บทที่ 8

ใบงานทดลองเกี่ยวกับการเขียนโปรแกรมติดต่อกับอุปกรณ์ I/O ที่เป็นองค์ประกอบการประยุกต์ใช้งานขั้นพื้นฐาน และการประมวลผลทางคณิตศาสตร์ การจัดการรีจิสเตอร์ ตัวแปร และสัญญาณเบื้องต้นเบื้องต้น การเขียนโปรแกรมการสแกนรับสัญญาณอินพุทที่มีจำนวนมากกว่าขา I/O การแสดงผลทาง LED 7-Segment การแสดงผลทางจอ LCD การสื่อสารข้อมูลทางพอร์ทอนุกรมแบบอะซิงโครนัส และซิงโครนัส การสื่อสารระหว่างตัวอุปกรณ์ภายนอกแบบ I2C อุปกรณ์สายเดียว (One Wire) การประยุกต์ Timer / Counter การประยุกต์ Interrupt โครงงานนาฬิกาตั้งเวลาแบบ Real Time Clock โครงงานระบบวัดและควบคุมอุณหภูมิอัตโนมัติ การประยุกต์ขับอุปกรณ์กำลังมอเตอร์ดีซี สเต็ปปิ้งมอเตอร์ โครงงานการควบคุมเซอร์โวมอเตอร์ โครงงานระบบควบคุมไฟสัญญาณรถยนต์

Experiment 1 การเขียนโปรแกรมติดต่อกับ Input - Output ขั้นพื้นฐาน

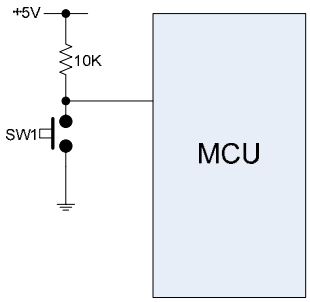
จุดประสงค์ เพื่อศึกษาการเขียนโปรแกรมให้ไมโครคอนโทรลเลอร์รับการกดสวิตช์ และส่งเอ้าพุท ออกทางขา I/O เพื่อเป็นพื้นฐานการประยุกต์ใช้งานต่อไป

ทฤษฎีพื้นฐาน

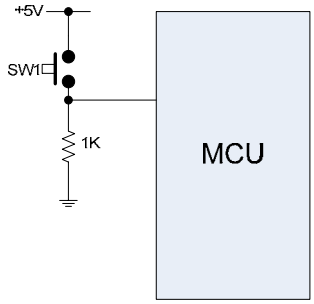
การทำงานของระบบไมโครคอนโทรลเลอร์โดยทั่วไปประกอบด้วย 3 ส่วน คือ ส่วนอินพุท ส่วนประมวลผลหรือส่วนควบคุม และส่วนเอ้าพุท

ส่วนอินพุทประกอบด้วยอุปกรณ์ที่เป็นวงจรทำหน้าที่ป้อนสัญญาณให้แก่ส่วนประมวลผล สัญญาณที่ป้อนมี 2 แบบคือ

- แบบดิจิตอล (Digital Signal) มีระดับแรงเคลื่อนแค่สองสถานะคือระดับที่เป็นลอจิก 1 คือ 5 โวลท์ และระดับลอจิก 0 คือ น้อยกว่า 1.5 โวลท์ หรือ 0 โวลท์ ตัวอย่างวงจรได้แก่ สวิทช์ปิด – เปิดแบบต่าง ๆ



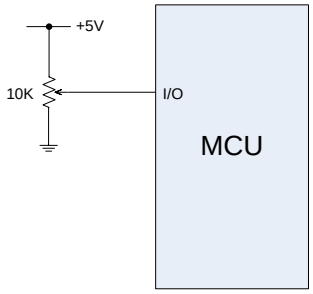
รูปที่ 1 การต่อสวิทช์กับ MCU แบบลอจิกลบ



รูปที่ 2 การต่อสวิทช์กับ MCU แบบลอจิกบวก

ในทางการนำไปใช้งานจริงนิยมต่อวงจรตามรูปที่ 1 เพื่อป้องกันสัญญาณรบกวนขณะที่ยังไม่กดสวิทช์ เนื่องจากตามรูปที่ 2 ขณะที่ไม่มีการกดสวิทช์ ขา I/Oของ MCU จะอยู่ในสภาพลอยอยู่ เนื่องจากไม่ได้ต่อโดยตรงกับ Ground

- แบบอนาล็อก (Analog Signal) มีระดับแรงเคลื่อนตั้งแต่ 0 – 5 โวลท์โดยสามารถรับได้ถึง 2 ระดับ หรือ 1064 ระดับ ตัวอย่างอุปกรณ์ได้แก่ตัวตรวจจับและส่งสัญญาณระดับอุณหภูมิ ความดัน ระดับของเหลว หรือ ตำแหน่งหมุนของ Potentiometer เป็นต้น ระดับทั้งหมดที่กล่าวนี้จะต้องถูกปรับเปลี่ยนให้เป็นระดับของแรงเคลื่อน 0-5 โวลท์ก่อนที่จะส่งเข้าขา I/O ของไมโครคอนโทรลเลอร์

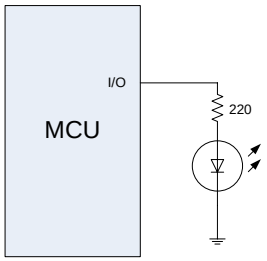


รูปที่ 3 การต่อวงจรรับสัญญาณอนาล็อกจาก Potentiometer ป้อนเข้าขา I/O ของ ไมโครคอนโทรลเลอร์

การต่อต่อสัญญาณอนาล็อกเข้ากับขา I/O ต้องต่อให้ตรงกับพ็อร์ทที่สามารถรับสัญญาณอนาล็อกได้ซึ่ง MCU

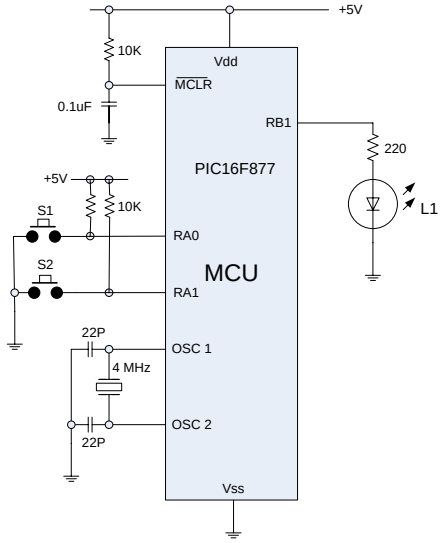
แต่ละยี่ห้อ และแต่ละรุ่นจะไม่เหมือนกัน ให้ดูใน Datasheet (รายละเอียด ให้ศึกษาในใบงานที่เกี่ยวข้องต่อไป)

ส่วนเอ้าพุทเป็นส่วนที่รับสัญญาณจาก MCU มาทำงาน โดยทั่วไปจะเป็นสัญญาณแบบดิจิตอล สิ่งที่ต้องคำนึงถึงในการออกแบบคือ กระแสที่ MCU ส่งออกมาขับอุปกรณ์เอ้าพุทของแต่ละขา I/O แต่ละยี่ห้อ และแต่ละรุ่นจะไม่เหมือนกัน สำหรับ MCU ของไมโครชิพตระกูล PIC I/O แต่ละขาสามารถขับกระแสได้เต็มที่ 25 mA ในการออกแบบจะต้องเผื่อความปลอดภัยไว้จะต้องไม่เกิน 10 mA



รูปที่ 4 การต่อวงจรแสดงผลด้วยหลอด LED

วงจรทดลองตาม Experiment 1



รูปที่ 5 วงจรทดลองตาม Experiment 1

การทำงานของโปรแกรม

เมื่อกดสวิตช์ S1 ทำให้หลอด L1 ติด กด S2 ทำให้หลอด L1 ดับ

โปรแกรมคำสั่ง

```
S1      VAR     PORTA.0
S2      VAR     PORTA.1
L1      VAR     PORTB.1
        TRISA     =   %111111
        TRISB     =   %00000000
        ADCON1 = 7
        PORTB   = 0
LOOP:   IF  (S1 =  0) AND (S2 = 1)  THEN   HIGH  L1
        IF  (S1 =  1) AND (S2 = 0)  THEN   LOW  L1
        PAUSE  50
        GOTO    LOOP
        END
```

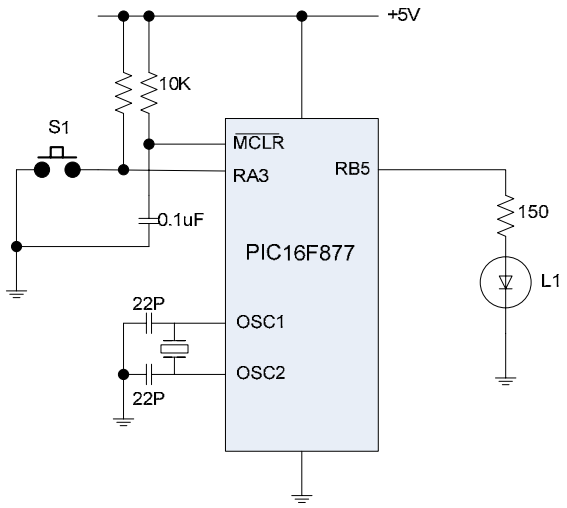
Experiment 2 การเขียนโปรแกรมให้เอ้าพุททำงานแบบ Toggle

จุดประสงค์ เพื่อศึกษาการเขียนโปรแกรมเปลี่ยน Tac Switch ให้ทำงานเป็น Toggle Switch เพื่อนำไปประยุกต์ใช้ทำสวิตช์ควบคุมการเปิด – ปิด อุปกรณ์ ไฟฟ้าเป็นต้น

ทฤษฎีพื้นฐาน

เนื่องจากอุปกรณ์สวิทช์ที่ใช้เชื่อมต่อกับไมโครคอนโทรลเลอร์ส่วนมากจะออกแบบให้ทำงานในลักษณะเป็นสวิทช์ปุ่มกด แบบกดติด-ปล่อยดับ ดังนั้นหากเราต้องการให้ได้ผลเอ้าพุทเป็นแบบ Toggle หรือกดครั้งแรกติด และกดครั้งที่สอง ดับ สลับกันไป เราจะต้องเขียนโปรแกรมบังคับเอาเอง

วงจรทดลองตาม Experiment 2



การทำงานของโปรแกรม

เมื่อกด S1 ทำให้หลอด L1 ติดเมื่อกด S1 อีกครั้ง หลอด L1 ดับ และเมื่อกด S1 อีกครั้ง หลอด L1 จะติดเป็นแบบนี้สลับกัน เรียกว่า “Toggle” โดยใช้คำสั่ง TOGGLE

โปรแกรมคำสั่ง

```
S1      VAR    PORTA.3
L1      VAR     PORTB.5
        TRISA   = %111111
        TRISB    = %00000000
        ADCON1 = 7
        PORTB  = 0
start:
        IF S1 = 1 Then goto start
        TOGGLE L1
        Pause   100
loop:
        IF S1 = 0 Then goto loop
        Pause   100
        GoTo    start
        End
```

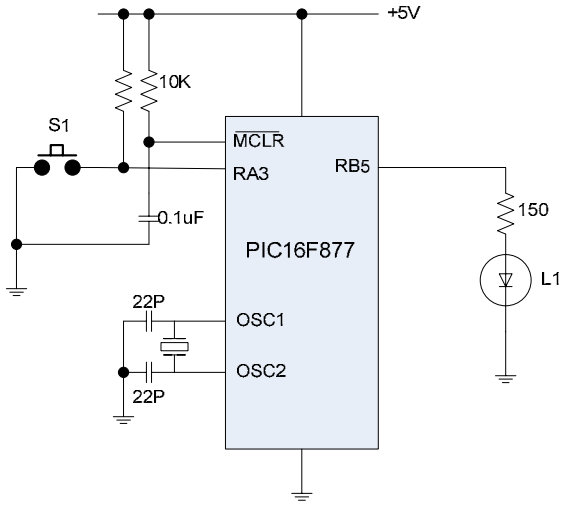
Experiment 3 การเขียนโปรแกรมเพื่อให้เอ้าพุททำงานแบบ Jogging

จุดประสงค์ เพื่อศึกษาการเขียนโปรแกรมให้ไมโครคอนโทรลเลอร์ส่งค่าเอ้าพุทเมื่อกดสวิทช์ และหยุดส่งเมื่อไม่กดปุ่มสวิทช์ เพื่อเป็นพื้นฐานในการประยุกต์ใช้งานต่อไป

ทฤษฎีพื้นฐาน

การทำงานแบบ Jogging คือการเขียนโปรแกรมให้ไมโครคอนโทรลเลอร์ส่งค่าเอ้าพุทออกมาตลอดเวลาขณะที่ยังกดสวิทช์อินพุทค้างอยู่ และหยุดส่งเมื่อเราปล่อยสวิทช์ ตัวอย่างงานที่ใช้การควบคุมแบบ Jogging ได้แก่ การควบเครน ปั้นจั่น และการบังคับเป็นจังหวะ เช่นการบังคับการเคลื่อนที่ของหุ่นยนต์แบบ Manual เป็นต้น หลักการเขียนโปรแกรมจะต้องใช้คำสั่งวน คอยตรวจสอบสถานการณ์กดสวิทช์ โดยมีเงื่อนไขว่า หากสวิทช์ยังถูกกดอยู่ต้องส่งค่าเอ้าพุทออกไป หากปล่อยสวิทช์จะหยุดส่งเอ้าพุทออก

วงจรทดลองตาม Experiment 3



โปรแกรมคำสั่ง ที่ 1 การทำงานในระดับบิท

```
S1      VAR    PORTA.3
L1      VAR    PORTB.5
        TRISA    = %111111
        TRISB    = %00000000
         ADCON1 = 7
         PORTB   = 0
start:
        if S1 = 1 Then
                HIGH   L1
        else
                LOW    L1
        endif
        pause  50
        GoTo   start
        End
```

โปรแกรมคำสั่ง ที่ 2 การทำงานในระดับพอร์ท

สำหรับใช้กับวงจรที่มีสวิทช์อินพุทต่ออยู่หลายตัว โดยไม่ต้องเขียนโปรแกรมให้ใช้คำสั่ง IF.. THEN คอยตรวจสอบเงื่อนไขการกดสวิทช์ทุกตัว ซึ่งจะทำให้โปรแกรมทำงานช้าลงโดยไม่จำเป็น และจะทำให้โปรแกรมสั้นลงมาก

```
S1      VAR     PORTA.3
L1      VAR     PORTB.5
        TRISA    =  %111111
        TRISB    =  %00000000
        ADCON1 = 7
        PORTB   = 0
start:
        portb  =  porta ^ %000000
        pause    50
        GoTo     start
        End
```

การทำงานของโปรแกรม

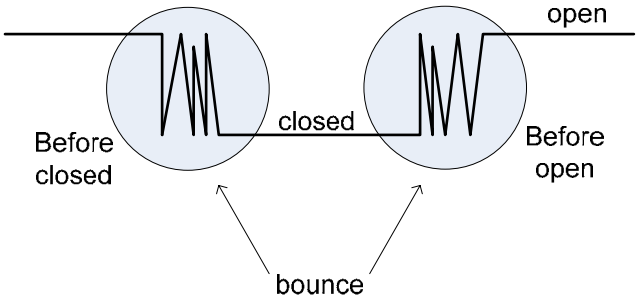
Portb = porta ^ %000000 หมายความว่า ให้เอาค่าลอจิกที่พอร์ท A ทำ Exclusive – OR กับเลขไบนารี %000000 กำหนดเป็นค่าพอร์ท B ซึ่งเป็นเอ้าพุท

Experiment 4 การเขียนโปรแกรมเพื่อแก้ไขการเด้งของหน้าสัมผัสสวิทช์

จุดประสงค์ เพื่อศึกษาการเขียนโปรแกรมในการแก้ไขการเด้งของหน้าสัมผัสสวิทช์(De - bounce) หรือ เพื่อให้ไมโครคอนโทรลเลอร์รับรู้การกด และการถอนนิ้วจากปุ่มสวิทช์

ทฤษฎีพื้นฐาน

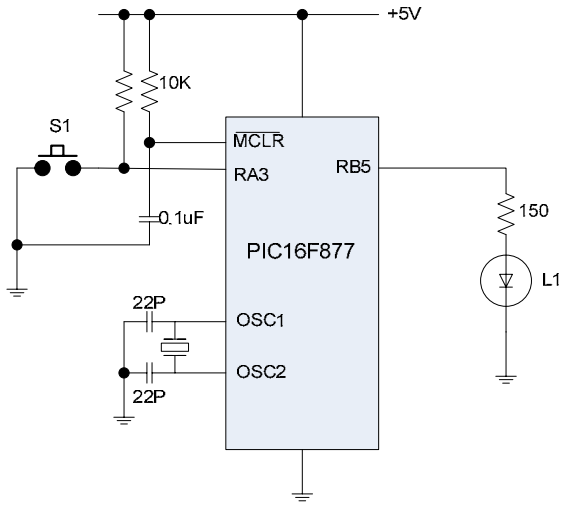
การใช้สวิทช์เป็นอุปกรณ์ป้อนสัญญาณลอจิกให้กับไมโครคอนโทรลเลอร์จะต้องระวังเรื่องการเด้งของหน้าสัมผัส ซึ่งจะเกิดขึ้นระหว่างหน้าสัมผัสกำลังจะติด และหน้าสัมผัสกำลังจะจากกันโดยจะเกิดรูปพัลส์หัวแตกขึ้นหลายลูกในช่วงเวลา 1 - 10 mS รูปพัลส์หัวแตกดังกล่าวจะทำให้เกิดการทำงานของโปรแกรมผิดพลาดขึ้นได้



รูปที่ 1 แสดงการเกิดการเด้งของหน้าสัมผัสสวิทช์

ในการเขียนโปรแกรมเพื่อแก้ไขการเด้ง จะต้องใช้คำสั่งหน่วงเวลาช่วงสั้น ๆ ก่อนที่จะไปตรวจสอบค่าลอจิกการกดสวิทช์ โดยทั่วไปจะอยู่ระหว่าง 10 – 50 mS เพื่อให้การรับรู้สัญญาณลอจิกเลยตำแหน่งการเด้งของหน้าสัมผัสสวิทช์ไปก่อน

วงจรทดลองตาม Experiment 4



การทำงานของโปรแกรม

จากวงจร ขณะยังไม่กดสวิทช์ S1 หลอด L1 ดับ เมื่อกดสวิทช์ S1 ค้าง หลอดยังคงดับ เมื่อปล่อย S1 หลอดจึงจะติด และเมื่อกด S1 ค้าง หลอด L1 ยังคงติด จนกว่าจะปล่อย S1 หลอดจึงดับ ซึ่งเป็นการทำงานแบบ Toggle แต่เอ้าพุทจะทำงานก็ต่อเมื่อเราได้ปล่อย S1 ซึ่งเป็นวิธีการที่ถูกต้องในการนำไปประยุกต์ใช้งาน

โปรแกรมคำสั่ง

```
S1        VAR       PORTA.3
L1        VAR         PORTB.5
  TRISA    =  %11111111
  TRISB    =  %00000000
  ADCON1 = 7
  PORTB   = 0
start:
    if s1 = 1 then goto start
    pause 50
loop:
    if s1 = 0 then goto loop
    toggle l1
    pause 50
    goto start
    end
```

Experiment 5 การเขียนโปรแกรมเพื่อการทำงานซ้ำตามจำนวนครั้งที่กำหนด

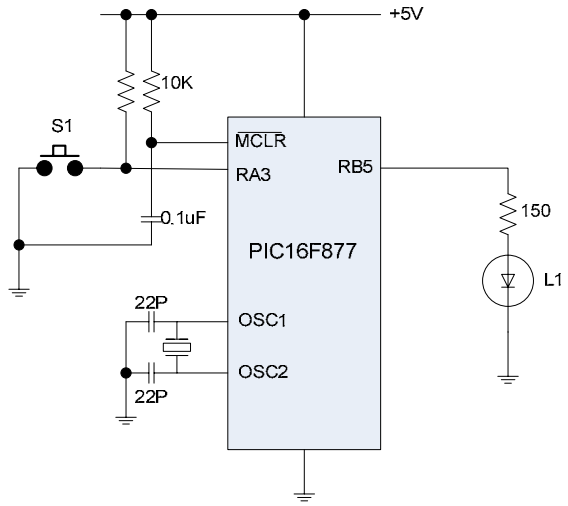
จุดประสงค์ เพื่อศึกษาการเขียนโปรแกรมในการ ประมวลผลการทำงานซ้ำตามจำนวนครั้งที่กำหนด

