Esercitazioni PIC Baseline

15A - Indirizzamento indiretto, EEPROM

Vediamo di conoscere in cosa consiste l'indirizzamento indiretto.
Inoltre, dato che alcuni Baseline integrano un'area EEPROM, vediamo di capirne il funzionamento.

Indirizzamento diretto

Nell' area RAM si trovano due elementi: la **RAM dati** (*General Purpose Registers*) vera e propria e i registri **SFR** (*Special File Registers*) che controllano le varie funzioni.

Se prendiamo un chip con una mappa di memoria semplice, ad esempio 12F508, troviamo che l'intera area è contenuta in un solo blocco, chiamato "banco" (*Bank*), come visualizzato nel diagramma accanto, con indirizzi da 00h a 1Fh.

Per i Baseline, il banco ha una ampiezza di 32 bytes.

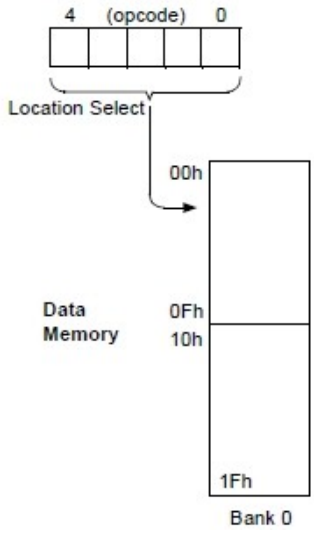
In questo caso si hanno due elementi:

- i primi 7, da 00h a 06h sono SFR
- e i rimanenti 25 sono RAM dati.

Per accedere agli indirizzi di questa area di memoria, utilizziamo il modo che viene definito "diretto", ovvero, semplicemente indicando come oggetto dell' istruzione la locazione su cui vogliamo operare:

```
 CBLOCK 0x10
  temp            ; locazione RAM temporanea
 ENDC
; copio il contenuto di GPIO in RAM
    movf  GPIO,w
    movwf temp
```

Direct Addressing



Ricordiamo che l'opcode dei Baseline ha una ampiezza di 12bit.

Nelle istruzioni di gestione dei dati (**bcf**, **bsf**, **movf**, **mowf**, **subwf**, **addwf**, **btfsc**, **btfss**, **decfsz**, **incf**, **decf**, ecc.), l'opocode contiene sia il codice dell' operazione da eseguire (nei 7 bit di peso maggiore, bit11:6), mentre i 5 di peso minore (bit 4:0) hanno lo scopo di contenere l' indirizzo della locazione RAM da raggiungere.

Dato che l' area RAM è di soli 32 bytes, 5 bit possono coprire tutti gli indirizzi, da 00h a 1Fh.

Quindi non esiste alcun problema nel maneggiare i byte dati e SFR, senza particolari attenzioni o meccanismi da rispettare.

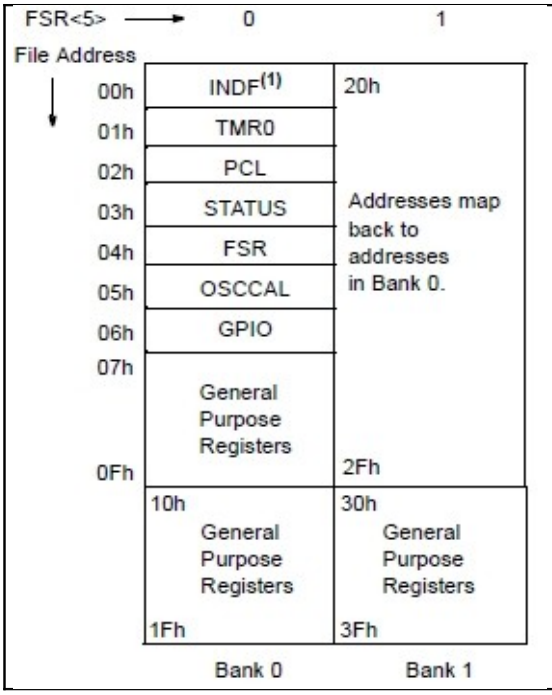
Dal punto di vista operativo, il compito di creare gli opcodes adeguati partendo dal sorgente è demandato al compilatore. Così, ad esempio, se il sorgente contiene la riga:

```
movwf     OSCCAL
```

L'Assembler, prelevando il valore numerico della label **OSCCAL** dal file ***nomeprocessore.inc***, produce nel file ***.hex*** il valore **025h**. Questo deriva da:

11	10	9	8	7	6	5	4	3	2	1	0
Codice istruzione **movwf : 02x**							Indirizzo **OSCCAL : 05h**				
0	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**1**

Se questa forma dell'opcode è identica per tutti i Baseline, va considerato che non tutti hanno la stessa quantità di RAM.



Se consideriamo una altro chip, come 12F509, ci troviamo con una diversa mappa di memoria:

- gli SFR sono sempre 7, ma
- la RAM dati si estende a 41 bytes

Poiché la dimensione del banco è 32 bytes, occorro **DUE** banchi, chiamati **Bank0** e **Bank1**.

Qui, la RAM è divisa in 5 parti:

1. da 00h a 06h : 7 SFR
2. da 07h a 0Fh : 9 bytes di RAM dati
3. da 10h a 1Fh : 16 bytes di RAM dati
4. da 20h a 2Fh : una copia di1 e 2

5. da 30h a 3Fh : 16 bytes di RAM dati

L' area da 00h a 1Fh, 32 bytes, costituisce il banco 0 e quella da 20h a 3Fh costituisce il banco 1.
Ci si può chiedere: dato che l'opcode contiene solamente 5 bit di indirizzo, come è possibile accedere al contenuto del secondo banco?
Per chiarire meglio, supponiamo di voler raggiungere la locazione RAM che ha indirizzo 30h: questo numero, per essere espresso in forma binaria, richiede 6 bit:

10h = 1 0000
30h = 11 0000

Occorre un bit in più per completare l'indirizzo e questo sesto bit, esterno all'opcode, è fornito dal registro **SFR**, per la precisione dal suo bit 5. Quindi, se scrivo:

```
movwf   0x10
```

il contenuto di WREG sarà copiato nella locazione 10h del banco0. Per copiarlo nella locazione 20h, in banco 1, dovrò scrivere:

```
bsf     FSR,5   ; accede al banco 1
movwf   0x30
```

Però, è necessario capire che, se scrivessi:

```
bsf     FSR,5
movwf   0x10
```

l'effetto sarebbe lo stesso: l'indirizzo contenuto nell' opcode è comunque a 5 bit. Quindi, scrivere 10h (10000) o 30h (110000) è la stessa cosa , dato che bit oltre il quinto non trovano comunque posto nell'opcode. Il bit addizionale dipende dal valore di **FSR,5**; i 5 bit più bassi sono uguali per entrambi gli indirizzi:

10h = 1 0000
30h = 11 0000

Ovvero, il bit della commutazione del banco è a 0, tutte le operazioni, comunque indirizzate, sono dirette alla Ram nel banco 0; se porto a 1 il bit, indirizzo tutte le operazioni al banco 1.

A questo proposito, va ricordato che **al POR il bit 5 di FSR viene portato a 0, rendendo accessibile per default il banco 0**; dove serve più RAM, lo spostamento sul banco successivo richiede la manipolazione del bit.

Quindi, se il programma non usa RAM maggiore di quella contenuta nel banco0, il problema della commutazione dei banchi non sussiste.
Dobbiamo, invece, avere a che fare con i banchi quando la necessità di RAM dati supera quanto disponibile nel banco0 o quando parte degli SFR si trova in banchi diversi dal banco 0, come vediamo più avanti in relazione a chip con un numero elevato di risorse integrate, che necessitano di registri di controllo e che non trovano posto nei 32 bytes di un solo banco.

Dato che la nostra regola è quella di non usare assoluti, la procedura corretta è quella di attribuire label agli indirizzi ed utilizzare lo pseudo-opcode **banksel** fornito da MPASM che agisce sul bit in funzione della posizione nei banchi della label in oggetto:

```
mem equ 0x30

  banksel  mem         ; passa al banco che contiene meme (banco1)
  movf     mem,w
  banksel  GPIO        ; passa al banco che contiene GPIO (banco0)
  movwf    GPIO
```

Da notare che, per rendere meno difficoltoso l'operato del programmatore, gli SFR sono resi accessibili in ugual modo da entrambi i banchi; e una questo è valido anche per una piccola parte di RAM dati, da 07h a 0Fh. Si tratta di 9 bytes chiamati Shared RAM (SHR), ovvero RAM condivisa.
Questa soluzione è adottata per ridurre la necessità di passare continuamente da un banco all' altro quando si usa, ad esempio, la memoria nel banco 1: se gli SFR non fossero accessibili anche lì, ci potrebbe essere la necessità di un continuo passaggio trai banchi, cosa che penalizzerebbe la velocità di esecuzione del programma, oltre ad occupare memoria con istruzioni supplementari. In generale, dunque, la tecnica è quella di usare per quanto possibile le risorse di RAM dati in banco 0 e passare ai banchi successivi solo se necessari, ma, una volta in un banco, cercare di restarci il più possibile in modo da minimizzare la necessità di maneggiare il bit di commutazione. La memoria condivisa fa da "ponte", dato che è accessibile allo stesso modo da entrambi i banchi. Quindi funziona correttamente

```
  banksel  1           ; banco 1
  movf     mem,w       ; copia con tenuto dell' indirizzo 0x30
  movwf    GPIO        ; in GPIO
```

dato che GPIO è accessibile in modo identico in entrambi i banchi. E anche:

```
  banksel  1           ; banco 1
  movf     TMR0,w      ; copia contenuto di TMR0 in W
  movwf    0x07        ; copia con tenuto dell' indirizzo 07h
  movwf    0x10        ; copia con tenuto dell' indirizzo 30h
  movf     0x08,w      ; copia contenuto dell' indirizzo 08h in W
  movwf    GPIO        ; in GPIO
```

GPIO, **TMR0** e la shared RAM (07h, 08h) sono accessibili da qualsiasi banco, mentre indirizzando 10h nel banco 1 in realtà agisco su 30h.

