10
high latch
Return
'----------------------------
```

## *Experiment 23* การเขียนโปรแกรมกำเนิดสัญญาณ PWM

จุดประสงค์ เพื่อศึกษาเทคนิคการเขียนโปรแกรมเพื่อการสร้างสัญญาณ PWM

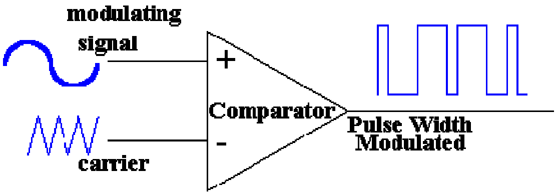
ทฤษฎีพื้นฐาน

สัญญาณ PWM มาจากชื่อเต็มว่า Pulse Width Modulation เป็นสัญญาณที่เกิดจากการผสมกันระหว่างสัญญาณรูปสามเหลี่ยม (Triangle wave) กับระดับแรงเคลื่อนสัญญาณไฟดีซีหรือสัญญาณอื่น ๆ ที่มีระดับไฟดีซีเป็นตัวรองรับ ผลที่ได้จากการผสมสัญญาณดังกล่าวจะได้รูปคลื่นสัญญาณพัลส์ที่มี 2 สถานะ คือ ON กับ OFF เมื่อเรานำสัญญาณพัลส์ที่ได้ไปขับอุปกรณ์กำลัง เช่น หลอดไฟ มอเตอร์ดีซี เราจะได้ผลการควบคุม คือเมื่อสัญญาณพัลส์มีสถานะ ON หลอดไฟจะติดสว่างเต็มที่ ถ้า OFF ไฟจะดับ หากสัญญาณมีความต่อเนื่องเช่นนี้ตลอดก็จะได้การติด – ดับ ของหลอดไฟต่อเนื่อง และหากการติดดับมีความถี่สูง ๆ จนสายตาไม่สามารถสังเกตการณ์ติด – ดับได้ เราก็จะสังเกตเห็นได้เพียงแสงที่หรี่ลง หากเราทำให้ช่วงเวลา ON น้อยลง และในทางตรงกันข้าม หากเราทำให้ช่วงเวลา ON มากขึ้นเรื่อย ๆ ก็จะทำให้หลอดไฟสว่างขึ้นเรื่อย ๆ จนถึงสว่างเต็มที่เมื่อ ON ตลอด



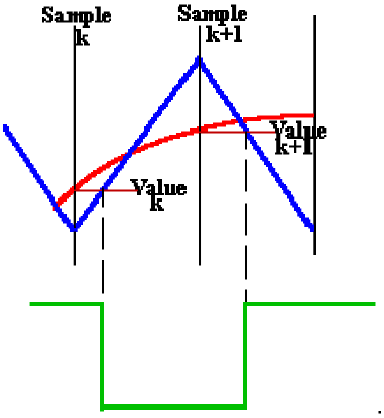

รูปที่ 1 แสดงการเกิดสัญญาณ PWM จากการผสมระหว่างสัญญาณรูปสามเหลี่ยม และรูปไซน์

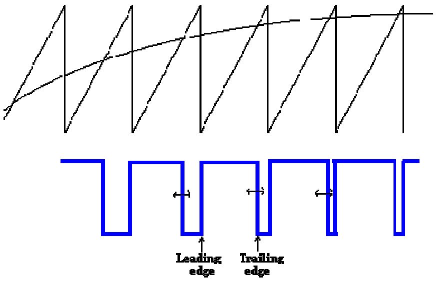
ในการออกแบบวงจรผสมสัญญาณ จะใช้วงจรไอซี Linear comparator หรือ Analog comparator ตามวงจรรูปที่ 2



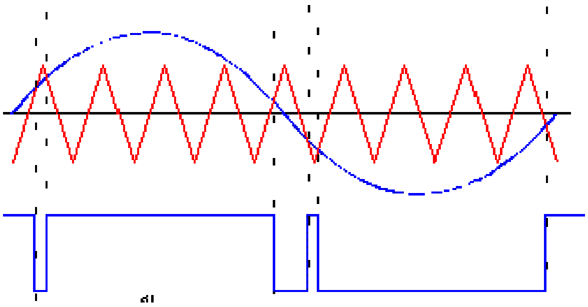

รูปที่ 2 แสดงวงจรการสร้างสัญญาณ PWM

สัญญาณรูปสามเหลี่ยมที่ใช้ในวงจร จะต้องมีความเป็นเชิงเส้น (Linearity) สูง จึงจะได้สัญญาณ PWM ที่มีความผิดเพี้ยนต่ำ สัญญาณรูปสามเหลี่ยมที่นำมาใช้จะมี 2 แบบ คือ Triangle wave และ Saw tooth wave




รูปที่ 3 แสดงการใช้คลื่นTriangle wave เป็นพาหะ รูปที่ 4 แสดงการใช้คลื่นSaw tooth wave เป็นพาหะ ในการผสมคลื่น สัญญาณคลื่นที่มาผสมจะต้องมีระดับความสูง (Amplitude) อย่างมากที่สุดไม่เกินยอดความสูงของรูปคลื่นสามเหลี่ยม ซึ่งเป็นรูปคลื่นพาหะ หากเกินไปกว่านั้นจะเกิดความผิดเพี้ยนขึ้นกับสัญญาณ PWM เราเรียกปรากฏการณ์นี้ว่า over modulation



รูปที่ 5 แสดงความผิดเพี้ยนของสัญญาณ PWM เนื่องจากการเกิด over modulation

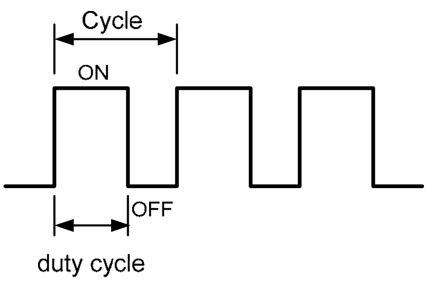
รายละเอียดของสัญญาณ PWM ประกอบด้วยส่วนต่าง ๆ ดังนี้ คือ

1. ความถี่ คือจำนวนรูปคลื่น ต่อวินาที ( Cycle / second) ในการใช้งาน ความถี่จะต้องคงที่ไม่เปลี่ยนแปลง ความถี่ที่เหมาะสมในการใช้งานควบคุมอุปกรณ์กำลังต่าง ๆ คือ ตั้งแต่ 400 Hz - 10 KHz ส่วนความถี่ที่ใช้เป็นพาหะของสัญญาณเสียง (Audio) ควรจะสูงกว่า 50 KHz - 1 MHz

2. ดิวตี้ไซเกิล (Duty cycle) เป็นค่าคาบเวลาช่วง ON ของไซเกิล โดยคิดเป็นเปอร์เซ็นต์ของ Full-cycle

3. ค่าแรงเคลื่อนเอ้าพุท เป็นค่าเฉลี่ยระหว่างแรงเคลื่อนช่วงเวลา ON กับ OFF คิดใน 1 ไซเกิล โดยใช้สมการ

$$\text{Vpwm} = \frac{\text{\%duty cycle} \times \text{Vpower}}{100}$$




รูปที่ 6 แสดงองค์ประกอบของสัญญาณ PWM

ในโครงสร้างของไมโครคอนโทรลเลอร์ตระกูลไมโครชิพ เบอร์ PIC16F877 ที่ใช้ปฏิบัติในใบงานทดลองนี้ มีโมดูลสร้างสัญญาณ PWM แบบต่อเนื่องโดยฮาร์ดแวร์จำนวน 2 ชุด โดยใช้ความละเอียดในการปรับค่าดิวตี้ไซเกิล ตั้งแต่ 0 - 100 % จำนวน 256 ค่า โดยมีสัญญาณ PWM ช่องที่ 1 ออกที่ขา RC2 และช่องที่ 2 ออกที่ขา RC1 ส่วนเบอร์อื่น ๆ ให้ดูใน datasheet ให้ดูในขาที่มีฟังก์ชั่น CCP1 CCP2 .. คำสั่งภาษา Pic Basic Pro Compiler มีคำสั่งที่ทำให้ส่งสัญญาณ PWM ออกมา ได้แก่

**HPWM *Channel, Duty cycle, Frequency***

Channel เป็นตำแหน่งขา I/O ที่ส่งสัญญาณ PWM ออกค่า 1 หมายถึง ขา RC2 และ 2 หมายถึง ขา RC1 บางตัวอาจมีมากกว่านี้

Duty cycle เป็นค่าเปอร์เซ็นต์คาบเวลา ON ของไซเกิล มีค่าระหว่าง 0 - 255 สามารถกำหนดแทนด้วยตัวแปรขนาด 1 byte ได้

Frequency เป็นค่าความถี่พาหะ มีหน่วยเป็น Hz ความถี่สูงสุดที่ MCU สร้างได้คือ 32767 Hz ส่วนความถี่ต่ำสุด ขึ้นอยู่กับเบอร์ของ MCU และ ค่า OSC. ที่ใช้ ซึ่งเป็นตามตาราง ตัวอย่างต่อไปนี้

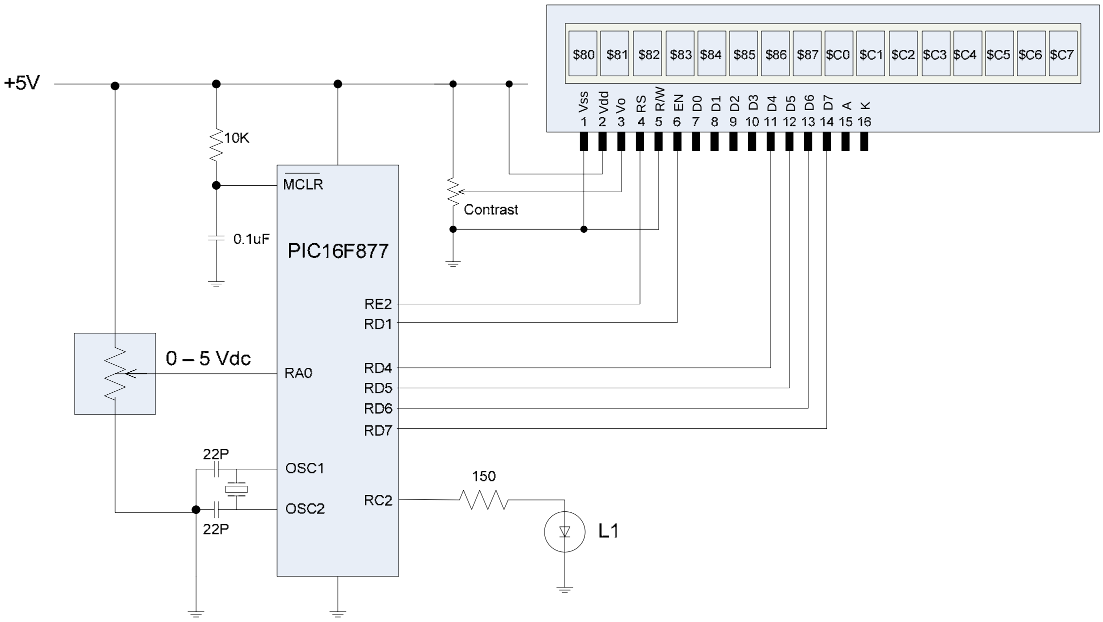
| OSC. | 14-bit core and 18-bit core | 17Cxxx |
|---|---|---|
| 4 MHz | 245 Hz | 3907 Hz |
| 10 MHz | 611 Hz | 9766 Hz |
| 12 MHz | 733 Hz | 11719 Hz |
| 20 MHz | 1221 Hz | 19531 Hz |
| 24 MHz | 1465 Hz | 23438 Hz |
| 33 MHz | 2015 Hz | 32227 Hz |
| 40 MHz | 2442 Hz | ไม่ได้กำหนด |

การทำงานของคำสั่ง จะทำตลอดเวลาโดยการสั่งเพียงครั้งเดียว และจะทำไปตามลำพัง ขณะที่โปรแกรมกำลังทำคำสั่งอื่นอยู่ เราจึงเรียกว่า Hardware Pulse Width Modulation หรือ HPWM ซึ่งจะมีข้อดี

คือ จะได้สัญญาณ PWM อย่างต่อเนื่อง โดยไม่ต้องเขียนโปแกรมให้วนย้อนกลับสร้างสัญญาณ PWM อยู่ตลอดเวลาสำหรับ MCU บางรุ่น ได้แก่ PIC18F452 สามารถสับเปลี่ยนขา I/O ที่ส่งสัญญาณ PWM ได้ โดยการใช้คำสั่งการนิยามขา I/O ดังตัวอย่างต่อไปนี้