ทฤษฎีพื้นฐาน

ในการเขียนโปรแกรมเพื่อกำหนดให้ประมวลผลการทำงานซ้ำ จะมีอยู่ 3 ลักษณะคือ

- ใช้คำสั่งกระโดดย้อนกลับไปเริ่มต้นทำงานใหม่โดยไม่มีเงื่อนไขกำหนดด้วยคำสั่ง GOTO คำสั่งนี้ จะใช้ในกรณีย้อนกลับไปเริ่มต้นทำงานที่จุดเริ่มต้นใหม่ ไม่รู้จบ
- ใช้คำสั่งในการประมวลผลการทำงานตามจำนวนครั้งที่กำหนด ด้วยคำสั่ง FOR … NEXT
- ใช้คำสั่งในการประมวลผลการทำงานซ้ำจนกว่าเงื่อนไขที่กำหนดจะเป็นจริง หรือ เท็จ ด้วยคำสั่ง WHILE … WEND

วงจรทดลองตาม Experiment 5



โปรแกรมคำสั่ง

```
S1      VAR     PORTA.3
L1      VAR     PORTB.5
i       VAR     byte
        TRISA   = %11111111
        TRISB   = %00000000
        ADCON1 = 7
         PORTB  = 0
start:  if s1 = 1 then goto start
      pause 50
loop:   if s1 = 0 then goto loop
      for i = 1 to 5
         high  l1
         pause 500
         low   l1
         pause 500
      next i
      goto start
      end
```

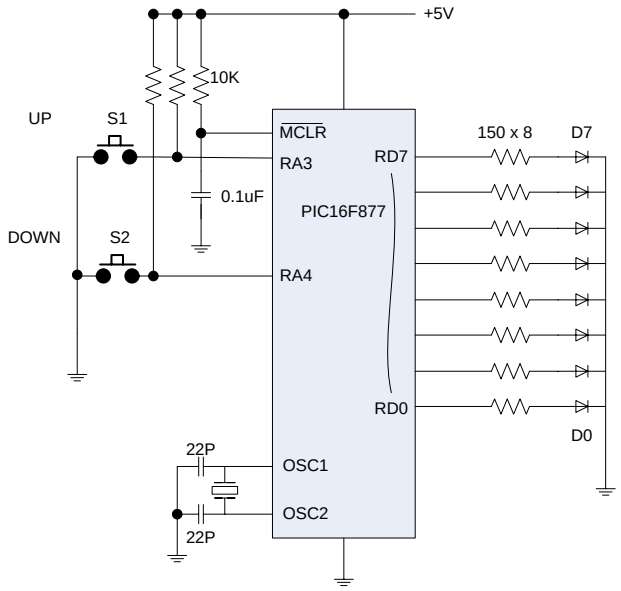
Experiment 6 การเขียนโปรแกรมเพื่อควบคุมการเพิ่ม – ลดค่าตัวแปร

จุดประสงค์ เพื่อศึกษาการเขียนโปรแกรมในการควบคุมการเพิ่ม-ลดค่าตัวแปรด้วยวิธีการต่าง ๆ

ทฤษฎีพื้นฐาน

การควบคุมการเพิ่ม – ลดค่าตัวแปรจากอุปกรณ์สวิทช์ที่ขา I/O เป็นพื้นฐานสำคัญในการนำไปประยุกต์เขียนโปรแกรมเพื่อการประมวลผลกการควบคุมค่าที่มีระดับมาก-น้อย ตัวอย่างเช่น การเพิ่ม-ลดความเร็วมอเตอร์ การเพิ่ม-ลดอุณภูมิเป็นต้น ในการออกแบบ อุปกรณ์ทางด้านอินพุทอย่างน้อยต้องประกอบด้วยสวิทช์ 2 ตัว สำหรับการกดเพิ่มค่า และลดค่า การเขียนโปรแกรมในการประมวลผลจะมี 2 ลักษณะคือ การควบคุมให้เพิ่ม-ลดค่าต่อเนื่อง เมื่อกดสวิทช์ค้างไว้ หรือเรียกว่า continuous operation และการควบคุมให้ทำงานค้างทีละสเต็ป หรือเรียกว่า step operation

วงจรทดลองตาม Experiment 6



โปรแกรมคำสั่งที่ 1 Continuous operation

```
S1      VAR     PORTA.3
S2      VAR     PORTA.4
k  var byte
     TRISA    =   %111111
     TRISD        =   %00000000
     ADCON1 = 7
     portd = 0
LOOP:   IF  (S1 =  0) AND (S2 = 1)  THEN
                pause 50
                k  =  k+1
                portd = k
                if k = 255 then k = 254
          endif
```

มีต่อหน้าถัดไป

```
      IF  (S1 =  1) AND (S2 = 0) THEN
            pause 50
            if k = 0  then  k=1
            k  =  k - 1
            portd = k
      endif
      PAUSE  150
      GOTO     LOOP
      END
```

โปรแกรมคำสั่งที่ 2 Step operation

```
S1        VAR      PORTA.3
S2        VAR      PORTA.4
k         var      byte
        TRISA    =   %111111
        TRISD    =   %00000000
        ADCON1  = 7
        portd = 0
LOOP:   IF  (S1 =  0) AND (S2 = 1)  THEN
              pause 50
              k  =  k+1
              portd = k
              if k = 255 then k = 254
          idle1: if (S1 =  0) AND (S2 = 1)  THEN idle1
          endif
               IF  (S1 =  1) AND (S2 = 0)  THEN
              pause 50
              if k = 0  then  k=1
              k  =  k - 1
              portd = k
          idle2: if (S1 =  1) AND (S2 = 0)  THEN idle2
          endif
           PAUSE  150
           GOTO    LOOP
           END
```

โปรแกรมทั้งสองมีลักษณะคล้ายกัน แต่โปรแกรมที่ 2 มีข้อแตกต่างจากโปรแกรมที่ 1 คือจะมีลูปการทำงานล็อกให้วนค้างไว้เมื่อมีการกดสวิทช์ใดสวิทช์หนึ่งค้างไว้จนกว่าจะถอนนิ้วจากปุ่มสวิทช์

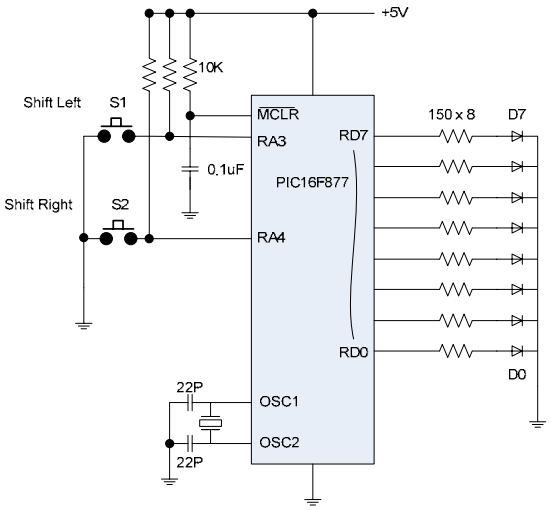
Experiment 7 การเขียนโปรแกรมเพื่อควบคุมการเลื่อนขยับบิทข้อมูลในตัวแปร

จุดประสงค์ เพื่อศึกษาการเขียนโปรแกรมในการควบคุมการเลื่อนขยับบิทของข้อมูลในตัวแปร ด้วยวิธีการต่าง ๆ

ทฤษฎีพื้นฐาน

การเลื่อน และขยับบิทข้อมูลในตัวแปรเป็นพื้นฐานสำคัญทั้งในการประมวลผลทางคณิตศาสตร์ และการประยุกต์ใช้งาน เช่นการควบคุมจังหวะการทำงานของเครื่องจักร การทำสัญญาณควบคุมไฟวิ่งเป็นต้น เครื่องหมายที่ใช้ในการประมวลผลที่สำคัญในการเลื่อนบิทไปทางซ้าย - ขวา คือ << และ >> ข้อสำคัญในการใช้คำสั่ง คือ เมื่อการเลื่อนข้อมูลไปถึงบิทสุดท้าย ทั้ง MSB หรือ LSB ก็ตาม เมื่อเลยไปจากนี้ บิทข้อมูลจะตกขอบ ไม่มีการย้อนกลับไปเริ่มต้นใหม่ ซึ่งแตกต่างไปจากการหมุน(Rotate) บิทข้อมูล และในภาษา Pic Basic จะไม่มีคำสั่งสำหรับการหหมุนข้อมูลรองรับ ดังนั้นหากต้องการ จะต้องใช้ภาษาแอสเซมบลี แทรกเอาเอง หรือต้องเขียนโปรแกรมให้โหลดข้อมูลย้อนกลับมาเริ่มต้นใหม่

วงจรทดลองตาม Experiment 7



โปรแกรมคำสั่งที่ 1 การควบคุมการเลื่อนบิทไปทางซ้าย – ขวา

```
S1      VAR     PORTA.3
S2      VAR     PORTA.4
k       VAR     byte
      TRISA    =   %111111
      TRISD    =   %00000000
      ADCON1   =   7
      portd = %10000000
LOOP:  PAUSE  400
          IF  (S1 =  0) THEN
                    portd  =  portd >> 1
                    if portd = %00000000 then portd = %10000000
           else
                    portd  =  portd << 1
                   if portd = %00000000 then portd = %00000001
           endif
           GOTO   LOOP
           END
```

โปรแกรมคำสั่งที่ 2 การควบคุมการขยับบิทไปทางซ้าย – ขวา ทีละสเต็ป

```
S1        VAR      PORTA.3
S2        VAR      PORTA.4
k         VAR      byte
      TRISA     =   %111111
      TRISD     =   %00000000
      ADCON1  = 7
      portd = %10000000
LOOP:
       IF  (S1 =  0) and (S2 = 1) THEN
                pause 50
                portd  =  portd >> 1
                if portd = %00000000     then portd    = %10000000
                idle1: IF  (S1 =  0) and (S2 = 1) THEN idle1
      endif
      if (S1 =  1) and (S2 = 0) THEN
                pause 50
                portd  =  portd << 1
                if portd = %00000000      then portd    = %00000001
                idle2: IF  (S1 =  1) and (S2 = 0) THEN idle2
      endif
      GOTO   LOOP
      END
```

โปรแกรมคำสั่งที่ 3 การควบคุมการขยับบิทไปทางซ้าย – ขวา แบบค้างสถานะต่อเนื่อง

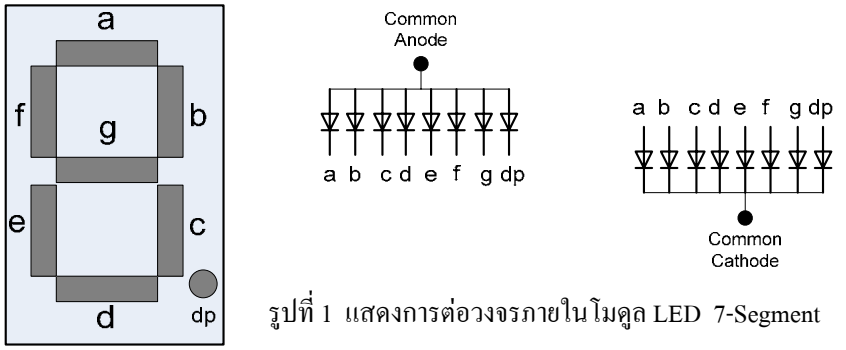
```
S1          VAR      PORTA.3
S2          VAR      PORTA.4
k           VAR      byte
direction   VAR      bit
      TRISA     =   %111111
      TRISD     =   %00000000
      ADCON1  = 7
      direction   = 0
      portd = %10000000
LOOP:
      IF   (S1 =  0) and (S2 = 1) THEN direction = 1
      if    (S1 =  1) and (S2 = 0) THEN direction = 0
      pause 50
      if direction = 1  then
            portd  =  portd >> 1
            if portd = %00000000 then portd = %10000000
      else
            portd  =  portd << 1
            if portd = %00000000 then portd = %00000001
      endif
      pause 150
      GOTO  LOOP
      END
```

Experiment 8 การเขียนโปรแกรมเพื่อควบคุมการแสดงผลทาง LED 7- Segment 1 หลัก

จุดประสงค์ เพื่อศึกษาการเขียนโปรแกรมในการควบคุมการแสดงผลทาง LED 7- Segment แบบ Common Anode

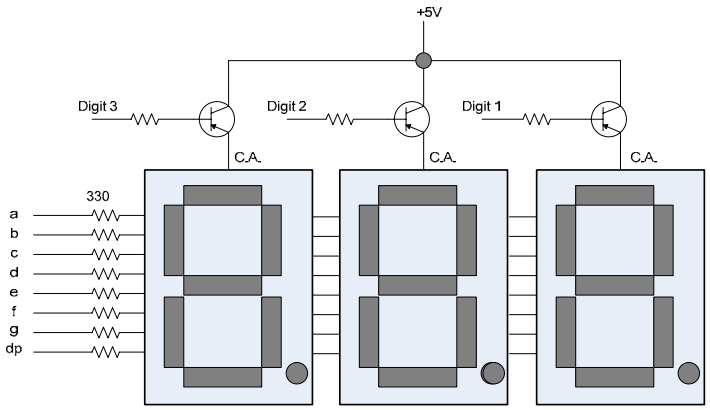
ทฤษฎีพื้นฐาน

การแสดงผลตัวเลขแบบ LED 7 – Segment ปัจจุบันยังคงมีใช้งานอย่างต่อเนื่อง เนื่องจากมีความสว่างเห็น และอ่านตัวเลขได้ชัดเจน LED 7 – Segment แบบตัวเล็กขนาดความสูงไม่เกิน 1 นิ้ว ส่วนมากจะสร้างเป็นโมดูล โมดูลละ 1 หลัก 2 หลัก 4 หลัก หรือมากกว่านั้น แต่ละโมดูลจะมีโครงสร้างประกอบด้วย หลอด LED ต่อเรียงกันเป็นตัวเลข 7 ส่วน การต่อจะมี 2 แบบ คือ แบบอะโนดร่วม(Common Anode) และแคโธดร่วม(Common Cathode)

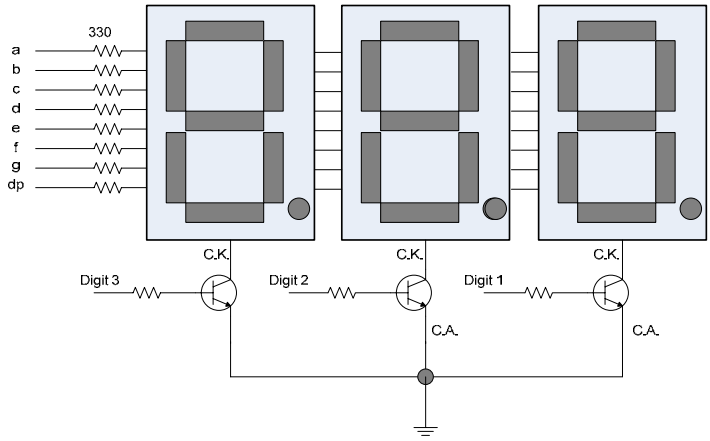


รูปที่ 1 แสดงการต่อวงจรภายในโมดูล LED 7-Segment แบบ Common Anode และแบบ Common Cathode

ในการต่อวงจรเพื่อแสดงผลสำหรับโมดูลแบบ Common Anode จะต้องต่อไฟ +5V เข้าที่ขา Common และหากต้องการให้ Segment ใดติดสว่าง จะต้องต่อ Segment นั้นลง Ground หรือป้อนลอจิก 0 ส่วนโมดูลแบบ Common Cathode จะตรงกันข้าม คือ ที่ขา Common ต้องต่อลง Ground และต้องป้อนไฟบวก หรือลอจิก 1 ที่ Segment ที่ต้องการให้ติดสว่าง ในการออกแบบวงจรขับจะต้องต่อตัวต้านทานอนุกรมเข้าที่ขา Segment เพื่อจำกัดกระแสไหลเข้าหลอด LED ซึ่งไม่ให้เกิน 20 mA ตัวต้านทานที่มาต่ออนุกรมจะมีค่าระหว่าง 150 – 470 โอห์ม สำหรับกระแสรวมที่ขา Common หากติดทุก Segment จะมีค่าประมาณ 100 – 160 mA ดังนั้นในการขับหลายโมดูลแบบมัลติเพล็กซ์ ไม่สามารถต่อกับขา I/O ของไมโครคอนโทรลเลอร์ได้โดยตรง จำเป็นต้องใช้ทรานซิสเตอร์เป็นตัวขับ ตามรูปที่ 2



รูปที่ 2 แสดงการต่อแสดงผล 3 หลักแบบมัลติเพล็กซ์ ของโมดูลแบบ Common Anode



รูปที่ 3 แสดงการต่อแสดงผลแบบมัลติเพล็กซ์ของโมดูลแบบ Common cathode

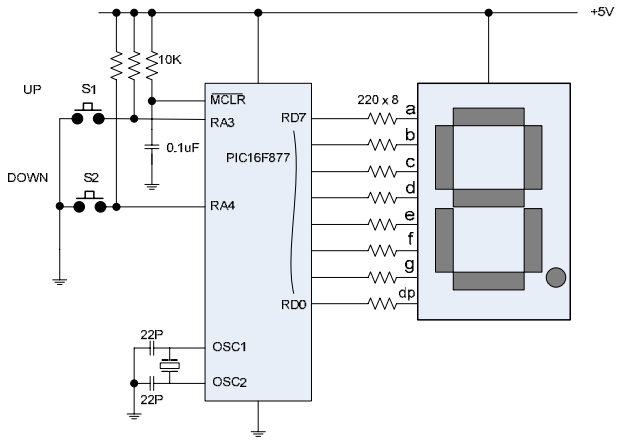
ในการต่อวงจรแสดงผลแบบมัลติเพล็กซ์ตามรูปที่ 2 และ 3 ขา Segment จะต่อพ่วงกันทุกหลัก และขา Common จะต้องใช้ทรานซิสเตอร์มาขับ เทคนิคในการแสดงผล 3 หลักแบบมัลติเพล็กซ์ จะต้องเขียนโปรแกรมให้ส่งเอ้าพุทออกมาเปิดแสดงผลทีละหลัก(Digit) เรียงกันไป โดยเริ่มตั้งแต่หลักหน่วย ไปจนถึงหลักร้อย แสดงซ้ำ ๆ กันด้วยความเร็วเกิน 30 ครั้ง ต่อนาทีสายตาเราจะมองเสมือนติดทั้ง 3 หลักพร้อม ๆ กัน เทคนิคการแสดงผลแบบมัลติเพล็กซ์มีข้อดี คือลดจำนวนขา I/O ของไมโครคอนโทรลเลอร์ที่จะใช้ขับ แต่มีข้อเสีย คือโปรแกรมจะต้องเสียเวลาส่วนใหญ่ในการวนกลับมาขับแสดงผลซ้ำเพื่อไม่ให้กระพริบ สำหรับการแสดงผลแบบหลักเดียวเราสามารถต่อขา Common เข้ากับแหล่งจ่ายไฟได้โดยตรงโดยไม่ต้องใช้ทรานซิสเตอร์มาขับ ตามวงจรในบอร์ดทดลอง จะต้องถอด Jumper ทั้ง 8 ตัวออกเพื่อใช้สายต่อขา I/O เข้ากับขา Segment โดยตรง ในการเขียนโปรแกรมเราต้องใช้คำสั่ง LOOKUP มาใช้เพื่อเก็บค่ารหัสที่ขับ Segment เป็นตัวเลข 0 – 9 ไว้ในตาราง รหัสขับ Segment สำหรับโมดูลแบบ Common Anode ได้จัดทำเป็นเลขฐานสิบหก(Hexa-decimal) ตามตารางรูปที่ 4

Decimal	Hex	dp	g	f	e	d	c	b	a
0	$C0	1	1	0	0	0	0	0	0
1	$F9	1	1	1	1	1	0	0	1
2	$A4	1	0	1	0	0	1	0	0
3	$B0	1	0	1	1	0	0	0	0
4	$99	1	0	0	1	1	0	0	1
5	$92	1	0	0	1	0	0	1	0
6	$82	1	0	0	0	0	0	1	0
7	$F8	1	1	1	1	1	0	0	0
8	$80	1	0	0	0	0	0	0	0
9	$90	1	0	0	1	0	0	0	0
dp		0							

รูปที่ 4 แสดงตารางเลขรหัสขับแสดงตัวเลข 0 -9 ของโมดูล LED 7-Segment แบบ Common Anode

หมายเหตุ ให้ศึกษาเพิ่มเติมเรื่องการแปลงเลขฐานสอง ฐานสิบ และฐานสิบหก

วงจรทดลองตาม Experiment 8



โปรแกรมคำสั่ง

การทำงานของโปรแกรม เริ่มต้นโปรแกรมจะส่งเอ้าพุทขับแสดงตัวเลขนับขึ้นตลอดเวลา การนับขึ้นหรือลงโดยการกดสวิทช์ S1 และ S2

