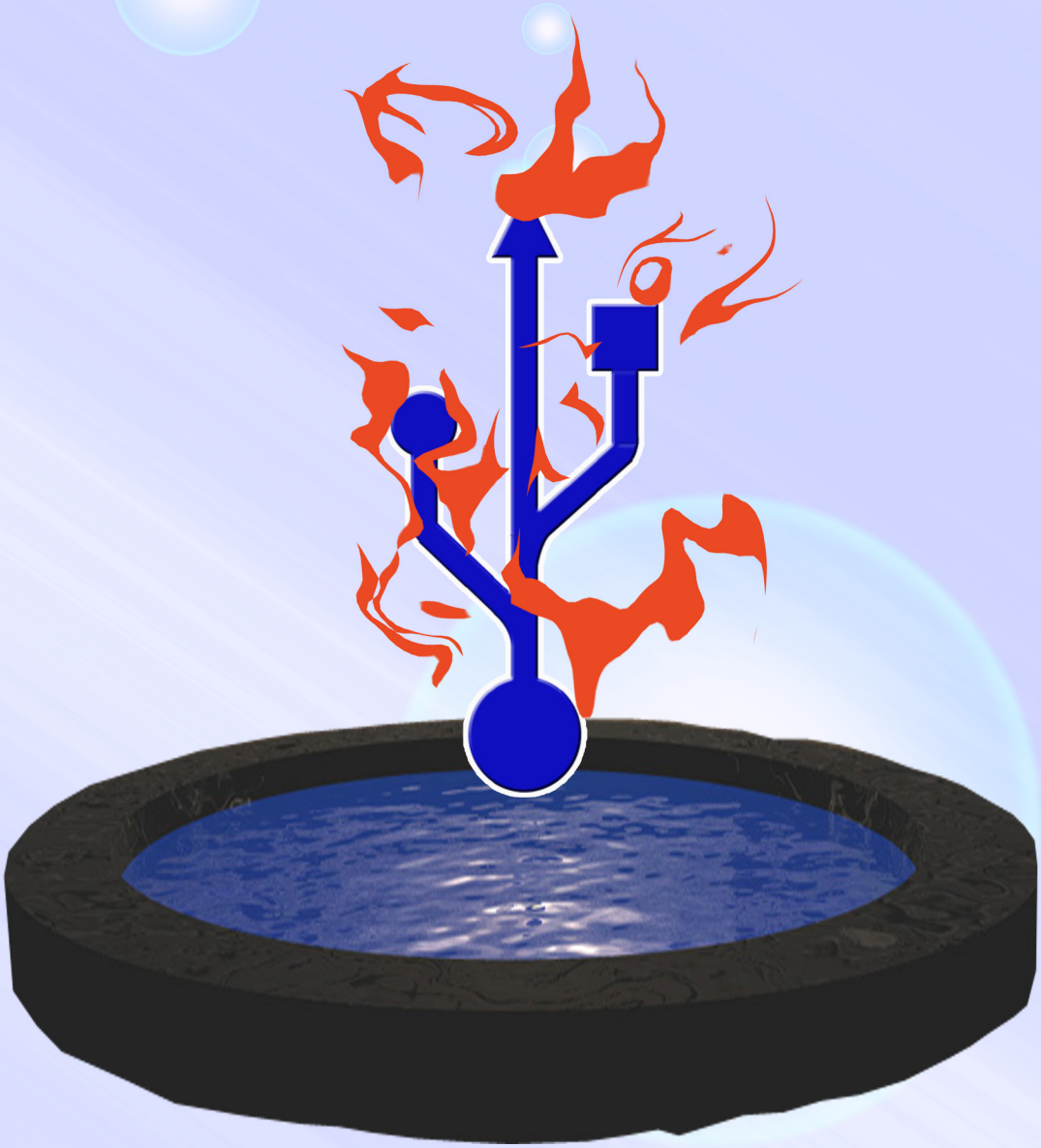
The Examples of CP-JR51USB V1.0



ETT
www.ett.co.th

1112/96-98 ถนนสุขุมวิท แขวงพระโขนง เขตคลองเตย กรุงเทพ 10110

1112/96-98 Sukhumvit Rd., Phrakanong Klongtoey BANKOK 10110

TEL. 02-712 1120 FAX : 02-391 7216

Contents

- Examples 1 : [P0.asm] 1
- Examples 2 : [24xx256_w.asm] 7
- Examples 3 : [24LC256_R_LED.asm] 12
- Examples 4 : [24xx256_2_UART.asm] 17
- Examples 5 : [DS1307_RW.asm] 23
- Examples 6 : [TWI_PCF8574A_CONST.asm] 29
- Examples 7 : [UART MODE1_INT_BRG.asm] 36

ภาคผนวก ก. การติดตั้ง และ การใช้งานภาษา C (Keil) เบื้องต้น

ภาคผนวก ข. การติดตั้ง และ การใช้งาน Cross32 V4.0 เบื้องต้น

ภาคผนวก ค. คำสั่งเทียมใน Cross 32 V4.0

Introduction

เนื้อหาภายในหนังสือคู่มือ The examples of CP-JR51USB v1.0 นี้ภายในประกอบไปด้วยตัวอย่างโปรแกรมที่เขียนด้วยภาษา Assembly, สำหรับรองรับ Hardware ต่างๆ ของบอร์ด CP-JR51USB v1.0 นี้

สำหรับผู้สนใจการเขียนด้วยภาษา C (Keil) ผู้อ่านสามารถเข้าไปดูตัวอย่างโปรแกรมได้ที่เวปไซด์

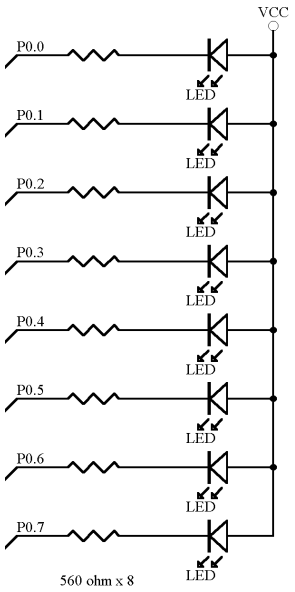
- http://www.atmel.fi/dyn/products/product_card.asp?part_id=3011
- http://www.keil.com/dd/chip/3533.htm

ซึ่งภายในเวปไซด์เหล่านี้จะมีตัวอย่างโปรแกรมทั้งภาษา C และ ภาษา Assembly มากมายให้ผู้อ่านได้ศึกษา ดังนั้น ผู้เขียนจะไม่ขออธิบายในหนังสือคู่มือนี้

การใช้งาน Port บน AT89C5131

การใช้งานกับพอร์ต P0 จนถึง P3 นั้นจะเหมือนกับ มาตรฐาน 8051 ปกติ ซึ่งผู้อ่านสามารถหาอ่านรายละเอียดได้จากหนังสือทั่วไป ในส่วนของพอร์ต P4 จะใช้เป็นการสื่อสารแบบ Two Wire Interface (TWI) หรือ I^2C

Examples 1: [P0.asm] โปรแกรมนี้เป็นการใช้งานพอร์ต P0 เพื่อขับ LED ให้เป็นไฟวิ่ง โดยผู้อ่านจะต้องต่อ Hardware ไว้ที่พอร์ต P0 ด้วย



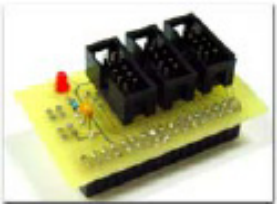
```
;*********************************************************************
; DESCRIBTION : THIS PROGRAM IS ROTATE RIGHT OF PORT 0
; FILE NAME   : P0.ASM
; COMPILER    : CROSS 32
; SUPPORT     : AT89C5131
; DATE        : 7 August 2004
;*********************************************************************

       CPU    "8051.TBL"    ; Processor declaration
       HOF    "INT8"        ; Intel 8-bit hexcode
       INCL    "AT89C5131.SFR"

;***********************************
;      DEFINE
;***********************************
DATA    EQU   0FEH
;***********************************

       ORG    0000H                ; Reset vector
       ANL    CKCON0,#11111110B    ; SYSTEM CLOCK = 12CLK/MACHINE
;      ORL    CKCON0,#00000001B    ; SYSTEM CLOCK = 6CLK/MACHINE
START: MOV    A,#DATA              ; DEFIND FIRST VALUE
LOOP:  RR     A
       MOV    P0,A
       ACALL  DELAY
       SJMP   LOOP

;***********************************
;      DELAY
;***********************************
DELAY:  MOV     R2,#100
L1:     MOV      R3,#50H
L2:     MOV      R4,#5H
        DJNZ     R4,$
        DJNZ     R3,l2
        DJNZ     R2,L1
        RET
        END
```

คำอธิบาย

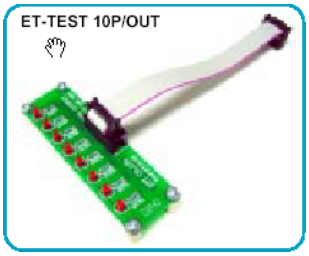
โปรแกรม P0.asm นี้เป็นโปรแกรมไฟวิ่งออกทางพอร์ต P0 ซึ่งจากโปรแกรมด้านบน จะมี คำสั่ง `ANL CKCON0,#11111110B` ซึ่งทำหน้าที่กำหนดให้ 1 Machine cycle นั้นใช้เวลา 12 clock , ในส่วนของการกำหนดให้ LED เลื่อนนั้นจะเป็นหน้าที่ของคำสั่ง RR A

ในการเขียนโปรแกรมทุกครั้ง, ผู้อ่านควรจะใช้คำสั่ง `ANL CKCON0,#11111110B` (12clk/M) หรือคำสั่ง `ORL CKCON0,#00000001B` (6clk/M) ทุกครั้งเพื่อป้องกันความผิดพลาดในส่วนของการกำหนดการทำงานของโปรแกรม (CPU จะทำงานเร็วขึ้น) เนื่องจากว่าโปรแกรม Flip ก็สามารถที่จะกำหนดค่านี้ได้เช่นกันจากช่อง X2 แต่การกำหนดนี้จะไม่มีผลถ้าผู้อ่านใส่คำสั่งใด คำสั่งหนึ่งทางด้านบน

จากการทดลองโปรแกรมของผู้เขียน, ผู้เขียนใช้บอร์ด ET-COV 34 TO 10 รวมกับบอร์ด ET-TEST 10P/OUT



รูปบอร์ดบอร์ด ET-COV 34 TO 10

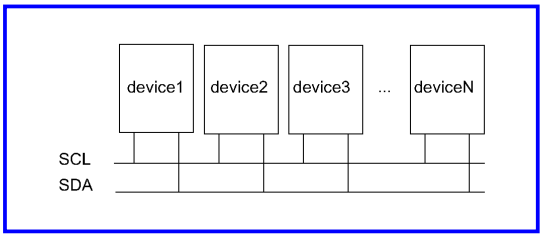


รูปบอร์ดบอร์ด ET-TEST 10P/OUT

จากรูปด้านบนบอร์ด ET-COV 34 TO 10 จะทำหน้าที่เปลี่ยนจากพอร์ต 34 Pin ให้กลายเป็นพอร์ต 10 Pin ได้แก่ P0, P1, P2 เพื่อให้ใช้งานได้กับบอร์ด ET-TEST 10P/OUT

Two Wire Interface (TWI)

Two Wire Interface คือ การสื่อสารอนุกรมมาตรฐาน แบบ Synchronous โดยใช้สายสัญญาณเพียง 2 เส้น คือ SCL(Serial Clock) และ SDA(Serial Data) สื่อสารแบบ Bi-directional โดยอัตราการส่งถ่ายข้อมูลสูงสุดจะอยู่ที่ 400Kbit/s ที่ Standard mode รูปด้านล่างแสดงการติดต่อโดยทั่วไปของบัส TWI (I^2C)



การติดตั้งระบบบัส TWI

CPU จะติดต่อกับส่วน TWI ผ่านทางรีจิสเตอร์ SFRs เหล่านี้ คือ

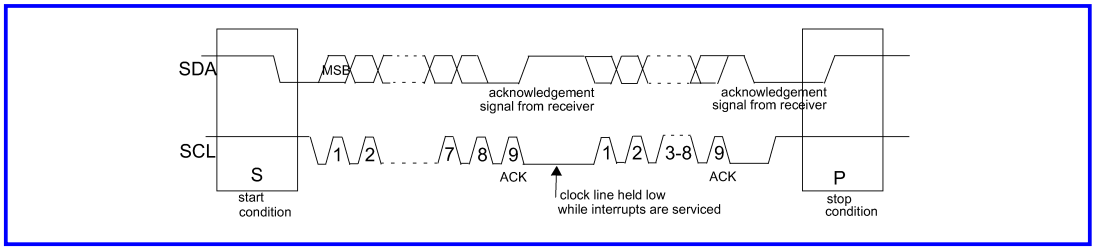
- SSCON : ดูตารางที่ 84 และ 79 ใน Datasheet
- SSDAT : ดูตารางที่ 85 ใน Datasheet
- SSCS : ดูตารางที่ 86 ใน Datasheet
- SSADR : ดูตารางที่ 87 และ 88 ใน Datasheet

รีจีสเตอร์ SSCON : ใช้สำหรับเปิดการใช้งาน TWI , กำหนดค่า bit rate (ตาราง 79) , กำหนดการทำงานเป็น Slave mode , ส่งเงื่อนไข Start หรือ Stop เป็นต้น นอกจากนั้นในขณะที่ CPU รีเซ็ต การทำงานในส่วนของ TWI จะถูก Disable

รีจีสเตอร์ SSCS : ใช้สำหรับเก็บค่า Status code ซึ่งมันจะสะท้อนถึงสถานะ บนบัส TWI โดย 3 บิตแรกจะเป็น ‘0’ เสมอ ส่วน 5 บิตที่เหลือจะใช้เก็บค่า Status code ซึ่งมีทั้งหมด 26 สถานะ โดย Status code ที่ถูกต้องจะเกิดขึ้นหลังจากที่บิต SI ในรีจิสเตอร์ SSCON นั้นถูกเซ็ตโดย Hardware และ ค่าสถานะจะยังคงค้างอยู่จนกว่าบิต SI จะถูกรีเซ็ตโดย Software (ผู้อ่านต้องเขียนโปรแกรมควบคุมบิต SI) โดยตารางที่ 20 – 25 ใน Datasheet เบอร์ AT89C5132 นั้นแสดงค่าสถานะต่างๆ ของแต่ละเหตุการณ์ที่จะเกิดขึ้น ***(สาเหตุที่ผู้เขียนให้ดู Datasheet เบอร์ AT89C5132 เนื่องจากว่าข้อมูลใน Datasheet ของเบอร์ AT89C5131 นั้นมีไม่ครบถ้วน)***

รีจีสเตอร์ SSDAT : ใช้สำเก็บค่าข้อมูล 1 ไบต์ ที่ใช้ส่งออกไปภายนอก หรือ ใช้รับข้อมูล 1 ไบต์จากภายนอกเข้ามา ซึ่งในขณะที่รีจิสเตอร์กำลังทำกระบวนการเลื่อนข้อมูลอยู่นั้นผู้อ่านจะไม่สามารถเข้าถึงรีจิสเตอร์นี้ได้ และ ตราบใดที่ Serial Interrupt Flag (SI) นั้นเซ็ต ‘1’ ข้อมูลในรีจิตเตอร์นี้จะไม่เปลี่ยนแปลง และ ทันทีที่ข้อมูลกำลังเลื่อนออกจากรีจิสเตอร์ SSDAT ข้อมูลนี้จะลงสู่บัสทันที

รีจีสเตอร์ SSADR : ใช้สำเก็บค่าแอดเดรสประจำตัวของ CPU ในกรณีที่ตัวมันเป็น Slave กล่าวคือ ผู้อ่านสามารถกำหนดค่า 7- bit slave address (7 บิต สูงสุด) ซึ่งการส่งถ่ายข้อมูลบนบัส TWI (I^2C) แสดงดังรูปด้านล่าง คือ



การส่งถ่ายข้อมูลบนบัส TWI ที่สมบูรณ์

ในส่วนของโหมดการทำงานจะมี 4 โหมด คือ

- Master Transmitter
- Master Receiver
- Slave Transmitter
- Slave Receiver

โหมดการทำงานทั้ง 4 ผู้เขียนจะอธิบายในรายละเอียดเพียงแค่ 2 โหมดเท่านั้น คือ Master Transmitter และ Master Receiver โดยภายใน 2 หัวข้อนี้จะอธิบายการทำงานด้วยแผนผังภาพ ซึ่งค่าตัวเลขในวงกลมจะเปลี่ยนแปลงเมื่อแฟลก SI เซ็ตเป็น '1' โดย Hardware และ ค่า Status code (ค่าตัวเลขในวงกลม) จะปรากฏในรีจิสเตอร์ SSCS ซึ่ง ณ. จุดนี้จะทำให้โปรแกรมตอบสนองการ Interrupt นั้นทำงาน

การทำโปรแกรมตอบสนอง Interrupt จะไม่หยุดลง อันเนื่องจากการเข้าสู่โหมด Suspend จนกว่าแฟลก SI จะถูก Clear โดย Software

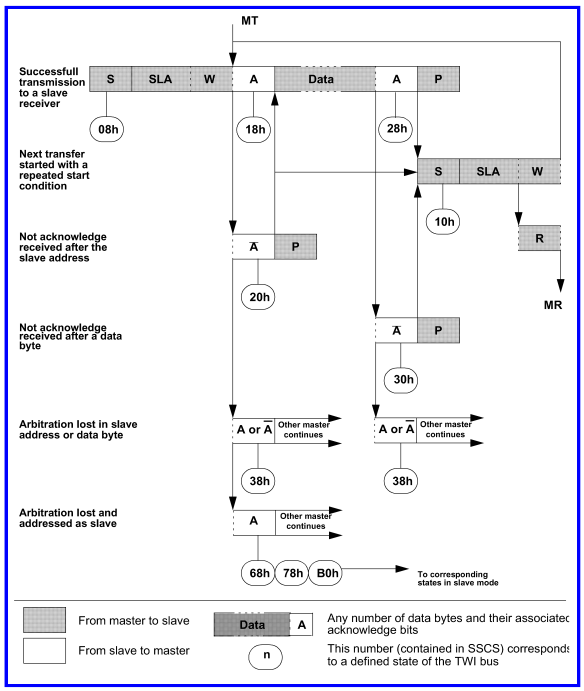
เมื่อเข้าสู่โปรแกรมตอบสนองการ Interrupt , ค่า Status code จะใช้เลือกโปรแกรมการทำงานที่เหมาะสม (ดูในตัวอย่างประกอบ)

Master Transmitter Mode

การทำงานในโหมดนี้ คือ กำหนดให้ CPU เป็น Master ที่ทำหน้าที่เป็นตัวส่งข้อมูล ซึ่งผู้อ่านจะต้องกำหนดค่าในรีจิสเตอร์ SSCON เป็นดังนี้ คือ

CR2	SSIE	STA	STO	SI	AA	CR1	CR0
Bit rate	1	0	0	0	X	Bit rete	Bit rate

จากตารางข้างบน, บิต CR0, CR1 และ CR2 จะใช้กำหนดค่า Internal bit rate (ในกรณีที่ไม่ได้ Enable การทำงานในส่วน External bit rate generator , บิต SSIE ผู้อ่านจะต้องเซ็ตเป็น '1' เพื่อ Enable การทำงานส่วนควบคุมการติดต่อสื่อสารแบบ TWI (SSLC)