```
DEFINE CCP1_REG PORTC
DEFINE CCP1_BIT 2
DEFINE CCP2_REG PORTC
DEFINE CCP2_BIT 1
```

วงจรทดลองตาม Experiment 23




การทำงานของโปแกรม

ตามวงจรทดลอง เป็นการศึกษาการเขียนโปรแกรมผลิตสัญญาณ PWM เพื่อนำไปขับหลอด L1 ให้สว่างมาก น้อยเป็นไปตามค่าดิวตี้ไซเกิล ค่ามาก น้อยของดิวตี้ไซเกิลถูกควบคุมโดยค่าตัวแปรที่รับจากการแปลง A /D ที่ Channel 0 ดังนั้นการปรับแรงเคลื่อน A/D จาก 0 – 5 V ก็จะทำให้หลอด L1 มืด และสว่างตามไปด้วย ค่าดิวตี้ไซเกิล จะถูกนำไปแสดงที่จอ LCD ด้วย

โปรแกรมคำสั่ง

```
DEFINE LCD_DREG PORTD
DEFINE LCD_DBIT 4
DEFINE LCD_RSREG PORTE
DEFINE LCD_RSBIT 2
DEFINE LCD_EREG PORTD
DEFINE LCD_EBIT 1
temp     var byte
temp1   var byte
        TRISA       = %111111
        ADCON1   = 0
```

มีต่อหน้าถัดไป

```
'-----  main program --------
        adcin 0,temp
        adcon1 = 7
        lcdout $fe,1,"Temp = "
        lcdout $fe,$c0,dec temp,$fe,$c5,"C"
'
LOOP:
                gosub get_temp
                if temp <> temp1 then
                            gosub display
                             gosub send_pwm
                             temp1 = temp
                 endif
                 PAUSE  50
                 GOTO  LOOP
        END
'--- End of Main Program -----
'
'-- Subroutine  Start  Here --
'
get_temp:
         adcon1 = 0
         adcin 0,temp
         pause 50
         return
'
display:
        adcon1 = 7
        lcdout $fe,$c0,"   "
        lcdout $fe,$c0,dec temp
        return
'
send_pwm:
        HPWM 1,temp,2000
        return
'
'------- End of Subroutine -----
```

มอบหมายงาน

เนื่องจากไมโครคอนโทรลเลอร์สามารถขับสัญญาณ PWM ออกมาได้ 2 ช่องโดยอิสระ ให้พัฒนาโปรแกรมให้สามารถควบคุมค่าดิวตี้ไซเกิลของสัญญาณ PWM ได้อย่างอิสระทั้ง 2 ช่อง

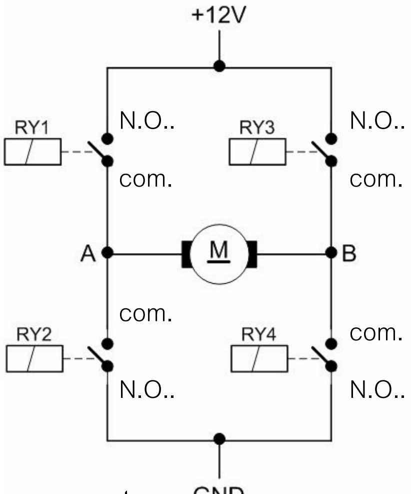
## *Experiment 24* การเขียนโปรแกรมควบคุมมอเตอร์ดีซีแบบหมุนต่อเนื่อง

จุดประสงค์ เพื่อออกแบบระบบการควบคุมมอเตอร์แบบใช้ไฟตรง (DC Motor) ในลักษณะการหมุน – หยุดหมุน และการกลับทิศทางการหมุน แบบการทำงานต่อเนื่อง

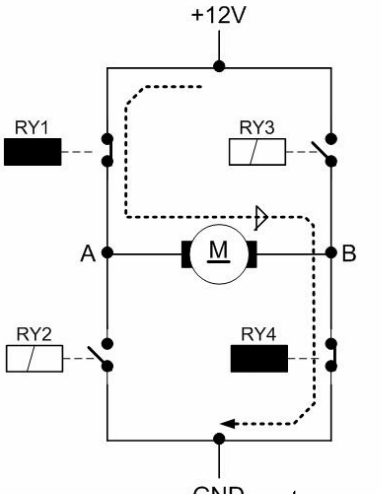
ทฤษฎีพื้นฐาน ในการควบคุมมอเตอร์ไฟตรง (DC Motor) มีการควบคุมอยู่ 3 ลักษณะคือ

1. หมุน - หยุดหมุน (ON - OFF Control)
2. ทิศทางการหมุน (Direction Control)
3. ความเร็วการหมุน (Speed Control)

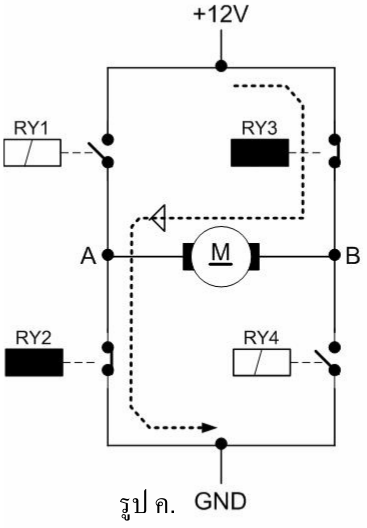
อุปกรณ์ที่ใช้ควบคุมที่นิยมใช้ส่วนมากได้แก่ หน้าคอนแท็ครีเลย์ต่อเป็นวงจรในลักษณะ H – Bridge ที่สามารถควบคุมการหมุน – หยุดหมุน พร้อมทั้งการกลับทิศทางการหมุนได้ แต่การใช้ หน้าคอนแท็ครีเลย์จะไม่สามารถควบคุมความเร็วของการหมุนได้



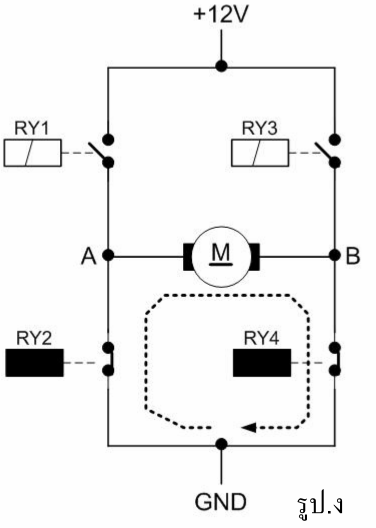

รูป ก.




รูป ข.




รูป ค.




รูป.ง

รูปที่ 1 แสดงบอร์ดรีเลย์แบบ 4 ช่องที่ใช้ทดลองตามใบงาน
(ได้รับความเอื้อเฟื้อจาก เอส เอ เอ็น อิเล็กทรอนิกส์ 02-9438490)

วงจรควบคุมประกอบด้วยรีเลย์ จำนวน 4 ตัว มีหน้าคอนแท็คต่อวงจรตามรูป ก. หากเราสามารถควบคุมรีเลย์ให้ทำงานได้ เราสามารถควบคุมการทำงานของมอเตอร์ได้ 4 โหมด คือ

1. Free Run Stop คือการสั่งให้หน้าคอนแท็คของรีเลย์เปิดทั้ง 4 ตัว ตามวงจรรูป ก. ทำให้มอเตอร์ไม่มีไฟ มอเตอร์จะหยุดไปเองเมื่อหมดแรงเฉื่อย

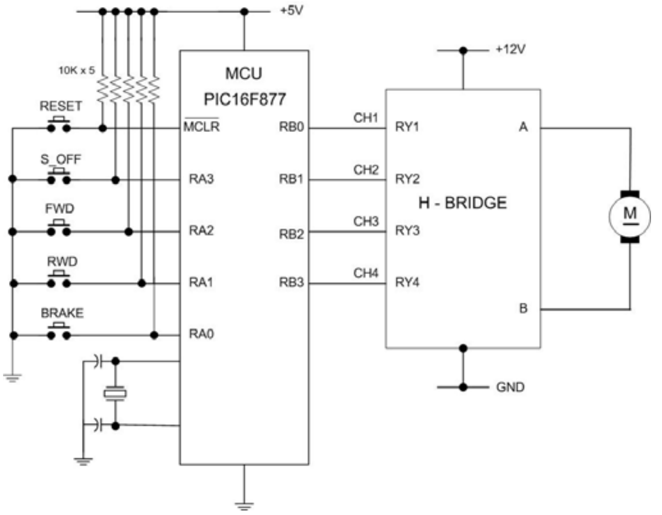
2. Forward Drive คือการสั่งให้รีเลย์ RY1 กับ RY4 ทำงานแค่ 2 ตัว ตามวงจรรูป ข. ทำให้มีกระแสไฟไหลเข้าขั้วมอเตอร์ในทิศทางจาก A ไป B เป็นทิศทางสมมติให้มอเตอร์หมุนในทิศทางตามเข็มนาฬิกา

3. Reward Drive คือการสั่งให้รีเลย์ RY2 กับ RY3 ทำงานแค่ 2 ตัว ตามวงจรรูป ค. ทำให้มีกระแสไฟไหลเข้าขั้วมอเตอร์ในทิศทางจาก B ไป A เป็นทิศทางสมมติให้มอเตอร์หมุนในทิศทางทวนเข็มนาฬิกา

4. Fast Stop หรือ Brake คือการสั่งให้รีเลย์ RY2 กับ RY4 หรือ RY1 กับ RY3 ทำงานคู่ใดคู่หนึ่งเพียง 2 ตัวตามวงจรรูป ง. ในลักษณะนี้ ทำให้ขั้วมอเตอร์ลัดวงจรกัน ในขณะที่มอเตอร์กำลังหมุนด้วยแรงเฉื่อยขดลวดทุ่นอาร์มเมเจอร์ที่ตัดสนามแม่เหล็กจะมีกระแสเหนี่ยวนำทำให้มีสนามแม่เหล็กเกิดขึ้นในทิศทางต้านกับการหมุนของขดลวดทุ่นอาร์มเมเจอร์ ทำให้แกนมอเตอร์หยุดหมุนอย่างรวดเร็ว

หากมีรีเลย์ทำงานนอกเหนือไปจากนี้จะมีผลเกิดขึ้น 2 อย่างคือ มอเตอร์ไม่หมุน กับไฟลัดวงจรทำให้หน้าคอนแท็ค
รีเลย์พัง ดังนั้นในการเขียนโปรแกรมควบคุมรีเลย์ทั้ง 4 ตัวจำเป็นต้องมีการป้องกันผลเสียหายที่เกิดจากการทำงานผิดพลาดนี้ด้วย

*Circuit Diagram*




รูปที่ 2 วงจรต่อทดลองตามใบงาน

<u>หมายเหตุ</u> วงจร H – Bridge ใช้บอร์ดรีเลย์ 4 ช่อง Model SE-RYI-4 โดยการใช้เพียง 4 ช่องโดยเดินสายต่อวงจรตามรูป ก. ใช้สายแพต่อเชื่อมกับ PortB ของบอร์ดทดลอง Model SE-PIC877 จะได้วงจรตามรูปข้างบน

ขั้นตอนการปฏิบัติ 1. ต่อวงจรทดลองตามรูปที่ 2 โดยต่อสายเข้ารีเลย์และมอเตอร์ตามรูป ก.