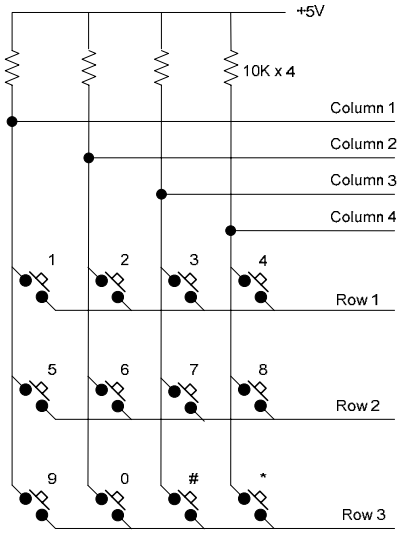
```
S1          var      PORTA.3  'for  count up
S2          var      PORTA.4  'for  count down
num         var      byte
disp        var      byte
direction   var      bit
      TRISA     =   %111111
      TRISD     =   %00000000
      ADCON1    =   7
      direction =   1
      num       =   0
      portd     =   255
LOOP:
          IF  (S1 =  0) and (S2 = 1) THEN direction = 1
          IF  (S1 =  1) and (S2 = 0) THEN direction = 0
          pause 50
          if direction = 1  then
                lookup num,[$c0,$f9,$a4,$b0,$99,_
                            $92,$82,$f8,$80,$90],disp
                portd = disp
                num  =  num + 1
                if num > 9 then num = 0
           else
                lookup num,[$c0,$f9,$a4,$b0,$99,_
                            $92,$82,$f8,$80,$90],disp
                portd = disp
                if num = 0 then num = 10
                num  =  num - 1
           endif
           pause  1000
          GOTO    LOOP
          END
```

Experiment 9 การเขียนโปรแกรมเพื่อรับอินพุทจาก Key Pad

จุดประสงค์ เพื่อศึกษาการเขียนโปรแกรมในการเพื่อรับอินพุทจาก Key Pad โดยใช้หลักการ Scan key

ทฤษฎีพื้นฐาน

การเขียนโปรแกรมเพื่อรับอินพุทจาก Key Pad เป็นพื้นฐานสำคัญที่จะนำไปประยุกต์ใช้งานที่จำเป็นต้องใช้จำนวนสวิทช์อินพุทจำนวนที่มากกว่าขา I/O ของไมโครคอนโทรลเลอร์ เช่นแป้นพิมพ์ตัวเลข ตัวอักษร สวิทช์กดควบคุมลิฟท์ เครื่องจักรกลการผลิต เป็นต้น วงจรประกอบด้วยขาสัญญาณในแนวระนาบ(row) และแนวตั้ง(column) และตัวสวิทช์แบบปุ่มกดจะต่ออยู่ตามจุดตัดระหว่างขาแนวระนาบ และขาแนวตั้งตามรูปที่ 1



รูปที่ 1 แสดงการต่อสวิทช์แบบ Key Pad ขนาด 3 Row x 4 Column

จากรูปที่ 1 จะเห็นว่าเราใช้ขา I/O เพียง 7 ขา แต่สามารถรับการติดตั้งสวิทช์อินพุทได้ถึง 12 ตัว ในทำนองเดียวกันนี้ หากเราใช้ขา I/O จำนวน 2 พอร์ท หรือ 16 ขา หรือ 8 Row x 8 Column จะสามารถต่อสวิทช์ป้อนสัญญาณอินพุทได้ถึง 64 ตัว จะทำให้ประหยัดจำนวนขา I/O ของไมโครคอนโทรลเลอร์ได้มาก

หลักในการเขียนโปรแกรมเพื่อสแกนรับสัญญาณลอจิกจากการกดปุ่มสวิทช์ที่ Key Pad จะต้องเขียนโปรแกรมให้ทำงานตามขั้นตอนดังต่อไปนี้คือ

1. กำหนดให้ขา I/O ของ MCU ที่ต่อกับสายสัญญาณ Column ทั้ง 4 เส้นเป็นอินพุท กำหนดให้ขา I/O ที่ต่อกับสายสัญญาณ Row ทั้ง 3 เส้นเป็นเอ้าพุท

2. ใช้คำสั่งให้ขา I/O ที่ต่อกับสายสัญญาณ Row ทั้ง 3 เส้น ส่งค่าลอจิก 1 เพื่อให้มีสถานะเดียวกันกับขา I/O ที่ต่อกับสาย Column ทั้ง 4 เส้น

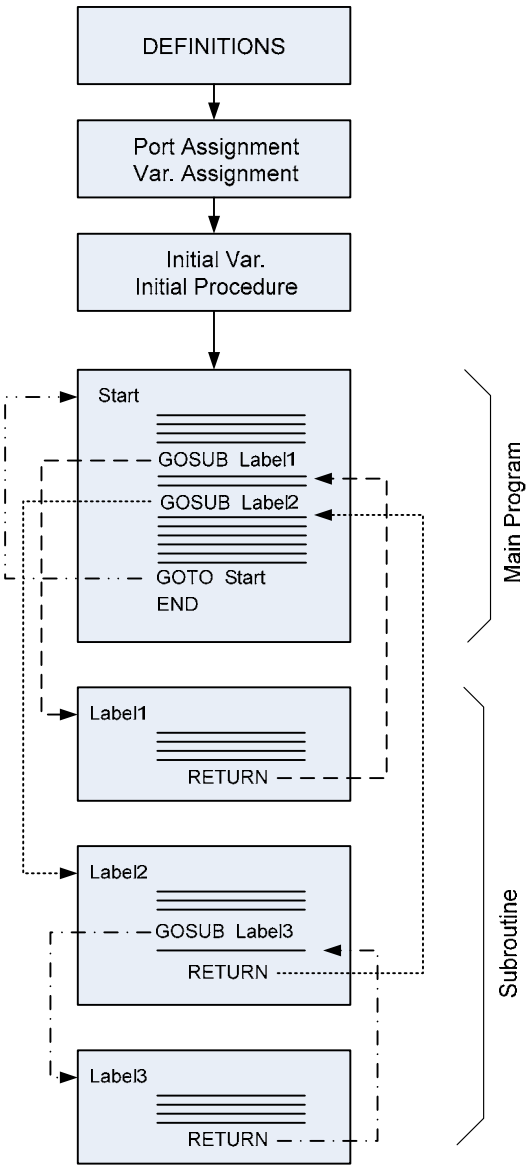
3. ส่งค่าลอจิก 0 ให้ Row 1 จากนั้นใช้คำสั่งตรวจรับสถานะลอจิกที่ Column 1 , Column 2 , Column 3 , และ Column 4 ตามลำดับ หากมีสถานะลอจิก 0 ที่ Column ใด แสดงว่าได้มีการกดสวิทช์ 1 , 2 , 3 , หรือ 4 เนื่องจากขาข้างหนึ่งของสวิทช์ทั้ง 4 ได้ต่อกับสาย Row 1 หาก Column ใดเป็นลอจิก 0 ก็ให้ไปทำงานในโปรแกรมย่อย (Subroutine) ที่ระบุไว้

4. เมื่อตรวจรับสถานการณ์กดสวิทช์ทั้ง 4 ตัวตามขั้นตอนที่ 3 เรียบร้อยแล้ว ให้ส่งค่าลอจิก 1 คืนให้ขา Row 1 แล้วส่งค่าลอจิก 0 ให้ และในทำนองเดียวกัน ช้คำสั่งตรวจรับสถานะลอจิกที่ Column 1 ,

Column 2 , Column 3 , และ Column 4 ตามลำดับ หากมีสถานะลอจิก 0 ที่ Column ใด แสดงว่าได้มีการกดสวิทช์ 5 , 6 , 7 , หรือ 8 หาก Column ใดเป็นลอจิก 0 ก็ให้ไปทำงานในโปรแกรมย่อย (Subroutine) ที่ระบุไว้

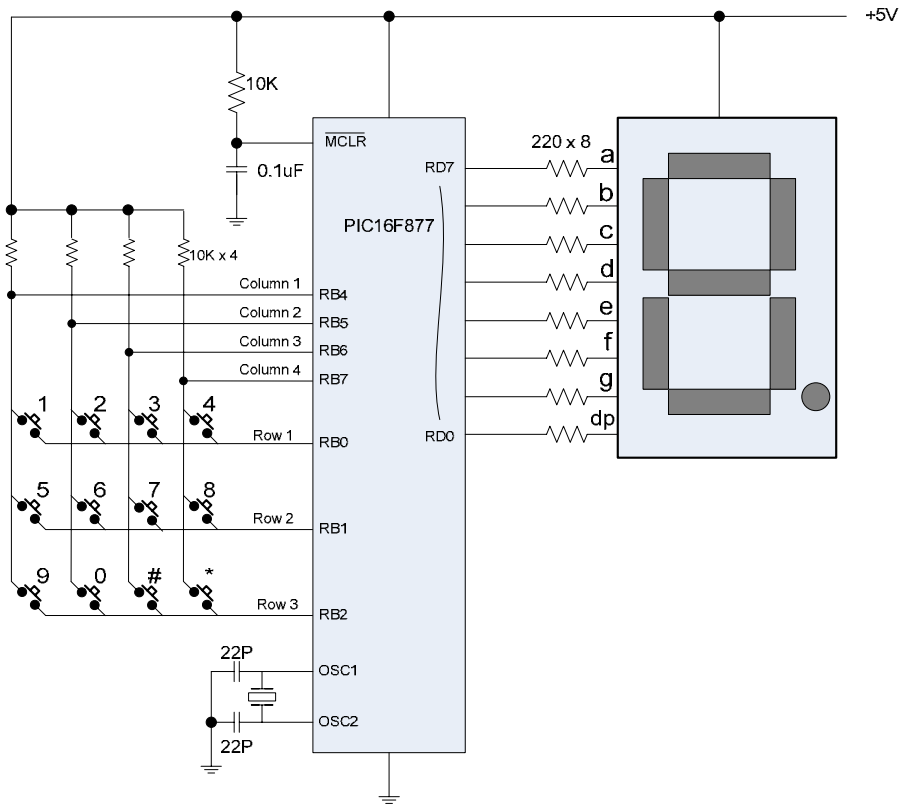
5. ทำในทำนองเดียวกันกับขั้นตอนที่ 3 หรือ 4 เพื่อตรวจรับสถานการณ์กดสวิทช์ของชุดที่ต่อกับ Row สุดท้าย จากนั้นจึงย้อนกลับไปเริ่มต้นทำขั้นตอนที่ 2 ใหม่ เราเรียกลูปการทำงานนี้ว่า “Scan key” เนื่องจากลูปการทำงานนี้เร็วมากเป็นไมโครวินาที การทำงานจึงเสมอนว่ามีการตรวจรับสถานะลอจิกจากการกดปุ่มสวิทช์พร้อมกันทุกตัว

เทคนิคในการเขียนโปรแกรมโดยใช้โปรแกรมหลัก(Main program) เป็นตัวเรียกโปรแกรมย่อย (Subroutine) ซึ่งทำงานเฉพาะงานใดงานหนึ่งมาประมวลผล ทำให้การเขียนโปรแกรมสั้น กระชับและประมวลผลเร็วขึ้นมาก คำสั่งในภาษา Pic Basic Pro Compiler ที่ใช้ได้แก่ GOSUB และ RETURN โดยมีรูปแบบการใช้งานตามรูปที่ 2



รูปที่ 2 แสดงเทคนิคการเขียนโปรแกรมโดยใช้โปรแกรมหลัก และโปรแกรมย่อย

วงจรทดลองตาม Experiment 9



การทำงานของโปรแกรมคำสั่ง

เมื่อเริ่มทำงานโปรแกรมจะวนลูปสแกนรับการกดปุ่มสวิทช์อยู่โดยไม่แสดงหมายเลข เมื่อกดปุ่มสวิทช์หมายเลขใด ให้แสดงหมายเลขปุ่มที่กดที่หมายเลขนั้นค้างไว้ และจะย้อนกลับไปวนลูปสแกนรับการกดปุ่มสวิทช์อยู่ จนกว่าจะกดหมายเลขอื่นต่อไป โปรแกรมนี้เป็นแค่พื้นฐานการเรียนรู้ ซึ่งจะมีการประยุกต์ต่อไป

```
row1    var     portb.0
row2    var     portb.1
row3    var     portb.2
col1    var     portb.4
col2    var     portb.5
col3    var     portb.6
col4    var     portb.7
num     var     byte
disp    var     byte
        TRISB    =    %11110000
        TRISD    =    %00000000
        num      = 0
        portd    = 255
        high  row1
        high  row2
        high  row3
'----------------------main  program
start:
        gosub scan_key
        pause 50
        goto  start
        end
'----------------end of main program
```

มีต่อหน้าถัดไป

```
'
'------------------------subroutines
scan_key:
        low   row1
        if col1 = 0 then gosub num1
        if col2 = 0 then gosub num2
        if col3 = 0 then gosub num3
        if col4 = 0 then gosub num4
        high  row1
        low   row2
        if col1 = 0 then gosub num5
        if col2 = 0 then gosub num6
        if col3 = 0 then gosub num7
        if col4 = 0 then gosub num8
        high   row2
        low    row3
        if col1 = 0 then gosub num9
        if col2 = 0 then gosub num0
        if col3 = 0 then gosub num10
        if col4 = 0 then gosub num11
        high   row3
        return
'-------------------------------------
num1:   num = 1
       lookup num,[$c0,$f9,$a4,$b0,$99,_
                        $92,$82,$f8,$80,$90],disp
       portd = disp
       return
'-------------------------------------
num2:   num = 2
       lookup num,[$c0,$f9,$a4,$b0,$99,_
                        $92,$82,$f8,$80,$90],disp
       portd = disp
       return
'-------------------------------------
num3:   num = 3
       lookup num,[$c0,$f9,$a4,$b0,$99,_
                        $92,$82,$f8,$80,$90],disp
       portd = disp
       return
'-------------------------------------
num4:   num = 4
       lookup num,[$c0,$f9,$a4,$b0,$99,_
                        $92,$82,$f8,$80,$90],disp
       portd = disp
       return
'-------------------------------------
num5:   num = 5
       lookup num,[$c0,$f9,$a4,$b0,$99,_
                        $92,$82,$f8,$80,$90],disp
       portd = disp
       return
'-------------------------------------
num6:   num = 6
       lookup num,[$c0,$f9,$a4,$b0,$99,_
                        $92,$82,$f8,$80,$90],disp
       portd = disp
       return
'-------------------------------------
num7:   num = 7
       lookup num,[$c0,$f9,$a4,$b0,$99,_
                        $92,$82,$f8,$80,$90],disp
       portd = disp
       return
'-------------------------------------
```

มีต่อ

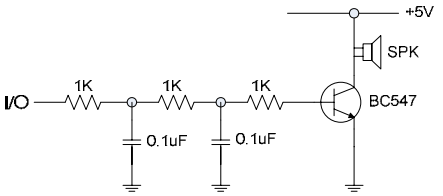
```
'-------------------------------------
num8:   num = 8
       lookup num,[$c0,$f9,$a4,$b0,$99,_
                        $92,$82,$f8,$80,$90],disp
       portd = disp
       return
'-------------------------------------
num9:   num = 9
       lookup num,[$c0,$f9,$a4,$b0,$99,_
                         $92,$82,$f8,$80,$90],disp
       portd = disp
       return
'-------------------------------------
num0:   num = 0
       lookup num,[$c0,$f9,$a4,$b0,$99,_
                        $92,$82,$f8,$80,$90],disp
       portd = disp
       return
'-------------------------------------
num10:   num = 10
       lookup num,[$c0,$f9,$a4,$b0,$99,_
                        $92,$82,$f8,$80,$90],disp
       portd = disp
       return
'-------------------------------------
num11:   num = 11
       lookup num,[$c0,$f9,$a4,$b0,$99,_
                        $92,$82,$f8,$80,$90],disp
       portd = disp
       return
'----------------------end of  subroutine
```

Experiment 10 การเขียนโปรแกรมเพื่อขับสัญญาณเสียง

จุดประสงค์ เพื่อศึกษาการเขียนโปรแกรมในการเพื่อส่งค่าเอ้าพุทออกเป็นสัญญาณคลื่นความถี่เสียงในลักษณะ ต่าง ๆ

ทฤษฎีพื้นฐาน

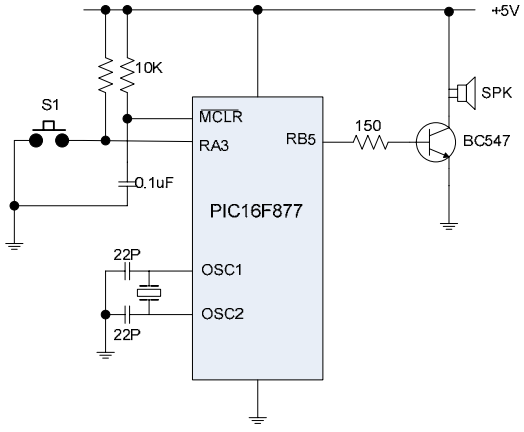
การเขียนโปรแกรมเพื่อให้ไมโครคอนโทรลเลอร์กำเนิดสัญญาณความถี่เสียงออกมา เป็นพื้นฐานสำคัญที่จะนำไปประยุกต์ใช้งานด้านต่าง ๆ เพื่อจะช่วยสื่อให้ผู้ใช้งานได้ยินว่ากำลังทำอะไรอยู่ เช่นเสียงคลิกขณะกำลังกดแป้นพิมพ์ หรือกดปุ่มสวิทช์อยู่ หรือเสียงเตือนให้ระวัง หรือเกิดการทำงานผิดพลาด เป็นต้น คำสั่งที่ทำให้เกิดเสียงในภาษา PIC BASIC PRO COMPILER มีหลายคำสั่งได้แก่ FREQOUT SOUND และ DTMFOUT แต่ละคำสั่งมีจุดประสงในการใช้งานที่แตกต่างกัน นอกจากนี้เรายังเขียนเป็นโปรแกรมที่จะทำให้เกิดเสียงได้อย่างง่าย หรืออาจใช้ฟังชั่นการทำงานของ Timer module ที่อยู่ภายในตัว MCU ก็ได้เช่นกัน เนื่องจากสัญญาณเสียงที่ขับออกมานี้เป็นสัญญาณพัลส์ ดังนั้นอาจทำให้มีสัญญาณฮาร์โมนิคออกมาด้วย หากต้องการกำจัดออกต้องใช้วงจรกรองความถี่สูง(Low pass filter)ออกตามรูปที่ 1 สำหรับการใช้งานในการ



รูปที่ 1 วงจรกรองความถี่ Low Pass Filter

กำเนิดเสียงบีบในบอร์ดไมโครคอนโทรลเลอร์ โดยทั่วไปไม่จำเป็นต้องใช้วงจร Low Pass Filter ตามรูปที่ 1

ตัวขับเสียงที่นิยมใช้ในบอร์ดไมโครคอนโทรลเลอร์ โดยทั่วไปใช้ขนาดเล็ก ๆ สีดำ หรือเรียกว่า “Electro-magnetic Transducer” ซึ่งมี 2 แบบ คือ Internal drive ซึ่งบรรจุวงจรกำเนิดเสียงบีบไว้ในตัว เพียงแต่ป้อนไฟตรง +5V ก็จะได้ยินเสียงบีบทันที หรือเราเรียกอีกชื่อหนึ่งว่า “Buzzer” ส่วนอีกแบบหนึ่ง คือ External drive แบบนี้ไม่มีวงจรขับเสียงในตัว จำเป็นต้องเขียนโปรแกรมให้ MCU ขับสัญญาณเสียงออกมาจึงจะมีเสียง สำหรับในบอร์ดทดลองตามใบงาน จะเป็นชนิด External drive



รูปที่ 2 วงจรทดลองตามExperiment 10

โปรแกรมคำสั่ง

โปรแกรมคำสั่งที่ปฏิบัติมี 3 โปรแกรม โดยจะใช้คำสั่งกำเนิดเสียงที่แตกต่างกัน เพื่อให้มีแนวความคิดที่จะนำไปประยุกต์ใช้งานที่หลากหลาย การทำงานของโปรแกรม เมื่อเริ่มทำงานโปรแกรมจะรอการกดสวิทช์ หลังจากผู้ใช้กดสวิทช์โปรแกรมจะทำให้ MCU กำเนิดเสียงออกมา ตามจำนวนครั้งที่กำหนด เสร็จแล้วจะย้อนกลับไปเริ่มต้นรอการกดสวิทช์ใหม่ รายละเอียดการทำงานคำสั่งให้ศึกษาจากคู่มือการใช้โปรแกรม Pic Basic Pro

โปรแกรมที่ 1

```
S1        var     PORTA.3
spk       var     PORTB.5
i         var     byte
  TRISA     =  %11111111
  TRISB     =  %00000000
  ADCON1 = 7
  PORTB    = 0
start:  if s1 = 1 then goto start
       pause 50
loop:   if s1 = 0 then goto loop
       for i = 1 to 5
          freqout spk,100,2000
          pause 1000
       next i
       goto start
       end
```

โปรแกรมที่ 2

```
S1        var     PORTA.3
spk       var     PORTB.5
i         var     byte
  TRISA     =  %11111111
  TRISB     =  %00000000
  ADCON1 = 7
  PORTB    = 0
start:  if s1 = 1 then goto start
       pause 50
loop:   if s1 = 0 then goto loop
       for i = 1 to 5
          sound spk,[100,10,50,10]
          pause 1000
       next i
       goto start
       end
```

