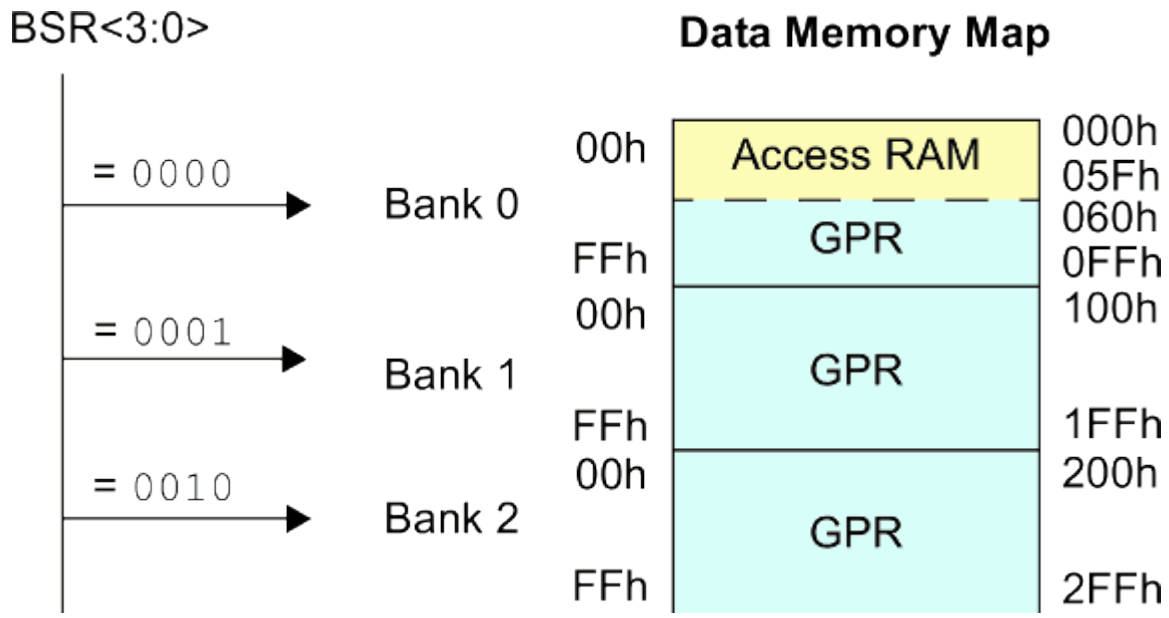
La memoria RAM del PIC18



I PIC18 contengono diversi tipi di memorie con caratteristiche nettamente diverse. In questa pagina si farà riferimento ai cosiddetti *File Registers*, che costituiscono la memoria RAM principale del PIC18. La dimensione, a seconda dei modelli, varia fino ad un massimo di 4 096 byte ed è suddivisa in 16 *banchi* (blocchi da 256 byte ciascuno). Essa include due tipi di registri:

- Special Function Registers (**SFR**), che, come dice il nome, hanno usi speciali, in genere associati a specifiche funzioni hardware. In questa pagina non verranno, salvo tre eccezioni, discussi gli SFR. Complessivamente gli SFR sono 160
- General Purpose Registers (**GPR**), utilizzati per contenere *variabili* generiche, a completa disposizione del programmatore. Il numero dei GPR cambia con la versione del PIC18: per esempio nel PIC18F45K22 sono circa 1500.

L'accesso ai vari registri avviene secondo due modalità, meglio illustrate nella prosecuzione di questa pagina:

- Accesso diretto ai registri (**Access Bank**). Questo metodo può essere utilizzato solo per gli SFR ed i primi 96 GPR (*Access RAM*). Nella figura di apertura i 96 registri della *Access RAM* sono evidenziati in giallo
- Accesso in due passaggi (**Bank Select Register**). Questo metodo permette di utilizzare tutti i registri, in due passaggi:

- Prima occorre selezionare in quale banco, tra i 16 possibili, si trova il registro. Questa operazione viene fatta usando l'apposito registro con funzioni speciali **BSR.**
- Successivamente occorre selezionare il singolo registro tra i 256 contenuti in ciascun banco

Quasi tutte le istruzioni possono utilizzare entrambi questi metodi di accesso. Il programmatore sceglierà il primo metodo per le variabili usate più frequentemente, il secondo per le variabili di dimensioni più ampie.

