บทที่ 2
คำสั่ง และรูปแบบการใช้งาน PIC BASIC PRO COMPILER

2.1 **Comment** ใช้สำหรับอธิบายขั้นตอนการทำงานของโปรแกรมในแต่ละบรรทัดคำสั่ง ใช้เครื่องหมาย ฝนทอง (Quotation mark) ‘ เช่น HIGH PORTB.0 ‘Turn ON LED
LOW PORTB.0 ‘Turn OFF LED

2.2 **Line Label** ใช้สำหรับกำหนดข้อความ เพื่อใช้อ้างอิงตำแหน่งของคำสั่งที่จะย้อนกลับไปทำงานซ้ำ เนื่องจาก PIC BASIC เป็นรูปแบบที่ไม่มีบรรทัดคำสั่งที่เป็นตัวเลข ข้อความที่กำหนดเป็น Label ต้องมีเครื่องหมาย Colon (:) ต่อท้ายด้วยเสมอ

เช่น
```
LOOP :  INPUT  S1
        IF  S1 = 1  THEN  LOOP
```

2.3 **Variables** เป็นที่สำหรับเก็บข้อมูลชั่วคราว โดยจะต้องกำหนดขนาด (Size) ซึ่งอาจเป็น bits, bytes หรือ Words PIC Basic Pro มองตัวแปรเป็นรีจิสเตอร์ที่ อ่าน / เขียนได้ มีรูปแบบดังนี้

Label VAR Size {.Modifier}

เช่น
```
SW1   VAR   bit    (ตัวแปรขนาด bit จะมีค่าระหว่าง 0 กับ 1 เท่านั้น)
L1    VAR   byte   (ตัวแปรขนาด byte จะมีค่าระหว่าง 0 - 255)
W0    VAR   word   (ตัวแปรขนาด word จะมีค่าระหว่าง 0 - 65535)
```

หมายเหตุ : .Modifier นั้น เป็น Option เพิ่มเติม สำหรับบอกว่า Variable ตัวนั้น สร้างมาได้อย่างไร

2.4 **Variable** ที่กำหนดให้ใช้กับ BASIC Stamps สามารถนำมาใช้กับ PICBASIC ได้ โดยจะต้องใส่คำสั่งดังต่อไปนี้ไว้บนหัวโปรแกรมก่อนนำตัวแปรไปใช้ คือ

Include “bs1 defs.bas” หรือ Include “bs2 defs.bas”

จากการที่กำหนดนิยามไว้ที่หัวของโปรแกรมตามข้างบนนี้ ทำให้เราได้ Variable ดังต่อไปนี้โดยอัตโนมัติ คือ

ถ้ากำหนด Include “bs1defs.bas” เราจะได้ VAR ดังนี้คือ B0 ~ B13, และ W0 ~ W6

ถ้ากำหนด Include “bs1defs.bas” เราจะได้ VAR ดังนี้คือ B0 ~ B25, และ W0 ~ W12

2.5 **Aliases** เป็นชื่ออื่น ๆ ที่กำหนดมาแทนชื่อ ตัวแปร ที่เรากำหนดตามข้อ 3 และข้อ 4 อีกข้อ เช่น

```
fido    VAR   dog
b0      VAR   w0.byte0      ‘b0 เป็น byte แรกของตัวแปร W0
b1      VAR   w0.byte1      ‘b1 เป็น byte ที่ 2 ของตัวแปร W0
flea    VAR   dog.0         ‘flea เป็น bit0 ของตัวแปร dog
```

2.6 **Arrays Variables** สามารถกำหนดชื่อตัวแปรหลายตัวในชื่อเดียวกันได้ มีรูปแบบคือ

Label	VAR	Size [No. of elements]

เช่น Shark VAR byte [10]

Fish VAR bit [8]

หมายเหตุ จำนวน Element สูงสุด มีได้ดังนี้

Size	No. of elements
Bit	256
Byte	96 *
word	48 *

* จำนวน element ของ byte และ word ขึ้นตรงต่อขนาด Register Bank ของ MCU

2.7 **Constants** เป็นชื่อที่กำหนดขึ้นแทนค่าคงที่ ซึ่งคล้ายกับกำหนดตัวแปร มีรูปแบบคือ

Label	CON	Constant expression

เช่น Mice CON 3

Traps CON mice * 1000

2.8 **Numeric Constants** PBP ได้กำหนดการใช้งานของตัวเลขได้ทั้ง 3 แบบด้วยกันคือตัวเลขฐาน 10 ฐาน 2 และฐาน 16 ด้วยรูปแบบตัวอักขระกำกับดังนี้คือ

8.1 ฐาน 10 ไม่ต้องกำหนดอักขระนำหน้า เช่น

100 คือ 100 ฐานสิบ

%100 คือ ค่าเลข 100 ฐานสอง

$100 คือ ค่าเลข 100 ฐานสิบหก

ส่วนค่า ASCII value นั้น กำหนดดังนี้

“A” ‘ASCII value for dec.65

“d” ‘ASCII value for dec.100

2.9 **String Constants ให้ใช้เครื่องหมายกำหนดเช่นเดียวกับ** ASCII value เช่น

Lcdout “Hello”

2.10 **การกำหนดค่า PORT และ Register ต่าง ๆ**

มีรูปแบบกำหนดดังต่อไปนี้คือ

```
PORTA  =  %01010101          ‘ส่งข้อมูลออกที่ PORTA ตามค่าที่กำหนด
VAR1   =  PORTB & $0F        ‘ทำ Logic AND ค่าที่ PORTB ด้วยค่า $0F แล้วกำหนดเป็น
                               ค่า VAR1
```

2.11 **การกำหนดค่า I/O Pins ของ MCU**

มีรูปแบบกำหนดดังต่อไปนี้คือ

```
PORTB.1  =     1       ‘กำหนดให้ขา 1 ของ PORTB มีค่าเป็น 1
LED  VAR  PORTA.0      ‘กำหนด ขา 0 ของ PORTA มีชื่อเป็น LED
     HIGH LED          ‘กำหนด LED (PortA.0) เป็น Logic high
```

สำหรับการกำหนด Pin ที่จะให้เข้ากันได้กับคำสั่ง Pin ของ BASIC Stamp MCU ที่ได้อ้างอิงถึง Pin 0 ~ 15 สามารถกำหนดได้ตาม PORT ของ MCU แบบต่าง ๆ ได้ตามตารางดังต่อไปนี้

	BASIC STAMP PIN	
PIC MCU	0 - 7	8 - 15
8 Pin	GPIO *	GPIO *
18 Pin	PORTB	PORTA *
28 Pin (ยกเว้น 14C000)	PORTB	PORTC
28 Pin (เฉพาะ 14C000)	PORTC	PORTD
40 Pin	PORTB	PORTC

***** เป็น Port ที่มีขาไม่ถึง 8 ขา

2.12 **การกำหนดให้ขาพอร์ท ของ MCU เป็น OUTPUT หรือ INPUT**

สามารถกำหนดได้โดยการกำหนด Tri-state Register (TRIS) ของ Port เช่นเดียวกับการใช้ภาษา Assembly โดยกำหนดค่ารีจิสเตอร์ที่ควบคุมทิศทางขา I/O คือ TRIS ถ้าบิทใดมีค่าเป็น 0 จะทำให้ขา I/O นั้นเป็น OUTPUT ถ้ามีค่าเป็น 1 จะทำให้ I/O ขานั้น เป็น INPUT

เช่น

```
TRISA = %00000000      ‘กำหนดให้ PortA เป็น Output ทุกขา
TRISB = %11110000      ‘กำหนดให้ 4 บิทบนของ PortB เป็น INPUT
                         ส่วน 4 บิทล่างเป็น OUTPUT
TRISA.0 = 0            ‘กำหนด Pin0 ของ PortA เป็น Output
```

2.13 **การเขียนคำสั่งหลายบรรทัด (Multi-Statement Lines)**

สามารถกำหนดได้ 2 แบบดังนี้คือ

W2 = W0

W0 = L1

W1 = W2

หรือ เขียนได้อีกแบบดังนี้คือ

W2 = W0 : W0 = L1 : W1 = W2

(การเขียนวิธีที่ 2 นี้ สามารถลดขนาดไฟล์ของ Code ได้)

2.14 **การกำหนดบรรทัดคำสั่งต่อเนื่อง (Line-extension)**

PIC BASIC ได้กำหนดจำนวนตัวอักขระในการเขียนคำสั่งในแต่ละบรรทัดไว้ไม่เกิน 250 ตัว แต่ถ้าในบรรทัดคำสั่งเดียวกันนั้น เราต้องการแบ่งออกเป็น 2 บรรทัด โดยบรรทัดทั้ง 2 ยังคงต้องต่อเนื่องกัน สามารถใช้เครื่องหมาย (_) ใส่ไว้ท้ายบรรทัดแรก ก็จะทำให้ 2 บรรทัดต่อเนื่องเป็นบรรทัดเดียวกัน เช่น

BRANCH B0, [Label1, Label 2, Label 3,…] , _

Label 4, Labels]

2.15 **INCLUDE** ใช้สำหรับรวมโปรแกรม.bas เข้าไปทำงานเป็นโปรแกรมเดียวกัน

เช่น INCLUDE “modedefs.bas” ให้ดูเพิ่มเติมใน บรรณานุกรมอ้างอิง

2.16 **DEFINE** ใช้กำหนดค่าพารามิเตอร์ของฮาร์ดแวร์ หรือ ซอฟท์แวร์ให้มีค่าและการทำงานที่แตกต่างไปจากที่โปรแกรมได้กำหนดเป็นเบื้องต้นไว้

เช่น DEFINE QSC 4 ‘กำหนด OSC. Speed 4 MHz

DEFINE ADC_BITS 8 ‘กำหนดจำนวน Bit การแปลง A/D เป็นขนาด 8 bit

***** การใช้คำสั่ง DEFINE ต้องเป็นตัวอักษรตัวพิพม์ใหญ่ (Capital letter)

ให้ดูเพิ่มเติมใน บรรณานุกรมอ้างอิง

2.17 **Math Operators**

+ บวก	MIN minimum Val.
- ลบ	NCD Encode
* คูณ	DCD Decode
** คูณ 16 บิท (บน)	REV Reverse bits
*/ คูณ 16 บิท (ล่าง)	SIN Sine
/ หาร	SQR Square root
// หารคิดเศษ (Modulus)	& AND
<< Shift Left	| OR
>> Shift Right	^ Ex. OR

ABS Absolute Value	~ NOT
COS Cosine	&/ NOT AND
DIG กำหนดค่าหลักตัวเลข	|/ NOT OR
MAX maximum Val.	^/ NOT Ex.OR

2.18 **การคูณ (Multiplication)** PIC BASIC ได้กำหนดการคูณไว้สองลักษณะคือ คูณ 16 bit และ คูณ 32 บิท

เช่น W1 = W0 * 1000 ‘คูณค่าใน W0 กับ 1000 และเก็บผลลัพธ์เข้าไว้ใน W1

W2 = W0**1000 ‘คูณค่าใน W0 กับ 1000 และเก็บ high order 16 bits ไว้ใน W2

2.19. **การหาร (Division)**

เช่น W1 = W0/1000 ‘หารค่า W0 โดย 1000 และเก็บผลลัพธ์ไว้ใน W1

W2 = W0//1000 ‘หารค่า W0 โดย 1000 และเก็บเศษไว้ใน W2

2.20 **การเลื่อนบิทข้อมูล (Bit shifting)** ใช้เครื่องหมาย << และ >> สำหรับเลื่อน bit ข้อมูลไปทางซ้ายและขวา ตามลำดับ ค่าที่ใช้เลื่อนมีค่าระหว่าง 0 - 15

เช่น B0 = B0 << 3 ‘Shift B0 ไปซ้าย 3 ตำแหน่ง

W1 = W0 >>1 ‘Shift W0 ไปขวา 1 ตำแหน่ง และนำผลลัพธ์ไปเก็บไว้ใน W1

2.21 **DCD** สำหรับหาค่า decode ค่า bit number (0-15) เป็นค่า binary number 8 bit

เช่น B0 = DCD 2 ‘กำหนด B0 = %00000100

2.22 **DIG** สำหรับกำหนดค่าจากจำนวนหลักของเลขฐาน 10 (ค่าที่กำหนด 0-4 คิดจากขวามือสุด)

เช่น B0 = 123 ‘Set B0 to 123

B1 = B0 **DIG** 1 ‘กำหนด B1 มีค่า 2 (หลักที่ 2 จากทางขวามือสุด)