2. ต่อไฟดีซี 12 โวลท์ เข้ากับวงจร H – Bridge ที่บอร์ดรีเลย์ แล้วพิมพ์โปรแกรม

โปรแกรมคำสั่ง

```
                s_off   var     portb.3
                fwd     var     portb.2
                rwd     var     portb.1
                brake   var     portb.0
                trisa   =       %111111
                trisb   =       %00000000
                adcon1 =        7
                portb   =       0
start:
        if  (s_off = 0) and (fwd = 1) and (rwd = 1) and (brake = 1)  then  portb  =  %0000
        if  (s_off = 1) and (fwd = 0) and (rwd = 1) and (brake = 1)  then  portb  =  %1001
        if  (s_off = 1) and (fwd = 1) and (rwd = 0) and (brake = 1)  then  portb  =  %0110
        if  (s_off = 1) and (fwd = 1) and (rwd = 1) and (brake = 0)  then  portb  =  %1010
        pause  50
        goto  start
        end
'--------------------------  End  of  Program  ---------------------------
```

มอบหมายงาน

เมื่อศึกษาการทำงานของโปรแกรมเข้าใจดีแล้ว ให้ปรับปรุงการทำงานของโปรแกรมให้ดีขึ้นจากเดิม คือ จากเดิมหากกดสวิทช์เปลี่ยนทิศทางการหมุนมอเตอร์จะเปลี่ยนทันทีทันใดทำให้เกิดกระแสกระชากอย่างรุนแรงหากเป็นมอเตอร์ขนาดใหญ่อาจเกิดความเสียหายกับวงจรควบคุม หรือตัวมอเตอร์ หรือวงจรจ่ายไฟ หรืออาจเป็นงานที่มอเตอร์ขับอยู่ก็ได้ การป้องกันคือต้องปรับปรุงแก้ไขโปรแกรมเดิมนี้ ให้มอเตอร์หยุดหมุนก่อนที่จะเปลี่ยนทิศทางการหมุน

## *Experiment 25* การเขียนโปรแกรมควบคุมมอเตอร์ดีซีแบบหมุนตามจังหวะที่กดสวิทช์ (Jogging control)

จุดประสงค์ เพื่อออกแบบระบบการควบคุมมอเตอร์แบบใช้ไฟตรง (DC Motor) ในลักษณะการหมุน – หยุดหมุน และการกลับทิศทางการหมุน แบบทำเมื่อกดสวิทช์ควบคุม (Jogging Control)

ทฤษฎีพื้นฐาน ในการควบคุมมอเตอร์ไฟตรง (DC Motor) แบบ Jogging ใช้วงจรและทฤษฎีการควบคุมเช่นเดียวกันกับ Experiment ที่ 24 แต่จะแตกต่างกันที่การเขียนโปรแกรมควบคุมมอเตอร์ทำงาน คือ การควบคุมแบบ Jogging มอเตอร์จะทำงานก็ต่อเมื่อมีการกดสวิทช์ค้างไว้เท่านั้น หากปล่อยมือมอเตอร์จะหยุดทันที ดังนั้นวงจรสวิทช์ควบคุมจะเหลือเพียง 2 ปุ่มคือ ปุ่มกดให้หมุนตามเข็มนาฬิกา fwd และ หมุนทวนเข็มนาฬิกา rwd

วงจรทดลอง ใช้วงจรทดลองตาม Experiment ที่ 25

โปรแกรมคำสั่ง

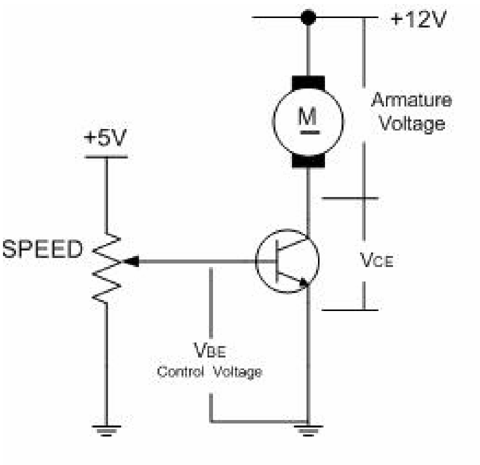
```
                fwd      var      portb.2
                rwd      var      portb.1
                trisa    =        %111111
                trisb    =        %00000000
                adcon1 =          7
                portb    =        0
start:
        if  (fwd = 1) and (rwd = 1)  then  portb  =  %1010
        if  (fwd = 0) and (rwd = 1)  then  portb  =  %1001
        if  (fwd = 1) and (rwd = 0)  then  portb  =  %0110
        pause  50
        goto  start
        end
'--------------------------  End  of  Program  ---------------------------
```

## *Experiment 26* การเขียนโปรแกรมควบคุมความเร็วของมอเตอร์ดีซี

จุดประสงค์ เพื่อออกแบบระบบการควบคุมมอเตอร์ดีซี (DC Motor) ในลักษณะควบคุมความเร็ว ( Speed Control )

ทฤษฎีพื้นฐาน ในการควบคุมความเร็วมอเตอร์ดีซี( DC Motor ) มีวิธีการควบคุม 2 แบบคือ

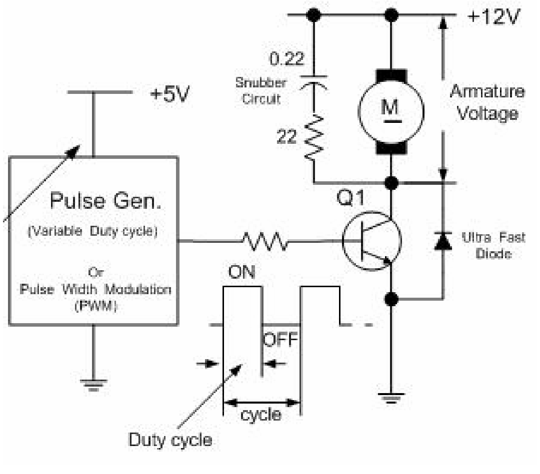
1. การควบคุมแบบลิเนียร์ ( Linear DC voltage Control )




รูปที่ 1 แสดงวงจรการควบคุมแบบลิเนียร์

ตามวงจรรูปที่ 1 พบว่าความเร็วของมอเตอร์จะแปรผันไปตามแรงเคลื่อนไฟที่ตกคล่อมขดลวดอาเมเจอร์ (Armature Voltage) ดังนั้นหากเราสามารถปรับค่าแรงเคลื่อน $V_{CE}$ ได้ ก็จะทำให้ Armature Voltage เปลี่ยนแปลงตามไปด้วย ก็จะมีผลทำให้ความเร็วของมอเตอร์เปลี่ยนแปลงตามไปด้วย แต่เนื่องจาก $V_{CE}$ จะแปรผันตาม แรงเคลื่อน VBE โดยการปรับค่าตัวต้านทานที่ปรับค่าได้ ดังนั้นเราสามารถควบคุมความเร็วมอเตอร์ได้โดยการปรับค่าตัวต้านทาน SPEED ได้ แต่การควบคุมวิธีนี้ไม่ค่อยนิยมใช้กันโดยเฉพาะกับมอเตอร์กำลังมาก ๆ เนื่องจากจะเกิดพลังงานสูญเสียภายในตัวทรานซิสเตอร์ในรูปของความร้อนสูงมากในขณะที่ทรานซิสเตอร์ทำงานอยู่ในช่วง Active หรือ Linear

2. การควบคุมแบบสวิทชิ่ง (Switching Control) คือการป้อนไฟ $V_{BE}$ หรือ Control - Voltage ให้ทรานซิสเตอร์ทำงานในลักษณะ Cutoff หรือหยุดทำงาน และ Saturate หรือ จุดอิ่มตัว โดยจะหลีกเลี่ยงการทำงานในช่วง Active ซึ่งจะทำให้มีการสูญเสียพลังงาน




รูปที่ 2 แสดงวงหลักการควบคุมความเร็วมอเตอร์ดีซีแบบสวิทชิ่ง หรือ PWM

ตามวงจรในรูปที่ 2 ประกอบด้วยตัวกำเนิดสัญญาณพัลส์ส่งไปควบคุมการทำงาน ของทรานซิสเตอร์ในลักษณะ ON และ OFF ทำให้มีกระแสไหลผ่านมอเตอร์หมุนเป็นช่วง ๆ ที่เป็นพัลส์บวก เราเรียกว่า “Duty cycle” โดยที่ตัวผลิตพัลส์ (Pulse Generator) **<u>จะต้องรักษาช่วงเวลาไซเกิลหรือความถี่ให้คงที่อยู่ค่าใดค่าหนึ่ง</u>** โดยปรกติจะเลือกค่าคงที่อยู่ระหว่าง 400 Hz - 20 KHz สำหรับงานควบคุมมอเตอร์ ดังนั้นในทุก ๆ ไซเกิล หากเราปรับเปลี่ยนค่าพัลส์บวก หรือ ค่า duty cycle ก็จะมีผลทำให้ค่าเฉลี่ยแรงเคลื่อนเอ้าท์พุทของทรานซิสเตอร์ที่ไปขับมอเตอร์เปลี่ยนแปลงตามไปด้วย ทำให้ความเร็วของมอเตอร์เปลี่ยนแปลงตามไปด้วย ตามสมการที่ 1 ค่า

$$\% \text{Duty cycle} = \frac{\text{Pulse width}}{\text{Cycle}} \times 100 \quad \text{------------------------} \quad 1$$

duty cycle จะคิดเป็นเปอร์เซ็นต์ ของไซเกิล และค่าเฉลี่ยของแรงเคลื่อนเอ้าพุทที่ขับโหลดคำนวณได้จากสมการที่ 2 ดังต่อไปนี้

$$V_{output} = \% \text{Duty cycle} \times V_{supply} \quad \text{------------------------} \quad 2$$

ตามสมการที่ 2 พบว่า แรงเคลื่อนเอ้าพุทที่จะออกไปขับมอเตอร์ให้หมุนที่ความเร็วเปลี่ยนไปนั้น จะแปรผันโดยตรงกับค่า %Duty- cycle หรือความกว้างของพัลส์ หรือเราเรียกสัญญาณควบคุมแบบนี้ว่า PWM (PWM : Pulse Width Modulation) นั่นหมายความว่า หากเราสามารถเขียนโปรแกรมผลิตสัญญาณ PWM ที่เปลี่ยนค่า Pulse Width ให้ออกมาควบคุมวงจรขับมอเตอร์ได้ เราก็สามารถควบคุมความเร็วของมอเตอร์ได้

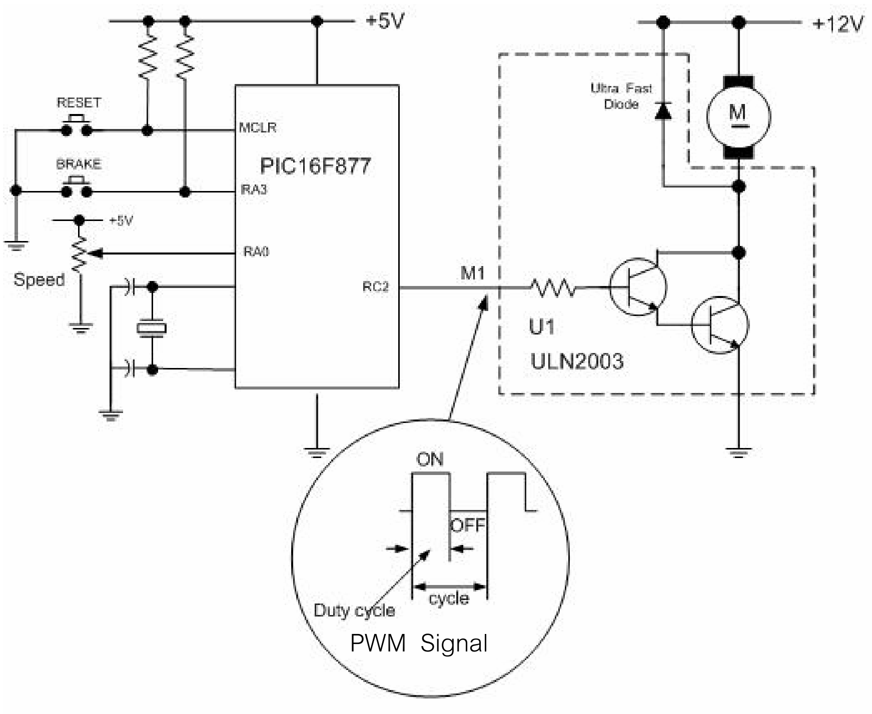
