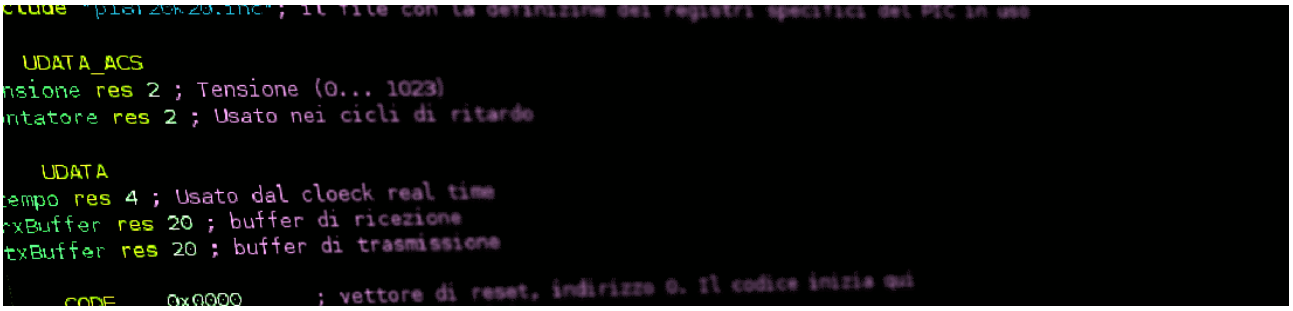
Le variabili



L'uso della memoria fatto scrivendo manualmente gli indirizzi con codifica esadecimale è spesso complicata da leggere e grande fonte di errori. Per questo si preferisce assegnare un nome a ciascuna variabile, lasciando all'assemblatore l'onere di individuare lo spazio effettivo in RAM da utilizzare.

Variabili create nell'Access Bank

Questa è la modalità più semplice per creare e usare variabili.

Le variabili da memorizzare all'interno dell'Access Bank devono essere dichiarate dopo la parola chiave `UDATA_ACS`, specificando una *label*, la parola chiave `res` e la dimensione in byte.

Esempio 1 - le righe 1-4 del codice seguente chiedono all'assemblatore di riservare lo spazio necessario per memorizzare tre variabili di dimensioni pari ad un byte; queste celle saranno identificate dalle tre label `a`, `b` e `pippo`)

```
1       UDATA_ACS       ; Variabili contenute nell'Access Bank
2   a       res 1       ; Prima variabile, dimensione 1 byte
3   b       res 1       ; Seconda variabile, dimensione 1 byte
4   pippo   res 1       ; Terza variabile, dimensione 1 byte
5
6       CODE 0
7         movlw 0x05    ; Copia il numero 0x05 in WREG
8         movwf a       ; Copia il contenuto di WREG nella variabile a
9         movwf b       ; Copia il contenuto di WREG nella variabile b
10        movwf pippo   ; Copia il contenuto di WREG nella variabile pippo
11        sleep
12      END
```

Output | Watches | Variables | Call Stack | Breakpoints | Configuration Bits

Name	Type	Address	Value
a	(1) Bytes (Select Size?)	0x0	0x05
b	(1) Bytes (User size)	0x1	0x00
pippo	(1) Bytes (Select Size?)	0x2	0x00
WREG	SFR	0xFE8	5

Qualche osservazione:

- L'esempio utilizza l'istruzione `movwf` (**MOV**e **W** to **F**) per la cui descrizione si rinvia ai datasheets
- Il nome della variabile (*label*, etichetta) deve essere posto nella prima colonna.
- Due le modalità per visualizzare all'interno di MPLABX il contenuto di una variabile:
  - utilizzare la finestra *Watch* o la finestra *Variables* durante l'esecuzione passo-passo del codice, come mostrato nella figura precedente
  - posizionare il cursore del mouse sopra il nome della variabile stessa dopo l'esecuzione di un breakpoint o durante l'esecuzione passo-passo
- La scelta della cella di memoria utilizzata per contenere il valore di una variabile è demandata al *linker* che sceglie in modo automatico. Per conoscere l'effettivo indirizzo (se e quando dovesse essere necessario) è possibile utilizzare le stesse modalità viste al punto precedente oppure consultare il listato prodotto del linker
- Il totale dello spazio occupato dalle variabili nell'Access Bank non deve superare i 96 byte, altrimenti viene generato un errore in fase di compilazione
- All'accensione, il contenuto delle variabili è casuale (anche se il simulatore mostra sempre 0). L'uso di variabili inizializzate può essere considerato un'operazione avanzata

