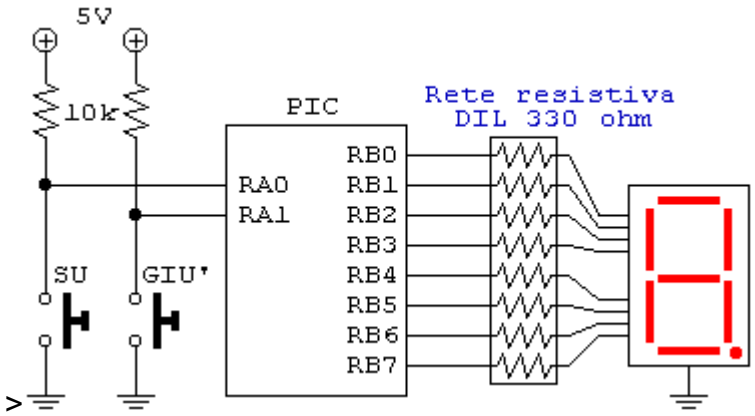
PIC
PROGRAMMAZIONE IN LINGUAGGIO MACCHINA PER PRINCIPIANTI

By Claudio Fin

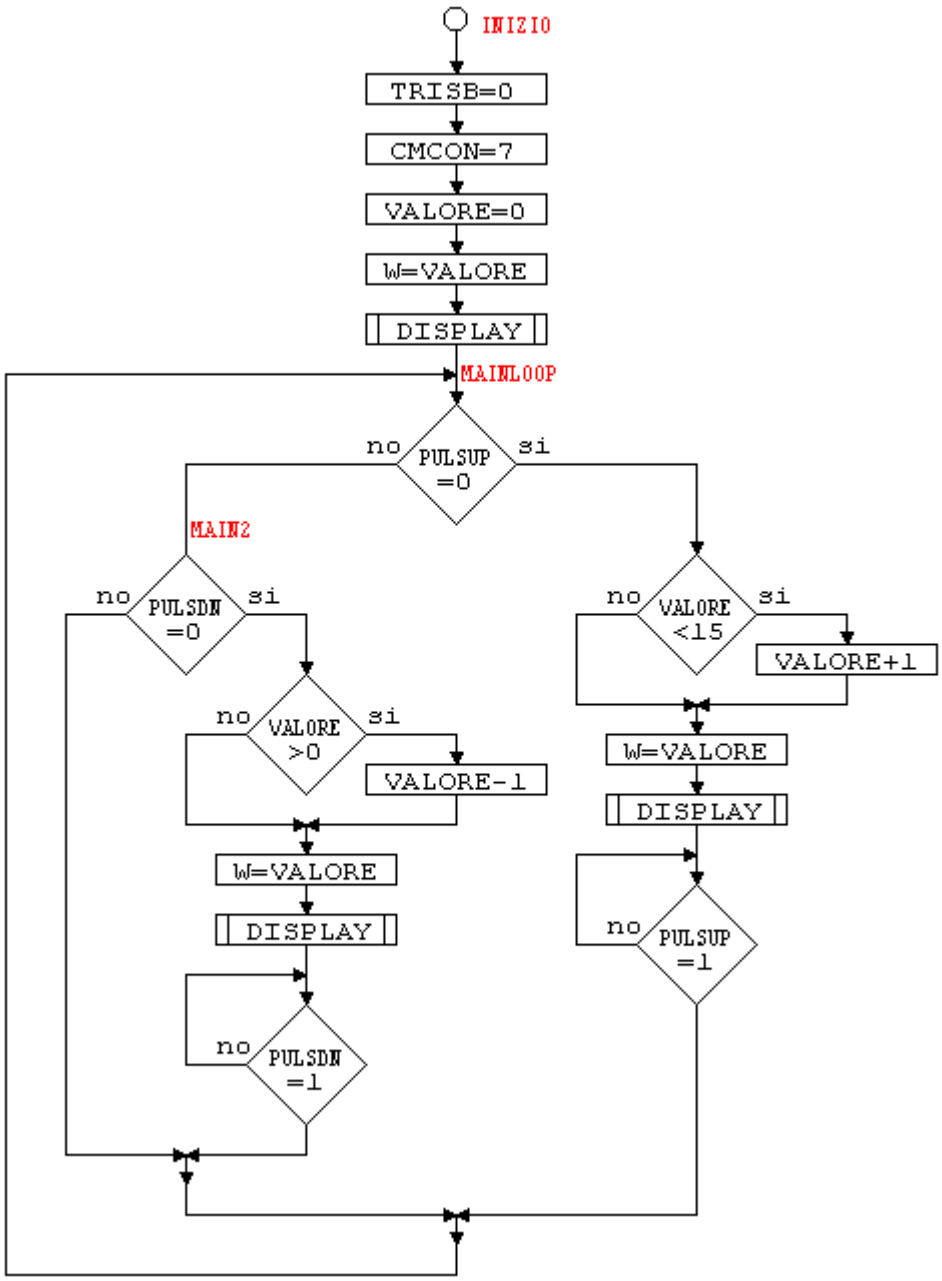
[Precedente] [Indice principale]

