หัวข้อการฝึกอบรมการใช้งาน CP-PIC877 V1.0

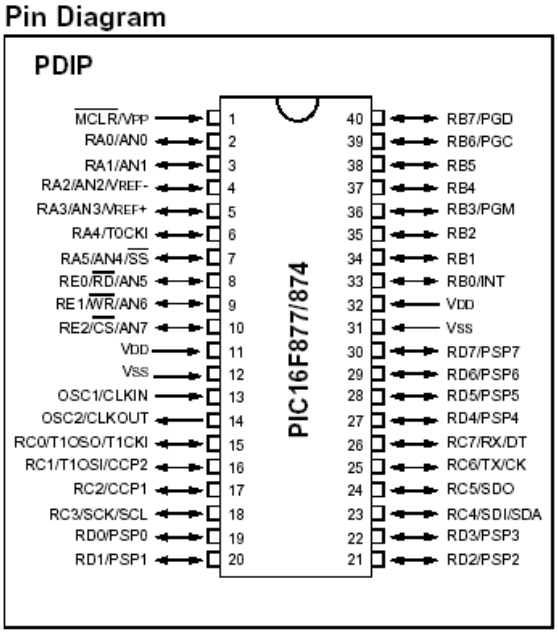
1. สถาปัตยกรรมและโครงสร้างทางด้านฮาร์ดแวร์ ของ PIC 16F877
2. การจัดสรรพื้นที่หน่วยความจำ และ รีจิสเตอร์ต่างๆ
3. ส่วนประกอบและข้อจำกัดต่างๆ ของบอร์ด CP-PIC877 V1.0
    - ส่วนประกอบต่างๆของ CP-PIC877 V1.0
    - การใช้งาน CP-PIC877 V1.0
    - การดาวน์โหลดโปรแกรม
4. แนวทางการพัฒนาโปรแกรม
    4.1 การพัฒนาโปรแกรมด้วยภาษาแอสเซมบลี
    - องค์ประกอบของภาษาแอสเซมบลี
    - การใช้งาน Assember หรือ Compiler
    - ตัวอย่างการเขียนโปรแกรมด้วยภาษาแอสเซมบลี

    4.2 การพัฒนาโปรแกรมด้วยภาษา BASIC (Pic Basic Pro)
    - องค์ประกอบของการเขียนโปรแกรมด้วยภาษา BASIC
    - การใช้งานและการติดตั้ง Pic Basic Pro Compiler
    - ตัวอย่างการเขียนโปรแกรมภาษา Basic

1. สถาปัตยกรรมและโครงสร้างทางด้านฮาร์ดแวร์ ของ PIC 16F877

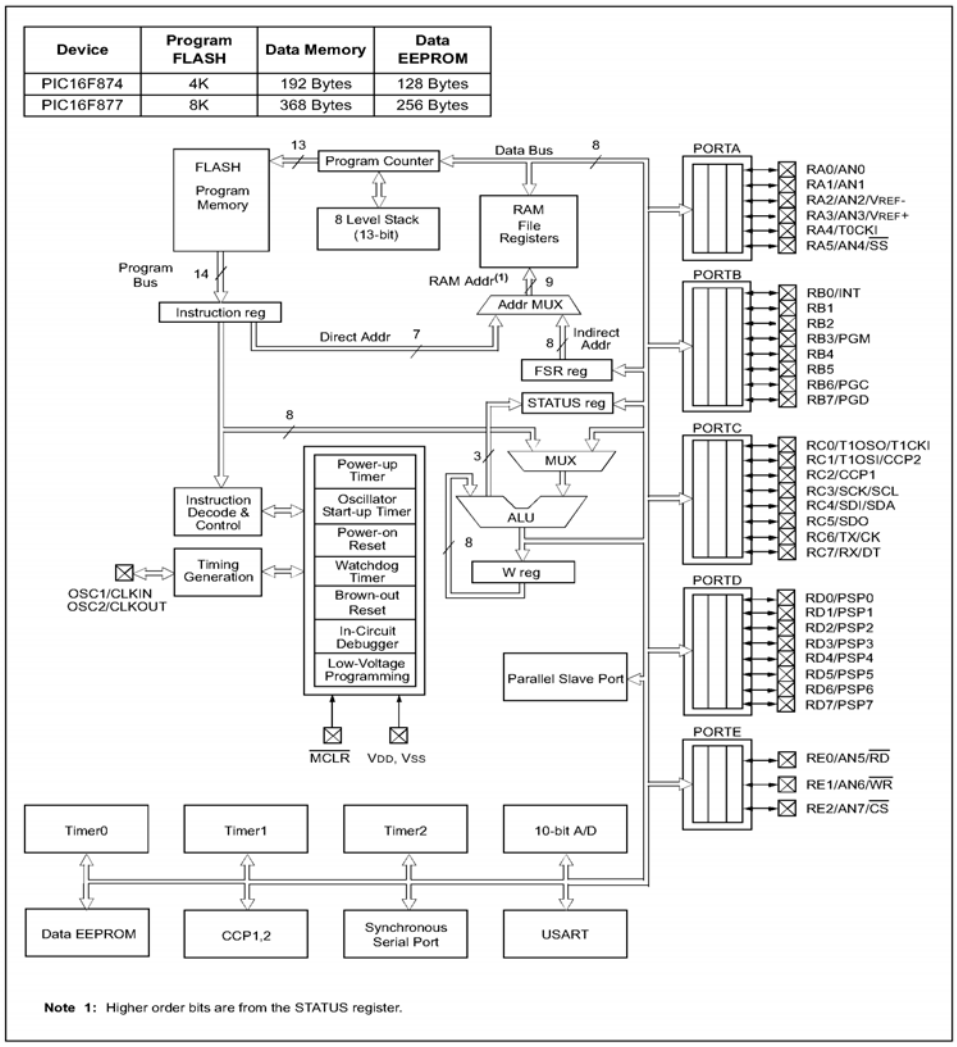
PIC 16F877 เป็น CPU ของบริษัท Microchip Technology ซึ่งเป็นผู้ผลิต CPU ตระกูล PIC โดยในเบอร์ 16F877 นี้เป็นเบอร์ที่มีความสามารถสูงพอสมควร ประกอบไปด้วยฟังก์ชันการทำงานต่างๆ มากมาย พอจะสรุปคุณสมบัติคร่าวๆ ได้ดังนี้

- มี 35 Instruction คำสั่ง
- ในการปฏิบัติงานคำสั่งต่างๆจะใช้ Cycle เดียว และ 2 Cycle ในคำสั่งที่เป็นการกระโดด
- ความถี่สูงสุดที่ทำงานได้คือ 20 MHz (16F877-20/P)
- การทำงานจะเป็นลักษณะ Pipeline ทำให้มีการทำงานที่เร็วขึ้น
- หน่วยความจำโปรแกรม FLASH Program Memory มีขนาด 8k (14-Bit Words)
- หน่วยความจำข้อมูล (RAM) 368 Bytes
- หน่วยความจำข้อมูล (EEPROM) 256 Bytes
- สามารถตอบสนองการอินเตอร์รัพได้ถึง 14 แหล่ง
- มี STACK 8 ระดับ
- มีเพาเวอร์ออนรีเซต(POR),เพาเวอร์อัปไทเมอร์(PWRT) และ Oscillator Start-Up Timer
- Watchdog Timer
- สามารถเลือกการป้องกันข้อมูลได้ (Code Protection)
- มีโหมดประหยัดพลังงาน (Sleep Mode)
- เลือกโหมดของ สัญญาณนาฬิกาได้หลายโหมด
- สามารถโปรแกรมโดยใช้แรงดัน +5V ได้
- มีฟังก์ชันการโปรแกรมแบบ ICSP (In-Circuit Serial Programing)
- ทำงานที่ไฟเลี้ยง 2.0V ถึง 5.5V
- กระแสทั้งซิงก์และซอร์สของพอร์ตคือ 25mA
- มี Timer/Counter จำนวน 3 ตัว คือ Timer0,Timer1 และ Timer2
- มีโมดูล Capture/Compare/PWM จำนวน 2 ชุด
- มี Analog to Digital Converter ความละเอียด 10 บิต 8 แชนเนล ภายในตัว
- มีโมดูลการสื่อสาร USART
- มีโมดูลตรวจจับระดับไฟเลี้ยง Brown – out reset (BOR)
- มีพอร์ต I/O 5 พอร์ตประกอบด้วย A,B,C,D และ E แต่ละพอร์ตจะมีจำนวนบิตไม่เท่ากัน ซึ่งรวมแล้วจะมี I/O จำนวน 33 บิต