2.23 **MAX และ MIN** กำหนดค่าพิกัดสูงสุดและต่ำสุด

B1 = B0 MAX 100 ‘ค่า B1 ระหว่าง 100 และ 255

B1 = B0 MIN 100 ‘ค่า B1 ระหว่าง B0 กับไม่เกิน 100

2.24 N**CD** เปลี่ยนค่า bit ของ Binary number มาเป็นค่าจำนวนของ Decimal (bit number ค่า 0-16)

เช่น B0 = NCD % 01001000 ‘Set B0 มีค่า 7

2.25 **REV** กลับค่าบิทของเลข Binary จำนวน bit (0-16) นับจากขวาสุด หรือ LSB

เช่น B0 = %10101100 REV 4 ‘Set B0 เป็น 10100011

2. 26 **Trigonometric SINE, COS**

เช่น B1 = COS B0 หาค่า Cosine ของ B0 แล้วเก็บค่าไว้ที่ B1

B2 = SIN B0 หาค่า Sine ของ B0 แล้วเก็บค่าไว้ที่ B2

2.27 **Square root, Absolute**

B0 = SQR W1 สำหรับหาค่ายกกำลังของตัวแปร W1 แล้วเก็บค่าไว้ที่ B0

B1 = ABS B0 สำหรับปรับ B0 ให้เป็นค่าบวกแล้วเก็บค่าไว้ที่ B1

2.28 **Bitwise Operators** การกระทำเกี่ยวกับบิท ซึ่งสามารถเลือกกระทำแยกบิท หรือ Set แต่ละบิท

เช่น B0 = B0 & % 00000001 ‘แยกบิท 0 ของตัวแปร B0 ออกมาใช้งาน

B0 = B0 | % 00000001 ‘Set bit0 ของตัวแปร B0

B0 = B0 ^ % 00000001 ‘กลับลอจิก bit0 ของตัวแปร B0

2.29 **Comparison Operators** เป็นเครื่องหมายเปรียบเทียบปริมาณระหว่าง expression หนึ่ง กับอีก Expression หนึ่ง ได้แก่

Operator	Description
= หรือ ==	เท่ากัน
<> หรือ !=	ไม่เท่ากัน
<	น้อยกว่า
>	มากกว่า
<=	น้อยกว่าหรือเท่ากับ
>=	มากกว่าหรือเท่ากับ

เช่น IF i > 10 THEN loop

2.30 **Logical Operators** เป็น Operator กระทำทางด้าน Logic โดยเปรียบเทียบว่าเป็นจริงหรือเท็จ ได้แก่

Operator	Description
AND หรือ 82	Logic AND
OR หรือ ||	Logic OR
XOR หรือ ^^	Logic XOR
ANDNOT	Logic NAND
ORNOT	Logic NOR
XORNOT	Logic NXOR

เช่น IF (A == big) AND (B > mean) THEN run

2.31 **คำสั่งในภาษา PIC BASIC PRO (PIC BASIC Statements)**

2.31.1 **@** เป็นคำสั่งสำหรับแทรกบรรทัดที่เป็น Assembly Code

รูปแบบ

```
@ Assembly Statement
```

ตัวอย่าง (1)

```
i       Var     byte    ‘กำหนดตัวแปร
rollme  VAR     byte    ‘กำหนดตัวแปร
FOR     i = 1 To 4      ‘ให้ I มีค่า ตั้งแต่ 1 ถึง 4
@  rlf  rollme, 1       ‘ให้เลื่อนค่าในตัวแปร rollme ไปทางขวา 1 บิท
NEXT  I
```

(2)

```
@ Include “fp.asm”
```

2.31.2 คำสั่ง ADCIN เป็นคำสั่งสำหรับค่าสัญญาณอนาล็อก สำหรับ MCU บางเบอร์ เช่น PIC 16F874/877 เป็นต้น

รูปแบบ

```
ADCIN channel, Var
```

***ข้อควรจำ** ก่อนใช้คำสั่ง ADCIN ต้องกำหนด PORT ที่จะรับค่าอนาล็อกให้มีสถานะเป็น INPUT ก่อน โดยกำหนดที่ register TRIS และต้องกำหนด PORT ที่จะรับค่าให้สามารถรับสัญญาณอนาล็อก โดยกำหนดที่ Register ADCON1 ก่อน คำสั่งนี้สามารถแปลงค่า A / D ได้ทั้ง 8 บิทและ 10 บิท รายละเอียดอธิบายในใบงานทดลองเรื่อง A / D

ตัวอย่าง

```
TRISA    = 255      ‘กำหนด PortA = input
ADCON1   =  0       ‘PortA เป็นอนาล็อก (ดูใน Data sheet ของ MCU)
ADCIN 0, B0         ‘อ่านอนาล็อกช่อง 0 เข้าตัวแปร B0
```

ค่าที่โปรแกรมกำหนดไว้เป็น Default แล้วคือ

```
DEFINE  ADC_BIT 8               ‘กำหนดจำนวน BIT ใช้งาน 8 บิท
DEFINE  ADC_SAMPLEUS 50         ‘Sampling rate = 50MS
```

2.31.3 คำสั่ง ASM .. ENDASM

เป็นคำสั่งสำหรับแทรกชุดคำสั่งภาษา Assembly มากกว่า 1 บรรทัด

รูปแบบ ASM

ภาษา Assembly

ENDASM

ตัวอย่าง ASM

```
bsf   PORTA,0
bcf   PORTB,0
```

ENDASM

2.31.4 คำสั่ง BRANCH

เป็นคำสั่งสำหรับใช้กระโดดไปทำงานตามตำแหน่ง ตามค่าตัวแปรที่เป็น INDEX

รูปแบบ BRANCH INDEX, [Label1, Label2, …]

ตัวอย่าง BRANCH B4, [dog, cat, fish]

อธิบาย ถ้า B4 มีค่า = 0 ไปทำงานที่ Label dog
ถ้า B4 มีค่า = 1 ไปทำงานที่ Label cat
ถ้า B4 มีค่า = 2 ไปทำงานที่ Label fish

คำสั่ง BRANCH มีได้ถึง 255 Label แต่ถ้าการกระโดดข้าม Register Page คือตำแหน่ง Label อยู่ห่างกันเกิน 1 K ต้องใช้คำสั่ง BRANCHL แทน BRANCHL มีรูปแบบการใช้งานเช่นเดียวกัน

2.31.5 คำสั่ง BUTTON

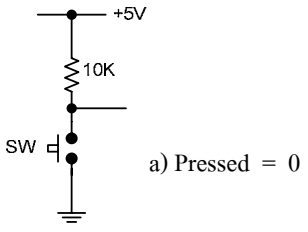
เป็นคำสั่งที่รออ่านค่าที่ขาของ Port แล้วเก็บค่าเข้าไว้ในตัวแปร โดยมีการแก้การเด้งของหน้าสัมผัสสวิทช์ และ Auto-repeat และทำให้ pin นั้นเป็น input โดยอัตโนมัติ

รูปแบบคำสั่ง

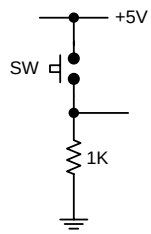
BUTTON Pin, Down, Delay, Rate, BVar, Action, Label

Pin เป็นขาของ Port เช่น PORTA.0 ~ PORTA.4 เป็นต้น

Down เป็นสถานะของ Pin เมื่อมีการกด (มีค่า 0 กับ 1)



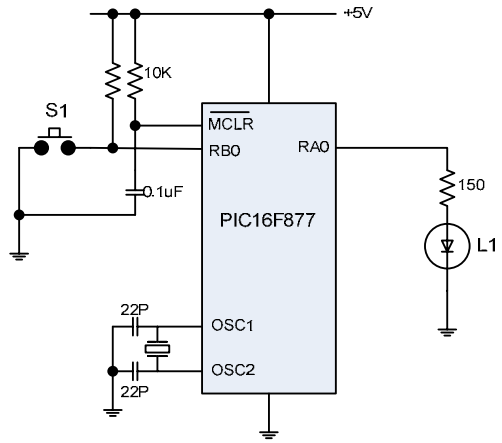
a) Pressed = 0



b) Pressed = 1

Rate	เป็นค่า Auto-repeat (0 ~ 255)
Delay	เป็นค่าหน่วงเวลาก่อนที่จะเกิด auto-repeat มีค่า 0 ~ 255 (ถ้ามีค่า 0 จะไม่มี denounce ถ้ามีค่า 255 จะไม่มี auto-repeat)
BVar	เป็นขนาดของตัวแปรที่ถูกใช้สำหรับการหน่วงเวลา และ auto-repeat ก่อนใช้ต้อง Set ให้เป็น 0 ก่อน
Action	เป็นสถานะ การกระทำ 0 = ไม่กด, 1 = กด
Label	เมื่อคำสั่ง BUTTON เป็นจริงก็กำหนดให้ไปกระทำที่ Label

Ex. จาก Hardware เมื่อกด S1 ให้หลอด L1 ติด 0.5 นาที แล้วดับ 0.5 นาที แล้วกลับไปตั้งต้นใหม่



รูปที่ 28 วงจรตัวอย่างการใช้คำสั่ง BUTTON

เขียนเป็นโปรแกรมภาษา PIC BASIC ได้ดังนี้

```
        INCLUDE "BS1DEFS.BAS"
        S1      VAR     PORTB.1
        L1      VAR     PORTA.0
        LOW     B2      <-- เอามาจาก File "BS1DEFS.BAS"
LOOP1 : BUTTON  S1, 0, 100, 10, B2, 0 Loop2
LOOP2 : PAUSE  50
                HIGH  L1
                PAUSE 500
                LOW  L1
                PAUSE  500
                GOTO  LOOP1
                END
```

เพื่อทำให้เกิด Auto-repeat

delay = 50 X 100 = 5000mS

และ Repeat rate = 50 X 10

= 500mS

2.31.6 คำสั่ง CALL

เป็นคำสั่งสำหรับเรียกโปรแกรม Assembly subroutine มา execute

เช่น

CALL pass.asm (extension ไม่ต้องใส่ก็ได้)

2.31.7 คำสั่ง CLEAR

เป็นคำสั่งกำหนดให้ค่าใน RAM register มีค่าเป็น 0 หรือ clear ตัวแปรที่กำหนดทั้งหมด

เช่น CLEAR B1 'ทำให้ B1 มีค่าเป็น 0'

CLEAR 'ทำให้ทุกตัวแปรมีค่าเป็น 0

2.31.8 คำสั่ง CLEARWDT

เป็นคำสั่ง Clear Watchdog Timer เพื่อไม่ให้ MCU เกิดการ Reset เนื่องจาก Watchdog Timer คำสั่งนี้ต้องแทรกไว้ตามตำแหน่งที่มีการวนลูปนาน ๆ ซึ่งอาจเกิดการ Reset ระหว่างการประมวลผลขึ้นได้ กรณีที่ไม่ต้องใช้คำสั่ง **CLEARWDT** เราต้องใส่คำสั่ง

DEFINE NO_CLRWDT 1 ไว้ที่หัวของโปรแกรม หรือยกเลิกการใช้ Watchdog Timer ในขั้นตอนการโปรแกรมตัวชิพ

2.31.9 คำสั่ง COUNT

เป็นคำสั่งสำหรับนับจำวนลูก pulse ที่ปรากฏที่ Pin ในช่วงคาบเวลาที่กำหนด

รูปแบบ COUNT Pin, Period, Var

Pin เป็นขาของ MCU Port

Period เป็นคาบเวลาที่รอการนับ

Var ตัวแปรที่กำหนดเพื่อเก็บค่าที่นับได้

Ex. ต้องการหาค่าความถี่ที่ป้อนเข้าขา Pin PortB.0 แล้วเก็บค่าไว้ใน W1

COUNT PORTB.0, 1000, W1

ช่วงเวลาในการ Scan เพื่อนับค่า Pulse ถ้าใช้วงจร Clock 4 MHz จะมีค่า = 20 mS และถ้าใช้วงจร Clock 20 MHz จะมีค่า = 4 mS ดังนั้น คำสั่ง Count สามารถจะนับ Pulse ที่มีความถี่อยู่ระหว่าง 0 ~ 125 kHz ที่ Clock 20 MHz

2.31.10 คำสั่ง INPUT

เป็นคำสั่งที่กำหนดให้ขา Pin ของ Port มีสถานะเป็น Input

รูปแบบ INPUT Pin

Pin อาจเป็นค่าตัวแปร หรือ ขาของ Port โดยตรงก็ได้

ตัวอย่างการใช้งาน

```
INPUT  PORTA.0     'ทำให้ PORTA ขา 0 เป็น INPUT
```

Ex. การใช้งานอีกแบบหนึ่ง

```
        S1   VAR   PORTA.0
LOOP:   INPUT      S1          'รับค่าลอจิกที่ PORTA.0 ที่ตั้งชื่อไว้เป็น S1
        IF  S1 = 1  THEN  LOOP  'หากค่า S1 ยังคงเป็นลอจิก 1 ก็ให้ไปที่ LOOP
        GOTO  SET_ON
```

2.31.11 คำสั่ง HIGH

เป็นคำสั่งให้ขาของ Port มีสถานะลอจิก “1” เมื่อใช้คำสั่งนี้ที่ขา Pin ใด ขานั้นจะมีสถานะเป็น OUTPUT โดยอัตโนมัติ

รูปแบบ HIGH Pin

Pin อาจเป็นชื่อขาของ Port หรือตัวแปรแทนขานั้น ๆ ก็ได้

เช่น

```
HIGH  PORTA.0        'ทำให้ขา 0 ของ PortA เป็นลอจิก “1”
L1    VAR    PORTB.0
      HIGH   L1
```

หรือในทำนองเดียวกัน เราสามารถใช้คำสั่งอีกแบบหนึ่งได้ดังนี้ คือ

```
PORTB.0 = 1
```