รูปแบบการส่งข้อมูล และ สถานะต่างๆ

ทันทีที่บิต STA ถูกเซ็ตผู้อ่านจะเข้าสู่โหมดการทำงานของ Master transmitter mode ทันที โดยบัสจะถูกตรวจสอบว่าอยู่ในสถานะว่าง หรือ ไม่ ถ้าว่างก็จะสร้างสัญญาณ START condition ทันที จากนั้นบิตแฟลก SI ในรีจิสเตอร์ SSCON จะเซ็ตเป็น '1' โดย Hardware ทันที และโปรแกรมกระโดดไปทำโปรแกรมตอบสนองการ Interrupt ส่งผลให้ค่า Status code ในรีจิสเตอร์ SSCS มีค่า 08H ซึ่งค่า 08H นี้จะใช้เป็นตัวชี้เพื่อเลือกโปรแกรมการทำงานต่างๆ ใน โปรแกรมตอบสนองการ Interrupt ซึ่งในที่นี้ค่าในรีจิสเตอร์ SSCS คือ 08H ดังนั้น โปรแกรมย่อยที่ถูกกระทำควรจะเป็นการส่งค่า Slave address และ ค่า Data direction bit (SLA + W) ไปให้อุปกรณ์ Slave (ผู้อ่านควรจะดูรูปแผนผังทางด้านบน ประกอบ)

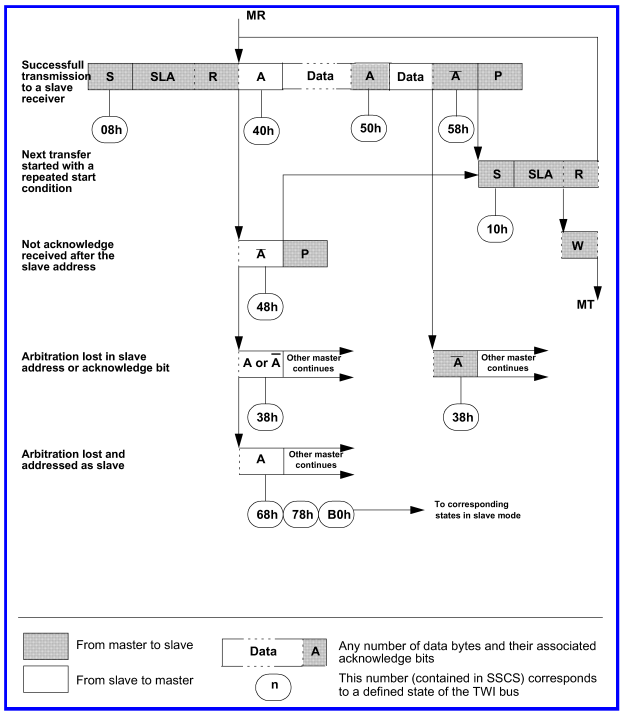
เมื่ออุปกรณ์ Slave ได้รับข้อมูลแอดเดรส และบิตกำหนดทิศทางแล้ว Slave จะส่งบิต Acknowledge –

ment กลับมาให้ CPU จากนั้นบิต SI จะถูกเซ็ตอีกครั้ง และ ค่า Status code ในรีจิสเตอร์ SSCS จะมีค่าต่างๆ ได้ตามนี้ คือ 18H, หรือ 20H หรือ 38H ซึ่งขึ้นอยู่กับการตอบกลับของอุปกรณ์ Slave ดูตารางที่ 48 ใน Data - sheet ประกอบ

เมื่อผู้อ่านส่งเงื่อนไข Start ไปแล้วบิต SSIE, CR2, CR1, และ CR0 จะไม่มีผลต่อการสื่อสารบนบัส TWI อีก และ ในกรณีที่ผู้อ่านส่งเงื่อนไข Start เป็นครั้งที่ 2 หน่วยควมคุมบัส TWI (SSLC) จะสลับโหมดการทำงานเป็น Master receiver mode (status code = 10H) และ จะส่งค่า SLA + R ออกไป

Master Receiver Mode

การทำงานในโหมด Master Receiver Mode นี้ จะมีรูปแบบการรับข้อมูล และ สถานะต่างๆ ดังนี้ คือ



การส่งถ่ายข้อมูล (Transfer) เริ่มแรกจะกำหนดการทำงานเป็น Master transmiter mode เสมอ เมื่อมีเงื่อนไข START เกิดขึ้นโปรแกรมตอบสนองการ Interrupt จะโหลดค่า (SLA + R) จากรีจิสเตอร์ SSDAT ออกไปให้อุปกรณ์ Slave และ บิต SI (Serial interrupt flag) จะเซ็ต ซึ่งผู้อ่านเคลียบิตนี้ก่อนที่การส่งข้อมูลจะทำงาน

เมื่อ Slave address กับ Direction bit ถูกส่งออกไปสมบูรณ์แล้ว อุปกรณ์ Slave จะส่งสัญญาณ A , Acknowledgement กลับออกมาให้กับ CPU เพื่อบอกว่าข้อมูลได้รับครบแล้ว จากนั้นบิต SI (Serial interrupt flag) จะถูกเซ็ตอีกครั้งหนึ่ง และ ค่าตัวเลขในรีจิสเตอร์ที่เป็นไปได้ คือ 40H, 48H, หรือ 38H ซึ่งต้องขึ้นอยู่กับเงื่อนไขต่างๆ (ดู Datasheet เพิ่มเติม)

ในส่วนของการกำหนดอัตราความเร็วในการรับส่งข้อมูลนั้นผู้อ่านสามารถดูได้จากตารางด้านล่าง แต่เนื่องจากในตารางมีค่า X'tal อยู่ 2 ค่า คือ 12MHz , และ 16 MHz เท่านั้น ดังนั้น บอร์ด CP-JR51USB v1.0 จะหาค่าที่ CR2, CR1, CR0 = '0' ได้ คือ

			Bit Frequency (kHz)		
CR2	CR1	CR0	F_{OSCA}= 12 MHz	F_{OSCA} = 16 MHz	F_{OSCA} divided by
0	0	0	47	62.5	256
0	0	1	53.5	71.5	224
0	1	0	62.5	83	192
0	1	1	75	100	160
1	0	0	12.5	16.5	960
1	0	1	100	-	120
1	1	0	-	-	60
1	1	1	0.5 < . < 62.5	0.67 < . < 83	96 · (256 - reload value Timer 1) (reload value range: 0-254 in mode 2)

$$BitFrequency(kHz) = 24MHz \div 256$$
$$= 93.75kHz \ldots\ldots\#$$

Examples 2: [24XX256_W.ASM] โปรแกรมนี้เป็นการเขียนข้อมูลจำนวน 256 bytes ลงไปบนหน่วยความจำ EEPROM ตั้งแต่แอดเดรส 00H – 0FFH ด้วยระบบบัส I^2C ในที่นี้ผู้เขียนใช้ชิป EEPROM เบอร์ 24LC256 ซึ่งมีความจุทั้งหมด 32 Kbytes จากโปรแกรมด้านล่าง, ค่าที่เขียนจะเริ่มจาก 10H แล้วจะเพิ่มค่าขึ้นทีละ 1 ไปเลื่อยๆ (ผู้อ่านจะต้องรันโปรแกรม [24xx256_2_UART.ASM] เพื่อดูค่าที่เขียนเข้าไปใน EEPROM ผ่านทางโปรแกรม PROCOMM 3)

```
;*********************************************************************
; DESCRIBTION : THIS PROGRAM IS WRITE DATA TO EEPROM
; FILE NAME   : 24XX256_W.ASM
; COMPILER    : CROSS 32
; SUPPORT     : AT89C5131
; DATE        : 26 August 2004
;*********************************************************************

       CPU    "8051.TBL"    ; Processor declaration
       HOF    "INT8"        ; Intel 8-bit hexcode
       INCL    "AT89C5131.SFR"

;*********************************************************************
;      DEFINE BYTE SECTION
;*********************************************************************
TWI_DATA      EQU    20H
SLAVE_ADR     EQU    21H
ADR_H         EQU    22H
ADR_L         EQU    23H
B_TWI_BUSY    EQU    25H    ; BIT TYPE
TWI_DATA_I    EQU    26H
STOP_BACKUP   EQU    27H

;*********************************************************************
;      DEFINE BIT SECTION
;*********************************************************************
RW            EQU    24H    ; 0=WRITE, 1=READ  ; BIT TYPE
;*********************************************************************

              ORG    0000H         ; Reset vector
START:        LJMP   BEGIN

;*************************************************
;             INTERRUPT SECTION
;*************************************************
              ORG    0043H
              LJMP   TWI_IT

;*************************************************
              ORG    0100H
;*************************************************
;             INITIALIZE EEPROM 24XX256
;*************************************************
BEGIN:        ORL    CKCON0,#00000001B   ; 6 CLK PERIOD/MACHINE
              ORL    SSCON,#01000000B    ; enable TWI */
              ANL    SSCON,#01000100B    ;
              SETB   EA                  ; interrupt enable */
              ORL    IEN1,#02h           ; enable TWI interrupt */
              CLR    B_TWI_BUSY
              MOV    TWI_DATA,#10H       ; data example to send */**
              MOV    ADR_H,#00H
              MOV    ADR_L,#00H
              MOV    STOP_BACKUP,#00H
              MOV    R0,#00H
              MOV    P0,#0FH
```

```
LOOP_W:       MOV    ACC,STOP_BACKUP     ; FIND STOP BIT, IF '1' JUMP
              JB     ACC.4,CONT_W

              JB     B_TWI_BUSY,END_IF_W ; jump if b_TWI_busy bit = '1'*/
              MOV    ACC,SSCON
              JB     ACC.4,END_IF_W      ; jump if acc.4 bit = '1' "STOP FLAG"
                                         ; usually not jump*/
              SETB   B_TWI_BUSY          ; flag busy =1 , now, I'm not Empty.*/
              MOV    SLAVE_ADR,#10100000B; slave adresse example
              CLR    RW                  ; 0= WRITE
              MOV    SSDAT,#00h          ; clear buffer before sending data
              ORL    SSCON,#00100000B    ; TWI start sending */ after that SI
                                         ; bit is set '1'
                                         ; and  the status code in SSCS will be
                                         ; 08h.

              JMP    LOOP_W
; ************************************************
CONT_W:       MOV    STOP_BACKUP,#00H
              INC    ADR_L
              INC    TWI_DATA
              INC    R0

              CJNE   R0,#00H,LOOP_W      ; 0 - 255h
              MOV    P0,#55H
              SJMP   $
END_IF_W:     JMP    LOOP_W

; ************************************************
;             INTERRUPT SERVICE ROUTINE
; ************************************************
TWI_IT:              MOV    R7,SSCS

; ************* TWI status tasking ***************

CASE_00:      CJNE   R7,#00h,CASE_08     ; A start condition has been sent
                                         ; SLR+R/W are transmitted, ACK bit
                                         ; received
              CLR    B_TWI_BUSY          ; TWI is free
              ORL    SSCON,#10H          ; SEND STOP
              LJMP   end_switch

CASE_08:      CJNE   R7,#08h,CASE_10     ; A start condition has been sent
                                         ; SLR+R/W are transmitted, ACK bit
                                         ; received
              ANL    SSCON,#~20h         ; clear start condition

;             send slave adress and read/write bit

              MOV    ACC,slave_adr
              MOV    C,RW
              MOV    ACC.0,C
              MOV    SSDAT,ACC
              LJMP   end_switch

CASE_10:      CJNE   R7,#10h,CASE_18     ; A repeated start condition has been
                                         ; sent
                                         ; SLR+R/W are transmitted, ACK bit
                                         ; received
              ANL    SSCON,#~20h         ; clear start condition
;             send slave adress and read/write bit
              MOV    ACC,slave_adr
```

```
                ;=========
                SETB   RW
                ;=========
                MOV    C,RW
                MOV    ACC.0,C
                MOV    SSDAT,ACC
                JMP    end_switch

CASE_18:        CJNE   R7,#18h,case_20      ; SLR+W was transmitted, ACK bit
                                            ; received

                ANL    SSCON,#11110111B     ; clear SI
                MOV    SSDAT,ADR_H
CHECK_FIR:      MOV    ACC,SSCON
                JNB    ACC.3,CHECK_FIR      ; JUMP IF SI = '0'

                ANL    SSCON,#11110111B     ; clear SI
                MOV    SSDAT,ADR_L
CHECK_SEC:      MOV    ACC,SSCON
                JNB    ACC.3,CHECK_SEC      ; JUMP IF ACC.3 = '0'

                ANL    SSCON,#11110111B     ; clear SI
                MOV    SSDAT,TWI_data       ; Transmit data byte, ACK bit received
                JMP    end_switch

CASE_20:        CJNE   R7,#20h,case_28      ; SLR+W was transmitted, NOT ACK bit
                                            ; received
                ORL    SSCON,#10h           ; Transmit STOP
                CLR    b_TWI_busy           ; TWI is free
                JMP    end_switch

CASE_28:        CJNE   R7,#28h,case_30      ; DATA was transmitted, ACK bit
                                            ; received
                ;ANL   SSCON,#11110111B     ; clear SI
                ORL    SSCON,#10h           ; send STOP
                MOV    STOP_BACKUP,SSCON    ; BACKUP
                CLR    b_TWI_busy           ; TWI is free
                JMP    end_switch

CASE_30:        CJNE   R7,#30h,case_38      ; DATA was transmitted, NOT ACK bit
                                            ; received
                ORL    SSCON,#10h           ; Transmit STOP
                CLR    b_TWI_busy           ; TWI is free
                JMP    end_switch

CASE_38:        CJNE   R7,#38h,case_40      ; Arbitration lost in SLA+W or DATA.
                ORL    SSCON,#10h           ; Transmit STOP
                CLR    b_TWI_busy           ; TWI is free
                JMP    end_switch

CASE_40:        CJNE   R7,#40h,case_58      ; As soon as, the DATA will move to
                                            ; SSDAT SFR.
                ANL    SSCON,#11110111B     ; clear SI
S40:            MOV    ACC,SSCON
                JNB    ACC.3,S40            ; JUMP IF SI = '0'
                MOV    TWI_DATA_I,SSDAT     ; Master will send ACK or NACK to slave

                ;=================================
                ;ORL SSCON,#00000100B        ; set AA  */ SENT ACK
                ANL SSCON,#11111011B        ; CLR AA  */ SENT NACK ( DEFAULT USED )
                ;=================================
                LJMP   end_switch
```

```
CASE_58:      CJNE   R7,#58h,case_5X
              ORL    SSCON,#00010000B    ; send STOP
              MOV    STOP_BACKUP,SSCON   ; BACKUP
              CLR    b_TWI_busy          ; TWI is free
              JMP    end_switch

CASE_5X:
end_switch:   ANL    SSCON,#11110111B    ; clear SI flag
              RETI

; ************************************************
;             DELAY
; ************************************************
DELAY:        MOV    R7,#9
D1:           MOV    R6,#5
D2:           NOP
              NOP
              DJNZ   R6,D2
              DJNZ   R7,D1
              RET

; ************************************************
;             DELAY
; ************************************************
DELAYS:              MOV    R7,#25
D1S:          MOV    R6,#20
D2S:          NOP
              NOP
              DJNZ   R6,D2S
              DJNZ   R7,D1S
              RET
              END
```

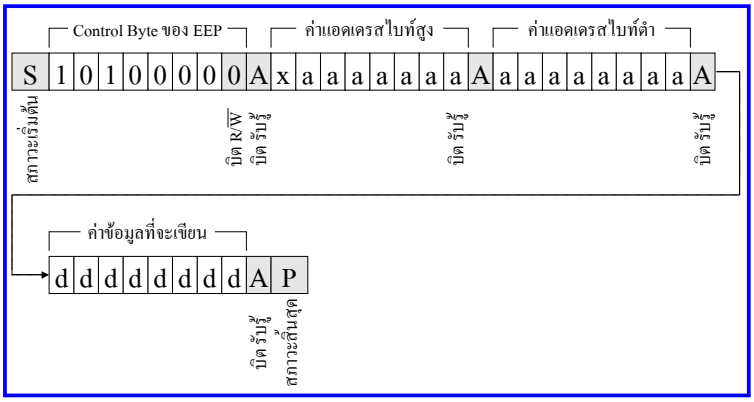
คำอธิบาย

จากที่ผู้อ่านได้ศึกษาทฤษฎีในตอนต้นมาแล้ว, ในโปรแกรมทางด้านบนเมื่อมองในภาพกว้างๆ แล้วจะเห็นว่าตั้งแต่ BEGIN ลงมาจนถึง END_IF นั้น จะเป็นส่วน BODY ของโปรแกรม และ ต่อจากนี้ลงไปจะเป็นโปรแกรมย่อย ซึ่งจะทำงานเมื่อมีการ Interrupt เกิดขึ้น และ เนื่องจากโปรแกรมนี้เป็นการเขียนข้อมูลเพียงอย่างเดียว ดังนั้น CASE_XX ใดๆ ในโปรแกรมย่อยที่จำเป็นต้องใช้งานคือ CASE_08, CASE_18, CASE_28 เท่านั้น ซึ่งอธิบายได้ดังนี้ คือ เริ่มแรกผู้อ่านต้องทราบรูปแบบหรือ Protocol ของการส่งข้อมูลบนระบบบัส I^2C ในแบบมาตรฐานก่อน ซึ่งเป็นมาตรฐานของบริษัท Philips จากนั้นนำรูปแบบการส่ง TWI ของชิป AT89C5131 จัดให้สอดคล้องกับมาตรฐาน I^2C ของ Philips , สำหรับตัวอย่างทางด้านบนมีรูปแบบการส่งเป็นดังนี้ คือ

ซึ่งผู้อ่านจะเห็นว่าโปรแกรมย่อย CASE_08 จะทำหน้าที่ส่ง Control byte ของ EEPROM , CASE_18 จะทำหน้าที่ส่งค่าตำแหน่ง Address ของ EEPROM จำนวน 2ไบต์ และ สุดท้าย CASE_28 จะส่ง STOP condition.