ตำแหน่งขาสัญญาณต่าง ๆ ของ PIC 16F877

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



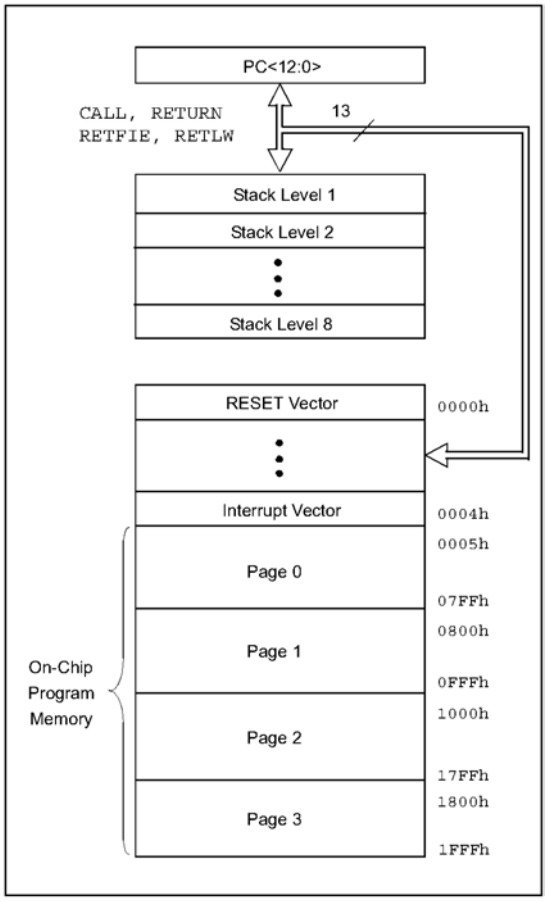
สถาปัตยกรรมภายในของ PIC 16F877

2. การจัดสรรพื้นที่หน่วยความจำ และ รีจิสเตอร์ต่างๆ

การจัดหน่วยความจำของ PIC 16F877 นี้จะแบ่งออกเป็น 3 ส่วนคือ หน่วยความจำโปรแกรม , หน่วยความจำข้อมูล(RAM) และหน่วยความจำข้อมูลที่เป็น EEPROM ซึ่งเราจะมาดูในแต่ละส่วนดังนี้

2.1 หน่วยความจำโปรแกรม

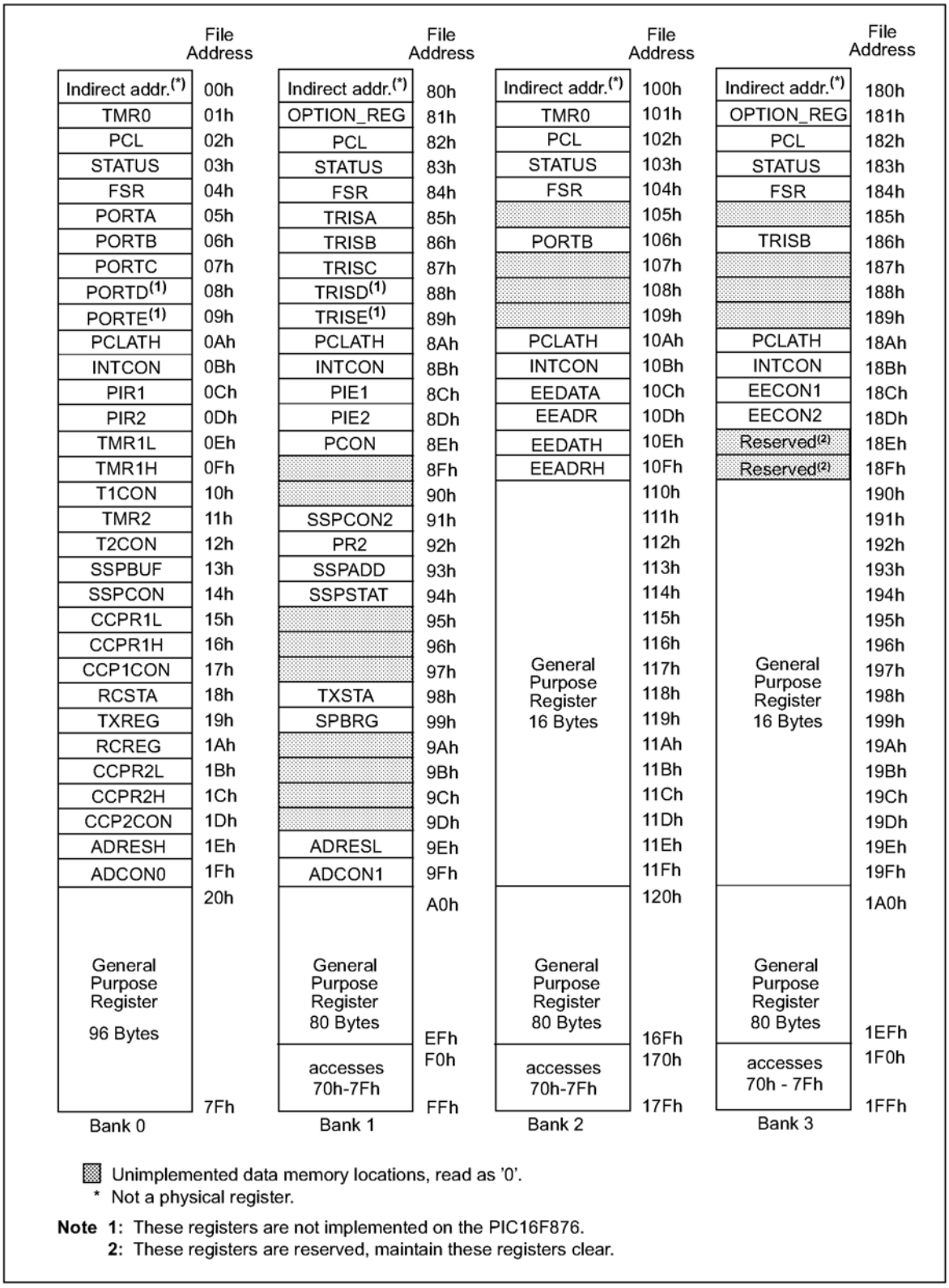
PIC 16F877 นี้มี Program Counter ขนาด 13 บิตซึ่งสามารถอ้างถึงตำแหน่งข้อมูลได้ถึง 8 กิโลเวิร์ด โดยจะมีตำแหน่ง Reset Vector ที่ 0000h และ Interupt Vector ที่ 0004h ดังนั้นในการเขียนโปรแกรมจึงควรสงวนพื้นที่ส่วนนี้ไว้สำหรับการใช้งานอินเตอร์รัพท์ จากรูปด้านล่างจะเห็นว่ามีพื้นที่ของ Stack 8 ระดับ และหน่วยความจำโปรแกรมแบ่งออกเป็น 4 Page (8 kwords) ซึ่งพื้นที่ในส่วนนี้มีไว้สำหรับเก็บข้อมูลคำสั่งทั้งหมดโดยโครงสร้างจะเป็นแบบแฟลช (flash memory) ทำให้ลบและเขียนใหม่ได้หลายครั้ง



การจัดสรรพื้นที่หน่วยความจำของ PIC 16F877

2.2 หน่วยความจำข้อมูล

ใน PIC 16F877 นี้หน่วยความจำข้อมูลจะแบ่งออกเป็นพื้นที่ของ RAM หน่วยความจำใช้งานทั่วไป (Gerneral Purpose Register) ขนาด 368 Bytes และ พื้นที่ของ รีจิสเตอร์ฟังก์ชันพิเศษ (Special Function Registers) ในการจัดวางพื้นที่จะแบ่งออกเป็น 4 แบงค์ ตั้งแต่แอดเดรส 00h ถึง 1FFh ดังรูป



File Address	Bank 0	File Address	Bank 1	File Address	Bank 2	File Address	Bank 3
00h	Indirect addr.(*)	80h	Indirect addr.(*)	100h	Indirect addr.(*)	180h	Indirect addr.(*)
01h	TMR0	81h	OPTION_REG	101h	TMR0	181h	OPTION_REG
02h	PCL	82h	PCL	102h	PCL	182h	PCL
03h	STATUS	83h	STATUS	103h	STATUS	183h	STATUS
04h	FSR	84h	FSR	104h	FSR	184h	FSR
05h	PORTA	85h	TRISA	105h		185h	
06h	PORTB	86h	TRISB	106h	PORTB	186h	TRISB
07h	PORTC	87h	TRISC	107h		187h	
08h	PORTD(1)	88h	TRISD(1)	108h		188h	
09h	PORTE(1)	89h	TRISE(1)	109h		189h	
0Ah	PCLATH	8Ah	PCLATH	10Ah	PCLATH	18Ah	PCLATH
0Bh	INTCON	8Bh	INTCON	10Bh	INTCON	18Bh	INTCON
0Ch	PIR1	8Ch	PIE1	10Ch	EEDATA	18Ch	EECON1
0Dh	PIR2	8Dh	PIE2	10Dh	EEADR	18Dh	EECON2
0Eh	TMR1L	8Eh	PCON	10Eh	EEDATH	18Eh	Reserved(2)
0Fh	TMR1H	8Fh		10Fh	EEADRH	18Fh	Reserved(2)
10h	T1CON	90h		110h	General Purpose Register 16 Bytes	190h	General Purpose Register 16 Bytes
11h	TMR2	91h	SSPCON2	111h		191h	
12h	T2CON	92h	PR2	112h		192h	
13h	SSPBUF	93h	SSPADD	113h		193h	
14h	SSPCON	94h	SSPSTAT	114h		194h	
15h	CCPR1L	95h		115h		195h	
16h	CCPR1H	96h		116h		196h	
17h	CCP1CON	97h		117h		197h	
18h	RCSTA	98h	TXSTA	118h		198h	
19h	TXREG	99h	SPBRG	119h		199h	
1Ah	RCREG	9Ah		11Ah		19Ah	
1Bh	CCPR2L	9Bh		11Bh		19Bh	
1Ch	CCPR2H	9Ch		11Ch		19Ch	
1Dh	CCP2CON	9Dh		11Dh		19Dh	
1Eh	ADRESH	9Eh	ADRESL	11Eh		19Eh	
1Fh	ADCON0	9Fh	ADCON1	11Fh		19Fh	
20h	General Purpose Register 96 Bytes	A0h	General Purpose Register 80 Bytes	120h	General Purpose Register 80 Bytes	1A0h	General Purpose Register 80 Bytes
7Fh		EFh		16Fh		1EFh	
		F0h	accesses 70h-7Fh	170h	accesses 70h-7Fh	1F0h	accesses 70h - 7Fh
		FFh		17Fh		1FFh	

Unimplemented data memory locations, read as '0'.
* Not a physical register.