หรือ

```
L1  VAR   PORTB.0
L1  =   1
```

2.31.12 คำสั่ง LOW

เป็นคำสั่งทำให้ขาของ Port มีสถานะลอจิก “0” และเป็น Output โดยอัตโนมัติ

รูปแบบ LOW Pin

Pin อาจเป็นชื่อขา Port หรือตัวแปรแทนขานั้น ๆ ก็ได้

เช่น

```
LOW  PortA.0
```

หรือ

```
PORTA.0 = 0
```

Ex.

```
L1    VAR    PORTB.2
      LOW    L1
```

2.31.13 คำสั่ง OUTPUT

เป็นคำสั่งที่กำหนดให้ขาของ Port มีสถานะเป็น Output

รูปแบบ OUTPUT PIN

Pin อาจเป็นชื่อขา หรือตัวแปรที่แทนชื่อขาของ Port ก็ได้

เช่น OUTPUT PORTA.3

หรืออีกวิธีหนึ่งอาจใช้วิธีกำหนดที่ Register TRIS แทนก็ได้ดังนี้คือ

TRISA.3 = 0

Ex. TRISB = %00000000 ‘กำหนดให้ PortB เป็น OUTPU ทุกขา

2.31.14 คำสั่ง PAUSE

เป็นคำสั่งหน่วงเวลามีหน่วยควบเป็นมิลิวินาที (mS)

รูปแบบ PAUSE Period

Period มีหน่วยเป็น mS มีค่าอยู่ระหว่าง 0 ~ 65535mS

Ex.
```
HIGH    PORTB.0
PAUSE   1000    ‘หน่วงเวลา 1 วินาที
LOW     PORTB.0
END
```

2.31.15 คำสั่ง PAUSEUS

เป็นคำสั่งหน่วงเวลามีคาบเป็นไมโครวินาที (μS)

รูปแบบ PAUSEUS Period

period มีหน่วยเป็นไมโครวินาที (μS) มีค่าสูงสุดถึง 65535 μS ส่วนค่าต่ำสุดนั้นขึ้นอยู่กับความถี่ของวงจร OSC เช่น ถ้าใช้ OSC 4 MHz ค่าต่ำสุดของ period จะมีค่า = 24 μS และถ้าใช้ OSC 20 MHZ ค่าต่ำสุดของ Period จะมีค่า = 3 μS

2.31.16 คำสั่ง PEEK

เป็นคำสั่งสำหรับอ่านค่าจาก Register ต่าง ๆ ที่ระบุเข้าไปเก็บไว้ในตัวแปร

รูปแบบ PEEK Address, Var

Address อาจเป็นค่า register ของ Port ต่าง ๆ ก็ได้ หรือ เป็นตำแหน่ง RAM หรือ
เป็นตำแหน่ง RAM หรือ Register ทั่วไปก็ได้
VAR เป็นตัวแปรที่กำหนดขึ้นเพื่อเก็บค่าใน register

คำสั่ง PEEK บางครั้งอาจไม่ต้องใช้ เราอาจเขียนเป็นสมการโดยตรงก็ได้ คือ

PEEK PORTA,B1

หรือ B 1 = PORTA ให้นำเอาสถานะ Pin ของ Porta มาเก็บไว้ที่ตัวแปร B1

2.31.17 คำสั่ง POKE

เป็นคำสั่งที่กำหนดค่าให้กับ register ต่าง ๆ ของ MCU

รูปแบบ POKE Address, Value

Address เป็นตำแหน่งของ register ต่าง ๆ ของ MCU ทั้ง Register พิเศษ
เช่น PORT ต่าง ๆ หรือเป็นตำแหน่ง register ใช้งานทั่ว ๆ ไป หรือเป็นตำแหน่ง RAM
Valve เป็นค่าตรงหรือเป็นค่าคงที่ใด ๆ

เช่น POKE TRISA.0 กำหนดให้ PORTA เป็น OUTPUT หรืออาจเขียนเป็นสมการตรง ๆ เลยก็ได้ เช่น
TRISA.0 = 0

2.31.18 คำสั่ง PULSIN

เป็นคำสั่งวัดความกว้างของ PULSE ที่มาปรากฏที่ขา Pin โดยนับเป็น Count ที่มีค่าสูงสุด 65535 ค่า

รูปแบบ PULSIN Pin, State,Var

pin เป็นขาของ Port
State เป็นสถานะ การตรวจจับสัญญาณ PULSG ที่เข้ามา ถ้าเป็น 0 จะตรวจจับสัญญาณ
Low Pulse ถ้าเป็น 1 จะตรวจจับสัญญาณ High Pulse
Var เป็นตัวแปรที่เก็บค่า Count ความกว้างของ pulse การกำหนดขนาดของตัวแปร
ต้องให้มีขนาดพอที่จะเก็บ จำนวน Count ความกว้างของ pulse ซึ่งมีค่าสูงสุด
65535 count แต่เราสามารถกำหนดว่าจะกำหนดค่าสูงสุดได้ด้วยคำสั่ง DEFINE เช่น DEFINE
PULSIN _MAX 1000

<u>หมายเหตุ</u> คำสั่ง DEFINE ที่กำหนดนี้จะมีผล ในคำสั่ง RCTIME ด้วย เช่นเดียวกัน

ความละเอียด แต่ละ Count จะขึ้นอยู่กับค่า OSC.ที่ใช้

เช่น ถ้า OSC. 4 MHz แต่ละ Count = 4 ไมโครวินาที
ถ้า OSC. 20 MHz แต่ละ Count = 2 ไมโครวินาที

Ex. ต้องการวัด Pulse width ขาที่ 4 Portb แล้วเก็บค่าไว้ในตัวแปร W3

PULSIN PORTB.4,1,W3

2.31.19 คำสั่ง PULSOUT

เป็นคำสั่งส่ง PULSE ออกที่ขาของ Port ตามคาบเวลาความกว้างที่กำหนด

รูปแบบ PULSOUT Pin, Period

Pin เป็นขาของ Port

Period เป็นคาบเวลาที่กำหนดให้ Pulse เกิดขึ้น

ความละเอียด ของ Pulse ขึ้นอยู่กับ OSC. ที่ใช้

ถ้า OSC. MHz ความละเอียด เท่ากับ 10 μS

ถ้า OSC. 20 MHz ความละเอียด เท่ากับ 2 μS

Ex. ต้องการสร้าง Pulse ให้หลอดกระพริบ ทุก 100 mS เป็นช่วง ๆ ละ 1 Sec.

```
        L 1  VAR PORTA.0
LOOP :  PAUSE 1000
        PULSOUT   L1, 10000
        GOTO   LOOP
        END
```

10000 x 10
= 100,000 μS
= 100 mS

2.31.20 คำสั่ง TOGGLE

เป็นคำสั่ง กลับสถานะของ Pin ในแต่ละขา Port ให้มีลอจิกเป็นตรงกันข้ามกับของเดิม

รูปแบบ TOGGLE Pin

Ex เขียนโปรแกรมเมื่อกด Reset ให้หลอด L1 ติด 5 วินาทีแล้วดับ

```
L1      VAR     PORTB 1
        HIGH  L1
        PAUSE 5000
        TOGGLE  L1
        END
```

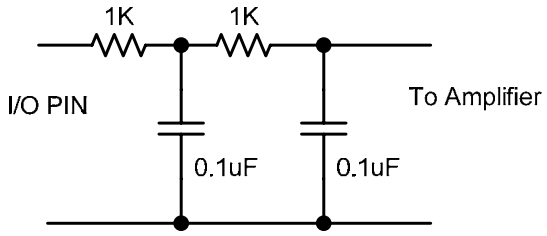
2.31.21 คำสั่ง FREQOUT

เป็นคำสั่งผลิตความถี่ออกทางขา Pin รูปแบบของ SING WAVE โดยขับออกมาในลักษณะเป็น PWM ความถี่ที่ออกมา มีค่าระหว่าง 0 ~ 32767 HZ สามารถผลิต 2 ความถี่ ในเวลาเดียวกับที่ขาเดียวกัน

รูปแบบ FREQOUT Pin, ONMS, Frequency1 ,{Freq 2 }

Pin	เป็นขาของ Port
ONMS	เป็นช่องคาบเวลาที่ผลิตความถี่มีค่าเป็น MS
Freq 1 , Freq 2	เป็นความถี่มีค่าเป็น Hz

<u>หมายเหตุ</u> เนื่องจากสัญญาณความถี่ที่ผลิตออกมานี้ เป็นลักษณะ PWM ดังนั้นถ้านำไปใช้งานในลักษณะที่เป็น Pore Sine Wave จะต้องมีวงจรกรองความถี่สูงออกไปดังนี้



รูปที่ 29 วงจร Low Pass Filter

Ex. ต้องการผลิตความถี่ 1KHz ออกที่ขา 2 Port A 2 วินาที

FREQOUT PORTA.2, 2000, 1000

Ex. ต้องการส่ง 350 Hz/440 Hz (Dial Tone) ออก 2 วินาที ที่ขา 1 Port B

FREQOUT PORTB.1, 2000, 350, 440

2.31.22 คำสั่ง PWM

เป็นคำสั่งสำหรับส่ง PULSE WIDTH MODULATION (PWM) ออกที่ขา Pin ตามจำนวน Cycle ที่ระบุ

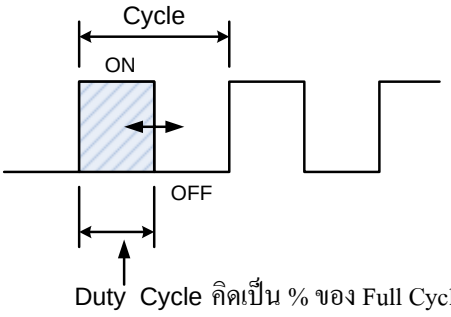
รูปแบบ PWM Pin , Duty, Cycle

Pin	เป็นขาของ Port
Duty	เป็นค่า % ของ duty cycle

เช่น ต้องการสร้าง Pulse width 50% ออกที่ขา 5 PortB จำนวน 100 ลูก

PWM PORTB.5, 127,100

เราสามารถนำเอาเทคนิคของ PWM มาประยุกต์ใช้ในการควบคุมแบบอนาล็อก เช่น ควบคุมความเร็วของมอเตอร์ไฟ DC หรือขดลวดความร้อน ทั้งนี้ เนื่องจากแรงเคลื่อนค่าเฉลี่ยในแต่ละ Cycle ที่ขับออก จะแปรผันโดยตรงกับความกว้างทางรูปคลื่น



รูปที่ 30 แสดงส่วนประกอบของสัญญาณ PWM

ค่าความกว้างของ pulse ที่แคบที่สุด จะมีค่า = 0 และ จะมีค่ามากที่สุด เมื่อเคลื่อนออกมาเต็ม cycle

* อัตราส่วนระหว่างความกว้างของ PULSE (PULSE WIDTH) กับคาบเวลาใน 1 Cycle เราเรียกว่าDUTY CYCLE มีหน่วยเป็น %

$$\text{DUTY CYCLE} = \frac{\text{PULSE WIDTH}}{\text{CYCLE}} \times 100\ \%$$

แรงเคลื่อนค่าเฉลี่ย Output (Vout) เราสามารถคำนวณได้จากสูตร

$$V_{out} = \text{DUTY CYCLE} \times V_{supply}$$

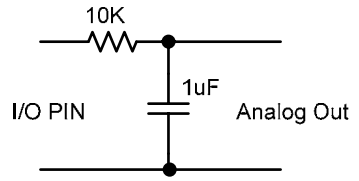
Duty Cycle คิดเป็น % (0 ~ 100%)

Vsupply คือแรงเคลื่อน P-To-P ของ Waveform

Ex. จงหาค่าแรงเคลื่อนค่าเฉลี่ย DC ของ PWM ที่มีค่า duty cycle = 50% โดยมีค่าแรงเคลื่อน ของรูปคลื่น = 5 V P - P

$$V_{out} = \frac{50 \times 5\ v}{100} = 2.5\ [V]$$ ←

<u>หมายเหตุ</u> 1. เนื่องจากแรงเคลื่อนค่าเฉลี่ย Output ที่ออกมานี้ ถึงแม้จะเป็นค่าเฉลี่ยไฟตรง แต่ก็ยังเป็นรูป ของ PULSE อยู่ ดังนั้น ในการใช้งานจริงที่ต้องการไฟ DC ที่ราบเรียบ ควรจะต้องมีวงจร กล่องความถี่สูง(Low Pass Filter)ออกไปก่อน



รูปที่ 31 วงจรกรองความถี่สัญญาณ PWM
เพื่อให้เหลือแต่ค่าเฉลี่ยที่เป็นไฟ DC

2. คำสั่ง PWM นี้ไม่สามารถส่งรูปเคลื่อนต่อเนื่องได้ ถ้าต้องการให้ MLU ส่งค่า PWM ออกมาต่อเนื่องโดยยังสามารถควบคุมความกว้างของ Pulse ได้ จะต้องเลือกใช้ MCU ที่มีโครงสร้างฮาร์ดแวร์ มี Pin ที่มีโมดูล CCP ภายในตัวชิพ เช่นเบอร์ PIC 16F877/874 เป็นต้น ซึ่งเราสามารถใช้คำสั่ง HPWM ได้โดยตรง