การออกแบบเชื่อมต่อการควบคุมความเร็วด้วยไมโครคอนโทรลเลอร์ จะต้องเลือกไมโคคอนโทรลเลอร์ ชนิดที่มีขีดความสามารถในการสร้างสัญญาณควบคุมแบบ PWM โดยฮาร์ดแวร์ที่ผู้ผลิตสร้างมาให้ เช่น ไมโครคอนโทรลเลอร์เบอร์ PIC16F877 ของบริษัทไมโครชิพ จะมีโมดูลที่สามารถสร้างสัญญาณควบคุมแบบ PWM โดยฮาร์ดแวร์ได้ถึง 2 ชุดที่สามารถสั่งทำงานตามลำพังได้ในขณะที่กำลังประมวลผลคำสั่งอื่นในโปรแกรมอยู่ ทำให้การผลิตสัญญาณควบคุมเป็นไปอย่างต่อเนื่องไม่ขาดช่วง โดยได้กำหนดขา I/O ไว้ 2 ขา คือ RC2

สำหรับ HPWM ช่อง 1 และ ขา RC1 สำหรับ HPWM ช่อง 2

รูปแบบการใช้คำสั่ง คือ

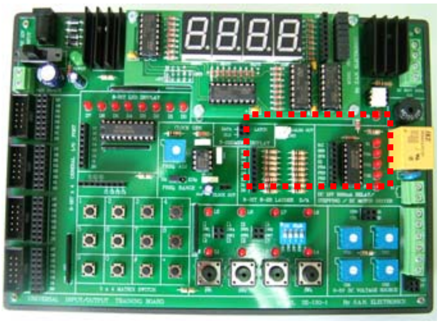
HPWM channel, duty cycle, frequency รายละเอียดให้ศึกษาได้จากคู่มือการใช้โปรแกรม PIC BASIC PRO ทดลองตามใบงาน จะใช้ขา RC2 ซึ่งผลิตสัญญาณ PWM channel 1 และค่า duty cycle จะใช้ค่า A/D ที่รับมาจาก RA0 นำมาใช้ปรับเปลี่ยนค่า duty cycle เพื่อควบคุมความเร็วมอเตอร์

วงจรทดลอง




รูปที่ 3 แสดงวงจรการควบคุมความเร็วมอเตอร์ดีซีด้วยไมโครคอนโทรลเลอร์

รูปที่ 4 บอร์ดไมโคร PIC40-1
PIC16F877-20



รูปที่ 5 บอร์ดUniversal I/O ขับมอเตอร์ดีซี
โดยใช้วงจรขับ ULN2003

หมายเหตุ บอร์ดไมโคร PIC40-1 และ Universal I/O ตามรูปที่ 4 และ 5 ได้รับความเอื้อเฟื้อจาก เอส เอ เอ็น อิเล็กทรอนิกส์ 02-9438490 01-2535810 Mail sombonie@hotmail.com

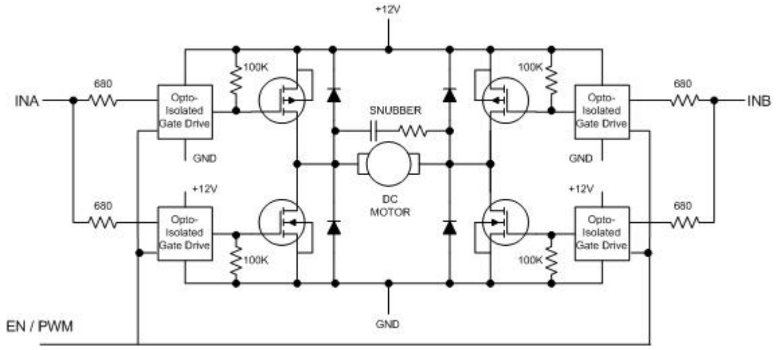
โปรแกรมคำสั่ง

```
'*************************************************
'*  Name    : Speed.BAS
'*  Author  : Somboon  Niamglam
'*  Notice  : PIC16F877  MCU
'*          : Speed   Control
'*  Date    : 15/09/2005
'*  Version : 1.0
'*************************************************
      DUTY   VAR  BYTE
      SPEED   VAR  BYTE
      BRAKE   VAR  PORTA.3
      ADCON1=14
      TRISA =%111111
      TRISC =%00000000
'
START: IF (BRAKE=0)  THEN
                  HPWM 1,0,1000
            ELSE
                  ADCIN 0,SPEED
                  HPWM 1, speed, 1000
            ENDIF
            PAUSE 50
            GOTO   START
            END
```

## *Experiment 27* การเขียนโปรแกรมควบคุมทั้งความเร็วและทิศทางของมอเตอร์ดีซี

จุดประสงค์ เพื่อออกแบบระบบการควบคุมมอเตอร์ดีซี (DC Motor) ในลักษณะควบคุมความเร็ว ( Speed Control ) พร้อมทั้งควบคุมทิศทางการหมุน (Direction Control)

ทฤษฎีพื้นฐาน ในการควบคุมทั้งความเร็ว และทิศทางการหมุนของมอเตอร์ดีซี (DC Motor) จำเป็น ต้องใช้วงจรอิเล็กทรอนิกส์ H-Bridge เพื่อให้สามารถเป็นสวิทช์ตัดต่อกระแสไฟควบคุมมอเตอร์แทนหน้าคอนแทครีเลย์ที่เร็วพอที่จะใช้สัญญาณ PWM ควบคุมความเร็วไปพร้อมกับการควบคุมการกลับทิศทางหมุนของมอเตอร์




รูปที่ 1 วงจรแสดงการทำงานของ MOSFET H – Bridge ขับมอเตอร์ดีซี

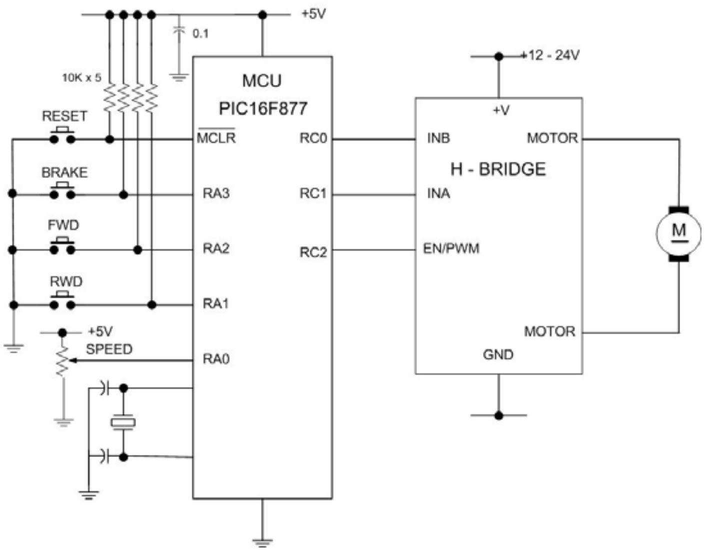
รูปที่ 2 บอร์ด MOSFET H-Bridge ขนาด 30 A ขับมอเตอร์ดีซี

การทำงานของวงจร ประกอบด้วย POWER MOSFET จำนวน 4 ตัว ทำหน้าที่เป็นหน้าคอนแท็คสวิทช์ จะถูกควบคุมโดยแรงดันจากวงจร Opto – Isolated Gate Drive อินพุทของวงจร Opto – Isolated Gate Drive แต่ละข้าง จะต่อรวมกันเป็น INA และ INB POWER MOSFET แต่ละข้างของ H – Bridge จะทำงานแบบ Complementary โดยที่ไม่มีโอกาสจะทำงานพร้อมกันได้เลย ได้ออกแบบให้มีขาสำหรับป้อนสัญญาณ EN / PWM เพื่อสามารถกำหนดช่วงเวลาในการเปิด – ปิด สวิทช์ของ POWER MOSFET แต่ละชุดได้ เพื่อใช้ในการควบคุมความเร็ว การทำงานของ H-Bridge เป็นไปตามตาราง รูปที่ 3

| EN | INB | INB | |
|---|---|---|---|
| 0 | x | x | Brake |
| 1 | 0 | 1 | Forward |
| 1 | 1 | 0 | Reward |
| 1 | 1 | 1 | Brake |
| 1 | 0 | 0 | Brake |

รูปที่ 3 แสดงการทำงานของวงจร MOSFET H – Bridge รุ่น SE – HB – 60

วงจรทดลอง




รูปที่ 3 แสดงการต่อวงจรควบคุม MOSFET H-Bridge ด้วยไมโครคอนโทรลเลอร์

โปรแกรมคำสั่ง

```
    DUTY     VAR  BYTE
    SPEED   VAR  BYTE
    BRAKE   VAR  PORTA.3
    FWD      VAR  PORTA.2
    RWD      VAR  PORTA.1
    ADCON1 = 14
    TRISA  = %111111
    TRISC  = %00000000
    ‘
START:  IF (BRAKE=0) AND (FWD=1) AND (RWD=1) THEN
              HPWM 1,0,1000
        ELSE
             IF (BRAKE=1) AND (FWD=0) AND (RWD=1) THEN LOW PORTC.0:HIGH PORTC.1
             IF (BRAKE=1) AND (FWD=1) AND (RWD=0) THEN HIGH PORTC.0:LOW PORTC.1
              ADCIN  0, SPEED
             DUTY = SPEED
             HPWM 1, DUTY, 1000
         ENDIF
         PAUSE  50
         GOTO  START
         END
```

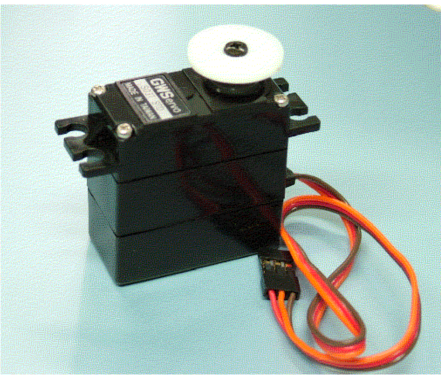
มอบหมายงาน

เมื่อศึกษาและปฏิบัติเข้าใจดีแล้ว ให้ปรับปรุงโปรแกรมให้มีการชะลอความเร็วก่อนเปลี่ยนทิศทางการหมุน

## *Experiment 28* การเขียนโปรแกรมควบเซอร์โวมอเตอร์( Servo Motor Controls )

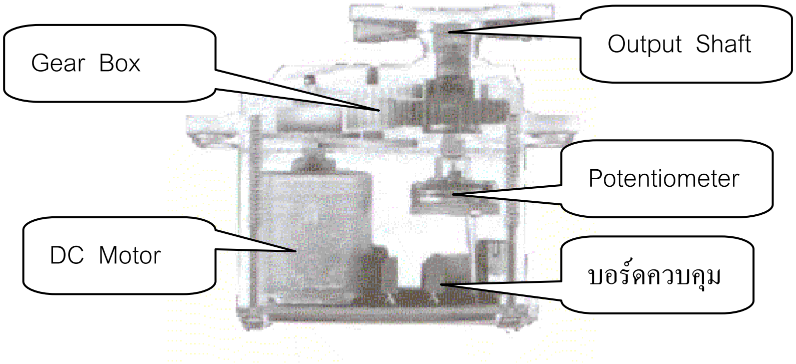
**จุดประสงค์** เพื่อสามารถออกแบบระบบและเขียนโปรแกรมควบคุมเซอร์โวมอเตอร์ ขนาดเล็ก (DC Servo Motor) ที่ใช้สัญญาณความกว้างพัลส์มาตรฐานได้

**ทฤษฎีพื้นฐาน** DC เซอร์โวมอเตอร์ คือ มอเตอร์ไฟตรงขนาดเล็กที่ถูกประกบเข้ากับส่วนประกอบต่าง ๆ ได้แก่ชุดเกียร์ทด ชุดวงจรควบคุมตำแหน่งการหมุนไว้ในโมดูลเดียวกัน โดยมีสายต่อใช้งานเพียง 3 เส้น คือ V+ GND และ Control line ซึ่งเป็นสายควบคุมที่ทำให้มอเตอร์หมุนซ้ายขวาโดยใช้สายสัญญาณ PWM เป็นตัวควบคุม ส่วนแรงเคลื่อน V+ ที่ป้อนให้ขับเซอร์โวมอเตอร์อยู่ที่ประมาณ 4 ถึง 6 โวลท์ เซอร์โวมอเตอร์มีข้อดีคือ มีขนาดเล็ก น้ำหนักเบา แต่ให้แรงบิดสูง และกินพลังงานน้อย ใช้ระดับสัญญาณควบคุมแบบ TTL Level ต่อโดยตรงกับไมโครคอนโทรลเลอร์ได้เลยโดยไม่ต้องมีวงจรขับ



รูปที่ 1 เซอร์โวมอเตอร์ รุ่น S03T STD

เซอร์โวมอเตอร์แบบนี้จะมีวงจรควบคุมอยู่ในตัว การควบคุมตำแหน่งการหมุนโดยทั่วไปจะมีช่วงประมาณ 180 องศา ถึง 210 องศา



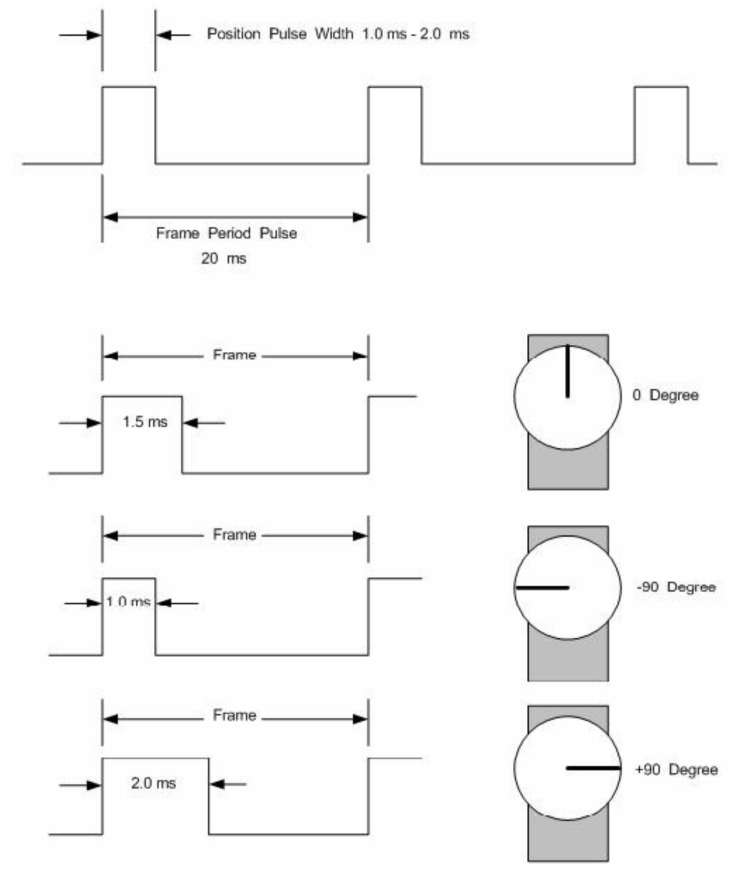

รูปที่ 2 แสดงโครงสร้างของเซอร์โวมอเตอร์

หลักการทำงานของ Servo Motor

การควบคุมตำแหน่งการหมุนของเซอร์โวมอเตอร์ ทำได้โดยการป้อนสัญญาณความกว้างพัลส์เข้าที่ขา Control line ตำแหน่งและทิศทางการหมุนของแกนเอาท์พุทจะขึ้นอยู่กับความกว้างของพัลส์ สัญญาณควบคุมจะประกอบด้วย

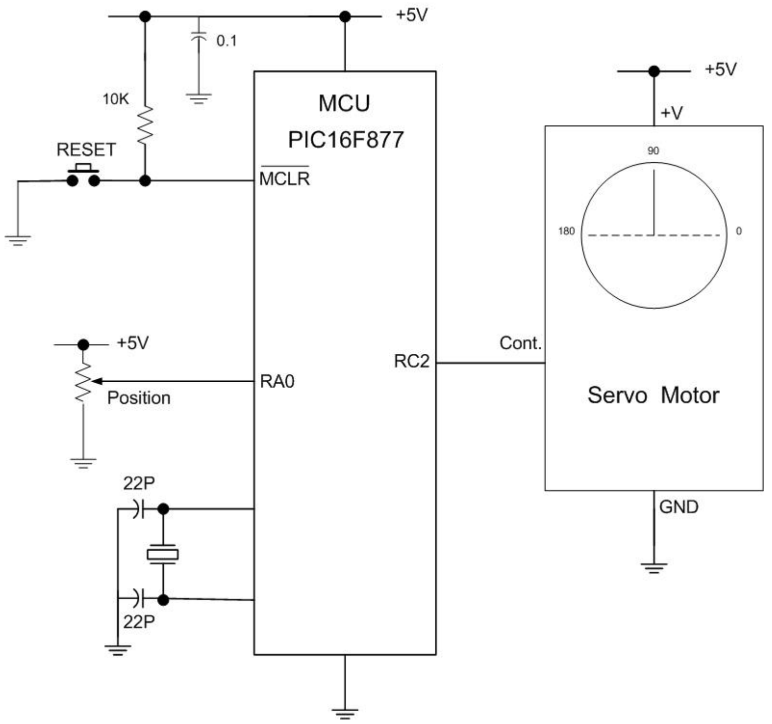
- Frame Period Pulse เป็นสัญญาณพัลส์ต่อเนื่องโดยจะเริ่มต้นห่างกันทุก ๆ 20 ms ตลอดเวลาเพื่อรักษาสภาพตำแหน่งการหมุนเอาไว้