Note 1: These registers are not implemented on the PIC16F876.
2: These registers are reserved, maintain these registers clear.

การจัดวางพื้นที่ของหน่วยความจำข้อมูล

ซึ่งจากรูป ในการเข้าถึงข้อมูลในแต่ละส่วน หรือ แต่ละแบงก์ สามารถทำได้โดยการกำหนดค่าในรีจิสเตอร์ STATUS ในบิตที่ 5 และ 6 (RP0,RP1) ซึ่งมีความหมายดังนี้

RP1	RP0	Bank Select
0	0	Bank0 : 00h – 7Fh
0	1	Bank1 : 80h – FFh
1	0	Bank2 : 100h – 17Fh
1	1	Bank3 : 180h – 1FFh

2.3 หน่วยความจำข้อมูล EEPROM

PIC 16F877 มีหน่วยความจำแบบ EEPROM จำนวน 256 ไบต์ โดยสามารถอ่านและเขียนในขณะทำงานปกติได้แต่ต้องไม่มีการ Enable Code protect bit โดยการเข้าถึงนั้นจะต้องทำผ่าน รีจิสเตอร์พิเศษ (Special Function Register) ซึ่งต้องใช้ถึง 4 ตัวดังนี้

EECON1 : ควบคุมการเข้าถึงหน่วยความจำ

EECON2 : จัดลำดับการเขียนข้อมูล

EEDATA : เป็นบัฟเฟอร์ใช้เก็บข้อมูล 8 บิต สำหรับการอ่านและเขียน

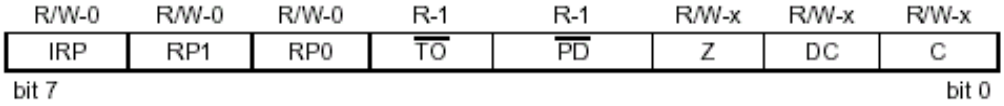
EEADR : รีจิสเตอร์ที่เก็บแอดเดรส 00h – FFh (256 ไบต์)

รีจิสเตอร์ที่สำคัญต่างๆ ของ PIC 16F877

รีจิสเตอร์จัดได้ว่าเป็นส่วนประกอบที่มีความสำคัญต่อการพัฒนาโปรแกรมเป็นอย่างมาก ซึ่งใน PIC 16F877 นี้มีรีจิสเตอร์ต่างๆ มากมาย เราจะมาดูเฉพาะรีจิสเตอร์ที่ใช้งานหลักๆ ดังนี้

- รีจิสเตอร์ STATUS

เป็นรีจิสเตอร์ที่ใช้เก็บข้อมูลสถานะ การทำงานของไมโครคอนโทรลเลอร์ ไม่ว่าจะเป็นแฟลกสถานะต่างๆ ที่ใช้บอกผลลัพธ์การทำงานของส่วนต่างๆ เช่น การคำนวณทางคณิตศาสตร์ การเลือกแบงก์ข้อมูล ซึ่งมีรายละเอียดในแต่ล่ะบิตดังนี้



IRP0 (Indirect Register Bank Select bit – บิต 7) ใช้เลือกแบงก์ในหน่วยความจำชนิด RAM ในกรณีที่ใช้การอ้างถึงข้อมูลแบบทางอ้อม (indirect addressing mode)

“0” : เลือกแบงก์ 0 และ 1

“1” : เลือกแบงก์ 2 และ 3

RP1,RP0 (Register Bank Select) ใช้เลือกแบงก์ของหน่วยความจำข้อมูล แรม และ รีจิสเตอร์ไฟล์ เมื่อเป็นการเข้าถึงข้อมูลแบบโดยตรง (direct addressing mode)

"00" เลือกแบงก์ 0

"01" เลือกแบงก์ 1

"10" เลือกแบงก์ 2

"11" เลือกแบงก์ 3

TO (Time-out bit) เป็นบิตที่แสดงการเกิดไทม์เอาต์ของวอตช์ด็อกไทเมอร์แอกตีฟลอจิก "0"

PD (Power-down) บิตแสดงการทำงานในโหมดสลีป (Sleep mode) คือ เมื่อเข้าสู่ Sleep mode บิตนี้จะกลายเป็น "0"

Z (Zero bit) เป็นบิตแสดงสถานะการทำงานทางคณิตศาสตร์

"0" เมื่อผลลัพธ์ทางคณิตศาสตร์ลอจิกไม่เป็นศูนย์

"1" เมื่อผลลัพธ์ทางคณิตศาสตร์ลอจิกเป็นศูนย์

DC (Digit carry/borrow) เป็นบิตทดหรือยืมระหว่างหลัก

"0" เมื่อไม่เกิดการทดจากบิต 3 ไปบิต 4 หรือ หากเกิดการยืมจากบิต 4 มาบิต 3

"1" เมื่อเกิดการทดจากบิต 3 ไปบิต 4 หรือ หากไม่เกิดเกิดการยืมจากบิต 4 มาบิต 3

C (Carry/borrow) บิตทดหรือยืม ใช้แสดงสถานะการทด หรือการยืมทางคณิตศาสตร์

"0" เมื่อไม่มีการทดบิต 7 (MSB) หรือ เกิดการยืมจากบิต 7 (MSB)

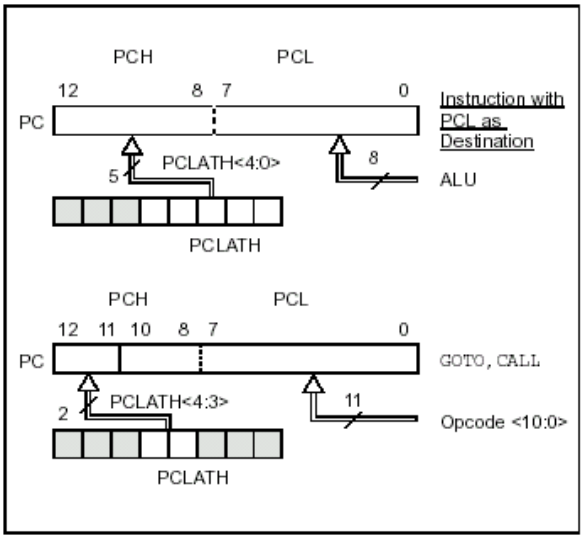
"1" เมื่อมีการทดจากบิต 7 (MSB) หรือ ไม่เกิดการยืมค่าของบิต 7 (MSB)

- รีจิสเตอร์ W (Working register)

เป็นรีจิสเตอร์ที่มีความสำคัญมากที่สุดตัวหนึ่งของไมโครคอนโทรลเลอร์ PIC ซึ่งรีจิสเตอร์ W เป็นรีจิสเตอร์ขนาด 8 บิต จะทำหน้าที่เหมือนเป็นรีจิสเตอร์ แอกคิวมูลเลเตอร์ ในการทำคำสั่งทางคณิตศาสตร์หรือการโอนย้ายข้อมูล จะต้องผ่านรีจิสเอตร์ W ทั้งสิ้น

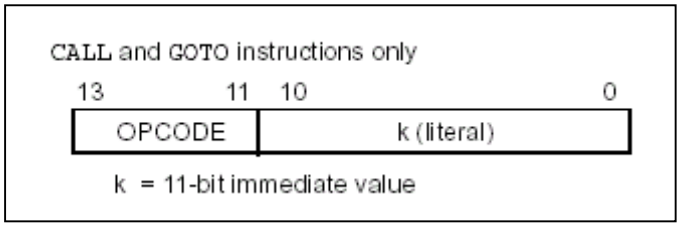
- โปรแกรมเคาน์เตอร์ (PC)

เป็นรีจิสเตอร์ที่สำคัญอีกตัวหนึ่ง ซึ่งทำหน้าที่เก็บตำแหน่งแอดเดรสที่จะให้ CPU ไปทำงานโดยรีจิสเตอร์ PC ของ PIC 16F877 จะมีขนาด 13 บิต โดย 8 บิตล่าง PC< 7:0 > จะอยู่ที่รีจิสเตอร์ PCL สามารถอ่านและเขียนได้เหมือนรีจิสเตอร์ทั่วไป ส่วน PC<12 :8 > จะไม่สามารถเข้าถึงได้ตามปกติ การเข้าถึงจะกระทำผ่านรีจิสเตอร์ PCLATH < 4 :0 > และเมื่อเกิดการรีเซต PCLATH จะเคลียร์ สถานะเป็น "0" ในการใช้คำสั่งในการกระโดดเช่น CALL หรือ GOTO จะเป็นการนำค่าแอดเดรสมาใส่ในรีจิสเตอร์ PC <10:0 > เพียง 11 บิตเท่านั้นส่วน PC<12:11> จะไม่เปลี่ยนแปลง ทำให้การใช้คำสั่ง CALL หรือ GOTO ไปได้ไกลเพียง 2Kwords (2048 ตำแหน่ง) แต่หน่วยความจำทั้งหมดมี 8 Kwords แบ่งเป็น 4 Page ฉนั้นในการกระโดดข้าม Page จะต้องมีการกำหนดค่าให้กับ PC<12:11> ด้วยโดยผ่านทางรีจิสเตอร์ PCLATH