2.31.23 คำสั่ง HPWM

เป็นคำสั่งส่งรูปคลื่น PWM ออกอย่างต่อเนื่องจากโมดูล CCP ของ MCU โดยสามารถส่งรูปคลื่นต่อเนื่องตามลำพังต่อเนื่องได้ ในขณะที่ยัง execute คำสั่งอื่นอยู่

รูปแบบ HPWM Channel, Duty cycle, Freq

Channel คือขา Pin ที่ฮาร์ดแวร์กำหนดให้เป็นขา PWM

Duty cycle คิดเป็นความละเอียดของ binary digit เช่น ถ้า 8 bit จะมีค่าระหว่าง (0 ~ 255) โดยที่ 0 = 0% และ 255 = 100%

Freq คือความถี่ของรูปคลื่น PWM มีค่าระหว่าง 245 ~32767 Hz

ค่าต่ำสุดของ Freq จะขึ้นอยู่กับวงจร OSC. ที่ใช้ เช่น ถ้าใช้ OSC. 4 MHz ความถี่ต่ำสุด = 245 Hz ถ้าใช้ OSC. 20 MHz ความถี่ต่ำสุด = 1221 Hz

Ex.1 ต้องการส่ง PWM 50 x duty cycle ที่ 1 KHz ต่อเนื่อง ออกไปที่ PORTC.2

HPWM 1, 127, 1000

Ex.2 ต้องการส่ง PWM 25 % duty cycle ที่ 20 KHz ต่อเนื่องออกไปที่ PORTC.2

HPWM 1, 64, 20000

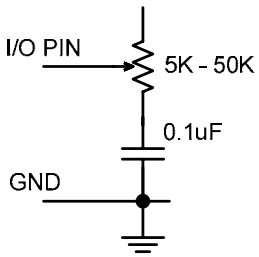
หมายเหตุ คำสั่ง HPWM นี้ใช้ได้เฉพาะไมโครคอนโทรลเลอร์ ที่มีขา Pin ที่มีโครงสร้างภายในตัวชิพ Support PWM เท่านั้น เช่น PIC 16F877 , PIC 16F874 เป็นต้น
(สำหรับ PIC16F877 จะมีขา I/O ที่รองรับคำสั่ง HPWM อยู่ 2 ขาแบบอิสระ ได้แก่ ขา RC2 เป็น HPWM ช่อง 1 และ RC1 เป็น HPWM ช่อง 2)

2.31.24 คำสั่ง POT

เป็นคำสั่งสำหรับอ่านค่า potentiometer ที่ Pin ของ Port

รูปแบบ POT Pin, Scale, Var

ค่าความต้านทานที่อ่านได้มาจากค่า Tine Constant ของการ discharge ของค่าประจุที่ Capacitor ผ่านตัวต้านทางที่ปรับค่าได้ (ปรกติจะมีค่าประมาณ 5K – 50K) Pin เป็นขาที่ต่อกับ R-C Network ตามรูป



รูปที่ 32 วงจรใช้กับคำสั่ง POT

Scale ใช้เป็นค่าสำหรับ adjust ค่า RC-Time Constant ให้เหมาะสม สำหรับวงจรที่มีค่า RC-Constant มาก Scale จะต้องปรับให้มีค่าน้อยลง สำหรับวงจรที่มีค่า RC-Constant น้อย จะต้องกำหนดให้ค่า Scale มีค่ามาก (ค่า Scale จะกำหนดได้ระหว่าง 0 ~ 255)

ถ้าปรับค่า Scale ได้เหมาะสมแล้ว ค่าที่อ่านได้ จะมีค่าเป็น 0 เมื่อปรับค่า ด.ต.ท. มีค่าน้อยสุดและมีค่าเป็น 255 เมื่อปรับค่า ด.ต.ท. มีค่าสูงสุด

Var เป็นชื่อตัวแปรที่กำหนดไว้ สำหรับเก็บค่า potentiometer ที่อ่านได้

Ex.

```
B0     VAR    byte
Scale  VAR    byte
        For   Scale =  1 To 255
                POT  PORTB.0 ,  Scale, B0
                IF (B0 > 253) THEN calibrate
        NEXT Scale
        SEROUT  2, 0, [“increase R”, 10, 13]
        STOP
Calibrate:
        SEROUT 2 , 0 [“Scale =  “ , # Scale , 10 , 13]
        END
```

2.31.25 คำสั่ง RCTIME

เป็นคำสั่งสำหรับวัดช่วงเวลาที่เกิดขึ้นเมื่อ Pin ของ Port อยู่ในสถานะใดสถานะหนึ่ง (คือ 0 หรือ 1) ที่ต่อเป็น RC- Network

รูปแบบ RCTIME Pin , State , Var

Pin	เป็นตัวแปรหรือค่าคงที่ (0-15) ใน BS1 , BS2 หรือเป็นชื่อ PORT ก็ได้ เช่น PORTA.1
State	เป็นสถานะเริ่มวัดช่วงเวลา (มีค่า 0 หรือ 1)
Var	เป็นตัวแปรที่มีค่าระหว่าง 0 ~ 65535

ความละเอียด ของควบเวลา ขึ้นอยู่กับ osc. ที่ใช้คือ

ถ้าใช้ OSC 4 MHz ควบเวลาจะมีค่า 10 mS. และ 2 mS. เมื่อใช้ OSC 20 MHz

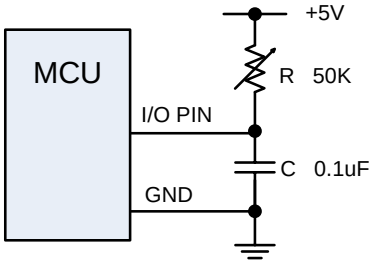
ค่า RCTIME จะนับจนกระทั่งมีค่าสูงสุด (65535) กรณีที่ไม่มี pulse มาทำให้เปลี่ยน state

กรณีที่ เราต้องการให้นับสูงสุดต่ำกว่าค่า 65535 สามารถกำหนดไว้ล่วงหน้าก่อนการใช้คำสั่งดังนี้

```
DEFINE   PULSIN_MAX 1000
```

Ex. 1

```
LOW  PORTB.3      'Discharge ค่า c ก่อน start
PAUSE  10  "      'To  Discharge  10 ms
RCTIME PORTB.3 , 0 , W0        'อ่านค่า RC เก็บไว้ ใน W0
```



รูปที่ 33 การต่อวงจรที่ใช้กับคำสั่ง RCTIME

2.31.26 คำสั่ง RANDOM

เป็นคำสั่งสำหรับสุ่มค่าตัวเลขแล้วเก็บค่าไว้ที่ตัวแปรที่กำหนด

รูปแบบ RANDOM Var

Var มีขนาด 16 bit ค่าสุ่มมี 65535 ค่า

:

Ex.

```
Random  B0
PORTB = B0
:
```

2.31.27 คำสั่ง NAP

เป็นคำสั่งให้ไมโครคอนโทรลเลอร์หยุดพักทำงานชั่วขณะในช่วงสั้น ๆ ในช่วงนี้ MCU จะไม่รับคำสั่งใด ๆ และจะใช้พลังงานน้อยที่สุด ประมาณ 20 ไมโครแอมป์

รูปแบบ

NAP Reriod

ค่า Period มีค่าระหว่าง 0 ~ 7 โดยมีช่วงเวลาที่แตกต่างกันดังนี้คือ

Period	Delay (ms)
0	18
1	36
2	72
3	144
4	288
5	576
6	1.152 Sec
7	2.304 Sec

Ex. ใช้คำสั่ง NAP เพื่อพักการทำ execute คำสั่ง

```
LOOP  :  HIGH PORTA.1
         PAUSE 5000
         LOW PORTA.1
         NAP 7
         GOTO LOOP
         END
```

2.31.28 คำสั่ง SLEEP

เป็นคำสั่งสำหรับให้ MCU หยุดทำงานโดยใช้พลังงานน้อย ประมาณ 20 uA มีช่วงเวลาครั้งละ 2.3 วินาที ความเที่ยงตรง 99.9 %

ขณะที่ MCU อยู่ใน SLEEP MODE CPU จะไม่รับคำสั่งใด ๆ ถ้าต้องการให้ออกจาก SLEEP MODE จะต้องปลุกให้ตื่นก่อน (Wakeup) ด้วยการกด RESET

รูปแบบ SLEEP Multiplier

Multiplier เป็นตัวคูณสำหรับตั้งเวลา Sleep ตั้งแต่ 1-65535 วินาที หรือ 18 ชั่วโมง

คำสั่ง **SLEEP** จะใช้ Watchdog Timer ดังนั้นช่วงเวลาจริงขณะที่ไม่ได้กำหนดค่า Multiplier จะมีค่าประมาณ 2.3 วินาที

Ex. 1 SLEEP 60 ‘ให้ MCU หลับประมาณ 1 นาที

Ex. 2
```
S1 VAR PORTB.0
L1 VAR PORTB.7
LOOP 1 : INPUT S1
         IF S1 = 1 THEN LOOP1
LOOP 2 : HIGH L1
         PAUSE 2000
         LOW L1
         SLEEP 60
         GOTO LOOP1
         END
```

หยุดทำงาน 60 x 2.3 ประมาณ 120 วินาที

2.31.29 คำสั่ง STOP

เป็นคำสั่งให้หยุดการทำงานของโปรแกรมทันที โดยไม่มีเงื่อนไข โดยไม่ต้องรอให้จบโปรแกรม ถ้าต้องให้ทำงานใหม่ต้องกด RESET

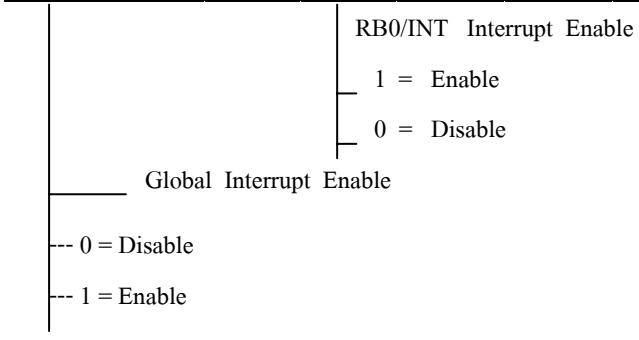
2.31.30 คำสั่ง ON INTERRUPT

คำสั่งนี้มีไว้สำหรับบอกให้ MCU คอยรับบริการการขัดจังหวะการทำงานปกติ โดยให้กระโดดไปทำงานที่ Label ที่กำหนดไว้เมื่อมีสัญญาณการจัดจังหวะ (Interrupt) เข้ามาและจากนั้นเมื่อเสร็จสิ้นจบงานแล้วจะกลับมาทำงานที่ค้างไว้ ขณะ interrupt ใน Main Program ด้วยคำสั่ง RESUME

รูปแบบ ON INTERRUPT GOTO Label

Register ที่ควบคุมการบริการ Interrupt ของ ไมโครคอนโทรลเลอร์ PIC คือ **INTCON** และรีจิสเตอร์ที่เกี่ยวข้องอีกหลายตัว (ดูรายละเอียดใน Datasheet และใบงานทดลอง)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
GIE	EEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF



- ในการใช้งาน ขา Pin ที่ต้องต่อกับฮาร์ดแวร์ สำหรับป้อนสัญญาณ Interrupt คือ ขา RBO / INT (ขา 6 ของ PIC 16F84A หรือขา 33 ของ PIC 16F877)
- ใน Main Program จะต้องใส่คำสั่ง ON INTERRUPT ไว้ที่ส่วนหัวของโปรแกรม และต้อง SET ค่าที่ Register INTCON ให้สามารถบริการ Interrupt ตัวอย่าง Set bit ที่ 7 และ bit ที่ 4 ให้มีค่าเป็น “1”
- ที่ส่วนท้ายของ Main Program กับ Interrupt Service Routineจะต้องใส่คำสั่ง DISABLE ปิดคั่นไว้
- ที่ Subroutine ที่บริการ Interrupt ต้องเคลียร์แฟล็ก (INTCON.1 = 0) และต้องปิดด้วยคำสั่ง RESUME

Ex รูปแบบการใช้คำสั่ง

```
            L1  VAR  PORTA.0
            L2  VAR  PORTA.1
            ON  INTERRUPT  GOTO  LOOP 2
            INTCON  =  %  10010000
LOOP1 :     HIGH  L1
            PAUSE  500
            LOW  L1
            PAUSE  500
            GOTO  LOOP1
            END
            DISABLE
LOOP2 :     HIGH  L2
            INTCON.1 = 0
            RESUME
```

* การปิดการใช้บริการ Interrupt ที่ ขา RBO / INT อย่างถาวรให้ใช้คำสั่ง

```
INTCON  =  $80
```

2.31.31 คำสั่ง DTMFOUT

เป็นคำสั่งสำหรับสร้างสัญญาณการกดหมายเลขโทรศัพท์แบบเสียงคู่หลายความถี่ ซึ่งเป็นสัญญาณของการกดปุ่มโทรศัพท์แบบ Tone Dial

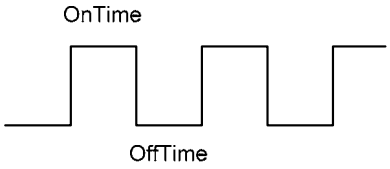
รูปแบบ DTMFOUT PIN, On time, Off time, [Tone,….]

