Chapter 12. Internal EEPROM Access

1. FLASH Memory and EEPROM

As discussed in Chapter 2 of the PIC 16F877 architecture, there is 8K Word of FLASH program memory and 256 bytes of EEPROM (Electrically Erasable Programmable Read Only Memory). EEPROM is a convenient memory which can be re-written almost indefinitely by again and again without using any external device. Since the RAM area where your filer register and variables are stored in your code is erased once power to the PIC board is turned on, this EEPROM can be a valuable storage space if you application needs a permanent data storage so that power turn-off does not affect your code. One example is storing telephone numbers into EEPROM so that it keeps the numbers even when power is lost.

FLASH memory is where your code resides and EEPROM can be access by your code.

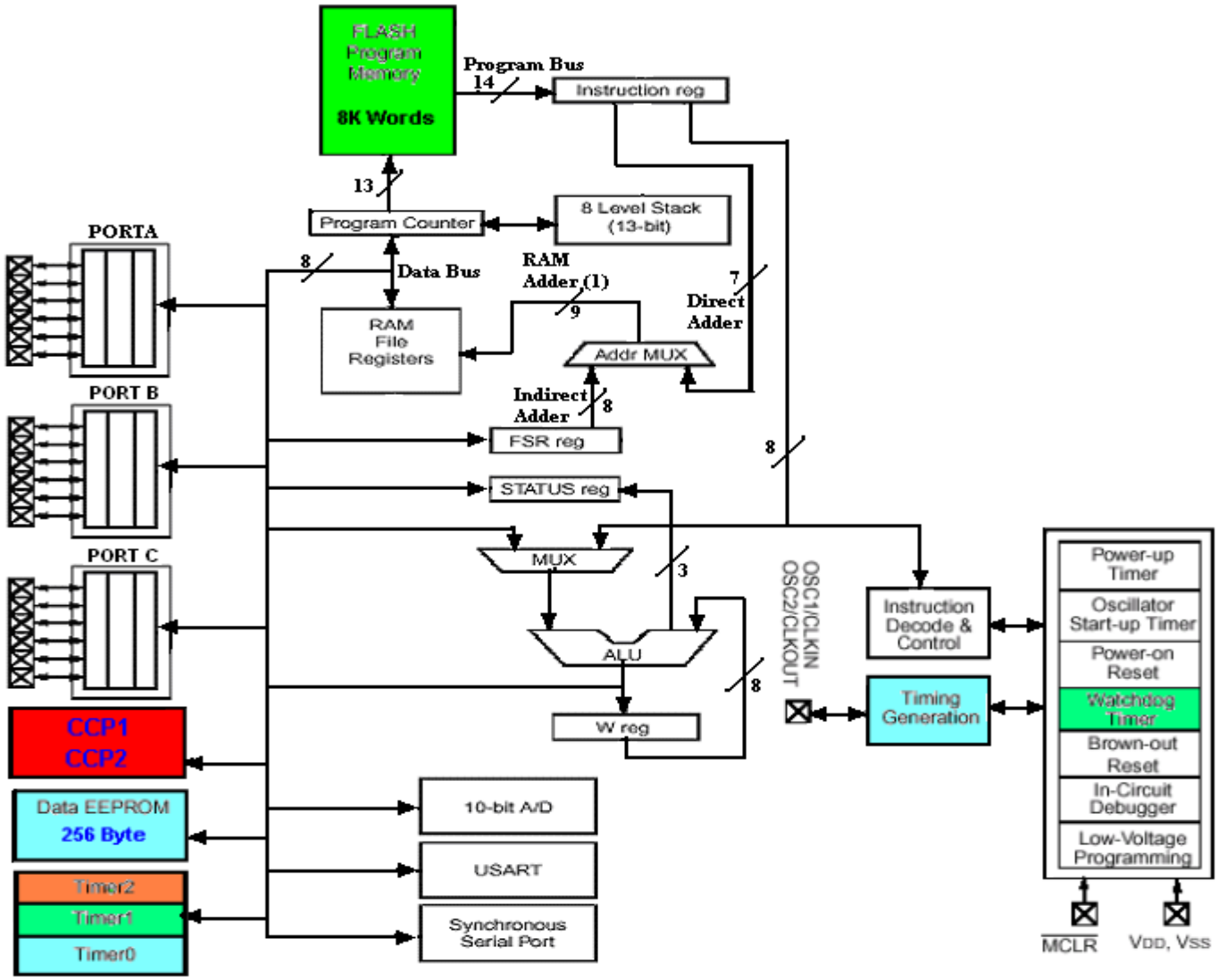


Fig. 84 FLASH memory and EEPROM

By the way, FLASH memory is a type of EEPROM chip. It has a grid of columns and rows with a cell that has two transistors at each intersection. The two transistors are separated from each other by a thin oxide layer. One of transistors is known as a floating gate and the other one is the control gate. The floating gate's only link to the row, or wordline, is through the control gate. As long as this link is in place, the cell has a value of "1". To change the value to a "0" requires a curious process called Fowler-Nordheim tunneling. Tunneling is used to alter the placement of electrons in the floating gate. An electrical charge, usually 10-13 volts, is applied to the floating gate. The charge comes from the column, or bitline, enters the floating gate and drains to a ground.