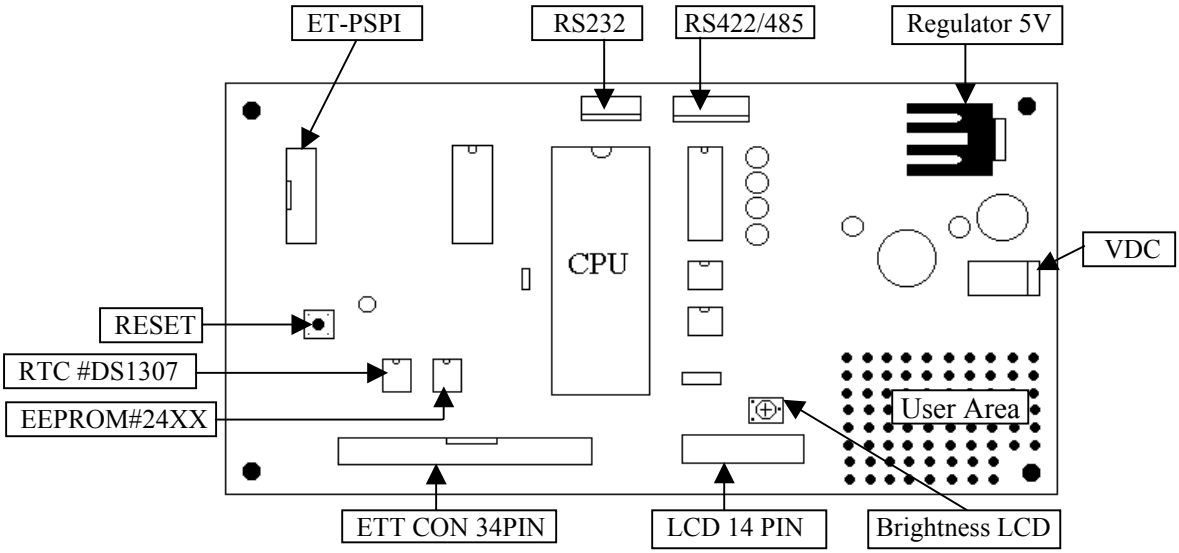
โครงสร้างของรีจิสเตอร์ PC

<u>ตัวอย่าง</u> โปรแกรมที่มีการกระโดดข้าม Page




```
                ORG 0x500
        BCF     PCLATH,4
        BSF     PCLATH,3        ; Select page 1
                                ; (800h-FFFh)
        CALL    SUB1_P1         ; Call subroutine in
                ……              ; page 1 (800h – FFFh)
                ...……
                ORG  0x900      ; page 1 (800h – FFFh)
SUB1_P1
                ……              ; Called subroutine page 1 (800 – FFFh)
                ……
        RETURN                  ; return to call subroutine in page 0 (000h-7FFh)
```

3. ส่วนประกอบและข้อจำกัดต่างๆ ของบอร์ด CP-PIC877 V1.0



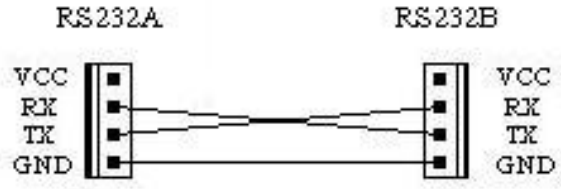
- ใช้ CPU PIC 16F877 RUN 4 MHZ
- ON CHIP FLASH PROGRAM MEMORY 8K x 14 WORDS
- ON CHIP 368 BYTES RAM / 256 BYTES EEPROM
- 31 BIT I/O PORT ใช้งานของ CPU บนบอร์ด CP-PIC877 V1.0 (34 PIN I/O ET-BUS)
- PORT แบบ SPI , I2C , RS232 ,RS422/485 (OPTIONS)
- กระแสสำหรับทำการซิงค์/ซอร์สสูงสุด (HIGH SINK / SOURCE CURRENT) 25 MA
- TWO CAPTURE , COMPARE , PWM MODULES
- ระบบ RTC ใช้ชิพ DS1307 (OPTIONS)
- Serial EEPROM 24XX (OPTIONS)
- POWER ON RESET/WATCHDOG TIMER
- A-TO-D ขนาด 10 BIT 8 CH (On Chip)
- LCD PORT 14 PIN ET-BUS สำหรับ LCD แบบตัวอักษร
- 7805 POWER SUPPLY ON BOARD
- PCB SIZE CP-PIC877V1.0 12 x 8.5 cm

การใช้งาน CP-PIC877 V1.0

ในบอร์ด CP-PIC877V1.0 ประกอบไปด้วยทรัพยากรย์ ต่างๆ ที่ได้ออกแบบไว้ให้สะดวกต่อการนำไปใช้งานต่างๆ ประกอบไปด้วยส่วนต่างๆ ดังนี้

RS-232

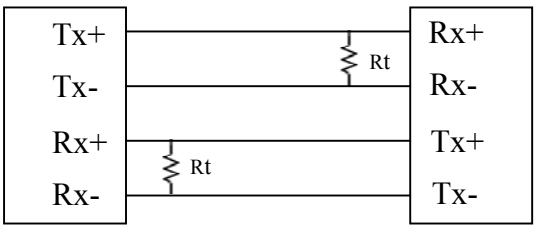
ในส่วนของ RS-232 นี้มีไว้เพื่อเพิ่มความสามารถในการสื่อสารกันระหว่าง CPU กับ PC หรือ กับบอร์ดอื่นๆ ซึ่งเป็นการสื่อสารแบบอนุกรมโดยในบอร์ดนี้ใช้ MAX232 หรือ ICL232 เป็น Line Driver ซึ่งสามารถสื่อสารโดยใช้สายสัญญาณเพียง 3 เส้นได้ไกลถึง 50 ฟุต หรือ ประมาณ 15 เมตร การใช้การใช้งานนั้นก็ทำได้โดยการต่อสายที่ออกจากคอนเนคเตอร์ RS-232 บนบอร์ด ไปต่อกับ PC หรือบอร์ดตัวอื่นแล้วเขียนโปรแกรมควบคุมให้ CPU ทำงาน แต่เนื่องจากสัญญาณอนุกรมที่ออกจาก CPU นี้ต้องไปต่อกับ Line Driver ของ RS-422/RS-485 ด้วย ดังนั้นการใช้งาน RS-232 นี้ให้ถอดไอซีที่เป็น Line Driver ของRS-422/RS-485 ออกทั้ง 2 ตัวด้วยซึ่งก็คือเบอร์ 75176



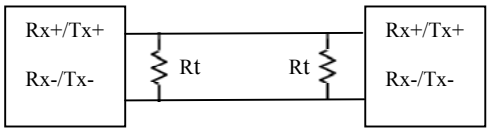
การต่อใช้งาน RS-232

RS-422/RS-485

ในส่วนของ RS-422/RS-485 เป็นพอร์ตสำหรับการสื่อสารอนุกรมอีกตัวหนึ่งที่สามารถติดต่อกับอุปกรณ์ตัวอื่นๆ ได้เป็นระยะทางไกลๆ ซึ่งได้ถึง 1.2 กิโลเมตร หรือ 4000 ฟุต และมีประโยชน์มากกับงานที่ต้องทำเป็น Network ซึ่งสามารถต่อกันเป็น Network ได้ 32 จุด โดยใช้ 75176 เป็น Line Driver ในกรณีที่เป็น RS-442 จะเป็นการสื่อสารแบบ “Full Duplex” ซึ่งจะต้องใช้สายสัญญาณ 4 เส้นและสามารถใช้โปรแกรมที่เขียนขึ้นเพื่อใช้งานกับ RS-232 ได้โดยไม่ต้องเปลี่ยนแปลงโปรแกรม ส่วน RS-485 เป็นการสื่อสารแบบ “Half Duplex” โดยการใช้งานจะต้องเลือกใช้แบบใดแบบหนึ่งเท่านั้นโดยการเซ็ทที่จัมเปอร์ “RS422/RS485”



การต่อ RS422 แบบ 4-WIRE Full Duplex

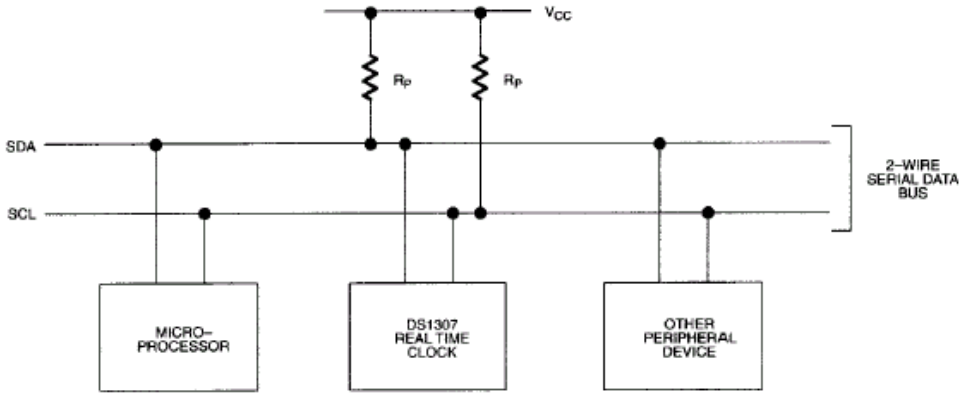


การต่อ RS 485 แบบ 2-WIRE Half Duplex

RTC(Real Time Clock) เบอร์ DS1307

สำหรับงานบางอย่างที่ต้องใช้ฐานเวลาเข้ามาเกี่ยวข้อง เช่น นาฬิกาจับเวลา หรือตั้งเวลา หรือ ต้องการบันทึกเวลา เช่น Data Logger ฯลฯ RTC จึงเป็นอุปกรณ์ที่จำเป็นอย่างยิ่งเพื่อสร้างระบบเวลาให้กับ CPU ซึ่ง