In ogni caso, è opportuno, come linea guida nella stesura del sorgente, quando è necessario utilizzare le commutazioni dei banchi, mantenere sotto stretto controllo la situazione.
Una via è quella di organizzare l'impiego della RAM in modo tale da ridurre la necessità di passaggio da un banco all'altro, mantenere quanto possibile le operazioni in banco0 o con l' aiuto della RAM condivisa e, passando in altro banco, riportarsi poi sul banco0 in modo tale da avere un riferimento costante.

Una nota per i piccoli PIC con un solo banco, come nel caso del 12F508: se inseriamo nel sorgente una linea del genere **`banksel 0`** otteniamo dal compilatore un messaggio che ci avvisa della sua inutilità:

```
Message[312] C:\PIC\PROVA.asm 39 : Page or Bank selection not needed for this device.  No code generated.
```

Questo è corretto, dato che la RAM in 12F508 non ha banchi e non sono necessarie istruzioni per la commutazione.

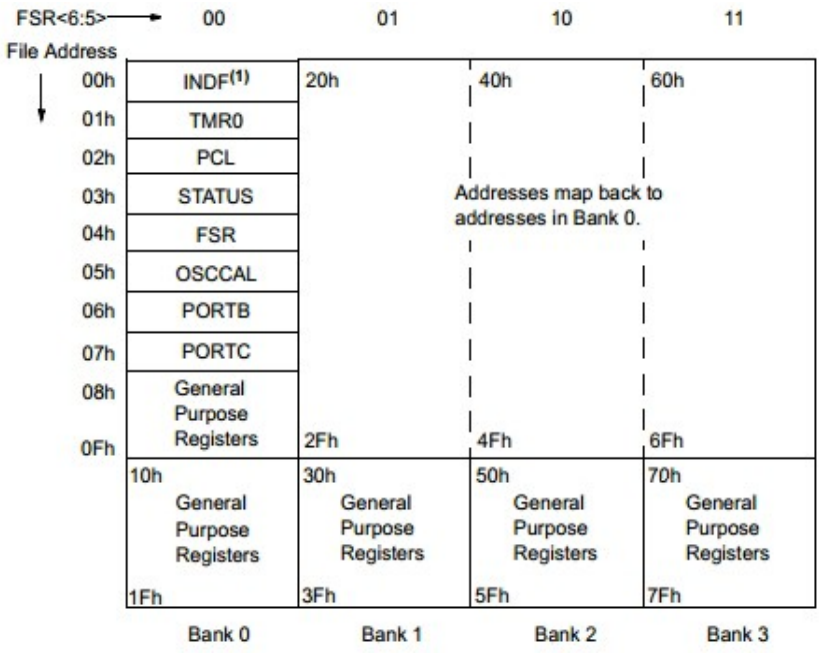
Trattandosi di un "Message", la compilazione ha successo e il file ***.hex*** viene generato: lo scopo del Message è quello di avvisare il programmatore di un evento a seguito del quale l'Assembler ha preso una decisone automatica.

Si deve, poi, considerare che esistono altri chip con mappe di memoria più complesse.

Ad esempio, 16F505 ha 72 bytes di RAM dati che richiedo ben 4 banchi da 32 bytes ciascuno per contenerla assieme agli 8 SFR. Qui, i bit addizionali per passare tra i banchi sono 2 e si tratta sempre di bit del registro FSR, per l' esattezza il bit 5 e 6.

Per chiarire meglio, supponiamo di voler raggiungere la prima locazione del General Purpose Registers di ogni banco: si tratta degli indirizzi 10h, 30h, 50h e 70h:

10h = 1 0000
30h = 11 0000
50h = 101 0000
70h = 111 0000



Lo pseudo-opcode **`banksel`** supportato da MPASM gestisce autonomamente i bit per il cambio del banco in relazione al chip su cui si sta lavorando:

```
temp0 = 0x10        ; in banco 0
temp1 = 0x30        ; in banco 1
temp2 = 0x50        ; in banco 2
temp3 = 0x70        ; in banco 3
; copia il contenuto di temp0 nelle altre locazioni
    movf    temp0, w
   banksel temp1
   movwf   temp1
   banksel temp2
   movwf   temp2
   banksel temp3
```

```
   movwf    temp3
```

dove le pseudo-istruzioni corrispondono a:

opcode	azione
`banksel temp0`	`bcf FSR,5` `bcf FSR,6`
`banksel temp1`	`bsf FSR,5` `bcf FSR,6`
`banksel temp2`	`bcf FSR,5` `bsf FSR,6`
`banksel temp3`	`bsf FSR,5` `bsf FSR,6`

Ricordiamo che scrivendo:

```
temp0 = 0x10       ; in banco 0
temp1 = 0x30       ; in banco 1
temp2 = 0x50       ; in banco 2
temp3 = 0x70       ; in banco 3

; copia il contenuto di temp0 nelle altre locazioni
   movf     temp0, w
   banksel  temp1
   movwf    temp0
   banksel  temp2
   movwf    temp0
   banksel  temp3
   movwf    temp0
```

si ottiene lo stesso risultato, dato che l' indirizzo di **`temp0`** e dei successivi è lo stesso per quanto riguarda i 5 bit bassi.

Dunque, possiamo concludere che:

- l' indirizzamento diretto è quello che utilizziamo scrivendo gli opcodes
- dobbiamo “aggiustare”in modo manuale i bit che commutano i banchi per poter accedere alle risorse dei diversi banchi

---

Indirizzamento indiretto

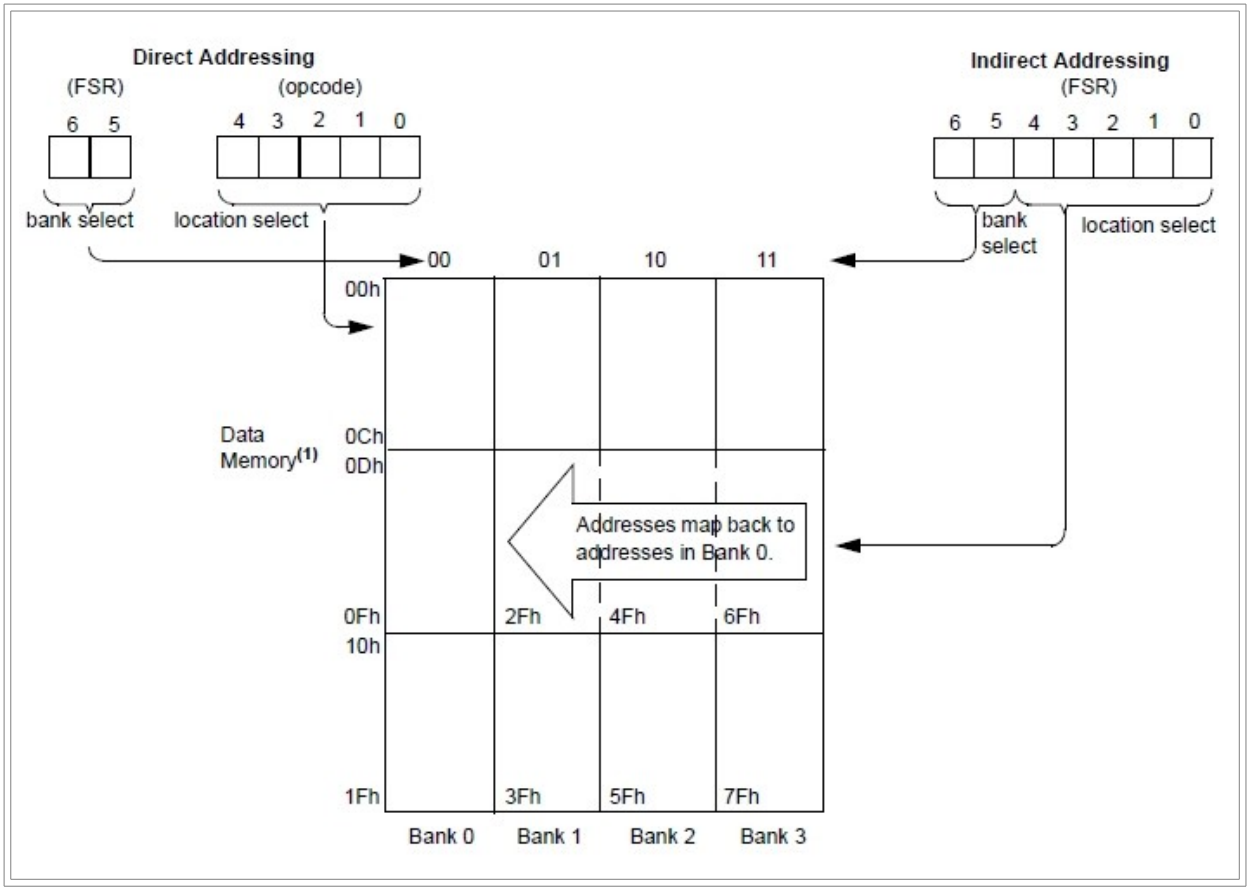
Tuttavia, l’uso intensivo degli switches tra i banchi aggrava il numero delle istruzioni che il processore deve eseguire, oltre a creare con facilità errori nella selezione dei registri.

Questo è uno dei motivi per cui i PIC dispongono di un indirizzamento definito "indiretto" che permette un accesso al contenuto dei banchi senza agire direttamente sui bit di FSR.

Questo sistema utilizza proprio il contenuto del registro **FSR** (*File Select Register*) come puntatore: dato che FSR è ampio 8bit, sono rappresentabili indirizzi in area RAM da 00 a 255 (FFh), il che supera ampiamente quanto disponibile fisicamente.
In effetti, il registro FSR di 16F505 ha 7 bit implementati, il che permette di raggiungere tutti e quattro i banchi; quello di 12F509 ne ha 6, il che permette di raggiungere il contenuto dei due banchi. SFR di 12F508 ha solamente 5 bit implementati, dato che non ne servono di più, non avendo banchi.

Nel modo indiretto si ricorre all' impiego del registro INDF (*Indirect File*): se inseriamo nel registro FSR l' indirizzo da raggiungere, il contenuto della locazione indirizzata sarà accessibile automaticamente attraverso il registro INDF sia in lettura che in scrittura.