This charge causes the floating gate transistor to act like an electron gun. The excited electrons are pushed through and trapped on other side of the thin oxide layer, giving it a negative charge. These negatively charged electrons act as a barrier between the control gate and the floating gate. A special device called a cell sensor monitors the level of the charge passing through the floating gate. If the flow through the gate is greater than fifty percent of the charge, it has a value of "1". When the charge passing through drops below the fifty percent threshold, the value changes to "0". Flash memory uses in-circuit wiring to apply the electric field to the entire chip, or to predetermined sections known as blocks. This erases the targeted area of the chip, which can then be rewritten. Flash memory works much faster than traditional EEPROMs because instead of erasing one byte at a time, it erases a block or the entire chip, and then rewrites it.

This chapter discusses how we write and rewrite to EEPROM.

2. EEPROM Access

EEPROM is readable and writable in the normal operation of the code. As we saw (or could not see) from the File Register Map, there is no EEPROM space in the filer register area. This is not surprising since filer register area is with RAM not with EEPROM. This causes some problem in accessing EEPROM, since our access to I/O ports in particular is done by reading from or writing to file registers such as PORTB, TRISB, etc. The access to EEPROM is done through the following 4 file registers: EECON1 (address at 0x18C), EECON2 (not a physically implemented register), EEADR (address at 0x10C), and EEDATA (address at 0x10D).

EECON1 register controls reading and writing of EEPROM. As illustrated in the figure, the EEPGD bit is for selecting FLASH memory or EEPROM access. Apparently, for EEPROM access, the EEPGD bit must be cleared. Other bits are discussed as we move on the read and write processes.

EEPGD	---	---	---	WRERR	WREN	WR	RD
bit 7							bit 0

bit 7: **EEPGD**: Program / Data EEPROM Select bit
1 = Accesses Program memory
0 = Accesses data memory
(This bit cannot be changed while a read or write operation is in progress)

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

bit 3: **WRERR**: EEPROM Error Flag bit
1 = A write operation is prematurely terminated
(any $\overline{\text{MCLR}}$ reset or any WDT reset during normal operation)
0 = The write operation completed

bit 2: **WREN**: EEPROM Write Enable bit
1 = Allows write cycles
0 = Inhibits write to the EEPROM

bit 1: **WR**: Write Control bit
1 = initiates a write cycle. (The bit is cleared by hardware once write is complete)
The WR bit can only be set (not cleared) in software.
0 = Write cycle to the EEPROM is complete

bit 0: **RD**: Read Control bit
1 = Initiates an EEPROM read RD is cleared in hardware.
The RD bit can only be set (not cleared) in software.
0 = Does not initiate an EEPROM read

EECON2 register is a dummy register only for EEPROM access control. This register is not a physical one and this is only for a behind-scene process for write process. EEADR register is to indicate the EEPROM address we try to access. By the way, EEPROM address starts from 0x00 and the highest address is 0xFF.

EEDATA hold an 8-bit data read from EEPROM or to be written to EEPROM. In other words, the data byte read from EEPROM is stored into EEDATA register and the data must be moved from EEDATA to another file register for the next byte read. In writing, a data byte to be written must be written to EEDATA. In the write process, the address space in EEPROM is first erased then written into the address space.

3. Reading EEPROM

To read a data from an address space in EEPROM, we must write the address to EEADR register. Then, we clear the EEPGD bit of EECON1 register to select EEPROM, not FLASH memory for our access. The third step is to set the read control bit RD of the EECON1 to initiate reading from EEPROM. The RD bit will be cleared automatically after the read process. At the very next instruction cycle after the RD bit is set, the read data from EEPROM is stored in the EEDATA register. EEDATA register holds the data until another read operation is performed or a write process is initiated.