Un esempio

L'istruzione `movwf` (**MOV**e **W** register to **F**ile register) sposta il dato a 8 bit contenuto del registro `W` in una qualunque cella di memoria del *File Register*. Leggiamo nel foglio tecnico la descrizione:

MOVWF	**Move W to f**
Syntax:	MOVWF f {,a}
Operands:	$0 \le f \le 255$ $a \in [0,1]$
Operation:	$(W) \rightarrow f$
Status Affected:	None
Encoding:	0110 | 111a | ffff | ffff
Description:	Move data from W to register 'f'. Location 'f' can be anywhere in the 256-byte bank. If 'a' is '0', the Access Bank is selected. If 'a' is '1', the BSR is used to select the GPR bank.

La cella di memoria `f` è identificata da un *indirizzo* compreso tra 0 e 255 (0000 0000 e 1111 1111 in binario, 0x00 e 0xFF in esadecimale). Si noti l'uso delle parentesi `()` con il significato di *numero contenuto in*.

Il secondo argomento dell'istruzione `{,a}` specifica se deve essere usato l'Access Bank (quando a=0) oppure un altro banco (quando a=1), scelto attraverso il registro `BSR`. Questo argomento è facoltativo, come indicato dall'uso delle parentesi graffe `{}` .

Uso dell'Access Bank

Le celle in cui è più facile scrivere o leggere sono le prime 96, con indirizzo da `0` a `0x5F`. Gli esempi e gli esercizi di questo paragrafo faranno riferimento a questa modalità, spesso indicata come indirizzamento *diretto* nell'Access Bank o semblicemente *Access*.

Osserviamo il seguente spezzone di programma:

```
movlw 0x55
movwf 0x10
```

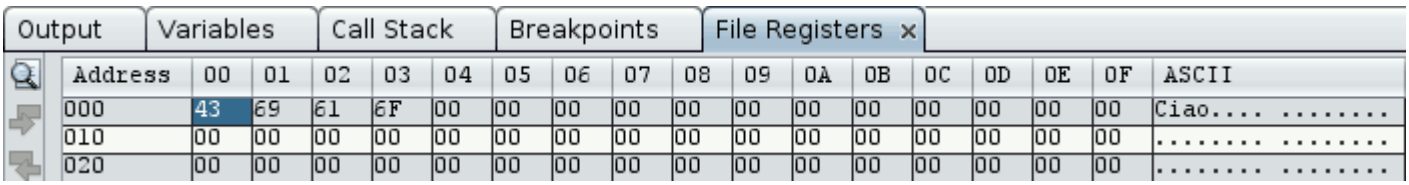
La seconda istruzione copia nel registro con indirizzo `0x10` il contenuto del registro `W` (cioè, nell'esempio, il numero `0x55`). Si noti che è presente il solo indirizzo in cui scrivere, cioè dopo lo `0x10` non vi è nulla. Come sintassi alternativa (e un po' ridondante) si sarebbe potuto scrivere anche `movwf 0x10,0` con lo stesso significato.

Esempio

Scrivere un programma che memoriazza nelle celle di memoria da 0 a 3 i quattro codici ASCII "Ciao". A volte questo modo di rappresentare parole o anche intere frasi viene indicato con il termine *stringa*.

Nota: per memorizzare nel registro W il codice ASCII del carattere "A" (=0x41) è possibile usare indifferentemente una delle due seguenti forme, identiche negli effetti:

```
movlw 0x41
movlw "A"
```



L'uso dei banchi

Per utilizzare tutte le celle di memoria occorre utilizzare l'indirizzamento *diretto* di tipo *Banked* ed il registro speciale `BSR`. Questo registro di quattro bit contiene il numero del banco che un'istruzione deve utilizzare; in pratica è la prima cifra esadecimale dell'indirizzo. Per esempio il registro `0x56` del banco `0x3` è indicata come registro `0x356`; il registro BSR contiene 3.