- Position Pulse Width เป็นค่าความกว้างของยอดพัลส์ของ Frame Period Pulse ใช้เป็นค่าควบคุมตำแหน่งและทิศทางการหมุน โดยจะมีค่าอยู่ระหว่าง 1.0 ms - 2.0 ms โดยจะมีจุดอ้างอิง 3 จุดที่สามารถควบคุมและรักษาตำแหน่งตั้งแต่ 0 - 180 องศา ตามรูปที่ 3




รูปที่ 3 แสดง Timing Diagram ของ Control Pulse

ตามรูปที่ 3 เป็นค่าประมาณในการควบุคมรักษาตำแหน่งของเซอร์โวมอเตอร์โดยทั่วไป แต่อาจมีบางยี่ห้อที่อาจแตกต่างไปบ้าง ให้ศึกษาจากคู่มือของแต่ละยี่ห้อไป




รูปที่ 4 แสดงการต่อวงจรควบคุมเซอร์โวมอเตอร์ด้วยไมโครคอนโทรลเลอร์

การควบคุมเซอร์โวมอเตอร์ด้วยไมโครคอนโทรลเลอร์ตระกูล PIC โดยใช้คำสั่งภาษาเบสิก ในการสัญญาณควบคุมตำแหน่งคือ PULSOUT โดยมีรูปแบบดังนี้ คือ

```
PULSOUT   Pin, Period
```

ค่า Period ของคำสั่งจะเปลี่ยนแปลงไปตามสัญญาณนาฬิกาที่จ่ายให้กับ CPU ซึ่งทำให้ คำสั่ง DEFINE OSC ไม่มีผลกับการตั้งค่า Period ของคำสั่ง PULSOUT ตัวอย่าง เช่น ถ้าใช้ CRYSTAL ความถี่ 4 MHz จะทำให้หนึ่งหน่วยค่า Period = 10 uS

ดังนั้น ถ้าต้องการสร้างสัญญาณพัลส์ 1 ms จะต้องใช้คำสั่งดังนี้

PULSOUT Pin, 100

Period 100 = 100 X 10uS = 1000 uS = 1 mS

Period 150 = 150 X 10uS = 1500 uS = 1.5 mS

Period 200 = 200 X 10uS = 2000 uS = 2 mS

ดังนั้นค่า Period ในการควบคุมเซอร์โวมอเตอร์จะอยู่ระหว่าง 100 - 200 กรณีที่ใช้ สัญญาณนาฬิกา CPU ที่ 4 MHz

โปรแกรมคำสั่ง การทำงานของโปรแกรมคำสั่งจะอาศัยการต่อวงจรตามรูปที่ 4 คือ จะต้องเขียนโปรแกรมให้รับค่าอนาล็อกจากขา RA0 มาใส่ไว้ในตัวแปร และเขียนคำสั่งประมวลผลให้มีค่าจาก 0 – 255 มาเป็น 100 - 200 เพื่อนำมากำหนดเป็นค่า Period ในคำสั่ง PulsOut เพื่อส่งออกไปควบคุมเซอรโวมอเตอร์ต่อไป

```
'*******************************************
'*  Name    :  Servo.BAS                   *
'*  Author  : Somboon  Niamglam            *
'*  Date    : 3/10/2005                    *
'*  Version : 1.0                          *
'*  Notes   :                              *
'*          :                              *
'*******************************************
      INCLUDE  "modedefs.bas"
      TRISC   =  %00000000
      TRISA   =  %111111
      adcon1  = 7
      an0     var    byte
      temp    var    byte
      pos     var    byte
      Low     Portc.1
'
start:
      adcin 0, an0                      'Get A/D value
      if an0 = 0 then an0 = 2           'Lower Limit
      if an0 => 200 then an0 = 200      'Upper Limit
      temp =  an0 / 2                   'temp value not over 100
      pos = 100 + temp                  'pos value 100 - 200
      pulsout portc.1, pos              'Send Servo pulse
      pause 18
      goto start
      end
‘----------------- End  of   Program  ---------------------
```

มอบหมายงาน เมื่อศึกษาและปฏิบัติจนเข้าใจดีแล้ว ให้นำไปประยุกต์ใช้ควบคุมแขนกลแบบหลายข้อต่อ

## *Experiment 29* การเขียนโปรแกรมควบคุมเต็ปปิ้งมอเตอร์( Stepping Motor Controls )

จุดประสงค์ เพื่อเรียนรู้โครงสร้าง หลักการทำงานของ และการควบคุมการทำงานของ สเต็ปปิ้งมอเตอร์

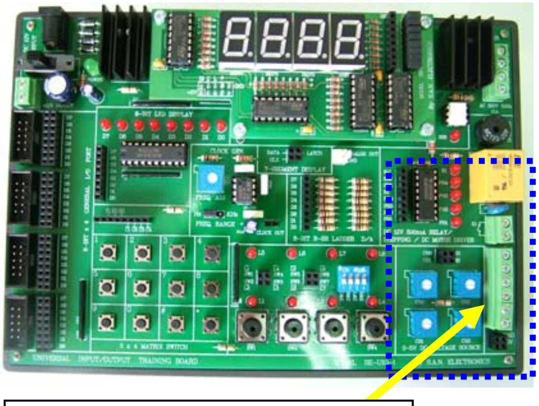
ทฤษฎีพื้นฐาน

สเต็ปปิ้งมอเตอร์ ( Stepping Motor ) หรือเรียกอีกอย่างหนึ่งว่า พัลส์มอเตอร์ (Pulse Motor) มีลักษณะการหมุนแบบสเต็ปตามพัลส์ที่มากระตุ้น โครงสร้างประกอบด้วยขดลวดอาเมเจอร์(Armature winding) อย่างน้อย 2 ขด พันอยู่ที่ส่วน Stator มีลักษณะการพันเป็น 2 แบบคือ Uni-polar และ Bi-polar โดยมีข้อสังเกตคือ แบบ Uni-polar จะมีสายต่อไฟเข้าอยู่ 4 เส้น ส่วนแบบ Bi-polar จะมีสายต่อไฟอยู่ 5 – 6 เส้น นอกจากนี้ยังมีสเต็ปปิ้งมอเตอร์แบบ 5 เฟส ที่ต้องการความละเอียดของมุมตำแหน่งการหมุน (Step angle) โดยทั่วไปจะมีตั้งแต่ 1.8 องศา 7.5 องศา เป็นต้น การพันของขดลวดจะทำให้เกิดสนามแม่เหล็ก 4 ขั้ว ที่หน้าขั้วจะเซาะร่อง (Pole teeth) เล็ก ๆ ที่ Rotor จะเป็นเหล็กทรงกลม ที่ส่วนผิวจะเซาะเป็นร่องเช่นเดียวกับ Stator ในลักษณะที่ขั้วหนึ่ง ซี่ฟันทั้งของ Rotor กับ Stator จะตรงกัน จะออกแบบให้ซี่ฟันระหว่าง Stator กับ Rotor จะเยื้องกันตามจำนวนองศา ของ Step angle ที่ระบุไว้

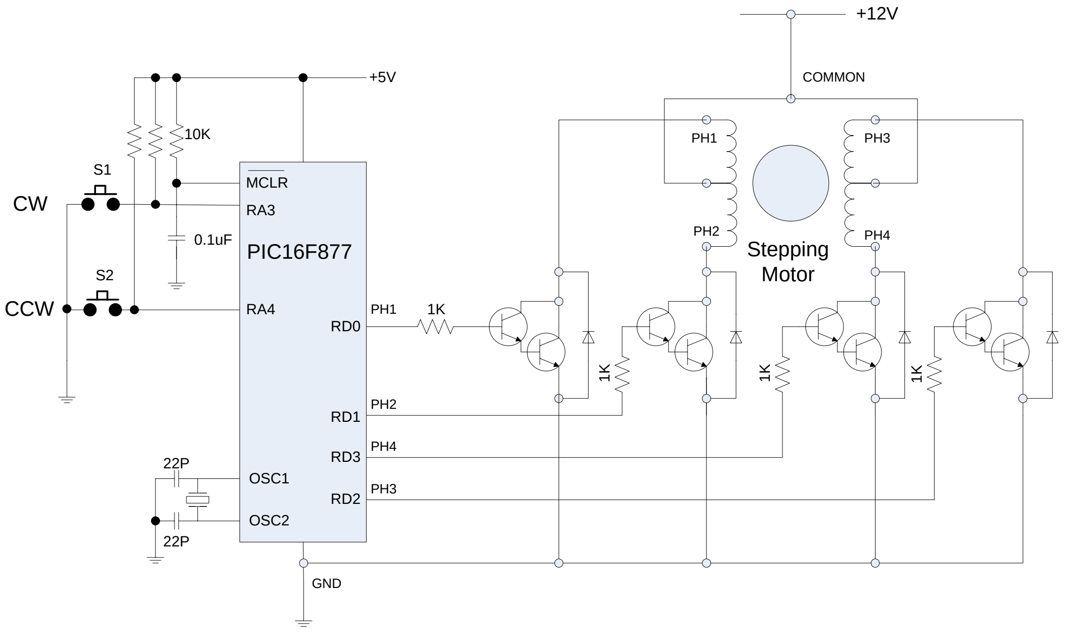
รูปที่ 1 STEPPING MOTOR

รูปที่ 2 โครงสร้างของ Rotor

รูปที่ 3 โครงสร้างของ Stator



รูปที่ 4 ขั้วต่อขับ Stepping Motor




รูปที่ 1 วงจรการควบคุมสเต็ปปิ้งมอตอร์แบบไบโพล่า

**<u>หมายเหตุ</u>** การต่อวงจรทรานซิสเตอร์ขับ แบบดาร์ลิงตัน(Darlington) เพื่อทำให้สามารถขับกระแสได้มากขึ้น และไดโอดที่ต่อคล่อมขา C และ E ทั้ง 4 ตัวมีไว้เพื่อระบายกระแสย้อนผ่านกลับไปยังแหล่งจ่ายไฟ เนื่องจากการเกิด back EMF ในช่วง OFF หากไม่มีจะทำให้รอยต่อทรานซิสเตอร์เสียหายเฉียบพลันได้ โดยปกติจะต้องใช้ไดโอดชนิด Ultra Fast ที่มีค่า Trr (Turn off reverse recovery time) ไม่เกิน 200 นาโนวินาที

การควบคุมการหมุน โดยการป้อนสัญญาณพัลส์ไปกระตุ้นที่ขั้วต่อขดลวด Stator โดยผ่านทรานซิสเตอร์ขยายกำลัง กระตุ้นเป็นจังหวะเรียงกันไป โดยมีลักษณะการกระตุ้นหลายแบบดังนี้

1. แบบเฟสเดียว Full Step โดยการกระตุ้นเรียงกันไปทีละขั้วเฟส ตามตาราง ที่ 1

| | PH1 | PH2 | PH3 | PH4 |
|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 |
| 2 | 0 | 1 | 0 | 0 |
| 3 | 0 | 0 | 1 | 0 |
| 4 | 0 | 0 | 0 | 1 |

ตารางที่ 1 Full–step
1 phase

2. แบบสองเฟส Two - phase โดยการกระตุ้นเรียงกันไปทีละ 2 ขั้วเฟส ตามตารางที่ 2

| | PH1 | PH2 | PH3 | PH4 |
|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 |
| 2 | 0 | 1 | 1 | 0 |
| 3 | 0 | 0 | 1 | 1 |
| 4 | 1 | 0 | 0 | 1 |

ตารางที่ 2 Full-step
2 phase

3. แบบครึ่งสเต็ป Half – step สำหรับงานที่ต้องการความละเอียดมากกว่า Full – step ตามตารางที่ 3

| | PH1 | PH2 | PH3 | PH4 |
|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 |
| 2 | 1 | 1 | 0 | 0 |
| 3 | 0 | 1 | 0 | 0 |
| 4 | 0 | 1 | 1 | 0 |
| 5 | 0 | 0 | 1 | 0 |
| 6 | 0 | 0 | 1 | 1 |
| 7 | 0 | 0 | 0 | 1 |
| 8 | 1 | 0 | 0 | 1 |

ตารางที่ 3 Half-step

โปรแกรมคำสั่งที่ 1 แบบ 1 เฟส Full – step

```
S1        VAR      PORTA.3
rot   var byte
      TRISA     =   %111111
      TRISD     =   %00000000
