Esercitazioni PIC Baseline

1Aw - Accendi un LED : macro istruzioni - Assembly

Cosa vogliamo ottenere

Vediamo una piccola, ma significativa modifica al sorgente descritto nella prima esercitazione.

Abbiamo detto che una delle azioni essenziali che consentono i linguaggi di programmazione è quella di sostituire valori assoluti con simboli o "etichette" (label).

Abbiamo visto come questo consenta di realizzare sorgenti leggibili e facilmente maneggiabili, oltre a costituire un elemento di prima astrazione dal livello puramente hardware.

Con "livello di astrazione" intendiamo qui una modalità che permetta di scrivere un sorgente quanto più possibile indipendente dalla disposizione delle risorse nella mappa di memoria e dalle caratteristiche dei registri che gestiscono le periferiche. Questo, abbiamo visto, permette di utilizzare una label al posto di un valore assoluto, valore che sarà assegnato dal compilatore in base alle risorse rese disponibili con l' inclusione di files ***.inc*** o simili.

Questa necessità diventa irrinunciabile nel momento in cui si scrivono programmi rilocabili ed è particolarmente avanzato quando si fa uso di linguaggi ad alto livello, dotati di librerie, funzioni, ecc..

Nel primo esercizio abbiamo visto come passare, nella definizione del pin di uscita per il comando del LED, da un indirizzo assoluto "porta, pin" in esadecimale ad una coppia di label equivalenti.

Nel nostro caso, le label **GPI0** e **GP5**, che abbiamo visto essere determinate dal file ***p12F519.inc*** , si tratta di label "pre confezionate" e riservate dall' Assembler. La label **LED** invece la creiamo noi ex novo con la direttiva **#define**, dove **GPI0,GP5** si fa corrispondere all' indirizzo del bit in quel particolare port. Dal momento della definizione in avanti, per quel sorgente, al posto di scrivere:

```
bsf  6,5              ; accendi il LED
```

in base alla precedente equivalenza potrò di scrivere:

```
bsf  GPI0,GP5         ; accendi il LED
```

Vediamo come ora la linea di istruzione assume un diverso aspetto: l' azione diventa più chiara, avendo fatto uso di label che "ricorda" a cosa si fa riferimento.

Dovrebbe essere, però, anche evidente che l' astrazione delle label in qualche modo nasconde la realtà hardware che è sottintesa, rendendola trasparente all' utilizzatore; e questo è un rischio che si corre quanto maggiore è il grado di astrazione, ad esempio con un linguaggio C, dove complesse azioni di inizializzazione di registri e porte vengono eseguiti con un solo comando, facendo guadagnare in semplicità del sorgente, ma facendo perdere all' utente la conoscenza delle

complessità che condensano ed impedendo così di usare periferiche e moduli al di fuori delle funzioni e delle librerie disponibili.

Nell' Assembly, basato solamente sugli opcodes nativi del processore, l' uso di funzioni e librerie è pure possibile, ma in generale occorre crearsele e questa azione, se da una parte può essere lavoro gravoso, dall' altra mantiene sempre in contatto il programmatore con l' hardware.

Un primo passaggio di "semplificazione" sorgente è ottenuto con un più intensivo uso di simboli.

Nello schema realizzato, abbiamo collegato al pin GP5, che corrisponde al bit 5 per registro GPIO, un LED.

#define

Possiamo trasformare in forma simbolica questa situazione con una label che riassuma in se le varie componenti, creando una definizione ad hoc. Questo si ottiene ancora con la direttiva dell' Assembler **`#define`**.

 La direttiva **`#define`** dispone affinchè il compilatore sostituisca alla label dichiarata un testo, che può essere un valore assoluto o un'altra label. Questo consente di utilizzare solo valori simbolici nel listato, sostituendo gli assoluti e, nello stesso tempo, permette una maggiore leggibilità del sorgente in quanto le label assegnate avranno di preferenza qualche collegamento logico (mnemonico) con l' assoluto rappresentato.

La sintassi è semplice:

`#define`	sp	`<label>`	sp	`[oggetto]`	sp	`[; commento]`

Da notare che la direttiva **`#define`** , contrariamente alla altre direttive, **deve iniziare in prima colonna**. Almeno uno spazio separa i vari componenti della riga.