Vediamo come funziona il meccanismo: nello schema qui sotto abbiamo graficamente la vista dell'indirizzamento diretto e di quello indiretto.
Nel primo, fornito all' opcode i 5 bit meno significativi dell'indirizzo da raggiungere, lo completiamo con i bit di FSR. La destinazione sarà raggiunta dall'opcode.
Nel secondo caso utilizziamo l'intero FSR come indirizzo della destinazione, che sarà raggiunto attraverso il registro "indiretto" INDF.



Va notato che il contenuto di FSR viene mantenuto fino allo spegnimento del sistema e può essere modificato solamente con una riscrittura; quindi, selezionato un banco attraverso FSR, questo resterà attivo fino ad una modifica dei bit 5 e 6.

Quindi:

```
temp0 = 0x10       ; in banco 0
temp2 = 0x30       ; in banco 2
temp3 = 0x70       ; in banco 3

; azzera il contenuto dei registri
   movlw   temp3, w   ; in banco 3
   movwf   FSR
   clrf    INDF
   movlw   temp0, w   ; in banco 0
   movwf   FSR
   clrf    INDF
   movlw   temp2, w   ; in banco 2
   movwf   FSR
   clrf    INDF
```

Questo ci consente di accedere a qualunque locazione in RAM, senza esplicite istruzioni di commutazione dei banchi, il che può essere un sensibile vantaggio in molti casi.
Inoltre, non occorre conoscere il banco in cui si trova uno specifico registro, dato che basta indicare la label relativa.

Da osservare **che la modifica di FSR agisce in senso più generale sulla selezione dei banchi; nell' esempio precedente, dopo l'ultima istruzione ci troviamo nel Bank2.**

Un'altra importante funzione che il meccanismo di indirizzamento indiretto consente, è la possibilità **di operare su blocchi di RAM, tabelle e array in RAM, usando FSR come puntatore**. Ad esempio:

```
; azzera la RAM del banco 0
Clrbank0
        movlw   0x10       ; indirizzo inizio RAM del Bank0
        movwf   FSR
clrloop clrf    INDF
        incf    FSR,f
        btfss   FSR,7      ; indirizzo > 1Fh ?
         goto   clrloop    ; no - altra locazione
        ...                ; fine loop
```

FSR viene incrementato ad ogni loop, facendo si che attraverso **INDF** si possa accedere alle successive locazioni.
Il check per la fine del loop è eseguito sul bit 5 che diventa 1 quando l' incremento fa passare il contenuto di FSR da 1Fh a 20h. Se si vuole limitare l' area da azzerare a indirizzi diversi si ricorrerà ad una comparazione con un **xorwf** o un **subwf**.

Vedremo più avanti una applicazione di quanto detto.

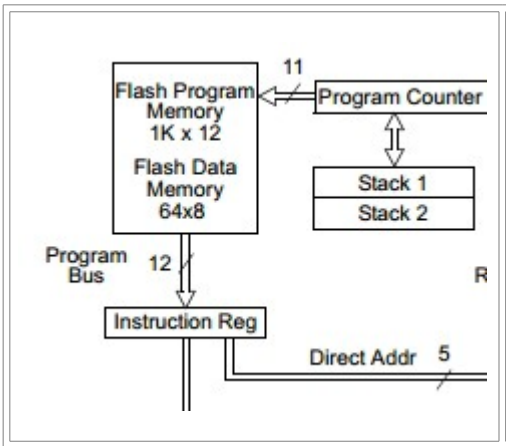
Vediamo ora qualcosa sulla EEPROM.

EEPROM

La Flash dei Baseline viene scritta durante la programmazione del chip ed è accessibile solamente come memoria programma. Questa non è accessibile come oggetto delle istruzioni che sono dirette, invece, alla memoria dati che è in RAM. Inoltre, se i meccanismi del Program Counter permetto di leggere sul bus indirizzi il contenuto della memoria programma, che intesa come elenco di istruzioni, per scrivere questa memoria occorre una tensione Vpp, più elevata della Vdd, e che si posiziona intorno a 13V nominali.
Quindi, per i Baseline, durante l' esecuzione del programma non c'è modo di alterare il contenuto della Flash (anche perchè questa contiene il programma, che non deve essere danneggiato da una qualsiasi possibile azione del programma stesso).

Però, il contenuto della memoria RAM va perduto al momento della mancanza della tensione di alimentazione, mentre quello della memoria programma resta inalterato fino ad una successiva programmazione.
In varie applicazioni sarebbe utile disporre di una memoria non volatile da usare per i dati, ad esempio in applicazioni data logger. Per questo, Microchip ha reso possibile l'inserimento di una zona di memoria non volatile, di piccole dimensioni, che può essere letta e scritta da programma come se si trattasse di memoria dati.
Il vantaggio di questa opzione è dato dalla possibilità di inserirvi dati durante l' esecuzione del programma e che non andranno persi al mancare dell' alimentazione. Questi dati saranno prontamente disponibili al riavvio e potranno essere aggiornati, modificati, ecc. Inoltre, è possibile pre caricare, in fase di programmazione del chip, il contenuto di questa memoria in modo tale da assegnare al chip una serie di dati di "default".

Anche se il nome usato comunemente per questa memoria è quello di EEPROM (***Electrically Erasable PROM***), si tratta di una indicazione non precisa, rimasta come residuo del periodo in cui una memoria ri programmabile senza estrarla da circuito era realizzata in una tecnologia differente; essa era stata sviluppata sulla base della preesistente tecnologia EPROM. Nei PIC attuali, invece, si tratta di una porzione di memoria Flash, la stessa usata per conservare la memoria programma, più esattamente indicabile come **Flash Data Memory**.



La notiamo se osserviamo il diagramma a blocchi di un PIC contenente EEPROM: vediamo che essa si trova appunto nell' area FLASH.
Questo vuol dire che **è accessibile attraverso il bus istruzioni (program Bus) e non il bus dati** (sappiamo che nei PIC, ad architettura Harvard, i due bus sono separati).

Da notare anche che, nei Baseline, il "byte" del Program Bus è ampio 12bit, ma la Flash Data Memory è ampia solo 8 bit, per poter essere trattata dal bus dati, che ha questa dimensione.

Il meccanismo implementato da Microchip per accedere alla Flash Data Memory da programma comporta una "via indiretta", necessaria a superare i due aspetti diversi che entrano a far parte di questa operazione.

Il primo è quello di rendere accessibile al bus dati una area che è di competenza del bus indirizzi.
Il secondo è quello della necessità di una tensione superiore alla Vdd per scrivere la Flash.
La soluzione di questi due problemi comporta una relativa complessità di accesso all' area EEPROM, sopratutto in scrittura.

Per quello che riguarda la tensione, non è difficile generarla internamente al chip con circuiti di charge-pump, ma l'operazione necessita comunque di un tempo molto maggiore di quello richiesto per scrivere una cella di memoria RAM, che è accessibile al ritmo del clock del processore. Inoltre, è indipendente da questo: qualsiasi sia la frequenza del clock, il tempo necessario alla scrittura della Flash Data Memory è lo stesso, dato che dipende dalle caratteristiche della Flash. Per avere un ordine di idee, mentre la RAM dati è accessibile, con il clock a 20MHZ, in un ciclo istruzione, ovvero 200ns, la Flash richiede lo steso tempo in lettura, ma ben 3-5ms per essere scritta.

Per quanto riguarda il passaggio da bus indirizzi a bus dati, si deve considerare che l' area EEPROM non è accessibile direttamente dalle istruzioni; il meccanismo adottato comporta l' uso di alcuni registri ausiliari intermedi, che hanno qualche analogia con l' indirizzamento indiretto: anche qui si usa un registro come puntatore ed un altro come tramite per il passaggio dei dati tra i due bus.

Questa complessità costruttiva (oltre al costo) mal si attaglia ai Baseline e la sua implementazione è limitata solo a pochi modelli:

PIC	EEPROM bit
12F519	64
16F526	64
16F527	64
16F570	132

In questi chip la EEPROM è da considerare alla stregua di un modulo funzione addizionale ed ha a disposizione tre registri per la sua gestione. Questi sono:

- **EECON** registro di controllo
- **EEDATA** registro scambio dati
- **EEADR** registro di indirizzamento

In sostanza, possiamo accedere alla EEPROM solamente in modo indiretto, attraverso un registro dati (**EEDATA**), un registro indirizzi (**EEADR**) ed uno di controllo (**EECON**) che permette di gestire l'operazione che si vuole eseguire.
E' come se, non potendo effettuare direttamente una commissione, ci vedessimo costretti ad utilizzare un intermediario.
E' evidente che occorre avere conoscere l' uso di questi registri dedicati. E qui sorge un problema non marginale.