Di seguito il codice generato in corrispondenza del codice appena mostrato, in cui è effettivamente stato inserito dall'assemblatore, in modo automatico ,`ACCESS`:

```
0000 0E05 MOVLW 0x5           7: movlw 0x05
0002 6E00 MOVWF 0x0, ACCESS   8: movwf a
0004 6E01 MOVWF 0x1, ACCESS   9: movwf b
0004 6E02 MOVWF 0x2, ACCESS  10: movwf pippo
```

Variabili create in un generico banco

Per utilizzare tutta la RAM disponibile, è necessario utilizzare i banchi

Per riservare spazio per una variabile all'interno di un generico banco è necessario utilizzare la parola chiava `UDATA`, associata ad una *label*.

Esempio - Il codice esemplificativo seguente riserva:

- 1+1 byte per le variabili "a" e "b" all'interno di un banco scelto dal linker
- 1+1+1 byte per le variabili "a1", "a2" e "b1" all'interno di un banco (coincidente o meno con il precedente, a discrezione del *linker*)
- 1 byte per la variabile a_acs nell'Access Bank (come già mostrato nel paragrafo precedente)

```
primo   UDATA
a       res 1
b       res 1

secondo UDATA
a1      res 1
a2      res 1
b1      res 1

acs     UDATA_ACS
a_acs   res 1
```

Qualche osservazione, in aggiunta a quelle già fatte a proposito dell'Access Bank:

- ciascun banco contiene al massimo variabili che occupano complessivamente 256 byte
- le scelte dell'indirizzo e del banco sono demandate al linker. L'indirizzo è normalmente indicato con tre cifre esadecimali: la più significativa indica il banco (in pratica è il contenuto del `BSR`), le altre due la posizione all'interno del banco

L'uso di una variabile posta in un banco richiede obbligatoriamente la selezione preventiva del banco in cui si trova; è consigliabile, invece dell'uso diretto dello Special Register BSR, l'utilizzo della direttiva assembler `banksel`, tradotta nell'opportuna istruzione assembly `movlb` (MOVe Literal to low nibble in Bsr), ricavando automaticamente l'indirizzo del banco in cui la variabile si trova.

Esempio 2 - Dopo aver definito le variabili come nell'esempio 2, è possibile utilizzare il seguente codice:

```
  movlw 0x05

  banksel a
  movwf a

  banksel a2
```

```
  movwf a2
  movwf a1

  movwf a_acs
```

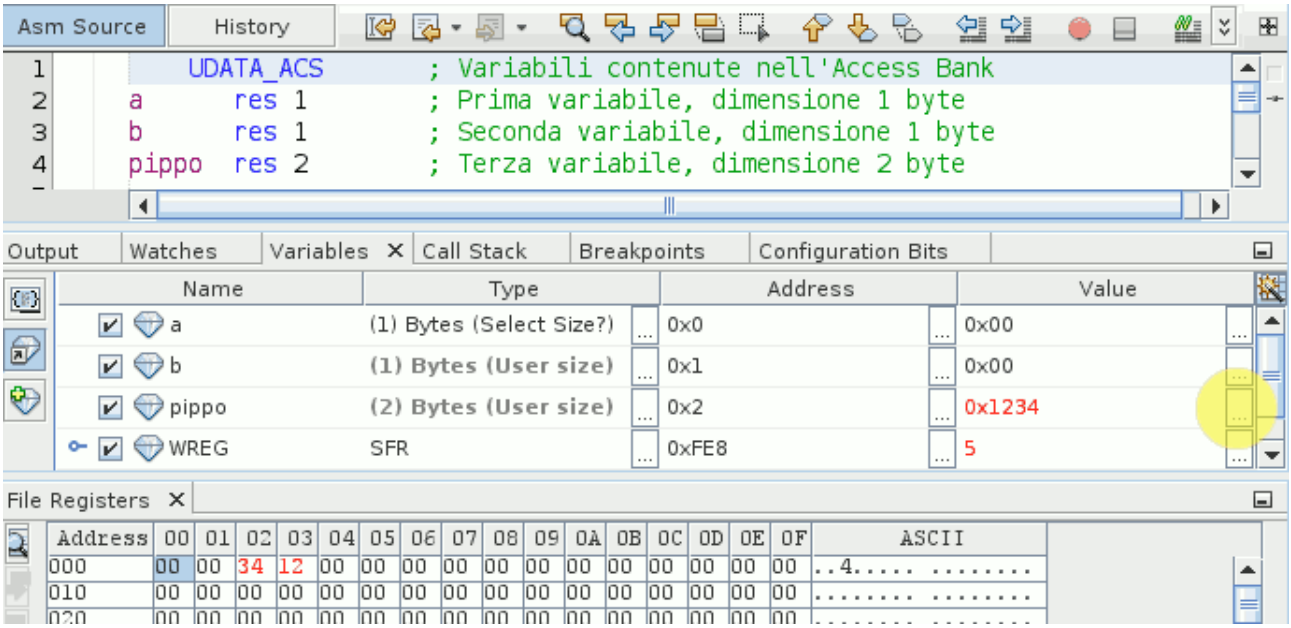
Alcune osservazioni utili per comprendere questo esempio:

- dopo aver selezionato il banco che contiene `a2`, è possibile accedere immediatamente anche alla variabile `a1`, perché creata nello stesso banco.
- la selezione di un banco errato non produce alcun tipo di segnalazione in fase di compilazione (anche se poi il programma, evidentemente, non funzionerà correttamente...)
- l'ultima riga (accesso ad una variabile in Access Bank) potrebbe essere inserita in un punto qualunque del codice, non dipendendo dal banco selezionato

Variabili di più byte

A volte una variabile di un byte è troppo piccola per contenere una certa informazione. Per esempio un numero maggiore di 255 oppure una stringa non possono essere contenuti in un solo byte. Per questo è necessario specificare che una variabile occupa più di un byte, scrivendolo dopo la parola chiave `res`.

Per esempio, la quarta riga del codice seguente specifica che la variabile `pippo` richiede 2 byte consecutivi. L'esempio fa riferimento all'*Access Bank*, ma è estensibile a tutti i banchi.



Nell'esempio riportato, l'assemblatore ha assegnato autonomamente alla variabile `pippo` le due celle con indirizzo `0x002` e `0x003`: per visualizzare due o più celle come un'unica variabile è possibile cliccare con il tasto destro sul pulsante evidenziato in giallo nella precedente figura e selezionare la quantità voluta di celle da raggruppare; analogamente è possibile selezionare la base numerica o altre rappresentazioni. La modalità con cui le variabili numeriche multi-byte sono mostrate è la *little endian*, come riconoscibile confrontando i contenuto della finestra *Variables* e della finestra *File Register*.

I singoli byte di una variabile di più byte sono accessibili aggiungendo al nome della variabile il simbolo + seguito da un numero, come fossero gli elementi di un array. Di seguito alcuni esempi.

Indirizzamento indiretto

Questo tipo di indirizzamento permette al PIC18 di accedere a tutta la RAM senza usare il concetto di banco, ma sfruttando il registro speciale a 12 bit `FSR` (*File Select Registers*) ed il registro speciale `INDF` (**IND**irect **F**ile operand):

- l'indirizzo della cella in cui scrivere o leggere è memorizzato nel registro `FSR`
- la generica istruzione di lettura o scrittura scrive o legge nel "registro virtuale" `INDF`

Il PCI18 possiede tre di tali insiemi di registri: `FSR0, FSR1, FSR2` e corrispondenti `INDF0, INDF1, INDF2`

Esempio 5 - Questo codice mostra la scrittura del numero `0x55` in una variabile `var` qualunque, usando la coppia di registri `FSR0` e `INDF0`:

```
primo   UDATA
var     res 1

        lfsr FSR0, var
        movlw 0x55
        movwf INDF0
```

- La prima istruzione (**L**oad **FSR**) inizializza il registro `FSR0` a 12 bit con l'indirizzo della variabile `var`, Si può dire, con proprietà di linguaggio, che `FSR0` "punta" alla variabile `var`.
- La terza istruzione copia nel corrispondente registro `INDF0` (**IND**irect **F**ile operands) il numero `0x55`, contenuto

in `WREG`. Tale registro viene definito nei fogli tecnici, come *virtuale*, in quanto in realtà si tratta del registro puntato da `FSR0`, cioè `var`

Si noti che nel codice non è presente alcun riferimento al banco in cui `var` si trova.

Oltre a `INDFx` sono disponibili anche altri *registri virtuali*:

- `POSTDECx`: dopo l'accesso alla variabile puntata, il puntatore viene automaticamente decrementato di 1
- `POSTINCx`: dopo l'accesso alla variabile puntata, il puntatore viene automaticamente incrementato di 1
- `PREINCx`: prima dell'accesso alla variabile puntata, il puntatore viene automaticamente incrementato di 1
- `PLUSWx`: il puntatore subisce un offset *temporaneo* pari al contenuto del registro W (-127, +128), che si comporta quindi come nelle modalità di indirizzamento indicizzato tipico di altri processori.