      ADCON1  = 7
      rot = 0
start:    PAUSE  300
         IF  (S1 =  0) THEN
                  lookup rot,[%1000,%0100,%0010,%0001],portd
                  rot   =  rot + 1
                  if rot = 4 then rot = 0
         else
                  lookup rot,[%1000,%0100,%0010,%0001],portd
                  if rot = 0 then rot = 4
                  rot   = rot - 1
         endif
         GOTO   start
         END
```

การทำงานของโปรแกรมเริ่มที่ start หน่วงเวลา 300 mS และจะตรวจสอบสถานการณ์กดสวิทช์ S1 ถ้ามีการกดค้างอยู่ จะทำให้การทำงานอยู่ในบล็อก if จะทำให้ปล่อยสัญญาณขับสเต็ปปิ้งมอเตอร์ให้หมุนขวา โดยการนำเอารูปแบบการปล่อยสัญญาณที่อยู่ในตาราง lookup[ ] ออกไปยัง Portd หากไม่เป็นเช่นนั้น ก็จะทำงานใน

บล็อก else ซึ่งโปรแกรมจะนำรูปแบบสัญญาณหมุนซ้ายออกมาจากตาราง lookup [ ] ออกไปยัง Portd ทำงานวนแบบนี้ตลอด

โปรแกรมคำสั่งที่ 2 แบบ 2 เฟส Full – step

```
S1    VAR            PORTA.3
S2    VAR            PORTA.4
stat   var            bit
rot     var            byte
      TRISA      =   %111111
      TRISD      =   %00000000
      ADCON1   = 7
      stat          = 1
      rot            = 0
begin:
      PAUSE  300
      if (s1 = 0) then stat = 1
      if (s2 = 0) then stat = 0
      IF  (stat =  1) THEN
               lookup rot,[%1100,%0110,%0011,%1001],portd
               rot   =  rot + 1
               if rot = 4 then rot = 0
      else
                lookup rot,[%1100,%0110,%0011,%1001],portd
                if rot = 0 then rot = 4
                rot   = rot - 1
      endif
      GOTO   begin
      END
```

โปรแกรมคำสั่งที่ 3 Half – step (ให้สังเกตความแตกต่างระหว่าง Full-step กับ Half-step)

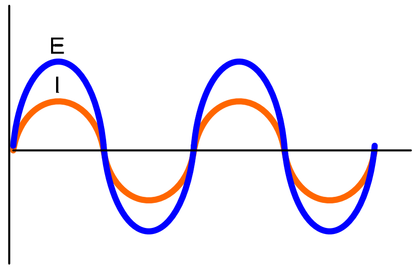
```
S1    var   PORTA.3
S1    var   PORTA.4
stat   var   bit
rot     var   byte
      TRISA     =    %111111
      TRISD     =    %00000000
      ADCON1  =      7
      stat         =      1
      rot           =     0
begin:
      if (s1 = 0) then stat = 1
      if (s2 = 0) then stat = 0
      PAUSE  200
      IF  (stat =  1) THEN
             lookup rot,[%1000,%1100,%0100,%0110,_
                            %0010,%0011,%0001,%1001],portd
              rot   =  rot + 1
              if rot = 9 then rot = 0
      else
              lookup rot,[%1000,%1100,%0100,%0110,_
                            %0010,%0011,%0001,%1001],portd
              if rot = 0 then rot = 9
              rot   = rot - 1
      endif
      GOTO   begin
      END
```

*Experiment 30* การเขียนโปรแกรมควบคุมโหลดไฟฟ้ากระแสสลับ(AC Loads Control)
(AC Loads Interfacing Techniques)

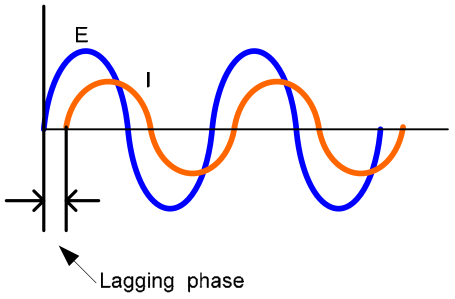
จุดประสงค์ เพื่อเรียนรู้หลักการเชื่อมประสานการควบคุมระหว่างไมโครคอนโทรลเลอร์และอุปกรณ์กำลังที่ใช้ไฟฟ้ากระแสสลับ

ทฤษฎีพื้นฐาน

การเชื่อมประสานการควบคุมระหว่างไมโครคอนโทรลเลอร์และอุปกรณ์กำลังที่ใช้ไฟฟ้ากระแสสลับเป็นพื้นฐานสำคัญในการนำไปประยุกต์สร้างโครงงาน และสิ่งประดิษฐ์รวมถึงผลิตภัณฑ์ที่ใช้ระบบไฟฟ้าและอิเล็กทรอนิกส์เป็นตัวควบคุมการทำงานได้อย่างมากมาย โหลดไฟฟ้ากระแสสลับโดยทั่วไปแบ่งออกเป็น 2 ชนิด ได้แก่ Resistive load คือโหลดที่เป็นความต้านทานบริสุทธิ์ เช่น หลอดไฟแบบมีไส้ ขดลวดความร้อนเป็นต้น โหลดชนิดนี้ เมื่อมีกระแสไฟฟ้าสลับไหลผ่านการเกิดแรงเคลื่อน E และกระแส I จะเกิดขึ้นพร้อมกันในเวลาเดียวกัน เรียกว่า In-phase กัน ส่วนโหลดอีกชนิดหนึ่งได้แก่ Inductive load คือโหลดที่เป็นขดลวดเหนี่ยวนำ เช่น มอเตอร์ไฟฟ้า หม้อแปลงไฟฟ้า โซลินอยด์ โหลดชนิดนี้เมื่อใส่ไฟฟ้ากระแสสลับจะทำให้กระแสไฟฟ้า I ล้าหลังแรงเคลื่อน E อยู่ค่าหนึ่ง เราเรียกว่า Lagging ปรากฏการณ์แบบนี้การควบคุมหากไม่คำนึงถึงเรื่องนี้ จะเกิดปัญหาขึ้นได้




รูปที่ 1 E และ I ใน Resistive load

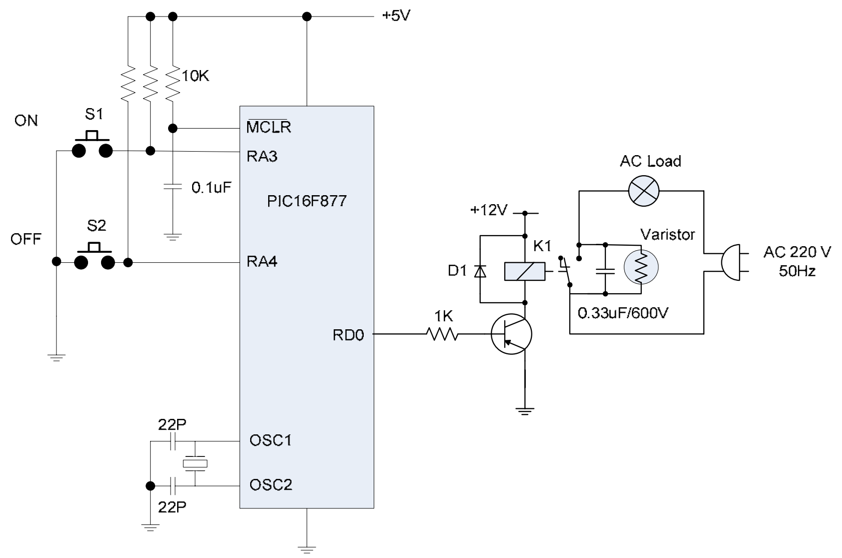



รูปที่ 2 E และ I ใน Inductive load

เนื่องจากระดับแรงเคลื่อนของสัญญาณควบคุม และ สัญญาณอุปกรณ์กำลังมีขนาดแตกต่างกันมาก และเป็นอันตรายมากหากสัญญาณดังกล่าวลัดวงจรถึงกัน ดังนั้นในการแยกสัญญาณทั้งสองส่วนจึงเป็นเรื่องจำเป็นและสำคัญมาก เทคนิคที่นิยมใช้กันโดยทั่วไปได้แก่ การใช้รีเลย์ และใช้อุปกรณ์เชื่อมต่อทางแสง

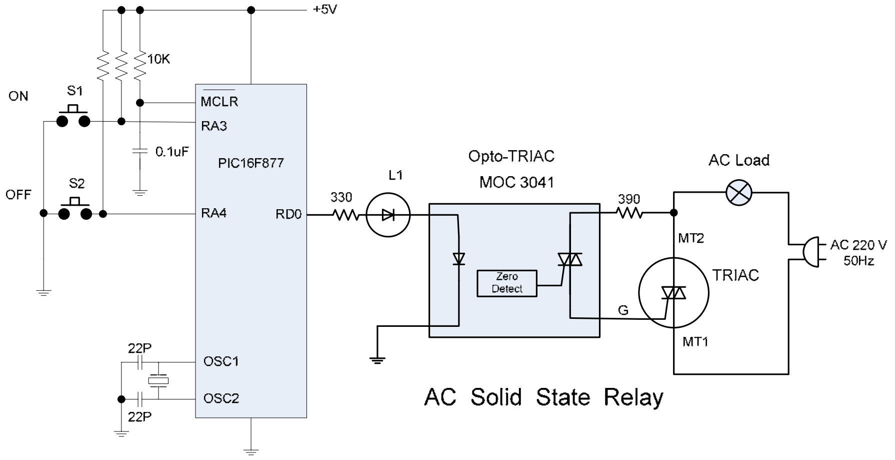
การเชื่อมต่อสัญญาณควบคุมโดยใช้รีเลย์ เป็นวิธีขั้นพื้นฐานดั้งเดิมปัจจุบันก็ยังคงใช้กันอยู่ เนื่องจากออกแบบง่าย อุปกรณ์หาซื้อได้ง่าย แต่มีข้อเสีย คือหน้าคอนสัมผัสชำรุดได้ง่าย มีเสียงดัง ไม่สามารถใช้งานตัดต่อที่มีความเร็วสูง ๆ ได้ ถ้าออกแบบไม่ดีอาจเกิดประกายไฟระหว่างหน้าสัมผัส และเกิดคลื่นความถี่ไปรบกวนการทำงานกับอุปกรณ์อื่น ๆ ได้ ประกายไฟเกิดจากการกระโดดข้ามของประจุไฟฟ้าจำนวนมากจนเกิดความร้อนและเห็นประกายไฟจากการลุกไหม้ของอากาศขณะที่หน้าสัมผัสกำลัง

จะจาก หรือกำลังจะติด การระงับการเกิดการกระโดดของประจุไฟฟ้าดังกล่าวในทางปฏิบัติ จะใช้ตัววาริสเตอร์ (Varistor) และตัวคาปาซิเตอร์แบบไม่มีขั้ว(None Polar) ขนาดความจุ 0.33 ไมโครฟารัด 600 V ต่อขนานไว้กับหน้าคอนแท็คของรีเลย์ ตามวงจรรูปที่ 3



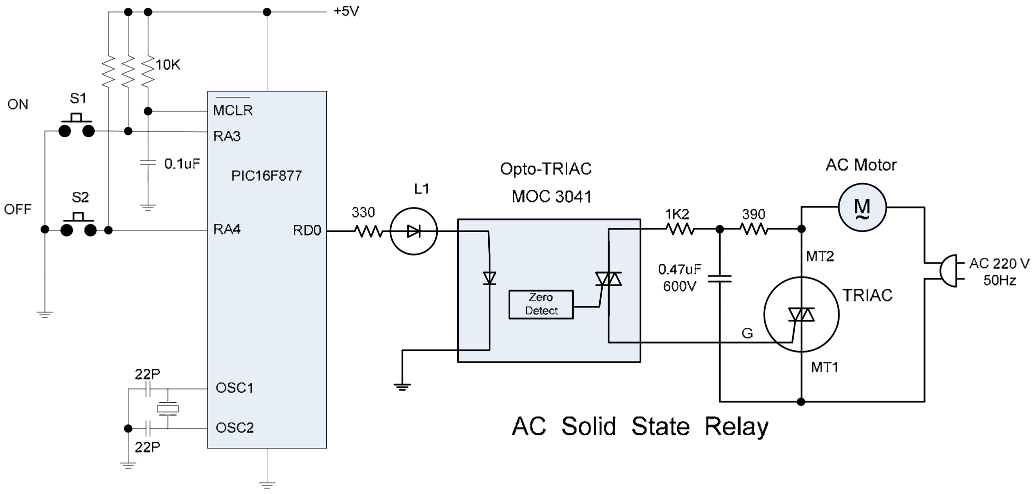

รูปที่ 3 แสดงวงจรเชื่อมต่อการควบคุมอุปกรณ์ไฟฟ้ากำลังด้วยรีเลย์

การเชื่อมต่อสัญญาณควบคุมทางแสง (Opto-Couple) วิธีการนี้จะมีความปลอดภัยสูงกว่าแบบใช้รีเลย์ เนื่องจากไฟสัญญาณจากบอร์ดถูกตัดขาดจากไฟฟ้ากำลังของโหลด ถึงแม้การใช้รีเลย์จะแยกส่วนกำลังโหลดด้วยหน้าคอนแท็คก็ตาม แต่อันตรายที่จะเกิดกับขา I/O ของไมโครคอนโทรลเลอร์ก็ยังมีอยู่ คือแรงเคลื่อนย้อนกลับ (back EMF) ของขดลวดรีเลย์อาจย้อนกลับเข้าสู่ขา I/O ได้ วิธีการเชื่อมต่อการควบคุมไฟฟ้ากำลังทางแสง เราเรียกรวมว่า " โซลิดสเตทรีเลย์" (SSR: Solid State Relay) อุปกรณ์ตัวหลัก ๆ ได้แก่ Opto-TRIAC ทำหน้าที่เชื่อมต่อการควบคุมระหว่าง MCU กับ main TRIAC และ main TRIAC ทำหน้าที่เป็นสวิทช์แบบไร้หน้าสัมผัส ตามวงจรรูปที่ 4




รูปที่ 4 แสดงวงจรเชื่อมต่อการควบคุมอุปกรณ์ไฟฟ้ากำลังด้วยโซลิดสเตทรีเลย์