โปรแกรมที่ 3

```
S1        VAR     PORTA.3
spk       VAR         PORTB.5
i   var     byte
  TRISA     =  %11111111
  TRISB     =  %00000000
  ADCON1 = 7
  PORTB   = 0
start:  if s1 = 1 then goto start
       pause 50
loop:   if s1 = 0 then goto loop
       for i = 1 to 3
          dtmfout spk,[0,2,9,4,3,8,4,9,0]
          pause 1000
       next i
       goto start
       end
```

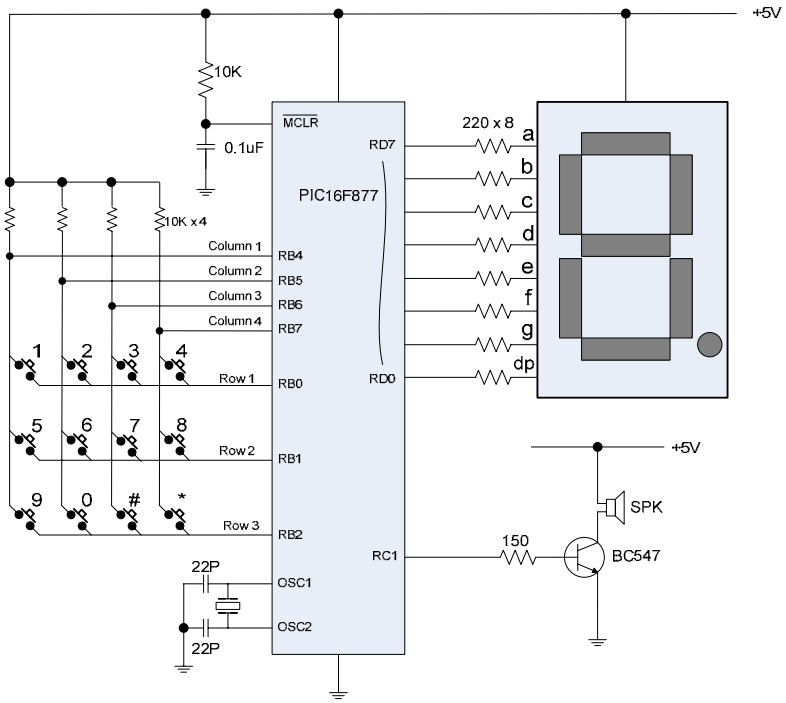
Experiment 11 การเขียนโปรแกรมเพื่อประยุกต์การกำเนิดเสียงในการกดแป้นคีย์ตัวเลข

จุดประสงค์ เพื่อศึกษาการเขียนโปรแกรมในการเพื่อประยุกต์ส่งค่าเอ้าพุทออกเป็นสัญญาณคลื่นความถี่เสียงในการกดแป้นคีย์ตัวเลข (Key pad)

ทฤษฎีพื้นฐาน

ในการกดแป้นคีย์ตัวเลข เช่นแป้นคีย์ตัวเลขป้อนค่าของเครื่องจักรกลต่าง ๆ แป้นคีย์โทรศัพท์ เป็นต้น เสียงกำเนิดในการกดแป้นคีย์โทรศัพท์ความถี่จะเปลี่ยนไปตามตัวเลข ซึ่งเป็นความถี่ที่ได้กำหนดไว้เป็นมาตรฐาน การส่งรหัสตัวเลข ในการแทรกเสียงเข้าไประหว่างการกดตัวเลข กระทำได้โดยแทรกโปรแกรมย่อยกำเนิดเสียงขณะกดคีย์สวิทช์ตัวเลข

วงจรทดลองตาม Experiment 11



โปรแกรมคำสั่ง

```
row1   var    portb.0
row2   var    portb.1
row3   var    portb.2
col1   var    portb.4
col2   var    portb.5
col3   var    portb.6
col4   var    portb.7
spk    var    portc.1
num    var    byte
disp   var    byte
       TRISB    =   %11110000
       TRISD    =   %00000000
       num = 0
       portd = 255
       high  row1
       high  row2
       high  row3
```

มีต่อหน้าถัดไป

```
'----------------------main  program
start:
            gosub scan_key
        pause 50
        goto  start
        end
'-----------------end of main program
'
'------------------------subroutines
scan_key:
        low   row1
        if col1 = 0 then gosub num1
        if col2 = 0 then gosub num2
        if col3 = 0 then gosub num3
        if col4 = 0 then gosub num4
        high  row1
        low   row2
        if col1 = 0 then gosub num5
        if col2 = 0 then gosub num6
        if col3 = 0 then gosub num7
        if col4 = 0 then gosub num8
        high   row2
        low    row3
        if col1 = 0 then gosub num9
        if col2 = 0 then gosub num0
        if col3 = 0 then gosub num10
        if col4 = 0 then gosub num11
        high   row3
        return
'--------------------------------------
num1:   num = 1
        lookup num,[$c0,$f9,$a4,$b0,$99,_
                $92,$82,$f8,$80,$90],disp
        gosub   speak1
        portd = disp
        return
'--------------------------------------
num2:   num = 2
        lookup num,[$c0,$f9,$a4,$b0,$99,_
                $92,$82,$f8,$80,$90],disp
        gosub   speak2
        portd = disp
        return
'--------------------------------------
num3:   num = 3
        lookup num,[$c0,$f9,$a4,$b0,$99,_
                $92,$82,$f8,$80,$90],disp
        gosub   speak3
        portd = disp
        return
'--------------------------------------
num4:   num = 4
        lookup num,[$c0,$f9,$a4,$b0,$99,_
                $92,$82,$f8,$80,$90],disp
        gosub   speak4
        portd = disp
        return
'--------------------------------------
num5:   num = 5
        lookup num,[$c0,$f9,$a4,$b0,$99,_
                $92,$82,$f8,$80,$90],disp
        gosub   speak5
        portd = disp
        return
'--------------------------------------
```

มีต่อหน้าถัดไป

```
num6:   num = 6
        lookup num,[$c0,$f9,$a4,$b0,$99,_
                $92,$82,$f8,$80,$90],disp
        gosub   speak6
        portd = disp
        return
'---------------------------------------
num7:   num = 7
        lookup num,[$c0,$f9,$a4,$b0,$99,_
                $92,$82,$f8,$80,$90],disp
        gosub   speak7
        portd = disp
        return
'---------------------------------------
num8:   num = 8
        lookup num,[$c0,$f9,$a4,$b0,$99,_
                $92,$82,$f8,$80,$90],disp
        gosub   speak8
        portd = disp
        return
'---------------------------------------
num9:   num = 9
        lookup num,[$c0,$f9,$a4,$b0,$99,_
                $92,$82,$f8,$80,$90],disp
        gosub   speak9
        portd = disp
        return
'---------------------------------------
num0:   num = 0
        lookup num,[$c0,$f9,$a4,$b0,$99,_
                $92,$82,$f8,$80,$90],disp
        gosub   speak0
        portd = disp
        return
'---------------------------------------
num10:   num = 10
        lookup num,[$c0,$f9,$a4,$b0,$99,_
                $92,$82,$f8,$80,$90],disp
        portd = disp
        return
'---------------------------------------
num11:   num = 11
        lookup num,[$c0,$f9,$a4,$b0,$99,_
                $92,$82,$f8,$80,$90],disp
        portd = disp
        return
'---------------------------------------
speak0: dtmfout spk,[0]:return
speak1: dtmfout spk,[1]:return
speak2: dtmfout spk,[2]:return
speak3: dtmfout spk,[3]:return
speak4: dtmfout spk,[4]:return
speak5: dtmfout spk,[5]:return
speak6: dtmfout spk,[6]:return
speak7: dtmfout spk,[7]:return
speak8: dtmfout spk,[8]:return
speak9: dtmfout spk,[9]:return
'----------------------end of  subroutine
```

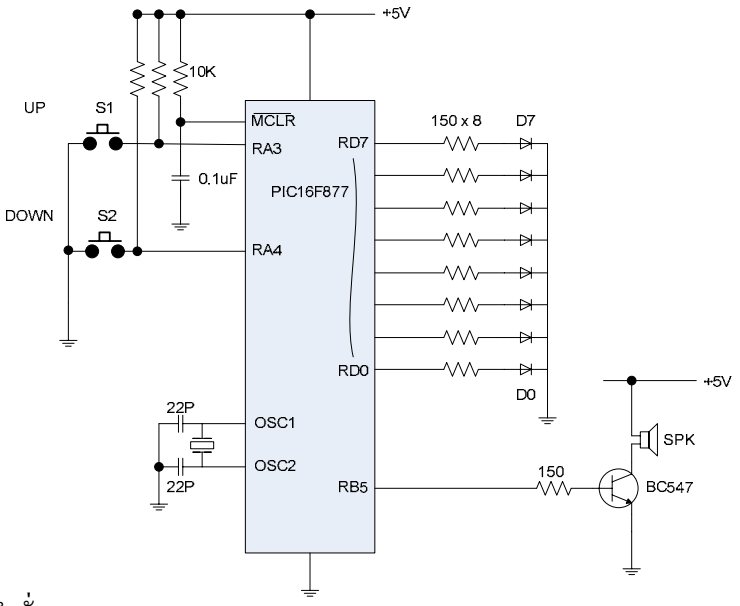
Experiment 12 การเขียนโปรแกรมเพื่อประยุกต์การกำเนิดเสียงคลิกขณะกดแป้นคีย์หรือสวิทช์

จุดประสงค์ เพื่อศึกษาการเขียนโปรแกรมในการเพื่อประยุกต์การสร้างเสียงคลิกขณะกดแป้นคีย์ หรือสวิทช์

ทฤษฎีพื้นฐาน

ในการกดสวิทช์เพื่อควบคุมการทำงานของเครื่องจักรกลและอุปกรณ์เครื่องใช้ต่าง ๆ จำเป็นต้องให้มีเสียงคลิกเพื่อยืนยันการรับค่าลอจิกจากการกดให้ผู้ใช้ได้รับรู้ ตัวอย่างเช่นรีโหมด หรือปุ่มควบคุมการปรับตั้งอุณหภูมิเครื่องปรับอากาศ เป็นต้น หลักการ คือการเขียนโปรแกรมกำเนิดเสียงบีบสั้น ๆ คล้ายกับเสียงคลิก เป็นโปรแกรมย่อยแทรกไว้ให้ทำงานหลังจากการกดปุ่มสวิทช์

วงจรทดลองตาม Exxperiment 12



โปรแกรมคำสั่ง

การทำงานของโปรแกรม เมื่อกดสวิทช์ S1 และ S2 เพื่อเพิ่ม และลดค่าทุกครั้งจะได้ยินเสียงคลิกจากการกด

```
S1        VAR      PORTA.3
S2        VAR      PORTA.4
Spk       var       portb.5
k         var       byte
      TRISA     =    %111111
      TRISD          =    %00000000
      ADCON1 = 7
      portd = 0
LOOP:   IF  (S1 =  0) AND (S2 = 1)  THEN
                    pause 50
                    k  =  k+1
                    gosub click
                     portd = k
                     if k = 255 then k = 254
          idle1: if (S1 =  0) AND (S2 = 1)  THEN idle1
          endif
```

มีต่อหน้าถัดไป

```
        IF  (S1 =  1) AND (S2 = 0)  THEN
                    pause 50
                   if k = 0  then  k=1
                    k  =  k - 1
                   gosub click
                    portd = k
          idle2: if (S1 =  1) AND (S2 = 0)  THEN idle2
          endif
          PAUSE  150
          GOTO     LOOP
      END
'
'------------------ End of Main Program -----------
'
'------- Subroutine  Start  Here ------------------
click:
    freqout spk,10,2000
    return
'------- End of Subroutine ------------------------
```

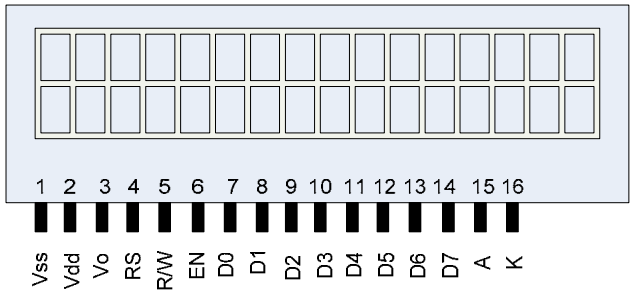
แนวคิด เราสามารถนำโปรแกรมย่อยที่สร้างเสียงคลิกใช้กับโปรแกรมอื่น ๆ ที่มีการกดปุ่มสวิทช์ควบคุม แต่ต้องคำนึงถึงเวลาที่ต้องแบ่งให้ไปกับการสร้างเสียง หากเป็นโปรแกรมที่ต้องการรับอินพุทที่รวดเร็ว การทำงานอาจจะมีปัญหา การสร้างเสียงคลิกควรทำให้เกิดเวลาสั้นที่สุดเท่าที่จะทำได้

Experiment 13 การเขียนโปรแกรมเพื่อแสดงผลผ่านจอแอลซีดี (LCD Display)

จุดประสงค์ เพื่อศึกษาการเขียนโปรแกรมในการเพื่อนำข้อมูลออกแสดงผลทางจอแอลซีดีในลักษณะต่าง ๆ

ทฤษฎีพื้นฐาน

จอแอลซีดี(LCD: Liquid Crystal Display) เป็นอุปกรณ์แสดงผลที่นิยมใช้กันมากเนื่องจากสามารถแสดงข้อมูลที่เป็นทั้งตัวเลข และตัวอักษรที่เป็นข้อความ ใช้งานง่าย และกินกระแสไฟน้อยกว่า LED ตัวจอLCD ผลิตเป็นโมดูล มีตัวไมโครคอนโทรลเลอร์ควบคุมการแสดงผลในตัวเอง สามารถแสดงผลได้ตั้งแต่บรรทัดละ 8 16 และ 32 ตัวอักขระ ตั้งแต่ 1 แถว ไปจนถึงหลายแถว จนถึงการแสดงแบบกราฟิก โปรแกรม Pic Basic Pro Compiler มีคำสั่งสนับสนุนการใช้งานกับจอ LCD ที่ใช้คอนโทรลเลอร์ ของ Hitachi 44780 หรือชิพเบอร์อื่นที่มีคุณสมบัติเดียวกัน เป็นตัวควบคุม ตัวโมดูลมีขา I/O ที่จะติดต่อกับ MCU จำนวน 14 และ 16 ขา ตามรูปที่ 1



รูปที่ 1 แสดงโครงสร้างของจอ LCD แบบ 16 ตัวอักขระ x 2 บรรทัด

รายละเอียดการต่อขาจอ LCD มีดังนี้คือ

ขา Vss เป็นขาต่อลงกราวด์ (GND)

ขา Vdd เป็นขาจ่ายไฟ +5 V

ขา Vo เป็นขาที่ต่อไฟสำหรับควบคุมความเข้ม (Contrast) ของตัวอักขระที่แสดงผล ปรับได้ 0 – 5 V ถ้าปรับเข้าใกล้ 0 V ตัวอักขระจะเข้มสุด หากไม่ต้องการปรับให้ต่อลง GND

ขา RS เป็นขาที่ใช้ควบคุมการป้อนคำสั่ง กับข้อมูลที่แสดงผล

ขา R/W เป็นขาที่ใช้ควบคุมว่าจะให้อ่านค่าตัวอักขระจากหน้าจอ หรือเขียนเขียนตัวอักขระลงไปที่หน้าจอ หากต้องการจะส่งข้อมูลออกแสดงผลเพียงอย่างเดียว ให้ต่อขานี้ลง GND

ขา EN เป็นขาควบคุมสถานการณ์อ่าน หรือ แสดงผลข้อความ

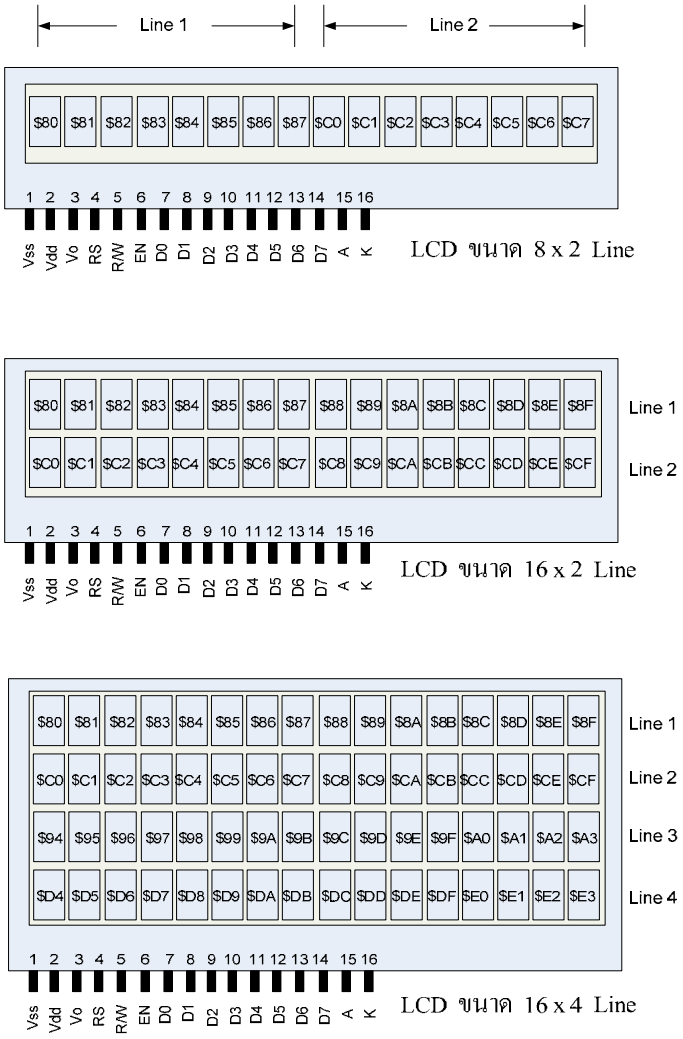
ขา D0 - D7 เป็นขาที่ส่งข้อมูลเข้า - ออก เพื่อแสดงผล

ขา A และ ขา K เป็นขาที่ต่อไฟ Back Light สำหรับเป็นจอที่มี Back light ที่เหมาะกับการใช้งานในที่มืด ไฟที่ต่อเป็นไฟตรง 5 V

ข้อความที่แสดงบนจอ แต่ละตัวอักขระจะมีหมายเลขตำแหน่งกำกับ โดยแต่ละแถว หรือบรรทัดจะมีหมายเลขตำแหน่งเริ่มต้น ซึ่งเป็นเลขฐาน 16 และจะมีรหัสควบคุมดังต่อไปนี้ คือ

Command	Operation
$FE, 1	Clear Display
$FE, 2	Return home (beginning of the first line)
$FE, $0C	Cursor off
$FE, $0E	Underline cursor on
$FE, $0F	Blinking cursor on
$FE, $10	Move cursor left one position
$FE, $14	Move cursor right one position
$FE, $C0	Move cursor to beginning of second line
$FE, $94	Move cursor to beginning of third line
$FE, $D4	Move cursor to beginning of fourth line

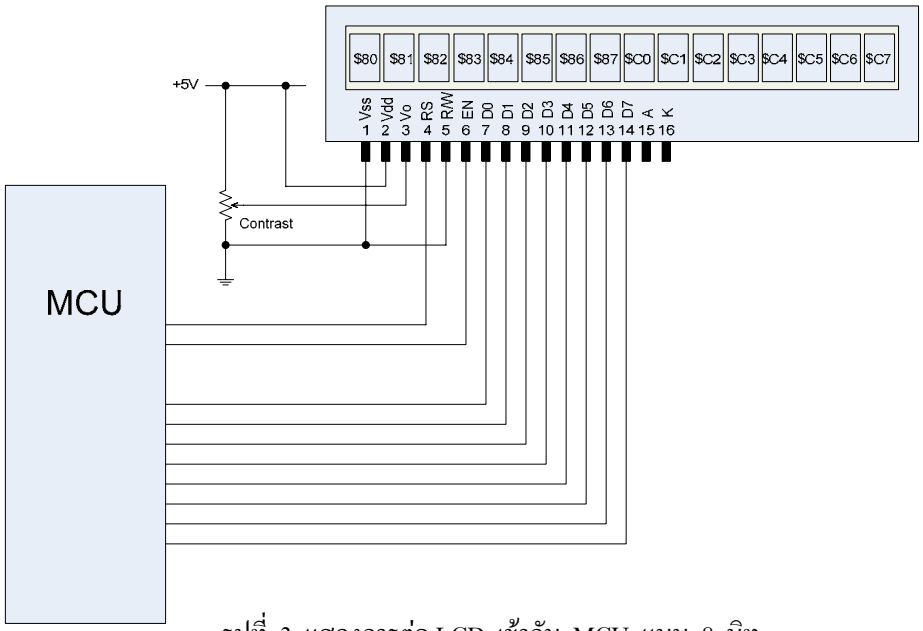
รหัสควบคุมเหล่านี้เป็นเลขฐานสิบหก การป้อนจะต้องมีรหัส $FE, นำหน้าก่อนเสมอ การป้อนตัวอักขระเข้าแสดงที่จอ LCD ต้องระบุตำแหน่งเริ่มต้นก่อนเสมอ หมายเลขรหัสประจำตำแหน่งของตัวอักขระมีดังนี้