E' possibile omettere il commento, ma anche la stringa, quando si vuole definire solamente la label. La **`label`** è il nome che intendiamo definire e che possiamo collegare ad una stringa. Se non dichiariamo alcuna stringa, la label viene comunque definita, anche se non corrisponde ad un valore specifico. L'Assembler userà questa definizione durante la compilazione.
Quindi la riga:

```
#define    New_name
```

definisce la label **`New_name`**, di fatto limitandosi ad occupare quello specifico nome ed inserirlo nella lista delle label. In pratica possiamo anche definire una label senza poi utilizzarla mai nel programma o utilizzandola con scopi diversi dal collegamento con un indirizzo reale.
Per contro:

```
#define    driver    PORTB,5
```

stabilisce il collegamento tra al label **`driver`** e il bit 5 di **`PORTB`** .

Se, per qualche esigenza della logica del programma, occorre annullare una definizione, si utilizzerà

la direttiva **#undefine**, seguita dalla label che si vuole eliminare dalla lista di quelle definite.

Ovviamente, le label che creiamo noi possono essere nominate come meglio si preferisce: quindi potremo definire **LED** o **LED_Rosso** o **LED1** o anche **Gustavo**, ma è opportuno che i nomi assegnati siano in qualche modo degli"mnemonici" che ricordino alla lettura la loro funzione.

 Dovrebbe essere ora più chiaro il concetto di "label" e l' importanza che hanno i simboli nella scrittura di un buon sorgente. Vedremo la cosa ancora meglio nelle varie esercitazioni.

Usare Label

L' uso di label rende il programma più leggibile e più portabile e consente di effettuare modifiche in pochi istanti.

Definiamo allora la parola "LED" come label a cui il compilatore sostituirà il bit di comando del LED.

```
;*****************************************************************
;                            DEFINIZIONI
;
;Uscita comando LED
;-------------------
#define   LED   GPI0,GP5            ; assegna label
```

Da questo momento il compilatore, ogni volta incontrerà la label **LED** la sostituirà con **GPI0,GP5** , che a sua volta sarà compilato come bit 5 del registro 6.

Il "LED" è ora collegato attraverso i simboli alla sua situazione hardware reale.

Quindi, dovendo accendere il LED, la linea:

```
      bsf   GPI0,GP5          ; accendi il LED    (1)
```

sarà sostituibile con l' equivalente:

```
      bsf   LED               ; accendi il LED    (2)
```

Il processo di compilazione sarà eseguito in questo modo:

1. il compilatore incontra la riga **bsf LED**, in cui **LED** è l'oggetto
2. in base al **define** imposto, sostituisce la label con **GPI0,GP5**
3. **GPI0,GP5** vengono rilevati come definiti dal file **nomeprocessore.inc** e sostituiti dai corrispondenti valori assoluti
4. il compilatore rende nell'eseguibile **.hex** il codice binario corretto per portare a 1 il bit collegato al LED

Si deve notare che, una volta definita la label con la corretta corrispondenza del bit **GP** nel registro **GPI0**, questa label potrà essere usata ogni volta in cui si voglia fare riferimento a questo bit.

Ovviamente questo procedimento sarà tanto più efficace quanto più complesso sarà il programma, permettendo oltre tutto un notevole vantaggio: volendo spostare il LED su un altro GP, ad esempio GP2, se ci limitassimo alla riga (1) dovremmo riscriverla nell' intero sorgente nella nuova forma
bsf GPI0,GP2.
Invece, una volta adottata la definizione della label LED e quindi dell' uso della riga (2), essa non avrà più la necessità di essere modificata, dato che basterà adeguare una sola volta la definizione
#define LED GPI0,GP2

In effetti nel template proposto facciamo uso di un tratto semi grafico per visualizzare immediatamente la correlazione tra bit del GPIO e funzione.

```
;=================================================================
;             DEFINIZIONE DI IMPIEGO DEI PORT
;
;GPIO map
;|  5  |  4  |  3  |  2  |  1  |  0  |
;|-----|-----|-----|-----|-----|-----|
;| LED |     |     |     |     |     |
;
;#define    GPIO,GP0    ;
;#define    GPIO,GP1    ;
;#define    GPIO,GP2    ;
;#define    GPIO,GP3    ;
;#define    GPIO,GP4    ;
#define    LED  GPIO,GP5    ; LED tra pin e Vss
```

Inoltre ci offre una rapida base per le definizioni: decommentando, ovvero togliendo il **;** dalla linea voluta possiamo trasformare la linea stessa da commento a parte attiva del programma. L' editor con la sua colorazione dei vari campi ci dà un ulteriore aiuto alla comprensione della linea.