คำเตือน (CAUTION) :
ผู้อ่านต้อนระลึกไว้เสมอว่า การตอบสนองอินเตอร์รัพต์ของ CPU จะไม่เกิดขึ้นในขณะที่ CPU กำลังทำโปรแกรมตอบสนองอินเตอร์รัพต์ของสัญญาณร้องขออินเตอร์รัพต์ที่มีระดับความสำคัญเท่ากันอยู่

Tip : เนื้อหาเกี่ยวกับ I^2C ผู้อ่านสามารถดูได้จากหนังสือ “ รู้จักและเข้าใจ I^2C BUS “ ของบริษัท ETT



แสดงการเขียนข้อมูล 1 byte (24LC256)

Examples 3: [24LC256_R_LED.ASM] โปรแกรมนี้เป็นการอ่านข้อมูลจาก EEPROM จำนวน 1 ไบต์ แล้วนำไปแสดงผลบนพอร์ต 0 ซึ่งผู้อ่านควรต่อ LED ไว้ที่พอร์ตนี้ (common A ผ่าน R = 470 โอห์ม)

```
;*********************************************************************
; DESCRIBTION : THIS PROGRAM IS READ DATA TO EEPROM
; FILE NAME   : 24LC256_R_LED.ASM
; COMPILER    : CROSS 32
; SUPPORT     : AT89C5131
; DATE        : 26 September 2004
; DESCRIBTION : DISPLAY THE DATA IN EEPROM, ADDRESS 00H - 0FFH
;*********************************************************************

       CPU    "8051.TBL"    ; Processor declaration
       HOF    "INT8"        ; Intel 8-bit hexcode
       INCL    "AT89C5131.SFR"

;*********************************************************************
;      DEFINE SECTION
;*********************************************************************
TWI_DATA      EQU    20H
SLAVE_ADR     EQU    21H
ADR_H         EQU    22H
ADR_L         EQU    23H
RW            EQU    24H    ; 0=WRITE, 1=READ  ; BIT TYPE
B_TWI_BUSY    EQU    25H    ; BIT TYPE
TWI_DATA_I    EQU    26H
STOP_BACKUP   EQU    27H
HEX2IN        EQU    30H
ASC_L         EQU    31H
ASC_H         EQU    32H
IE            EQU     0A8H
;*********************************************************************

              ORG    0000H         ; Reset vector
              LJMP   BEGIN

;**************************************************
;             INTERRUPT SECTION
;**************************************************
              ORG    0043H
              LJMP   TWI_IT

;**************************************************
              ORG    0100H

BEGIN:        MOV     SP,#256-32          ;define stack = 32 byte
              ORL    CKCON0,#00000001B    ; 6 CLK PERIOD/MACHINE

;**************************************************
;             INITIALIZE EEPROM 24XX256
;**************************************************
              ORL    SSCON,#40h           ; enable TWI */
              SETB   EA                   ; interrupt enable */
              ORL    IEN1,#02h            ; enable TWI interrupt */
              CLR    B_TWI_BUSY
              MOV    TWI_DATA,#00H        ; data example to send
              MOV    STOP_BACKUP,#00H
              MOV    ADR_H,#00h
              MOV    ADR_L,#00h
              MOV    R0,#00H
```

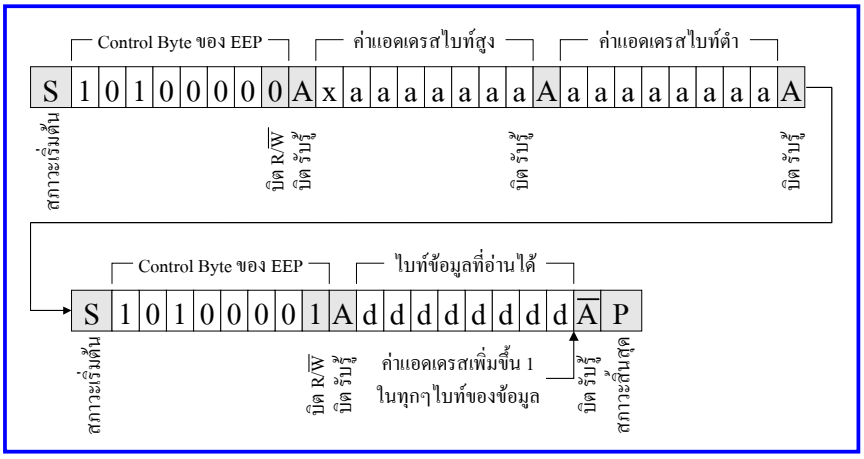
```
;==============================================
;             READ EEPROM 24XX256
;==============================================
LOOP:         MOV    ACC,STOP_BACKUP
              JB     ACC.4,CONT          ; FIND STOP BIT, IF '1' JUMP

              JB     B_TWI_BUSY,END_IF   ; jump if b_TWI_busy bit = '1'*/
              MOV    ACC,SSCON
              JB     ACC.4,END_IF        ; jump if acc.4 bit = '1' "STOP BIT"
                                         ; usually not jump*/
              SETB   B_TWI_BUSY          ; flag busy =1 , now, I'm not Empty.*/
              MOV    SLAVE_ADR,#10100000B; slave adresse example
              CLR    RW                  ; 0=write
              MOV    SSDAT,#00h          ; clear buffer before sending data

              ;=============
START:        ORL    SSCON,#20h          ; TWI start sending */ after that SI bit
                                         ; is set '1'
                                        ; and the status code in SSCS will be 08h.
              JMP    LOOP
;==============================================
;             DISPLAY TO LED
;==============================================
CONT:         MOV    STOP_BACKUP,#00H
              MOV    P0,TWI_DATA_I
              sjmp   $

END_IF:       LJMP   LOOP

; ***********************************************
;             INTERRUPT SERVICE ROUTINE
; ***********************************************
TWI_IT:       MOV R7,SSCS

;============== TWI status tasking  ============

CASE_00:      CJNE   R7,#00h,CASE_08     ; A start condition has been sent
                                         ; SLR+R/W are transmitted, ACK bit
                                         ; received
              CLR    B_TWI_BUSY          ; TWI is free
              ORL    SSCON,#10H          ; SEND STOP
              LJMP   end_switch

CASE_08:      CJNE   R7,#08h,CASE_10     ; A start condition has been sent
                                         ; SLR+R/W are transmitted, ACK bit
                                         ; received
              ANL    SSCON,#~20h         ; clear start condition

;             send slave adress and read/write bit

              MOV    ACC,slave_adr
              MOV    C,RW
              MOV    ACC.0,C
              MOV    SSDAT,ACC
              LJMP   end_switch

CASE_10:      CJNE   R7,#10h,CASE_18     ; A repeated start condition has been
                                         ; sent
                                         ; SLR+R/W are transmitted, ACK bit
                                         ; received
              ANL    SSCON,#~20h         ; clear start condition
;             send slave adress and read/write bit
              MOV    ACC,slave_adr
```

```
                ;=========
                SETB   RW
                ;=========
                MOV    C,RW
                MOV    ACC.0,C
                MOV    SSDAT,ACC
                JMP    end_switch

CASE_18:        CJNE   R7,#18h,case_20      ; SLR+W was transmitted, ACK bit
                                            ; received

FIRST_D:        MOV    SSDAT,ADR_H
                ANL    SSCON,#11110111B     ; clear SI
S18:            MOV    ACC,SSCON
                JNB    ACC.3,S18            ; JUMP IF SI = '0'

                MOV    SSDAT,ADR_L
                LJMP   end_switch

CASE_20:        CJNE   R7,#20h,case_28      ; SLR+W was transmitted, NOT ACK bit
                                            ; received
                ORL    SSCON,#10h           ; Transmit STOP
                CLR    b_TWI_busy           ; TWI is free
                JMP    end_switch

CASE_28:        CJNE   R7,#28h,case_30      ; DATA was transmitted, ACK bit received
                ORL    SSCON,#20h           ; send start
                JMP    end_switch

CASE_30:        CJNE   R7,#30h,case_38      ; DATA was transmitted, NOT ACK bit
                                            ; received
                ORL    SSCON,#10h           ; Transmit STOP
                CLR    b_TWI_busy           ; TWI is free
                JMP    end_switch

CASE_38:        CJNE   R7,#38h,case_40      ; Arbitration lost in SLA+W or DATA.
                ORL    SSCON,#10h           ; Transmit STOP
                CLR    b_TWI_busy           ; TWI is free
                JMP    end_switch

CASE_40:        CJNE   R7,#40h,case_58      ; As soon as, the DATA will move to
                                            ; SSDAT SFR.
                ANL    SSCON,#11110111B     ; clear SI
S40:            MOV    ACC,SSCON
                JNB    ACC.3,S40            ; JUMP IF SI = '0'
                MOV    TWI_DATA_I,SSDAT     ; Master will send ACK or NACK to slave

                ;================================
                ;ORL SSCON,#00000100B        ; set AA  */ SENT ACK
                ANL SSCON,#11111011B        ; CLR AA  */ SENT NACK  ( DEFAULT USED )
                ;================================
                LJMP   end_switch

CASE_58:        CJNE   R7,#58h,case_5X
                ORL    SSCON,#00010000B     ; send STOP
                MOV    STOP_BACKUP,SSCON    ; BACKUP
                CLR    b_TWI_busy           ; TWI is free
                JMP    end_switch

CASE_5X:
end_switch:     ANL    SSCON,#11110111B     ; clear SI flag
                RETI
```

```
; ************************************************
;             DELAY
; ************************************************
DELAY:        MOV    R7,#9
D1:           MOV    R6,#5
D2:           NOP
              NOP
              DJNZ   R6,D2
              DJNZ   R7,D1
              RET

; ************************************************
;             DELAY
; ************************************************
DELAYS:               MOV    R7,#25
D1S:          MOV    R6,#20
D2S:          NOP
              NOP
              DJNZ   R6,D2S
              DJNZ   R7,D1S
              RET

DELAY_1S:     MOV    R7,#150
D1M:          MOV    R6,#50
D2M:          MOV    R5,#50
D3M:          NOP
              NOP
              DJNZ   R5,D3M
              DJNZ   R6,D2M
              DJNZ   R7,D1M
              RET
              END
```

คำอธิบาย

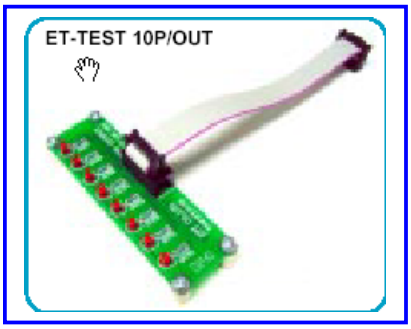
จากที่ผู้อ่านได้ศึกษาตัวอย่างการเขียนข้อมูลให้กับ EEPROM ไปแล้วก่อนหน้านี้ ซึ่งจะเห็นว่าโปรแกรมบริการ Interrupt ที่ถูกใช้งานจะมีแค่ CASE_08, CASE_18, CASE_28 เท่านั้น ส่วนโปรแกรมการอ่าน EEPROM นี้จะใช้ CASE_08, CASE_18, CASE_28, CASE_10, CASE_40 และ CASE_58 (สาเหตุที่ไม่ทำ CASE_50 เนื่องจากผู้เขียนอ่านทีละ 1 byte จึงตอบกลับด้วยสัญญาณ $\overline{A}$), ในการรับข้อมูล โดยเริ่มแรกต้องกำหนดให้เป็นการเขียนข้อมูลไปให้ EEPROM ก่อน ซึ่งรูปแบบการอ่านข้อมูลในแบบมาตรฐาน แสดงดังภาพด้านล่าง คือ



แสดงรูปแบบการรับข้อมูลของ EEPROM

สิ่งที่ผู้อ่านต้องระวัง คือ CASE_18 กับ CASE_28 นั้นจะไม่เหมือนกันกับ การเขียนข้อมูล เนื่องจากว่า CASE_18 ในโปรแกรมการอ่านข้อมูล จะต้องส่งค่า Address ของข้อมูลบนตัว EEPROM เท่านั้น ในขณะที่โปรแกรมการเขียนจะต้องใส่ข้อมูล 1 ไบต์ลงไปในส่วนนี้ด้วย , ในส่วนของ CASE_28 ในโปรแกรมการอ่านนั้นจะไม่ส่งสถานะ STOP แต่จะส่ง START แทนเพื่อจะกระโดดไปทำ CASE_10 เพื่อส่ง SLA + R จากนั้นจะกระโดดไปทำ CASE_40 ของรูปที่ 50 ต่อไป (ดูรูปที่ 49 Master Transmitter Mode ใน Datasheet ประกอบ)

ค่าที่ได้จากการอ่านจะอยู่ใน TWI_DATA_I ซึ่งอยู่ใน CASE_40 และ จะส่งผ่านพอร์ต P0 ซึ่งตรงนี้ผู้อ่านจะต้องต่อ Hardware รองรับไว้ด้วย โดยในที่นี้ผู้เขียนใช้ชุด ET-TEST 10P/OUT ในการทดสอบโปรแกรมนี้



คำเตือน (CAUTION) :
ผู้อ่านต้อนระลึกไว้เสมอว่า การตอบสนองอินเตอร์รัพต์ของ CPU จะไม่เกิดขึ้นในขณะที่ CPU กำลังทำโปรแกรมตอบสนองอินเตอร์รัพต์ของสัญญาณร้องขออินเตอร์รัพต์ที่มีระดับความสำคัญเท่ากันอยู่

Examples 4: [24xx256_2_UART.ASM] โปรแกรมนี้เป็นการอ่านข้อมูลจากบัส TWI แล้วส่งข้อมูลผ่านพอร์ต UART ด้วยอัตรา Baud rate = 115200bps (ก่อนรันโปรแกรมนี้ผู้อ่านควรจะรันโปรแกรม 24XX256_W.ASM มาก่อน และ จะต้องเปิดโปรแกรม Procomm 3 ไว้ด้วย)

```
;*********************************************************************
; DESCRIBTION : THIS PROGRAM IS READ DATA TO EEPROM
; FILE NAME   : 24xx256_2_UART.ASM
; COMPILER    : CROSS 32
; SUPPORT     : AT89C5131
; DATE        : 25 August 2004
; DESCRIBTION : DISPLAY THE DATA IN EEPROM, ADDRESS 00H - 0FFH
;*********************************************************************

       CPU    "8051.TBL"    ; Processor declaration
       HOF    "INT8"        ; Intel 8-bit hexcode
       INCL    "AT89C5131.SFR"

;*********************************************************************
;      DEFINE SECTION
;*********************************************************************
TWI_DATA      EQU    20H
SLAVE_ADR     EQU    21H
ADR_H         EQU    22H
ADR_L         EQU    23H
RW            EQU    24H    ; 0=WRITE, 1=READ  ; BIT TYPE
B_TWI_BUSY    EQU    25H    ; BIT TYPE
TWI_DATA_I    EQU    26H
STOP_BACKUP   EQU    27H
HEX2IN        EQU    30H
ASC_L         EQU    31H
ASC_H         EQU    32H
IE            EQU     0A8H
;*********************************************************************

              ORG    0000H         ; Reset vector
              LJMP   BEGIN

;**************************************************
;             INTERRUPT SECTION
;**************************************************
              ORG    0043H
              LJMP   TWI_IT

;**************************************************
              ORG    0100H

BEGIN:        MOV     SP,#256-32          ;define stack = 32 byte
              ORL    CKCON0,#00000001B    ; 6 CLK PERIOD/MACHINE
;**************************************************
;             UART 115200bps
;**************************************************
INIT_SER:     ORL    CKCON0,#00000001B    ; 6 CLK PERIOD/MACHINE
              ORL    PCON,#11000000B            ; ENABLE THE FRAMING BIT ERROR
DETECTION, DOUBLE BAUD RATE
                                          ; AND CHECK FE BIT OF SCON SFR
              ORL    BDRCON,#00001110B    ; SPD=1,RBCK =1, TBCK =1
              MOV    SCON,#01010000B            ; uart in mode 1 (8 bit), REN=1
              MOV    BRL,#243             ; U CAN SEE THIS VALUE FROM PAGE 71 OF
DATASHEET
              ORL    BDRCON,#00010000B    ; BRR = 1 , START BAUD RATE RUN
              CLR     ES
              CLR     EA
```

```
                ;==================
                ;  CLEAR DISPLAY
                ;==================
                MOV    A,#0CH              ; "CLEAR DISPLAY"
                LCALL  TX_BYTE

                ;==================
                ;  MASSAGE
                ;==================
                MOV    DPTR,#SHOW1
                LCALL  TRANSMIT

;**************************************************
;               INITIALIZE EEPROM 24XX256
;**************************************************
                ORL    SSCON,#40h          ; enable TWI */
                SETB   EA                  ; interrupt enable */
                ORL    IEN1,#02h           ; enable TWI interrupt */
                CLR    B_TWI_BUSY
                MOV    TWI_DATA,#00H       ; data example to send
                MOV    STOP_BACKUP,#00H
                MOV    ADR_H,#00h
                MOV    ADR_L,#00h
                MOV    R0,#00H

;==================================================
;               READ EEPROM 24XX256
;==================================================
LOOP_R:         MOV    ACC,STOP_BACKUP
                JB     ACC.4,CONT_R        ; FIND STOP BIT, IF '1' JUMP

                JB     B_TWI_BUSY,END_IF_R ; jump if b_TWI_busy bit = '1'*/
                MOV    ACC,SSCON
                JB     ACC.4,END_IF_R      ; jump if acc.4 bit = '1' "STOP BIT"
                                           ; usually not jump*/
                SETB   B_TWI_BUSY          ; flag busy =1 , now, I'm not Empty.*/
                MOV    SLAVE_ADR,#10100000B; slave adresse example
                CLR    RW                  ; 0=write
                MOV    SSDAT,#00h          ; clear buffer before sending data

                ;=============
START_R:        ORL    SSCON,#20h          ; TWI start sending */ after that SI bit
                                           ; is set '1'
                                           ; and  the status code in SSCS will be
                                           ; 08h.
                JMP    LOOP_R
;==================================================
;               DISPLAY TO LED
;==================================================
CONT_R:         MOV    STOP_BACKUP,#00H

                MOV    HEX2IN,TWI_DATA_I
                LCALL  DELAYS              ; JUST HAVE
                LCALL  HEX2ASC2            ; OUTPUT STAY IN "ASC_L","ASC_H"

                MOV    A,ASC_H
                LCALL   TX_BYTE

                MOV    A,ASC_L
                LCALL  TX_BYTE

                MOV    A,#48H              ; "H"
                LCALL  TX_BYTE
```

```
                MOV    A,#20H              ; "Space"
                LCALL  TX_BYTE

                INC ADR_L
                INC R0

                CJNE R0,#00H,LOOP_R        ; amount = 256, rang = 0 - 255
                sjmp $

END_IF_R:       LJMP   LOOP_R

; ************************************************
;               INTERRUPT SERVICE ROUTINE
; ************************************************
TWI_IT:         MOV R7,SSCS

;============== TWI status tasking  ============

CASE_00:        CJNE   R7,#00h,CASE_08     ; A start condition has been sent
                                           ; SLR+R/W are transmitted, ACK bit
                                           ; received
                CLR    B_TWI_BUSY          ; TWI is free
                ORL    SSCON,#10H          ; SEND STOP
                LJMP   end_switch

CASE_08:        CJNE   R7,#08h,CASE_10     ; A start condition has been sent
                                           ; SLR+R/W are transmitted, ACK bit
                                           ; received
                ANL    SSCON,#~20h         ; clear start condition

;               send slave adress and read/write bit

                MOV    ACC,slave_adr
                MOV    C,RW
                MOV    ACC.0,C
                MOV    SSDAT,ACC
                LJMP   end_switch

CASE_10:        CJNE   R7,#10h,CASE_18     ; A repeated start condition has been
                                           ; sent
                                           ; SLR+R/W are transmitted, ACK bit
                                           ; received
                ANL    SSCON,#~20h         ; clear start condition

;               send slave adress and read/write bit

                MOV    ACC,slave_adr
                ;=========
                SETB   RW
                ;=========
                MOV    C,RW
                MOV    ACC.0,C
                MOV    SSDAT,ACC
                JMP    end_switch
CASE_18:        CJNE   R7,#18h,case_20     ; SLR+W was transmitted, ACK bit
                                           ; received

FIRST_D:        MOV    SSDAT,ADR_H
                ANL    SSCON,#11110111B    ; clear SI
S18:            MOV    ACC,SSCON
                JNB    ACC.3,S18           ; JUMP IF SI = '0'

                MOV    SSDAT,ADR_L
                LJMP   end_switch
```

```
CASE_20:      CJNE   R7,#20h,case_28      ; SLR+W was transmitted, NOT ACK bit
                                         ; received
              ORL    SSCON,#10h          ; Transmit STOP
              CLR    b_TWI_busy          ; TWI is free
              JMP    end_switch

CASE_28:      CJNE   R7,#28h,case_30      ; DATA was transmitted, ACK bit received
              ORL    SSCON,#20h          ; send start
              JMP    end_switch

CASE_30:      CJNE   R7,#30h,case_38      ; DATA was transmitted, NOT ACK bit
                                         ; received
              ORL    SSCON,#10h          ; Transmit STOP
              CLR    b_TWI_busy          ; TWI is free
              JMP    end_switch

CASE_38:      CJNE   R7,#38h,case_40      ; Arbitration lost in SLA+W or DATA.
              ORL    SSCON,#10h          ; Transmit STOP
              CLR    b_TWI_busy          ; TWI is free
              JMP    end_switch

CASE_40:      CJNE   R7,#40h,case_58      ; As soon as, the DATA will move to
                                         ; SSDAT SFR.
              ANL    SSCON,#11110111B    ; clear SI
S40:          MOV    ACC,SSCON
              JNB    ACC.3,S40           ; JUMP IF SI = '0'
              MOV    TWI_DATA_I,SSDAT    ; Master will send ACK or NACK to slave

              ;================================
              ;ORL SSCON,#00000100B      ; set AA  */ SENT ACK
              ANL SSCON,#11111011B       ; CLR AA  */ SENT NACK  ( DEFAULT USED )
              ;================================
              LJMP   end_switch

CASE_58:      CJNE   R7,#58h,case_5X
              ORL    SSCON,#00010000B    ; send STOP
              MOV    STOP_BACKUP,SSCON   ; BACKUP
              CLR    b_TWI_busy          ; TWI is free
              JMP    end_switch

CASE_5X:
end_switch:   ANL    SSCON,#11110111B    ; clear SI flag
              RETI

;*****************************************************
;*                Send 1-Byte to SERIAL              *
;*                Input   : ACC                      *
;*                Output  : Serial port              *
;*****************************************************

TX_BYTE:     ;LCALL  DELAY                ; JUST HAVE
             CLR    TI
             MOV    SBUF,A
             JNB    TI,$
             CLR    TI
             RET
```

```
TX_CHAR:            MOV    SBUF,A              ; Send Data to SBUF
                    JNB    TI,$                ; Wait until TX already (TI=1)
                    CLR    TI                  ; Clear TI
                    AJMP   TX_LOOP             ; Jump to TX_LOOP

;*****************************************************
;             CONVERT HEX 1 DIGIT TO ASCII CODE
;             INPUT : A
;             OUTPUT : A
;*****************************************************
HEX2ASC1:     PUSH   ACC
              CLR    C
              SUBB   A,#0AH
              POP    ACC
              JC     L_CON
              ADD    A,#7
L_CON:        ADD    A,#30H
              RET

;*****************************************************
;             CONVERT HEX 2 DIGIT TO ASCII CODE
;             INPUT : HEX2IN
;             OUTPUT : ASC_L
;             OUTPUT : ASC_H
;*****************************************************
HEX2ASC2:     MOV    A,HEX2IN
              ANL    A,#0FH
              LCALL  HEX2ASC1
              MOV    ASC_L,A
              MOV    A,HEX2IN
              SWAP   A
              ANL    A,#0FH
              LCALL  HEX2ASC1
              MOV    ASC_H,A
              RET

;-------------------------------------------------------------------------------
;-------------------------------------------------------------------------------
;-------------------------------------------------------------------------------
SHOW1:        DFB    "TEST 24XX256 RANGE 00H - 0FFH",00AH,00DH,00AH,00DH,0FFH

;-------------------------------------------------------------------------------
SHOW_ENTER:         DFB    "ENTER",0FFH

;-------------------------------------------------------------------------------
ENTER:              DFB    0AH,0DH,0FFH

;-------------------------------------------------------------------------------
; ***********************************************
;             DELAY   100us
; ***********************************************
DELAY:        MOV    R7,#9
D1:           MOV    R6,#5
D2:           NOP
              NOP
              DJNZ   R6,D2
              DJNZ   R7,D1
              RET
```

```
; ************************************************
;             DELAY  1ms
; ************************************************
DELAYS:                 MOV    R7,#25
D1S:            MOV     R6,#20
D2S:            NOP
                NOP
                DJNZ    R6,D2S
                DJNZ    R7,D1S
                RET

DELAY_1S:       MOV     R7,#150
D1M:            MOV     R6,#50
D2M:            MOV     R5,#50
D3M:            NOP
                NOP
                DJNZ    R5,D3M
                DJNZ    R6,D2M
                DJNZ    R7,D1M
                RET
                END
```