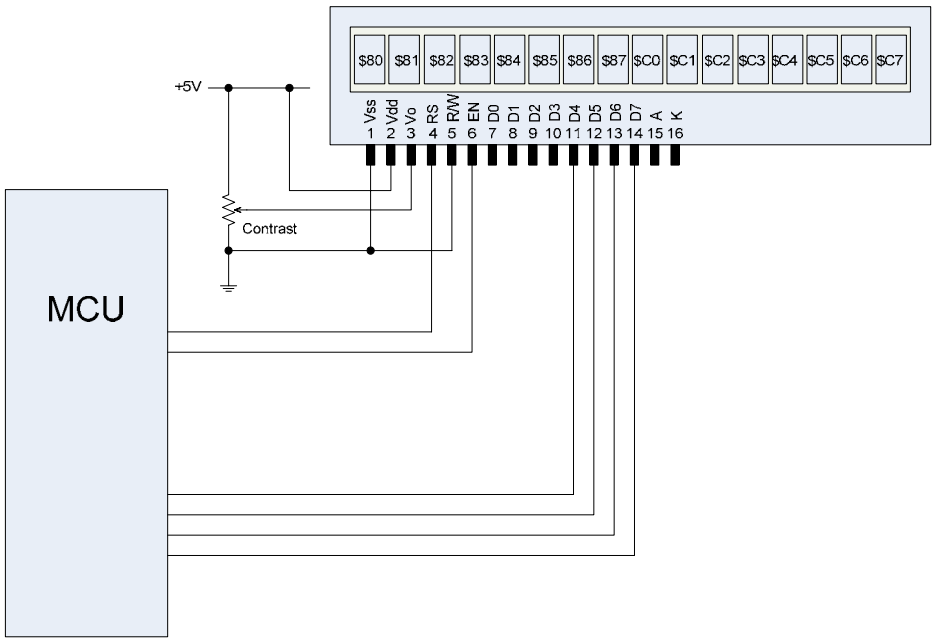
รูปที่ 2 แสดงโครงสร้าง และรหัสตำแหน่งตัวอักขระของ LCD แบบต่าง ๆ

นอกจากนั้นยังมี LCD ขนาดอื่น ๆ อีก เช่น 16 x 1 Line เป็นต้น LCD ทุกรุ่นทุกขนาด จะมีขา I/O แบบเดียวกัน ถ้าเป็นแบบที่ใช้คอนโทรลเลอร์แบบเดียวกัน หรือเหมือนกัน

การเชื่อมต่อจอ LCD เข้ากับไมโครคอนโทรลเลอร์ มีการต่อ 2 แบบ คือ แบบ 4 บิท และแบบ 8 บิท



รูปที่ 3 แสดงการต่อ LCD เข้ากับ MCU แบบ 8 บิท



รูปที่ 4 แสดงการต่อ LCD เข้ากับ MCU แบบ 4 บิท

สำหรับในการทดลองตามใบงานนี้ จะใช้การต่อวงจรแบบ 4 บิท ตามรูปที่ 4 เนื่องจากประหยัดขา I/O ของ MCU และในคำสั่งของ Pic Basic Pro ได้รองรับการการทำงานแบบนี้อยู่แล้ว

ในการเขียนโปรแกรมการแสดงผลผ่านทางจอ LCD ก่อนการเข้าสู่การประมวลผลเราต้องให้โปรแกรมรู้จักตำแหน่งของขา I/O ที่ต่อกับจอ LCD ขาต่าง ๆ ตามรูปที่ 4 ก่อน ด้วยการนิยามไว้ที่ส่วนหัวของโปรแกรม ตามตัวอย่างดังนี้ คือ

```
DEFINE LCD_DREG PORTD
DEFINE LCD_DBIT 4
DEFINE LCD_RSREG PORTE
DEFINE LCD_RSBIT 2
DEFINE LCD_EREG PORTD
DEFINE LCD_EBIT 1
```

ตามตัวอย่าง บรรทัดที่ 1 และ 2 กำหนดว่า ขา Data 4 เส้นต่อที่ Port D เริ่มบิทที่ 4 เป็นต้นไป

บรรทัดที่ 3 และ 4 กำหนดว่า ขา RS ต่ออยู่ที่ Port E ขา 2 (RE2)

บรรทัดที่ 5 และ 6 กำหนดว่า ขา EN ต่ออยู่ที่ Port D ขา 1 (RD1)

รายละเอียดนอกเหนือจากนี้ ให้ศึกษาเพิ่มเติมในคู่มือการใช้โปรแกรม

ในการแสดงผล เราสามารถปรับแต่งรูปแบบของข้อมูลที่จะส่งมาแสดงได้ เช่น เป็นเลขฐานสอง ฐานสิบ และฐานสิบหก เป็นต้น โดยจำมีตัวปรับค่าตามตารางดังต่อไปนี้

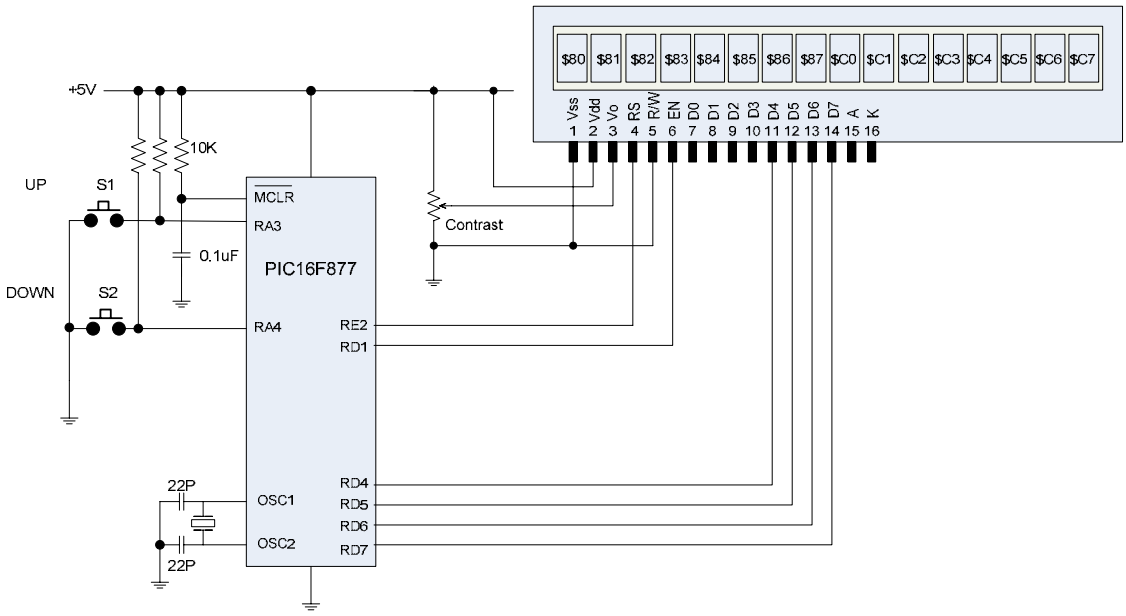
Modifier	Operation
BIN {1…16}	Send binary digit
DEC {1…5}	Send decimal digit
HEX {1…4}	Send hexa-decimal digit
REP c\n	Send character c repeated n times
STR ArrayVar{\n}	Send string of n characters

หมายเหตุ 1. หากมีการต่อขาของจอ LCD เข้ากับพอร์ท ที่สามารถรับสัญญาณอนาล็อกได้ เช่น PortA และ PortE ก่อนการใช้คำสั่งแสดงผล จะต้องกำหนดให้พอร์ทดังกล่าวทำงานเป็นดิจิตอลก่อน ด้วยการกำหนดค่าในรีจิสเตอร์ ADCON1 เช่น ADCON1 = 7 เป็นต้น

2. เนื่องจากการแสดงผลของจอ LCD เป็นลักษณะ Static display เนื่องจากมีคอนโทรลเลอร์ของตัวเอง ตัวอักขระยังคงค้างจออยู่จากว่าจะมีการส่งมาแสดงใหม่ ดังนั้นในการเขียนโปรแกรมแสดงผล ถ้าเป็นข้อความนิ่ง ๆ พยายามหลีกเลี่ยงการเขียนโปรแกรมวนมาแสดงซ้ำ ๆ ซึ่งอาจทำให้ตัวข้อความสั่นพลิ้ว และเสียเวลาการทำงานของโปรแกรมโดยไม่จำเป็น

3. การต่อไฟผิดขั้วจะทำให้จอ LCD เสียหายได้ จึงควรระมัดระวัง เนื่องมีราคาค่อนข้างแพง

วงจรทดลองตาม Experiment 13



โปรแกรมคำสั่งที่ 1 ส่งข้อความแสดงใน Line 1 และ Line 2

```
DEFINE LCD_DREG PORTD
DEFINE LCD_DBIT 4
DEFINE LCD_RSREG PORTE
DEFINE LCD_RSBIT 2
DEFINE LCD_EREG PORTD
DEFINE LCD_EBIT 1
'----------------------
adcon1 = 7
lcdout $fe,1,"Hello.."
lcdout $fe,$c0,"World !"
end
```

โปรแกรมคำสั่งที่ 2 ส่งข้อความ และค่าตัวแปรเป็นเลขฐานสิบ

```
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
'----------------------
adcon1 = 7
temp   = 25
lcdout $fe,1,"Temp = "
lcdout $fe,$c0,dec temp,$fe,$c4,"C"
end
```

Experiment 14 การเขียนโปรแกรมประยุกต์การแสดงผลผ่านจอแอลซีดี(LCD Display)

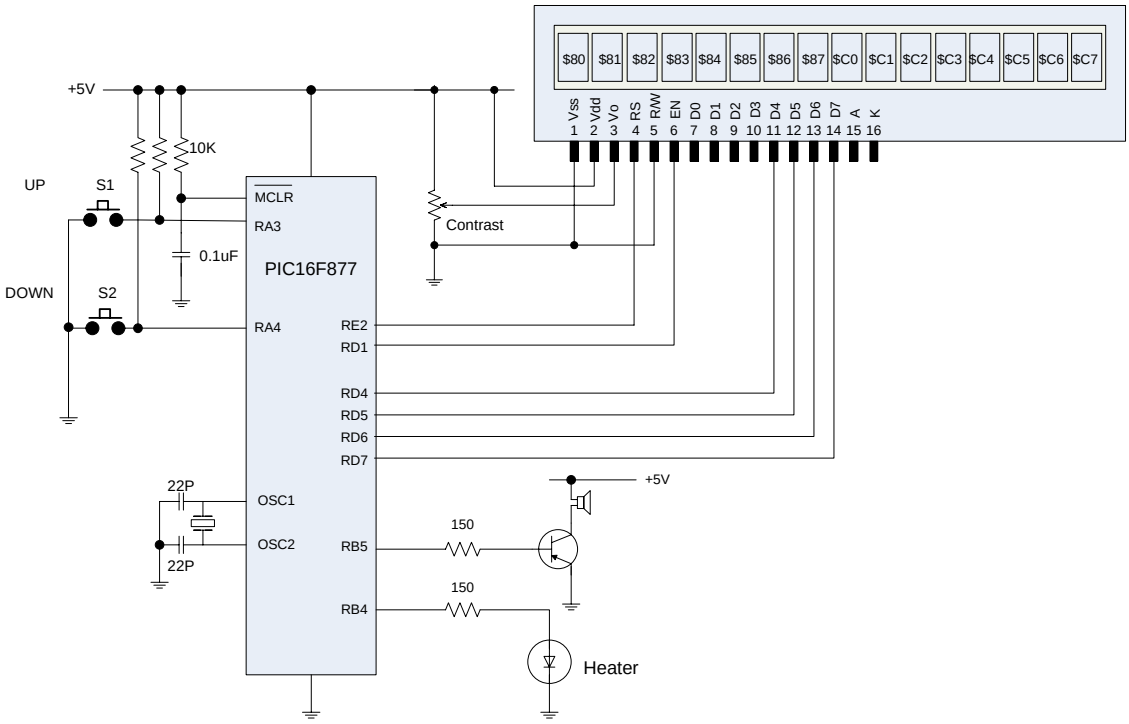
จุดประสงค์ เพื่อศึกษาการเขียนโปรแกรมในการประยุกต์การแสดงผลทางจอ LCD สำหรับใช้งานในระบบการปรับควบคุมอุณหภูมิ หรือตู้อบความร้อน

ทฤษฎีพื้นฐาน

ระบบการทำงานของตู้อบความร้อนประกอบด้วย 4 ส่วน คือ ส่วนตัวตู้อบมีประตูปิด – เปิด มีฉนวนกันความร้อนไม่ให้ระบายออกนอกตัวตู้ ส่วนที่ให้ความร้อน เป็นไส้ขดลวดความร้อน(Heater) เมื่อปล่อยไฟเข้าจะทำให้ขดลวดร้อนและแผ่รังสีความร้อนออกมาเต็มที่ หากจะควบคุมอุณหภูมิของตู้อบต้องคอยปิด - เปิดไฟให้กับขดลวดความร้อนนี้ตลอดเวลา ส่วนควบคุมประกอบด้วยวงจรไมโครคอนโทรลเลอร์ และวงจรขับไส้ขดลวดความร้อนเรียกว่า Solid State Relay ส่วนการตรวจจับอุณหภูมิ ซึ่งมีอยู่หลายแบบได้แก่ Thermocoupleขดลวดต้านทาน(RTD) และอุปกรณ์สารกึ่งตัวนำ เช่น ไอซี DS1820 เป็นต้น และส่วนแสดงผลเป็นส่วนที่ทำให้ทราบสถานะของการควบคุม โดยจะแสดงอุณหภูมิที่ปรับตั้ง อุณหภูมิจริงในตู้อบ ส่วนนี้ได้รวมปุ่มสวิทช์ควบคุมปรับตั้งอุณหภูมิ และส่วนปิดเปิดการทำงานของตู้อบด้วย

สำหรับในใบงานนี้จะแบ่งส่วนการควบคุม และการแสดงผลนำมาศึกษาปฏิบัติก่อน เพื่อให้เข้าใจในด้านการเขียนโปแกรมประยุกต์ใช้งาน LCD การปรับตั้งอุณหภูมิ และการแสดงผล

วงจรทดลองตาม Experiment 14



การทำงานของโปรแกรมคำสั่ง

เมื่อเปิดเครื่องอุณหภูมิจะถูกตั้งค่าเริ่มตนไว้ 25 องศาเซลเซียส สวิทช์ S1 และ S2 ใช้ปรับค่าขึ้น-ลง

ถ้าปรับค่าเกินกว่า 25 หลอด Heater จะดับ และปรับน้อยกว่า 25 หลอดจะติด ค่าอุณหภูมิจะแสดงที่จอ LCD การปรับแต่ละครั้งจะมีเสียงคลิกการกดปุ่มสวิทช์ อุณหภูมิต่ำสุด 20 องศา และสูงสุด 80 องศา

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
S1        VAR          PORTA.3
S2        VAR          PORTA.4
spk       var portb.5
heater  var portb.4
temp    var byte
        TRISA     =    %111111
        ADCON1 = 7
        temp   = 25
        lcdout $fe,1,"Temp = "
        lcdout $fe,$c0,dec temp,$fe,$c4,"C"
LOOP:     IF  (S1 =  0) AND (S2 = 1)  THEN
            pause 50
            temp  =  temp+1
            gosub click
            gosub display
            gosub action
            if temp = 80 then temp = 79
        idle1: if (S1 =  0) AND (S2 = 1)  THEN idle1
        endif
                            IF  (S1 =  1) AND (S2 = 0)  THEN
                              pause 50
            if temp = 20  then  temp=21
            temp  =  temp - 1
            gosub click
            gosub display
            gosub action
        idle2: if (S1 =  1) AND (S2 = 0)  THEN idle2
        endif
                            PAUSE  50
                            GOTO     LOOP
        END
'----------------- End of Main Program -----------
'
'------- Subroutine  Start  Here ------------------
click:
        freqout spk,5,2000
        return