Now let's make an EEPROM read subroutine. Since when we read we read multiple data from EEPROM, this subroutine after reading a byte increases the address pointer by one so that we don't have to put an address to EEADR every time we read. The name of subroutine is EEREAD.

```
;READING A DATA from EEPROM ----SUBROUTINE
EEREAD
      banksel     EECON1
      bcf         EECON1, EEPGD     ;point to DATA MEMORY
      bsf         EECON1, RD  ;READ ENABLE
      banksel     EEDATA
      movf        EEDATA, 0   ;MOVE THE DATA To W Reg
      incf        EEADR
      RETURN
```

This subroutine, after reading a byte, stores the data in to **W** register. The EECON1 register bits are to be declared on top of the code like this:

```
;for EEPROM
EEADR       EQU   0x10D       ;BANK 2
EEDATA      EQU   0x10C       ;BANK 2
EECON1      EQU   0x18C       ;BANK 3
EECON2      EQU   0x18D       ;BANK 3
EEPGD       EQU   0x07        ;for EECON1
WR          EQU   0x01        ;for EECON1
RD          EQU   0x00        ;for EECON1
WREN        EQU   0x02        ;for EECON1
```

Now then discus about how we call the subroutine EEREAD in a situation that we read 24 ten-digit phone numbers already stored starting from address 0x00 in EEPROM, and print the numbers on the PC screen using serial communication we already discussed. Remember that each digit of a phone number occupies 1 byte, and thus 1 memory address space.

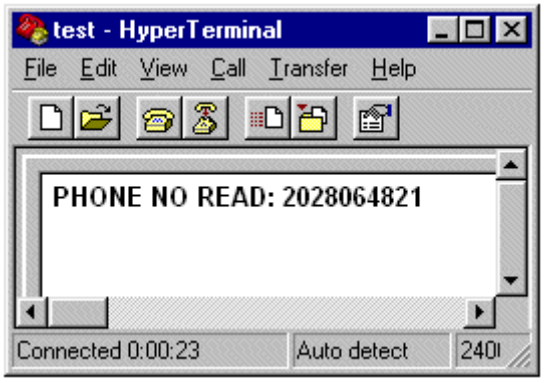
In the code below, COUNT will track the number of phone numbers. The COUNT can go up to 24d or 0x18 since we stored 24 phone numbers. The variable TEN tracks the number of digits in a phone number. Since each phone number has 10 digits, the variable TEN can go up to 10d or 0x0A. When a byte data is retrieved to W register from EEDATA, it is sent TXREG register for a serial communication with PC. For serial communication the TXPOLL subroutine we developed in the Serial communication for serial transmission is recycled in the code.

```
      banksel     COUNT
      clrf        COUNT       ;number of phone numbers reset to 0
      banksel     EEADR
      movlw       0x00
      movwf       EEADR       ;The starting address for EEPROM as 0
      banksel     TEN
      clrf        TEN         ;the number of bytes in a phone number
                              ;reset to 0
AGAIN                         ;this label is for the next phone number
RAGAIN                        ;this label is for next byte in a phone number
      call        EEREAD      ;EEPROM read subroutine is called
                              ;with starting address=0
                              ;the byte data read was now in W register
```

```
;Type to SCREEN


      banksel     PIR1

TXPOLL
      btfss       PIR1, TXIF
      goto        TXPOLL      ;check if transmission can start
      movwf       TXREG       ;now send the content of W register to PC
      incf        TEN         ;TEN is now increased by 1
      btfss       TEN, 0x03
      GOTO        RAGAIN
      btfss       TEN, 0x01
      GOTO        RAGAIN      ;check if TEN is now 10
                              ;if NOT, goto RAGAIN to read the next byte

      call        CRLF        ;if TEN=0x0A (i.e., all 10 bytes are read
                              ;then in the PC screen, we want to change
                              ;the line


      banksel     COUNT
      incf        COUNT       ;COUNT is now increased by 1 since
                              ;the first phone number is successfully
                              ;read from EEPROM and printed on PC
      btfss       COUNT, 0x04
      GOTO        AGAIN
      btfss       COUNT, 0x03 ;now check if COUNT=0x18
      GOTO        AGAIN       ;if Not, read the next phone number
FINI  NOP
      GOTO        FINI        ;if COUNT=0x18, stop the code
```

If you read the comment lines, the code is self-explanatory. The subroutine CRLF is a routine we discussed in serial communication of changing a line to the next. In the code, checking if a number is 0x0A (for TEN) is done by bit checking of the number is focus. Since 0x0A is 1010b, therefore, if bit 1 and bit 3 of TEN are 1, then we can say TEN=0x0A. For COUNT, since the number in focus is 0x18=0001 1000b, we check bit 3 and 4 of COUNT if COUNT=0x18.