Pin ขาของ Port ที่ใช้ส่งสัญญาณการกดออกไปใช้งาน

On time ช่วงเวลาเกิด On (“1”) Pulse (0-65535 ms)

Off time ช่วงเวลาเกิด Off (“0”) Pulse (0-65535 ms)

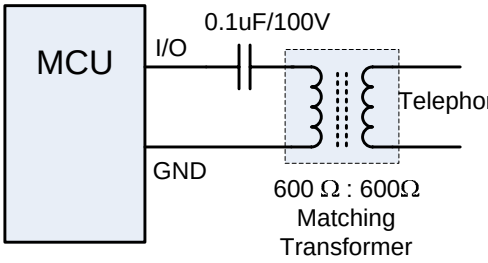
Tone เลขรหัสที่ปุ่มโทรศัพท์ (0-9)



ปกติถ้าไม่ได้กำหนดค่า On time และ Off time โปรแกรมจะกำหนดไว้ดังนี้คือ

ON Time = 200 ms OFF Time = 50 ms

ขาที่ต่อออกจากไมโครคอนโทรลเลอร์ไปยังสายโทรศัพท์จะต้องผ่าน Matching Transformer เพื่อ เป็นตัวแยกวงจรระหว่างไมโครคอนโทรลเลอร์ กับ Telephone Lines

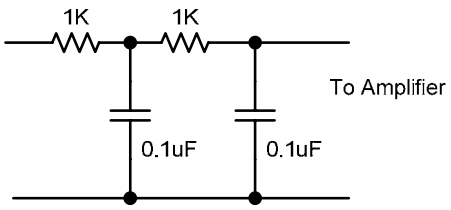


รูปที่ 34 วงจรต่อพ่วงกับสายโทรศัพท์

Ex. สร้างปุ่มกดโทรอัตโนมัติ

```
        SPK  VAR  PORTB.7
        S1   VAR  PORTB.0
        SPK  =  0  :  S1   =  1
LOOP0  :  IF  S1 = 1  THEN  LOOP0
LOOP1  :  DTMFOUT  SPK,  500, 100, [0, 2, 9, 4, 3, 6, 0, 2, 0]
LOOP2  :  IF  S1 = 0 THEN  LOOP2
          LOW  SPK
          GOTO  LOOP0
          END
```

<u>หมายเหตุ</u> ถ้าต้องการต่อเสียง Dial Tone ออกทางลำโพงต้องผ่าน Amplifier และต้องมีวงจร Low pass filter เป็นตัวกรองเอาความถี่สูงออกไปก่อน



รูปที่ 35 Low Pass Filter

2.31.32 คำสั่ง EEPROM

เป็นคำสั่งสำหรับเก็บค่าคงที่ไว้ใน ON-Chip EEPROM ในช่วงการโปรแกรมตัวชิพ

รูปแบบ EEPROM Location, [Constant, Constant, . .]

Location เป็นตำแหน่ง address เริ่มต้นมีค่าตั้งแต่ 0 – ตำแหน่งสุดท้ายของ Memory ของ EEPROM ที่อยู่ใน chip ของ M.C. แต่ละเบอร์

Constant เป็นรหัสค่าตัวเลข, อักขระที่เป็น ASCII Code.

Ex. ต้องการเก็บค่า 10, 20 และ 30 ไว้ใน EEPROM ในตัวชิพ โดยเริ่มตั้งแต่ Location 4 เป็นต้นไป EEPROM 4, [10, 20, 30]

2.31.33 คำสั่ง WRITE

เป็นคำสั่งสำหรับเขียน byte ของข้อมูลลงใน EEPROM ของตัวชิพ ช่วง Runtime

รูปแบบ WRITE Address, Value

Address เป็นตำแหน่งข้อมูลบน EEPROM ของตัวชิพ

Value เป็นข้อมูลที่เป็นตัวเลข, อักขระ เขียนได้ทีละ byte

<u>หมายเหตุ</u> คำสั่งนี้สามารถที่จะ กำหนด ค่าOn - Chip EEPROM ได้ในช่วง Runtime ซึ่งต่างกับคำสั่ง EEPROM ที่สามารถใช้งานได้เฉพาะช่วง Programming Time

Ex. ต้องการส่งค่าที่อยู่ใน B0 เข้าสู่ EPPROM ในตำแหน่งที่ 4 WRITE 4, B0
การเขียนข้อมูลเป็น WORD (ประกอบด้วย 2 byte) จะต้องกระทำการเขียนทีละ byte แยกกันดังนี้

```
Ex.     W       VAR     WORD
        WRITE 0, W.BYTE0
        WRITE 1, W.BYTE1
```

2.31.34 คำสั่ง DATA

เป็นคำสั่งสำหรับเก็บข้อมูลเข้าไปใน EEPROM ของไมโครคอนโทรลเลอร์ครั้งแรกที่ตัวชิพถูกโปรแกรมแต่ละครั้ง

รูปแบบ DATA allocation, Constant, constant,…

allocation เป็นตำแหน่ง address ที่เริ่มต้นเก็บข้อมูลถ้ามีข้อมูลหลายค่า ตำแหน่ง Address จะถูกจัดลำดับเรียงต่อกันไปโดยอัตโนมัติ

Constant เป็นค่าข้อมูลที่เป็นตัวเลข, หรือตัวอักขระ (string) ในรูปแบบของ ASCIT-Code

Ex. ต้องการเก็บข้อมูล 10, 20, และ 30 ลงใน EEPROM ของตัวชิพ โดยเริ่มจาก Location ที่ 4 เป็นต้นไป

```
DATA @4, 10, 20, 30
```

2.31.35 คำสั่ง READ

เป็นคำสั่งสำหรับอ่านข้อมูลในตำแหน่งต่าง ๆ ของ EEPROM ภายในตัวชิพมาใส่ลงตัวแปรที่กำหนดขึ้น

รูปแบบ READ Address, Var

Address เป็นตำแหน่งของข้อมูลใน EEPROM

Var เป็นตัวแปรที่กำหนดเพื่อทำข้อมูลมาเก็บไว้

Ex. ต้องการอ่านข้อมูลใน EEPROM ตำแหน่งที่ 5 มาเก็บไว้ใน B2

```
READ 5, B2
```

Ex. ต้องการอ่านข้อมูลใน EEPROM ของตัวชิพ M.C. ทีละ Word (2 byte)

```
W  VAR  WORD
READ 0,  W.BYTE0
READ 1, W.TYTE1
```

หมายเหตุ 1. คำสั่งที่เกี่ยวกับการเขียนและการอ่านข้อมูลจาก EEPROM ที่อยู่ภายในตัวชิพไมโครคอนโทรลเลอร์ ใช้ได้กับไมโครคอนโทรลเลอร์ เบอร์ที่มี EEPROM ชนิด FLASH ภาย ในตัวชิพเท่านั้น เช่น เบอร์ PIC 16F84/84A, 16F874/877 เป็นต้น

2. จำนวนตำแหน่งของ EEPROM ขึ้นอยู่กับขนาดความจุของ EEPROM ซึ่งไมโครคอนโทรลเลอร์แต่ละเบอร์จะมีขนาดไม่เท่ากัน

2.31.36 คำสั่ง DEBUG

เป็นคำสั่งสำหรับส่งข้อความออกทาง I/O Pin ที่กำหนดตามมาตรฐานการรับ-ส่ง แบบ Asynchronous โดยใช้รูปแบบ data 8bit, no parity, และ 1 stop bit (8N1) ขา Pin ที่ใช้คำสั่งนี้จะถูกกำหนดให้มีสถานะเป็น OUTPUT โดยอัตโนมัติ

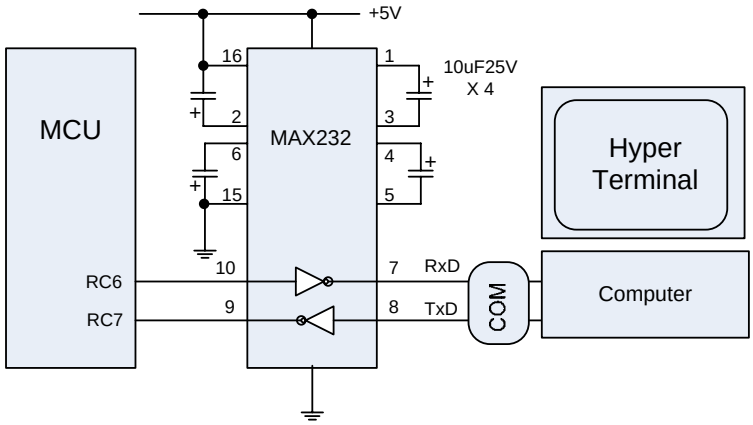
รูปแบบ DEBUG Item, {Item,..}

คำสั่งนี้สามารถจะส่งข้อมูลขณะที่ Run Program ค่าต่าง ๆ ได้ เช่น ตัวแปร, program position เป็นต้น ไปยัง Terminal Com Port ของคอมพิวเตอร์ที่ต่ออยู่

ก่อนการใช้คำสั่ง DEBUG ควรต้องกำหนดขา I/O Pin และ baud rate ให้ตรงกับที่ Computer ต่ออยู่ก่อน โดยใช้คำสั่งเหล่านี้คือ

DEFINE DEBUG REG PORTB	กำหนด PortB ใช้งาน
DEFINE DEBUG BIT	กำหนด Pin0 เป็นขาส่ง
DEFINE DEBUG BAUD 2400	กำหนด Boud rate = 2400
DEFINE DEBUG MODE 1	กำหนด Mode 1 = inverted 0 = true
DEFINE DEBUG PACING 1000	กำหนด rate การส่งตัวอักขระ 1 ms/ตัวอักขระ

* การต่อ Hardware เข้ากับ RS-232 Porter ของคอมพิวเตอร์ สามารถต่อได้ตามวงจรข้างล่างนี้



รูปที่ 36 การต่อวงจรประกอบการใช้คำสั่ง DEBUG

Ex. ต้องการส่งข้อความ “B0 =” แล้วตามด้วยตัวเลขข้าม 10 ของค่าตัวแปร B0 พร้อมทั้งรหัสการขึ้นบรรทัดใหม่ไปยัง Terminal RS-232 ของคอมพิวเตอร์

```
DEBUG “B0 =” , DEC B0 , 10
```

ตัว Modifier เป็นตัวที่กำกับข้อมูลว่าจะส่งข้อมูลไปยัง PC ในลักษณะใดตามตารางข้างล่างนี้

Modifier	Operation
BIN	ส่ง Binary
DEC	ส่ง เลขฐาน 10
HEX	ส่ง เลขฐาน 16
REP C\N	ส่งตัวอักษร : (ซ้ำจำนวน N ครั้ง)
STR Array Var\ n	ส่งตัวอักขระ จำนวนครั้งละ n ตัว

2.31.37 คำสั่ง DEBUGIN

เป็นคำสั่งที่ใช้รับตัวอักขระหรือค่าตัวเลขจากคอมพิวเตอร์ PORT Com 1 หรือ Com 2 ตามมาตรฐานของ RS-232 รูปแบบเดียวกันกับคำสั่ง DEBUG

รูปแบบ DEBUGIN {Timeout , Label} , [Item , Item….]

คำสั่ง DEUGIN ใช้งานตรงกันข้ามกับคำสั่ง DEBUG ดังนั้นก่อนการใช้คำสั่งต้องกำหนดค่ามาตรฐานการส่งข้อมูล , ขา Pin , Port ที่ต้องใช้งานก่อนดังนี้ เช่น

```
DEFINE    DEBUGIN_REG   PORTB
DEFINE    DEBUGIN_BIT           0
DEFINE    DEBUG_BAUD            2400
DEFINE    DEBUGIN_MODE          1  0=true , 1= inverted
```

Timeout หากกำหนดมีค่า 1 การ โดยโปรแกรมจะ Idle รอการรับข้อมูลหากครบค่า Timeout โปรแกรมจะออกจากคำสั่ง DEBUGIN และจะ Jump ไปทำงานที่ Label ที่กำหนดทางโปรแกรม

***** การต่อวงจร Hardware เพื่อรอรับตัวอักขระหรือข้อมูลจากคอมพิวเตอร์ ผ่าน Port com 1 หรือ com 2 ใช้วงจรตามรูปที่ 36

Ex. 1 โปรแกรมรอจนกว่าจะได้รับตัว “A” และจะทำตัวอักขระถัดไปใส่ใน B0

```
DEBUGIN   WAIT (“A”),B0
```

Ex. 2 ขยับไป 2 ตัวและจับ 4 ตัวถัดไปที่เป็นตัวเลขฐาน 10 ใส่ B0

```
DEBUGIN  SKIP2 , DEC4  B0
```

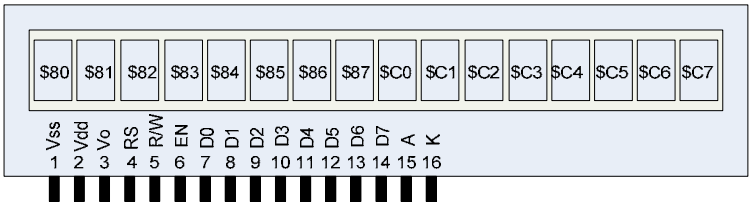
ตัว **Modifier ที่ใช้กำกับการรับตัวอักขระหรือตัวเลขมีดังต่อไป**

