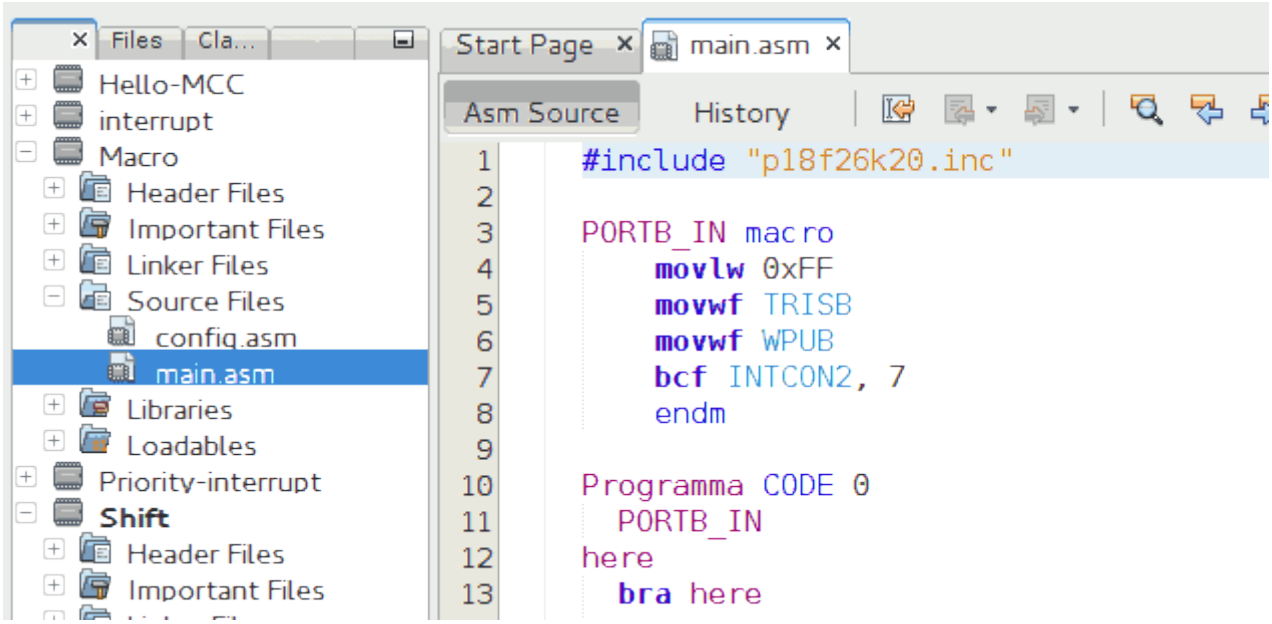
PIC18: macro



Una macro è un *pezzo di codice* generato in fase di compilazione che permette di scrivere parti ripetitive in modo compatto e semplice da comprendere. A volte viene erroneamente confusa con il concetto di ***subroutine***.

Ogni volta che viene usata una macro è come venisse eseguita un'operazione di copia_e_Incolla automaticamente dall'assemblatore, che si occupa di duplicare il codice tutte le volte che è necessario.

Un semplice esempio

Questo primo esempio mostra come scrivere una macro che configura gli otto pin di `PORTC` come uscita. Ovviamente è sufficiente azzerare il corrispondente registro `TRIS` (e quindi in questo caso l'uso delle macro non rende più compatto il numero di righe da scrivere, solo più leggibile)