MPASM - le label

Se non avete idee sull' argomento, può essere utile consultare le pagine relative alle **regole sulle label utilizzate in MPASM.** Inseriamo qui qualche parola sull' argomento**.**

Importante da considerare è il fatto che, la prima volta in cui una label appare nel sorgente Assembly, questo corrisponde alla sua dichiarazione.

Se non si utilizza una direttiva specifica per la dichiarazione, possono venire dichiarate direttamente, semplicemente scrivendole con inizio in prima colonna. Ad esempio:

```
fiat_lux   bsf  LED      ; accendi il LED
```

la label **fiat_lux** si riferisce all' intera riga su cui è scritta e corrisponde come valore numerico a quello dell' indirizzo della memoria programma in cui verrà posizionata l' istruzione **bsf**. Qui la label si dichiara "da sè" senza l' uso del **#define**.

 Nel caso di duplicazione di label o di altri errori sintattici, l' Assembler genererà messaggi di errore e, solitamente, **sarà impossibilitato a completare la compilazione.**

Macro Istruzioni

Ma possiamo fare un altro passo in avanti nell' uso di metodi tesi a migliorare la struttura del sorgente. Uno di questi è l' impiego delle **MACRO**, abbreviazione di **macroistruzione**.

Abbiamo accennato al fatto che **MPASM** è un **Macro-Assembler** e la creazione di macro istruzioni è la caratteristica principale di un Macro-Assembler.

 **Una macro istruzione, come suggerisce il nome, è una serie di istruzioni, o altri elementi del sorgente, raggruppate sotto un nome simbolico in modo da permetterne l'esecuzione per mezzo di un unico comando.**

Vediamone una applicazione nel nostro caso.

Per accendere il LED abbiamo usato la riga:

```
        bsf  GPIO, GP5    ; accende LED
```

che contiene l' istruzione **bsf**.

Con un primo approccio simbolico abbiamo definito **LED = GPIO,GP5.**

Ora, con un successivo passo, trasformiamo tutta la riga in un unico simbolo, ovvero assegniamogli una label. Questo sostituisce una riga di istruzione con una singola label: è una macroistruzione.

```
Accendi_LED  MACRO
             bsf  LED      ; accende LED
             ENDM
```

La direttiva **MACRO** è la chiave necessaria: quanto posto tra il comando e la sua chiusura **END** viene considerato dal compilatore Macro -Assembler come equivalente della label indicata.
Ovvero, richiamando la label **Accendi_LED** sarà come avere scritto quanto definito nella macro.

MACRO - ENDM

Una "Macro" è definita con la direttiva omonima e consiste in una lista di opcode o altre componenti dell' Assembler che vengono raccolte sotto una unica label. Quando, durante la sua azione, il compilatore incontra la direttiva, essa viene sostituita con quanto indicato nel sorgente al momento della definizione della macro.
La sintassi consente di utilizzare più linee:

label	sp	MACRO	sp	nomeMacro	sp	[Parametri]	sp	[; commento]
		istruzioni istruzioni ……						
		ENDM						

Notiamo che non si fa uso, come visto prima, dello statement **#define** per la label: essa si auto definisce apparendo in prima colonna, mentre la coppia **MACRO-ENDM** stabilisce il "contenuto" della label stessa.

L'utilizzo più semplice di una macro potrebbe essere la denominazione di una serie di istruzioni ripetitive per evitare di riscriverle più volte nel sorgente. Durante la compilazione, alla label della macro verrà sostituito questo contenuto.
Il contenuto delle righe può essere il più vario, comprendendo anche altre macro. Inoltre le macro possono essere collezionate in file da includere con la direttiva **#include**, in modo da formare librerie di funzioni utili che possono essere richiamate secondo la necessità.