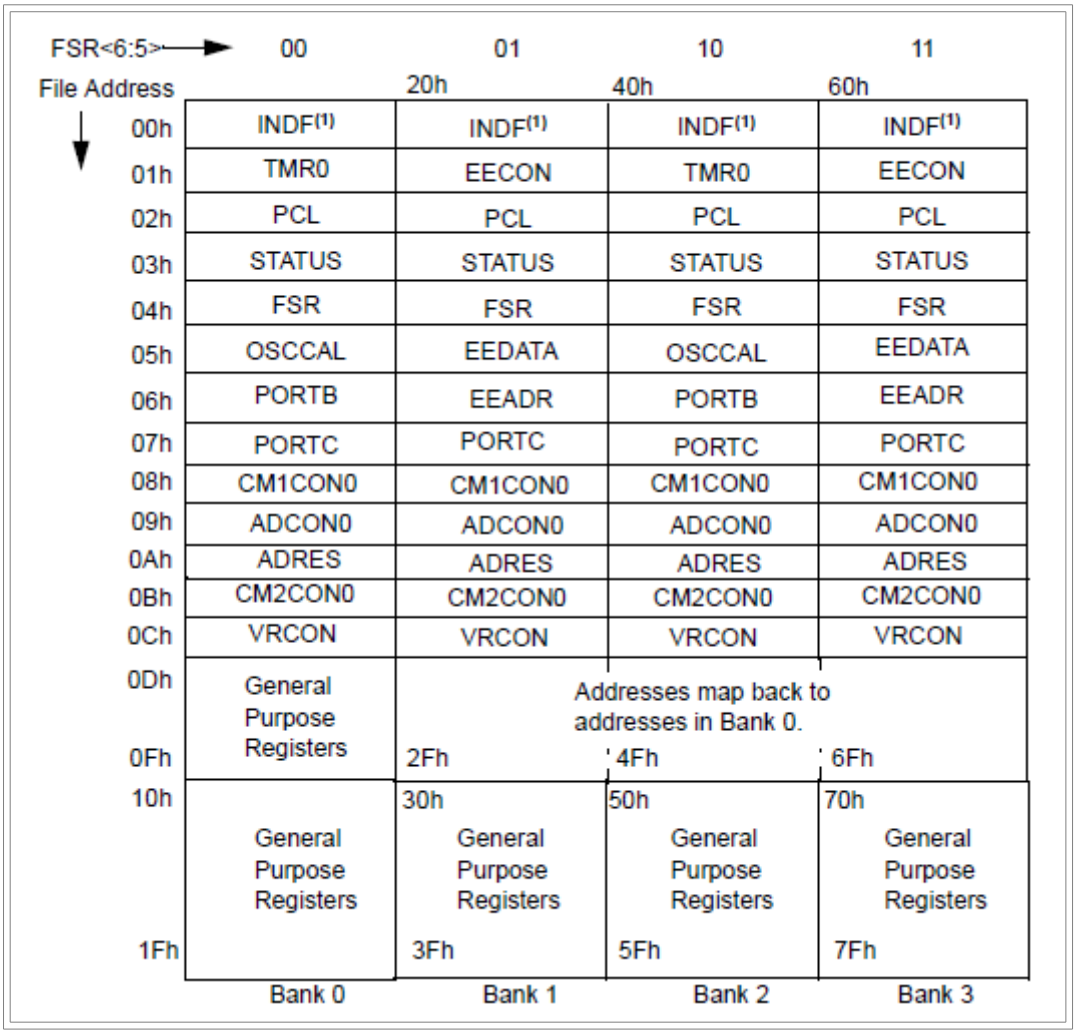
RAM dati e banchi

Vediamo di ribadire ulteriormente la questione dell' indirizzamento sui banchi, che un argomento sempre spinoso per chi si avvicina ai PIC e, assieme alla paginazione della memoria programma, l'aspetto più sgradito di questi microcontroller.

Abbiamo visto , parlando dell' indirizzamento indiretto, che l'area RAM, che contiene gli SFR e la RAM dati , a seconda del chip, può essere frazionata in più blocchi, detti banchi, e il cui accesso è regolato dall'uso di bit addizionali a causa dell' impossibilità di un opcode di contenere indirizzi superiori all' ampiezza di un banco.

Se osserviamo la mappa RAM di 16F526, che dispone di EEPROM, vediamo come gli SFR di controllo siano passati nel secondo banco, diviso come il primo in due sezione: una parte dedicata a questi registri e l' altra alle locazioni RAM dati.

Inoltre, il contenuto degli SFR dei banchi 0 e 1 è riflesso identicamente nei banchi 2 e 3 (mentre l' area RAM dati di ogni banco è indipendente).



Possiamo anche osservare come alcuni SFR fondamentali, come **INDF**, **PCL**, **STATUS** e **FSR** vengono riprodotti identicamente su più banchi, con **EECON**, **EEDATA**, **EEADR** che, invece sono alterativi rispettivamente a **TMR0**, **OSCCAL** e **PORTB**.

Questo vuol dire che occorre prestare attenzione alla situazione dei banchi, cosa che fino ad ora non abbiamo tenuto in considerazione.
Per essere più chiari:

- al POR, il default fa si che sia accessibile quanto contenuto nel Bank0

Quindi, con l' istruzione **clrf PORTB** otteniamo l' azzeramento del contenuto di questo registro.
Se siamo in banco 0 o 2, ma se siamo in banco 1 o 3 l' azione sarà **clrf EEADR**
I progettisti hanno cercato di rendere più semplici le cose e, ad esempio, 16F526 i banchi 1 e 3 sono copie di 0 e 2 per quanto riguarda gli SFR ed esiste un'area di General Pur pose Registers comune ai banchi (anche se di soli 3 bytes).
Però ogni chip ha caratteristiche differenti e la situazione potrà essere diversa da questa in esempio.
Ma non occorre conoscere a memoria la struttura del PIC che si vuole usare per sapere quali bit azionare nel cambio di banco e la posizione dei vari SFR, dato che questo si risolve semplicemente usando due elementi già visti:

- le label al posto degli assoluti
- gli pseudo opcodes dell' Assembler MPASM

La prima azione rende inutile sapere dove si trova mappato un elemento, dato che ci penserà il compilatore a sostituire le label con i valori adeguati.
La seconda azione **rende inutile la conoscenza del meccanismo di commutazione dei banchi e pure la scrittura di macro per questa funzione**, dato che esiste **banksel**. Quindi, semplicemente:

```
; accesso al registro dati della EEPROM
   banksel  EEDATA
```

Occorre, ovviamente, ricordare che, stando in un banco, se è possibile accedere comunque ai regisitri di "uso generale", per altri l'accesso è possibile solo dal proprio banco. Così:

```
; accesso al registro dati della EEPROM
   banksel  EEDATA
   movf     EEDATA,w   ; copio dato in w
   movwf    PORTB      ; questo non copia EEDATA in PORTB, ma in EEADR
```

per agire su **PORTB** devo ritornare al suo banco di appartenenza esclusiva:

```
; copia EEDATA in PORTB
   banksel  EEDATA
   movf     EEDATA,w   ; copio dato in w
   banksel  PORTB
   movwf    PORTB
```

Inoltre, come abbiamo visto all' inizio, possiamo usare l'indirizzamento indiretto:

```
; copia EEDATA in PORTB
   movlw    EEDATA     ; accesso indiretto a EEDATA
```

```
movwf   FSR
movf    INDF,w      ; copio dato in w
banksel PORTB
movwf   PORTB
```

Analizzata l'essenza della questione dei banchi, vediamo cosa occorre ancora per scrivere e leggere la EEPROM.

Lettura e scrittura della EEPROM

Il registro **EEDATA** è quello che viene utilizzato per scambiare i dati tra il bus dati e la EEPROM; in breve, è quello in cui dobbiamo mettere il byte da scrivere e da cui recuperiamo quello da leggere (analogamente a INDF).

Il registro **EEADR** ha lo scopo di comunicare alla EEPROM a quale indirizzo vogliamo accedere, ovvero in quale locazione vogliamo scrivere o leggere (analogamente a FSR).

In sintesi:

- **scrivendo un indirizzo in EEADR, il contenuto della locazione di EEPROM corrispondente diventa accessibili nel registro EEDATA**, in scrittura e lettura

Da quanto sopra, risulta anche evidente che è possibile scrivere o leggere un solo byte di EEPROM alla volta.

Il terzo registro, **EECON**, serve per comandare le operazioni sulla EEPROM.

U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	FREE	WRERR	WREN	WR	RD
bit 7							bit 0

Solamente i bit4:0 sono implementati.

```
bit 4  FREE    1  Abilita la cancellazione di una riga
               0  Abilita la scrittura

bit 3  WRERR   1  Scrittura o cancellazione non eseguite
                  (a causa di un reset)
               0  Scrittura eseguita correttamente

bit 2  WREN    1  Abilita scrittura
               0  Disabilita scrittura

bit 1  WR      1  Avvia scrittura
               0  Ciclo di scrittura completato

bit 0  RD      1  Abilita lettura
               0  Disabilita lettura
```

La procedura per **la lettura**:

1. si pone nel registro indirizzi EEADR quello della locazione che si vuole raggiungere
2. si imposta nel registro di controllo EECON l' operazione da fare
3. e si recupera nel registro dati EEDATA il valore contenuto nella locazione richiesta

In istruzioni:

```
; leggi dato in EEPROM - indirizzo in W
      banksel  EEDATA      ; seleziona banco dei registri EEPROM
      movwf    EEADR       ; metti indirizzo nel registro
      bsf      EECON,RD    ; attiva lettura
      movf     EEDATA,w    ; ritorna col dato in W
```

La lettura, quindi, è "immediata" e necessita solamente del meccanismo indiretto.
La scrittura, invece, è più complessa per due ragioni: la prima riguarda il tempo di scrittura delle celle Flash che è più lento del ciclo istruzione e che è indipendente da esso. La seconda riguarda, come accennato, la necessità di proteggere da accessi indesiderati i dati salvati in EEPROM.

Per prima cosa occorre cancellare le celle in cui si vuole scrivere. E,' però, possibile cancellare la EEPROM, ma non a byte singolo, bensì una intera riga per volta:

```
; cancella dato in EEPROM - indirizzo EE_ADR
      banksel  EEDATA      ; seleziona banco dei registri EEPROM
      movwf    EEADR       ; indirizzo nella linea da cancellare
      bsf      EECON,FREE  ; seleziona modo cancellazione
      bsf      EECON,WREN  ; abilita scrittura
      bsf      EECON,WR    ; avvia cancellazione
```

I bit 5:3 dell' indirizzo passato a **EEADR** identificano la riga da cancellare.

Importante notare che, se il bit **FREE** può essere settato da qualsiasi opcode adeguato, **bsf** è l'unico a permettere l'avvio della cancellazione.
Questo opcode costituisce la "chiave di accesso" alla scrittura della EEPROM.
Inoltre:
- se il bit **WREN** non viene settato subito dopo **FREE**, questo viene cancellato automaticamente
- se **WR** non è settato subito dopo **WREN**, questo è cancellato automaticamente.