Now since we are familiar with EEPROM read, let's further proceed to have somewhat nicer printout on the screen. What we want is to print out a prompt line (or a banner line) saying that "PHONE NO READ:" followed by a 10 digit phone number as illustrated in the figure.



The banner part can be best handled if we use indirect instruction with FSR and IDNF register. Remember that FSR is the pointer for INDF. And increasing or decreasing FSR actually does the operation to INDF register. The bottom line is this: INDF contains a data in the RAM of an address pointed by the content of FSR. For example, if FSR=0x31, then INDF holds the data stored in the address 0x31 of the RAM.

So the structure of this venture is, first, to write the banner bytes. Second, read the first phone number of 10 digits and print them after the colon(:) sign of the banner. Then, we pint the banner again in the next line, followed by another phone number. This process goes on until we read and print all 24 phone numbers.

First consider how we store each data byte of the banner '`PHONE NO READ:`' in our code. Since we have only 14 bytes of data, we choose to allocate 14 variables, one for each byte data, in the `CBLOCK ..ENDC`, then we will put each data byte to each variable, the following segments of the code show the process of variable allocation inside the RAM and give a byte value to each of the variables, respectively.

First, the variable allocation part declares variables like EMB1, ELB2, etc for 'P', "H', etc:

```
CBLOCK      0x30  ;to store 14 byte data for banner
                  ;from RAM address 0x30
      EMB1
      EMB2
      EMB3
      EMB4
      EMB5
      EMB6
      EMB7
      EMB8
      EMB9
      EMB10
      EMB11
      EMB12
      EMB13
      EMB14       ;the address for this variable is 0x3E
ENDC
```

Second, now we declare or define the specific data byte for each of the variables:

```
      banksel     EMB1
; Write Banner Characters into RAM area
      movlw       'P'
      movwf       EMB1
      movlw       'H'
      movwf       EMB2
      movlw       'O'
      movwf       EMB3
      movlw       'N'
      movwf       EMB4
      movlw       'E'
      movwf       EMB5
      movlw       0x20
      movwf       EMB6        ;EMB6 now holds 'space' key
      movlw       'N'
      movwf       EMB7
      movlw       'O'
      movwf       EMB8
      movlw       0x20
      movwf       EMB9        ;EMB9 is also with 'space'
      movlw       'R'
      movwf       EMB10
      movlw       'E'
      movwf       EMB11
      movlw       'A'
      movwf       EMB12
      movlw       'D'
      movwf       EMB13
      movlw       ':'
      movwf       EMB14
```