Da notare che la definizione delle macro deve essere effettuata nel sorgente prima di richiamarle.

Le macro possono assumere aspetti complessi ed utilizzare anche parametri; vedremo durante le esercitazioni prossime alcuni di queste possibilità.

In questo caso il contenuto è una sola riga, ma può essere ben più complesso, sottintendendo più righe;

Il processo del compilatore nel nostro caso è il seguente:

1. il compilatore incontra nel sorgente la label **Accendi_LED** , che trova definita come macro
2. sostituisce alla label la riga **bsf LED**
3. rileva che la label **LED** è definita e la sostituisce con **GPIO,GP5**
4. rileva che le label **GPIO** e **GP5** sono definite nel file ***.inc*** e le sostituisce con i valori relativi
5. genera il file ***.hex*** con il corretto codice binario per portare a 1 il bit **GP5**

Tutto questo processo di sostituzioni è completamente a carico del compilatore; all' utente resta solamente l' obbligo di avervi inserito solamente label che sono state preventivamente definite. Se una label non risulta definita prima del suo uso oppure la sua definizione è erroneamente duplicata, il compilatore genera una lista di errore e non procede alla creazione del file ***.hex*** fino a quando l' errore non viene corretto.

Possiamo divertirci a scrive anche altre macro:

```
Set_Osccal    MACRO
              movwf  OSCCAL
              END

Clear_GPIO    MACRO
              clrf   GPIO
              END

GP5_out       MACRO
              movlw b'11011111' ; maschera direzione PORT
              tris  GPIO        ; al registro direzione
              END

Blocca_PC     MACRO
              goto  $
              END
```

In questo caso, la parte del sorgente che effettua le operazioni sul pin si ridurrà semplicemente a:

```
MAIN:
   Clear_GPIO
   GP5_out
   Accendi_LED
   Blocca_PC
```

Non è C o BASIC, ma Assembly con un impiego intensivo di macro: nella compilazione, il Macro-Assembler sostituirà ogni label con le istruzioni indicate nella creazione delle macro.
Compilando questa versione, potrete osservare nel file **.lst** l' espansione delle macro eseguita da MPASM.

Nella pratica, l' impiego delle macro è a discrezione dell' utilizzatore, ma va considerato che il loro scopo è essenzialmente quello di facilitare il lavoro del programmatore e rendere il sorgente più agile e leggibile.
Non è, però, il caso di esagerare ed è necessario, a seconda della destinazione d' uso, valutare la scelta migliore tra macro e subroutine, che vedremo nell' esercitazione successiva.

Per maggiori dettagli sulle macro, potete consultare queste pagine, dove si raffrontano con le subroutines, che vedremo nelle prossime esercitazioni, due tecniche comuni nella stesura di un programma.

Va notato che, come una label deve essere definita prima di essere usata, altrettanto vale per una macro. Se non definiamo le label o le macro prima di utilizzarle, il compilatore genererà una segnalazione di errore.

Dato il programma minimale, anche la macro è minimale; vedremo nel corso delle esercitazioni un impiego più massiccio di macro più complesse.

Una nota

In coda ai sorgenti non sono più inserite le documentazioni del processori, allo scopo di non appesantirle.

Questi testi di documentazione sono però disponibili a parte e, volendo, possono essere inclusi alla fine del sorgente con un copia-e-incolla.

Ricordiamo che l' inclusione del testo dopo lo statement `END` fa si che questo sia presente nel sorgente, ma non influisca minimamente nella compilazione.