Modifier	Operation
Bin	รอรับเลขฐาน2 (Binary)
DEC	รอรับเลขฐาน10
HEX	รอรับเลขฐาน16

SKIP n	ข้ามไป n ตัว ที่รับไปแล้ว
STR Array Var \ n	รับอักขระ n ตัว
WAIT ( )	รอจนกว่าจะรับตัวอักขระในวงเล็บ
WAITSTR Array Var \ n	รอจนกว่าจะรับข้อความจน n ตัว

2.31.38 คำสั่ง LCDOUT

เป็นคำสั่งสำหรับส่งข้อความออกที่จอ LCD คำสั่งนี้ Support LCD Module แบบ 2 บรรทัด หรือ 1 บรรทัดที่ใช้ตัว Controller ของ Hitachi เบอร์ 44780 หรือเบอร์อื่นที่เข้ากันได้ รายละเอียดนอกเหนือจากนี้ ให้ศึกษาเพิ่มเติมในใบทดลองการแสดงผลทางจอ LCD จอ LCD โมดูลโดยทั่วไปมีขั้วต่อ 14-16 ขั้วแบบ Single row header โดยมีขาต่อดังนี้



Vss = GND

Vdd = +5 V

Vo = Brightness Control

RS = Register Select

R/w = R/w register

E = Enable

รูปแบบคำสั่ง LCDOUT Item ,{Item, …}

คำสั่งที่ใช้ Initialize LCD กรณีที่เกิดไฟดับกลางคัน ขณะที่มีการส่งข้อมูลไปยัง LCD คำสั่งที่ต้องส่งไปให้ MCU คือการ clear flag คือ

FLAGS = 0

ส่วนคำสั่งต่าง ๆ ที่ใช้ควบคุม LCD แสดงผลตามต้องการจะต้องเริ่มต้นด้วย $FE แล้วตามด้วย Code คำสั่งต่อไปนี้

Command	Operation
$FE , 1	Clear Display
$FE , 2	Return home (beginning of first line)
$FE , $0C	Cursor off
$FE , $0E	Underline cursor on
$FE , $0F	Blinking cursor on
$FE , $10	Move cursor left on position
$FE , $14	Move cursor right one position
$FE , $C0	Move cursor to beginning of 2nd line
$FE , $94	Move cursor to beginning of 3rd line
$FE , $D4	Move cursor to beginning of 4th line

Ex. 1 LCDOUT $FE , 1, “Hello”

Ex. 2 LCDOUT $FE , $C0 , “World”

Ex. 3 LCDOUT B0 , # B1

LCD โมดูลสามารถต่อกับไมโครคอนโทรลเลอร์ได้ทั้ง 4 bit bus และ 8-bit bus โดยมีเงื่อนไขการต่อดังนี้

1. ถ้าต่อแบบ 8-bit bus Data bus ทั้งหมดต้องต่อยู่ port เดียวกัน
2. ถ้าต่อแบบ 4-bit bus Data bus ทั้ง 4 เส้นต้องต่ออยู่ไม่ 4-bit ล่าง หรือก็ 4-bit บนของ post
3. บน Enable และ Register Select จะต้องอยู่ที่ขั้วไหนของ port ก็ได้
4. ขา R/W ถ้าไม่ใช้คำสั่ง LCDIN ให้ต่อลง GROUND

Pic Basic Pro ได้กำหนดค่า Default ไว้ดังนี้คือ ใช้ M.C. เบอร์ PIC 16 F8 4A , ขา Data D4 ~ D7 ต่อกับ porta.0~Porta03 Resister Select ต่อกับ Port A.4 และขา Enable ต่อกับ Port B.3 , และได้กำหนดค่า initialize LCD ไว้ให้แสดง 2 บรรทัด PORT B.3 , แนะได้กำหนดค่า initialize LCD ไว้ให้แสดง 2 บรรทัด

***** ถ้าต้องการ เปลี่ยนตำแหน่งที่นอกเหนือจากนี้ต้องใช้คำสั่ง DEFINE กำหนดดังนี้ คือ :

```
‘Set LCD Data port
DEFINE LCD_DREG PORTB
‘Set starting Data bit (0 or 4) if 4-bit bus
DEFINE LCD_DBIT 4
```

```
‘Set  LCD  Register  Select  Port
DEFINE LCD_RSREG  PORTB
‘Set  LCD  Register  Select  Bit
DEFINE  LCD_RSBIT  1
‘Set  LCD Enable  Port
DEFINE  LCD_EREG  PORTB
‘Set  LCD  Enable  Bit
DEFINE  LCD_EBIT  0
‘Set  LCD Bus  Size (8 or 4 bits)
DEFINE  LCD_BITS  4
‘Set  number  of Lines on LCD
DEFINE  LCD_LINES  2
‘Set  data  delay  time in μs
DEFINE  LCD_DATAUS  50
‘Set  Command  delay  time  in  μS
DEFINE  LCD_COMMANDUS  2000
```

2.31.39 คำสั่ง LOOKDOWN

เป็นคำสั่งค้นหาค่า Constant ขนาด 8-bit โดยค้นหาให้ match กับค่ากำหนดโดย : ถ้าค้นหาพบโปรแกรมจะเก็บค่าลำดับที่ค้นเจอะตัวแรกไว้ใน Variable โดยลำดับจะเรียงตั้งแต่ 0 เป็นตัวแรก, 1 เป็นตัวถัดมา และ 2, 3…ไปจนถึง 255 ตัว แต่ถ้าค้นหาไม่พบตัวที่ Match ก็จะไม่มีการเปลี่ยนแปลงในตัว Variable

รูปแบบ LOOKDOWN SEARCH , [Constant , Constant,..],Var

Search	เป็นค่าที่ต้องการค้นหา
Constant	เป็นค่าคงที่ใน List ที่เป็นได้ทั้งตัวเลข (Numeric) และ string
Var	เป็นตัวแปรสำหรับเก็บค่า index หรือลำดับของตำแหน่งค่าคงที่ที่ match กับค่าที่ต้องการค้นหา

```
Ex.  SERIN  PORTA.1, N2400, B0                  ‘รับค่า hex. char. จาก Port ขา1 ทางอนุกรม
     LOOKDOWN B0, [“0123456789ABCDEF”], B1      ‘ค้นหาค่าคงที่ที่ Match กับ B0
                                                ‘เก็บค่า Index (ลำดับ) ไว้ที่ B1
     SEROUT  PORTA.0, N2400, [#B1]              ‘ ส่งค่าลำดับ B1 ไปยัง PORTA ขา 0 แบบอนุกรม
```

2.31.40 คำสั่ง LOOKDOWN2

เป็นคำสั่งที่ค้นหาค่าที่เป็น Value ที่ตรงกับเงื่อนไขค่าที่กำหนด ถ้าพบจะเก็บค่าลำดับไว้ในตัวแปร (Var) ถ้าเป็นค่าแรกใน List ค่าลำดับจะมีค่าเป็น 1,2,…ตามลำลับ ถ้าค้นหาไม่พบค่าตัวแปรจะไม่เปลี่ยนแปลง

รูปแบบ LOOKDOWN2 Search , {Test} [Value, Value,…], Var

Search เป็นค่าที่ต้องการค้นหาค่าใน List

Test เป็น option สำหรับเป็นเงื่อนไขเปรียบเทียบระหว่างค่า Search กับ Value ใน List ว่าจะ match กันในลักษณะไหน ซึ่งมีเครื่องหมายที่ใช้ได้แก่

= หมายถึง เท่ากัน

> หมายถึง ค่า Search มากกว่าค่า Value

< หมายถึง ค่า Search น้อยกว่าค่า Value

Value เป็นค่าตัวเลข หรือตัวแปรที่เป็นตัวเลข หรือค่าที่เป็นอักขระ ทั้งขนาด 8-bit และ 16-bit จำนวน Value ใน List มีได้ถึง 85 ตัว

หมายเหตุ คำสั่ง LOOKDOWN2 กับ คำสั่ง LOOKDOWN ตรงที่

คำสั่ง LOOKDOWN2 สูงมาก ค้นหา string หรือ Value ใน List ได้มากกว่า 8 bit

แต่ถ้า ค้นหา string หรือ Value ที่มีขนาด 8-bit ควรใช้คำสั่ง **LOOKDOWN**

Ex.
```
LOOKDOWN2   W0 [512,W1,1024] , B0
LOOKDOWN2   W0 < [10 , 100 , 1000] , B0
```

2.31.41 คำสั่ง LOOKUP

เป็นคำสั่งที่นำค่าคงที่ขนาด 8-bit ใน List ตามตำแหน่งที่ระบุในค่า Index ไปกำหนดเป็นค่าตัวแปร Var

รูปแบบ LOOKUP Index , [Constant , Constant,..] , Var

Index เป็นค่าตำแหน่งของค่า constant ใน List มีได้ตั้งแต่ 0 ~ 255

Constant เป็นค่า numeric , หรือ string

Var เป็นที่เก็บค่า Constant ตามตำแหน่งของ Index

Ex.
```
For   B0 = 0    TO 5  'นับจาก 0 ถึง 5
    LOOKUP  B0,  ["Hello!"] ,B1          'นำตัวอักขระตามตำแหน่ง B0 ไปใส่ใน B1
    SEROUT  PORTA 0 , N2400 , [B1]    'ส่งค่า B1 ออก PortA ขา 0 อนุกรม
Next  B0
```

หมายเหตุ ค่า Index มีค่ามากกว่าจำนวนค่า Constant ใน List จำนวนที่เกินจะไม่ทำให้ตัวแปร Var เปลี่ยนค่า

2.31.42 คำสั่ง LOOKUP2

เป็นคำสั่งที่ทำงานเช่นเดียวกับคำสั่ง LOOKUP แต่สามารถยอมให้ค่าคงที่หรือตัวแปรหรือ Value ใน List มีขนาดได้ถึง 16-bit

รูปแบบ LOOKUP2 Index , [Valve ,Valve,…] , Var

Index เป็นค่าชี้ตำแหน่งของ Value มีค่าระหว่าง 0~85
Valve เป็นค่าตัวเลข , ตัวแปร , อักขระ ที่มีได้ทั้ง 8-bit และ 16-bit (มีได้ไม่เกิน 85 ตัว)

Ex. LOOKUP B0 , [256 , 512 , 1024], W1

หมายเหตุ ทั้งคำสั่ง LOOKDOWN2 และคำสั่ง LOOKUP2 เมื่อ Generate code ออกมาแล้วจะมีขนาดใหญ่กว่าขนาดของคำสั่ง LOOKDOWN และ LOOKUP ประมาณ 3 เท่า ดังนั้นถ้าจะเขียนโปรแกรมเพื่อที่จะทำงานกับข้อมูล 8-bit ควรจะใช้คำสั่ง LOOKUP และ LOOKUP

2.31.43 คำสั่ง SERIN

เป็นคำสั่งรับข้อความ/ข้อมูล ที่ขาทาง pin ที่กำหนดในรูปแบบอะซิงโครนัส 8-bit, no parity และ 1 stop bit (8N1)

รูปแบบ SERIN Pin , Mode , [Qual,…] , {Var }

pin เป็นขาทาง Port เช่น Porta.0
Mode เป็นชื่อวิธีการรับ/ส่งข้อมูลได้ถูกกำหนดไว้ในโปรแกรม MODEDEFS.BAS ดังนั้นก่อนใช้คำสั่ง SERIN ต้อง include file นั้นไว้บนหัวโปรแกรมก่อนดังนี้

INCLUDE “MODEDEFS.BAS”

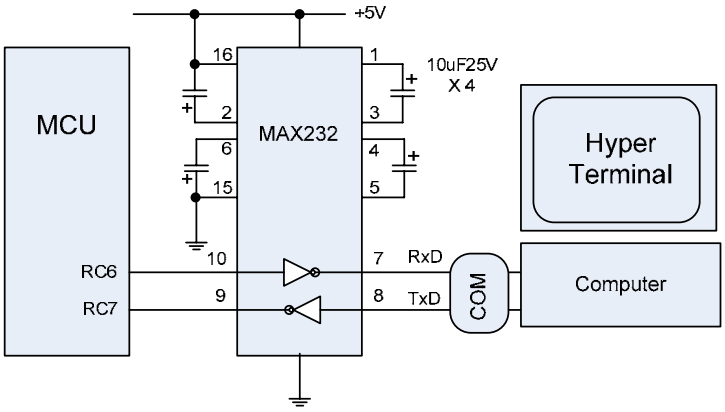
เมื่อ include file ดังกล่าวไว้ในโปรแกรมแล้วไม่ต้อง include ไฟล์ BS1DEFS.BAS และ BS2DEFS.BAS เนื่องจาก MODEDEFS.BAS ได้นิยาม Mode ไว้ครอบคลุมหมุดแล้ว ดังนี้

Mode	Mode No.	Baud Rate	State
T2400	0	2400	
T1200	1	1200	True
T9600	2	9600	

T300	3	300	
N2400	4	2400	INVERTED
N1200	5	1200	
N9600	6	9600	
N300	7	300	

Qualifier เป็นค่าคงที่ , ตัวแปร หรือตัวอักขระ ถ้าค่าดังกล่าวมีเครื่องหมาย # อยู่ข้างหน้า คำสั่ง SERIN จะเปลี่ยนจากค่า decimal value เป็น ASCII และเก็บค่าไว้ใน Variable นั้น

<u>หมายเหตุ</u> 1. ถ้าไม่กำหนดค่าใด ๆ ไว้คำสั่งนี้จะถือว่าไมโครคอนโทรลเลอร์ใช้ความถี่ OSC 4
แต่ถ้าใช้ OSC นอกเหนือจากนี้ต้องกำหนดนิยามก่อน



รูปที่ 37 วงจรสำหรับการใช้คำสั่ง SERIN และ SEROUT

Ex.