The following code is a complete code including all necessary subroutines and such. One small problem of this code is that, since we have not written to EEPROM, the phone number part of the output on your PC would be all blank. For a complete and meaningful run would come after we write phone numbers to the EEPROM.

```
;EEP-r.asm
;
;
;This reads the EEPROM stored max 24 10-digit phone numbers
;
;EEPROM-->W-->PC (READING)
;
;
;Asynchronous mode
;
;Terminal set up: 8N1 19200

        list P = 16F877
INDF        EQU   0x00        ;Indirect Register
INTCON      EQU   0x0b
STATUS      EQU   0x03
PIR2        EQU   0x0d
```

```
FSR         EQU   0x04        ;File Selection Register (Indirect addr ptr)
TXSTA       EQU   0x98        ;TX status and control
RCSTA       EQU   0x18        ;RX status and control
SPBRG       EQU   0x99        ;Baud Rate assignment
TXREG       EQU   0x19        ;USART TX Register
RXREG       EQU   0x1A        ;USART RX Register
PIR1        EQU   0x0C        ;USART RX/TX buffer status (empty or full)
RCIF        EQU   0x05        ;PIR1<5>: RX Buffer 1-Full  0-Empty
TXIF        EQU   0x04        ;PIR1<4>: TX Buffer 1-empty 0-full
TXMODE      EQU   0x20        ;TXSTA=00100000 : 8-bit, Async
RXMODE      EQU   0x90        ;RCSTA=10010000 : 8-bit, enable port, enable RX
BAUD        EQU   0x0F        ;0x0F (19200), 0x1F (9600)

;for EEPROM
EEADR       EQU   0x10D       ;BANK 2
EEDATA      EQU   0x10C       ;BANK 2
EECON1      EQU   0x18C       ;BANK 3
EECON2      EQU   0x18D       ;BANK 3
EEPGD       EQU   0x07        ;for EECON1
WR          EQU   0x01        ;for EECON1
RD          EQU   0x00        ;for EECON1
WREN        EQU   0x02        ;for EECON1
GIE         EQU   0x07        ;for INTCON
EEIF        EQU   0x04        ;for PIR2
;

;RAM for DELAY SUBROUTINE
        CBLOCK    0x20              ; RAM AREA for USE at address 20h
            Kount120us   ;Delay count (number of instr cycles for delay)
            Kount100us
            Kount1ms
            Kount10ms
            Kount1s
            Kount10s
            Kount1m
            TEN         ;0x0a
            COUNT       ;0x18
        ENDC                  ;end of ram block

;RAM for Banner Printing on a Monitor
      CBLOCK      0x30
            EMB1
            EMB2
            EMB3
            EMB4
            EMB5
            EMB6
            EMB7
            EMB8
            EMB9
            EMB10
            EMB11
            EMB12
            EMB13
            EMB14
      ENDC
```

```
;The Next 5 lines must be here
;because of bootloader arrangement
;Bootloader first execute the first 4 addresses
;then jump to the address what the execution directs
;=======================================================
        org       0x0000            ;line 1
        GOTO      START       ;line 2 ($0000)

;=======================================================
;start of the program from $0005
      org         0x0005
START call        delay1s
      call        delay1s
      call        Async_mode        ;RX TX initialization
;
;Sending data by Software Polling
      banksel     EMB1
; Write Banner Characters into RAM area
      movlw       'P'
      movwf       EMB1
      movlw       'H'
      movwf       EMB2
      movlw       'O'
      movwf       EMB3
      movlw       'N'
      movwf       EMB4
      movlw       'E'
      movwf       EMB5
      movlw       0x20
      movwf       EMB6
      movlw       'N'
      movwf       EMB7
      movlw       'O'
      movwf       EMB8
      movlw       0x20
      movwf       EMB9
      movlw       'R'
      movwf       EMB10
      movlw       'E'
      movwf       EMB11
      movlw       'A'
      movwf       EMB12
      movlw       'D'
      movwf       EMB13
      movlw       ':'
      movwf       EMB14

;starting logo: TYPE!
;=======================================================
      banksel     COUNT

      clrf        COUNT

      banksel     EEADR
      movlw       0x00
      movwf       EEADR
;Type Inquiry
```

```
Again

      banksel     FSR
      movlw       0x30        ;
      movwf       FSR         ;pointer @ 0x30

      banksel     PIR1
TXPOLL2
      btfss       PIR1, TXIF
      goto        TXPOLL2           ;if full, wait
      movf        INDF,0            ;W is the content of (FSR)
      movwf       TXREG
      incf        FSR
      btfss       FSR, 0x06   ;0100 0000 is the next of 0011 1111
      goto        TXPOLL2

;Banner Printing Finished

;EEPROM READING
      banksel     TEN
      clrf        TEN

RAGAIN
      call        EEREAD
;Type to SCREEN

      banksel     PIR1

TXPOLL
      btfss       PIR1, TXIF
      goto        TXPOLL
      movwf       TXREG
      incf        TEN
      btfss       TEN, 0x03
      GOTO        RAGAIN
      btfss       TEN, 0x01
      GOTO        RAGAIN
      call        delay1s           ;a little delay down (For displaying,
                                    ;this time delay is important)
      call  CRLF              ;End of a phone number

;Just to show the process

      banksel     COUNT
      incf        COUNT
      btfss       COUNT, 0x04
      GOTO        AGAIN
      btfss       COUNT, 0x03
      GOTO        AGAIN
FINI  NOP
      GOTO        FINI


;===EEPROM READING SUBROUTINE ================
EEREAD
      banksel     EECON1
      bcf         EECON1, EEPGD     ;point to DATA MEMORY
```

```
      bsf         EECON1, RD  ;READ ENABLE
      banksel     EEDATA
      movf        EEDATA, 0   ;MOVE THE DATA To W Reg
      incf        EEADR       ;data pointer increased
      RETURN
;

;=======================================================
;RX TX Initialization with Asyc Mode
;Async_mode Subroutine
Async_mode
      banksel     SPBRG
      movlw       BAUD        ;B'00001111' (19200)
      movwf       SPBRG
      banksel     TXSTA
      movlw       TXMODE            ;B'00100000' Async Mode
      movwf       TXSTA
      banksel     RCSTA
      movlw       RXMODE            ;B'10010000' Enable Port
      movwf       RCSTA
      return


;CRLF subroutine
CRLF
      banksel     PIR1
CR    btfss       PIR1, TXIF
      goto        CR
      movlw       0x0D
      movwf       TXREG
LF    btfss       PIR1, TXIF
      goto        LF
      movlw       0x0A
      movwf       TXREG
      return

;DELAY SUBROUTINE
;=========================================================
;DELAY SUBROUTINES


; 1 instruction cycle for 20MHz clock is 0.2 us
; Therefore 120 uS delay needs 600 instuction cycles
;  600 =199*3 +3 ---->Kount=199=0xC7
;  or  =198*3 +6 ---->Kount=198=0xC6
;  or  =197*3 +9 ---->Kount=197=0xC5

Delay120us
      banksel     Kount120us
      movlw       0xC5        ;D'197'
      movwf       Kount120us
R120us
      decfsz      Kount120us
      goto        R120us
      return
;
;100us delay needs 500 instruction cycles
```

```
;  500 =166*3 +2 ---->Kount=166=0xA6
;  or  =165*3 +5 ---->Kount=165=0xA5
;  or  =164*3 +8 ---->Kount=164=0xA4
Delay100us
      banksel     Kount100us
      movlw       0xA4
      movwf       Kount100us
R100us
      decfsz      Kount100us
      goto        R100us
      return

;
;10ms delay
; call 100 times of 100 us delay  (with some time discrepancy)
Delay10ms
      banksel     Kount10ms
      movlw       0x64              ;100
      movwf       Kount10ms
R10ms call        delay100us
      decfsz      Kount10ms
      goto        R10ms
      return
;
;

;1 sec delay
;call 100 times of 10ms delay
Delay1s
      banksel     Kount1s
      movlw       0x64
      movwf       Kount1s
R1s   call        Delay10ms
      decfsz      Kount1s
      goto        R1s
      return
;
;
;10 s delay
;call 10 times of 1 s delay
Delay10s
      banksel     Kount10s
      movlw       0x0A              ;10
      movwf       Kount10s
R10s  call        Delay1s
      decfsz      Kount10s
      goto        R10s
      return
;
;1 min delay
;call 60 times of 1 sec delay
Delay1m
      banksel     Kount1m
      movlw       0x3C                    ;60
      movwf       Kount1m
R1m   call        Delay1s
      decfsz      Kount1m
```

```
      goto        R1m
      return
;=====================================================
        END                                 ;END OF THE CODE
```