12F519 - 1Aw_519.asm

```
;*****************************************************************
;-----------------------------------------------------------------
;
;    Titolo        :  Corso Assembly & C - Esercitazione 1Aw_519
;                     Accendere un LED collegato al pin GP5.
;                     Il LED si accende all' arrivo della tensione
;                     di alimentazione e rimane acceso finchè questa
;                     è presente

;    PIC           :  12F519
;    Supporto      :  MPASM
;    Versione      :  1.0
;    Data          :  01-05-2013
;    Ref. hardware :
;    Autore        :  afg
;
;-----------------------------------------------------------------
;
;  Impiego pin :
;  ---------------
;       12F519 @ 8 pin
;
;                    |__\/__|
;              Vdd -|1    8|- Vss
;              GP5 -|2    7|- GP0
;              GP4 -|3    6|- GP1
;         GP3/MCLR -|4    5|- GP2
;                    |______|
;
;    Vdd                 1: ++
;    GP5/OSC1/CLKIN      2: Out  LED alla Vss (R in serie)
;    GP4/OSC2            3:
;    GP3/!MCLR/VPP       4:
;    GP2/T0CKI           5:
;    GP1/ICSPCLK         6:
;    GP0/ICSPDAT         7:
;    Vss                 8: --
;
;*****************************************************************
;=================================================================
;            DEFINIZIONE DI IMPIEGO DEI PORT
;
;GPIO map
;|  5  |  4  |  3  |  2  |  1  |  0  |
;|-----|-----|-----|-----|-----|-----|
;| LED |     |     |     |     |     |
;
;#define    GPIO,GP0    ;
;#define    GPIO,GP1    ;
;#define    GPIO,GP2    ;
;#define    GPIO,GP3    ;
;#define    GPIO,GP4    ;
#define    LED  GPIO,GP5    ; LED tra pin e Vss
;
;*****************************************************************
;#################################################################

        LIST      p=12F519         ; Definizione del processore
        #include <p12F519.inc>
```

```
        radix     dec               ; radice decimale

;#####################################################################
;=====================================================================
;                           CONFIGURAZIONE
;
; Oscillatore interno, no WDT, no CP, pin4=GP3
  __config  _IntRC_OSC & _IOSCFS_4MHz & _WDTE_OFF & _CP_OFF & _CPDF_OFF &
_MCLRE_OFF


;#####################################################################
;=====================================================================
;                           MACRO LOCALI
Accendi_LED  MACRO
           bsf   LED     ; accende il LED
             ENDM

;#####################################################################
;=====================================================================
;                           RESET ENTRY
;
; movlw valore_calibrazione   prima istruzione intrinseca

; Reset Vector
RESET_VECTOR   ORG     0x00

 ; calibrazione oscillatore interno
        movwf   OSCCAL

;#####################################################################
;=====================================================================
;                       MAIN PROGRAM
;
MAIN:
; inizializzazioni dell' I/O al reset
        clrf    GPIO            ; preset GPIO latch a 0

; Assegna a GP5 la funzione di uscita digitale
; TRISGPIO        xx011111     GP5 out
        movlw   b'11011111'
        tris    GPIO

; accende LED portando a livello 1 il pin GP5
        Accendi_LED

; blocco - loop chiuso
        goto    $

;*********************************************************************
;=====================================================================
;                             THE END
; fine sorgente
        END
```