คำอธิบาย

โปรแกรมนี้จะเหมือนกันกับโปรแกรมที่ผ่านมา, เพียงแต่ว่าโปรแกรม 24xx256_2_UART.ASM นี้นั้นจะส่งข้อมูลผ่านพอร์ตการสื่อสารอนุกรม UART ด้วยอัตรา Baud rate = 115200 ซึ่งหมายความว่าผู้อ่านจะต้องเปิดโปรแกรม Procomm ไว้ก่อน และ จะต้องกำหนดค่า Baud rate ของโปรแกรมนี้ให้ตรงกันด้วย, ซึ่งวิธีการติดตั้ง และ วิธีการใช้งานโปรแกรม Procomm นั้นผู้อ่านสามารถอ่านได้จากไฟล์ใน CD-ROM และ รายละเอียดเกี่ยวกับการใช้งาน UART ผู้อ่านสามารถศึกษาได้จากตัวอย่างการใช้งานพอร์ต UART ได้โดยตรง

Examples 5: [DS1307_RW.ASM] โปรแกรมนี้เป็นการใช้งานชิป DS1307 โดยภายในโปรแกรมจะมี 2 ส่วน คือ ส่วนการเขียนข้อมูลซึ่งจะเป็นการกำหนดให้หลักวินาทีเริ่มทำงานที่ 00 และ ส่วนการอ่านข้อมูลจากชิป DS1307 ซึ่งจะอ่านค่าวินาทีออกไปแสดงที่ พอร์ต P0 (ผู้อ่านจะต้องต่อ LED ที่พอร์ต P0 ด้วย common A ผ่าน R = 470 โอห์ม)

```
;*********************************************************************
; DESCRIBTION : This program is showed second part on port P0.
; FILE NAME   : DS1307_RW.ASM
; COMPILER    : CROSS 32
; SUPPORT     : AT89C5131
; DATE        : 26 August 2004
;*********************************************************************

       CPU    "8051.TBL"    ; Processor declaration
       HOF    "INT8"        ; Intel 8-bit hexcode
       INCL    "AT89C5131.SFR"

;*********************************************************************
;      DEFINE BYTE SECTION
;*********************************************************************
;
SLAVE_ADR     EQU    21H
ADR_H         EQU    22H
ADR_L         EQU    23H
TWI_DATA      EQU    24H
TWI_DATA_I    EQU    26H
STOP_BACKUP   EQU    27H
SEC           EQU    40H
SEC_R         EQU    41H
DIRECTION     EQU    33H    ; 01H = WRITING, 02H = READING
;*********************************************************************
;      DEFINE BIT SECTION
;*********************************************************************
RW            EQU    00H    ; 0=WRITE, 1=READ  ; BIT TYPE
B_TWI_BUSY    EQU    25H    ; BIT TYPE
;*********************************************************************

              ORG    0000H         ; Reset vector
START:        LJMP   BEGIN_W

;*************************************************
;             INTERRUPT SECTION
;*************************************************
              ORG    0043H
              LJMP   TWI_IT

;*************************************************
              ORG    0100H
;*************************************************
;             INITIALIZE DS 1307
;*************************************************
BEGIN_W:      ORL    CKCON0,#00000001B   ; 6 CLK PERIOD/MACHINE
              ORL    SSCON,#01000000B    ; enable TWI */
              ANL    SSCON,#01000100B    ;
              SETB   EA                  ; interrupt enable */
              ORL    IEN1,#02h           ; enable TWI interrupt */
              CLR    B_TWI_BUSY
              MOV    ADR_L,#00H
              MOV    SEC,#00H
              MOV    STOP_BACKUP,#00H
              MOV    DIRECTION,#01H
```

```
                ;MOV   R0,#00H
                MOV    P1,#0FH

LOOP_W:         MOV    ACC,STOP_BACKUP     ; FIND STOP BIT, IF '1' JUMP
                JB     ACC.4,CONT_W

                JB     B_TWI_BUSY,END_IF_W ; jump if b_TWI_busy bit = '1'*/

                MOV    ACC,SSCON
                JB     ACC.4,END_IF_W      ; jump if acc.4 bit = '1' "STOP FLAG"
                                           ; usually not jump*/
                SETB   B_TWI_BUSY          ; flag busy =1 , now, I'm not Empty.*/
                MOV    SLAVE_ADR,#11010000B; slave adresse example
                CLR    RW                  ; 0= WRITE
                MOV    SSDAT,#00h          ; clear buffer before sending data
                ORL    SSCON,#00100000B    ; TWI start sending */ after that SI bit
                                           ; is set '1'
                                   ; and  the status code in SSCS will be 08h.
                JMP    LOOP_W
; ************************************************
CONT_W:         MOV    P1,#55H
                LJMP   BEGIN_R
END_IF_W:       JMP    LOOP_W

;************************************************
;               INITIALIZE DS1307 READ
;************************************************
BEGIN_R:
;************************************************
;               INITIALIZE DS1307
;************************************************
                ORL    SSCON,#40h                 ; enable TWI */
                SETB   EA                         ; interrupt enable */
                ORL    IEN1,#02h                  ; enable TWI interrupt */
                CLR    B_TWI_BUSY
                MOV    TWI_DATA,#00H              ; data example to send
                MOV    DIRECTION,#02H
                MOV    STOP_BACKUP,#00H
                MOV    ADR_L,#00h
                MOV    P2,#03H

;================================================
;               READ DS1307
;================================================
LOOP:           MOV    ACC,STOP_BACKUP
                JB     ACC.4,CONT          ; FIND STOP BIT, IF '1' JUMP

                JB     B_TWI_BUSY,END_IF   ; jump if b_TWI_busy bit = '1'*/

                MOV    ACC,SSCON
                JB     ACC.4,END_IF        ; jump if acc.4 bit = '1' "STOP BIT"
                                           ; usually not jump*/
                SETB   B_TWI_BUSY          ; flag busy =1 , now, I'm not Empty.*/
                MOV    SLAVE_ADR,#11010000B; slave adresse example
                CLR    RW                  ; 0=write
                MOV    SSDAT,#00h          ; clear buffer before sending data

                ;=============
                ORL SSCON,#20h             ; TWI start sending */ after that SI bit
                                           ; is set '1'
                                   ; and  the status code in SSCS will be 08h.

                JMP    LOOP
```

```
;==============================================
;             DISPLAY TO LED
;==============================================
CONT:         MOV    STOP_BACKUP,#00H
              MOV    P0,SEC_R
              LCALL  DELAYS
              MOV    P2,#18H
              sjmp loop

END_IF:
              LJMP LOOP

; ***********************************************
;             INTERRUPT SERVICE ROUTINE
; ***********************************************
TWI_IT:       MOV    R7,SSCS

; ************* TWI status tasking ***************

CASE_00:      CJNE   R7,#00h,CASE_08      ; A start condition has been sent
                                          ; SLR+R/W are transmitted, ACK bit
                                          ; received
              CLR    B_TWI_BUSY           ; TWI is free
              ORL    SSCON,#10H           ; SEND STOP

              LJMP   end_switch

CASE_08:      CJNE   R7,#08h,CASE_10      ; A start condition has been sent
                                          ; SLR+R/W are transmitted, ACK bit
                                          ; received
              ANL    SSCON,#~20h          ; clear start condition

;             send slave adress and read/write bit

              MOV    ACC,slave_adr
              MOV    C,RW
              MOV    ACC.0,C
              MOV    SSDAT,ACC

              LJMP   end_switch

CASE_10:      CJNE   R7,#10h,CASE_18      ; A repeated start condition has been
                                          ; sent
                                          ; SLR+R/W are transmitted, ACK bit
                                          ; received
              ANL    SSCON,#~20h          ; clear start condition

;             send slave adress and read/write bit

              MOV    ACC,slave_adr
              ;=========
              SETB   RW
              ;=========
              MOV    C,RW
              MOV    ACC.0,C
              MOV    SSDAT,ACC
              JMP    end_switch

CASE_18:      CJNE   R7,#18h,case_20      ; SLR+W was transmitted, ACK bit
                                          ; received
              ; JJJJJJJJJJJJJJJJJJJJJJJJJJJ
              MOV    R1,DIRECTION
              CJNE   R1,#01H,RECEIVE_DIREC_18
              ; JJJJJJJJJJJJJJJJJJJJJJJJJJJ
```

```
                ANL     SSCON,#11110111B    ; clear SI
                MOV     SSDAT,ADR_L
CHECK_FIR:      MOV     ACC,SSCON
                JNB     ACC.3,CHECK_FIR     ; JUMP IF SI = '0'

                ANL     SSCON,#11110111B    ; clear SI
                MOV     SSDAT,SEC           ; Transmit data byte, ACK bit received
                JMP     end_switch

RECEIVE_DIREC_18:
                MOV     SSDAT,ADR_L
                LJMP    end_switch

CASE_20:        CJNE    R7,#20h,case_28     ; SLR+W was transmitted, NOT ACK bit
                                            ; received
                ORL     SSCON,#10h          ; Transmit STOP
                CLR     b_TWI_busy          ; TWI is free
                JMP     end_switch

CASE_28:        CJNE    R7,#28h,case_30     ; DATA was transmitted, ACK bit received

                ; JJJJJJJJJJJJJJJJJJJJJJJJJJ
                MOV     R1,DIRECTION
                CJNE    R1,#01H,RECEIVE_DIREC_28
                ; JJJJJJJJJJJJJJJJJJJJJJJJJJ

                ORL     SSCON,#10h          ; send STOP
                MOV     STOP_BACKUP,SSCON   ; BACKUP
                CLR     b_TWI_busy          ; TWI is free
                JMP     end_switch
RECEIVE_DIREC_28:
                ORL     SSCON,#20h          ; send start
                JMP     end_switch

CASE_30:        CJNE    R7,#30h,case_38     ; DATA was transmitted, NOT ACK bit
                                            ; received
                ORL     SSCON,#10h          ; Transmit STOP
                CLR     b_TWI_busy          ; TWI is free
                JMP     end_switch

CASE_38:        CJNE    R7,#38h,case_40     ; Arbitration lost in SLA+W or DATA.
                ORL     SSCON,#10h          ; Transmit STOP
                CLR     b_TWI_busy          ; TWI is free
                JMP     end_switch

CASE_40:        CJNE    R7,#40h,case_58     ; As soon as, the DATA will move to
                                            ; SSDAT SFR.
                ANL     SSCON,#11110111B    ; clear SI
S40:            MOV     ACC,SSCON
                JNB     ACC.3,S40           ; JUMP IF SI = '0'
                MOV     SEC_R,SSDAT   ; Master will send ACK or NACK to slave

                ;================================
                ;ORL SSCON,#00000100B       ; set AA  */ SENT ACK
                ANL SSCON,#11111011B        ; CLR AA  */ SENT NACK  ( DEFAULT USED )
                ;================================

                LJMP    end_switch
```

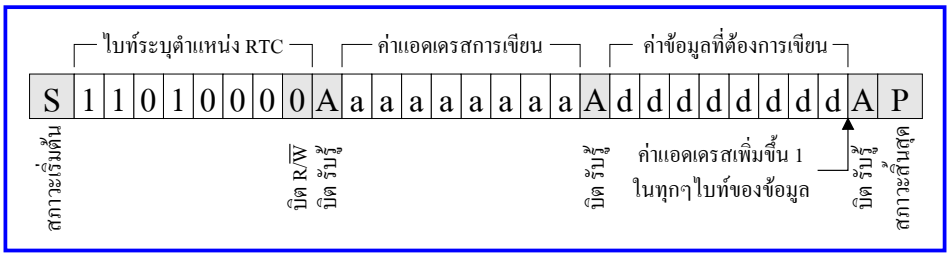
```
CASE_58:      CJNE   R7,#58h,case_5X
              ORL    SSCON,#00010000B    ; send STOP
              MOV    STOP_BACKUP,SSCON   ; BACKUP
              CLR    b_TWI_busy          ; TWI is free
              JMP    end_switch

CASE_5X:
end_switch:   ANL    SSCON,#11110111B    ; clear SI flag
              RETI

; ************************************************
;             DELAY
; ************************************************
DELAY:        MOV    R7,#9
D1:           MOV    R6,#5
D2:           NOP
              NOP
              DJNZ   R6,D2
              DJNZ   R7,D1
              RET

; ************************************************
;             DELAY
; ************************************************
DELAYS:              MOV    R7,#25
D1S:          MOV    R6,#20
D2S:          NOP
              NOP
              DJNZ   R6,D2S
              DJNZ   R7,D1S
              RET
              END
```

คำอธิบาย

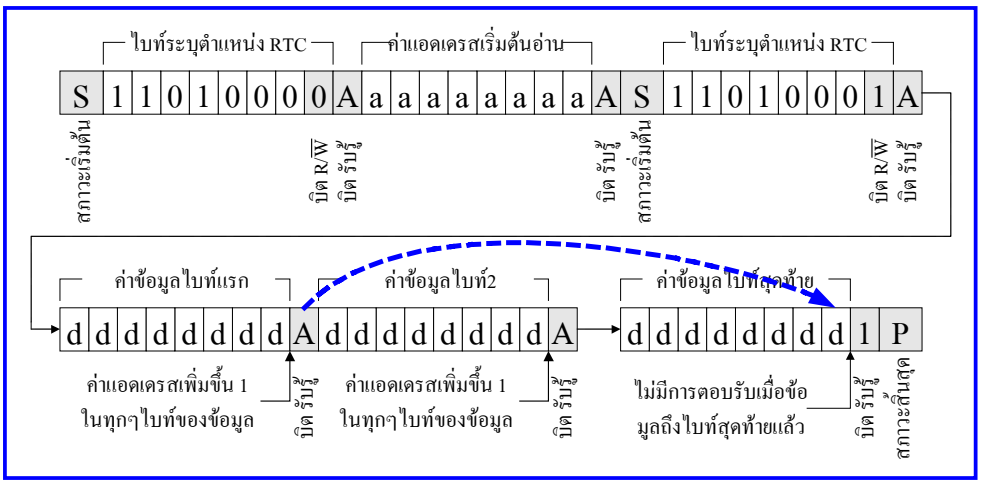
จากโปรแกรมข้างบนซึ่งเป็นโปรแกรมที่รวมเอาส่วนของการรับ และ การส่ง เข้าไว้ด้วยกัน โดยถ้าผู้อ่านดูโปรแกรมไม่เข้าใจ ผู้อ่านสามารถดูโปรแกรมที่แยกการรับ และ การส่งออกจากกัน ได้จากไฟล์โปรแกรมที่อยู่ใน CD

ภายในโปรแกรมทางด้านบน, ซึ่งการเขียน และการอ่านข้อมูลจากชิป DS1307 นั้นจะเป็นแบบทีละไบต์ ซึ่งรูปแบบการรับ และ ส่งแสดงดังภาพด้านล่าง คือ

การตรวจสอบโปรแกรมจะมีพอร์ต P3 ที่ใช้สำหรับการตรวจสอบโปรแกรมโดยถ้ามีค่า 18H ออกทางพอร์ต P3 แสดงว่าโปรแกรมนั้นทำงานถูกต้อง ไม่ติด Loop ใด Loop หนึ่งในโปรแกรม ซึ่งค่าวินาทีควรจะแสดงออกมาอย่างต่อเนื่องที่พอร์ต P0