CONTEGGIO DA 0 A F CON PULSANTI UP/DOWN

Fino ad ora abbiamo usato il PIC come «attore protagonista», facendogli generare effetti di animazione o visualizzare valori numerici. Naturalmente però è anche possibile inviare dei segnali dall'esterno per controllare il comportamento del programma... o, più precisamente, il programma può non solo generare segnali verso l'esterno ma anche leggerli (provenienti da pulsanti, interruttori, integrati logici, transistor, fotoaccoppiatori, contatti di relè, sensori ecc...). Nell'esempio che andiamo a vedere adesso vengono usati due comuni pulsanti per incrementare e decrementare il valore contenuto nel registro VALORE. Si vuole anche limitare l'escursione del valore da 0 a 15 e rappresentarla sul display in formato esadecimale (da 0 a F). Il programma deve perciò prevedere di decrementare il registro solo se il suo contenuto è maggiore di 0, e di incrementarlo solo se il suo contenuto è minore di 15. Avendo più simboli da visualizzare, va naturalmente estesa anche la tabella di conversione dei codici display, in modo da poter rappresentare le lettere dalla A alla F. Per quanto riguarda gli ingressi vengono usati i pin RA0 e RA1. Nel PIC16F628 questi pin all'accensione sono collegati ai comparatori di tensione interni. Per poterli usare come normali linee di ingresso/uscita digitali bisogna scrivere il valore 7 nel registro CMCON (presente nel banco 0 della RAM), in un PIC16F84 questo non è necessario. Per creare due livelli logici ben distinguibili dagli ingressi, questi devono essere collegati o all'alimentazione positiva (livello 1) o a massa (livello 0). Volendo leggere lo stato di un pulsante normalmente aperto, in genere si collega una resistenza di pull-up da qualche kohm tra il positivo e l'ingresso (che ne blocca a 1 il livello di riposo), e si collega il pulsante in modo da chiudere l'ingresso verso massa quando premuto. In questo caso gli ingressi con i pulsanti non premuti si trovano a 1, quelli con i pulsanti premuti si trovano a 0.

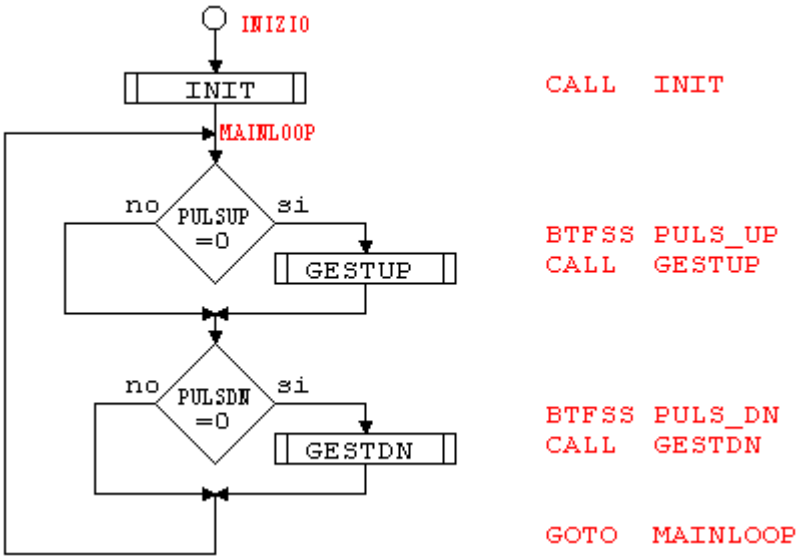


Un flowchart che realizza tutte le funzioni richieste è il seguente:



All'inizio vi sono le solite predisposizioni dell'hardware, l'azzeramento del registro VALORE, e una prima chiamata alla subroutine di visualizzazione sul display (appare la cifra 0). Questa fase viene detta di inizializzazione del sistema e delle variabili di lavoro. Dopo questa fase vi sono due strutture condizionali che controllano lo stato dei pulsanti, se nessuno è premuto si percorrono i due rami di sinistra (risposte no e no) e si ritorna all'inizio per controllare di nuovo i pulsanti in un ciclo senza fine. Nel caso in cui uno di essi sia premuto l'elaborazione prende la strada «si» relativa. Questa porta alla verifica che non siano stati raggiunti i limiti estremi del conteggio (0 o 15). Il nuovo valore viene scritto sul display, ed infine si attende che il pulsante in questione venga rilasciato prima di ritornare a mainloop. L'algoritmo è sicuramente funzionante, però comincia a diventare abbastanza complesso da scrivere in un unico blocco, sarebbero necessarie diverse istruzioni GOTO con le relative etichette a cui saltare, e nella lista delle istruzioni ci si potrebbe facilmente perdere, nel senso di non riuscire più a capire a quale punto della stuttura appartengano. Inoltre c'è un problema che vedremo tra breve che richiederebbe

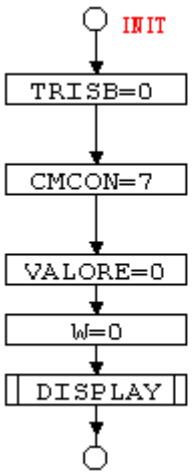
l'aggiunta di ulteriori 4 strutture condizionali... la modifica sarebbe estremamente difficile. Ragionando invece per sottoprogrammi, le sezioni che diventano troppo complesse da rappresentare e da scrivere si possono spezzare in qualcosa di più maneggevole. Per esempio la sezione principale del programma potrebbe essere scritta in un modo molto leggero ed elegante con solo 6 istruzioni di «coordinamento» del flusso di esecuzione, che richiamano 3 sottoprogrammi, uno di inizializzazione, e due specifici per gestire le operazioni da compiere quando viene premuto uno o l'altro pulsante:



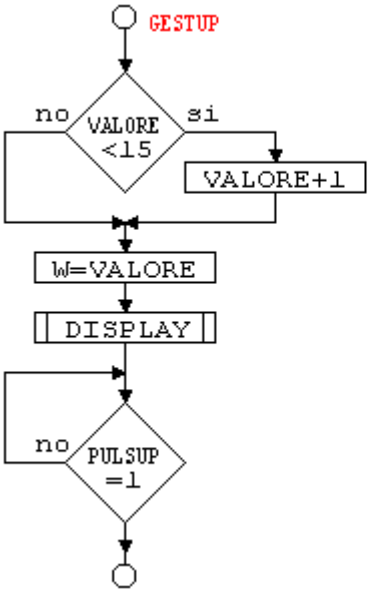
Ecco allora il programma completo, contiene qualche istruzione in più rispetto allo stretto necessario, però risulta composto da spezzoni brevi di istruzioni, più comprensibili e facilmente modificabili:

```
;-----------------------------------------------------
; Programma conteggio esadecimale up/down con pulsanti
;-----------------------------------------------------
            PROCESSOR  16F628
            RADIX      DEC
            INCLUDE    "P16F628.INC"
            __CONFIG   11110100010000B
;-----------------------------------------------------
            ORG        32
VALORE      RES        1
#DEFINE     PULS_UP    PORTA,0      ;Pulsante incremento
#DEFINE     PULS_DN    PORTA,1      ;Pulsante decremento
;-----------------------------------------------------
            ORG        0
            CALL       INIT         ;Chiama subr.inizializzazione
MAINLOOP    BTFSS      PULS_UP      ;Se non premuto PULS_UP skip
            CALL       GESTUP       ;altrimenti chiama GESTUP
            BTFSS      PULS_DN      ;Se non premuto PULS_DN skip
            CALL       GESTDN       ;altrimenti chiama GESTDN
            GOTO       MAINLOOP     ;Nuovo ciclo del programma
;-----------------------------------------------------
INIT        BSF        STATUS,RP0   ;Attiva banco 1
            CLRF       TRISB        ;Rende PORTB un'uscita
            BCF        STATUS,RP0   ;Ritorna al banco 0
            MOVLW      7
            MOVWF      CMCON        ;PORTA = I/O digitali
            CLRF       VALORE       ;Azzera valore
```