12F509-12F509 1Aw_5089.asm

```
;*****************************************************************
;-----------------------------------------------------------------
;
;    Titolo        :  Assembly & C - Esercitazione 1Aw_5089
;                     Accendere un LED collegato al pin GP5.
;                     Il LED si accende all' arrivo della tensione
;                     di alimentazione e rimane acceso finchè questa
;                     è presente
;
;    PIC           :  12F508/509
;    Supporto      :  MPASM
;    Versione      :  1.0
;    Data          :  01-05-2013
;    Ref. hardware :
;    Autore        :  afg
;
;-----------------------------------------------------------------
;
;  Impiego pin :
;  ---------------
;       12F508/509 @ 8 pin
;
;                   |__\/__|
;              Vdd -|1    8|- Vss
;              GP5 -|2    7|- GP0
;              GP4 -|3    6|- GP1
;       GP3/MCLR -|4    5|- GP2
;                   |______|
;
;    Vdd                 1: ++
;    GP5/OSC1/CLKIN      2: Out  LED alla Vss (R in serie)
;    GP4/OSC2            3:
;    GP3/!MCLR/VPP       4:
;    GP2/T0CKI           5:
;    GP1/ICSPCLK         6:
;    GP0/ICSPDAT         7:
;    Vss                 8: --
;
;*****************************************************************
;=================================================================
;            DEFINIZIONE DI IMPIEGO DEI PORT
;
;GPIO map
;|  5  |  4  |  3  |  2  |  1  |  0  |
;|-----|-----|-----|-----|-----|-----|
;| LED |     |     |     |     |     |
;
;#define    GPIO,GP0    ;
;#define    GPIO,GP1    ;
;#define    GPIO,GP2    ;
;#define    GPIO,GP3    ;
;#define    GPIO,GP4    ;
#define    LED  GPIO,GP5    ; LED tra pin 2 e Vss
;
;#################################################################
; scelta del processore
 #ifdef   __12F509
        LIST      p=12F509          ; Definizione del processore
        #include <p12F509.inc>
```

```
 #endif
 #ifdef  __12F508
        LIST      p=12F508          ; Definizione del processore
        #include <p12F508.inc>
 #endif

        radix     dec               ; radice decimale

;*****************************************************************
;=================================================================
;                          CONFIGURAZIONE
;
; Oscillatore interno, no WDT, no CP, pin4=GP3
  __config  _IntRC_OSC & _WDT_OFF & _CP_OFF & _MCLRE_ON

;*****************************************************************
;=================================================================
;                       LOCAL MACROS
;
; Comandi per il LED
 Accendi_LED   MACRO
        bsf     LED
               ENDM

;*****************************************************************
;=================================================================
;                         RESET ENTRY
;
; movlw valore_calibrazione   prima istruzione intrinseca

; Reset Vector
        ORG     0x00

 ; calibrazione oscillatore interno
        movwf   OSCCAL


;*****************************************************************
;=================================================================
;                        MAIN PROGRAM
;
MAIN:
; inizializzazioni al reset
        clrf    GPIO            ; preset GPIO latch a 0

; Assegna a GP5 la funzione di uscita digitale
; TRISGPIO        xx011111     GP5 out
        movlw   b'11011111'
        tris    GPIO


; accende LED portando a livello 1 il pin GP5
        Accendi_LED

; blocco - loop chiuso
        goto    $

;*****************************************************************
; ================================================================
;                             THE END
; fine sorgente
        END
```

10F200-10F202 1Aw_20x.asm

```
;*****************************************************************
;-----------------------------------------------------------------
;
;    Titolo        :  Corso Assembly & C - Esercitazione 1Aw_200
;                     Accendere un LED collegato al pin GP0.
;                     Il LED si accende all' arrivo della tensione
;                     di alimentazione e rimane acceso finchè questa
;                     è presente

;    PIC           :  10F200/2
;    Supporto      :  MPASM
;    Versione      :  1.0
;    Data          :  01-05-2013
;    Ref. hardware :
;    Autore        :  afg
;
;
;  Impiego pin :
;  ---------------
;       10F200/202 @ 8 pin DIP           10F200/202 @ 6 pin SOT-23
;
;                   |__\/__|                    *______|
;              NC -|1    8|- GP3            GP0 -|1    6|- GP3
;             Vdd -|2    7|- Vss            Vss -|2    5|- Vdd
;             GP2 -|3    6|- NC             GP1 -|3    4|- GP2
;             GP1 -|4    5|- GP0                  |______|
;                  |______|
;
;                        DIP  SOT
;    NC                   1:     nc
;    Vdd                  2:  5: ++
;    GP2/T0CKI/FOSC4      3:  4:
;    GP1/ICSPCLK          4:  3:
;    GP0/ICSPDAT          5:  1: Out LED alla Vss
;    NC                   6:     nc
;    Vss                  7:  2: --
;    GP3/MCLR/VPP         8:  6:
;
;*****************************************************************
;=================================================================
;            DEFINIZIONE DI IMPIEGO DEI PORT
;
;GPIO map
; |  3  |  2  |  1  |  0  |
; |-----|-----|-----|-----|
; | in  |     |     | LED |
;
#define   LED  GPIO,GP0    ; LED tra pin e Vss
;#define       GPIO,GP1    ;
;#define       GPIO,GP2    ;
;#define       GPIO,GP3    ; solo input
;
;#################################################################
 #ifdef __10F200
       LIST      p=10F200     ; definizione del processore
       #include <p10F200.inc>
 #endif
 #ifdef __10F202
       LIST      p=10F202     ; definizione del processore
       #include <p10F202.inc>
```

```
 #endif

        radix     dec           ; default numeri decimali

;#########################################################################
;=========================================================================
;                            CONFIGURAZIONE                               ;
;
; No WDT, no CP, pin4=GP3
  __config  _CP_OFF & _MCLRE_OFF  & _WDT_OFF

;#########################################################################
;=========================================================================
;                       LOCAL MACROS
;
; Comandi per il LED
 Accendi_LED   MACRO
        bsf     LED
               ENDM

;#########################################################################
;=========================================================================
;                          RESET ENTRY
;
; Reset Vector
RESET_VECTOR   ORG     0x00

 ; calibrazione oscillatore interno
        movwf   OSCCAL

;#########################################################################
;=========================================================================
;                         MAIN PROGRAM
;
MAIN:
; inizializzazioni al reset
        clrf    GPIO            ; preset GPIO latch a 0

; TRISGPIO        -------0     GP0 out
        movlw   b'11111110'
        tris    GPIO            ; al registro direzione

; accende LED
        Accendi_LED

; blocco
        goto    $

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

16F526/505 - 1Aw_526.asm