Per indicare se verrà utilizzata la modalità *Banked* oppure *Access*, ciascuna istruzione utilizza una virgola seguita da 0 oppure da 1 dopo l'operando (nota 4):

- `,0` (spesso sottointeso) indica l'uso dell'Access Bank, come negli esempi precedenti
- `,1` indica l'uso del registro `BSR` per identificare uno dei 16 banchi

Esempio:

- l'istruzione `movwf 0x10,0` (o anche `movwf 0x10`) copia il registro `W` nella cella dell'`Access Bank` con indirizzo `0x10` (cioè con indirizzo `0x`**`0`**`10`, infatti l'Access Bank è contenuto nel banco 0, come si vede nella figura di apertura di questa pagina)
- l'istruzione `movwf 0x10,1` copia il registro W nella cella il cui indirizzo è ottenuto anteponendo a `0x10` il contenuto del BSR (esempio: se `BSR=3` viene usata la cella con indirizzo `0x`**`3`**`10`)

La scrittura dei quattro bit nel `BSR` deve essere effettuata con l'apposita istruzione assembly `movlb` (**MOV**e **L**iteral to low nibble in **B**sr).

Usando il simulatore, il banco che verrà utilizzato è mostrato in alto a destra, a fianco del `WREG`, del `PC` e dei `flag`.

Gli Special Function Registers

Gli SFR sono registri con usi speciali. Tra i principali possiamo individuare tre registri già visti:

Working Register (WREG)

> Questo registro è utilizzato per molte istruzioni per contenere risultati e/o operandi. Può essere per molti versi paragonato a quello che in altre architetture è indicato come *Accumulatore*. Come quasi tutti i registri è formato da 8 bit

Status register (STATUS)

> Questo registro contiene i *flag* della ALU, cioè alcuni bit che hanno un particolare significato dopo l'esecuzione di un'istruzione.

Bank Select Register (BSR)

> Viene utilizzato per selezionare uno dei banchi del *File Register*

I tre registri `WREG`, `STATUS` e `BSR` descrivono lo *stato* del processore e possono essere considerati i registri fondamentali del PIC18

Tutti i registri speciali sono sempre direttamente accessibili al codice (è come appartenessero all'Access Bank) e di fatto costituiscono l'ultima parte della memoria RAM.

Dichiarazione RAM in Assembly

EQU

Un modo molto semplice è quello di dichiarare un "blocco" di RAM da usare come equivalenza: sappiamo che la RAM disponibile in quel dato chip parte da uno specifico indirizzo; quindi, con la direttiva **EQU** (equate, =) assegniamo le label a successivi indirizzi di memoria.

```
;=                  MEMORIA RAM                         =
;================================================================
d1   EQU   0x07        ; contatori per ritardo
d2   EQU   0x08
d3   EQU   0x09
```

In questo modo dichiariamo semplicemente che le **tre label sono equivalenti per il compilatore ai tre valori numerici**:

d1 = 0x07
d2 = 0x08
d3 = 0x09

Attenzione: **valori numerici, non indirizzi**.
L' uso che poi faremo di queste label ne definirà lo scopo. Quindi è possibile:

```
    movlw   d3          ; W = 09h
    movwf   d2          ; copia W in RAM all' indirizzo 08h ←
    bsf     PORTB,d1    ; bit 7 di PORTB = 1
```

Questo modo è molto semplice, ma non è particolarmente consigliabile a meno avere la necessità di conoscere o fissare con immediata certezza la corrispondenza tra label e indirizzo.
Infatti, osservando le caratteristiche di diversi PIC l' indirizzo di partenza dell' area dati è variabile a seconda del modello di chip.
Occorre quindi, per ognuno, conoscere dove è allocata la RAM, e cambiando processore, si dovrà probabilmente riscrivere il tutto. L' equivalenza diretta con valori assoluti fa si che il codice risultate sia minimamente portabile.