Così **è possibile una cancellazione solamente se il programma esegue la sequenza indicata**, che non può essere variata rispetto all' esempio fornito.
Dato che la cancellazione avviene a righe di 8 bytes ciascuna, se tutti i bytes della riga contengono dati utili ed occorre modificarne solo una parte, è necessario salvare il contenuto della riga, copiandolo momentaneamente in RAM dati; si modificheranno lì i bytes necessari e si riscriverà l' intera riga:

```
; copia riga EEPROM indicata in W e cancella contenuto
      banksel  EEDATA      ; seleziona banco dei registri EEPROM
```

```
      movwf    EEADR       ; metti indirizzo nel registro
rdlp  bsf      EECON,RD    ; attiva lettura
; copia in RAM in banco 1
   movf     EEADATA,w   ;
      movwf    INDF        ;
      incf     FSR,f       ;
      incf     EEADR,f     ;
      btfss  FSR,3
   goto  rdlp
; cancella riga in EEPROM
      decf     EEADR,f     ; riporta indirizzo nella linea da cancellare
      bsf      EECON,FREE  ; seleziona modo cancellazione
      bsf      EECON,WREN  ; abilita scrittura
      bsf      EECON,WR    ; avvia cancellazione
```

Nella locazione cancellata è possibile **scrivere** un nuovo dato in questo modo:

1. si pone nel registro indirizzi quello della locazione che si vuole raggiungere
2. si pone nel registro dati il valore da scrivere
3. si imposta nel registro di controllo l' operazione da fare
4. si utilizza una "chiave" per abilitare la scrittura, chiave che difende il contenuto della EEPROM da cancellazioni o scritture indesiderate
5. e si attende la conclusione dell' operazione, che richiede un certo tempo rispetto a quello tipico di un ciclo istruzione.

In istruzioni:

```
; scrivi dato in EEPROM - dato in W, indirizzo EE_ADR
      banksel  EEADR       ; seleziona banco dei registri EEPROM
      movlw    EE_ADR      ; carica indirizzo
      movwf    EEADR
      movlw    EE_DATA     ; carica dato
      movwf    EEDATA
      bsf      EECON,WREN  ; abilita scrittura
      bsf      EECON,WR    ; avvia scrittura
```

La "chiave" di sicurezza della scrittura, nei Baseline è ancora costituita dall' opcode **`bsf`**: solo con questo è possibile attivare la scrittura. Inoltre, come sopra:
- se **WR** non è settato subito dopo **WREN**, questo è cancellato automaticamente.

Così **è possibile una scrittura solamente se il programma esegue la sequenza indicata**.
Questi "sistemi di sicurezza" sono implementati proprio allo scopo di evitare false scritture del contenuto della EEPROM nel caso in cui il programma fosse fuori controllo per un bug o un disturbo.

Dovendo riscrivere l'intera riga, si potrà usare una routine adeguata:

```
bufstart = 0x70            ; buffer in banco 3
; copia 8 byte dal buffer in RAM ad una riga EEPROM
      movlw    bufstart    ; indirizzo inizio buffer
```

```
      movwf     FSR          ; nel puntatore indiretto
      movlw     EE_ADR       ; indirizzo inizio riga
      movwf     EEADR
wrdlp movf      INDF,w       ; recupera dato dal buffer in RAM
      movwf     EEDATA       ; scrivilo nel registro EEPROM
      bsf       EECON,WREN   ; abilita scrittura
      bsf       EECON,WR     ; avvia scrittura
      incf      FSR,f        ; incrementa puntatore RAM
      incf      EEADR,f      ; incrementa indirizzo EEPROM
      btfss     FSR,3        ; 8 passi effettuati?
       goto     wrdlp        ; no - un altro loop
; fine scrittura riga
      banksel   0
```

Il buffer in RAM è stato posizionato nel banco 3, cioè nell'area che va da 70h a 7FH. I registri di controllo della EEPROM sono disponibili sia nel banco 1 che nel banco 3, quindi non occorre alcuna movimentazione degli switches dei banchi, dato che la scrittura di FSR ha portato la selezione del banco3. Al termine dell' operazione viene riattivato il banco 0.

L' area EEPROM ha la stessa tenuta dati garantita per la flash programma e può essere scritta e cancellata , ma è opportuno verificarne il contenuto dopo la scrittura:

```
; verifica dato in EEPROM - dato in W, indirizzo EE_ADR
      movf    EEDATA, W
      bsf     EECON, RD     ; attiva lettura
      xorwf   EEDATA, W     ; compara dati
      skpz                  ; se uguali, salta
       goto   WREE_ERR      ; se no gestisci l' errore
```

Come accennato prima:

Il tempo per la scrittura o cancellazione è mediamente di 3.5ms, con un massimo di 5ms (param. 43 e 44), **indipendentemente dal clock del chip**.
Una volta avviata la scrittura (o cancellazione) il Program Counter non avanza sull'istruzione successiva fino a che questo tempo è esaurito.

Per la lettura, invece, non esiste tempo di attesa.

Se l' operazione di scrittura o cancellazione viene interrotta da un reset (dovuto a MCLR o WDT), lo stato del bit **WRERR** di **EECON** va a livello 1, indicando che l'operazione non è andata a buon fine. Se il reset è causato dalla caduta della tensione e successivo riavvio (POR), il flag non è valido. Se esiste la possibilità di questi eventi, occorre all' avvio una gestione delle cause di reset, come visto nell' esercitazione 14, che tenga anche conto della EEPROM.

Righe della EEPROM?

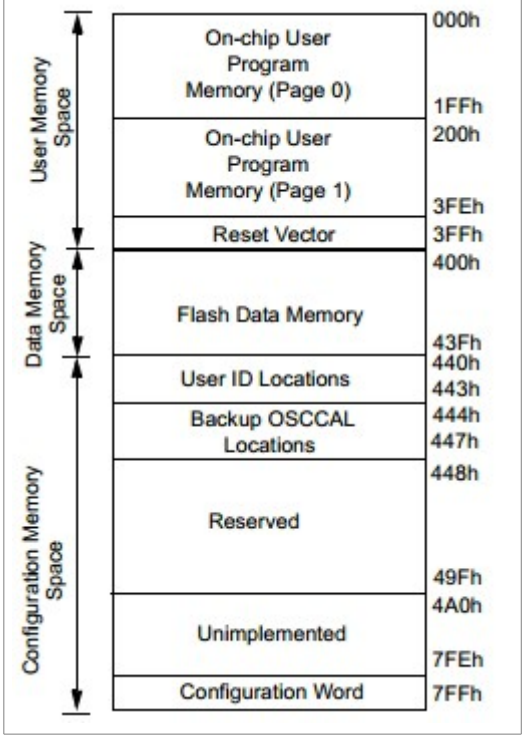
Abbiamo disinvoltamente parlato di cancellazione della EEPROM a righe. Vediamo meglio cosa significa questo.

La EEPROM è organizzata in 8 righe da 8 bytes ciascuna (8 x 8 = 64 bytes totali).
Nell' indirizzo, i bit2:0 identificano la posizione della cella nella riga, mentre i bit5:3 indicano la riga. I bit 7:6 non sono implementati. Ad esempio:

Bit di EEADR	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	
Funzione	***non impl.***		***da 000 a 111***			***da 00 a 07***			
Esempio	-	-	0	0	0	0	0	0	prima locazione della riga 0
	-	-	0	0	1	0	0	0	prima locazione della riga 1
	-	-	0	1	1	0	0	1	seconda locazione della riga 3
	-	-	1	1	1	x	x	x	riga 7
	-	-	1	1	1	1	1	1	ultima locazione della riga 7

Quindi, volendo scrivere la 64° locazione, l' indirizzo da passare a EEADR sarà 3F.
Volendo scrivere la prima locazione si dovrà passare 00.
Volendo cancellare la riga che contiene la locazione 60 occorrerà cancellare l' intera riga 7.
Volendo cancellare la prima locazione della riga 1, si cancelleranno tutte le otto locazioni della riga.

Da notare che questi sono gli **indirizzi relativi** dell'area EEPROM. La posizione assoluta in memoria è identificabile nella mappa di memoria del chip, rilevabile dal foglio dati.



Ad esempio, per 16F526, la EEPROM (*Flash Data Memory*) si trova tra 400h e 43Fh.

Questo, però, non ci interessa durante l' uso della stessa, dato che non possiamo accedervi direttamente, ma solo attraverso il meccanismo di registri visto prima.

Quindi dobbiamo considerare l' area EEPROM come un blocco di memoria che va da 00 a 3Fh, per un totale di 64 elementi, organizzati come abbiamo appena tratteggiato.

Abbiamo una relazione simbolica per gli indirizzi assoluti dell'area EEPROM nel file ***nomeprocessore.lkr***. Ad esempio, per 16F526, vi troviamo:

```
CODEPAGE NAME=flashdata START=0x400
         END=0x43F
```

Sapere dove è posizionata la EEPROM come indirizzo assoluto ci può essere utile solamente se vogliamo pre caricarla all' atto della programmazione, utilizzando la direttiva DE.