```
;*****************************************************************
;-----------------------------------------------------------------
;
;    Titolo        :  Corso Assembly & C - Esercitazione 1A_526
;                     Accendere un LED collegato al pin GP5.
;                     Il LED si accende all' arrivo della tensione
;                     di alimentazione e rimane acceso finchè questa
;                     è presente
;
;    PIC           : 16F526-16F505
;    Supporto      : MPASM
;    Versione      : 1.0
;    Data          : 01-05-2013
;    Ref. hardware :
;    Autore        :  afg
;
;-----------------------------------------------------------------
;
;  Impiego pin :
;  ----------------
;       16F505 - 16F526 @ 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  LED alla Vss
;    RB4/OSC2/CLKOUT           3:
;    RB3/!MCLR/VPP             4:
;    RC5/T0CKI                 5:
;    RC4/[C2OUT]               6:
;    RC3                       7:
;    RC2/[Cvref]               8:
;    RC1/[C2IN-]               9:
;    RC0/[C2IN+]              10:
;    RB2/[C1OUT/AN2]          11:
;    RB1/[C1IN-/AN1/]ICSPC    12:
;    RB0/[C1IN+/AN0/]ICSPD    13:
;    Vss                      14: --
;
;    [ ] solo 16F526
;
;*****************************************************************
;=================================================================
;            DEFINIZIONE DI IMPIEGO DEI PORT
;
;PORTC non utilizzato
;
; PRTB map
; |  5  |  4  |  3  |  2  |  1  |  0  |
; |-----|-----|-----|-----|-----|-----|
; | LED |     |     |     |     |     |
```

```
;
#define   LED  PORTB,RB5    ; LED tra pin e Vss
;#define       PORTB,RB3
;#define       PORTB,RB2
;#define       PORTB,RB1
;#define       PORTB,RB0

;#####################################################################
;                          SELEZIONE PROCESSORE
 #ifdef __16F526
      LIST p=16F526
      #include <p16F526.inc>
  #endif
  #ifdef __16F505
      LIST p=16F505
      #include <p16F505.inc>
 #endif

;#####################################################################
;                           CONFIGURAZIONE
;
 #ifdef __16F526
; Oscillatore interno, 4MHz, no WDT, no CP, no MCLR
 __config _IntRC_OSC_RB4 & _IOSCFS_4MHz & _WDTE_OFF & _CP_OFF & _CPDF_OFF &
MCLRE_OFF
 #endif

 #ifdef __16F505
; Oscillatore interno, 4MHz, no WDT, no CP, no MCLR
 __config _IntRC_OSC_RB4EN & _WDT_OFF & _CP_OFF & _MCLRE_OFF
 #endif

;#####################################################################
;=====================================================================
;                       LOCAL MACROS
;
; Comandi per il LED
 Accendi_LED   MACRO
         bsf     LED
               ENDM

;#####################################################################
;                          RESET ENTRY
;
; Reset Vector
        ORG     0x00

 ; calibrazione oscillatore interno
        movwf   OSCCAL

;#####################################################################
;                         MAIN PROGRAM
;
Main:
; inizializzazioni al reset
        clrf    PORTB           ; preset port latch a 0

; RB5 come out
        movlw   b'00011111'
        tris    PORTB           ; al registro direzione

; accende LED
```

```
         Accendi_LED

; loop
        goto    $

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