display:
        lcdout $fe,$c0,dec temp
        return

action:
        if temp > 25 then
           low  heater
        else
           high  heater
        endif
        return
'
'------- End of Subroutine ------------------------
```

<u>มอบหมายงาน</u>

เมื่อศึกษาและปฏิบัติเข้าใจดีแล้วให้ปรับแก้โปรแกรมให้สามารถกดปุ่มปรับอุณหภูมิโดยกดแช่ให้เพิ่มและลดได้อย่างต่อเนื่อง แทนการกดได้ทีละครั้ง

Experiment 15 การเขียนโปรแกรมแปลงสัญญาณอนาล็อกเป็นดิจิตอล(A/D) แบบ 8 บิท

จุดประสงค์ เพื่อศึกษาการเขียนโปรแกรมในการแปลงสัญญาณอนาล็อกเป็นดิจิตอล(A/D) และแสดงผลทางจอ LCD

ทฤษฎีพื้นฐาน

การแปลงสัญญาณอนาล็อกเป็นดิจิตอลเป็นวงจรสำคัญในการนำไปใช้ในการตรวจจับ(Sensor) สัญญาณที่เป็นระดับเช่น ความดัน ระดับของเหลว อุณหภูมิ เป็นต้น ในการตรวจจับสัญญาณดังกล่าว อันดับแรกต้องมีวงจรแปลงสัญญาณเหล่านี้เป็นสัญญาณทางไฟฟ้าก่อน จากนั้นจึงได้แปลงมาเป็นสัญญาณดิจิตอล ระดับสัญญาณทางไฟฟ้าโดยปกติจะมีค่ามาตรฐานอยู่ที่ 0 – 5 Vdc หรือ 0 – 2.5Vdc ปัจจุบันผู้ผลิตชิพไมโครคอนโทรลเลอร์จะฝังวงจร A/D เข้าไปไว้ในตัวชิพ โดยสามารถป้อนสัญญาณอนาล็อกมาตรฐานเข้าที่ขา I/O ได้โดยตรง โดยมีรีจิสเตอร์ และคำสั่งที่เป็นฟังก์ชั่นการทำงานไว้เบ็ดเสร็จ และถือเป็นขีดความสามารถในการแข่งขันในการพัฒนาชิพ MCU อย่างหนึ่ง

สำหรับชิพ MCU ของไมโครชิพเบอร์ PIC16F877 รวมถึงเบอร์อื่น ๆที่มี 40 ขา จะมีขา I/O ที่รับสัญญาณ A/D ได้ถึง 8 ช่องโดยมีพอร์ท A จำนวน 5 ขาและ พอร์ท E จำนวน 3 ขา วงจร A/D ที่อยู่ในตัวชิพจะเป็นแบบมัลติเพล็กซ์ ความเร็วในการแปลงสัญญาณ ต่อช่องประมาณ 18 ไมโครวินาทีที่ความละเอียด 8 บิท สำหรับความละเอียดสามารถกำหนดได้ 2 ระดับคือ ระดับ 8 บิท และระดับ 10 บิท โดยระดับ 8 บิทจะแปลงสัญญาณอนาล็อก 0 – 5 V มาเป็นสัญญาณดิจิตอลได้ 256 ค่า และระดับ 10 บิทจะแปลงได้ 1096 ค่า ซึ่งมีความละเอียดกว่ามาก แต่จะใช้เวลาในการแปลงมากกว่า 8 บิท ในการแปลงค่า A/D จะมีรีจิสเตอร์ที่ทำหน้าที่อยู่ 4 ตัว ได้แก่

ADCON0 เป็นรีจิสเตอร์ควบคุมการทำงาน A/D ตัวที่ 1

ADCON1 เป็นรีจิสเตอร์ควบคุมการทำงาน A/D ตัวที่ 2

ADRESH เป็นรีจิสเตอร์เก็บค่าผลลัพธ์การแปลง A/D ในไบท์สูง

ADRESL เป็นรีจิสเตอร์เก็บค่าผลลัพธ์การแปลง A/D ในไบท์ต่ำ

สำหรับโมดูล A/D ของ MCU เบอร์นี้ยังคงทำงานได้แม้จะอยู่ใน Sleep Mode และยังจะเชื่อต่อกับระบบการ Interrupt

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on MCLR, WDT
0Bh,8Bh, 10Bh,18Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000u
0Ch	PIR1	PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
8Ch	PIE1	PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
1Eh	ADRESH	A/D Result Register High Byte								xxxx xxxx	uuuu uuuu
9Eh	ADRESL	A/D Result Register Low Byte								xxxx xxxx	uuuu uuuu
1Fh	ADCON0	ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/$\overline{\text{DONE}}$	—	ADON	0000 00-0	0000 00-0
9Fh	ADCON1	ADFM	—	—	—	PCFG3	PCFG2	PCFG1	PCFG0	--0- 0000	--0- 0000
85h	TRISA	—	—	PORTA Data Direction Register						--11 1111	--11 1111
05h	PORTA	—	—	PORTA Data Latch when written: PORTA pins when read						--0x 0000	--0u 0000
89h(1)	TRISE	IBF	OBF	IBOV	PSPMODE	—	PORTE Data Direction bits			0000 -111	0000 -111
09h(1)	PORTE	—	—	—	—	—	RE2	RE1	RE0	---- -xxx	---- -uuu

Legend: x = unknown, u = unchanged, - = unimplemented, read as '0'. Shaded cells are not used for A/D conversion.
Note 1: These registers/bits are not available on the 28-pin devices.

ตารางที่ 1 แสดงค่ารีจิสเตอร์ที่เกี่ยวข้องกับการทำงาน A/D และค่าเมื่อเกิดสภาวะเริ่มทำงาน

ตารางที่ 2 : **ADCON0 REGISTER (ADDRESS: 1Fh)**

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0
ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/$\overline{\text{DONE}}$	—	ADON
bit 7							bit 0

bit 7-6 **ADCS1:ADCS0:** A/D Conversion Clock Select bits
00 = Fosc/2
01 = Fosc/8
10 = Fosc/32
11 = Frc (clock derived from the internal A/D module RC oscillator)

bit 5-3 **CHS2:CHS0**: Analog Channel Select bits
000 = channel 0, (RA0/AN0)
001 = channel 1, (RA1/AN1)
010 = channel 2, (RA2/AN2)
011 = channel 3, (RA3/AN3)
100 = channel 4, (RA5/AN4)
101 = channel 5, (RE0/AN5)(1)
110 = channel 6, (RE1/AN6)(1)
111 = channel 7, (RE2/AN7)(1)

bit 2 **GO/$\overline{\text{DONE}}$:** A/D Conversion Status bit
If ADON = 1:
1 = A/D conversion in progress (setting this bit starts the A/D conversion)
0 = A/D conversion not in progress (this bit is automatically cleared by hardware when the A/D conversion is complete)

bit 1 **Unimplemented**: Read as '0'

bit 0 **ADON**: A/D On bit
1 = A/D converter module is operating
0 = A/D converter module is shut-off and consumes no operating current

Note 1: These channels are not available on PIC16F873/876 devices.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

ตารางที่ 3 **ADCON1 REGISTER (ADDRESS 9Fh)**

U-0	U-0	R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	—	—	—	PCFG3	PCFG2	PCFG1	PCFG0
bit 7							bit 0

bit 7 **ADFM:** A/D Result Format Select bit
1 = Right justified. 6 Most Significant bits of ADRESH are read as '0'.
0 = Left justified. 6 Least Significant bits of ADRESL are read as '0'.

bit 6-4 **Unimplemented:** Read as '0'

bit 3-0 **PCFG3:PCFG0**: A/D Port Configuration Control bits:

PCFG3: PCFG0	AN7(1) RE2	AN6(1) RE1	AN5(1) RE0	AN4 RA5	AN3 RA3	AN2 RA2	AN1 RA1	AN0 RA0	VREF+	VREF-	CHAN/ Refs(2)
0000	A	A	A	A	A	A	A	A	VDD	Vss	8/0
0001	A	A	A	A	VREF+	A	A	A	RA3	Vss	7/1
0010	D	D	D	A	A	A	A	A	VDD	Vss	5/0
0011	D	D	D	A	VREF+	A	A	A	RA3	Vss	4/1
0100	D	D	D	D	A	D	A	A	VDD	Vss	3/0
0101	D	D	D	D	VREF+	D	A	A	RA3	Vss	2/1
011x	D	D	D	D	D	D	D	D	VDD	Vss	0/0
1000	A	A	A	A	VREF+	VREF-	A	A	RA3	RA2	6/2
1001	D	D	A	A	A	A	A	A	VDD	Vss	6/0
1010	D	D	A	A	VREF+	A	A	A	RA3	Vss	5/1
1011	D	D	A	A	VREF+	VREF-	A	A	RA3	RA2	4/2
1100	D	D	D	A	VREF+	VREF-	A	A	RA3	RA2	3/2
1101	D	D	D	D	VREF+	VREF-	A	A	RA3	RA2	2/2
1110	D	D	D	D	D	D	D	A	VDD	Vss	1/0
1111	D	D	D	D	VREF+	VREF-	D	A	RA3	RA2	1/2

A = Analog input D = Digital I/O

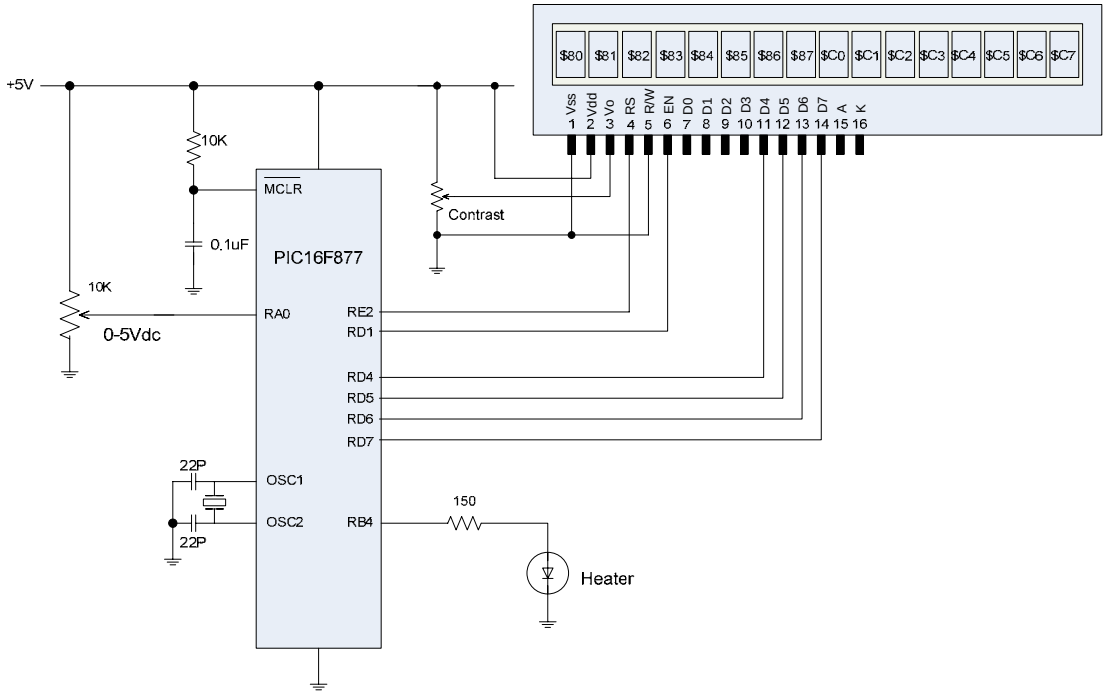
ค่าความเร็วในการแปลง A/D สามารถคำนวณได้จากสมการต่อไปนี้

$$
\begin{aligned}
T_{ACQ} &= \text{Amplifier Settling Time} + \\
&\quad \text{Hold Capacitor Charging Time} + \\
&\quad \text{Temperature Coefficient} \\
&= T_{AMP} + T_C + T_{COFF} \\
&= 2\mu s + T_C + [(\text{Temperature} - 25^\circ C)(0.05\mu s/^\circ C)] \\
T_C &= C_{HOLD}(R_{IC} + R_{SS} + R_S)\ \ln(1/2047) \\
&= -120pF\,(1k\Omega + 7k\Omega + 10k\Omega)\ \ln(0.0004885) \\
&= 16.47\mu s \\
T_{ACQ} &= 2\mu s + 16.47\mu s + [(50^\circ C - 25^\circ C)(0.05\mu s/^\circ C) \\
&= 19.72\mu s
\end{aligned}
$$

ในการออกแบบวงจรป้อนสัญญาณ A/D จะต้องกำหนดให้มีค่าอิมพีแด้นซ์ (Input impedance) ประมาณ 10 กิโลโอห์ม ขั้นตอนในการแปลงสัญญาณ A/D สำหรับในภาษาแอสเซมบลี้จะมีความซับซ้อนบ้าง รายละเอียดให้ศึกษาเพิ่มเติมใน Data sheet

สำหรับการใช้ภาษา Pic Basic Pro Compiler จะมีรีจิสเตอร์ที่เกี่ยวข้องเพียงตัวเดียวคือ ADCON1 ซึ่งจะเป็นตัวกำหนดว่า I/O ขาใดของพอร์ท A และ E ที่จะรับสัญญาณดิจิตอล หรืออนาล็อก และจะจัดเรียงบิทชิดไปทางซ้าย หรือชิดไปทางขวา(กรณีการแปลง A/D แบบ 10 บิท)

วงจรทดลองตาม Experiment 15



คำสั่งที่ใช้ในการแปลง A/D คือ ADCIN หมายเลขช่อง , ตัวแปรที่เก็บค่าผลลัพธ์

รายละเอียดให้ศึกษาเพิ่มเติมในคู่มือการใช้โปรแกรม Pic Basic Pro Compiler

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
heater   var portb.4
temp     var byte
temp1    var byte
        TRISA      =    %111111
        ADCON1 =     0
'----------------  main program -----------------
        adcin 0,temp
        adcon1 = 7
        lcdout $fe,1,"Temp = "
        lcdout $fe,$c0,dec temp,$fe,$c4,"C"
'
LOOP:   gosub get_temp
             if temp <> temp1 then
                  gosub display
                  gosub action
                 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
'
action:
        if temp > 25 then
           low  heater
        else
           high  heater
        endif
        return
'
'------- End of Subroutine ------------------------
```

Experiment 16 การเขียนโปรแกรมแปลงสัญญาณอนาล็อกเป็นดิจิตอล(A/D) แบบ 10 บิท

จุดประสงค์ เพื่อศึกษาการเขียนโปรแกรมในการแปลงสัญญาณอนาล็อกเป็นดิจิตอล(A/D) ที่ความละเอียด 10 บิทและแสดงผลทางจอ LCD

ทฤษฎีพื้นฐาน

ในการแปลงสัญญาณ A/D แบบ 10 บิทจะมีข้อแตกต่างกับการแปลงแบบ 8 บิท คือ

1. ค่าสัญญาณดิจิตอลที่แปลงได้จะมากกว่า 3 หลัก(เกินกว่า 256) ดังนั้นการกำหนดตัวแปรที่ใช้เก็บค่าต้องมีขนาด Word (16 bit) เพื่อรองรับค่า A/D ค่า 0 - 1096

2. ค่าที่แปลงได้จะถูกนำเก็บพักไว้ที่รีจิสเตอร์ 2 ตัว คือ ADRESH และ ADRESL ดังนั้นจะต้องมีการจัดเรียงให้ชิดขอบขวา (Right Justify) ก่อน โดยจะต้องกำหนดบิทที่ 7 ของรีจิสเตอร์ ADCON1 ดังนี้ คือ

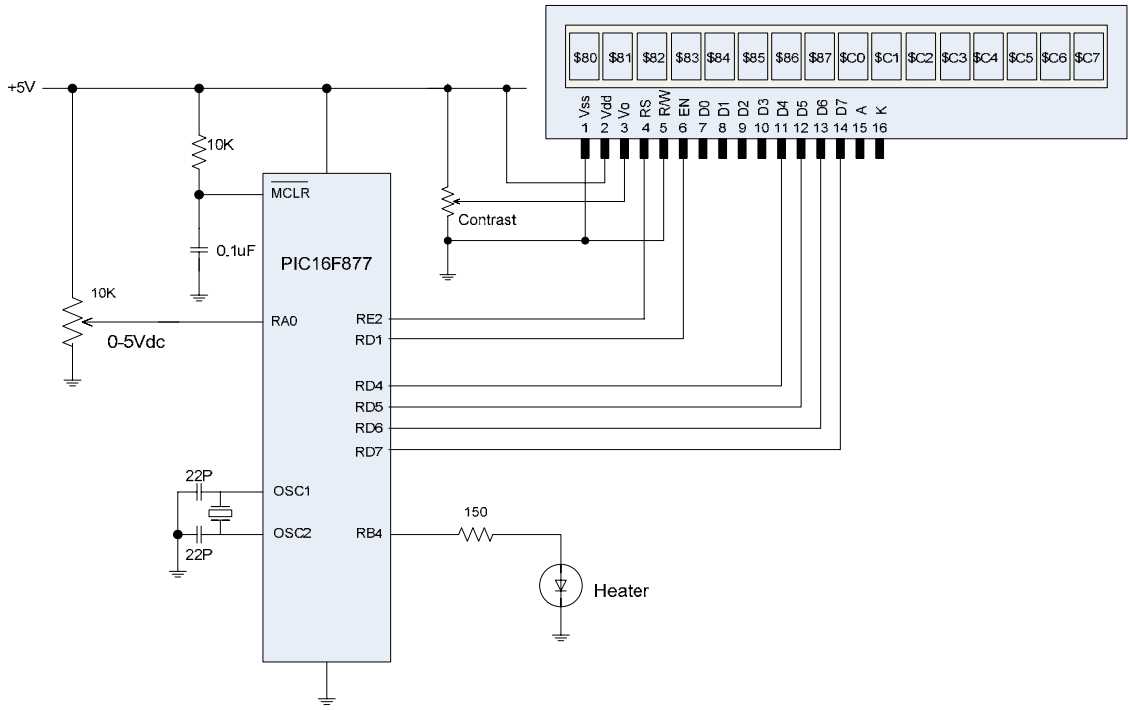
ADCON1 .7 = 1

หรือ ADCON1 = %10000000

3. ต้องนิยามการแปลง A/D แบบ 10 บิท คือ DEFINE ADC_BITS 10 ไว้ที่หัวของโปรแกรม

4. ในการแสดงผลผ่าน LED 7-Segment หรือ LCD จะต้องกำหนดเนื้อที่การแสดงค่าตัวเลขไว้อย่างน้อย 4 หลัก เนื่องจากค่าดิจิตอลที่แปลงได้สูงสุดเกินหลักพันในฐานสิบ

วงจรทดลองตาม Experiment 16



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
DEFINE ADC_BITS 10
‘
heater   var portb.4
temp     var word
temp1    var word
        TRISA       =   %111111
        ADCON1    =  %10000000
'----------------  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 action
                  temp1 = temp
              endif
              PAUSE  50
              GOTO    LOOP
        END
'--------------- End of Main Program ---------------
'
'------- Subroutine  Start  Here ------------------
'
get_temp:
         adcon1 = %10000000
         adcin 0,temp
         pause 50
         return
'
display:
        adcon1 = 7
        lcdout $fe,$c0,"     "
        lcdout $fe,$c0,dec temp
        return
'
action:
        if temp > 25 then
           low  heater
        else
           high  heater
        endif
        return
'
'------- End of Subroutine ------------------------
```

Experiment 17 การเขียนโปรแกรมแปลงสัญญาณ A/D แบบ 8 บิทแบบหลายช่อง

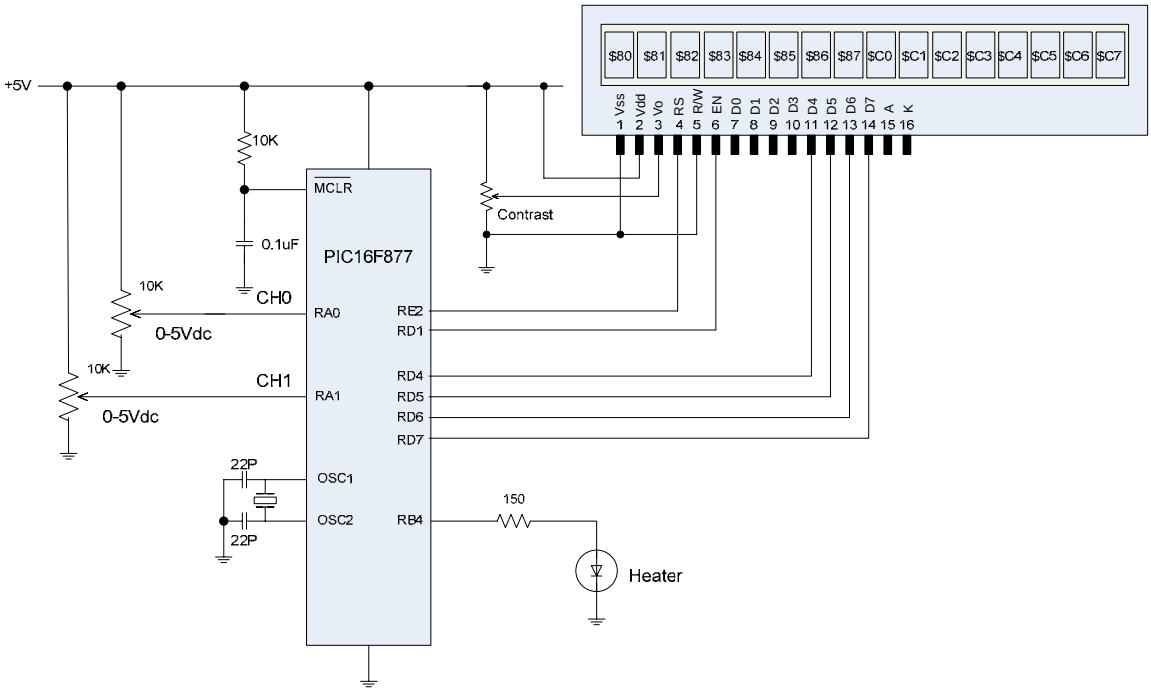
จุดประสงค์ เพื่อศึกษาการเขียนโปรแกรมในการแปลงสัญญาณอนาล็อกเป็นดิจิตอล(A/D) แบบทีละหลายช่องและนำมาแสดงผลพร้อมกัน

ทฤษฎีพื้นฐาน

เทคนิคการแปลงสัญญาณ A/D ทีละหลายช่อง เพื่อนำมาประมวลผลเป็นพื้นฐานสำคัญในการนำ MCU ไปประยุกต์ใช้งาน เช่น ในระบบตรวจจับสัญญาณในระบบควบคุมการทำงานของเครื่องยนต์แบบหัวฉีดซึ่งมีเซ็นเซอร์มากมายที่ต้องมาประมวลผล ระบบตรวจจับและบันทึกค่าตัวแปรในกระบวนการผลิตในอุตสาหกรรม (Data Logging) สัญญาณดังกล่าวเหล่านี้ส่วนมากเป็นสัญญาณอนาล็อก จะต้องถูกแปลงเป็นดิจิตอลก่อนนำไปใช้งาน ตัวแปลง A/D ส่วนมากจะใช้ไมโครคอนโทรลเลอร์เนื่องจากมีความเชื่อถือในด้านเสถียรภาพการทำงาน ขั้นตอนในการเขียนโปรแกรมคือ ใช้คำสั่งแปลง A/D ให้ครบทุกช่องเรียงกัน เพื่อเก็บไว้ในตัวแปร จากนั้นจึงนำค่าตัวแปรเหล่านั้นไปใช้งาน บันทึกหรือแสดงผลต่อไป

ในการเลือกใช้ขา I/O รับสัญญาณอนาล็อก ของไมโครคอนโทรลเลอร์ PIC 16F877 ที่ใช้ปฏิบัติในบอร์ดทดลองสามารถใช้ได้ทุกขาของพอร์ท A ยกเว้นขา RA4 และทุกขาของพอร์ท E ยกเว้นขา RE2 เนื่องจากถูกใช้งานกับจอ LCD การกำหนดให้ขา I/O ใดเป็นดิจิตอลหรืออนาล็อกนั้น ให้ดูตารางที่ 3 ของใบงานที่ 15

วงจรทดลองตาม Experiment 17



โปรแกรมคำสั่ง

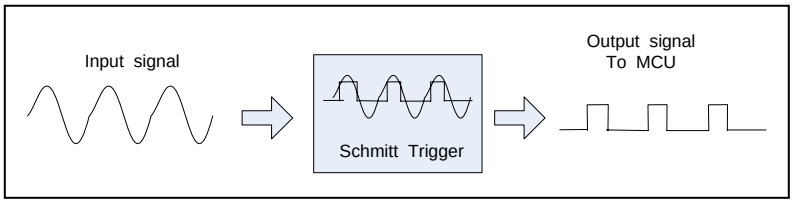
```
DEFINE LCD_DREG PORTD
DEFINE LCD_DBIT 4
DEFINE LCD_RSREG PORTE
DEFINE LCD_RSBIT 2
DEFINE LCD_EREG PORTD
DEFINE LCD_EBIT 1
'
heater   var portb.4
temp     var byte
temp1    var byte
level    var byte
level1   var byte
        TRISA       = %111111
        ADCON1   = 0
'----------------  main program -----------------
        adcin 0,temp
        adcin 1,level
        adcon1 = 7
        lcdout $fe,1,"Temp="
        lcdout $fe,$85,dec temp
        lcdout $fe,$c0,"Lev="
        lcdout $fe,$c4,dec level
'
LOOP:   gosub get_temp
        if (temp <> temp1) or (level <> level1) then
                gosub display
                gosub action
                temp1 = temp
                level1 = level
        endif
        PAUSE  50
        GOTO  LOOP
        END
'-------------- End of Main Program --------------
'
'------- Subroutine  Start  Here -----------------
'
get_temp:
         adcon1 = 0
         adcin 0,temp
         adcin 1,level
         pause 50
         return
'
display:
        adcon1 = 7
        lcdout $fe,$85,"   "
        lcdout $fe,$85,dec temp
        lcdout $fe,$c4,"   "
        lcdout $fe,$c4,dec level
        return
'
action:
        if  temp  =  level    then
           low  heater
        else
           high  heater
        endif
        return
'
'------- End of Subroutine ------------------------
```

Experiment 18 การเขียนโปรแกรมเพื่อนับสัญญาณความถี่พัลส์

จุดประสงค์ เพื่อศึกษาการเขียนโปรแกรมประยุกต์ใช้คำสั่งเพื่อนับจำนวนความถี่พัลส์ที่มาปรากฏที่ขา I/O ของไมโครคอนโทรลเลอร์

ทฤษฎีพื้นฐาน

การนับจำนวนความถี่พัลส์ที่มาปรากฏที่ขา I/O ของไมโครคอนโทรลเลอร์ เป็นพื้นฐานที่สำคัญในการนำไปประยุกต์ใช้ออกแบบเครื่องวัดความถี่ต่าง ๆ เช่น ความเร็วรอบเครื่องยนต์ และการหมุนของเครื่องจักรต่าง ๆ หลักการคือ จะต้องเขียนโปรแกรมให้ไมโครคอนโทรลเลอร์เปิดประตูรับลูกพัลส์เข้าที่ขา I/O ที่กำหนดและนับสะสมไว้และปิดประตูเมื่อถึงกำหนดคาบเวลา นำค่าที่นับได้เก็บไว้ในตัวแปร และนำไปประมวลผลต่อไป ซึ่งเราจะได้ค่าจำนวนลูกพัลส์ ต่อคาบเวลา หรือเรียกว่า ความถี่นั่นเอง ลักษณะของลูกพัลส์ที่จะป้อนเข้าที่ขา I/O ของ MCU จะต้องมีลักษณะเป็นพัลส์เหลี่ยม หากเป็นลักษณะอื่น เช่น Sine wave หรือ Triangle wave หรืออื่น ๆ จะต้องปรับรูปคลื่นเหล่านั้นด้วยวงจร Schmitt trigger เพื่อให้ได้รูปคลื่นสี่เหลี่ยม และต้องให้มีระดับเป็นสัญญาณลอจิกได้ คือ 0 กับ 5 V หรือ 3 V จึงจะป้อนเข้าขา I/O ได้



รูปที่ 1 แสดงการทำงานวงจรปรับสภาพสัญญาณ Schmitt Trigger

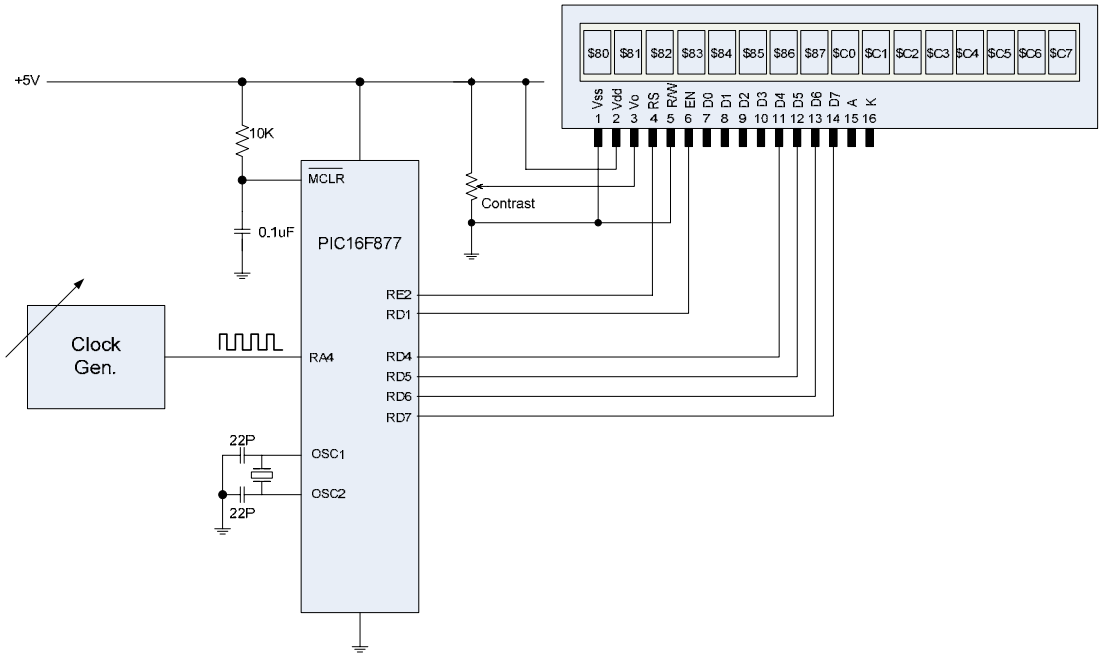
ปัจจุบันวงจร Schmitt Trigger เป็นชิพวงจรรวม หรือ ไอซี ประเภท TTL และ CMOS อยู่หลายเบอร์ (ให้ดูในคู่มือไอซี TTL และ CMOS)

สิ่งที่สำคัญในการนับจำนวนความถี่พัลส์คือ คาบเวลาในการเปิด และปิดประตูจะต้องมีความเร็วที่เพียงพอที่จะไม่ให้เกิดการล้น (Over flow) ซึ่งจะทำให้เกิดการนับผิดพลาดไม่เป็นค่าความจริง ในคำสั่งของภาษา Pic Basic Pro จะมีไว้รองรับเบ็ดเสร็จ และสามารถจะกำหนดค่าคาบเวลาการปิด-เปิดประตูการนับได้ มีรูปแบบดังนี้ คือ