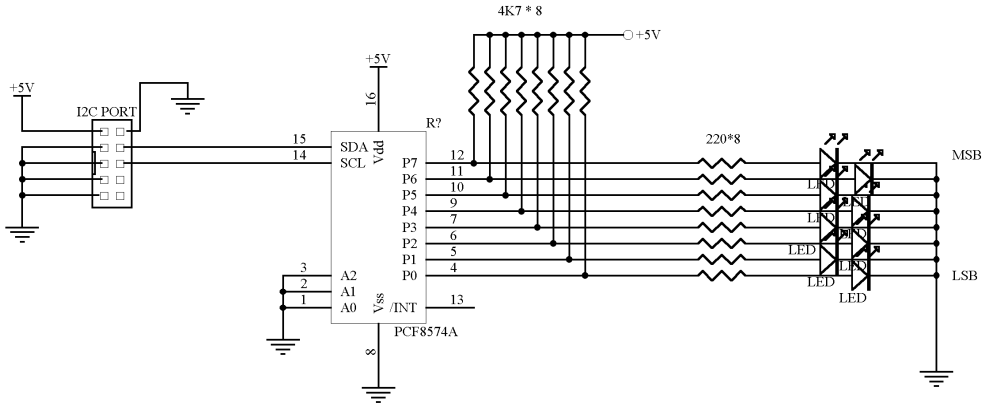
ลักษณะลำดับขั้นในการเขียนข้อมูลให้ DS1307 แบบไบท์เดียว



แสดง ลำดับขั้นตอนในการอ่านข้อมูลจาก RTC เบอร์ DS1307

รูปด้านบนในส่วนของการอ่านค่าข้อมูล จาก RTC ซึ่งจะอยู่ในรูปของการอ่านค่าแบบหลายไบต์ แต่ ในที่นี้ผู้เขียนจะรับข้อมูลครั้ง ละ 1 ไบต์ ดังนั้น เมื่อรับข้อมูล 1ไบต์แล้วจะกระโดดไปส่งบิต NACK ในส่วนท้ายทันที

Examples 6: [TWI_PCF8574A_CONST.ASM] โปรแกรมนี้เป็นการสื่อสารอนุกรมบนบัส I^2C ด้วยพอร์ต I2C BUS โดยโปรแกรมจะเป็นการเขียนข้อมูลให้แก่ชิป PCF8974A ด้วยค่าคงที่ 0FH (ผู้อ่านจะต้องต่อ Hardware ดังแสดงดังภาพด้านล่าง)




```
;*******************************************************************
; DESCRIBTION :     THIS PROGRAM IS ROTATE RIGHT OF PCF8574
; FILE NAME   : TWI_PCF8574A.ASM
; COMPILER    : CROSS 32
; SUPPORT     : AT89C5131
; DATE        : 7 August 2004
;*******************************************************************

       CPU    "8051.TBL"    ; Processor declaration
       HOF    "INT8"        ; Intel 8-bit hexcode
       INCL    "AT89C5131.SFR"

;*******************************************************************
;      DEFINE BYTE SECTION
;*******************************************************************
;
SLAVE_ADR     EQU    21H
ADR_H         EQU    22H
ADR_L         EQU    23H
B_TWI_BUSY    EQU    25H    ; BIT TYPE
TWI_DATA_I    EQU    26H
STOP_BACKUP   EQU    27H
TWI_DATA      EQU    28H
;*******************************************************************
;      DEFINE BIT SECTION
;*******************************************************************
RW            EQU    00H    ; 0=WRITE, 1=READ  ; BIT TYPE
;*******************************************************************

              ORG    0000H        ; Reset vector
START:        LJMP   BEGIN
```

```
;*************************************************
;             INTERRUPT SECTION
;*************************************************
             ORG    0043H
             LJMP   TWI_IT

;*************************************************
             ORG    0100H
;*************************************************
;            INITIALIZE PCF8574A
;*************************************************
BEGIN:       ORL    CKCON0,#00000001B   ; 6 CLK PERIOD/MACHINE
             ORL    SSCON,#01000000B    ; enable TWI */
             ANL    SSCON,#01000100B    ;
             SETB   EA                  ; interrupt enable */
             ORL    IEN1,#02h           ; enable TWI interrupt */
             CLR    B_TWI_BUSY
             MOV    TWI_DATA,#0FH       ; data example to send */**
             MOV    ADR_H,#00H
             MOV    ADR_L,#00H
             MOV    STOP_BACKUP,#00H
             MOV    R0,#00H
             MOV    P0,#0FH

LOOP_W:             MOV    ACC,STOP_BACKUP ; FIND STOP BIT, IF '1' JUMP
             JB     ACC.4,CONT_W

             JB     B_TWI_BUSY,END_IF_W ; jump if b_TWI_busy bit = '1'*/
             MOV    ACC,SSCON
             JB     ACC.4,END_IF_W      ; jump if acc.4 bit = '1' "STOP FLAG"
                                        ; usually not jump*/
             SETB   B_TWI_BUSY          ; flag busy =1 , now, I'm not Empty.*/
             MOV    SLAVE_ADR,#01110000B; slave adresse example
             CLR    RW                  ; 0= WRITE
             MOV    SSDAT,#00h          ; clear buffer before sending data
             ORL    SSCON,#00100000B    ; TWI start sending */ after that SI bit
                                        ; is set '1'
                                        ; and  the status code in SSCS will be
                                        ; 08h.

             JMP    LOOP_W
; ************************************************
CONT_W:      MOV    STOP_BACKUP,#00H
             MOV    P0,#55H
             SJMP   $
END_IF_W:    JMP    LOOP_W

; ************************************************
;            INTERRUPT SERVICE ROUTINE
; ************************************************
TWI_IT:             MOV    R7,SSCS

; ************* TWI status tasking ***************

CASE_00:     CJNE   R7,#00h,CASE_08     ; A start condition has been sent
                                        ; SLR+R/W are transmitted, ACK bit
                                        ; received
             CLR    B_TWI_BUSY          ; TWI is free
             ORL    SSCON,#10H          ; SEND STOP
             LJMP   end_switch
```

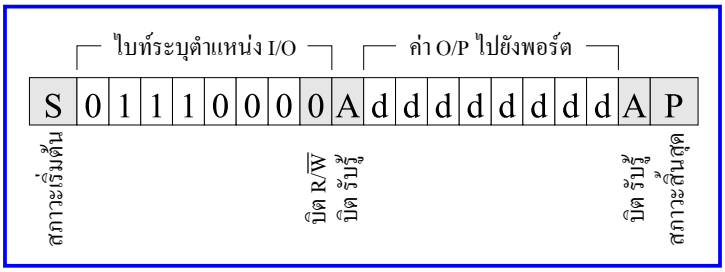
```
CASE_08:      CJNE   R7,#08h,CASE_10      ; A start condition has been sent
                                         ; SLR+R/W are transmitted, ACK bit
                                         ; received
              ANL    SSCON,#~20h         ; clear start condition

;             send slave adress and read/write bit

              MOV    ACC,slave_adr
              MOV    C,RW
              MOV    ACC.0,C
              MOV    SSDAT,ACC
              LJMP   end_switch

CASE_10:      CJNE   R7,#10h,CASE_18      ; A repeated start condition has been
                                         ; sent
                                         ; SLR+R/W are transmitted, ACK bit
                                         ; received
              ANL    SSCON,#~20h         ; clear start condition

;             send slave adress and read/write bit

              MOV    ACC,slave_adr
              ;=========
              SETB   RW
              ;=========
              MOV    C,RW
              MOV    ACC.0,C
              MOV    SSDAT,ACC
              JMP    end_switch

CASE_18:      CJNE   R7,#18h,case_20      ; SLR+W was transmitted, ACK bit
                                         ; received

              ANL    SSCON,#11110111B    ; clear SI
              MOV    SSDAT,TWI_data      ; Transmit data byte, ACK bit received
              JMP    end_switch

CASE_20:      CJNE   R7,#20h,case_28      ; SLR+W was transmitted, NOT ACK bit
                                         ; received
              ORL    SSCON,#10h          ; Transmit STOP
              CLR    b_TWI_busy          ; TWI is free
              JMP    end_switch

CASE_28:      CJNE   R7,#28h,case_30      ; DATA was transmitted, ACK bit received
              ;ANL   SSCON,#11110111B    ; clear SI
              ORL    SSCON,#10h          ; send STOP
              MOV    STOP_BACKUP,SSCON   ; BACKUP
              CLR    b_TWI_busy          ; TWI is free
              JMP    end_switch

CASE_30:      CJNE   R7,#30h,case_38      ; DATA was transmitted, NOT ACK bit
                                         ; received
              ORL    SSCON,#10h          ; Transmit STOP
              CLR    b_TWI_busy          ; TWI is free
              JMP    end_switch

CASE_38:      CJNE   R7,#38h,case_40      ; Arbitration lost in SLA+W or DATA.
              ORL    SSCON,#10h          ; Transmit STOP
              CLR    b_TWI_busy          ; TWI is free
              JMP    end_switch
```

```
CASE_40:      CJNE   R7,#40h,case_58      ; As soon as, the DATA will move to
                                          ; SSDAT SFR.
              ANL    SSCON,#11110111B     ; clear SI
S40:          MOV    ACC,SSCON
              JNB    ACC.3,S40            ; JUMP IF SI = '0'
              MOV    TWI_DATA_I,SSDAT     ; Master will send ACK or NACK to slave

              ;================================
              ;ORL SSCON,#00000100B             ; set AA  */ SENT ACK
              ANL SSCON,#11111011B        ; CLR AA  */ SENT NACK  ( DEFAULT USED )
              ;================================
              LJMP   end_switch

CASE_58:      CJNE   R7,#58h,case_5X
              ORL    SSCON,#00010000B     ; send STOP
              MOV    STOP_BACKUP,SSCON    ; BACKUP
              CLR    b_TWI_busy           ; TWI is free
              JMP    end_switch

CASE_5X:
end_switch:   ANL    SSCON,#11110111B     ; clear SI flag
              RETI

; ************************************************
;             DELAY
; ************************************************
DELAY:        MOV    R7,#9
D1:           MOV    R6,#5
D2:           NOP
              NOP
              DJNZ   R6,D2
              DJNZ   R7,D1
              RET

; ************************************************
;             DELAY
; ************************************************
DELAYS:              MOV    R7,#25
D1S:          MOV    R6,#20
D2S:          NOP
              NOP
              DJNZ   R6,D2S
              DJNZ   R7,D1S
              RET
              END
```

คำอธิบาย

จากโปรแกรมด้านบนซึ่งเป็นการเขียนข้อมูลออกไปที่ตัวชิป PCF8574A เพียงอย่างเดียว ซึ่งผู้อ่านจะต้องต่อ Hardware เข้ากับพอร์ต I2C BUS ของบอร์ด CP-JR51USB V1.0 ด้วยโปรแกรมจึงจะทำงานอย่างสมบูรณ์

โปรแกรมนี้ถือเป็นโปรแกรมที่ง่ายที่สุดก็ว่าได้เกี่ยวกับการใช้งาน TWI เนื่องจากว่าหลังจากส่ง Control Byte ให้กับชิป PCF8974A แล้ว ผู้อ่านจะสามารถส่งข้อมูลออกไปแสดงได้เลย ซึ่งมีรูปแบบการส่งข้อมูลดังนี้ คือ



รูปแสดง การเขียนค่า Output ไปยัง PCF8574A

สิ่งที่ต้องระวังอีกประการหนึ่งก็ คือ รหัส Control Byte ของ PCF8574 กับ PCF8574A จะไม่เหมือนกันผู้อ่านต้องดูเบอร์ที่ใช้งานให้ดี

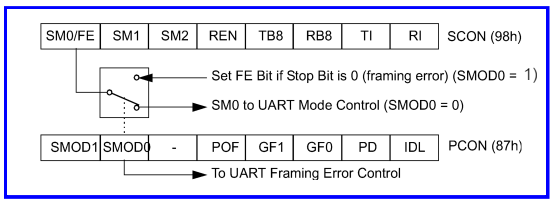
Tip : เนื้อหาเกี่ยวกับ I^2C ผู้อ่านสามารถดูได้จากหนังสือ " รู้จักและเข้าใจ I^2C BUS " ของบริษัท ETT

Serial I/O Port (UART)

การสื่อสารข้อมูลด้วย Serial I/O port ในชิป AT89C5131 นั้นจะมีความเหมือนกันกับของชิป 80C52 คือ มีทั้งโหมดการสื่อสารอนุกรมแบบ Synchronous และ Asynchronous ซึ่งในการสื่อสารแบบ UART นั้นจะมีการทำงาน 3 โหมด ในแบบ Full-duplex ซึ่งหมายความว่าผู้อ่านสามารถส่ง และ รับ ข้อมูลได้พร้อมกันโดยที่การส่ง และ การรับ อาจจะใช้ค่า Baud rate ที่แตกต่างกันได้ นอกจากนั้นสำหรับชิป AT89C5131 ยังได้เพิ่ม **Framing error detection** ซึ่งทำหน้าที่ ตรวจสอบความผิดพลาดในการรับส่งข้อมูล และ **Automatic address recognition** ที่ใช้ในกรณีที่ผู้อ่านต้องการใช้งานแบบ Multiprocessor communication

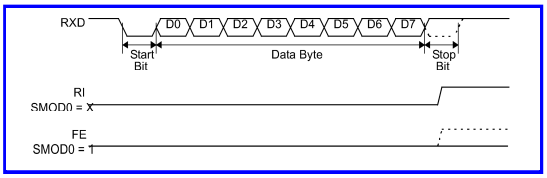
Framing Error Detection

Framing bit error detection สามารถใช้งานได้กับการสื่อสารแบบ Asynchronous ในโหมด 1 , 2 , และ 3 ในการเปิดการใช้งานคุณสมบัตินี้สามารถทำได้โดยเซ็ตบิต SMOD0 ในรีจิสเตอร์ PCON ดังแสดงในรูปด้านล่าง คือ



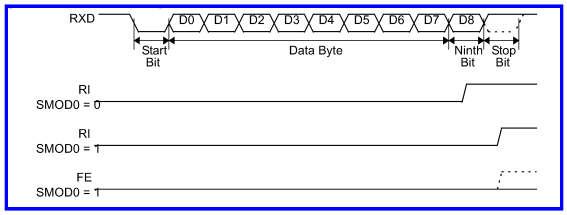
แสดง Framing Error Block Diagram

จากรูปด้านบน เมื่อคุณสมบัตินี้ทำงาน, คือ ผู้อ่านทำการเซ็ตบิต SMOD0 ให้เป็น ‘1’ ซึ่งผลจะทำให้บิตที่ 7 ของรีจิสเตอร์ SCON กลายเป็นบิต FE ซึ่งจะเป็น ‘1’ เมื่อตัวรับข้อมูลทำการเซ็กข้อมูลที่เข้ามาในแต่ละเฟรมแล้วมีความผิดพลาดของบิต STOP (Invalid stop bit) ซึ่งสาเหตุอาจจะมาจากสัญญาณ Noise ในสายสัญญาณ หรือ มีการส่งข้อมูลพร้อมกันระหว่าง 2 CPU ซึ่งส่งผลให้ข้อมูลชนกัน และ บิต FE จะเป็น ‘0’ เมื่อการตรวจสอบบิต STOP ของการสื่อสารอนุกรมนั้นถูกต้อง (Valid stop bit) ดังนั้น ในการเขียนโปรแกรมของผู้ใช้งานก็ควรจะตรวจสอบบิต FE หลังการตรวจสอบ Error ในแต่ละเฟรมแล้ว ซึ่งถ้าผลปรากฏว่าบิต FE = ‘1’ , ผู้อ่านจะต้อง Clear บิตนี้ด้วยการเขียนโปรแกรมของผู้อ่านเอง หรือ รีเซ็ต CPU เท่านั้น แต่ถ้าการตรวจสอบเฟรมข้อมูลที่รับมานั้นถูกต้อง บิต FE = ‘0’ ผู้อ่านก็ไม่ต้องเคลีย์บิต FE แต่อย่างใด รูปด้านล่างแสดงตัวอย่างการรับข้อมูลด้วย UART โหมด 1



รูปแสดงการรับข้อมูล UART โหมด 1 (ข้อมูล 8 บิต)

จากรูปด้านบนจะเห็นว่าบิต RI จะเป็น ‘1’ ในบิตที่ 8 ซึ่งถ้าตำแหน่งนี้ไม่มี Stop บิต ซึ่งหมายความว่าตรวจพบความผิดพลาดในการรับข้อมูล, ผลจะทำให้บิต FE เป็น ‘1’ ดังแสดงด้วยรอยเส้นประ



รูปแสดงการรับข้อมูล UART โหมด 2, 3 (ข้อมูล 9 บิต)

Automatic Address Recognition

การเปิดใช้งานคุณสมบัติ Automatic Address Recognition นี้ก็ต่อเมื่อผู้อ่านต้องการจะใช้งานการสื่อสารแบบ Multiprocessor Communication (บิต SM2 ในรีจิสเตอร์ SCON = ‘1’) ซึ่งรายละเอียดการใช้งานใน

ส่วนนี้ผู้เขียนของไม่กล่าวในที่นี้ โดยผู้อ่านสามารถหาอ่านได้จาก Datasheet หน้า 66

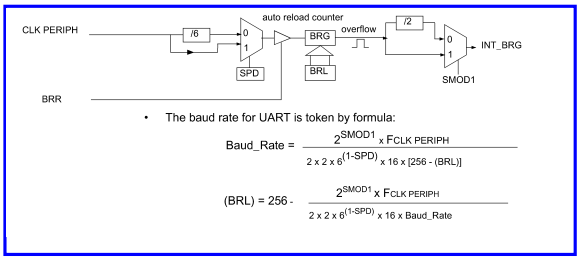
Baud Rate Selection for UART for Mode 1 and 3

ในการกำหนดค่า Baud rate ในการรับส่งข้อมูลนั้นสัญญาณ Clocks สามารถเลือกได้จากหลายแหล่งด้วยกัน ซึ่งกำหนดได้จากรีจิสเตอร์ T2CON และ BDRCON โดยสามารถแบ่งใหญ่ๆ ได้ 2 ทาง ได้แก่

- Timer
- Internal Baud rate Generator (BRG)

โดยทางแรก คือ การใช้ Timer, ซึ่งวิธีนี้เป็นวิธีที่ผู้อ่านคุ้นเคยเป็นอย่างดีอยู่แล้วซึ่ง CPU มาตรฐาน 8051 จะต้องใช้ในการสร้าง Baud rate แต่วิธีนี้ผู้อ่านจะต้องเลือก Crystal ที่หาค่า Baud rate ได้ลงตัว ซึ่งได้แก่ ค่า 11.0592 MHz หรือ 18.0432MHz เป็นต้น มิฉะนั้นจะไม่สามารถสร้าง Baud rate ที่เป็นค่ามาตรฐานได้

ทางที่สอง คือ การใช้งาน Internal Baud Rate Generator ซึ่งวิธีนี้ผู้อ่านจะสามารถสร้างค่า Baud rate มาตรฐานได้ โดยที่ใช้ Crystal ค่าอื่นๆ นอกเหนือจากวิธีแรกได้ เช่น 16.384MHz หรือ 24MHz เป็นต้น ซึ่งบอร์ด CP-JR51USB v1.0 นี้ใช้ค่า Crystal = 24MHz ดังนั้น ตัวอย่างต่างๆ ที่เกี่ยวกับการใช้งานพอร์ต UART ผู้เขียนจะใช้วิธีการนี้ทั้งหมด (สูตรคำนวณ Baud rate ดูในรูปด้านล่าง)



แสดง โครงสร้างของ Internal Baud Rate Generator

และ เพื่อความสะดวกในการใช้งาน ตารางด้านล่างเป็นการสรุปค่า Baud rate ที่ Crystal = 16.384MHz และ 24MHz

Example of computed value when X2 = 1, SMOD1 = 1, SPD = 1

Baud Rates	F_{OSCA} = 16.384 MHz		F_{OSCA} = 24 MHz	
	BRL	Error (%)	BRL	Error (%)
115200	247	1.23	243	0.16
57600	238	1.23	230	0.16
38400	229	1.23	217	0.16
28800	220	1.23	204	0.16
19200	203	0.63	178	0.16
9600	149	0.31	100	0.16
4800	43	1.23	-	-

Example of computed value when X2 = 0, SMOD1 = 0, SPD = 0

Baud Rates	F_{OSCA} = 16.384 MHz		F_{OSCA} = 24 MHz	
	BRL	Error (%)	BRL	Error (%)
4800	247	1.23	243	0.16
2400	238	1.23	230	0.16
1200	220	1.23	202	3.55
600	185	0.16	152	0.16

The baud rate generator can be used for mode 1 or 3 (refer to Figure 34.), but also for mode 0 for UART, thanks to the bit SRC located in BDRCON register (Table 60.)