ในบอร์ดของ CP-PIC877 V1.0 นี้ได้เลือกใช้ เบอร์ DS1307 ซึ่งสามารถเก็บค่าเวลาไว้ได้แม้ไฟดับ โดยใช้แบตเตอร์รี่แบ็คอัพ 3 โวลท์ และสามารถเขียนและอ่านข้อมูลจาก RTC ได้ด้วยสายสัญญาณเพียง 2 เส้น หรือที่เรียกว่า I2C บัส ทำให้ประหยัด I/O ไปได้อีก DS1307 นี้จะมีคอนโทรลแอดเดรสเป็น D0H (11010000)

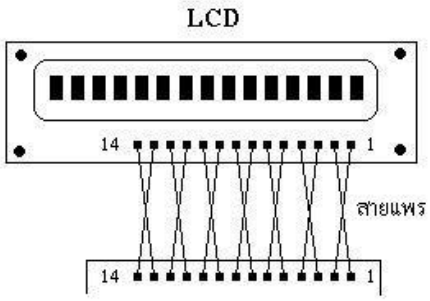


EEPROM 24XX

ปกติในตัว CPU จะมี EEPROM จำนวนหนึ่งอยู่แล้ว แต่สำหรับบางงานที่ต้องการหน่วยความจำมากกว่านี้จึงจำเป็นต้องใช้หน่วยความจำภายนอกมาต่อเพิ่ม ซึ่งในบอร์ด CP-PIC877 V1.0 ได้ออกแบบให้มีการเชื่อมต่อกับอุปกรณ์ที่เป็น EEPROM โดยใช้การติดต่อผ่านสายสัญญาณเพียง 2 เส้นเรียกว่า I2C BUS ซึ่งจะเหมือนกับการอ่านและเขียน RTC ซึ่งจากวงจรจะเห็นว่าขา DATA และขา CLK ถูกต่ออยู่กับ I/O ของ CPU เส้นเดียวกันแต่ไม่เป็นปัญหาในการอ่านและเขียน เพราะอุปกรณ์สองตัวนี้มีแอดเดรสต่างกันคือ RTC มีแอดเดรสเป็น D0H และ EEPROM เบอร์ 2416 นี้มีคอนโทรลแอดเดรสอยู่ที่ A0H

LCD 14 PIN

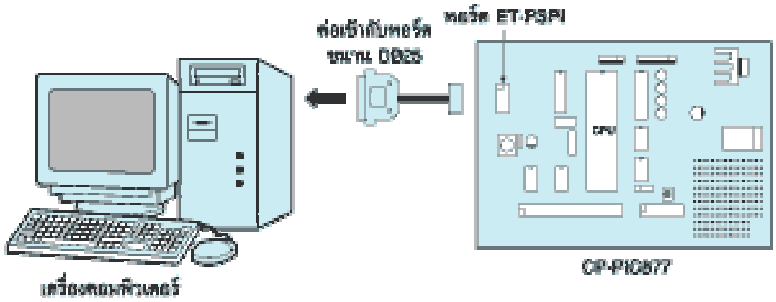
LCD 14 PIN เป็นคอนเนคเตอร์มาตรฐานอีกอันหนึ่งบนบอร์ดอีทีทีที่มีไว้ต่อกับ LCD แบบ Character ไม่จำกัดจำนวนบรรทัด ซึ่งในบอร์ดของอีทีทีนี้ใช้การ Interface แบบ 4 Bit Data โดยการวางตำแหน่งของขาสัญญาณต่างๆ แสดงดังรูปต่อไปนี้



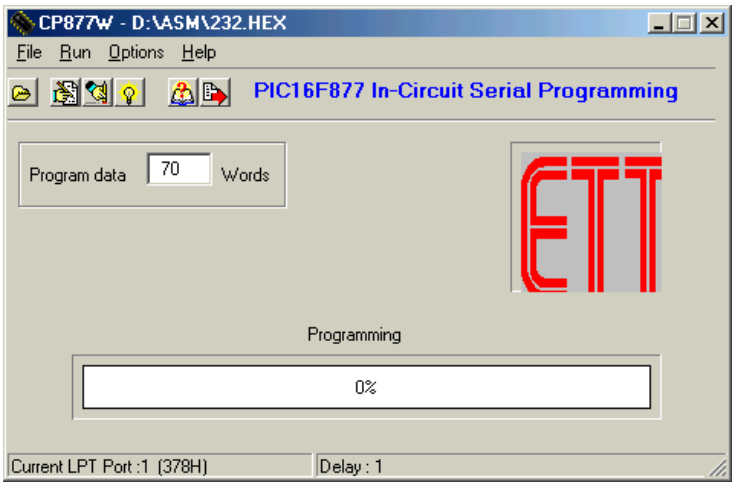
ในการต่อสายระหว่างบอร์ดไปยัง LCD จะต้องมีการไขว้สายสลับกันไปมาดังรูป คือ1-2 ,3-4 ,…13-14 เมื่อป้อนไฟให้กับ แอลซีดี แล้วทำการปรับความสว่าง (Brighness) ของจอแอลซีดี โดยการปรับที่ VR 10K ถ้าหากต่อถูกต้อง แอลซีดี แถวบนจะติดทั้งหมด

การดาวน์โหลดโปรแกรม

ในการดาวน์โหลดโปรแกรมจะต้องต่อสายดาวน์โหลดระหว่างคอมพิวเตอร์กับบอร์ด CP-PIC877 โดยปลายด้านที่ต่อกับคอมพิวเตอร์จะเป็น DB-25(PRINTER PORT) และ ด้านที่ต่อกับ CP-PIC877 จะเป็นคอนเนคเตอร์ 10PINดังรูป



การเชื่อมต่อสายดาวน์โหลด



โปรแกรมที่ใช้ในการดาวน์โหลด CP877W


Open : เปิดไฟล์ที่ต้องการดาวน์โหลด


Program (Erase -> Program -> Verify) : โปรแกรมข้อมูลลง CPU

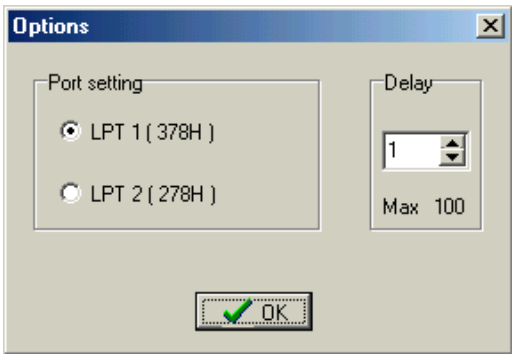

Erase : ลบข้อมูลใน CPU


Reset CPU


About product

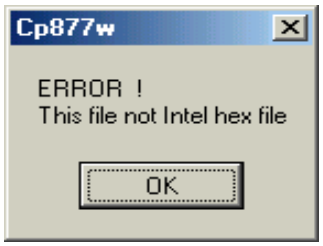

Exit : ออกจากโปรแกรม

Options เป็นเมนูที่ใช้ในการกำหนด พอร์ตสำหรับดาวน์โหลด และกำหนดค่า Delay เนื่องจาก PC แต่ละเครื่องมีความเร็วที่แตกต่างกันจึงต้องมีการใส่ค่า Delay เพื่อให้สามารถโปรแกรมข้อมูลลง CPU ได้



การเกิดความผิดพลาดในกรณีต่างๆ

ค่า Delay อาจน้อยไปให้เพิ่มค่า Delay และตรวจสอบสายดาวน์โหลด หรือ แหล่งจ่ายไฟให้ถูกต้อง



ไฟล์ที่นำมาดาวน์โหลดไม่อยู่ในรูปแบบของ Intel hex format ให้ลองเปลี่ยนตัว Compiler เป็นตัวอื่นแทน

4. แนวทางการพัฒนาโปรแกรม

การพัฒนาโปรแกรมของ PIC 16F877 นี้สามารถใช้ได้ทั้งภาษา แอสเซมบลี Basic และ ภาษา C แต่ในที่นี้จะกล่าวถึงเฉพาะภาษา แอสเซมบลี และ Basic เท่านั้น

4.1 การพัฒนาโปรแกรมด้วยภาษาแอสเซมบลี

ภาษาแอสเซมบลีเป็นภาษาที่มาคู่กับ CPU ประกอบไปด้วยชุดคำสั่งต่างๆ ที่ทางผู้ผลิตได้ออกแบบข้อดีของภาษาแอสเซมบลีก็คือ จะทำงานเร็วกว่าภาษาอื่นๆ เนื่องจากมีความใกล้เคียงกับภาษาเครื่องมาก แต่ต้องอาศัยความรู้ทางด้านโครงสร้างการทำงานของ CPU นั้นๆด้วยจึงจะสามารถใช้งาน CPU ได้อย่างมีประสิทธิภาพ ดังนั้นจึงต้องเสียเวลาในการศึกษารายละเอียดต่างๆ ของ CPU พอสมควร ในการอธิบายจะยึด Assembler ของ Microchip เป็นหลัก โดยทั่วไปแล้วโปรแกรมภาษาแอสเซมบลีจะประกอบด้วยส่วนต่างๆ ดังนี้

```
        List  p=16f877                  ; list directive to define processor
        #include <p16f877.inc>          ; processor specific variable definitions
        __CONFIG                        ; การกำหนดค่า Configuration ให้ CPU

        #define   output   PORTA,0      ; การตั้งชื่อตัวแปรแทนรีจิสเตอร์

        temp     EQU     0x70           ; variable used for context saving
        ; -----------------------------------------------------------------------------
        ORG    0x000                    ; processor reset vector
LABEL:          Operation    Operand    ; comment
                ……..
                ……..
                END                     ; directive 'end of program'
```