```
COUNT Pin, Period, Variable
```

ค่า Period มีค่าเป็น มิลิวีนาที (mS) การตรวจจับพัลส์ จะตรวจจับที่ขอบขาขึ้น ความไวในการตรวจจับจะขึ้นอยู่กับความถี่ OSC ของ MCU คือถ้าใช้ OSC = 4 MHz จะมีความไวอยู่ที่ 20 ไมโครวินาที และหากใช้ OSC = 20 MHz จะมีความไวที่ 4 ไมโครวินาที หรืออาจกล่าวความหมายอีกนัยหนึ่ง คือ ถ้า MCU ใช้ OSC = 4 MHz จะสามารถนับความถี่พัลส์ได้สูงสุด 25 KHz และหากใช้ OSC = 20 MHz จะสามารถนับความถี่พัลส์ได้สูงสุด 125 KHz ที่กล่าวมาทั้งหมดนี้เป็นการใช้คำสั่ง COUNT หากต้องการให้ MCUมีความไวสามารถนับความถี่ที่สูงกว่านี้ ต้องใช้วิธีการทำงานของวงจร Timer / Counter ที่อยู่ในตัวชิพ โดยการเขียนด้วยภาษาแอสเซมบลี และต้องป้อนสัญญาณพัลส์เข้าที่ขา RA4 ของ MCU เพียงขาเดียว (รายละเอียดให้ศึกษาการใช้โปรแกรมภาษาแอสเซมบลีของชิพตระกูลไมโครชิพ เรื่อง Timer / Counter

วงจรทดลองตาม Experiment 18



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
c_val  var  word
c_in   var  porta.4
adcon1 = 7
Lcdout $fe, 1
Lcdout "Freq:"
loop:
  count c_in,100,c_val
  c_val = c_val * 10
  pause 200
  Lcdout $fe,$c0,dec c_val
  Pause 100
  Goto loop
  End
```

<u>มอบหมายงาน</u>

เมื่อศึกษาและปฏิบัติเข้าใจดีแล้ว ให้พัฒนาเป็นโปรแกรมสำหรับวัดรอบเครื่องยนต์ต่อไป และวัดความเร็วรถเป็นกิโลเมตร ต่อชั่วโมง

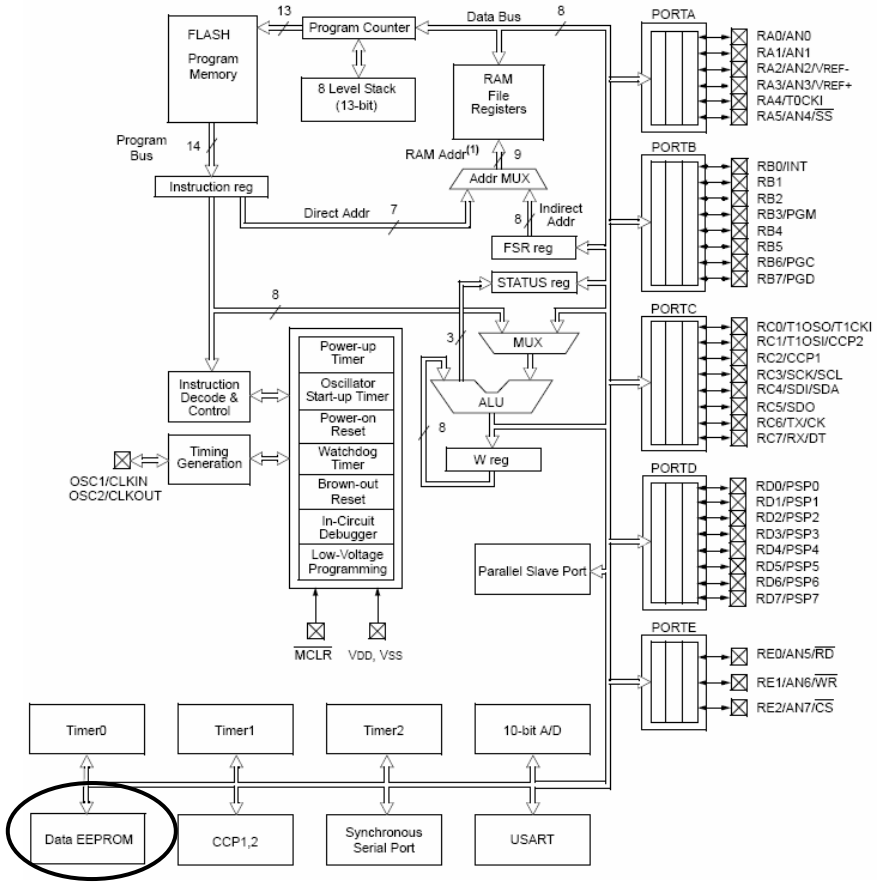
Experiment 19 การเขียนโปรแกรมเพื่อติดต่อกับหน่วยความจำ Data EEPROM ภายในตัว MCU

จุดประสงค์ เพื่อศึกษาการเขียนโปรแกรมประยุกต์ใช้คำสั่งเพื่อบันทึกและอ่านข้อมูลจากหน่วยความจำ Data EEPROM ภายในตัวของไมโครคอนโทรลเลอร์

ทฤษฎีพื้นฐาน

ไมโครคอนโทรลเลอร์ของตระกูลไมโครชิพ PIC16FXXX และ 18FXXX ขึ้นไป ภายในตัวชิพจะมีหน่วยความจำถาวรแบบอ่านและเขียนได้ หรือเรียกว่า Data EEPROM ที่ให้ผู้ใช้สามารถเก็บบันทึกข้อมูลที่จำเป็น เพื่อมาอ่านนำกลับมาใช้ต่อไปได้โดยที่ข้อมูลยังคงสภาพแม้จะไม่มีไฟเลี้ยงวงจรแล้วก็ตาม และยังคงรักษาสภาพได้เกินกว่า 50 ปีโดยสภาพที่ไม่มีไฟเลี้ยง ขนาดของหน่วยความจำแบบนี้จะมีค่ามากน้อยขึ้นอยู่กับเบอร์และรุ่น สำหรับ MCU เบอร์ PIC16F877 ที่ใช้ปฏิบัติจะมีหน่วยความจำแบบนี้ 256 x 8 ไบท์ ซึ่งมากเพียงพอที่จะใช้เก็บรักษาข้อมูลที่จำเป็นจะต้องนำกลับมาใช้ต่อเนื่อง ในการเปิดใช้งานครั้งต่อไป ตัวอย่างการใช้งาน เช่น อุณหภูมิเครื่องปรับอากาศครั้งสุดท้ายที่ปรับไว้ ช่องทีวีครั้งสุดท้ายก่อนปิด ระดับความดังและการปรับแต่งเสียงครั้งสุดท้ายก่อนปิดเครื่อง หรือค่าการปรับตั้งค่าในกระบวนการผลิต เหล่านี้เป็นต้น

Device	Program FLASH	Data Memory	Data EEPROM
PIC16F874	4K	192 Bytes	128 Bytes
PIC16F877	8K	368 Bytes	256 Bytes



รูปที่ 1 แสดงโครงสร้างของไมโครคอนโทรลเลอร์ PIC16F877 และ Data EEPROM ภายใน

คำสั่งภาษา PIC Basic Pro Compiler สำหรับใช้ในการอ่าน และเขียนข้อมูลลงใน Data EEPROM ในขณะโปรแกรมกำลังประมวลผลอยู่นี้ได้แก่

คำสั่ง READ สำหรับอ่านข้อมูลหนึ่งไบท์มาใส่ไว้ในตัวแปรจากตำแหน่งหน่วยความจำที่ระบุ รูปแบบ คือ

READ *Address,Variable*

คำสั่ง WRITE สำหรับเขียนข้อมูลหนึ่งไบท์ เก็บไว้ในตำแหน่งหน่วยความจำที่ระบุ รูปแบบ คือ

READ *Address,Value*

เนื่องคำสั่ง READ และ WRITE จะอ่านและเขียนข้อมูลได้ทีละ 1 ไบท์เท่านั้น ดังนั้นหากเป็นข้อมูลระดับ Word จะต้องแยกเป็นไบท์สูง และไบท์ต่ำก่อนแล้วใช้คำสั่งเขียนบันทึกทีละไบท์ ในการอ่านก็เช่นกัน ต้องอ่านออกมาทีละไบท์แล้วมาต่อกันเป็น Word เช่น

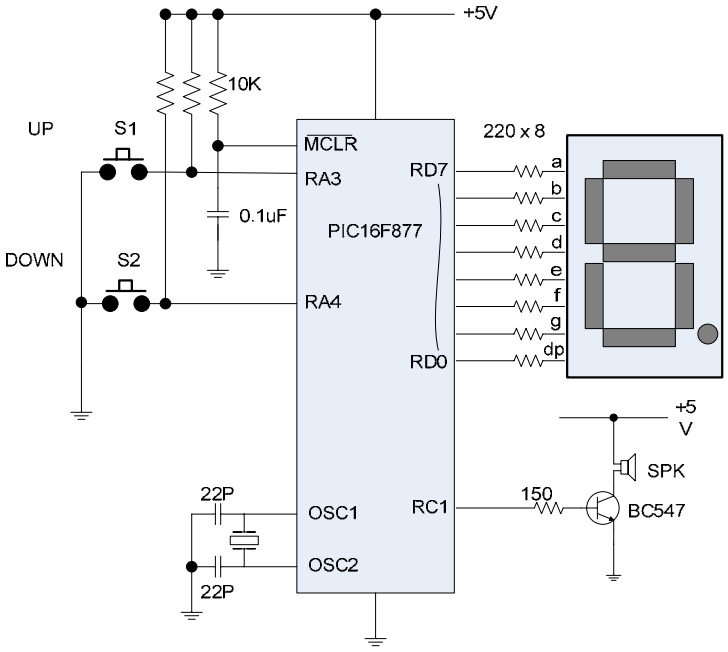
การเขียน

```
temp    Var    Word
WRITE   0,temp.byte0
WRITE   1,temp.byte1
```

การอ่าน

```
w0      Var    byte
w1      Var    byte
w       Var    word
READ    0,w0
READ    1,w1
w.byte0  =  w0
w.byte1  =  w1
```

วงจรทดลองตาม Experiment 19



การทำงานของโปรแกรม

เมื่อเปิดไฟเริ่มทำงานโปรแกรมจะอ่านค่าตัวเลขเริ่มต้นจากครั้งก่อนจากหน่วยความจำ Data EEPROM มากำหนดเป็นค่าตัวแปรแล้วนำไปแสดงผลเป็นตัวเลข และทุกครั้งที่มีการกดปุ่มสวิทช์ S1 และ S2 เพิ่ม หรือลดค่าตัวเลข โปรแกรมจะสร้างเสียงกดคลิก และเขียนค่าตัวเลขใหม่ลงใน Data EEPROM ทุกครั้ง ดังนั้นหากปิดไฟเลี้ยงวงจร และเปิดไฟเริ่มทำงานใหม่ โปรแกรมก็ยังสามารถนำค่าตัวเลขของครั้งสุดท้ายอออกมาแสดงได้ทุกครั้ง

โปรแกรมคำสั่ง

```
S1        var       PORTA.3       'for  count up
S2        var       PORTA.4       'for  count down
num       var byte
disp      var byte
mem       var byte
spk       var portc.1
      TRISA     =   %111111
      TRISD     =   %00000000
      ADCON1  =   7
      read 0,mem
      num = mem
      lookup num,[$c0,$f9,$a4,$b0,$99,_
                $92,$82,$f8,$80,$90],disp
      portd = disp

LOOP:
      if (s1 = 0 and s2 = 1)  then
            pause 50
            gosub click
            num  =  num + 1
            if num > 9 then num = 0
            lookup num,[$c0,$f9,$a4,$b0,$99,_
                              $92,$82,$f8,$80,$90],disp
            portd = disp
             write 0,num
      endif
      if (s1 = 1) and (s2 = 0) then
           pause 50
           gosub click
           if num = 0 then num = 10
           num  =  num - 1
           lookup num,[$c0,$f9,$a4,$b0,$99,_
                             $92,$82,$f8,$80,$90],disp
            portd = disp
           write 0,num
      endif
      pause 300
      GOTO  LOOP
      END
'
'-----------Subroutine-----------
click:
     freqout spk,5,2000
     return
```

มอบหมายงาน

เมื่อศึกษาและปฏิบัติเข้าใจแล้ว ให้ดัดแปลงโปรแกรมให้ปรับค่าขึ้นสูงสุด ค้างที่เลข 9 และต่ำสุดที่ 0

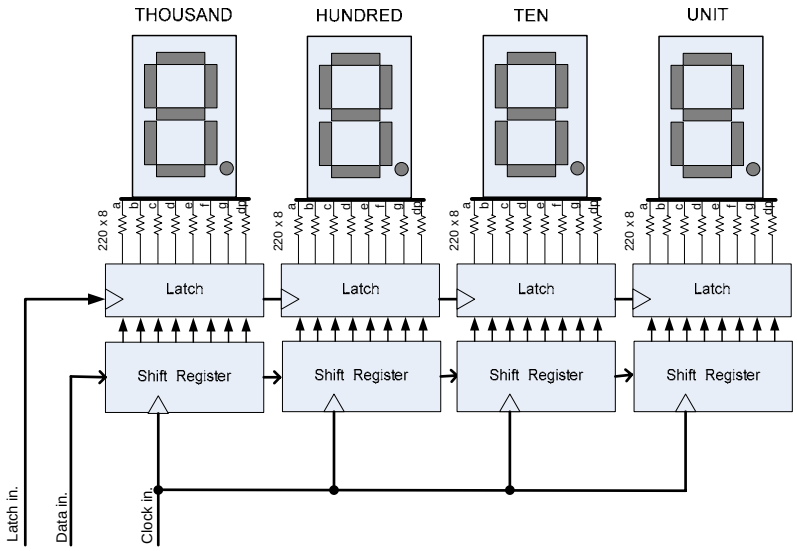
Experiment 20 การเขียนโปรแกรมประยุกต์แสดงผลทาง LED 7-Segment แบบ 4 หลัก

จุดประสงค์ เพื่อศึกษาการเขียนโปรแกรมประยุกต์ในการแสดงผลทาง LED 7-Segment แบบตัวเลขหลายหลัก โดยใช้วิธีส่งข้อมูลตัวเลขแบบอนุกรม ซิงโครนัส