DE

La direttiva **De** (*Declare Eeprom data bytes*) consente di caricare dati in EEPROM al momento della programmazione del chip:

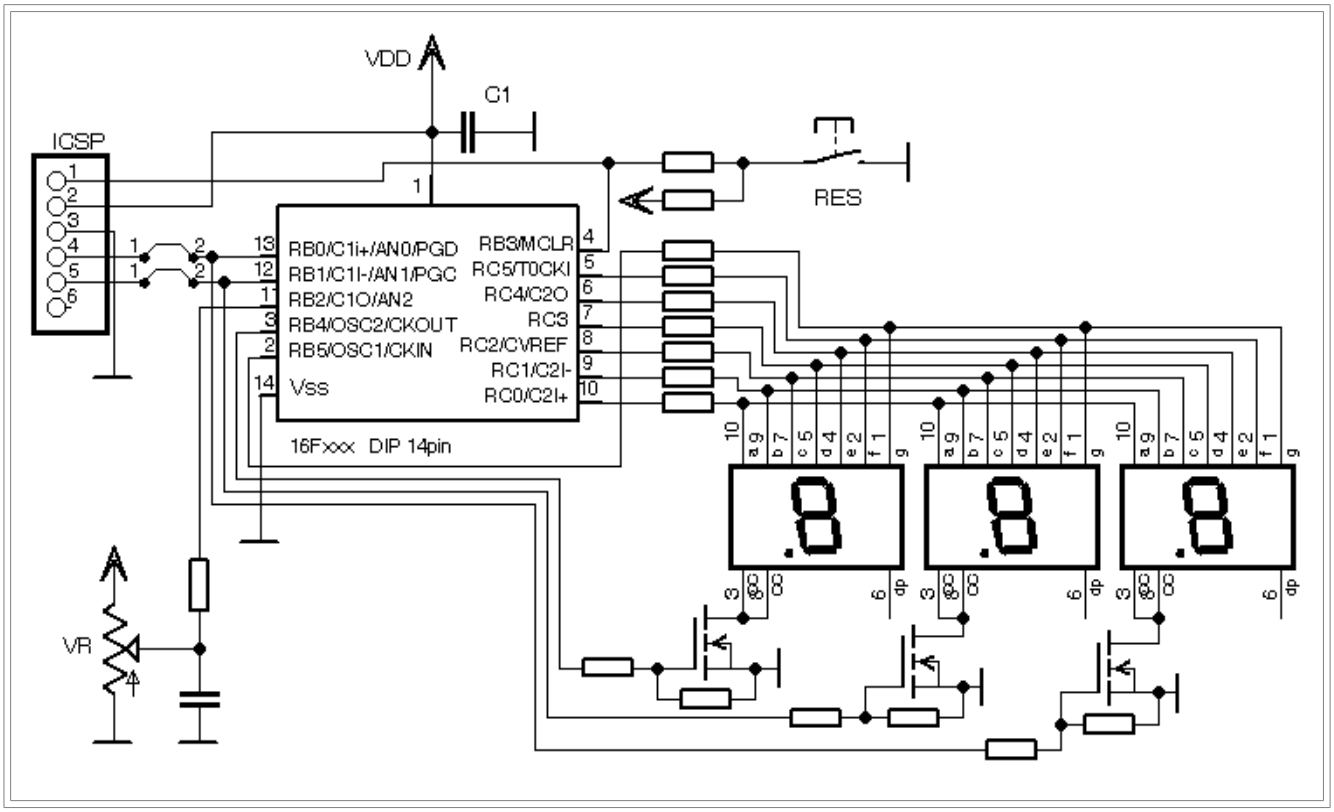
[label]	sp	de	sp	expr, expr,...

La direttiva riserva 8 bit per ogni espressione in oggetto. Ogni espressione deve rendere un valore a 8 bit.
Va preceduta dall' indicatore della posizione in memoria:

```
ORG  0x400
de   "www.microcontroller.it",0xAA,0x55,0
```

Una applicazione.

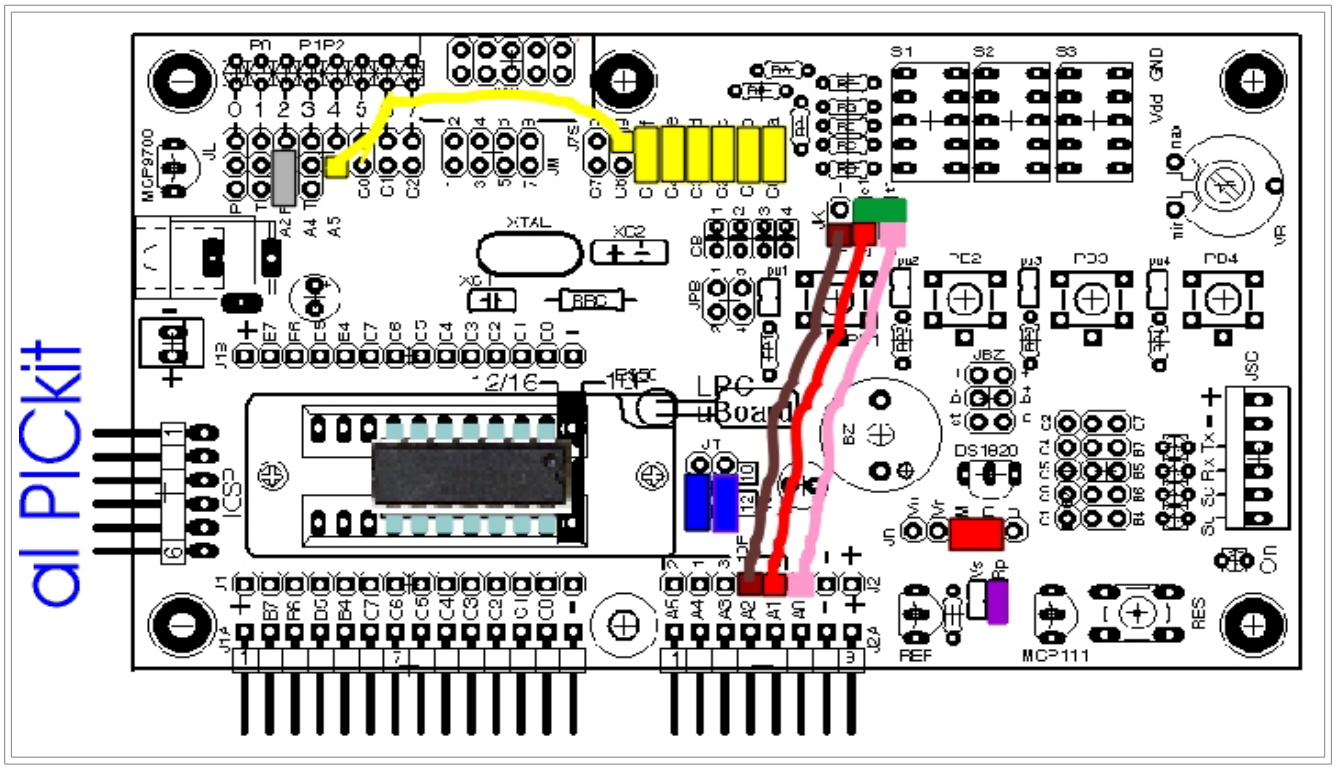
Utilizziamo l'indirizzamento indiretto e la funzione EEPROM per immagazzinare dati che vengono inseriti dall' utente.



Alla prima accensione, tre display riportano 3 cifre 0; questo valore è stato inserito durante la compilazione del sorgente e scritto nella EEPROM all' atto della programmazione del chip. Premendo il pulsante, si attiva il convertitore AD che legge la tensione sul cursore del potenziometro. Il valore risultante dalla conversione, in decimale, viene riportato sul display e

salvato in EEPROM. Alla prossima accensione sarà questo valore ad essere presentato sui display. Ruotando il potenziometro su una diversa posizione prima di premere il pulsante, varieremo il valore convertito tra 000 8min.) e 255 (max.).

Sulla **LPCuB** il circuito è realizzato con i soliti jumper:



I jumper "gialli" collegano i segmenti
I jumper volanti "rosa", "rosso" e "marrone" collegano i gate dei MOSFET.
Il jumper "grigio" collega il cursore del potenziometro.

Durante la programmazione ICSP i jumper "rosso" e "rosa" vanno scollegati, dato che il carico costituito dai MOSFET può rendere difficoltosa la programmazione.

Il programma.

Utilizziamo buona parte di quanto abbiamo già scritto per le esercitazioni precedenti.

Per prima cosa, occorrerà impostare il generatore di tempo per la cadenza del multiplex, che realizziamo come già visto, ricorrendo al Timer0.

All' avvio, il programma legge il contenuto della EEPROM e lo copia in RAM:

```
; copia EEPROM, riga 0 , nel buffer in RAM
         pagesel RdEepRow
         call    RdEepRow
```

Viene letta l'intera prima riga di 8 bytes, anche se poi sarà modificato solamente la prima locazione. Il buffer viene impostato nella RAM in banco3; questo consente di effettuare le operazioni sui registri della EEPROM e sul buffer senza cambiare banco: se osserviamo la mappa di memoria RAM del 16F526, notiamo che i registri di comando della Flash dati sono accessibili in banco 1 e 3 e negli stessi sono disponibili 16bytes di RAM dati:

```
; copia riga0 da EEPROM a RAM
RdEepRow
     movlw rambuf   ; puntatore indiretto per inizio
     movwf FSR      ; del buffer in RAM e il banco per la EEPROM
     movlw eeprow0  ; seleziona primo byte della riga 0
     movwf EEADR    ;
rdlp bsf   EECON,RD ; attiva lettura
; copia in RAM
     movf  EEDATA,w ; leggi EEPROM
     movwf INDF     ; copia indiretto nel buffer
     incf  FSR,f    ; incrementa pointer del buffer
     incf  EEADR,f  ; incrementa indirizzo EEPROM
     btfss FSR,3    ; 8 bytes?
      goto rdlp     ; no - ancora un altro
     retlw 0        ; si - fine
```

Usiamo l'indirizzamento indiretto per puntare il buffer in RAM; dato che questo si trova in banco3, contemporaneamente si ha anche il giusto accesso ai registri della EEPROM.
I dati sono copiati da **EEDATA** al buffer, incrementando ad ogni loop sia il puntatore indiretto (**FSR**) che l' indirizzo della EEPROM (in **EEADR**). Verificando lo stato del bit 3 interrompiamo l' operazione dopo il trasferimento di 8 bytes.