SERIN PORTA.1 , N2400 , [“A”] , B0 ‘รอจนกว่าจะได้รับตัว “A” ที่ pin1 PortA และนำค่าตัวอักขระต่อไปใส่ B0

2.31.44 คำสั่ง SEROUT

เป็นคำสั่ง ข้อความ/ข้อมูล ออกจากขา I/O pin ของ port แบบอะซิงโครนัส แบบ 8-bit data, no parity one-stop (8N1)

รูปแบบ SEROUT Pin , Mode , [Item , Item ,…]

pin เป็นขาของ port เช่น porta.0

Mode เป็นรูปแบบการรับ-ส่งข้อมูลแบบอนุกรมตามตารางข้างล่างนี้

Mode	Mode No.	Baud Rate	State
T2400	0	2400	True
T1200	1	1200	
T9600	2	9600	
T300	3	300	
N2400	4	2400	INVERT
N1200	5	1200	
N9600	6	9600	
N300	7	300	

Item เป็นข้อความซึ่งคำสั่ง SEROUT SUPPORT ข้อความหรือข้อมูลทั้ง 3 แบบด้วยกัน คือ

1. ตัวอักขระ (String Constant)

2. ค่าตัวเลข (ที่มีทั้งตัวเลขค่าคงที่และตัวแปร)จะถูกส่งออกในรูปของ ASCIT ตามด้วย char. 13 (คือ บรรทัด) และ char 10 (ขึ้นบรรทัดใหม่) ที่ละตัวแบบ ASCIT ของเลขฐาน 10

3. ค่าตัวเลขถ้านำหน้าด้วยเครื่องหมาย # ข้อความนั้นจะถูกส่งทีละตัวแบบ ASCII ของเลขฐาน 0

หมายเหตุ 1. คำสั่ง SEROUT ถ้าไม่กำหนดนิยามใด ๆ ไว้ล่วงหน้าจะถือว่าไมโครคอนโทรลเลอร์ OSC 4 MHz แต่ถ้าใช้ ความถี่นอกเหนือจากนี้ ต้องกำหนดนิยาม (DEFINE) ของ OSC ไว้ด้วย

2. ในบางกรณีความเร็วของฮาร์ดแวร์ที่รองรับการส่งข้อมูลของไมโครคอนโทรลเลอร์ช้ากว่าตัวส่งจะทำให้การส่งข้อมูลสะดุดและมีปัญหาได้ PIC BASIC ได้กำหนดคำสั่งสำหรับกำหนดความเร็วในการส่งข้อมูลระหว่างตัวอักขระแต่ละตัวไว้ดังนี้

3. ถ้าต้องการให้หยุดช่วงเวลา 1 mS ระหว่างการส่งอักขระแต่ละตัว ดังนี้

```
DEFINE  CHAR_PACING 1000
```

Ex. SEROUT PORT.0 , N2400 , [#B0 ,10]

(ส่ง ASCIT Value ของ B0 แล้วตามด้วย ขึ้นบรรทัดใหม่ที่ PORTB.0)

2.31.45 คำสั่ง SHIFTIN

เป็นคำสั่งรับข้อมูลเข้าทางขา Pin ของ Port อนุกรมแบบ Synchronous เข้าไปเก็บไว้ในตัวแปร Var ตามจังหวะสัญญาณ Clock

รูปแบบ SHIFTIN Data Pin , clock Pin , Mode , [Var \ bits..]

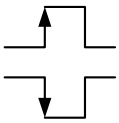
Data Pin	ขา Pin สำหรับรับข้อมูล (Data)
Clock Pin	ขา Pin สำหรับรับสัญญาณ clock
Mode	เป็นชื่อ Mode ในการรับข้อมูล ก่อนใช้ Mode จะต้อง Include File

เข้าไว้ที่หัวของโปรแกรมดังนี้

INCLUDE “MODEDEFS.BAS”

เมื่อ Include File ตามข้างบนนี้แล้ว **ไม่จำเป็นต้อง include** ไฟล์ BS1DEFS.BAS และ BS2DEFS.BAS อีก

Mode ในการรับข้อมูลมีทั้งหมด 8 Mode โดย

Mode ที่ 0 ~ 3 จะรับสัญญาณ clock แบบ Idles Low คือ 

Mode ที่ 4 ~ 7 จะรับสัญญาณ clock แบบ Idles high คือ

Var เป็นชื่อตัวแปรที่เก็บข้อมูลที่ SHIFT เข้ามา

\Bits เป็นตัวกำหนดว่ารับข้อมูลที่ละกี่บิท **ถ้าไม่กำหนด** ถือว่ารับที่ละ 8 บิท

Mode	Mode No.	Operation
MSBPRE	0	- ต้องส่งข้อมูลบิตนัยสำคัญสูงเข้ามาก่อน - ต้องส่งข้อมูลเข้ามาก่อน , แล้ว clock ตาม - clock เป็นแบบ Idle Low
LSBPRE	1	- ต้องส่งข้อมูลบิตนัยสำคัญต่ำเข้ามาก่อน - ต้องส่งข้อมูลเข้ามาก่อน , แล้ว clock ตาม - clock เป็นแบบ Idle Low
MSBPOST	2	- ต้องส่งบิตนัยสำคัญสูงเข้ามาก่อน - ส่ง clock เข้าก่อน , แล้วส่ง Data ตาม - clock เป็นแบบ Idle Low
LSBPOST	3	- ต้องส่งบิตนัยสำคัญต่ำเข้ามาก่อน - ส่ง clock เข้าก่อน , แล้วส่ง Data ตาม - clock เป็นแบบ Idle Low
	4	- ส่งข้อมูลบิตนัยสำคัญสูงเข้าก่อน - ส่งข้อมูลเข้าก่อน , ส่ง Data ตาม - clock แบบ Idle High
	5	- ส่งข้อมูลบิตนัยสำคัญต่ำเข้าก่อน - ส่งข้อมูลเข้าก่อน , ส่ง Data ตาม - clock แบบ Idle High
	6	เหมือนกับ Mode 2 แต่ clock แบบ Idle High
	7	เหมือนกับ Mode 3 แต่ clock แบบ Idle High

<u>หมายเหตุ</u> สัญญาณ clock ในการ SHIFT ข้อมูลเข้ามีความถี่ประมาณ 50 KHz ที่สัญญาณ clock ของไมโครคอนโทรลเลอร์ 4 MHz แต่เราสามารถกำหนดความเร็วในการรับข้อมูลแต่ละบิทได้ โดยกำหนดค่าหน่วยเวลาในการเลื่อนข้อมูลได้ตั้งแต่ 0~65535 mS โดยกำหนดด้วยคำสั่งดังนี้ คือ

```
DEFINE  SHIFT_PAUSEUS  100
```

จากคำสั่ง DEFINE ข้างบนกำหนดค่าหน่วงเวลาในการ SHIFT ข้อมูลแต่ละบิท = 100 ไมโครวินาที

Ex. ต้องการรับข้อมูลแบบอนุกรม ซิงโครนัส เข้าไว้ในตัวแปร B0 ทีละ 4 บิท โดยรับข้อมูลบิตนัยสำคัญสูงเข้าก่อน ใช้ขา PORTA.0 เป็นขารับข้อมูล และขา PORTB.0 เป็นขารับสัญญาณ clock

```
SHIFT  PORTA.0 , PORTA.1 , MSBPRE , [B0\4]
```

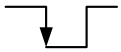
2.31.46 คำสั่ง SHIFTOUT

เป็นคำสั่งสำหรับเลื่อนข้มูลออกจากไมโครคอนโทรลเลอร์อนุกรมแบบซิงโครนัส

รูปแบบ SHIFTOUT Data , Clock Pin , Mode , [Var\Bits]

Data Pin	เป็นขาที่ส่งข้อมูลออก
VAR	ตัวแปรที่ส่งข้อมูลออก
\Bits	จำนวนบิตในการส่งข้อมูลออก
Clock Pin	เป็นขาที่ส่งสัญญาณ clock ออก
Mode	เป็นรูปแบบของการส่งข้อมูลออกโดยจะต้อง include file ต่อไปนี้ที่หัวโปรแกรมก่อน คือ INCLUDE 'MODEDEFS.BAS'

สำหรับ **Mode 0-1** ใช้กับสัญญาณ clock แบบ Idle Low 

สำหรับ **Mode 4-5** ใช้กับสัญญาณ clock แบบ Idle High 

Mode	Mode No.	Operation
LSBFIRST	0	- เลื่อนข้อมูลนัยสำคัญต่ำออกก่อน - clock แบบ Idle Low
MSBFIRST	1	- เลื่อนบิตข้อมูลนัยสำคัญสูงออกก่อน - clock แบบ idle Low
	4	- เลื่อนบิตข้อมูลนัยสำคัญต่ำออกก่อน - clock แบบ idle high
	5	- เลื่อนบิตข้อมูลนัยสำคัญสูงออกก่อน - clock แบบ idle High

ความเร็วในการเลื่อนข้อมูลของ Clock ประมาณ 50 KHz ทั้งนี้ขึ้นอยู่กับสัญญาณ Clock ของไมโครคอนโทรลเลอร์ แต่เราสามารถใช้คำสั่งหน่วงเวลา Clock สำหรับ SHIFT ข้อมูลออกได้ดังนี้

DEFINE SHIFT_PAUSEUS 100

Ex.

SHIFTOUT PORTA.1 , PORTA.2 , 1, [Wardvar\4]

2.31.47 คำสั่ง I2CREAD

เป็นคำสั่งสำหรับอ่านข้อมูลจากไอซีชิพประเภทรับ - ส่ง ข้อมูลอนุกรม 2 สาย แบบ I^2C-bus ไอซีประเภทนี้ได้แก่ Memory , Digital I/O , A/D Converter , Temperature Sensor เป็นต้น

รูปแบบ I2CREAD DataPin , clockPin , Control , [Var]

DataPin เป็นขาที่กำหนดให้รับข้อมูล (SDA)

ClockPin เป็นขาที่กำหนดให้รับสัญญาณ clock (SCL)

Control byte เป็น byte ควบคุมในการติดต่อกับชิพประเภท I^2C ซึ่งชิพแต่ละตัวแต่ละชนิดจะมี Control byte ค่าต่างกัน (ดูจากตารางตัวอย่างของไอซีหน่วยความจำแบบ EEPROM)

Var เป็นซึ่งตัวแปรสำหรับเก็บค่าที่อ่านเข้ามาได้

ช่วงเวลาในการรับ-ส่งข้อมูลแบบ I^2C มีมาตรฐานอยู่ที่ความเร็ว 100 ซึ่งจะทำงานกับวงจรที่ใช้ clock 8 ช่วงเวลารับ-ส่งข้อมูลแบบ Fast mode อยู่ที่ความเร็ว 400 แต่วงจรต้องใช้ระบบ clock 20 แต่ถ้าต้องการใช้ความเร็วมาตรฐาน (100 KHz) ต้องกำหนดนิยามต่อไปนี้

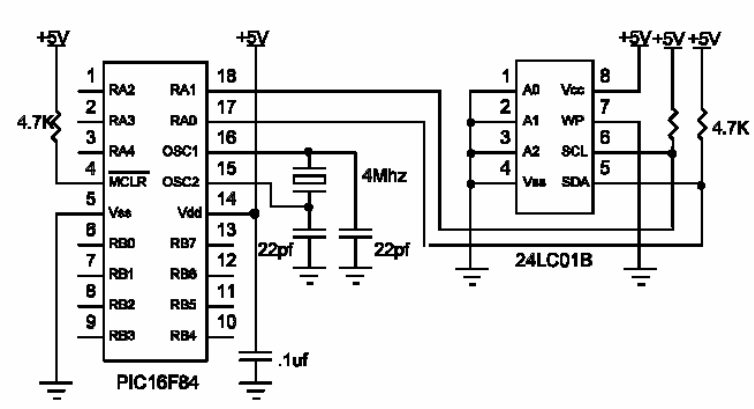
DEFINE I^2C_SLOW1

หมายเหตุ ในการต่อวงจรใช้งานที่สายรับ-ส่งข้อมูล (SDA) และสาย clock (SCL) ต้องต่อ R ค่า 4.7 K Pull-up ไว้ด้วย เนื่องจากวงจรเป็นแบบ Open-collector

Device	Capacity	Control	Address size
24LC01B	128 bytes	%1010xxx0	1 byte
24LC02B	256 bytes	%1010xxx0	1 byte
24LC04B	512 bytes	%1010xxb0	1 byte
24LC08B	1K bytes	%1010xbb0	1 byte
24LC16B	2K bytes	%1010bbb0	1 byte
24LC32B	4K bytes	%1010ddd0	2 bytes
24LC65	8K bytes	%1010ddd0	2 bytes

bbb = block select (high order address) bits
ddd = device select bits
xxx = don't care