ทฤษฎีพื้นฐาน

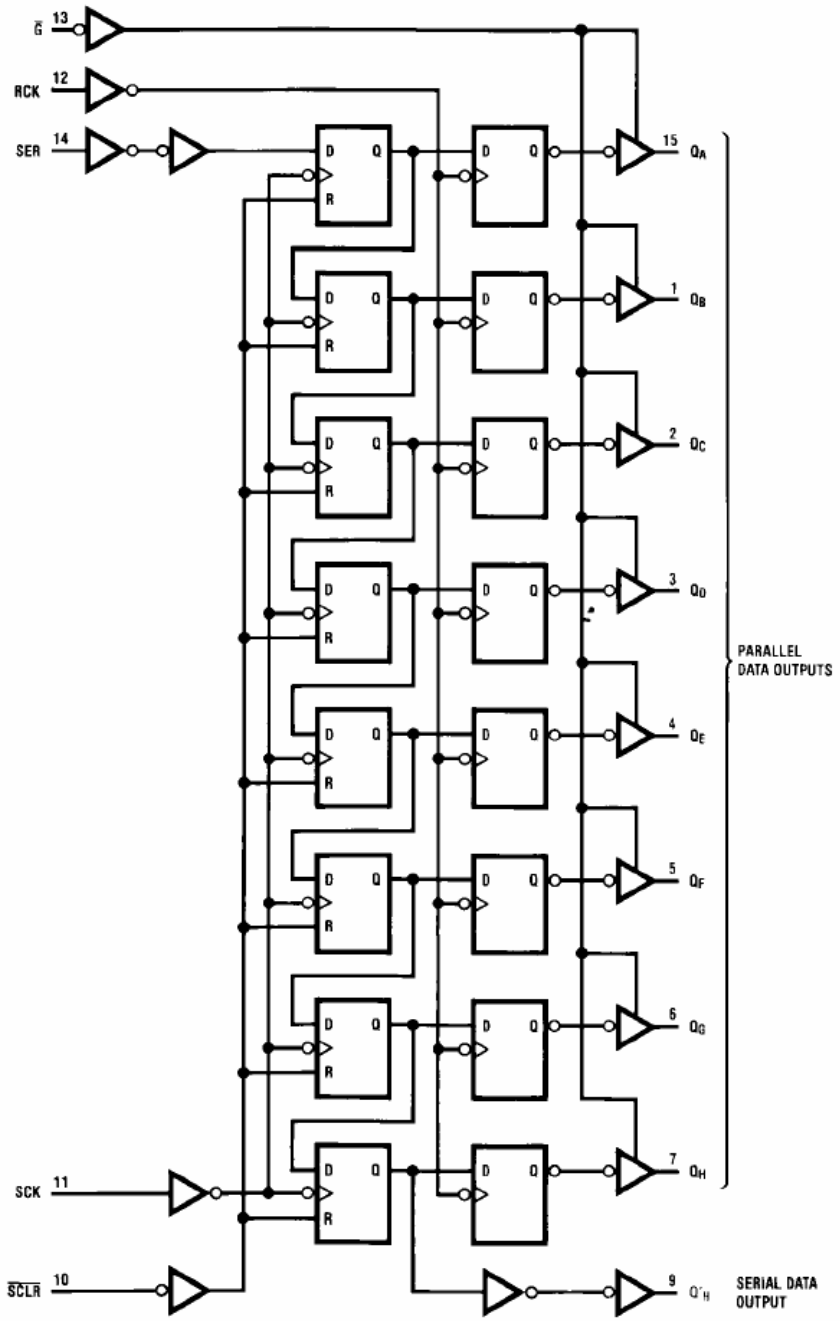
การแสดงผลตัวเลขทาง LED 7-Segment ปัจจุบันยังมีความจำเป็นอยู่เนื่องจากมีข้อดีคือ มีความสว่างสามารถมองเห็นได้ชัดเจนทั้งในเวลากลางวัน และกลางคืน และยังสามารถมองได้ในระยะไกล ตัวอย่างการใช้งานได้แก่ แผงตัวเลขการนับถอยหลังของระบบไฟสัญญาณจราจร ระบบตัวเลขลำดับการจัดลำดับคิวการให้บริการ เครื่องซักผ้าแบบหยอดเหรียญ เป็นต้น

หลักการออกแบบของวงจรแสดงผลตัวเลขทาง LED 7-Segment ที่จะใช้ปฏิบัติในใบงานนี้เป็นแบบแสดงผล 4 หลัก ตัวเลขแต่ละหลักจะถูก Latch ค้างไว้จนกว่าจะมีค่าใหม่มาเปลี่ยน เป็นลักษณะแสดงนิ่ง ๆ หรือเรียกว่า Static Display การแสดงผลแบบนี้มีข้อดี คือ MCU จะไม่ต้องเสียเวลาการทำงานมาแสดงผลซ้ำ ๆ แบบวิธีการมัลติเพล็กซ์เพื่อไม่ให้ตัวเลขกระพริบ การส่งข้อมูลตัวเลขมาแสดงจำใช้เทคนิคการส่งแบบอนุกรมซิงโครนัส ไม่ว่าจะแสดงกี่หลักก็ตามจะใช้สายสัญญาณข้อมูล และสายควบคุมเพียง 3 เส้น คือ สาย Data สาย Clock และสาย Latch ข้อมูล การทำงานของระบบ ตามรูปที่ 1



รูปที่ 1 แสดงการทำงานของระบบ LED 7-Display แบบ 4 หลัก

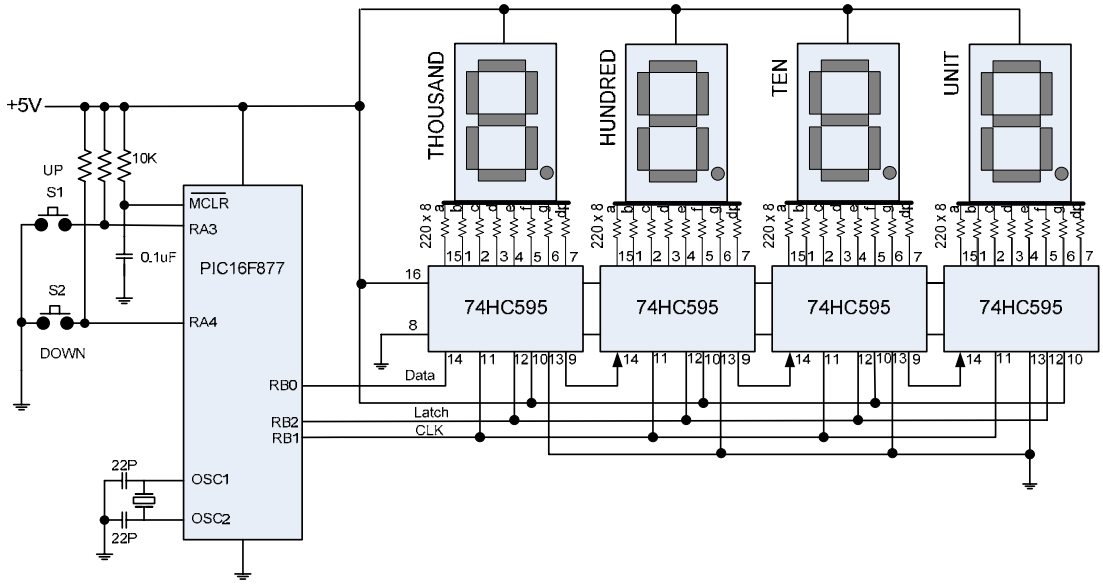
ตามรูปที่ 1 วงจร Shift Register เป็นแบบ Serial in.- Parallel out โดยมีสัญญาณ Clock in เป็นตัวป้อนจังหวะให้เลื่อนขยับข้อมูล ส่วนวงจร Latch เป็นรีจิสเตอร์ชนิด Parallel in - Parallel out โดยมีสัญญาณ Latch in เป็น Clock โหลดข้อมูลจาก Shift Register มาค้างไว้ใน Latch Register ปัจจุบันวงจรรีจิสเตอร์ทั้งสองถูกสร้างไว้ในชิพไอซีเดียวกัน คือ เบอร์ 74HC595 เป็นไอซีดิจิตอล CMOS ตามรูปที่ 2



รูปที่ 2 แสดงโครสร้างการทำงานของไอซีดิจิตอลเบอร์ 74HC595

การต่อวงจรใช้งานมีการต่อพ่วงกัน ไอซี74HC595 หนึ่งตัวจะขับ LED 7-Segment แต่ละหลัก นอกจากต่อไฟบวก ลบเข้าตัวชิพแล้วจะต้องต่อขาต่าง ๆ ตามรูปที่ 3 ดังนี้

วงจรทำลองตาม Experiment 20



การทำงานของโปรแกรม

โปรแกรมที่ใช้ปฏิบัติตามใบงานนี้ เป็นโปรแกรมนับเลขขึ้น – ลง ตั้งแต่ 0000 - 9999 โดยต่อเนื่อง การนับขึ้นต่อเนื่อง หรือลงต่อเนื่องโดยการควบคุมด้วยการกดสวิทช์ S1 และ S2 ข้อมูลตัวเลขการนับจะถูกแยกเป็นหลักหน่วย หลักสิบ หลักร้อย และหลักพัน แล้วเก็บไว้ในตัวแปร 4 ตัวด้วยคำสั่ง DIG จากนั้นจะทยอยส่งออกไปแสดงผลทีละหลัก เรียงกันไปจนครบ 4 หลัก ส่งออกไปแบบอนุกรมที่ขา RB0 ออกไปทีละบิท โดยมีสัญญาณ Clock ที่ขา RB1 เป็นตัวควบคุมจังหวะพร้อมกันไปแบบซิงโครนัส เมื่อครบทั้ง 4 หลักแล้ว จะมีสัญญาณ Latch เพื่อผลักค่าตัวแปรทั้ง 4 ตัวออกไปยังขาของ LED 7-Segment พร้อม ๆ กันแบบขนาน แล้ว

ปิดสัญญาณ Latch เพื่อล็อกค่าค้างไว้ จะทำให้ตัวเลขทั้ง 4 หลักติดสว่างนิ่ง และจะเป็นไปเช่นนี้ เมื่อเปลี่ยนค่าเข้ามาแทน การทำงานกระบวนนี้ทั้งหมด ในภาษา Pic Basic Pro Compiler เพียงคำสั่งเดียว คือคำสั่ง SHIFTOUT การนำคำสั่งนี้มาใช้งานจะต้องเรียกโปรแกรมจัดการเกี่ยวกับการรับ – ส่งข้อมูล ชื่อ MODEDEF.BAS เข้ามาผนวกทำงานด้วยเสมอ โดยใช้คำสั่ง INCLUDE"MODEDEFS.BAS" ไว้ที่ส่วนหัว

โปรแกรมคำสั่งที่ 1

```
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    word
```

มีต่อหน้าถัดไป

```
      TRISA     =    %111111
      ADCON1 = 7
      stat = 1
      num = 0
'
LOOP:
          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
               if num > 9999 then num = 0
               gosub  display
          else
                if num = 0 then num = 10000
                num  =  num - 1
               gosub  display
           endif
           pause 100
          GOTO  LOOP
           END
'------------------------------
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,$90],hun
      digit  =  num dig 3
      lookup digit,[$c0,$f9,$a4,$b0,$99,_
                   $92,$82,$f8,$80,$90],thou
            high latch
            pauseus 10
            ShiftOut dat,clk,1,[unit,ten,hun,thou]
            Pauseus 10
            Low latch
            Pauseus 10
            high latch
        Return
```

โปรแกรมคำสั่งที่ 2

ในโปรแกรมที่ 1 การทำงานขณะที่กำลังนับอยู่ถึงค่าหนึ่ง หากเรากดปุ่ม Reset หรือปิดไฟ ทำให้ค่าที่นับไว้หายไป เมื่อเปิดไฟโปรแกรมจะเริ่มต้นกลับมานับใหม่ ทำให้มีข้อจำกัดในการประยุกต์ใช้งานที่ต้องการความต่อเนื่อง ในโปรแกรมที่ 2 นี้จึงได้นำการใช้คำสั่ง READ และ WRITE หน่วยความจำ Data EEPROM ในตัว MCU มาประยุกต์ใช้ แต่เนื่องจากตัวแปร num ที่ใช้เก็บค่าการนับเป็นตัวแปรขนาด Word หรือ 2 ไบท์ ดังนั้นการเก็บต้องมีการแยกตัวแปร num ออกเป็นระดับไบท์ และต้องเอาไบท์มารวมกันเป็นเพื่อเป็น Word ในการอ่านกลับมาใช้

<u>มอบหมายงาน</u> เมื่อศึกษาและปฏิบัติเข้าใจดีแล้ว ให้ดัดแปลง ไม่ให้มีเลขศูนย์หน้าหลักที่ยังนับไม่ถึง

โปรแกรม 2

```
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    word
b0          var    byte
b1          var    byte
        TRISA      =    %111111
        ADCON1 = 7
        stat = 1
        gosub read_mem
LOOP:
        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
             if num > 9999 then num = 0
            gosub  display
             gosub  write_mem
        else
              if num = 0 then num = 10000
               num  =  num - 1
              gosub  display
              gosub  write_mem
        endif
        pause 80
       GOTO  LOOP
        END
'------------------------------
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,$90],hun
        digit  =  num dig 3
        lookup digit,[$c0,$f9,$a4,$b0,$99,_
                  $92,$82,$f8,$80,$90],thou
        high latch
        pauseus 10
                  ShiftOut dat,clk,1,[unit,ten,hun,thou]
                  Pauseus 10
                  Low latch
                  Pauseus 10
                  high latch
                  Return
'------------------------------
write_mem:
         write 0,num.byte0
         write 1,num.byte1
         return
'------------------------------
read_mem:
        read 0,b0
        read 1,b1
        num.byte0 = b0
        num.byte1 = b1
        return
'------------------------------
```

Experiment 21 การเขียนโปรแกรมประยุกต์แสดงผลทาง LED 7-Segment แบบ 4 หลัก เอนกประสงค์ แบบตั้งจำนวนนับได้

จุดประสงค์ เพื่อศึกษาการเขียนโปรแกรมประยุกต์ในการแสดงผลทาง LED 7-Segment แบบตัวเลขหลายหลัก โดยใช้วิธีส่งข้อมูลตัวเลขแบบอนุกรม ซิงโครนัสที่สามารถตั้งจำนวนนับได้

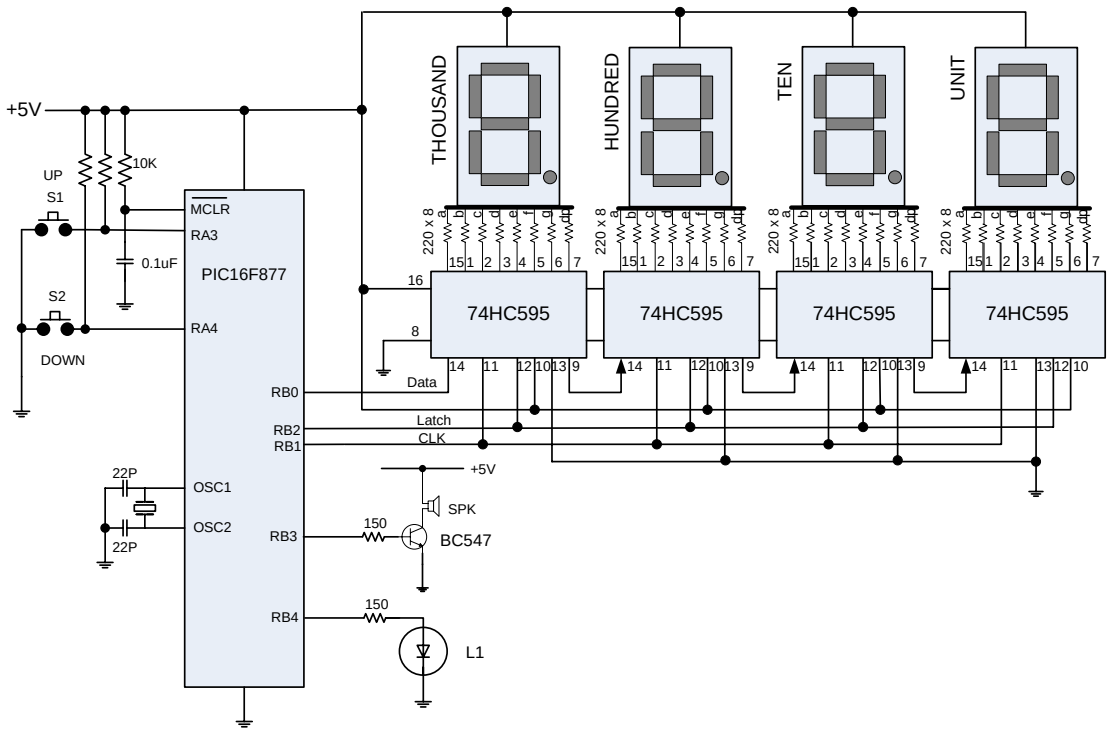
ทฤษฎีพื้นฐาน

โปรแกรมการนับตามใบงานทดลองที่ 20 ทั้งโปรแกรมที่ 1 และ 2 นั้นเป็นเพียงหลักการเรียนรู้ แต่ยังไม่สามารถนำมาประยุกต์ใช้ในงานจริงได้ เนื่องจากยังขาดในส่วนที่จะให้ผู้ใช้สามารถตั้งจำนวนการนับได้ ดังนั้น ในใบงานทดลองนี้จะต้องมาฝึกการเรียนรู้วิธีการเขียนโปรแกรมในส่วนที่สามารให้บริการตั้งค่าจำนวนการนับไว้ล่วงหน้าได้ และสามารถควบคุมการหยุดนับชั่วคราวได้ด้วย

การทำงานของโปรแกรม

เริ่มแรกหากผู้ใช้ยังไม่ต้องการจะปรับตั้งจำนวนนับใหม่ โปรแกรมจะนำข้อมูลการนับที่ปรับตั้งไว้เดิมมาทำงานต่อไป หากผู้ใช้ต้องการจะปรับตั้งการนับใหม่ จะต้องกดสวิทช์ทั้ง S1 และ S2 พร้อม ๆกันเป็นเวลา 3 วินาที จะทำให้โปรแกรมเข้าสู่งานบริการ ปรับตั้งการนับใหม่ จากนั้น S1 และ S2 จะทำหน้าที่กดตั้งจำนวนนับเมื่อตั้งจำนวนได้แล้ว ต้องการออกมาสู่การทำงานนับปกติโดยการกด S1 และ S2 พร้อม ๆกันเป็นเวลา 3 วินาที เมื่อออกสู่การนับปกติ จะนับไปเรื่อย ๆ จนถึงจำนวนที่ตั้งจะหยุดค้างไว้ และส่งเอ้าพุทออกมาเป็นหลอด LED ติดสว่าง

วงจรทดลองตาม Experiment 21



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
spk         var    portb.3
L1          var    portb.4
stat        var    bit
unit        var    byte
ten         var    byte
hun         var    byte
thou        var    byte
digit       var    byte
num         var    word
set         var    word
b0          var    byte
b1          var    byte
      TRISA     =    %111111
      portb     =    %00000000
      ADCON1 = 7
      low  L1
      gosub read_mem
LOOP:
           IF  (S1 =  0) and (S2 = 1) THEN stat = 1
           if  (S1 =  1) and (S2 = 0) THEN stat = 0
           if  (s1 =  0) and (s2 = 0) then
                pause 3000
                if (s1 =  0) and (s2 = 0) then
                     gosub beep2
                     gosub setting
                else
                     goto loop
                endif
           endif
           pause 50
           if stat = 1  then
                num  =  num + 1
                if num > 9999 then num = 0
                gosub  display
                gosub  write_mem
                gosub  action
           else
                 if num = 0 then num = 10000
                 num  =  num - 1
                 gosub  display
                 gosub  write_mem
                 gosub  action
           endif
           pause 80
           GOTO  LOOP
           END
'-----------------------------------------------------
read_mem:
      read 0,b0
      read 1,b1
      read 2,stat
      read 3,set
      num.byte0 = b0
      num.byte1 = b1
      return
```

มีต่อหน้าถัดไป

```
'------------------------------
write_mem:
        write 0,num.byte0
        write 1,num.byte1
        write 2,stat
        return
'------------------------------
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,$90],hun
      digit  =  num dig 3
      lookup digit,[$c0,$f9,$a4,$b0,$99,_
                    $92,$82,$f8,$80,$90],thou
      high latch
      pauseus 10
            ShiftOut dat,clk,1,[unit,ten,hun,thou]
            Pauseus 10
            Low latch
            Pauseus 10
            high latch
            Return
'------------------------------
setting:
      read 3,set
      gosub disp_setting
      pause 1000
      LOOP1:
             IF  (S1 =  0) and (S2 = 0) THEN
                pause 3000
                if (S1 =  0) and (S2 = 0) THEN
                     gosub beep1
                     return
                   else
                       goto loop1
             endif
          endif
          if  (S1 =  0) and (S2 = 1) then
             pause 50
             set  =  set + 1
             if set > 9999 then set = 9999
             gosub disp_setting
             write 3,set
          endif
          if  (S1 =  1) and (S2 = 0) then
             pause 50
             if set = 0 then set = 1
             set  =  set - 1
             gosub disp_setting
             write 3,set
          endif
          pause 80
                        GOTO  LOOP1
'---------------------------------
```

มีต่อหน้าถัดไป

```
beep1:
    freqout spk,20,2000
    return
'-------------------------------------
beep2:
    freqout spk,20,2000
    pause 200
    freqout spk,20,2000
    pause 100
    return
'-------------------------------------
action:
    if num = set then
       high L1
       if stat = 1 then
          num = num - 1
       else
          num = num + 1
       endif
    else
       low L1
    endif
    return
'-------------------------------------
```

มอบหมายงาน

1. เมื่อศึกษาและปฏิบัติเข้าใจดีแล้ว ให้พัฒนาต่อไปโดยการนำเอาแป้นคีย์ตัวเลขมาต่อในวงจรเพื่อสามารถกดตัวเลขกำหนดค่าได้แทนการกดปุ่มสวิทช์เพิ่ม – ลดค่าตัวเลข
2. มีปุ่มสวิทช์สำหรับกดเพื่อหยุดค้างสถานะ การนับฉุกเฉิน