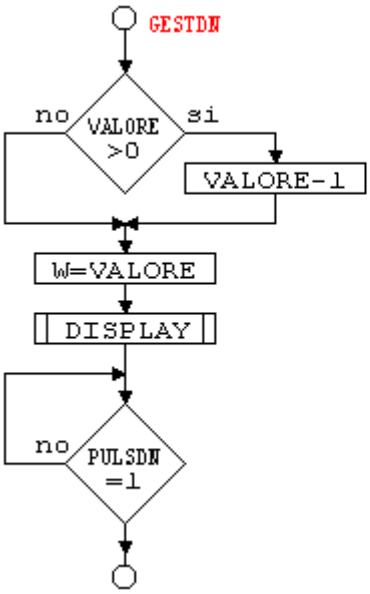
```
            CLRW
            CALL        DISPLAY      ;Chiama subroutine display
            RETURN
;-----------------------------------------------------
GESTUP      MOVLW       15
            SUBWF       VALORE,W     ;W=VALORE-15
            BTFSS       STATUS,C     ;Se VALORE >= 15 skip
            INCF        VALORE,F     ;altrimenti incrementa VALORE
            MOVF        VALORE,W     ;W=VALORE
            CALL        DISPLAY      ;Chiama subroutine display
            BTFSS       PULS_UP      ;Se non premuto PULS_UP skip
            GOTO        $-1          ;altrimenti attendi
            RETURN
;-----------------------------------------------------
GESTDN      MOVF        VALORE,W
            SUBLW       0            ;W=0-VALORE
            BTFSS       STATUS,C     ;Se VALORE <=0 skip
            DECF        VALORE,F     ;altrimenti decrementa VALORE
            MOVF        VALORE,W     ;W=VALORE
            CALL        DISPLAY      ;Chiama subroutine display
            BTFSS       PULS_DN      ;Se non premuto PULS_DN skip
            GOTO        $-1          ;altrimenti attendi
            RETURN
;-----------------------------------------------------
DISPLAY     CALL        TABDISP      ;Chiama subroutine conversione
            MOVWF       PORTB        ;Scrive valore dei LED su PORTB
            RETURN
;-----------------------------------------------------
TABDISP     ADDWF       PCL,F        ;Conversione esa->display
            RETLW       00111111B    ;0
            RETLW       00000110B    ;1
            RETLW       01011011B    ;2          -0-
            RETLW       01001111B    ;3        5|   |1
            RETLW       01100110B    ;4          -6-
            RETLW       01101101B    ;5        4|   |2
            RETLW       01111101B    ;6          -3- .7
            RETLW       00000111B    ;7
            RETLW       01111111B    ;8
            RETLW       01101111B    ;9
            RETLW       01110111B    ;A
            RETLW       01111100B    ;B
            RETLW       00111001B    ;C
            RETLW       01011110B    ;D
            RETLW       01111001B    ;E
            RETLW       01110001B    ;F
;-----------------------------------------------------
            END
```




```
BSF     STATUS,RP0
CLRF    TRISB
BCF     STATUS,RP0

MOVLW   7
MOVWF   CMCON

CLRF    VALORE

CLRW

CALL    DISPLAY

RETURN
```




```
MOVLW   15
SUBWF   VALORE,W
BTFSS   STATUS,C
INCF    VALORE,F

MOVF    VALORE,W

CALL    DISPLAY


BTFSS   PULS_UP
GOTO    $-1

RETURN
```




```
MOVF    VALORE,W
SUBLW   0
BTFSS   STATUS,C
DECF    VALORE,F

MOVF    VALORE,W

CALL    DISPLAY


BTFSS   PULS_DN
GOTO    $-1

RETURN
```

Una cosa nuova è l'indicazione $-1 messa dopo le istruzioni GOTO, questa simbologia indica semplicemente di saltare all'istruzione precedente ed evita di scrivere un'apposita etichetta. Se si scrivesse solo GOTO $ il programma si bloccherebbe perché l'istruzione GOTO continuerebbe all'infinito a saltare a sè stessa.