4. EEPROM Writing

Now let's move on to writing process of EEPROM. And we're all anxious of doing this. The sequence of writing to EEPROM is long and each must be followed as explained here. First, write the address of EEPROM where you want to store your byte data into EEADR register. Second, write the byte data itself into EEDATA register. Now we have memory address and data. Third, enable the EEPROM writing process by setting the WREN bit (bit 2) of EECON1 register. The fourth step involves several things to be done in the following order:

1. Disable Interrupt: We have not discussed about this subject yet, but take it easy. Just clearing the GIE (Global Interrupt Enable) bit (INTCON<7>) of INTCON register would be suffice for disabling interrupt.
2. Write the value 0x55 to EECON2: Remember that EECON2 is not a physical register and this process is only for giving necessary writing time to EEPROM. Just follow this.
3. Write the value of 0xAA to EECON2: Same as above. No question. Just do this.
4. Set WR bit (bit 1) of EECON1 register: This actually starts the writing itself.
5. Enable Interrupt: Setting the GIE bit of INTCON register would do this step.

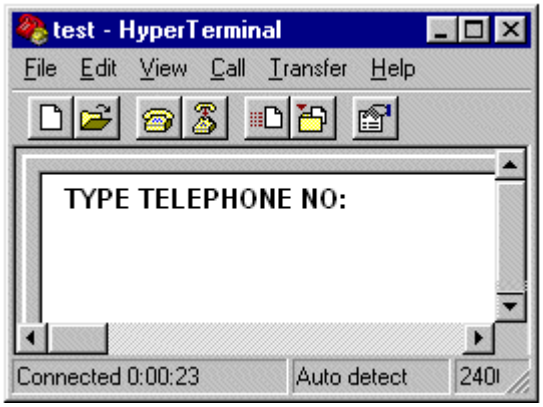
During the writing, WR bit is remained set and it is cleared when the writing is done (in addition, when writing is done, the EEIF (bit 4) bit of PIR2 register is set). Therefore, we have to monitor the WR bit and if it's cleared, we clear WREN bit to disable (or end) writing to EEPROM. Also, we have to clear EEIF bit of PIR2 register. So here goes the fifth step: monitor WR bit if it is cleared. If not, we have keep monitoring until it is cleared. Six, when WR bit is cleared, clear WREN bit (EECON1<2>) and EEIF bit (PIR2<4>).