ทั้งวงจรตามรูปที่ 3 และ 4 การเขียนโปรแกรมควบคุมจะไม่ต่างกัน สามารถทดแทนกันได้ แต่การออกแบบวงจร Solid State Relay ตามรูปที่ 4 จำเป็นต้องคิดถึงชนิดของโหลดด้วยว่า เป็นชนิด Resistive load หรือ Inductive load ตามรูปที่ 4 เป็นวงจรที่ออกแบบสำหรับใช้งานกับ Resistive load เช่น หลอดไฟมีไส้ ขดลวดความร้อน เป็นต้น หากโหลดเป็นชนิด Inductive เช่น มอเตอร์ไฟฟ้า หม้อแปลง หรืองานที่เป็นขดลวดเหนี่ยวนำอื่น ๆ จำเป็นต้องออกแบบวงจรให้มีการหน่วงเวลาการจุดชนวนให้ main TRIAC เนื่องจากในโหลดดังกล่าว กระแส I จะล้าหลังแรงเคลื่อน E อยู่ค่าหนึ่ง ตามวงจรในรูปที่ 5




รูปที่ 5 แสดงวงจร Solid State Relay ที่ใช้กับ Inductive load

โปรแกรมคำสั่ง

```
S1   var PORTA.3
S2   var PORTA.4
L1   var portd.0
     TRISA    =   %111111
     ADCON1 = 7
     low L1
start:
     if (s1 = 0)and (s2 = 1) then high L1
     if (s1 = 1)and (s2 = 0) then low  L1
     PAUSE  200
     GOTO   start
     END
```

**<u>หมายเหตุ</u>** บอร์ดที่ใช้การทดลองทั้งหมด ใช้บอร์ด MCU ร่วมกับบอร์ด Universal I/O

## *Experiment 31* การเขียนโปรแกรมเพื่อรองรับการขัดจังหวะทำงาน( Interrupt Service Routine)

จุดประสงค์ เพื่อเรียนรู้เทคนิคการเขียนโปรแกรมขอรับบริการ Interrupt จากสัญญาณภายนอก

ทฤษฎีพื้นฐาน

การขัดจังหวะการทำงาน(Interrupt) หมายถึงการทำให้ CPU ของไมโครคอนโทรลเลอร์พักการประมวลผลในตำแหน่งขณะนั้นอยู่ ให้มาทำงานในตำแหน่งโปรแกรมแอดเดรสที่ระบุของการเกิดอินเตอร์รัพท์ เมื่อเสร็จสิ้นแล้วจึงจะย้อนกลับมาทำงานที่พักไว้ การทำงานเปรียบได้เหมือนเรากำลังเขียนหนังสืออยู่ ในขณะนั้นมีเสียงโทรศัพท์ดังขึ้น เราจะพักงานเขียนไว้แล้วมารับโทรศัพท์ เมื่อพูดโทรศัพท์เสร็จวางหูโทรศัพท์ เราก็มาทำงานเขียนหนังสือต่อจากที่ค้างไว้ต่อไป การอินเตอร์รัพท์ก็เช่นกัน การเกิดอินเตอร์รัพท์ของไมโครคอนโทรลเลอร์ เบอร์ PIC16F877 มาจากทั้งแหล่งภายในและภายนอกรวม 14 แหล่งด้วยกัน โดยมีรีจิสเตอร์ชื่อ INTCON ทำหน้าที่ควบคุมการทำงานของฟังก์ชั่นการทำงานของอินเตอร์รัพท์ทั้งหมด และยังมีรีจิสเตอร์ช่วยอีกหลายตัวที่ทำหน้าที่ร่วมกับรีจิสเตอร์หลัก INTCON

INTCON REGISTER

| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|
| GIE | PEIE | T0IE | INTE | RBIE | T0IF | INTF | RBIF |

Bit 7 GIE เป็นบิทที่มีนัยสำคัญสูงสุด ที่ควบคุมการเกิดอินเตอร์รัพท์ทั้งหมด

1 = อนุญาตให้เกิดอินเตอร์รัพท์ทุกประเภท

0 = ไม่อนุญาตทั้งหมด

Bit 6 PEIE เป็นบิทควบคุมการเกิดอินเตอร์รัพท์ที่มาจากอุปกรณ์รอบข้างซีพียู (Peripheral interrupt)

1 = อนุญาตให้มี

0 = ไม่อนุญาต

Bit 5 T0IE เป็นบิทควบคุมการเกิดอินเตอร์รัพท์ที่มาจากการเกิด Overflow ของ TMR0 หรือ Timer0

1 = อนุญาตให้มี

0 = ไม่อนุญาต

Bit 4 INTE เป็นบิทควบคุมการเกิดอินเตอร์รัพท์ที่มาจากภายนอกซีพียูที่ขา RB0/INT

1 = อนุญาตให้มี

0 = ไม่อนุญาต

Bit 3 RBIE เป็นบิทควบคุมการเกิดอินเตอร์รัพท์ที่มาจากภายนอกซีพียูที่ขา RB4 - RB7

1 = อนุญาตให้มี

0 = ไม่อนุญาต

Bit 2 T0IF เป็นบิทแสดงสถานะการเกิด Overflow ของ TMR0

1 = เกิด Overflow ที่ TMR0

0 = ไม่เกิด

Bit 1 INTF เป็นบิทแสดงสถานะการเกิดอินเตอร์รัพท์จากภายนอกที่ขา RB0/INT

1 = เกิด เกิดอินเตอร์รัพท์จากภายนอกที่ขา RB0/INT

0 = ไม่เกิด

Bit 0 RBIF เป็นบิทแสดงสถานะการเกิดอินเตอร์รัพท์จากการเปลี่ยนแปลงที่ขา RB4 - RB7

1 = เกิด เกิดอินเตอร์รัพท์จากภายนอกที่ขา RB4 - RB7

0 = ไม่เกิด

**หมายเหตุ** เมื่อเกิดอินเตอร์รัพท์ขึ้น ทั้งการเกิด Overflow ของ TMR0 หรือ จากสัญญาณที่ขา RB0/INT หรือ จากการเปลี่ยนแปลงลอจิกที่ขา RB4 – RB7 ก็ตามเมื่อเสร็จการทำงานใน Interrupt service routine แล้ว ต้องรีบเคลียร์แฟล็ก ในบิทที่เกี่ยวข้องด้วยได้แก่ ที่บิท 2 บิท 1 หรือบิท 0

นอกจากนี้ยังมีรีจิสเตอร์ที่ควบคุม และแสดงสถานะ การเกิดอินเตอร์รัพท์ที่อุปกรณ์ Peripherals คือ PIE1 PIE2 PIR1 และ PIR2 รายละเอียดมีดังนี้ คือ

PIE1 REGISTER

| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|
| PSPIE | ADIE | RCIE | TXIE | SSPIE | CCP1IE | TMR2IE | TMR1IE |

Bit 7 PSPIE เป็นบิทควบคุมการเกิดอินเตอร์รัพท์จากการอ่าน เขียนของ Parallel Slave Port

1 = อนุญาตให้เกิดอินเตอร์รัพท์

0 = ไม่อนุญาต

Bit 6 ADIE เป็นบิทควบคุมการเกิดอินเตอร์รัพท์จากการแปลง A/D ในโมดูล Analog to Digital

1 = อนุญาตให้มี

0 = ไม่อนุญาต

Bit 5 RCIE เป็นบิทควบคุมการเกิดอินเตอร์รัพท์จากการรับสัญญาณการสื่อสารอนุกรม USART

1 = อนุญาตให้มี

0 = ไม่อนุญาต

Bit 4 TXIE เป็นบิทควบคุมการเกิดอินเตอร์รัพท์จากการส่งสัญญาณการสื่อสารอนุกรม USART

1 = อนุญาตให้มี

0 = ไม่อนุญาต

Bit 3 SSPIE เป็นบิทควบคุมการเกิดอินเตอร์รัพท์จากการสื่อสารอนุกรมแบบซิงโครนัส (SSP)

1 = อนุญาตให้มี

0 = ไม่อนุญาต

Bit 2 CCP1IE เป็นบิทควบคุมการเกิดอินเตอร์รัพท์จากการใช้งานโมดูล CCP1

1 = อนุญาตให้มี

0 = ไม่อนุญาต

Bit 1 TMR2IE เป็นบิทควบคุมการเกิดอินเตอร์รัพท์จากค่าของ TMR2 เท่ากันกับ PR2

1 = อนุญาตให้มี

0 = ไม่อนุญาต

Bit 0 TMR1IE เป็นบิทควบคุมการเกิดอินเตอร์รัพท์จากการเกิด Overflow ของ TMR1

1 = อนุญาตให้มี

0 = ไม่อนุญาต

PIR1 REGISTER เป็นรีจิสเตอร์แสดงสถานะ การเกิดอินเตอร์รัพท์ ของ Peripheral devices

| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|
| PSPIF | ADIF | RCIF | TXIF | SSPIF | CCP1IF | TMR2IF | TMR1IF |

Bit 7 PSPIF เป็นบิทแสดงสถานะ การเกิดอินเตอร์รัพท์จากการอ่าน เขียนของ Parallel Slave Port

1 = เกิดอินเตอร์รัพท์

0 = ไม่เกิด

Bit 6 ADIF เป็นบิทแสดงสถานะ การเกิดอินเตอร์รัพท์จากการแปลง A/D ในโมดูล Analog to Digital

1 = เกิดอินเตอร์รัพท์

0 = ไม่เกิด

Bit 5 RCIF เป็นบิทแสดงสถานะ เกิดอินเตอร์รัพท์จากการรับสัญญาณการสื่อสารอนุกรม USART

1 = เกิดอินเตอร์รัพท์

0 = ไม่เกิด

Bit 4 TXIE เป็นบิทแสดงสถานะ การเกิดอินเตอร์รัพท์จากการส่งสัญญาณการสื่อสารอนุกรม USART

1 = เกิดอินเตอร์รัพท์

0 = ไม่เกิด

Bit 3 SSPIE เป็นบิทแสดงสถานะ การเกิดอินเตอร์รัพท์จากการสื่อสารอนุกรมแบบซิงโครนัส (SSP)

1 = เกิดอินเตอร์รัพท์

0 = ไม่เกิด

Bit 2 CCP1IE เป็นบิทแสดงสถานะ การเกิดอินเตอร์รัพท์จากการใช้งานโมดูล CCP1

1 = อนุญาตให้มี

0 = ไม่อนุญาต

Bit 1 TMR2IE เป็นบิทแสดงสถานะ การเกิดอินเตอร์รัพท์จากค่าของ TMR2 เท่ากันกับ PR2