ข้อกำหนดอื่นๆ ของ Assembler

- รูปแบบของเลขฐานสิบหก ฐานสิบ และ ฐานสอง เป็นดังนี้

  0xf9 หรือ 0xF9 กรณีฐานสิบหก b'11111001' กรณีฐานสอง d'23' หรือ .23 กรณีฐานสิบ

- การกำหนดค่า CONFIG

  Brown-out Reset Enable bit

  _BODEN_ON , _BODEN_OFF

  Flash Program Memory Code Protection bits

  _CP_ALL , _CP_HALF , _CP_UPPER_256 , _CP_OFF

  Flash Program Memory Write Enable bit

  _WRT_ENABLE_ON , _WRT_ENABLE_OFF

  Power-up Timer Enable bit

  _PWRTE_OFF , _PWRTE_ON

  Watchdog Timer Enable bit

  _WDT_ON , _WDT_OFF

  Oscillator Selection bits

  _LP_OSC , _XT_OSC , _HS_OSC , _RC_OSC

Background Debugger Mode

_DEBUG_ON , _DEBUG_OFF

Data EE Memory Code Protection

_CPD_ON , _CPD_OFF , _LVP_ON , _LVP_OFF

ON = 0 , OFF = 1 และมีความหมายดังตารางข้างล่างนี้

U-0	U-0	U-0	U-0	U-0	U-0	U-0	R/P-1	U-0	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1
CP1	CP0	RESV	—	WRT	CPD	LVP	BODEN	CP1	CP0	$\overline{\text{PWRTE}}$	WDTE	F0SC1	F0SC0
bit 13													bit 0

bit 13-12 **CP1:CP0:** FLASH Program Memory Code Protection bits(2)
bit 5-4
4 K Devices:
11 = Code protection off
10 = 0F00h to 0FFFh code protected
01 = 0800h to 0FFFh code protected
00 = 0000h to 0FFFh code protected
8 K Devices:
11 = Code protection off
10 = 1F00h to 1FFFh code protected
01 = 1000h to 1FFFh code protected
00 = 0000h to 1FFFh code protected

bit 11 **Reserved:** Set to '1' for normal operation

bit 10 **Unimplemented:** Read as '1'

bit 9 **WRT:** FLASH Program Memory Write Enable bit
1 = Unprotected program memory may be written to by EECON control
0 = Unprotected program memory may not be written to by EECON control

bit 8 **CPD:** Data EE Memory Code Protection bit
1 = Code protection off
0 = Data EE memory code protected

bit 7 **LVP:** Low Voltage ICSP Programming Enable bit
1 = RB3/PGM pin has PGM function, low voltage programming enabled
0 = RB3 is digital I/O, HV on $\overline{\text{MCLR}}$ must be used for programming

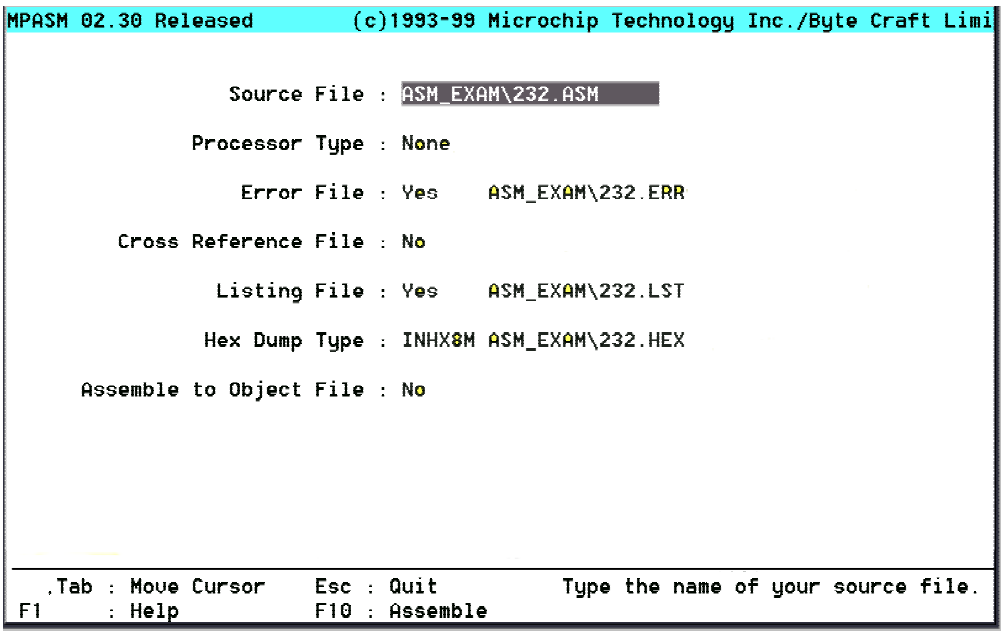
bit 6 **BODEN:** Brown-out Reset Enable bit(2)
1 = BOR enabled
0 = BOR disabled

bit 3 **$\overline{\text{PWRTE}}$:** Power-up Timer Enable bit
1 = PWRT disabled
0 = PWRT enabled

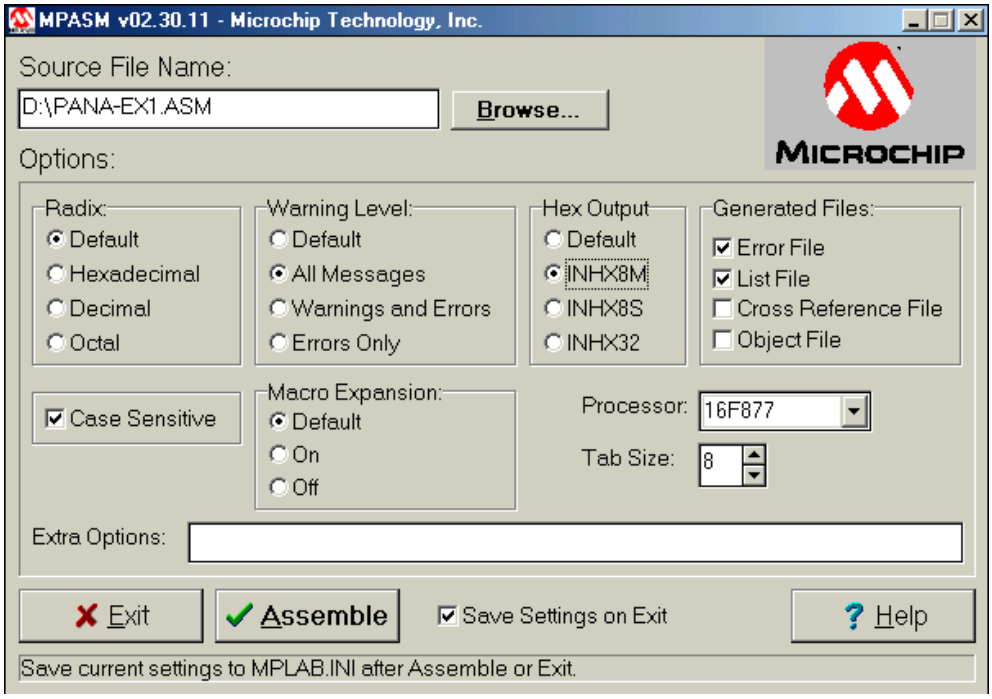
bit 2 **WDTE:** Watchdog Timer Enable bit
1 = WDT enabled
0 = WDT disabled

bit 1-0 **FOSC1:FOSC0:** Oscillator Selection bits
11 = RC oscillator
10 = HS oscillator
01 = XT oscillator
00 = LP oscillator

Note 1: Enabling Brown-out Reset automatically enables Power-up Timer (PWRT), regardless of the value of bit $\overline{\text{PWRTE}}$. Ensure the Power-up Timer is enabled any time Brown-out Reset is enabled.
2: All of the CP1:CP0 pairs have to be given the same value to enable the code protection scheme listed.