รูปที่ 38 ตัวอย่างชิพหน่วยความจำแบบ I^2C



รูปที่ 39 ตัวอย่างวงจรสำหรับติดต่อกับไอซีหน่วยความจำ แบบ I^2C

ตัวอย่าง การเขียนโปรแกรม

```
ADDR   VAR   BYTE
CONT   CON     %10100000
       ADDR  =17
       I2CFREAD   PORTA.0 , PORT.1 , CONT , ADDR, [B2]
```

จากโปรแกรมตัวอย่างเป็นการอ่านข้อมูลจาก IC ชิพ หน่วยความจำ EEPROM I^2C เบอร์ 24LC01B ซึ่งมี Control Code % 10100000 เป็นชนิดของชิพได้แก่ EEPROM และตามด้วยรหัส Chip Select ซึ่งรวมกันจะได้ % 10100000 หรือ $A0 (รูปแบบของ control byte ดูได้จากคู่มือของไอซีประเภท I^2C)

2.31.48 คำสั่ง I2CWRITE

เป็นคำสั่งสำหรับส่งข้อมูลจากไมโครคอนโทรลเลอร์ไปยังชิพ I^2C ซึ่งเป็นคำสั่งตรงกันข้ามกับคำสั่ง I2CREAD

รูปแบบ I2CWRITE DataPin , ClockPin , control , { Address} , [Value , {Value…}]

DataPin	เป็นขารับ-ส่งข้อมูล (SDA)
ClockPin	เป็นขาส่ง Clock (SCL)
Control	เป็นค่า Control byte ของชิพ I^2C
Address	ถ้ามีเป็นขนาดว่าจะส่งเป็น byte (8-bit) หรือ word (16-bit)
Value	เป็นค่าข้อมูลที่จะส่ง ถ้ากำหนดส่งเป็น word (2 byte) จะถูกส่งไปที่สูงออกก่อนแล้วตามลำดับ byte

ในการทำให้คำสั่ง I2CWRITE สามารถส่งข้อความ (strings) ที่ประกอบด้วยหลาย byte ด้วยคำสั่งครั้งเดียว ทำได้ดังนี้คือ

- กำหนดตัวแปรเป็น Array ที่มีจำนวน byte พอที่จะเก็บข้อความ
- ใช้คำสั่งในการเขียนข้อมูลลงชิพหน่วยความจำ EEPROM แบบ I^2C ดังนี้

ตัวอย่าง

```
a VAR byte[8]
I2CWRITE PORTC.4 , PORTC.3 , $A0 , 0 , [STR a\8]
```

Ex. เขียนข้อมูลลง EEPROM แบบ I^2C

```
Addr    VAR   byte
Cont    Con % 10100000
        addr = 17          'Set address to 177
        I2CWRITE PORTA.0 , PORTA.1 , CONT , addr , [6]
        PAUSE 10           'รอเวลา 10 mS สำหรับการเขียนข้อมูลให้เสร็จ
        Addr = 1           'set
        I2CWRITE PORTA.0 , PORTA.1 , CONT , addr , [B2]
        PAUSE 10           'รอเวลาเขียนค่า B2 ให้เสร็จ
```

2.31.49 คำสั่ง FOR…NEXT

เป็นคำสั่งสำหรับสร้างลูป การทำงานซ้ำแบบมีเงื่อนไขการจบโดยใช้ตัวแปรกำหนดค่าจำนวนครั้งเริ่มต้น และจะเพิ่ม/หรือลดค่าทุกครั้งที่มีการวนลูปมาทำงานซ้ำ จนกว่าตัวแปรจะมีค่าสุดท้ายที่กำหนด จึงจะหลุดออกจากลูป และทำงานตามโปรแกรมต่อไป

รูปแบบ

```
FOR   ตัวแปร = ค่าเริ่มต้น   TO   ค่าสุดท้าย  {STEP Val}
   . . .
   งานที่ทำซ้ำ
   . . .
NEXT   ตัวแปร
```

ตัวแปร	เป็นค่าที่กำหนดจำนวนรอบที่ทำซ้ำ
ค่าเริ่มต้น	เป็นค่าที่เริ่มต้นนับทำซ้ำ
ค่าสุดท้าย	เป็นค่าที่กำหนดทำซ้ำถึงสุดท้าย
STEP Val	เป็นค่าที่กำหนดให้ตัวแปรลดหรือเพิ่มค่าแต่ละครั้ง จากค่าเริ่มต้น จนถึงค่าสุดท้าย ถ้าไม่กำหนดจะกำหนดค่าเพิ่มทีละ 1
NEXT	เป็นคำสั่งปิดท้ายเพื่อให้ย้อนกลับไปทำซ้ำ

Ex. I VAL BYTE

```
FOR I = 1 To 5
        HIGH PORTB.5
        PAUSE 500
        LOW PORTB.5
        PAUSE 500
NEXT I
```

2.31.50 คำสั่ง IF. . THEN (แบบที่ 1)

เป็นคำสั่งสำหรับตรวจสอบเงื่อนไขเพื่อกำหนดทิศทางการทำงานของโปรแกรม

รูปแบบ **IF** เงื่อนไขที่ตรวจสอบ **THEN** งานที่ทำเมื่อเงื่อนไขเป็นจริง

เงื่อนไขที่ตรวจสอบ	เป็นสมการ (Expression) ที่เป็นปริมาณหรือ Logic
งานที่ทำเมื่อเงื่อนไขเป็นจริง	เป็นคำสั่งเมื่อการทดสอบเงื่อนไขเป็นจริง ถ้าไม่เป็นจริงโปรแกรมจะผ่านไปทำคำสั่งถัดไป

Ex.
```
B = B + 1
IF (B >= 25) THEN PORTB = 1
PORTB = 4
```

(จากตัวอย่างการตรวจสอบ ถ้า B มากกว่าหรือเท่ากับ 25 ค่า PORTB = 1 แต่ถ้าไม่ใช่ PROTB = 4)

2.31.51 คำสั่ง IF. . THEN (แบบที่ 2)

คำสั่ง IF . . THEN . . ELSE . . .ENDIF

เป็นคำสั่งสำหรับตรวจสอบเงื่อนไข เพื่อกำหนดทิศทางการทำงานของโปรแกรมแบบมีทางเลือกมากกว่า 2 ทิศทาง

รูปแบบ
```
IF    เงื่อนไขที่ตรวจสอบ    THEN
        งานที่ทำเมื่อเงื่อนไขเป็นจริง
ELSE
        งานที่ทำเมื่อเงื่อนไขเป็นเท็จ หรือนอกเหนือจากนั้นทั้งหมด
ENDIF
```

Ex. **IF** B0 < > 10 **THEN**

B0 = B0 + 1

B1 = B1 – 1

ENDIF

IF B0 = 20 **THEN**

LED = 1

ELSE

LED = 0

ENDIF

2.31.52 คำสั่ง SELECT CASE

เป็นคำสั่งสำหรับตรวจสอบเงื่อนไขตัวค่าตัวแปรหลายค่า ที่สามารถกำหนดการทำงานของโปรแกรมได้หลายทิศทาง

รูปแบบ

```
SELECT CASE Var
        CASE Expression 1
                Statement
        CASE Expression 2
                Statement
        CASE ELSE
                Statement
END SELECT
```

Ex. SELECT CASE X

CASE 1

Y = 10

CASE 2, 3

Y = 20

CASE IS > 5

Y = 100

CASE ELSE

Y = 0

END SELECT

2.31.53 คำสั่ง WHILE . . WEND

เป็นคำสั่งสำหรับทำซ้ำภายใต้เงื่อนไขที่ตรวจสอบเป็นจริง หากเงื่อนไขเป็นเท็จจะหลุดออกจากลูปทำซ้ำ

รูปแบบ **WHILE** Condition

.

Statement

:

WEND

คำสั่ง WHILE .. WEND จะทำการตรวจสอบเงื่อนไข ก่อนทำ Statement

Ex.
```
I = 1
WHILE  i <= 10
        SEROUT 0, N 2400, [“NO”, #i, 13, 10]
        i = i + 1
WEND
```

2.31.54 คำสั่ง OWIN

เป็นคำสั่งสำหรับรับข้อมูลจากอุปกรณ์ขาเดียว ได้แก่ ตัววัดอุณหภูมิ และหน่วยความจำ EEPROM หรือ RAM เป็นต้น อุปกรณ์ขาเดียวได้แก่ DS1820 Temperature sensor

รูปแบบ **OWN** Pin, Mode, [Item . .]

Pin เป็นขาของ Port MCU

Mode เป็น Mode การทำงานของอุปกรณ์ขาเดียว ซึ่งมีอยู่ 3 Mode คือ

Mode 0 1 = ส่ง Reset pulse ก่อน Data

Mode 1 1 = ส่ง Reset pulse หลัง Data

Mode 2 0 = byte-sized data, 1 = bit sized data

Ex. OWIN PORTC.0, 0, [Temp\2, Skip 4, Byte1, byte2]

จากตัวอย่าง เป็นคำสั่งสำหรับรับข้อมูลจากอุปกรณ์ขาเดียวจาก PortC ขา 0 ด้วย No reset pulse ขณะส่ง โดยได้รับทีละ 2 byte ใส่ตัวแปร Array ชื่อ Temp มี 2 byte และขยับข้ามไปอีก 4 byte แล้วอ่าน 2 byte สุดท้ายเข้าตัวแปร byte1 และ byte2

2.31.55 คำสั่ง OWOUT

เป็นคำสั่งสำหรับส่งข้อมูลออกจากขา I/O Pin ไปยังอุปกรณ์ขาเดียวภายนอก

รูปแบบ **OWOUT** Pin, Mode, [item . .]

Pin เป็นขาของ Port

Mode เป็น Mode การทำงานได้แก่

Mode0 ส่ง Reset pulse ก่อนส่ง Data

Mode1 ส่ง Reset pulse หลังส่ง Data

Ex. OWOUT PORTC.0, 1, [$CC, $be]

จากตัวอย่างเป็นการส่งข้อมูลไปยังอุปกรณ์ขาเดียวทาง PORTC ขา 0 โดยส่ง Reset pulse แล้วตามด้วย byte $CC และ byte $be

2.31.56 คำสั่ง GOSUB . . RETURN

เป็นคำสั่งให้โปรแกรมกระโดดไปทำงานที่ Subroutine เสร็จแล้วกลับมาที่ยังค้างอยู่ในโปรแกรมหลักด้วยคำสั่ง RETURN

รูปแบบ

GOSUB ชื่อ Label

```
Ex. MAIN : B = 4
           GOSUB  DELAY
           B = B >> 1
           PORTB = B
           GOTO  MAIN
           END
    DELAY : PAUSE  500
           RETURN
```

2.31.57 คำสั่ง SOUND

รูปแบบ **SOUND** Pin, [Note, Duration {, Note, Duration, .. }]

เป็นคำสั่งกำเนิดสัญญาณ Tone และสัญญาณ White noise ผ่านขา Pin ที่ระบุ

Note มีค่า 1 – 127 จะเป็นสัญญาณ Tone และ 128 – 255 จะเป็นสัญญาณ White noise ถ้า Note ค่า 0 คือ เสียงเงียบ

Duration เป็นค่าระยะเวลาการเล่นเสียง มีค่าระหว่าง 0 - 255 การเพิ่มแต่ละค่าจะใช้เวลา 12 mSec.

คำสั่ง SOUND จะได้สัญญาณออกมาเป็นรูปสี่เหลี่ยม (Square Wave)

Ex.

```
SOUND PORTB.7, [100,10,50,10]    ' ส่งสัญญาณเสียง 2 Tone
```

3.31.58 คำสั่ง SWAP

รูปแบบ **SWAP** Variable, Variable

เป็นคำสั่งสลับค่าระหว่าง 2 ตัวแปร คำสั่งนี้สามารถใช้ได้กับตัวแปร ทั้งระดับ Bit Byte และ Word

Ex. .

```
SWAP B0,B1
```

2.31.59 คำสั่ง REPEAT … UNTIL

รูปแบบ **REPEAT**

.

งานที่ทำซ้ำ

.

UNTIL เงื่อนไขที่กำหนดเป็นจริง

เป็นคำสั่งให้ประมวลผลซ้ำ จนกระทั่งเงื่อนไขที่คำสั่ง UNTIL เป็นจริง การทำงานของคำสั่งชุดนี้จะทำการประมวลผล งานที่ทำซ้ำก่อน จึงจะตรวจสอบเงื่อนไขเมื่อเสร็จงานที่ทำซ้ำ ว่ายังเป็นจริงหรือไม่ หากยังเป็นเท็จจะย้อนไปทำงานซ้ำอีก จนกว่าจะตรวจสอบเงื่อนไขที่คำสั่ง UNTIL เป็นจริง ซึ่งตรงกันข้ามกับคำสั่ง WHILE … WEND ที่ตรวจสอบเงื่อนไขก่อนประมวลผลซ้ำ

Ex.

```
        i var byte
        adcon1 = 7
        i = 0
start:
    repeat
        toggle porta.0
        pause 1000
        i = i +1
    until i > 10
    end
```