1 = เกิดอินเตอร์รัพท์

0 = ไม่เกิด

Bit 0 TMR1IE เป็นบิทแสดงสถานะ การเกิดอินเตอร์รัพท์จากการเกิด Overflow ของ TMR1

1 = เกิดอินเตอร์รัพท์

0 = ไม่เกิด

**<u>หมายเหตุ</u>** ค่าบิทสถานะในรีจิสเตอร์นี้ หากเกิดอินเตอร์รัพท์ขึ้นแล้ว หลังจากเสร็จการบริการต้องเคลียร์ทันที

PIE2 REGISTER

| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|
| - | Reserved | - | EEIE | BCLIE | - | - | CCP2IE |

Bit 7 ไม่ใช้งาน

Bit 6 Reserved สงวนไว้

Bit 5 ไม่ใช้งาน

Bit 4 EEIE บิทควบคุมการอินเตอร์รัพท์จากกระบวนการเขียนข้อมูลลง EEPROM ในตัวชิพ

1 = อนุญาตให้มี

0 = ไม่อนุญาต

Bit 3 BCLIE เป็นบิทควบคุมการเกิดอินเตอร์รัพท์จากการชนกันของข้อมูลบนบัส

1 = อนุญาตให้มี

0 = ไม่อนุญาต

Bit 2 ไม่ใช้งาน

Bit 1 ไม่ใช้งาน

Bit 0 CCP2IE เป็นบิทควบคุมการเกิดอินเตอร์รัพท์จากการใช้งานโมดูล CCP2

1 = อนุญาตให้มี

0 = ไม่อนุญาต

PIR2 REGISTER

| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|
| - | Reserved | - | EEIE | BCLIE | - | - | CCP2IE |

Bit 7 ไม่ใช้งาน

Bit 6 Reserved สงวนไว้

Bit 5 ไม่ใช้งาน

Bit 4 EEIE บิทควบแสดงสถานะ อินเตอร์รัพท์จากกระบวนการเขียนข้อมูลลง EEPROM ในตัวชิพ

1 = เมื่อกระบวนการเขียนข้อมูลเสร็จสิ้น

0 = ยังไม่เสร็จสิ้น

Bit 3 BCLIE เป็นบิทแสดงสถานะ การเกิดอินเตอร์รัพท์จากการชนกันของข้อมูลบนบัส

1 = มีการชนเกิดขึ้น

0 = ไม่มี

Bit 2 ไม่ใช้งาน

Bit 1 ไม่ใช้งาน

Bit 0 CCP2IE เป็นบิทแสดงสถานะ การเกิดอินเตอร์รัพท์จากการใช้งานโมดูล CCP2

1 = มีการใช้โมดูลนี้(ยกเว้นการใช้ในโหมด PWM)

0 = ไม่มี

การใช้งานอินเตอร์รัพท์ในภาษา Pic Basic Pro Compiler มีคำสั่งที่จัดการเกี่ยวกับการใช้งานอินเตอร์รัพท์ดังนี้ คือ

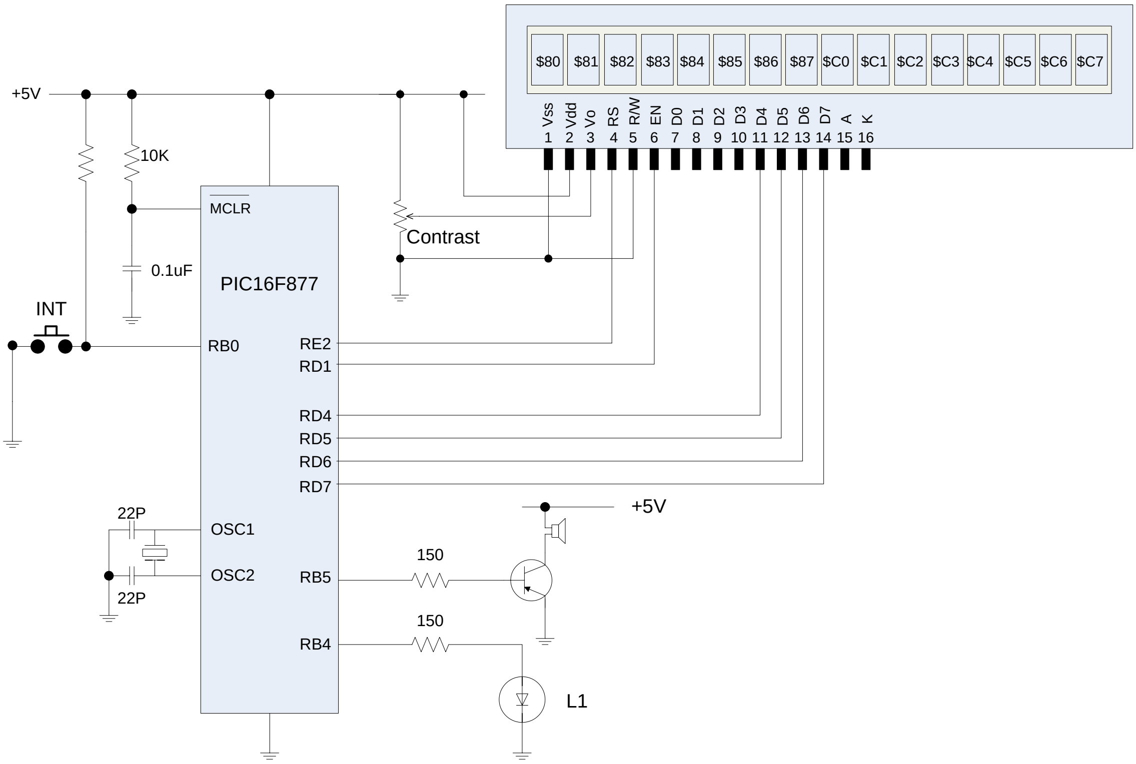
ON INTERRUPT GOTO *Label* เป็นคำสั่งที่ให้มีการจัดการอินเตอร์รัพท์ คำสั่งนี้จะต้องกำหนดไว้ก่อนที่หัวของโปรแกรมหลัก (Main Program) หรือ ก่อนที่จะอนุญาตให้ใช้งานอินเตอร์รัพท์ และสิ่งที่ต้องกำหนดตามมา คือ รีจิสเตอร์ INTCON ว่าจะเปิดใช้อินเตอร์รัพท์ตัวใดบ้าง ตัวอย่างเช่น ต้องการเปิดรับอินเตอร์รัพท์จากขา RB0/INT กำหนดดังนี้

INTCON = %10010000

DISABLE เป็นคำสั่งที่ระงับการเพิ่มการเกิดอินเตอร์รัพท์ซ้อนกัน คำสั่งนี้จะวางไว้ก่อนบรรทัด Label ของโปรแกรมอินเตอร์รัพท์

RESUME เป็นคำสั่งปิดท้ายของโปรแกรมอินเตอร์รัพท์เพื่อให้ย้อนกลับไปทำงานเดิม ต่อจากที่พักไว้ที่โปรแกรมหลัก แต่ก่อนหน้าที่จะใช้คำสั่ง RESUME จะต้องเคลียร์แฟล็กที่เกิดขึ้นในรีจิสเตอร์ INTCON หรือในรีจิสเตอร์ที่เกี่ยวข้องก่อน เช่น INTCON.1 = 0

วงจรทดลองตาม Experiment 31




รูปที่ 1 วงจรทดลองการเกิดอินเตอร์รัพท์จากขา RB0/INT

การทำงานของโปรแกรม

เริ่มทำงานจะไม่มีอะไรเกิดขึ้น เนื่องจากที่ Main program จะมีแค่คำสั่งให้วนลูปอยู่เท่านั้น แต่พอมีการเปลี่ยนแปลงค่าลอจิกที่ขา RB0/INT โดยการกดสวิทช์ INT จะทำให้เกิดอินเตอร์รัพท์ขึ้น โปรแกรมจะวิ่งเข้าสู่โปรแกรมบริการอินเตอร์รัพท์โดยจะสั่งให้แสดงข้อความที่จอ LCD และมีเสียงบีป 3 ครั้งจากนั้นจะลบข้อความที่จอ LCD เคลียร์แฟล็กที่เกิดอินเตอร์รัพท์ แล้วย้อยกลับมาทำงานวนลูปใน Main program อีก

โปรแกรมคำสั่ง

```
spk     var  portb.5
L1      var  portb.4
i        var  byte
on interrupt  goto int
intcon = %10010000
'-------------Main  program-------------
'
       low    L1
start:
       goto start
       end
'
'--------- End  of  Main program --------
'
'
'--------- Interrupt  routine  ---------------
'
       disable
int:
       high   L1
       lcdout $fe,1,"Fault..!"
       for i = 1 to 3
          freqout spk,20,2000
          pause 500
       next i
       pause  2000
       low    L1
       lcdout $fe,1,"         "
       intcon.1 = 0               ' Clear  flag
       resume                     ' Return  to  main program
'
'----------End  of  Interrupt  routine ------
```

**ข้อควรจำ**

1. ในการเขียนโปรแกรมหากมีการเปิดใช้อินเตอร์รัพท์ จะต้องคำนึงถึงค่า Latency คือค่าช่วงเวลาที่มีการเกิดอินเตอร์รัพท์ขึ้นจริง ไปจนถึงเวลาที่เข้าสู่โปรแกรมบริการอินเตอร์รัพท์ ซึ่งเกิดจากการทำงานของ Timer หรือคำสั่งหน่วงเวลาที่ยังไม่สิ้นสุดเหล่านี้ ช่วงเวลาดังกล่าวนี้ หากเกิดอินเตอร์รัพท์ขึ้นจะไม่สามารถใช้คำสั่งอื่นได้เลย นอกจากรอให้เสร็จ หากเลี่ยงไม่ได้ที่จะต้องใช้คำสั่งหน่วงเวลานาน ๆ ควรจะต้องใช้คำสั่ง PAUSE ช่วงเวลาสั้น ๆ แต่ทำซ้ำ ๆ ด้วยคำสั่ง FOR … NEXT เช่น

ต้องการหน่วงเวลา 1 วินาที PAUSE 1000

สามารถใช้คำสั่งทดแทนได้ดังนี้

```
I        var      byte
FOR  I = 1  TO   100
     PAUSE 10
NEXT I
```

2. ในการวางคำสั่งสำหรับเคลียร์แฟล็กในโปรแกรมส่วนบริการอินเตอร์รัพท์ (INTCON.1 = 0) หากเราวางไว้บรรทัดแรก ๆ จะมีผลทำให้โปรแกรมสามารถเปิดรับการเกิดอินเตอร์รัพท์ขึ้นมาได้ในขณะที่กำลังทำงานอยู่ในส่วนอินเตอร์รัพท์ เรียกว่า ยอมให้เกิดอินเตอร์รัพท์ซ้อนขึ้นมาได้ ขณะที่ของเก่ายังไม่เสร็จ แต่ถ้าเราวางคำสั่งดังกล่าวไว้ท้ายสุดก่อน RESUME ผลการทำงานที่ได้คือ โปรแกรมจะทำงานในส่วนอินเตอร์รัพท์ที่เกิดขึ้นครั้งแรกเสร็จเสียก่อน จึงจะยอมเปิดรับอินเตอร์รัพท์ใหม่ได้

3. เนื่องจากซีพียู ไม่สามารถรับบริการอินเตอร์รัพท์ได้หลายจุดพร้อม ๆ กันได้ หากเปิดอินเตอร์รัพท์ไว้หลายแหล่ง หากเกิดอินเตอร์รัพท์ขึ้น เราจะต้องเขียนโปรแกรมเพื่อตรวจสถานะ แฟล็กในรีจิสเตอร์ ของ อินเตอร์รัพท์ที่เปิดไว้ทุกตัว จึงจะรู้ว่าเกิดอินเตอร์รัพท์จากแหล่งใด

**ประโยชน์ และการนำอินเตอร์รัพท์ไปใช้งาน**

1. ใช้กรณีเป็นปุ่มควบคุมฉุกเฉิน โดยไม่จำเป็นต้องให้โปรแกรมมาสแกนวนตรวจสอบสถานะ การกดสวิทช์ของปุ่มฉุกเฉิน ทำให้การทำงานของโปรแกรมมีประสิทธิภาพด้านความเร็วเพิ่มขึ้น

2. ใช้เป็นปุ่มปลุกซีพียูให้ตื่นจากการทำงานใน Sleep Mode เพื่อประหยัดพลังงาน

3. ใช้ตัดภาระที่โปรแกรมจะต้องทำงานวนตรวจสอบสถานนะด้านอินพุทตลอดเวลาโดยไม่จำเป็น ซึ่งจะทำให้โปรแกรมประมวลผลช้าลงและไม่ทันเวลาในการตรวจจับสัญญาณอินพุท หากการออกแบบโดยการนำเอาอินเตอร์รัพท์เข้ามาใช้ในการรับค่าด้านอินพุทที่เหมาะสมแล้ว จะทำให้ไม่ต้องเขียนโปรแกรม วนตรวจสอบอินพุทเลย โดยที่โปรแกรมจะรับรู้และทำงานก็ต่อเมื่อมีสัญญาณเข้ามาทางอินพุททางใดทางหนึ่งเท่านั้น