Examples 7: [UART MODE1_INT_BRG.ASM] โปรแกรมนี้เป็นการใช้งาน UART ที่ Baud rate =115200bps โดยใช้โหมด 1 และ ไม่ใช้ Interrupt , ในการรันโปรแกรมนี้ผู้อ่านควรเปิดโปรแกรม Procomm 3 หรือ Hyper Terminal ก่อนรันโปรแกรมนี้ และ เซ็ตค่า Baud rate ให้ตรงกัน, เมื่อรันโปรแกรมผู้อ่านจะเห็นข้อความขึ้นบนหน้าต่าง Procomm จากนั้นให้ผู้อ่านพิมพ์ตัวอักษรอะไรก็ได้แล้วกดปุ่ม Enter ผู้อ่านจะเห็นข้อความที่พิมพ์ปรากฏ

```
;*********************************************************************
; DESCRIBTION :-
; FILE NAME    : UART MODE1_INT_BRG.ASM
; COMPILER     : CROSS 32
; SUPPORT      : AT89C5131
; X'TAL        : 24MHz
; BAUD RATE    : 115200bps
; DATE         : 6 SEPTEMBER 2004
; MODE         : UART MODE1 INTERNAL BAUD RATE GENERATOR,NO USED INTERRUPT
;*********************************************************************

        CPU     "8051.TBL"                      ; Processor declaration
        HOF     "INT8"                          ; Intel 8-bit hexcode
        INCL    "AT89C5131.SFR"

;*********************************************************************
;       DEFINE SECTION
;*********************************************************************
BUFFER_RX       EQU     70H
COUNT_RX        EQU     6FH
;*********************************************************************

                ORG     0000H                   ; Reset vector
START:          LJMP    BEGIN

;*************************************************
;               INTERRUPT SECTION
;*************************************************

;*************************************************
                ORG     0100H

BEGIN:          MOV      SP,#256-32             ; define stack = 32 byte
                ORL     CKCON0,#00000001B       ; 6 CLK PERIOD/MACHINE
                ORL     PCON,#11000000B         ; ENABLE THE FRAMING BIT ERROR DETECTION,
                                                ; DOUBLE BAUD RATE
                                                ; AND CHECK FE BIT OF SCON SFR
                ORL     BDRCON,#00001110B       ; SPD=1,RBCK =1, TBCK =1
                MOV     SCON,#01010000B         ; uart in mode 1 (8 bit), REN=1
                MOV     BRL,#243                ; U CAN SEE THIS VALUE FROM PAGE 71 OF
                                                ; DATASHEET
                ORL     BDRCON,#00010000B       ; BRR = 1 , START BAUD RATE RUN
                ;=========================
                ; TRANSMITION
                ;=========================
                MOV     DPTR,#TEXT_1
                ACALL   TRANSMIT

                ;=========================
                ; RECEIVER
                ;=========================
                MOV     R0,#BUFFER_RX
                ACALL   RECEIVE

                MOV     A,R0
                MOV     R0,#BUFFER_RX
                ;=========================
```

```
                ;=========================
                ; TRANSMITION
                ;=========================
REPLY:          CLR     TI
                MOV     SBUF,@R0
                JNB     TI,$
                CLR     TI
                INC     R0
                CJNE    A,00H,REPLY             ; 00H = RO
                ;=========================
                ; SEND ENTER
                ;=========================
                MOV     DPTR,#ENTER
                ACALL   TRANSMIT
                JMP     BEGIN

;********************** ROUTINE: TX ****************************************
;                       INPUT : DPTR            ; DEFINE TEXT LOCATION
;                       OUTPUT : -
;*******************************************************************************
TRANSMIT:       CLR     TI                      ; Clear TI
TX_LOOP:        CLR     A                       ; Clear ACC.
                MOVC    A,@A+DPTR               ; Get Data from ROM with Pointer
                INC     DPTR                    ; Increase Pointer
                CJNE    A,#0FFH,TX_CHAR         ; Check 0FFH End of Text Char.
                RET                             ; End => Return

TX_CHAR:        MOV     SBUF,A                  ; Send Data to SBUF
                JNB     TI,$                    ; Wait until TX already (TI=1)
                CLR     TI                      ; Clear TI
                AJMP    TX_LOOP                 ; Jump to TX_LOOP

;********************** ROUTINE: RX ****************************************
;                       INPUT : R0              ; DEFINE BUFFER ADDRESS
;                       OUTPUT : ADDRESS 70H....
;*******************************************************************************

RECEIVE:        SETB    REN                     ; Set RX Enable
RE_LOOP:        JNB     RI,$                    ; Wait received 1 byte FROM KEY BOARD[ JUMP IF
                                                ; BIT RI = '0']
                CLR     RI                      ; CLEAR RECEIVE INTERRUPT FLAG
;               ****************************
;               FRAMEING BIT ERROR DETECTION
;               ****************************
                MOV     ACC,SCON
                CLR     SCON.7
                JB      ACC.7,RE_LOOP           ; JUMP IF SCON.7(FE) = '1'
;               ****************************
                MOV     A,SBUF                  ; Get byte from SBUF
                MOV     @R0,A                   ; KEEP 1 BYTE DATA TO MEMORY 70H
                CJNE    A,#0DH,INC_R0           ; Check Enter key receive? [IF REG "A" UNEQUAL
                                                ; 0DH JUMP]
                RET
;               ************************
;               COUNT THE AMOUNT WORDS
;               ************************
INC_R0:         INC     R0                      ; Increase pointer
                AJMP    RE_LOOP                 ;

; ************************************************
;               SEND ENTER
; ************************************************
ENTER:                  DFB     0AH,0DH,0FFH

; ************************************************
;               DELAY
; ************************************************
DELAY:          MOV     R7,#50
D1:             MOV     R6,#50
D2:             NOP
                NOP
                DJNZ    R6,D2
                DJNZ    R7,D1
                RET
```

```
; ************************************************
;               MESSAGE
; ************************************************
TEXT_1:         DFB 0AH,0DH
                DFB "Please,Press any key.<15 Charactor are limited>",0AH,0DH,0FFH
                END
```

คำอธิบาย

จากโปรแกรมด้านบนผู้อ่านจะเห็นว่า ในส่วนต้นของโปรแกรมกำหนดให้การทำงานเป็นแบบ 6 clk/Machine cycle และ กำหนดการทำงานของ UART เป็นโหมด 1 (8 bit) และ กำหนดค่าในรีจิสเตอร์ BRL = 243 ซึ่งเป็นการกำหนดค่า Baud rate ให้มีค่าเท่ากัน 115200 bps จากนั้นสั่งรัน ด้วยการเซ็ตบิต BRR = '1' แล้วส่งข้อความ TEXT_1 ซึ่งมีข้อความว่า " Please, Press any key.<15 Charactor are limited> " จากนั้นให้ผู้อ่านพิมพ์ตัวอักษรอะไรก็ได้ ไม่เกิน 15 ตัวอักษร จากนั้นกดปุ่ม Enter , ข้อมูลที่พิมพ์จะปรากฏบนจอภาพ ซึ่ง ในส่วนของการรับจะมีรีจิสเตอร์ R0 เป็นตัวชี้ตำแหน่งการเก็บค่าตัวอักษร

ภาคผนวก ก.

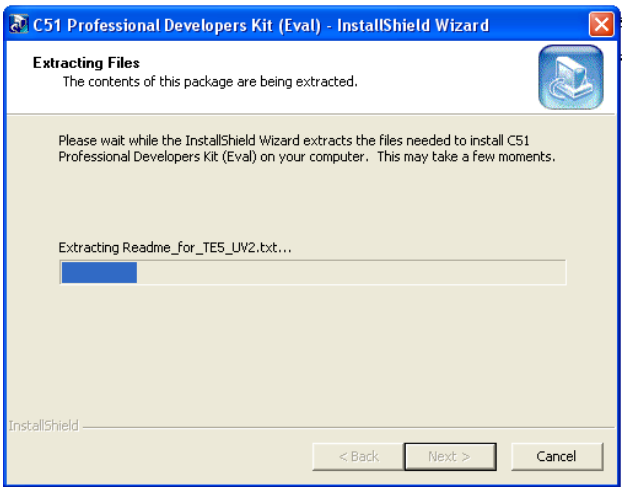
การติดตั้ง และ การใช้งานภาษา C (Keil) เบื้องต้น

เนื้อหาภายในภาคผนวก ก. นี้อธิบายเกี่ยวกับการติดตั้ง และ การใช้งาน Keil Compiler เบื้องต้น ซึ่ง Keil Compiler นี้สามารถใช้งานได้กับ Windows95, Windows98, WindowsNT, Windows2000, Windows XP

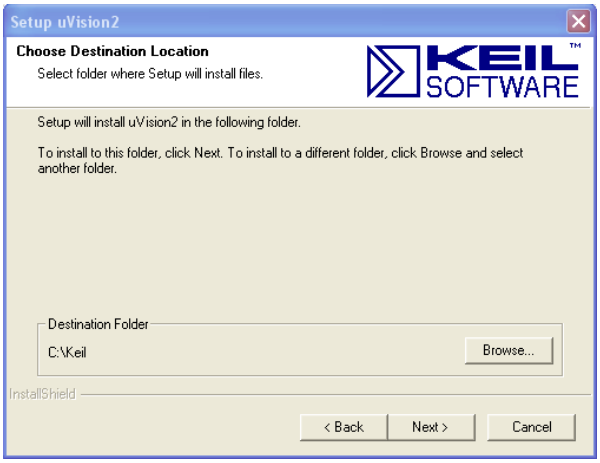
โปรแกรม Keil ที่ผู้เขียนจะนำเสนอนี้เป็นเวอร์ชั่นทดลองใช้ 7.10 ซึ่งเวอร์ชั่นนี้ผู้อ่านสามารถดาวน์โหลดได้จาก www.keil.com ซึ่งขั้นตอนการติดตั้งทำได้ดังนี้ คือ

ขั้นที่ 1 : เปิด CD-ROM และ Double-click ที่ไฟล์ **EK51V710.exe** ในโฟลเดอร์ Tools

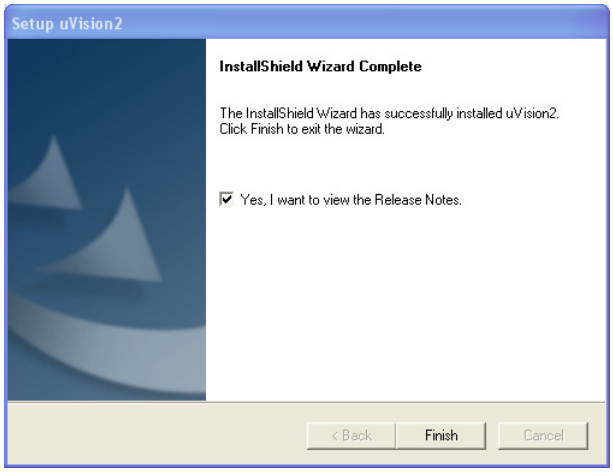
ขั้นที่ 2 : กดปุ่ม Yes เพื่อทำขั้นตอนการติดตั้งโปรแกรม หลังจากนั้นกดปุ่ม Next และ Yes ไปเลื่อยๆ



ขั้นที่ 3 : เมื่อผู้อ่านทำตามขั้นตอนจนมาถึงไดอะล็อก “ Choose Destination Location” ดังแสดงในรูปด้านล่าง ผู้อ่านสามารถเปลี่ยนแปลงตำแหน่งที่จะติดตั้งโปรแกรมได้ แต่ในที่นี้ผู้เขียนไม่เปลี่ยนแปลงใดๆ



ขั้นที่ 4 : จากนั้น กดปุ่ม Next ไปเลื่อยๆ จนสิ้นสุดการติดตั้งโปรแกรมโดยกดปุ่ม Finish เป็นอันเสร็จพิธี



การใช้งาน Keil เบื้องต้น

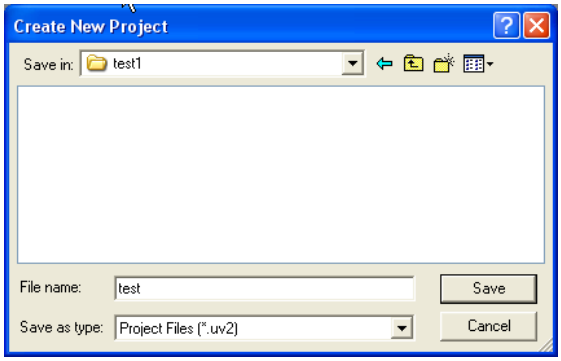
ในหัวข้อนี้ผู้เขียนจะนำผู้อ่านเข้าสู่การเขียนโปรแกรมด้วยภาษา C โดยใช้เครื่องมือชื่อ *µVision2* ซึ่งเวอร์ชั่นนี้จะเป็นเวอร์ชั่นทดลองใช้ หรือ เป็นเวอร์ชั่นที่ลิมิตขนาดของโค๊ดโปรแกรมที่จะใช้ Compile ในโปรแกรม *µVision2* นี้

การสร้างโปรเจ็คไฟล์ (Creating Projects)

ภายใน *µVision2* จะมีส่วนของ Project manager ซึ่งจะทำให้การออกแบบ หรือ การพัฒนาโปรแกรม กับ CPU ตระกูล 8051 ของผู้ใช้นั้นทำได้ง่าย และ สะดวกมากยิ่งขึ้น ซึ่งการสร้างโปรเจ็คไฟล์ทำได้ด้วยขั้นตอนต่อไปนี้ คือ

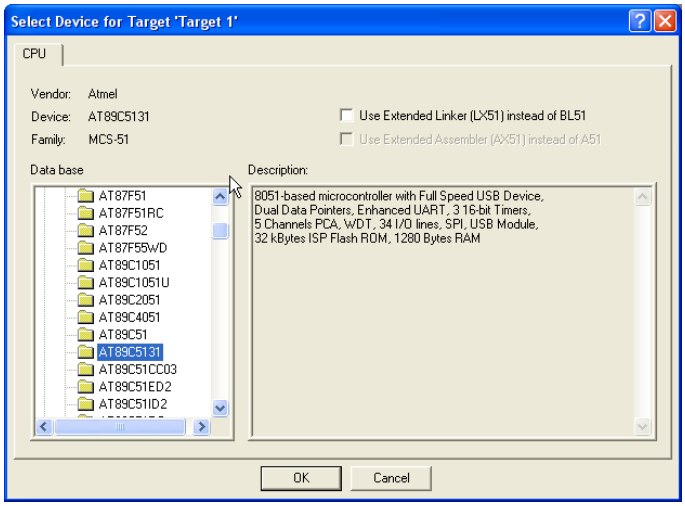
ขั้นที่ 1. เปิดโปรแกรม *µVision2* จาก Icon บน Desktop

ขั้นที่ 2. ทำการสร้างโปรเจ็คไฟล์ใหม่ โดยเลือก Project => New Project... จากเมนู ซึ่งหน้าต่างที่ขึ้นมาจะให้ผู้อ่านตั้งชื่อ New Project File ดังแสดงในรูปด้านล่าง



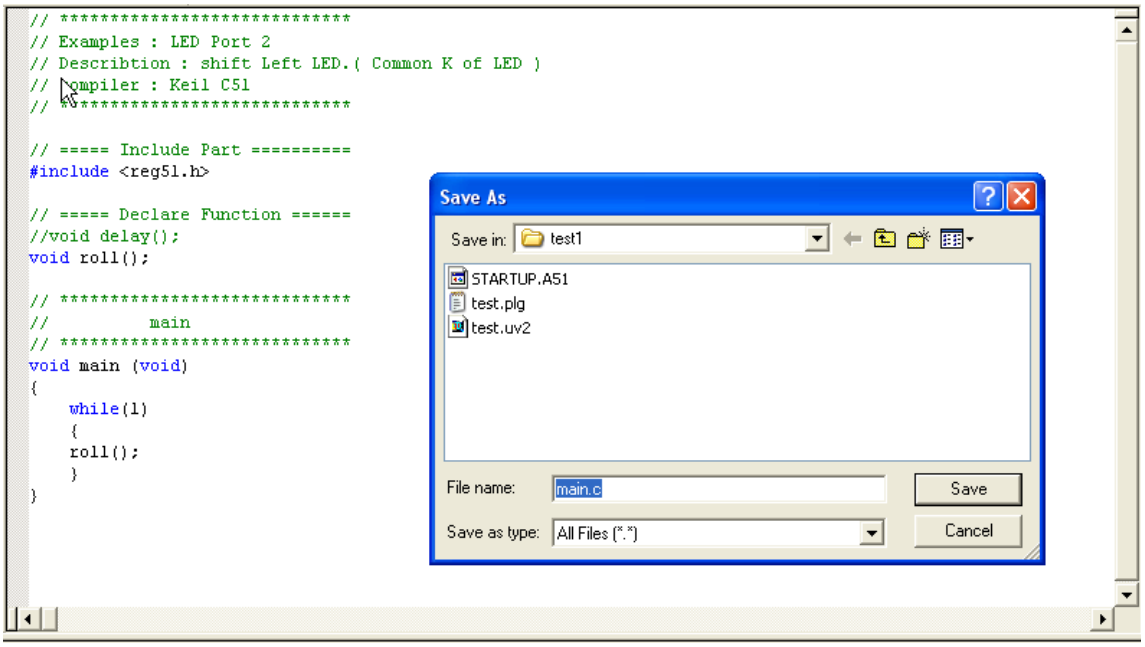
Tip : ผู้เขียนแนะนำว่าควรจะสร้าง Folder สำหรับแต่ละ Project ด้วยการกดที่ Icon (ดูรูปด้านบน) จากนั้นให้พิมพ์ชื่อ New Project ลงในช่อง File name เช่น test ซึ่ง *µVision2* จะสร้างไฟล์ test.UV2 ขึ้นมาให้

ขั้นที่ 3. จากนั้นกดปุ่ม Save หลังจากนั้นโปรแกรมจะให้อ่านเลือกเบอร์อุปกรณ์ที่ผู้อ่านใช้งาน ซึ่งในที่นี้ผู้เขียนใช้ CPU ของบริษัท Atmel เบอร์ AT89C5131 จากนั้นกดปุ่ม OK