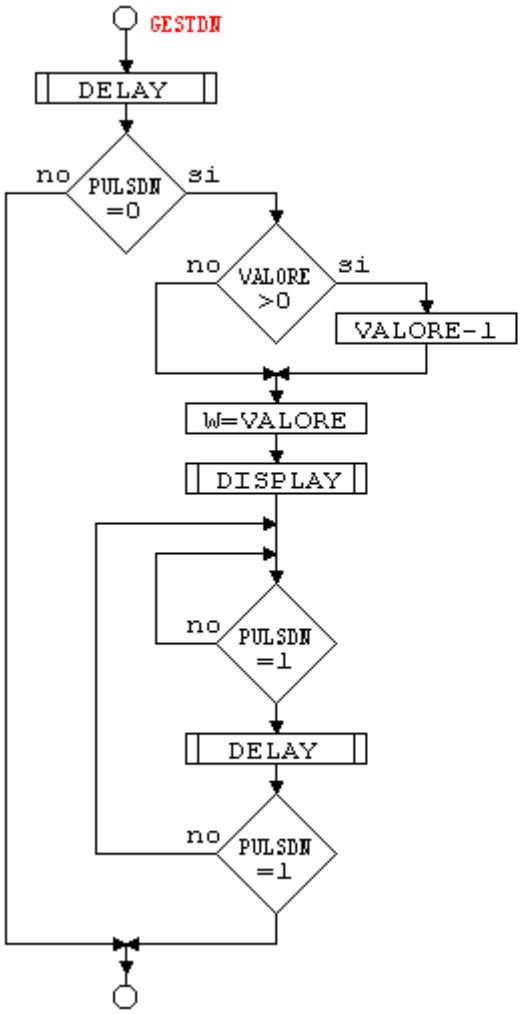
La logica del programma è perfetta, però se lo si prova in pratica ci si accorgerà di un'imperfezione, e cioè che a volte premendo un pulsante la cifra incrementa (o decrementa) di più di una unità. Questo è dovuto al fatto che i pulsanti non sono componenti ideali, e nella transizione da aperto a chiuso (e viceversa) il contatto è imperfetto e si genera un treno di impulsi (rimbalzi) che il programma naturalmente scambia per veloci pressioni ripetute. Per ovviare a questo inconveniente si possono realizzare dei circuiti antirimbalzo (debouncing) hardware, oppure si può modificare il programma in modo da renderlo insensibile ad essi. Il modo più semplice è attendere un certo tempo dopo la rilevazione della prima chiusura (poche decine di ms) e controllare se dopo questo tempo il pulsante risulta ancora chiuso. Se lo è si prende per buona la pressione, altrimenti la si ignora. Lo stesso discorso