Now let's develop an EEPROM write subroutine, `EEWRITE`. As in `EEREAD` subroutine, at the end of writing, we will increase the EEPROM memory address by 1for the next byte write. One thing I did not mention in the above six steps of EEPROM writing is the clearing of EEPGD bit of EECON1 for EEPROM access: setting would try to access FLASH memory. We have to clear EEPGD bit before we proceed, as we did in `EEREAD`.

```
;EEWRITE subroutine
EEWRITE
      banksel     EEDATA
      movwf       EEDATA            ;writing to EEPROM

      banksel     EECON1
      bcf         EECON1, EEPGD     ;EEMPROM access (not FLASH)
      bsf         EECON1, WREN      ;Write Enable
;Required Sequence
      bcf         INTCON, GIE       ;Disable Interrupt
                                    ;GIE = INTCON<7>
      movlw       0x55
```

```
      movwf       EECON2
      movlw       0xAA
      movwf       EECON2
      bsf         EECON1, WR        ;Begin to WRITE
      bsf         INTCON, GIE       ;Enable Interrupt
;End of the Sequence
;Check if WR bit is cleared
WAIT  btfsc       EECON1,WR
      goto        WAIT
      bcf         EECON1, WREN      ;DISABLE WRITING
;End of Writing
;Clear the EEIF bit of PIR2 (Write Complete Interrupt)
;BANK 0
      banksel     PIR2
      bcf         PIR2, EEIF
      banksel     EEADR
      incf        EEADR
      RETURN
```

The way we repeat for 10 digit 24 telephone numbers is the same as in EEPROM reading example code. And the way we type a banner asking for you to type your telephone number using keyboard and on to PC screen is the same. The banner for writing telephone numbers is shown in the figure.



After the banner prompt, as soon as the first number is typed, the number is stored in to W, then echoed back to screen and written to EEPROM. This sequence of processes goes on for the entire 10 numbers of the first phone number. Then at the next line, another banner will be printed waiting for the first numeral of the second phone number. This thing repeats for 24 times for the 24 phone number we write to EEPROM.