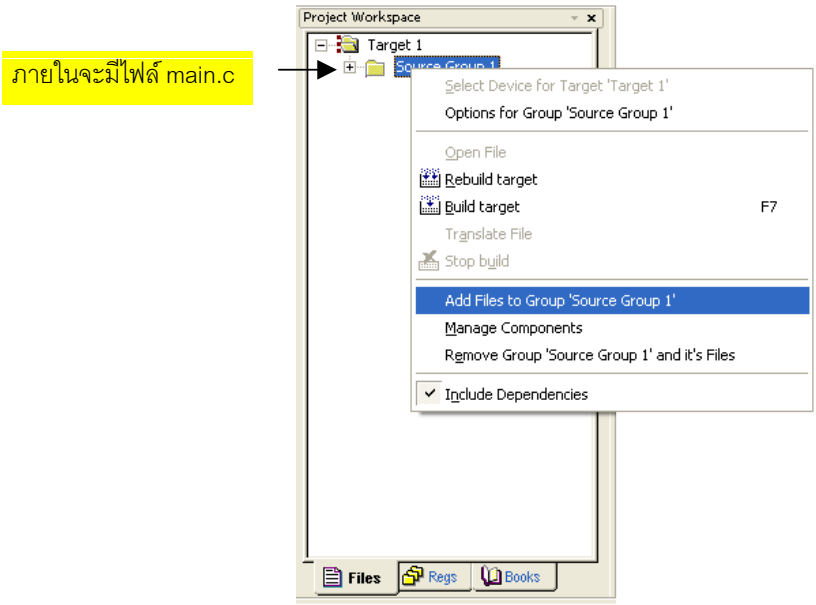


การสร้างไฟล์โค๊ด (Creating New Source Files)

ขั้นที่ 4. เมื่อมาถึงตรงนี้ ผู้อ่านได้ทำการสร้าง Project file ใหม่เป็นที่เรียบร้อยแล้ว ต่อไปผู้อ่านจะต้องสร้าง Source file ใหม่ขึ้นมาโดยไปที่เมนู File => New ซึ่งผู้อ่านจะได้ Editor window เปล่าๆ มา 1 แผ่น ซึ่งผู้อ่านสามารถจะเขียนโค๊ดของผู้อ่านลงไปได้, จากนั้นทำการ Save โดยไปที่เมนู File => Save as... จากนั้นใส่ชื่อ แล้วตามด้วยนามสกุล .C ในที่นี้ผู้เขียน Save ชื่อ main.c



ขั้นที่ 5. จากนั้นให้ผู้อ่านเพิ่ม Source file (ที่ได้เขียนไว้ก่อนนี้) เข้ามาใน Project ซึ่งทำได้หลายวิธีในการที่จะเพิ่ม Source file เข้ามาใน Project เช่น Click ขวาที่โฟล์เดอร์ Source Group1 จากนั้นเลือก add files to Group 'Source Group 1' แล้วตามด้วยเลือกไฟล์ main.c จากนั้นกดปุ่ม Add

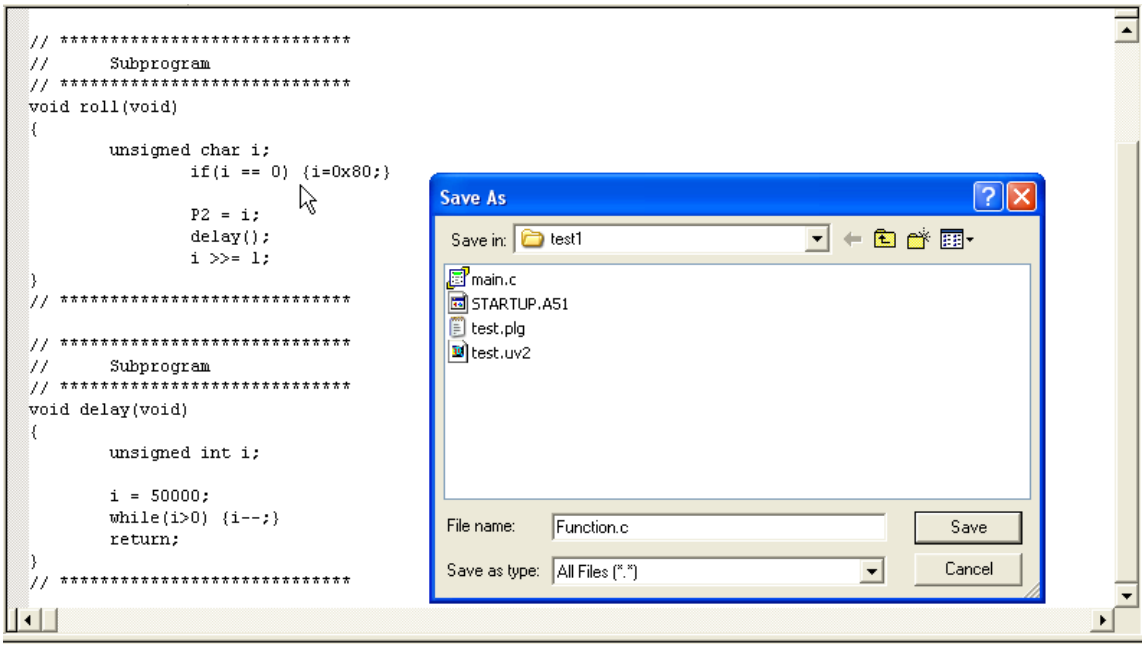


ขั้นที่ 6. จากโปรแกรมในขั้นตอนที่ 4 ผู้อ่านจะเห็นว่ามีการเรียกใช้งานฟังก์ชั่นชื่อ roll ซึ่งผู้เขียนได้สร้างไว้อีกไฟล์หนึ่งชื่อ Function.C ซึ่งภายในไฟล์นี้จะมี 2 ฟังก์ชั่น คือ ฟังก์ชั่น roll และ ฟังก์ชั่น delay ซึ่งเขียนได้ดังนี้ คือ

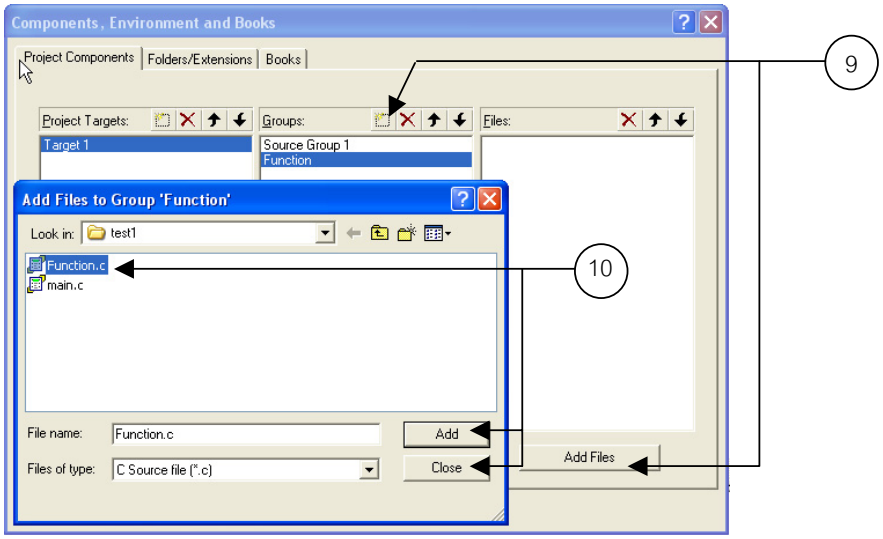
```
// ===== Include Part ==========
#include <reg51.h>      // Tool will fine in Path :" C:\Keil\C51\INC "

// ===== Declare Function ======
void delay();                   // Necessary
void roll();                    // Necessary

// ****************************
//    Subprogram
// ****************************
void roll(void)
{
      unsigned char i;
            if(i == 0) {i=0x80;}

            P2 = i;
            delay();
            i >>= 1;
}
// ****************************
```

```
// ****************************
//     Subprogram
// ****************************
void delay(void)
{
      unsigned int i;

      i = 50000;
      while(i>0) {i--;}
      return;
}
// ****************************
```

ขั้นที่ 7. ให้ผู้อ่านเปิด Editer แผ่นใหม่ขึ้นมาจากเมนู File => New… แล้วพิมพ์โค๊ดโปรแกรมในขั้นที่ 6 จากนั้นทำการ Save จากเมนู File => Save as… โดยตั้งชื่อว่า Function.C



ขั้นที่ 8. จากนั้นผู้เขียนจะสร้างโฟล์เดอร์ใน Project file อีก 1 อันชื่อ Function ซึ่งจะใช้เก็บไฟล์ Function.c ในความเป็นจริงผู้อ่านไม่จำเป็นต้องสร้างโฟล์เดอร์ Function ก็ได้โดยรวมไฟล์ Function.c ไว้ในโฟล์เดอร์เดียวกับไฟล์ main.c แต่ทั้งนี้เพื่อการจัดกลุ่มไฟล์ให้เป็นหมวดหมู่มากยิ่งขึ้น ให้ผู้อ่านไปที่เมนู Project => Components,Environment, Books

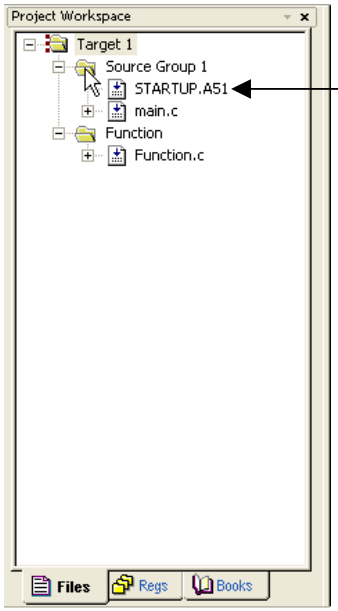


ขั้นที่ 9. ให้ผู้อ่านสร้างโฟล์เดอร์โดยกดที่ Icon แล้วใส่ชื่อ Function จากนั้นให้คลิ๊กที่ชื่อ Function เพื่อทำให้มีแถบสีขึ้นมา แล้วกดปุ่ม Add Files

ขั้นที่ 10. ให้เลือกไฟล์ Function.c แล้วกดปุ่ม Add และ Close ตามลำดับ

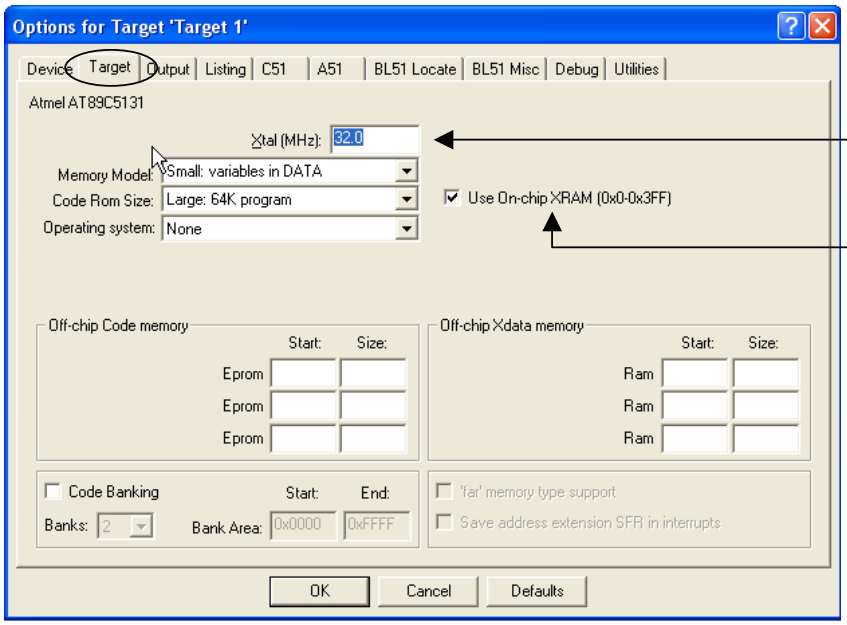
ขั้นที่ 11. กดปุ่ม OK เพื่อออกจากหน้าต่าง Components,Environment, Books

ขั้นที่ 12. ให้ผู้อ่านดูใน Project Wrokspace – Files ผู้อ่านจะเห็นว่ามีโฟล์เดอร์ Function และ ภายในมีไฟล์ Function.c เพิ่มเข้ามา



Note : รายละเอียดเกี่ยวกับไฟล์ Startup.a51 ผู้ใช้สามารถหาอ่านได้จากไฟล์ชื่อ GS51.pdf ในโฟล์เดอร์ Datasheets\Get start keil file ในบทที่ 10 หน้า197 ใน CD

ขั้นที่ 13. จากนั้นผู้อ่านจะต้องกำหนดค่าต่างๆของ Hardware โดยไปที่เมนู Project => Option for Target 'Target 1' และ คลิ๊กแทบ Target

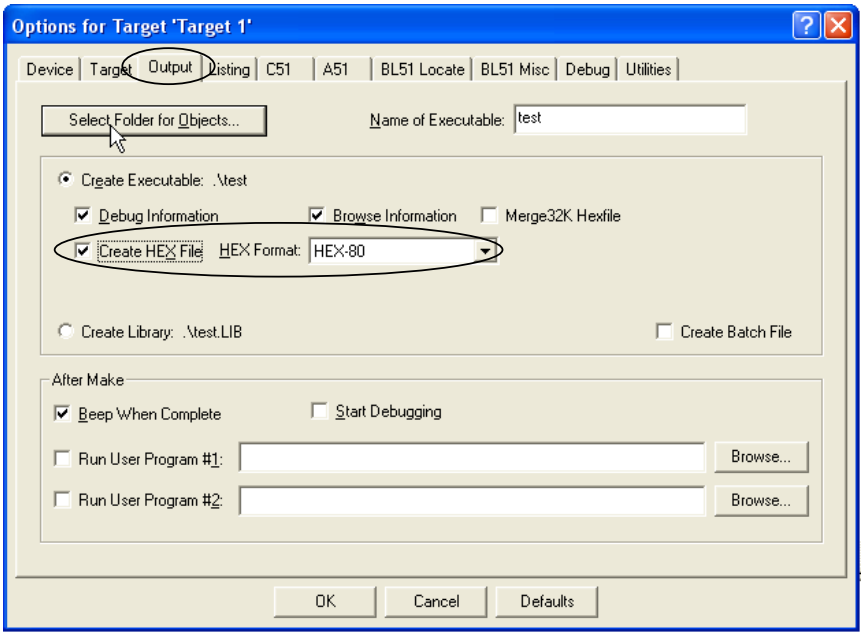


กำหนดค่าความถี่ x'tal โดยทั่วไปผู้อ่านไม่ต้องเปลี่ยนแปลงก็ได้

กรณี CPU AT89C5131 ซึ่งภายในมี XRAM = 1k ผู้อ่านจะต้องติ๊กในช่อง User On-chip XRAM…

Note : รายละเอียดอื่นผู้อ่านสามารถอ่านได้จาก หน้า 62 ในไฟล์ GS51.pdf

ขั้นที่ 14. จากนั้นไปที่แทบ Output แล้ว ติ๊กที่ช่อง Create HEX File จากนั้นกดปุ่ม OK



ขั้นที่ 15. จากนั้นให้ผู้อ่านไปที่เมนู Project => Rebuild all target files เพื่อทำการ Compile ซึ่งผู้อ่านได้ได้ไฟล์ test.Hex มาถ้าการ Compile ไม่มี Error .....!!

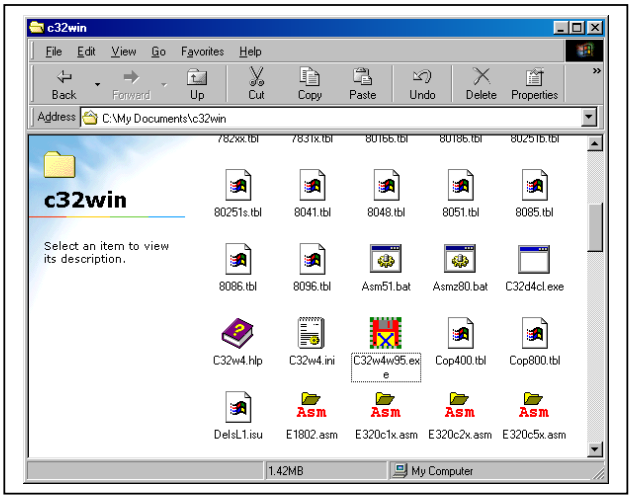
เมื่อมาถึงตรงนี้ผู้เขียนหวังว่าผู้อ่านจะสามารถใช้งาน C Compiler ของ Keil ได้ในระดับหนึ่งแล้ว เพื่อเป็นจุดเริ่มต้นสำหรับผู้ที่เริ่มต้นที่จะเขียนโปรแกรมภาษา C กับ Microcontroller ในตระกูล 8051 ซึ่งถ้าเนื้อหามีข้อผิดพลาดประการใดก็ขอน้อมรับไว้ ณ. ที่นี้

ภาคผนวก ข.

การติดตั้ง และ การใช้งาน Cross 32 V4.0 เบื้องต้น

การติดตั้งโปรแกรม Cross 32 V4.0

โปรแกรม Cross 32 นั้นจัดเป็นโปรแกรมครอสแอสแซมเบอร์ (Cross-Assembler) ถูกพัฒนาโดยบริษัท " Data Sync Engineering" โปรแกรมตัวนี้มีจุดเด่นที่หน้าสนใจตรงที่มันเป็นโปรแกรมจำพวก Universal Cross-Assembler ซึ่ง สามารถใช้แปลงคำสั่งได้ทั้ง CPU Mcs51, Z80 และ อีกหลายๆ ตระกูล ซึ่งโปรแกรม Cross 32 ที่ทางบริษัทแถมไปให้พร้อมกับบอร์ดนั้นจะเป็น Version ทดลองใช้งาน การติดตั้งโปรแกรม Cross 32 ใน Version ทดลองใช้นี้ติดตั้งง่ายมาก เพียงแค่ Copy ไฟล์โปรแกรม Cross 32 ไปวางไว้ใน ไดร์ C (ไดร์ไหนก็ได้) แค่นี้ก็เป็นอันเสร็จเรียบร้อย เมื่อต้องการจะเปิดโปรแกรมก็เพียงแค่ Double Click ที่ไฟล์ C32w4w95.exe ซึ่งเป็นรูปไอคอนสีแดง ดังแสดงในรูปข้างล่าง



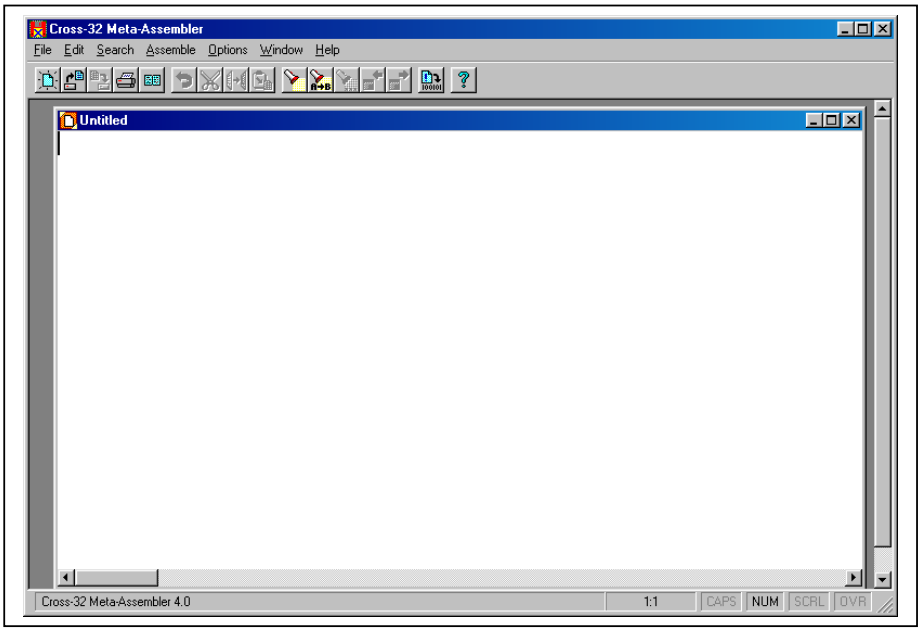
รูปที่ 1

สิ่งสำคัญอีกประการหนึ่ง คือ ใน Folder ที่ใช้เก็บโค๊ดโปรแกรม (ที่ผู้อ่านได้พัฒนาขึ้น) ซึ่งจะเป็นไฟล์นามสกุล .ASM จะต้องทำการ Copy ไฟล์ 8051.tbl ไว้ใน Folder นั้นด้วย ถ้าอย่างนั้นเวลาทำการแปลงไฟล์จะเกิด error