Il dato nella locazione 0 della EEPROM viene convertito da esadecimale a 3 byte il cui MSB è a 0 e l'LSB corrisponde al valore decimale; si tratta di un BCD non compresso. Questa routine è gia stata trattata nell' esercitazione riguardante la matematica.
Ad esempio, se il byte vale **AAh**, ovvero 170 decimale, la conversione a BCD produce questo risultato:

bin = unità = 00 bcdL= decine = 07 bcdH = centinaia = 01

Questi valori sono passati uno alla volta attraverso la solita lookup table per comandare i segmenti dei display:

```
; segment data table - display catodo comune
segtbl ;andlw 0x0F        ; solo nibble basso
       addwf PCL,f       ; punta PC
       retlw b'00111111' ; "0" -|-|F|E|D|C|B|A
       retlw b'00000110' ; "1" -|-|-|-|-|C|B|-
       retlw b'01011011' ; "2" -|G|-|E|D|-|B|A
       retlw b'01001111' ; "3" -|G|-|-|D|C|B|A
```

```
       retlw b'01100110' ; "4" -|G|F|-|-|C|B|-
       retlw b'01101101' ; "5" -|G|F|-|D|C|-|A
       retlw b'01111101' ; "6" -|G|F|E|D|C|-|A
       retlw b'00000111' ; "7" -|-|-|-|-|C|B|A
       retlw b'01111111' ; "8" -|G|F|E|D|C|B|A
       retlw b'01101111' ; "9" -|G|F|-|D|C|B|A
       ;retlw b'01110111' ; "A" -|G|F|E|-|C|B|A
       ;retlw b'01111100' ; "b" -|G|F|E|D|C|-|-
       ;retlw b'00111001' ; "C" -|-|F|E|D|-|-|A
       ;retlw b'01011110' ; "d" -|G|-|E|D|C|B|-
       ;retlw b'01111001' ; "E" -|G|F|E|D|-|-|A
       ;retlw b'01110001' ; "F" -|G|F|E|-|-|-|A
```

Osserviamo che la tabella è ridotta ai soli valori tra 0 e 9, dato che trattiamo solo cifre BCD; non serve neppure l'and iniziale dato che i valori ottenuti dalla conversione HEX->BCD hanno già i 4 bit alti azzerati.

Alla fine della scansione dei tre digit inseriamo la verifica del pulsante:

```
        btfss btn    ; pulsante aperto?
        goto btnchk ; no - chiuso
; si - pulsante aperto - verifica stato precedente
       btfss btnflg ; prima era aperto?
       bsf   btnflg ; no - aggiorna flag
; prima era chiuso
       goto  displp ; done

; pulsante chiuso - verifica stato precedente
btnchk btfss btnflg ; prima era aperto ?
        goto displp ; no - done
; prima era aperto
       bcf   btnflg ; aggiorna flag
```

Alla pressione del pulsante, viene confrontata la situazione precedente e si passa alla fase successiva solo se si è verificata una transizione da aperto a chiuso. In caso contrario si ritorna al loop dei display.
Il tempo di scansione del multiplex viene usato come tempo di debounce per il pulsante.

Se questo è premuto, si avvia la conversione AD e il dato risultante in ADRES viene scritto in EEPROM, usando il puntatore indiretto, che non richiede un cambio di banco esplicito:

```
; pulsante premuto - avvia conversione AD
     pagesel ADConv
     call    ADConv
     movlw   rambuf     ; salva risultato
     movwf   FSR
     movf    ADRES,w
     movwf   INDF
```

Subito dopo, si opera la cancellazione della riga 0 della EEPROM e la riscrittura del buffer dalla RAM.
Il loop si chiude sulla conversione del dato esadecimale per il display.

Quindi, premendo il pulsante, il valore di tensione presente sul cursore del potenziometro diventerà dato digitale e sarà salvato in EEPROM, oltre che immediatamente visualizato.

Alla prima accensione del circuito, il display presenterà il valore pre programmato nella Flash dati attraverso lo statement **de** , ovvero 0:

```
EEPROMdata CODE 0x400
 de 0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07
```

Dopo la pressione del pulsante, il valore salvato sarà quello risultante dalla conversione AD; spegnendo il circuito e accendendolo, questo sarà il nuovo valore rappresentato sui display.

L'uso dei **banksel** è limitato dall' indirizzamento indiretto, dalla scelta del buffer RAM nella stessa pagina dei registri di accesso alla EEPROM e dall' avere dichiarato alcune variabili nell' area di RAM condivisa tra i banchi:

```
SHR1 UDATA_SHR ; RAM condivisa - 3 bytes
; necessari per la conversione
bin  res 1     ; binario e unità
bcdL res 1     ; decine
bcdH res 1     ; centinaia
```

Purtroppo si tratta di soli 3 bytes; occorrendo ulteriore RAM viene dichiarata come generica e va a sovrapporsi a questi primi bytes, ma, dato che viene usata in momenti diversi del programma, non si crea alcun conflitto.
Inseriamo **banksel** solo dove necessari, ad esempio:

```
; converte in BCD
convert movlw   rambuf
        movwf   FSR
        movf    INDF,w
        pagesel Hex1Bcd3
        call    Hex1Bcd3
        pagesel $
        banksel 0
        clrf    TMR0      ; inizializza timer
```

dato che il banco è stabilito su 3 dall' **FSR** e il registro **TMR0** è disponibile solo sul banco 0 o 2.

Una del genere può essere facilmente modificata per raccogliere un numero maggiore di dati, nei limiti della capacità della Flash dati e realizzare così una funzione di data logger. Se occorre conservare una maggiore quantità di dati si dovrà ricorrere ad una EEPROM esterna.

I messaggi della compilazione

Andando a verificare il messaggio della finestra di Output del compilatore, troviamo elencati numerosi messaggi del genere:

```
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
```

Questi messaggi sono relativi al fatto che stiamo usando registri SFR che non sono accessibili in pagina 0: l' Assembler genera questi messaggi per avvisare l' utente di prestare attenzione.
Li troviamo nel file ***.lst*** in relazione alle righe in cui appaiono questi registri:

```
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
002C   0026           00224       movwf    EEADR       ;
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
002D   0501           00225 rdlp  bsf      EECON,RD    ; attiva lettura
                      00226 ; copia in RAM
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
002E   0205           00227       movf     EEDATA,w    ; leggi EEPROM
002F   0020           00228       movwf    INDF        ; copia indiretto nel buffer
0030   02A4           00229       incf     FSR,f       ; incrementa pointer del buffer
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0031   02A6           00230       incf     EEADR,f     ; incrementa indirizzo EEPROM
```

I **`Message`**, come abbiamo detto in altre esercitazioni, non impediscono la conclusione della compilazione e la generazione del file ***.hex***, ma **servono ad attrarre l' attenzione del programmatore su punti che potrebbero rende inutilizzabile il file eseguibile.**

Come detto, in prima compilazione, è decisamente sconsigliabile di abolire questa segnalazione. Solo ed esclusivamente dopo aver controllato la correttezza di quanto scritto ed essere sicuri della giusta gestione dei banchi, potremo, volendo, aggiungere in testa al sorgente la famigerata riga:

```
   errorlevel -302
```

Questo elimina i messaggi `[302]` dal listato.

Evitate di inserire questa riga fino a quando non avete compreso a fondo il meccanismo dei banchi e comunque mai durante le prime compilazioni del sorgente.

Solo dopo aver eliminato ogni altro errore ed aver verificato la posizione di registri e banchi si potrà abolire la segnalazione, al solo scopo pratico di avere un file listato "pulito".

Nel sorgente dell' esempio, la riga è già presente, ma come commento. Per attivarla, basta decommentarla. Si consiglia di procedere come segue:

- compilate il sorgente con la linea commentata e verificate sia il messaggio nella finestra di ***Out***, sia il file ***.lst***
- decommentate la linea e ripetete la compilazione, verificando come prima la finestra di ***Ou***t e il file listato

La comparazione tra le due letture renderà chiara l' azione della line **`errorlevel -`**