รูปแสดงหน้าตาของ MPASM.EXE for DOS



รูปแสดงหน้าต่าง Mpasmwin for Windows

การใช้งานพอร์ต อินพุต/เอาต์พุต

PIC16F877 นี้จะมีพอร์ตให้ใช้งานทั้งหมด 5 Port คือ PORTA(7Bit) , PORTB(8Bit) , PORTC(8Bit) , PORTD(8Bit) และ PORTE(3Bit) โดยแต่ละบิตสามารถกำหนดการทำงานให้เป็นอินพุตหรือ เอาท์พุตได้อย่างอิสระ แต่ใน CP-PIC877 นี้ I/O บางส่วนถูกออกแบบไว้สำหรับการ Program CPU คือ RB3 บิตนี้จึงไม่สามารถนำไปใช้งานได้ เพราะหาก RB3 ได้รับลอจิก “1” CPU จะเข้าสู่โหมด Program CPU ทันที ซึ่งการใช้งาน I/O CPU ตระกูล PIC นั้นก่อนการใช้งานจะต้องทำการกำหนดหน้าที่การทำงานให้กับ บิตต่างๆก่อน ว่าจะให้เป็นอินพุต หรือ เอาต์พุต ดังตัวอย่างการใช้งานต่อไปนี้

```
MOVLW   0xCF            ; ค่าสำหรับกำหนดการทำงานของ PortA (11001111)
MOVWF   TRISA           ; Set RA<3:0> as inputs
                        ; RA<5:4> as outputs
                        ; RA<7:6> as inputs
```

พอร์ตอื่นๆ ก็เช่นกันโดยการเปลี่ยนจาก A เป็น B , C , D หรือ E และ ค่าที่กำหนดให้รีจิสเตอร์นั้นถ้าเป็น “1” จะหมายถึง อินพุต และ “0” หมายถึง เอาต์พุต

ตัวอย่างโปรแกรมภาษาแอสเซมบลี

<u>ตัวอย่างที่ 1</u> โปรแกรมรับค่าจากสวิตช์ แล้ว นำไปแสดงผลที่ LED

```
    LIST P=16f877
    include <p16f877.inc>
  __CONFIG _CP_OFF & _WDT_OFF & _BODEN_ON & _PWRTE_ON & _XT_OSC & _WRT_ENABLE_ON &
_LVP_ON & _DEBUG_OFF & _CPD_OFF
                LED     EQU     PORTC
                SW      EQU     PORTD
          ORG     0x0000
;********* initial port **************
                bsf             STATUS,RP0      ; select bank 1
                clrf            TRISC           ; port c is output
                movlw           0xff
                movwf           TRISD           ; port d is input
                bcf             STATUS,RP0      ; select abnk 0
                clrf            LED             ; clear LED
loop            movf            SW,w            ; รับข้อมูลจากสวิตช์ หรือ พอร์ต D
                movwf           LED             ;  นำข้อมูลที่ได้ไปแสดงผลที่ พอร์ต C
                goto            loop
              end
```

ตัวอย่างที่ 2 โปรแกรมการสื่อสาร RS232

```
list p=16f877                    ; list directive to define processor
   #include <p16f877.inc>        ; processor specific variable definitions
   __CONFIG _CP_OFF & _WDT_OFF & _BODEN_ON & _PWRTE_ON & _XT_OSC & _WRT_ENABLE_ON &
_LVP_ON & _DEBUG_OFF & _CPD_OFF
                    offset   EQU     0x20
                    temp     EQU     0x21
       ORG     0x0000

;************ initial ******************
init       bsf        STATUS,RP0          ; select bank 1
           movlw      .25                 ; BAUD rate 9600
           movwf      SPBRG
           clrf       TXSTA               ; 8 bits data ,no,1 stop
           BSF        TXSTA,BRGH          ; HI SPEED
           bcf        STATUS,RP0          ; select bank 0
           bsf        RCSTA,SPEN          ; Asynchronous serial port enable
           bsf        RCSTA,CREN          ; continuous receive
           bsf        STATUS,RP0          ; select bank 1
           bsf        TXSTA,TXEN          ; Transmit enable
           bcf        STATUS,RP0          ; select bank 0
;********** start to send **********************
           clrf       offset              ; load offset of character table
start      movf       offset,w
           call       TAB
           addlw      0                   ; Character = 00 ?
           btfsc      STATUS,Z            ; Character = 00 ?
           goto       wait2               ; Yes , Z = 1
                                          ; No  , Z = 0
           movwf      TXREG               ; Send recent data to TX
wait1      movlw      TXSTA               ;
           movwf      FSR                 ; FSR <= TXSTA
           btfss      INDF,1              ; check TRMT bit in TXSTA (FSR)
           goto       wait1               ; TXREG full  or TRMT = 0
           incf       offset,f            ; TXREG empty  or TRMT = 1
           goto       start               ; Send again
```

```
;********** start to receive ******************
wait2   btfss       PIR1,RCIF          ; Check RCIF  bit in PIR1 register
        goto        wait2              ; RCREG empty or RCIF = 0
        movf        RCREG,w            ; RCREG full or RCIF = 1
        movwf       TXREG
        goto        wait2
;********* Tebel of message ********************
TAB     addwf   PCL                    ; Move offset to PC lower
        retlw   "E"
        retlw   "T"
        retlw   "T"
        retlw   " "
        retlw   "C"
        retlw   "O"
        retlw   ","
        retlw   "L"
        retlw   "T"
        retlw   "D"
        retlw   0xA
        retlw   0xD
        retlw   0
        END
```

การหาค่า Baud Rate

SYNC	BRGH = 0 (Low Speed)	BRGH = 1 (High Speed)
0	(Asynchronous) Baud Rate = Fosc/(64(X+1))	Baud Rate = Fosc/(16(X+1))
1	(Synchronous) Baud Rate = Fosc/(4(X+1))	N/A

X = value in SPBRG (0 to 255)

4.2 การพัฒนาโปรแกรมด้วยภาษา BASIC (Pic Basic Pro)

ภาษา BASIC เป็นภาษาที่ได้รับความนิยมอย่างแพร่หลายเนื่องจาก เรียนรู้ง่ายมีฟังก์ชั่นการทำงานต่างๆ ที่สามารถทำงานได้เหมือนกับการเขียนด้วย แอสเซมบลี และผู้พัฒนาไม่จำเป็นต้องมีความรู้ในส่วนของฮาร์ดแวร์มากนัก ก็สามารถเขียนโปรแกรมควบคุมไมโครคอนโทรลเลอร์ได้ ซึ่ง Soft Ware จะไม่ยึดติดกับ Hard Ware ทำให้เมื่อเปลี่ยน CPU เป็นเบอร์ใหม่ก็ยังสามารถใช้โปรแกรมเดิมได้ และยังสามารถเขียนภาษาแอสเซมบลีเข้าไปได้อีกด้วย ทำให้มีความอ่อนตัวในการใช้งานพอสมควร ในการอธิบายนี้จะยึดซอฟแวร์ Pic Basic Pro ของ microEngineering Lab ซึ่งผู้ใช้งานจะต้องมีพื้นฐานในภาษาเบสิกพอสมควร แต่ก็ไม่ยากหากจะเริ่มศึกษาเพราะเป็นภาษาที่เข้าใจง่าย

องค์ประกอบหลักๆ ของภาษาเบสิก(Pic Basic Pro)

- การกำหนดตัวแปร

Label **var** Size

เช่น

```
Temp  var  word          ; ตัวแปรประเภท word
Temp1 var  byte          ; ตัวแปรประเภท byte
Temp2 var  bit           ; ตัวแปรประเภท bit
SW_1  var  PORTA.0       ; ตัวแปรที่เป็นรีจิสเตอร์
```

- การกำหนดตัวแปรชนิด Arrays

Label **var** Size[Number of elements]

เช่น Cat **var byte[10]**

Dog **var bit[8]**

- การกำหนดค่าคงที่ Constants

Label **CON** Constant expression

เช่น dataA **CON** 8

- การกำหนดตัวแปร String จะต้องเขียนอยู่ในเครื่องหมาย “ ”

เช่น **“Hello”**

- การกำหนด Labels ซึ่งในการเขียนโปรแกรมที่มีการกระโดดจำเป็นต้องมี Labels เพื่อบอกตำแหน่งปลายทาง รูปแบบคือ ชื่อลาเบล ตามด้วยเครื่องหมาย colon(:) เช่น

```
here:  serout  PORTC.6,T9600,[“Hello”,13,10]
       goto    here
```

- การเขียน Comments จะใส่เครื่องหมาย single quote(‘) หรือ semi-colon (;) นำหน้าข้อความที่ไม่ต้องการให้ CPU ประมวลผล

```
เช่น  Lcdout “Hello”  ‘ Output String “Hello”
      Lcdout “Hello”  ; Output String “Hello”
```

- การเขียนค่าคงที่เลขฐาน สอง,สิบ และ สิบหก

ฐานสองจะใช้เครื่องหมาย % นำหน้าตัวเลข เช่น **%10010100**

ฐานสิบจะเขียนตามปกติ เช่น **100**

ฐานสิบหกจะใช้เครื่องหมาย $ นำหน้าตัวเลข เช่น **$9F**

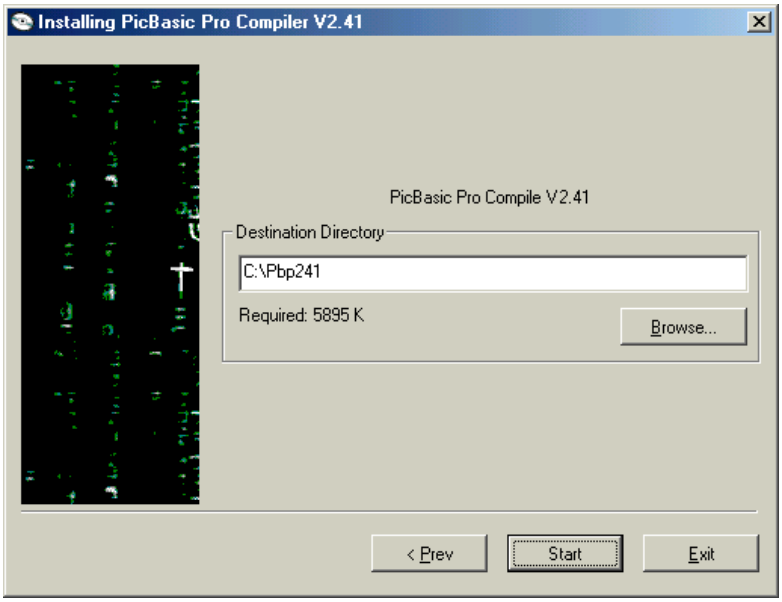
- การกำหนด INCLUDE จะต้องใส่ไว้บนสุดของโปรแกรม