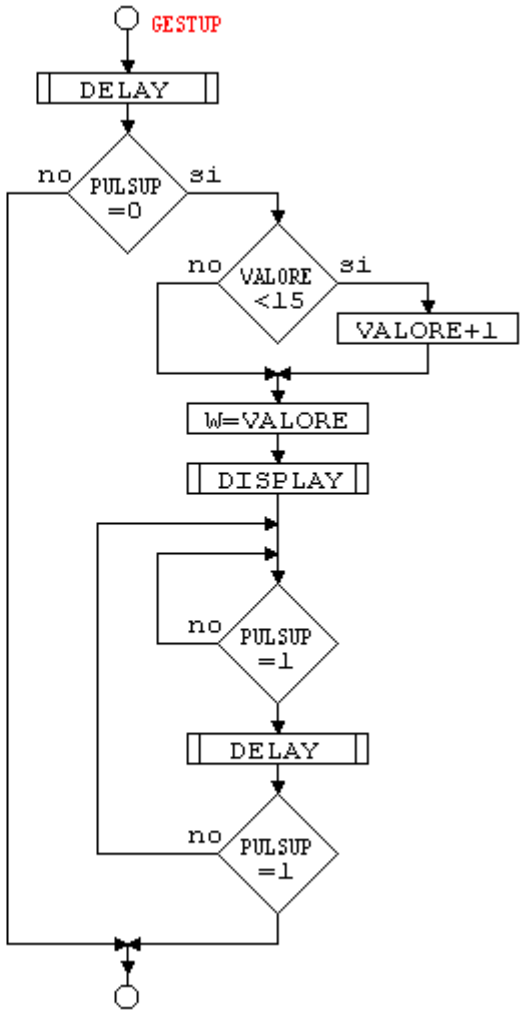
vale per l'apertura. Modificando le subroutine GESTUP e GESTDN come riportato di seguito, i rimbalzi non danno più fastidio.

```
;-----------------------------------------------------
; Prog. conteggio esa con pulsanti up/down e antirimbalzo
;-----------------------------------------------------
            PROCESSOR  16F628
            RADIX      DEC
            INCLUDE    "P16F628.INC"
            __CONFIG   11110100010000B
;-----------------------------------------------------
            ORG        32
VALORE      RES        1
H_CONT      RES        1
L_CONT      RES        1
#DEFINE     PULS_UP    PORTA,0
#DEFINE     PULS_DN    PORTA,1
;-----------------------------------------------------
            ORG        0
            CALL       INIT          ;Chiama subr.inizializzazione
MAINLOOP    BTFSS      PULS_UP       ;Se non premuto PULS_UP skip
            CALL       GESTUP        ;altrimenti chiama GESTUP
            BTFSS      PULS_DN       ;Se non premuto PULS_DN skip
            CALL       GESTDN        ;altrimenti chiama GESTDN
            GOTO       MAINLOOP      ;Nuovo ciclo del programma
;-----------------------------------------------------
INIT        BSF        STATUS,RP0    ;Attiva banco 1
            CLRF       TRISB         ;Rende PORTB un'uscita
            BCF        STATUS,RP0    ;Ritorna al banco 0
            MOVLW      7
            MOVWF      CMCON         ;PORTA = I/O digitali
            CLRF       VALORE        ;Azzera valore
            CLRW
            CALL       DISPLAY       ;Chiama subroutine display
            RETURN
;-----------------------------------------------------
GESTUP      CALL       DELAY         ;Ritardo antirimbalzo
            BTFSC      PULS_UP       ;Se ancora premuto skip
            RETURN                   ;altrimenti ritorna
            MOVLW      15
            SUBWF      VALORE,W      ;W=VALORE-15
            BTFSS      STATUS,C      ;Se VALORE >= 15 skip
            INCF       VALORE,F      ;altrimenti incrementa VALORE
            MOVF       VALORE,W      ;W=VALORE
            CALL       DISPLAY       ;Chiama subroutine display
GESTUP2     BTFSS      PULS_UP       ;Se non premuto PULS_UP skip
            GOTO       GESTUP2       ;altrimenti attendi
            CALL       DELAY         ;Ritardo antirimbalzo
            BTFSS      PULS_UP       ;Se non premuto skip
            GOTO       GESTUP2       ;Altrimenti attendi
            RETURN
;-----------------------------------------------------
GESTDN      CALL       DELAY         ;Ritardo antirimbalzo
            BTFSC      PULS_DN       ;Se ancora premuto skip
            RETURN                   ;Altrimenti ritorna
            MOVF       VALORE,W
            SUBLW      0             ;W=0-VALORE
            BTFSS      STATUS,C      ;Se VALORE <=0 skip
            DECF       VALORE,F      ;altrimenti decrementa VALORE
            MOVF       VALORE,W      ;W=VALORE
            CALL       DISPLAY       ;Chiama subroutine display
GESTDN2     BTFSS      PULS_DN       ;Se non premuto PULS_DN skip
            GOTO       GESTDN2       ;altrimenti attendi
            CALL       DELAY         ;Ritardo antirimbalzo
```