การใช้งานโปรแกรม Cross 32 V4.0

โปรแกรม Cross 32 เป็นโปรแกรม Assembler ที่มี Text Editor ในตัว ดังนั้น ผู้ใช้สามารถทำการพัฒนาโปรแกรมได้จากโปรแกรมนี้เลย หรือ ผู้ใช้จะใช้โปแกรม Notepad หรือ โปรแกรม Text Editor อื่นๆ ก็ได้ แต่ในที่นี้ผู้เขียนจะอธิบายการใช้งานโปรแกรม Cross 32 พอสังเขป โดยใช้ Text Editor ที่มีในตัว

โปรแกรมนี้เลย ซึ่งถ้าผู้อ่านต้องการรายระเอียดของกฎเกณฑ์การใช้งานต่างๆ ของโปรแกรมนี้ผู้อ่านสามารถหาอ่านได้จากหัวข้อ " **ภาคผนวก ค. คำสั่งเทียมใน** Cross 32 V4.0 " จากนั้นเปิดโปรแกรม Cross 32 ขึ้นมา และ click ที่เมนู File => New จะมีหน้าต่าง ดังแสดงในรูปข้างล่าง



รูปที่ 2

เมื่อเริ่มต้นเขียนโปรแกรม, 2 บรรทัดแรกของโปรแกรมผู้อ่านจะต้องกำหนดคำสั่งเทียมของเบอร์ CPU และ ชนิดของ Output File ให้กับโปรแกรมเสียก่อนเนื่องจากโปรแกรม Cross 32 สามารถใช้งานกับ CPU ได้หลายเบอร์ และ นอกจากนั้นยังสร้าง Output File ได้หลาย Format ดังนั้น สำหรับ บอร์ดทดลอง CP-JR51-USB v1.0 นี้ใช้ MCU เบอร์ AT89C5131 ซึ่งมีการทำงานหลัก (Core) แบบ 8052 ดังนั้น จึงใช้งานได้กับชุดคำสั่ง 8051 แต่ผู้อ่านจะต้องไปเพิ่มรีจิสเตอร์ SFR บางตัวที่อยู่นอกเหนือจากตัวมาตรฐาน 8052 โดยดูได้จากตัวอย่างในหัวข้อต่อๆ ไป ซึ่งไฟล์ส่วนหัวเขียนได้ดังนี้ คือ

```
CPU    "8051"      ; แปลเป็น Code คำสั่งของ MCS-51
HOF    "INT8"      ; สร้าง Output File เป็นแบบ Intel Hex 8 bit
```

หลังจาก 2 บรรทัดข้างบนแล้วในบรรทัดต่อมาจะเป็นพื้นที่ของโปรแกรมของผู้ใช้ โดยหลักการเขียนโปรแกรมคร่าวๆ มีหลักดังนี้คือ

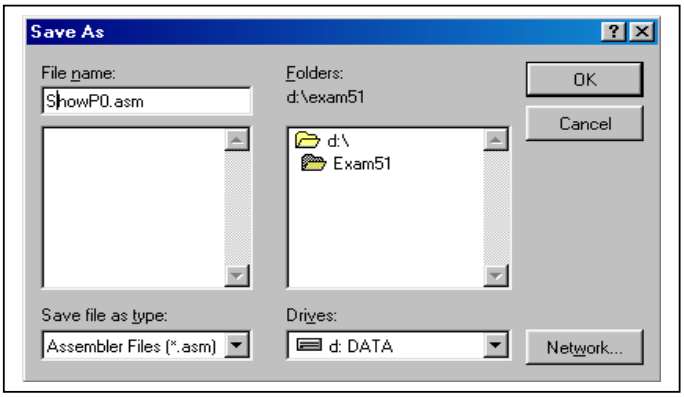
- การเริ่มต้นโปรแกรมจะต้องขึ้นต้นด้วยคำสั่งเทียม ORG 0000H ซึ่งมีความหมายว่า เป็นการกำหนดตำแหน่งเริ่มต้นของหน่วยความจำโปรแกรม และ จะตามด้วย Address เริ่มต้นซึ่งในที่นี้คือ Address 0000H การเขียนห้ามเขียนชิดทางซ้ายสุดต้องเว้นวรรคอย่างน้อย 1 ตัวอักษร
- การกำหนดค่าคงที่ให้กับตัวแปรนั้นสามารถทำได้โดยใช้คำสั่งเทียม EQU ซึ่งดูได้จากตัวอย่างข้างล่าง โดย ชื่อตัวแปลจะต้องอยู่ชิดทางซ้ายมือสุด (ห้ามเว้นวรรค)
- ในการเขียน Label จะใช้เป็นตัวกำหนดตำแหน่งอ้างอิงให้กับ Address ของหน่วยความจำโปรแกรม โดยสิ่งที่ต้องระวัง คือ จะต้องใส่เครื่องหมาย " : " ต่อหลังชื่อ Label โดยห้ามเว้นวรรค และ ตัวอักษร "&" ห้ามใช้ในการเขียน Label
- ในการเขียนโปรแกรมคำสั่งต่างๆ เช่น MOV A,#3EH คำสั่งจะต้องเว้นวรรคอย่างน้อย 1 ตัวอักษรจากทางซ้ายมือเสมอ
- เมื่อเขียนโปรแกรมจนถึงบรรทัดสุดท้ายจะต้องปิดด้วยคำสั่ง END ซึ่งเป็นคำสั่งจบโปรแกรม ซึ่งจะบอกให้ตัวแปลภาษาหยุดการแปลโปรแกรม ซึ่งถ้าไม่มีคำสั่งนี้อาจจะทำให้การแปลภาษาเกิดความผิดพลาดได้

Example : ตัวอย่างข้างล่างนี้เป็นตัวอย่างไฟวิ่งออกทางพอร์ต 0 โดยการวิ่งจะวิ่งไปทางขวา

```
            CPU    "8051.TBL"    ; Processor declaration
            HOF    "INT8"        ; Intel 8-bit hexcode

            ORG    0000H         ; Reset vector
DATA        EQU    0FEH
START:      MOV    A,#DATA       ; DEFIND FIRST VALUE
loop:       RR     A
            MOV    P0,A
            MOV    R2,#100
L1:         mov    R3,#5h
L2:         mov    R4,#5h
            djnz   R4,$
            djnz   R3,l2
            djnz   R2,L1
            sjmp   loOP

            END
```

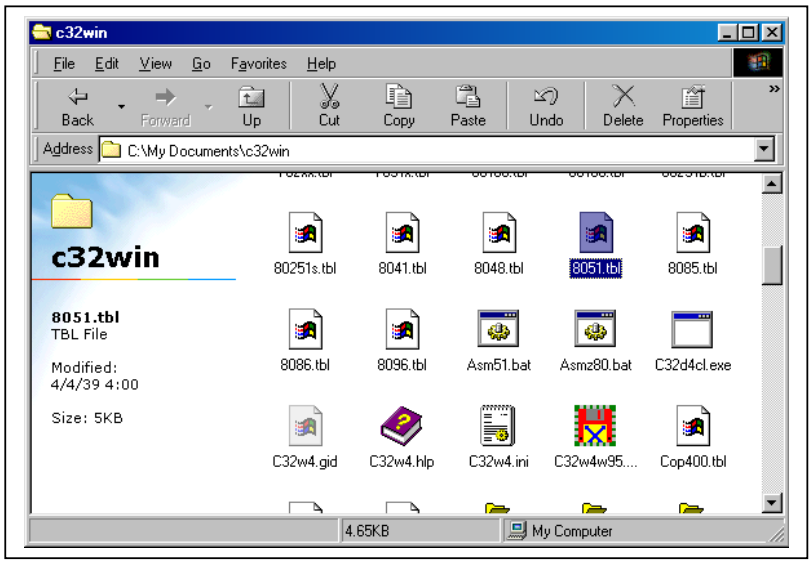
รูปที่ 3 แสดงโปรแกรม ShowP0.asm

เมื่อเขียนโปรแกรมเสร็จแล้ว ให้ผู้ใช้ Save As ไฟล์ ซึ่งในที่นี้ผู้เขียนจะ Save ไว้ที่ ไดร์ D:\Exam51 แล้ว ตั้งชื่อ ในช่อง File name ชื่อ ShowP0.asm จากนั้นกดปุ่ม OK ซึ่งแสดงได้ดังรูปข้างล่าง คือ



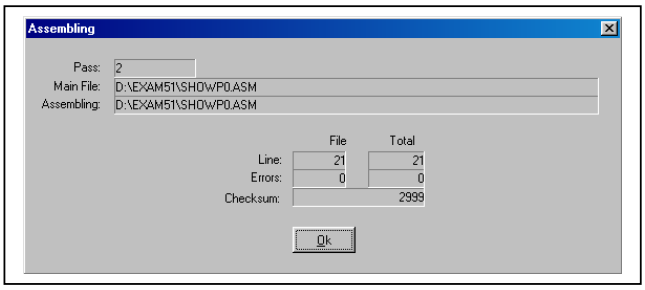
รูปที่ 4 รูปแสดงการ Save ไฟล์

จากนั้นให้ Copy ไฟล์ 8051.tbl ไว้ในโฟล์เดอร์ Exam51 ด้วย โดยไฟล์นี้จะอยู่ในโฟล์เดอร์ c32win ดังแสดงในรูปข้างล่าง



รูปที่ 5

หลังจากนั้นจะเริ่มต้นแปลโปรแกรมกันเลยโดยให้ผู้อ่านไปที่เมนู Assemble => Mail File… เพื่อกำหนดไฟล์ที่ต้องการแปลเป็นไฟล์ .HEX จากนั้นเลือกไปที่ไฟล์ D:\Exam51\ ShowP0.asm และ กดปุ่ม OK จากนั้นให้ไปที่เมนู Assemble => Asseble… จะปรากฏดังภาพข้างล่าง จากนั้นให้กดที่ปุ่ม Assemble ซึ่งผู้ใช้จะต้องสังเกตในช่อง Error: จะต้องเป็น 0 ถ้าเป็นตามนี้แสดงว่าการแปลโปรแกรมนั้นสมบูรณ์ ซึ่งในโฟล์เดอร์ Exam51 จะมีไฟล์เพิ่มขึ้นมา 3 ไฟล์ คือ ShowP0.ERR, ShowP0.HEX, ShowP0.LST ซึ่งถึงตรงจุดนี้ผู้ใช้จะได้ไฟล์ .Hex แล้ว...!!!



รูปที่ 6

จากที่กล่าวมาทั้งหมดหวังว่าผู้อ่านจะสามารถใช้งานโปรแกรม Cross 32 ได้ ในระดับหนึ่งแล้ว เพื่อเป็นจุดเริ่มต้นสำหรับผู้อ่านที่ไม่เคยใช้โปรแกรมนี้มาก่อน ความผิดพลาดของเนื้อหาใดๆ ทางผู้เขียนต้องขออภัยไว้ ณ. ที่นี้...!!

ภาคผนวก ค.

คำสั่งเทียมใน Cross 32 V4.0

การเขียนภาษา Assembly นั้นจะมีความยุ่งยากกว่าการเขียนภาษาระดับสูงอื่นๆ โดยเฉพาะอย่างยิ่ง เมื่อต้องการเรียกข้อมูลของหน่วยความจำออกมาใช้จะเป็นการยุ่งยากอย่างมากว่าตำแหน่ง Address ใดถูกจัดสรรหน่วยความจำไว้สำหรับเก็บค่าข้อมูลอะไรบ้าง และ ยังอาจเกิดความสับสนในขณะที่เรียกใช้ได้ ทำให้เกิดความผิดพลาดขึ้นได้ง่าย ดังนั้น เพื่อความสะดวก โปรแกรมที่ทำหน้าที่เป็นตัว Assembler ต่างๆ จึง มีการคิดค้นสร้างคำสั่งอีกกลุ่มหนึ่งขึ้นมาเพื่ออำนวยความสะดวกให้กับผู้เขียนโปรแกรม โดยกลุ่มคำสั่งดังกล่าวนี้นิยมเรียกกันทั่วๆ ไปว่า คำสั่งเทียม ซึ่งคำสั่งเทียมนี้จะไม่ใช่คำสั่งสำหรับสั่งงาน CPU แต่จะถูกนำมาใช้เพื่อบ่งบอกให้โปรแกรม Assembler ได้ทราบถึงสิ่งที่ได้กำหนดขึ้น เพื่อนำไปใช้ในการแปลโปรแกรมเท่านั้น ซึ่งคำสั่งเทียมของ Cross32 นั้นมีอยู่มากมาย แต่ในที่นี้จะขอกล่าวถึงเฉพาะในส่วนของคำสั่งเทียมที่มีความจำเป็น และ ใช้งานกันบ่อยให้ทราบพอสังเขป คือ

1. **คำสั่ง** ORG (Origin)

คำสั่ง ORG นี้เป็นคำสั่งที่ใช้กำหนดค่าเริ่มต้นของหน่วยความจำโปรแกรม และ หน่วยความจำข้อมูล โดยตัวเลขที่อยู่ถัดจากคำสั่ง ORG นี้จะเป็นตำแหน่งเริ่มต้นของโปรแกรม เช่น

```
        ORG   0000H
Main:   MOV   SP,#60H
        .
        .
```

2. **คำสั่ง** EQU (Equate)

เป็นคำสั่งที่ใช้กำหนดค่าคงที่ให้กับตัวแปร ปกติคำสั่งนี้จะอยู่ในส่วนต้นของโปรแกรม โดยคำสั่ง EQU จะช่วยในการกำหนดค่าตัวแปรของค่าตำแหน่ง Address หรือ ค่าต่างๆ ได้ ดังแสดงในตัวอย่างข้างล่าง คือ

```
PORT_A:   EQU   20H
PORT_B:   EQU   PORT_A+1
PORT_C:   EQU   PORT_A+2
```

จากตัวอย่างข้างบน ในทุกๆ ตำแหน่งของโปรแกรมที่มีการอ้างถึง PORT_A จะมีความหมายเหมือนกับอ้างถึงค่า 20H เสมอ ส่วนค่าของ PORT_B และ PORT_C จะมีค่าเป็น 21H และ 22H ตามลำดับ ถ้ามีการเปลี่ยนแปลงค่าของ PORT_A เป็นค่าอื่นๆ ก็จะส่งผลให้ PORT_B และ PORT_C ถูกเปลี่ยนแปลงค่าตามไปด้วย ซึ่งลักษณะของคำสั่ง EQU จะเหมือนกับการกำหนดค่า Constant ในภาษาสูงนั้นเอง

3. **คำสั่ง** DFB (Define Byte)

เป็นคำสั่งที่ใช้กำหนดค่าข้อมูลคงที่ขนาด 1 Byte ไว้ในหน่วยความจำโปรแกรม โดยคำสั่งนี้จะถูกนำมาใช้ประโยชน์ในการกำหนด ตาราง TABLE ต่างๆ ที่จะนำมาใช้ในโปรแกรมโดยจำนวนข้อมูลแต่ละค่าจะถูกแบ่งแยกออกจากกันด้วยเครื่องหมาย คอมม่า(,) ดังแสดงในตัวอย่างข้างล่าง คือ

```
TAB_LED:    DFB    00000001B
            DFB    02H,04H,08H
TAB_MSG:    DFB    “ABC”,00H
```

จากตัวอย่างที่ผ่านมาเมื่อสั่งให้โปรแกรม Assembler แปลโปรแกรมแล้ว ที่ตำแหน่ง Address ของ TAB_LED จะมีค่าคงที่อยู่ 4 ไบต์ คือ 01H, 02H, 04H และ 08H เรียงต่อเนื่องกันไป ส่วน TAB_MSG จะได้ค่าเป็นรหัส ASCII ของตัวอักษร ABC เรียงกันไป คือ 41H, 42H, 43H และ ปิดท้ายด้วย 00H

4. **คำสั่ง** DWL (Define Word : LSB First)

เป็นคำสั่งที่ใช้สำหรับกำหนดข้อมูลค่าคงที่ ขนาด 2 ไบต์ (Word) โดยต้องกำหนด ค่าของ Byte ต่ำก่อนแล้วจึงตามด้วย Byte สูง และ แบ่งแยกข้อมูลแต่ละ Word ด้วยเครื่องหมาย คอมม่า (,)

5. **คำสั่ง** DFS (Define Storage)

เป็นคำสั่งที่ใช้สำหรับจองพื้นที่ของหน่วยความจำโปรแกรม โดยจะจองเท่ากับ จำนวนตัวเลขที่ระบุไว้หลังคำสั่ง DFS เช่น

```
            ORG    0000H
DSP_BUFF:   DFS    4
```

จากตัวอย่างข้างบนจะเป็นการบอกให้โปรแกรม Assembler ได้ทราบว่าชื่อ DSP_BUFF นั้น ถูกกำหนดให้มีขนาดของหน่วยความจำ 4 ไบต์ โดยมีตำแหน่ง 0000H – 0003H ซึ่งถ้ามีการอ้างถึงชื่อ DSP_BUFF นี้ในโปรแกรมก็จะมีความหมายเหมือนกับการอ้างถึงค่าตำแหน่ง Address 0000H – 0003H ด้วย โดยถ้าใช้การอ้างแบบ 8 บิต เมื่อต้องการใช้ตำแหน่งที่ 2 ของชื่อ DSP_BUFF ก็อาจจะใช้รูปแบบเป็น DSP_BUFF+1 แทนได้ ซึ่งจะเป็นประโยชน์ มากในการเขียนโปรแกรมเพราะสามารถสื่อความหมายถึงหน้าที่ของหน่วยความจำได้ดีกว่าการอ้างเป็นค่าตัวเลข และ ยังง่ายต่อการเปลี่ยนแปลงแก้ไขตำแหน่ง Address ในหน่วยความจำด้วย

6. **คำสั่ง** END

เป็นคำสั่งที่ใช้สำหรับบอกให้ Assembler ทราบว่าได้ถึงจุดสิ้นสุดการแปลโปแกรมแกรมแล้วซึ่งต้องมีคำสั่งนี้ไว้ท้ายของโปรแกรมเสมอ และ หากเขียนโปรแกรมต่อหลังจากคำสั่ง END นี้ Assembler จะไม่แปลส่วนที่ต่อจาก คำสั่ง END นี้ให้

7. **คำสั่ง** HOF (HEXADECIMAL OUTPUT FORMAT)

เป็นคำสั่งที่ใช้สำหรับกำหนดรูปแบบของ Output File โดย กำหนดได้ 3 ลักษณะ คือ Binary Output, Intel Hex Output, Motorola Hex Output ดังนี้ คือ

Binary File ประกอบด้วย BIN8, BIN16, BIN32 เป็น Output File แบบ Binary 8, 16 และ 32

Intel Hex File ประกอบด้วย INT8 และ INT16 เป็น Output File แบบ Intel Hex ขนาด 8 บิต และ 16 บิต ตามลำดับ

Motorola Hex File ประกอบด้วย MOT8, MOT16, MOT32 เป็น Output File แบบ Motorola Hex ขนาด 8 บิต, 16 บิต และ 32 บิต