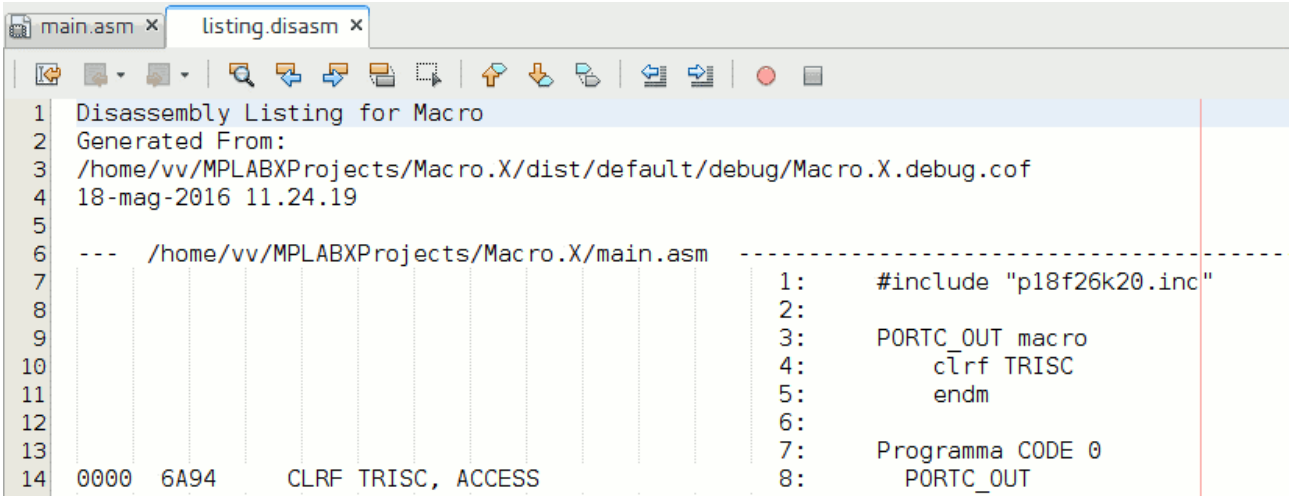
Innanzitutto occorre scrivere il codice sorgente della macro, compreso tra le parole chiave `macro` ed `endm`, assegnando un nome:

```
PORTC_OUT macro
          clrf TRISC
          endm
```

Per utilizzarla, occorre semplicemente scrivere all'interno del codice, come fosse una *normale istruzione*:

```
Programma CODE 0
  PORTC_OUT
  ...
```

Il codice eseguibile generato ignora la presenza della macro in corrispondenza della sua definizione e, quando utilizzata all'interno del programma, la sostituisce il codice corrispondente:

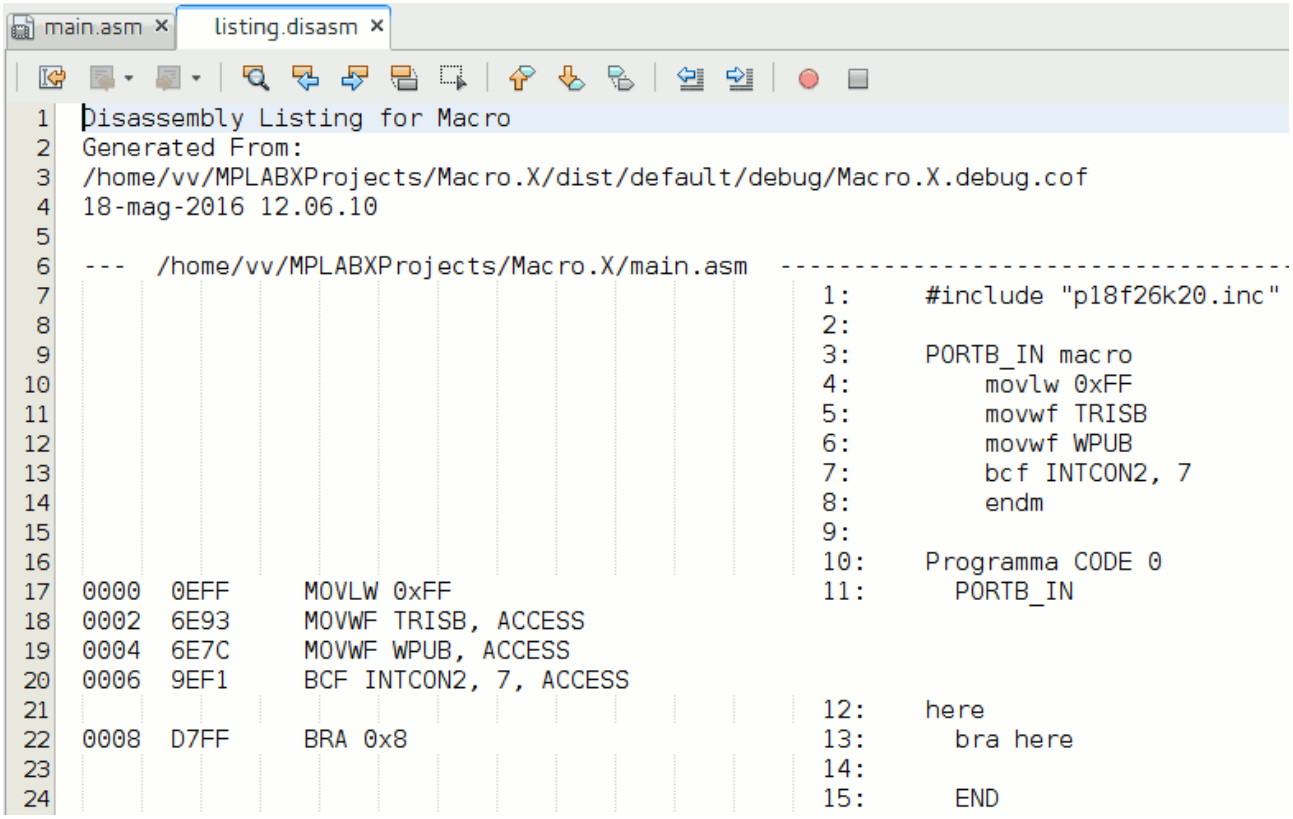


Il seguente esempio mostra una macro per configurare `PORTB` come *ingresso con pull-up*. In questo caso il codice sorgente appare, oltre che più chiaro, anche più compatto.

```
PORTB_IN macro
  movlw 0xFF
  movwf TRISB
  movwf WPUB
  bcf INTCON2, 7
  endm

Programma CODE 0
  PORTB_IN
  ...
```

Di seguito il codice eseguibile generato dall'assemblatore. Tra le righe 17 e 20, la cosiddetta *espansione della macro*.



Esempio: ciclo di ritardo

Abbiamo già scritto un ***ciclo di ritardo***. Trasformiamolo in una macro:

```
data_for_macro UDATA_ACS
contatore res 2

DELAY_100MS macro
; Ciclo di ritardo - 100 ms (circa) @ 1 MHz
  movlw 0x15
  movwf contatore+1
  clrf contatore

ripeti
  decfsz contatore
  bra ripeti
  decfsz contatore+1
  bra ripeti
  endm
```

Per far lampeggiare un LED 5 volte al secondo potremo scrivere il codice seguente:

```
main CODE 0x0000
  movlw 0
  movwf TRISC
loop
  btg LATC,0
  DELAY_100MS
  bra loop
```

Per rallentare ulteriormente potremmo pensare di invocare due volte di seguito la macro `DELAY_100MS`... ma questo causa un errore in corrispondenza della label `ripeti`:

```
Address label duplicated or different in second pass
(ripeti)
```

Per risolvere il problema è necessario dichiarare dentro la macro le label locali scrivendo subito dopo la prima riga della macro:

```
   local ripeti
```

Possiamo anche realizzare "macro di macro". Per esempio per ottenere un ritardo di un secondo la seguente soluzione funziona (anche se non è *elegantissima*...):

```
DELAY_1S macro
  DELAY_100MS
  DELAY_100MS
  DELAY_100MS
  DELAY_100MS
  DELAY_100MS
  DELAY_100MS
  DELAY_100MS
  DELAY_100MS
  DELAY_100MS
  DELAY_100MS
  endm
```

Dettagli per un uso più semplice

Due osservazioni che possono aiutare a scrivere programmi complessi. Il file corrispondente è scaricabile cliccando su **simple_delay.inc**.

Uso di un file separato per le macro

Un file con le macro poste all'inizio è poco intuitivo da leggere. Per questo si può spostare il codice relativo alle macro in un file separato e salvato in genere nella cartella virtuale `Header File`. Nel programma principale occorre poi usare (per questo esempio) con la direttiva `#include "simple_delays.inc"`.

```
#include "p18f26k20.inc"       ; Il file con la definizione
dei registri specifici del PIC in uso (standard)
#include "simple_delays.inc"   ; Il file contenete le macro
(scritto dal programmatore)

main CODE 0x0000 ; vettore di reset, indirizzo 0. Il codice
inizia qui

  movlw 0 ; azzera il registro W (accumulatore)
  movwf TRISA ; copia il contenuto del registro W nel
registro speciale TRISC

  movlw 0xFF
  movwf LATC
loop ; (imposta PORTC come uscita)
  btg LATC,0
  movlw 0x12
  DELAY_1S
  bra loop

  END
```

Il fatto di avere un file contenente le varie macro personali rende anche facile il riutilizzo in progetti diversi senza i problemi del copia e incolla fatto volta per volta a mano.

Salvataggio dei registri

Una macro potrebbe modificare i registri ed in effetti l'ultimo esempio modifica diversi registri (quali?). Nel file **simple_delay.inc** questi registri vengono salvati in variabili temporanee prima dell'esecuzione e ripristinati alla fine.