เช่น **INCLUDE “modedefs.bas”**

ขั้นตอนการติดตั้งโปรแกรม PicBasic Pro Compiler V2.41

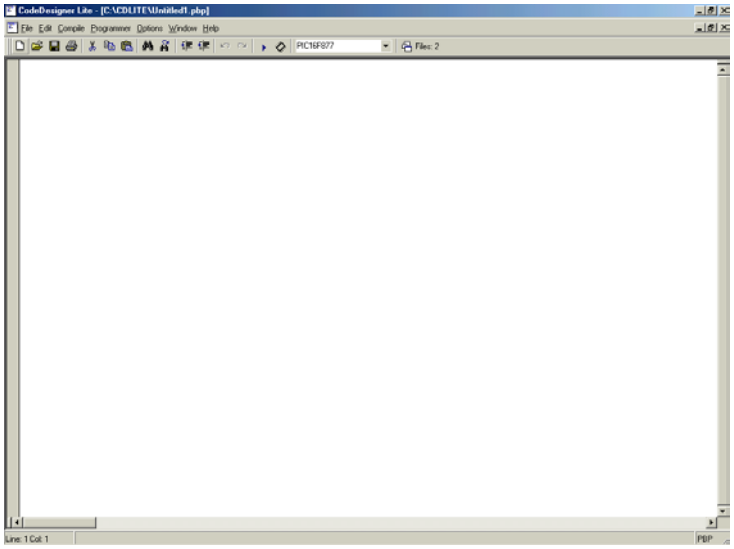
1. ใส่แผ่น CD โปรแกรมแล้วเข้าไปที่ Folder PicBasic Pro จะเห็นไฟล์ชื่อ pbp241.exe ให้ Double Click จะมีหน้าต่างแสดงรายละเอียดต่างดังรูป

2. คลิกที่ปุ่ม Next ให้ใส่ชื่อ Drive และ Folder ที่ต้องการจากคลิก Start เพื่อเริ่มติดตั้งโปรแกรม

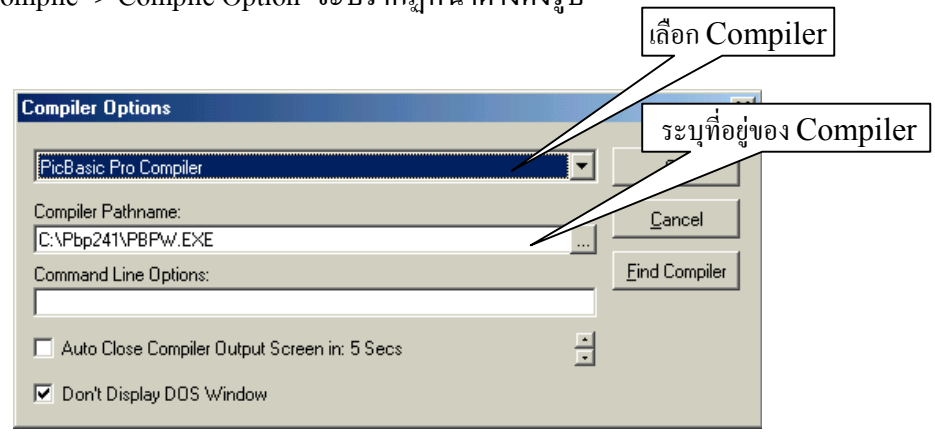


การใช้งาน PicBasic Pro Compiler ในการใช้งานเราจะต้องมีตัว Editor ที่ทำหน้าที่เขียนโปรแกรมอาจจะใช้ตัว Editor ตัวใดก็ได้ แต่ในที่นี้จะแนะนำโปรแกรม CodeDesingner Lite V1.63 ซึ่งทำงานบน Windows ทำให้สะดวกต่อการใช้งาน โดยเริ่มจากการติดตั้งโปรแกรมลงเครื่องด้วยการเข้าไปใน Folder ของ PicBasic Pro แล้ว Double Click ที่ไฟล์ CDLite163.exe โปรแกรมก็จะเริ่มทำการติดตั้งเองโดยอัตโนมัติและเมื่อเราเปิด

โปรแกรม CodeDesingner Lite V1.63 ก็จะประกอบไปด้วยส่วนต่างๆดังรูป แต่ยังไม่สามารถใช้งานได้ทันที โดยเราจะต้อง ทำการเชื่อมโยงระหว่างตัว Editor กับ Compiler เสียก่อน ซึ่งสามารถทำได้ดังนี้



เข้าไปที่เมนู Compile -> Compile Option จะปรากฏหน้าต่างดังรูป



ให้เลือกตัว Compiler โดยเลือกเป็น PicBasic Pro Compiler และ ระบุที่อยู่ของ Compiler ดังรูปแล้วคลิก OK โปรแกรมต่างๆก็จะพร้อมใช้งาน ซึ่งสามารถที่จะสร้าง Project ใหม่หรือ เปิด Project ที่มีอยู่แล้ว

ตัวอย่างการเขียนโปรแกรมภาษา BASIC

ตัวอย่างที่ 1 การใช้งานคำสั่ง การรับส่งข้อมูลผ่าน RS232

```
        INCLUDE "modedefs.bas"          ' Include serial modes
        SI      VAR     PORTC.7         ' กำหนด PORTC.7 เป็น RX
        SO      VAR     PORTC.6         ' กำหนด PORTC.6 เป็น TX
        B0      VAR     Byte
Loop:   SERIN SI,T9600,B0               ' รอรับข้อมูลจาก RS232
        SerOut SO,T9600,[B0]            ' ส่งข้อมูลใน B0 ออกไปยังพอร์ต RS232
        Goto   Loop
```

ตัวอย่างที่ 2 การใช้งาน ADC

```
            INCLUDE "modedefs.bas"          ' Include serial modes
            Adval   VAR     WORD            ' สร้างตัวแปรสำหรับเก็บค่าข้อมูล
            SO      VAR     PORTC.6         '  กำหนดให้ PORTC.6 เป็น TX
M_10BIT:    TRISA = %11111111               '  PORTA   ทั้งหมดเป็นอินพุต
            ADCON1 = %10000010              ' Set PORTA analog and RIGHT justify result
            ADCON0 = %11000001              ' Configure and turn on A/D Module
loop:       ADCON0.2 = 1                    ' Start Conversion
notdon:     Pause 5
          IF ADCON0.2 = 1 Then notdon       ' Wait for low on bit-2 of ADCON0, conversion finished
            adval.highbyte = ADRESH         ' Move HIGH byte of result to adval
            adval.lowbyte = ADRESL          ' Move LOW byte of result to adval
            serout SO,T9600,[12,"Value = ",#adval]   ‘ ส่งข้อมูลออกไปแสดงผลที่พอร์ต RS232
           Pause 300                        ' Wait
           GoTo loop                        ' Do it forever
```

ตัวอย่างที่ 3 การใช้งาน RTC DS1307

```
          INCLUDE "modedefs.bas"            ' Include serial modes
          SO      VAR     PORTC.6           '  PORTC.6   is  TX
          DPIN    VAR     PORTC.4           ' I2C data pin
          CPIN    VAR     PORTC.3           ' I2C clock pin
          B1      VAR     BYTE
          B2      VAR     BYTE
          B3      VAR     BYTE
          BB      VAR     BYTE
          B_H     VAR     BYTE
          B_L     VAR     BYTE
             I2CWrite    DPIN,CPIN,$D0,$00,[$00,$00,$00]     ' เขียนข้อมูล วินาที , นาที และ ชั่วโมง
             Pause 10                                        ' Delay 10ms after each write
loop:        I2CRead DPIN,CPIN,$D0,00,[B1,B2,B3]             ' อ่านข้อมูล วินาที, นาที และ ชั่วโมง
             pause 10
             BB = B1
             GOSUB CONV                                      ' โปรแกรมย่อย CONV
```

```
        SerOut SO,T9600,[12,"SEC  ",#B_H,#B_L,10,13]    ' ส่งข้อมูล วินาทีไปแสดงผล
          BB = B2
         GOSUB CONV
          SerOut  SO,T9600,["MIN  ",#B_H,#B_L,10,13]        ' ส่งข้อมูล นาทีไปแสดงผล
          BB = B3
          GOSUB CONV
          SerOut  SO,T9600,["HOUR  ",#B_H,#B_L,10,13]     ' ส่งข้อมูลชั่วโมงไปแสดงผล
          Pause   200
          GoTo     loop
CONV:     B_L = BB & %00001111                             ' แปลงข้อมูลเป็น 2 ไบต์
          B_H = BB & %11110000
          B_H = B_H >> 4
        RETURN
```

ตัวอย่างที่ 4 การใช้งาน LCD โหมด 4 Bit

```
INCLUDE "modedefs.bas"              ' Include serial modes
Define  LCD_DREG   PORTD
Define  LCD_DBIT   4                ' Data  = PORTD < 7:4 >
Define  LCD_RSREG  PORTD
Define  LCD_RSBIT  2                ' RS  = PORTD.2
Define  LCD_EREG   PORTD
Define  LCD_EBIT   3                ' E = PORTD.3
Define  LCD_BITS   4                ' LCD  mode  4  bit  data
        Pause 500                   ' Wait for LCD to startup
loop:   LCDOut $fe, 1               ' Clear LCD screen
        LCDOut "CP-PIC877"          ' Display
        Pause 1000                  ' Wait
        LCDOut $fe, 1               ' Clear LCD screen
        LCDOut "ETT CO.,LTD"
        Pause 1000                  ' Wait
      GoTo   loop                   ' Do it forever
END
```