CBLOCK - Codice assoluto

La sintassi di **CBLOCK** è semplice:

```
CBLOCK   indirizzo_di_partenza
   <label>[:<incremento>][,<label>][,<label>]  <;commento eventuale>
ENDC
```

L' oggetto di **CBLOCK** è l' indirizzo di memoria da cui partire, in questo caso 07h, dove inizia la RAM dati.

```
CBLOCK 0x07     ; inizio area RAM
```

Le label potranno essere assegnate a righe singole oppure sulla stessa linea, separate da virgole.
Esaurite le dichiarazioni, occorre chiudere il blocco, con la direttiva **ENDC**.

Esempio:

```
CBLOCK 0x07     ; inizio area RAM 12F519
  d1, d2, d3    ; 3 contatori per ritardo
  memA:3        ; buffer 3 bytes
  sGPIO         ; byte shadow
ENDC
```

abel	Indirizzo in RAM		Label	Indirizzo in RAM
d1	07h		memA	0Ah
d2	08h			0Bh, 0Ch
d3	09h		sGPIO	0Dh

Osserviamo che se alla label non viene assegnato un incremento, essa riserva una sola locazione. Però, se occorre, la label può essere assegnata ad un gruppo di locazioni. Ad esempio:

```
CBLOCK 0x0A
    memA:3
ENDC
```

assegna alla label **memA** tre locazioni successive a partire dalla posizione in cui si trova nella lista. Da notare che in questo caso l' equivalenza simbolica è:

Label	Indirizzo in RAM
memA	0Ah
memA+1	0Bh
memA+2	0Ch

ovvero delle tre locazioni solamente la prima ha un vero e proprio "nome", mentre le successive, se devono essere identificate, usano la label della prima e l' indicazione **+n** a seconda della loro posizione.

La direttiva **CBLOCK**:

- non va posta in prima colonna del testo.
- deve essere chiusa da **ENDC**
- può comprendere più righe, composte dalle label e da una indicazione di quante locazioni occupano. Se questo non è indicato, il valore è 1.

Da notare che le label definite nel blocco non iniziano in prima colonna, anche si si tratta di una definizione, dato che essa dipende dalla direttiva. Esistono, comunque, altri modi per assegnare la RAM, che vedremo in seguito.

La sintassi di **CBLOCK** è analoga a quella di **ORG**. La differenza di uso è sostanziale:

- **CBLOCK fa riferimento all' area RAM dati**, dove determina un blocco di definizioni e richiede **ENDC** alla fine del blocco
- **ORG fa riferimento alla memoria programma** e determina a quale indirizzo si posiziona l' istruzione successivamente indicata. Non richiede, quindi, una "chiusura".

UDATA - Codice rilocabile

Utilizzando la dichiarazione con **UDATA** posso fare definire in modo automatico l' allocazione della RAM dal compilatore.
La sintassi è semplice:

[<label opzionale>] UDATA [<indirizzo>] [<;commento eventuale>]
<label> res <quantità> [<;commento eventuale>]

La **<label>** iniziale è opzionale: si utilizza se esistono più sezioni con la direttiva.
Se è specificato un indirizzo, l'area di RAM definita sarà iniziata con quello; altrimenti inizierà nella prima locazione disponibile.
Le locazioni sono specificate con la relativa label e la direttiva **res** a cui va applicato il numero di locazioni impegnate. Così, ad esempio:

```
;=                    MEMORIA RAM                    =
;===================================================================
    UDATA
d1  res 1
d2  res 1
d3  res 1
```

UDATA stabilisce l' indirizzo a cui far corrispondere le label successive e la direttiva **res** (*reserve* - riserva) attribuisce la quantità indicata alle label. Otteniamo così lo stesso effetto dei metodi precedenti, ma senza la necessità di conoscere il punto di start della memoria RAM dati: esso verrà derivato dalle informazione che l' aver definito un chip su cui lavorare fornisce automaticamente all' Assembler.

Da notare che la direttiva ***non deve iniziare in prima colonna***, mentre le label, venendo dichiarate per la prima volta, devono iniziare in prima colonna.

Rispetto al **CBLOCK** o agli **EQU** , **UDATA** è un ulteriore passo di allontanamento dai valori assoluti delle risorse del chip. Infatti, se usassi la direttiva con un altro PIC, essa adatterebbe **automaticamente** l' indirizzo di partenza del blocco di RAM dati in base alle informazioni fornite dal file .lkr.