15A_526.asm

```
;*****************************************************************
; 15A_526.asm
;-----------------------------------------------------------------
;
;    Titolo        : Corso Assembly & C - Esercitazione 15A
;                    Risultato ADC in hex su tre cifre e
;                    salvataggio in EEPROM
;    PIC           : 16F526
;    Supporto      : MPASM
;    Versione      : V.519-1.0
;    Data          : 01-05-2013
;    Ref. hardware :
;    Autore        :  afg
;
;-----------------------------------------------------------------
;
;  Impiego pin :
;  ----------------
;       16F506/526 @ 14 pin
;
;                   |‾‾\/‾‾|
;             Vdd -|1   14|- Vss
;             RB5 -|2   13|- RB0
;             RB4 -|3   12|- RB1
;        RB3/MCLR -|4   11|- RB22
;             RC5 -|5   10|- RC0
;             RC4 -|6    9|- RC1
;             RC3 -|7    8|- RC2
;                  |______|
;
;    Vdd                     1: ++
;    RB5/OSC1/CLKIN          2: Out  segm g
;    RB4/OSC2/CLKOUT         3: Out  gate centinaia
;    RB3/!MCLR/VPP           4: In   pulsante
;    RC5/T0CKI               5: Out  segm f
;    RC4/C2OUT               6: Out  segm e
;    RC3                     7: Out  segm d
;    RC2/CVref               8: Out  segm c
;    RC1/C2IN-               9: Out  segm b
;    RC0/C2IN+              10: Out  segm a
;    RB2/C1OUT/AN2          11: AN2  potenziometro
;    RB1/C1IN-/AN1/ICSPC    12: Out  gate decine
;    RB0/C1IN+/AN0/ICSPD    13: Out  gate unità
;    Vss                    14: --
;
;#################################################################
; scelta del processore
         LIST      p=16F526
         #include <p16F526.inc>

        radix     dec

        ;errorlevel -302
```

```
;#####################################################################
;                          CONFIGURAZIONE
; Oscillatore interno, 4MHz, no WDT, no CP, MCLR
 __config  _IntRC_OSC_RB4 & _IOSCFS_4MHz & _WDTE_OFF & _CP_OFF &
_CPDF_OFF & _MCLRE_OFF

;#####################################################################
;                           MEMORIA RAM
; general purpose RAM
SHR1  UDATA_SHR ;RAM condivisa - 3 bytes
; necessari per la conversione
bin    res 1                ; unità
bcdL   res 1                ; decine
bcdH   res 1                ; centinaia

GRAM      UDATA
temp   res 1                ; temporaneo
flags  res 1                ; flag

#define  btnflg  flags,7    ; flag pulsante premuto

; pre carica riga0 in EEPROM
EEPROMdata    CODE 0x400
 de 0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07

eeprow0 equ 0               ; prima locazione EEPROM

 ;*******************************************************************
;===================================================================
;            DEFINIZIONE DI IMPIEGO DEI PORT
;
;PORTC map
;|  5  |  4  |  3  |  2  |  1  |  0  |
;|-----|-----|-----|-----|-----|-----|
;|segmf|segme|segmd|segmc|segmb|segma|
;
#define     segma  PORTC,0    ; segmenti f:a
#define     segmb  PORTC,1    ;
#define     segmc  PORTC,2    ;
#define     segmd  PORTC,3    ;
#define     segme  PORTC,4    ;
#define     segmf  PORTC,5    ;

;PORTB map
;|  5  |  4  |  3  |  2  |  1  |  0  |
;|-----|-----|-----|-----|-----|-----|
;|segmg|gate3| byn | AN2 |gate1|gate2|
;
#define     gate2  PORTB,0    ; gate MOSFET cifra unità
#define     gate1  PORTB,1    ; gate MOSFET cifra decine
;#define           PORTB,2    ; AN2
#define     btn    PORTB,3    ; pulsante RES
#define     gate3  PORTB,4    ; gate MOSFET cifra centinaia
#define     segmg  PORTB,5    ; segmento g
;
;#####################################################################
;                          COSTANTI
```

```
;
#define  bit05k  TMR0,1 ; bit TMR0 per 512us
#define  bit1k   TMR0,2 ;             1024us
#define  bit2k   TMR0,3 ;             2048us
#define  bit4k   TMR0,4 ;             4096us
#define  bit8k   TMR0,5 ;             8192us

#define  bittmr   bit4k ; 4096us

rambuf = 0x70           ; ram buffer in banco 3

;*****************************************************************
;                          MACRO LOCALI
; comando gate MOSFET
UNITon   MACRO
        bsf gate2
          ENDM
UNIToff  MACRO
        bcf gate2
          ENDM
DECon   MACRO
        bsf gate1
          ENDM
DECoff  MACRO
        bcf gate1
          ENDM
CENTon   MACRO
        bsf gate3
          ENDM
CENToff  MACRO
        bcf gate3
          ENDM

; include set macro base
 #include C:\PIC\Library\Baseline\basemacro.asm

;*****************************************************************
;                          RESET ENTRY
;
; Reset Vector
RESVEC    CODE     0x00

Main:
; calibrazione oscillatore interno
        movwf   OSCCAL
        pagesel Init
        goto    Init

;*****************************************************************
;             TABELLE E SUBROUTINES IN PAGINA 0

; segment data table - display catodo comune
segtbl  andlw  0x0F          ; solo nibble basso
        addwf  PCL,f         ; punta PC
        retlw  b'00111111'   ; "0" -|-|F|E|D|C|B|A
        retlw  b'00000110'   ; "1" -|-|-|-|-|C|B|-
        retlw  b'01011011'   ; "2" -|G|-|E|D|-|B|A
```

```
        retlw  b'01001111'   ; "3" -|G|-|-|D|C|B|A
        retlw  b'01100110'   ; "4" -|G|F|-|-|C|B|-
        retlw  b'01101101'   ; "5" -|G|F|-|D|C|-|A
        retlw  b'01111101'   ; "6" -|G|F|E|D|C|-|A
        retlw  b'00000111'   ; "7" -|-|-|-|-|C|B|A
        retlw  b'01111111'   ; "8" -|G|F|E|D|C|B|A
        retlw  b'01101111'   ; "9" -|G|F|-|D|C|B|A
        ;retlw  b'01110111'   ; "A" -|G|F|E|-|C|B|A
        ;retlw  b'01111100'   ; "b" -|G|F|E|D|C|-|-
        ;retlw  b'00111001'   ; "C" -|-|F|E|D|-|-|A
        ;retlw  b'01011110'   ; "d" -|G|-|E|D|C|B|-
        ;retlw  b'01111001'   ; "E" -|G|F|E|D|-|-|A
        ;retlw  b'01110001'   ; "F" -|G|F|E|-|-|-|A

;--------------
; Misura tensione
ADConv:
; attesa
        DLY10US
; avvia conversione
        bsf     ADCON0,GO
adlp    btfsc   ADCON0,NOT_DONE
         goto   adlp
        retlw   0

;--------------
; Conversione Hex -> BCD unpacked
; Ingresso numero hex in W
; Uscita su bcdH (centinaia), bcdL (decine) e bin (unità)
; con MSB a 0
Hex1Bcd3
        movwf   bin
        clrf    bcdL
        clrf    bcdH
hcnt    movlw   .100
        subwf   bin,W
        skpc
         goto   hdec
        incf    bcdH,F
        movwf   bin
        goto    hcnt
hdec    movlw   .10
        subwf   bin,W
        skpc
         retlw  0          ; ritorno
        movwf   bin
        incf    bcdL,F
        goto    hdec

;--------------
; copia riga0 da EEPROM a RAM
RdEepRow
      movlw    rambuf      ; puntatore indiretto per inizio
      movwf    FSR         ; del buffer in RAM
      ;banksel  EEDATA      ; seleziona banco dei registri EEPROM
      movlw    eeprow0     ; seleziona primo byte della riga 0
      movwf    EEADR       ;
```

```
rdlp  bsf      EECON,RD    ; attiva lettura
; copia in RAM
      movf     EEDATA,w    ; leggi EEPROM
      movwf    INDF        ; copia indiretto nel buffer
      incf     FSR,f       ; incrementa pointer del buffer
      incf     EEADR,f     ; incrementa indirizzo EEPROM
      btfss    FSR,3       ; 8 bytes?
       goto    rdlp        ; no - ancora un altro
      retlw    0           ; si - fine

;--------------
; cancella riga0 in EEPROM
ClEepRow0
      banksel  EEDATA
      movlw    eeprow0     ; seleziona riga 0
      movwf    EEADR
      bsf      EECON,FREE  ; modo cancellazione
      bsf      EECON,WREN  ; abilita scrittura
      bsf      EECON,WR    ; avvia cancellazione
      retlw    0           ; fine

;--------------
; copia 8 byte dal buffer in RAM ad una riga EEPROM
WrEeprow
      movlw    rambuf      ; indirizzo inizio buffer
      movwf    FSR         ; nel puntatore indiretto
      movlw    eeprow0     ; indirizzo inizio riga
      movwf    EEADR
wrdlp movf     INDF,w      ; recupera dato dal buffer in RAM
      movwf    EEDATA      ; scrivilo nel registro EEPROM
      bsf      EECON,WREN  ; abilita scrittura
      bsf      EECON,WR    ; avvia scrittura
      incf     FSR,f       ; incrementa puntatore RAM
      incf     EEADR,f     ; incrementa indirizzo EEPROM
      btfss    FSR,3       ; 8 passi effettuati?
       goto    wrdlp       ; no - un altro loop
      retlw    0           ; fine scrittura riga



;#################################################################
;                        MAIN PROGRAM
Init:
; disabilita comparatori per liberare la funzione digitale
        bcf     CM1CON0, C1ON
        bcf     CM2CON0, C2ON

; disabilita T0CKI da RB5, prescaler 1:256 al Timer0
         ;      b'11111111'
         ;        1-------     GPWU abilitato
         ;        -1------     GPPU abilitato
         ;        --0-----     clock interno
         ;        ---1----     falling
         ;        ----0---     prescaler al Timer0
         ;        -----111     1:256
        movlw   b'11010111'
        OPTION
```

```
        clrf    PORTB     ; clear latch del port
        clrf    PORTC
        bsf     btnflg    ; preset flag

; Tutti i port utili come out
        movlw   0
        tris    PORTB
        tris    PORTC

; configura ADC per AN2, INTOSC/4 e abilita
        movlw    b'01111011'
        movwf    ADCON0

;-----------------------------------------------------------------
; copia EEPROM, riga 0 , nel buffer in RAM
        pagesel  RdEepRow
        call     RdEepRow

; converte in BCD
convert movlw    rambuf
        movwf    FSR
        movf     INDF,w
        pagesel  Hex1Bcd3
        call     Hex1Bcd3
        pagesel  $
        banksel  0
        clrf     TMR0      ; inizializza timer

;-----------------------------------------------------------------
; ciclo display
; digit unità
displp  btfss   bittmr     ; 4ms ?
         goto   displp     ; no - attesa
        movf    bin,w      ; si - carica unità
        call    segtbl     ; lookup table
        banksel temp
        movwf   temp       ; salva in temporaneo
        movwf   PORTC      ; scrivi sul port segm f:a
        btfsc   temp,6     ; comanda segmento g
         bsf    segmg
        btfss   temp,6
         bcf    segmg
        UNITon             ; cifra unità accesa
dslp_1  btfsc   bittmr     ; 4ms ?
         goto   dslp_1     ; no - attesa
dslp_2  UNIToff            ; si - spegni unità

; digit decine
        movf    bcdL,w     ; carica decine
        call    segtbl
        movwf   temp
        movwf   PORTC
        btfsc   temp,6
         bsf     segmg
        btfss   temp,6
         bcf     segmg
```

```
        DECon               ; accendi decine
dslp_3  btfss   bittmr      ; 4ms ?
         goto   dslp_3      ; no - attesa
        DECoff              ; si - spegni decine

; digit centinaia
        movf    bcdH,w
        call    segtbl
        movwf   temp
        movwf   PORTC
        btfsc   temp,6
         bsf     segmg
        btfss   temp,6
         bcf     segmg
        CENTon              ; accendi centinaia
dslp_4  btfsc   bittmr      ; 4ms ?
         goto   dslp_4      ; no - attesa
        CENToff             ; si - spegni centinaia

        btfss   btn         ; pulsante aperto?
         goto   btnchk      ; no - chiuso
; si - pulsante aperto - verifica stato precedente
        btfss   btnflg      ; prima era aperto?
         bsf    btnflg      ; no - aggiorna flag
; prima era chiuso
        goto    displp      ; done

; pulsante chiuso - verifica stato precedente
btnchk  btfss   btnflg      ; prima era aperto ?
         goto   displp      ; no - done
; prima era aperto
        bcf     btnflg      ; aggiorna flag

; pulsante premuto - avvia converisione AD
        pagesel ADConv
        call    ADConv
        movlw   rambuf      ; salva risultato
        movwf   FSR
        movf    ADRES,w
        movwf   INDF

; cancella riga 0 in EEPROM
        pagesel ClEepRow0
        call    ClEepRow0
; scrivi ram buffer in EEPROM
        pagesel WrEeprow
        call    WrEeprow

        goto    convert     ; loop

;*****************************************************************
;                            THE END
        END
```