The following code is again a full complete code (except subroutines) for EEPROM write. The subroutines needed for this code are the same ones we used for EEPROM read. However, the EEWRITE subroutine is included in the code. So readers should include other necessary subroutines copied from the EEPROM read code. Banner writing using indirect instruction using FSR and INDF registers are the same with EEPROM read case.

```
;EEP-w.asm
;
```

```
;This program is to read a 10-digit telephone number from PC keyboard
;And store them in to EEPROM Data Memory
;
;EEPROM has 256 Byte
;Therefore This PROGRAM allows to read maximum 25 phone numbers
;But for programming simplicity it allows only 24 numbers (0x18=0001 1000)
;
;A companion program EEP-r.asm retrieves the number of phone numbers
; from the result of this program
;
;PC--->W--->EEPROM  (WRITING)
;
;Asynchronous mode
;Terminal set up: 8N1 19200

        list P = 16F877
INDF        EQU   0x00        ;Indirect Register
INTCON      EQU   0x0b
STATUS      EQU   0x03
PIR2        EQU   0x0d
FSR         EQU   0x04        ;File Selection Register (Indirect addr)
TXSTA       EQU   0x98        ;TX status and control
RCSTA       EQU   0x18        ;RX status and control
SPBRG       EQU   0x99        ;Baud Rate assignment
TXREG       EQU   0x19        ;USART TX Register
RXREG       EQU   0x1A        ;USART RX Register
PIR1        EQU   0x0C        ;USART RX/TX buffer status (empty or full)
RCIF        EQU   0x05        ;PIR1<5>: RX Buffer 1-Full  0-Empty
TXIF        EQU   0x04        ;PIR1<4>: TX Buffer 1-empty 0-full
TXMODE      EQU   0x20        ;TXSTA=00100000 : 8-bit, Async
RXMODE      EQU   0x90        ;RCSTA=10010000 : 8-bit, enable port, enable RX
BAUD        EQU   0x0F        ;0x0F (19200), 0x1F (9600)
RP0         EQU   0x05        ;STATUS<5>
RP1         EQU   0x06        ;STATUS<6>
;for EEPROM
EEADR       EQU   0x10d       ;BANK 2
EEDATA      EQU   0x10c       ;BANK 2
EECON1      EQU   0x18c       ;BANK 3
EECON2      EQU   0x18d       ;BANK 3
EEPGD       EQU   0x07        ;for EECON1
WR          EQU   0x01        ;for EECON1
RD          EQU   0x00        ;for EECON1
WREN        EQU   0x02        ;for EECON1
GIE         EQU   0x07        ;for INTCON
EEIF        EQU   0x04        ;for PIR2
;

;RAM for DELAY SUBROUTINE
CBLOCK      0x20
            Kount120us   ;Delay count (number of instr cycles for delay)
            Kount100us
            Kount1ms
            Kount10ms
            Kount1s
            Kount10s
            Kount1m
            TEN         ;ten digit
```

```
            COUNT       ; for counting up to 24 numbers (0x18)
        ENDC                    ;end of ram block

;RAM for Banner Printing on a Monitor
      CBLOCK      0x30
            EMB1
            EMB2
            EMB3
            EMB4
            EMB5
            EMB6
            EMB7
            EMB8
            EMB9
            EMB10
            EMB11
            EMB12
            EMB13
            EMB14
            EMB15       ;0x3f
      ENDC

;=======================================================
        org       0x0000            ;line 1
        GOTO      START       ;line 2 ($0000)
;=======================================================
      org         0x0005
START call        delay1s
      call        delay1s
      call        Async_mode        ;RX TX initialization
;
;Sending data by Software Polling
      banksel     EMB1

; Write Banner Characters into RAM area
      movlw       'T'
      movwf       EMB1
      movlw       'Y'
      movwf       EMB2
      movlw       'P'
      movwf       EMB3
      movlw       'E'
      movwf       EMB4
      movlw       0x20        ;SPACE
      movwf       EMB5
      movlw       'P'
      movwf       EMB6
      movlw       'H'
      movwf       EMB7
      movlw       'O'
      movwf       EMB8
      movlw       'N'
      movwf       EMB9
      movlw       'E'
      movwf       EMB10
      movlw       0x20        ;space
      movwf       EMB11
```

```
      movlw       'N'         ;
      movwf       EMB12
      movlw       'O'         ;
      movwf       EMB13
      movlw       ':'         ;
      movwf       EMB14
      movlw       0x20        ;space
      movwf       EMB15
;starting logo: TYPE!
;=====================================================
;Type Inquiry
      banksel     COUNT
      movlw       0x0         ;
      movwf       COUNT       ;initialize COUNT

      banksel     EEADR
      movlw       0x0
      movwf       EEADR       ;EEADR initialization at 0x00
again
      banksel     FSR
      movlw       0x30
      movwf       FSR         ;Pointer @0x30

      banksel     PIR1
TXPOLL2
      btfss       PIR1, TXIF
      goto        TXPOLL2           ;if full, wait
      movf        INDF,0            ;W is the content of (FSR)
      movwf       TXREG
      incf        FSR
      btfss       FSR, 0x06   ;0100 0000 is the the next of 0011 1111
      goto        TXPOLL2
;     call        delay1s
;Banner Printing Finished
;
;Reading Phone Number Part
      movlw       0x00
      movwf       TEN         ;10 - digit
RXPOLL
      btfss       PIR1, RCIF  ;RX Buffer Full?  (i.e. Data Received?)
      goto        RXPOLL
      movf        RXREG,0           ;received data  to W
;echo the data
ECHO  btfss       PIR1, TXIF
      goto        ECHO        ;if full, wait
      movwf       TXREG       ;echo the character from Terminal
;now WRITING to EEPROM
      call        EEWRITE
      banksel     TEN
      incf        TEN
      btfss       TEN, 0x03
      goto        RXPOLL
      btfss       TEN, 0x01
      goto        RXPOLL
;TYPE LF and CR
      call        CRLF
      incf        COUNT
```

```
      btfss       COUNT, 0x04
      goto        again
      btfss       COUNT, 0x03
      goto        again
fini  nop
      goto        fini

;
;===== EEWRITE subroutine
EEWRITE
      banksel     EEDATA
      movwf       EEDATA            ;writing to EEPROM

      banksel     EECON1
      bcf         EECON1, EEPGD     ;point to DATA Memory
      bsf         EECON1, WREN      ;Enable it
;Required Sequence
      bcf         INTCON, GIE       ;Disable Interrupt
                                    ;GIE = INTCON<7>
      movlw       0x55
      movwf       EECON2
      movlw       0xAA
      movwf       EECON2
      bsf         EECON1, WR        ;Begin to WRITE
      bsf         INTCON, GIE       ;Enable Interrupt
;End of the Sequence
;Check if WR bit is cleared
WAIT  btfsc       EECON1,WR
      goto        WAIT
      bcf         EECON1, WREN      ;DISABLE WRITING
;End of Writing
;Clear the EEIF bit of PIR2 (Write Complete Interrupt)
;BANK 0
      banksel     PIR2
      bcf         PIR2, EEIF
      banksel     EEADR
      incf        EEADR
      RETURN

;=====================================================

;Other subroutines must be placed here
;=====================================================
        END                         ;END of CODE
```