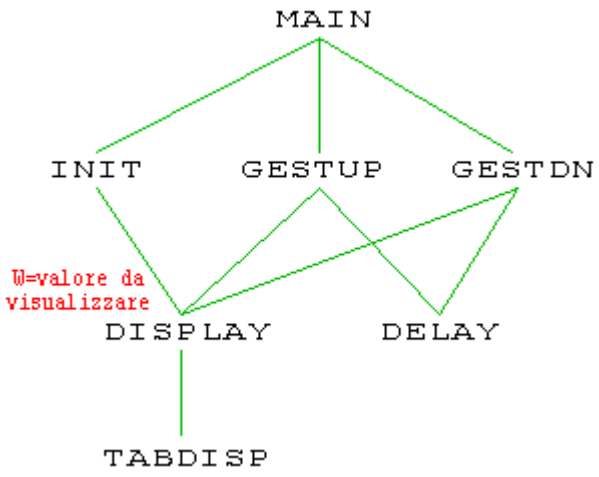
```
            BTFSS       PULS_DN      ;Se non premuto skip
            GOTO        GESTDN2      ;Altrimenti attendi
            RETURN
;-----------------------------------------------------
DELAY       MOVLW       15
            MOVWF       H_CONT
            CLRF        L_CONT
DELAY2      DECF        L_CONT,F     ;Decrementa parte bassa CONT
            COMF        L_CONT,W     ;Inverte i bit
            BTFSC       STATUS,Z     ;Se tutti zero c'è stato rollover
            DECF        H_CONT,F     ;allora decrementa parte alta
            MOVF        L_CONT,W     ;Carica in W la parte bassa
            IORWF       H_CONT,W     ;Mettila in OR con la parte alta
            BTFSS       STATUS,Z     ;Se tutto zero skip (fine ciclo)
            GOTO        DELAY2       ;Altrimenti ritorna a DELAY2
            RETURN
;-----------------------------------------------------
DISPLAY     CALL        TABDISP      ;Chiama subroutine conversione
            MOVWF       PORTB        ;Scrive valore dei LED su PORTB
            RETURN
;-----------------------------------------------------
TABDISP     ADDWF       PCL,F        ;Conversione esa->display
            RETLW       00111111B    ;0
            RETLW       00000110B    ;1
            RETLW       01011011B    ;2          -0-
            RETLW       01001111B    ;3        5|   |1
            RETLW       01100110B    ;4          -6-
            RETLW       01101101B    ;5        4|   |2
            RETLW       01111101B    ;6          -3- .7
            RETLW       00000111B    ;7
            RETLW       01111111B    ;8
            RETLW       01101111B    ;9
            RETLW       01110111B    ;A
            RETLW       01111100B    ;B
            RETLW       00111001B    ;C
            RETLW       01011110B    ;D
            RETLW       01111001B    ;E
            RETLW       01110001B    ;F
;-----------------------------------------------------
            END
```

Le subroutines GESTDN e GESTUP così modificate sono rappresentabili dai due flowcharts seguenti. Se si fa attenzione si vede che il punto di uscita finale dalle subroutines è semplicemente il ritorno al chiamante. Questo vuol dire che qualsiasi istruzione RETURN (ovunque sia messa) può essere pensata anche come un ramo vuoto che si congiunge al punto finale, questo è proprio il caso delle RETURN messe come terza istruzione in GESTUP e GESTDN, che nei grafici sottostanti sono rappresentate dal percorso «no» di uscita sulla sinistra. Questo fatto permette di creare strutture condizionali molto complesse in modo semplice evitando l'uso di GOTO (a patto naturalmente che una delle due possibili risposte conduca direttamente all'uscita).





Quando i programmi cominciano a diventare molto lunghi diventa pressochè impossibile rappresentarli completamente attraverso i flowcharts, questi si usano allora per indicare l'algoritmo (procedimento) di massima, o per chiarire il funzionamento di alcune sezioni critiche o complesse, senza però scendere fino al dettaglio della singola istruzione. Quando si devono elaborare molti dati diventa più utile un diagramma del flusso dati (dataflow). Per l'esempio precedente un dataflow è praticamente inutile, perché si ridurrebbe all'indicazione che il main trasmette al driver display il valore da visualizzare nel registro W. Un altro schema riassuntivo puòessere quello dell' albero delle chiamate ai sottoprogrammi in modo da vedere chi chiama chi. Nella figura seguente si vede che il nostro programma raggiunge tre livelli di profondità (depth) nelle chiamate (e nell'occupazione dello stack).



Vedendo che con questo semplice programma usiamo già tre livelli di profondità, si potrebbe pensare che gli 8 livelli ammessi dallo stack siano del tutto insufficienti per scrivere programmi molto più complessi. In realtà questo non è vero, perché anche in programmi più sofisticati di solito è difficile avere la necessità di andare oltre il quinto livello.

[Segue]

Pagina creata nel febbraio 2004 - Ultimo aggiornamento 7-1-2005