PIC
PROGRAMMAZIONE IN LINGUAGGIO MACCHINA PER PRINCIPIANTI

By Claudio Fin

L'OUTPUT A TERMINALE

Scrittura di una stringa

La stampa di una stringa di caratteri consiste nell'invio uno dopo l'altro di tutti i codici ASCII dei caratteri da visualizzare a video. I PIC sono piuttosto limitati per quanto riguarda la memorizzazione di stringhe nella memoria programma, e bisogna ricorrere anche questa volta alle tabelle di istruzioni RETLW.

```
STR1        ADDLW       PCL,F
            RETLW       'H'
            RETLW       'E'
            RETLW       'L'
            RETLW       'L'
            RETLW       'O'
            RETLW       ' '
            RETLW       'W'
            RETLW       'O'
            RETLW       'R'
            RETLW       'L'
            RETLW       'D'
```

Per inviare questi codici, cioè la frase «HELLO WORLD» occorre una piccola subroutine che li prelevi uno ad uno dalla tabella (leggendola con un indice progressivo) e li passi alla subroutine di trasmissione vera e propria.
Per la lunghezza della stringa si possono usare due sistemi, o si conosce già a priori la sua lunghezza e si imposta un registro contatore per inviare un numero ben preciso di caratteri, oppure si inserisce una RETLW aggiuntiva in fondo alla stringa che ritorna il valore 0. In questo caso la subroutine continuerà ad inviare caratteri finchè non incontra uno 0. Una stringa di questo tipo in altri linguaggi di programmazione si chiama "null terminated string".
Entrambi i sistemi hanno vantaggi e svantaggi, nel primo caso si risparmia una riga nella tabella ma occorre usare un registro contatore, nel secondo caso è esattamente l'opposto, si risparmia un registro contatore ma occorre usare una RETLW in più

```
;-----------Caso1

STRINGA     MOVLW       11
            MOVWF       CONT
            CLRF        INDX
STRINGA2    MOVF        INDX,W
            CALL        STR1
            MOVWF       DL
            CALL        TX00
            INCF        INDX,F
```

```
            DECFSZ      CONT,F
            GOTO        STRINGA2
            RETURN


;-----------Caso2

STRINGA     CLRF        INDX
STRINGA2    MOVF        INDX,W
            CALL        STR1
            XORLW       0
            BTFSC       STATUS,Z
            RETURN
            MOVWF       DL
            CALL        TX00
            INCF        INDX,F
            GOTO        STRINGA2
```

Ecco quindi il programma completo per scrivere la prima frase, che usa il metodo della null terminated string e la USART interna:

```
            PROCESSOR   16F628
            RADIX       DEC
            INCLUDE     "P16F628.INC"
            __CONFIG    11110100010000B
DL          EQU         32
INDX        EQU         33
            ORG         0
            BSF         STATUS,RP0   ;banco 1
            MOVLW       25
            MOVWF       SPBRG
            BSF         TXSTA,BRGH   ;9600 BAUD
            BSF         TXSTA,TXEN   ;ABILITA TX
            BCF         STATUS,RP0   ;banco 0
            BSF         RCSTA,SPEN   ;ABILITA SERIALE
            CALL        STRINGA      ;Invia stringa
            SLEEP                    ;Stop programma
;-----------------------------------------------------
STRINGA     CLRF        INDX
STRINGA2    MOVF        INDX,W
            CALL        STR1
            XORLW       0
            BTFSC       STATUS,Z
            RETURN
            MOVWF       DL
            CALL        TX00
            INCF        INDX,F
            GOTO        STRINGA2

STR1        ADDLW       PCL,F
            RETLW       'H'
            RETLW       'E'
            RETLW       'L'
            RETLW       'L'
            RETLW       'O'
            RETLW       ' '
            RETLW       'W'
            RETLW       'O'
            RETLW       'R'
            RETLW       'L'
            RETLW       'D'
            RETLW       0
;-----------------------------------------------------
```

```
TX00        BTFSS       PIR1,TXIF    ;Attende trasmettitore libero
            GOTO        $-1
            MOVF        DL,W
            MOVWF       TXREG        ;Invia dato
            RETURN
;-------------------------------------------------
            END
```

E'utile ricordare che il set dei caratteri ASCII prevede simboli stampabili a partire dal codice 32 (spazio) fino al 127. I codici dallo 0 al 31 sono invece definiti "caratteri di controllo", perché servivano a dare comandi ai vecchi terminali e stampanti. I due codici forse più comuni sono il ritorno carrello (CR) e avanzamento riga (LF), i cui codici sono rispettivamente 13 e 10 (0D e 0A in esadecimale).

Per andare a caporiga e scrivere una nuova stringa occorre inviare questi due valori, che tra l'altro sono ancora oggi presenti (anche se normalmente non visibili) alla fine di ogni riga di un qualsiasi file di testo. La stringa seguente verrà scritta su due righe differenti:

```
STR1        ADDLW       PCL,F
            RETLW       'H'
            RETLW       'E'
            RETLW       'L'
            RETLW       'L'
            RETLW       'O'
            RETLW       ' '
            RETLW       'W'
            RETLW       'O'
            RETLW       'R'
            RETLW       'L'
            RETLW       'D'
            RETLW       13
            RETLW       10
            RETLW       'P'
            RETLW       'I'
            RETLW       'C'
            RETLW       '1'
            RETLW       '6'
            RETLW       'F'
            RETLW       '6'
            RETLW       '2'
            RETLW       '8'
            RETLW       ' '
            RETLW       'H'
            RETLW       'E'
            RETLW       'R'
            RETLW       'E'
            RETLW       '.'
            RETLW       0
```

E' naturalmente possibile, e utile, prevedere una piccola subroutine apposta per andare a caporiga, magari chiamata proprio CRLF:

```
CRLF        MOVLW       13
            MOVWF       DL
```

```
        CALL        TX00
        MOVLW       10
        MOVWF       DL
        CALL        TX00
        RETURN
```

Scrittura di un numero binario a 8 bit

Per scrivere un numero binario si devono inviare 8 caratteri «0» o «1» a seconda che il bit corrispondente sia 0 o 1. La subroutine seguente va chiamata caricando in EL il valore da visualizzare. Per 8 volte si prepara in W il codice dello «0» (48), si controlla il valore del bit più significativo di EL e eventualmente si aggiunge 1 a W se risulta settato. Il codice, 48 o 49, corrispondente ai caratteri «0» e «1», viene quindi passato alla subroutine di trasmissione TX00 attraverso il solito registro DL. IL registro CH ha la funzione di contatore dei bit/caratteri.

Alla routine si può passare un valore a 8 bit compreso tra 0 e 255, che verrà visualizzato in forma binaria da 00000000 a 11111111.

```
WRBIN       MOVLW       8
            MOVWF       CH          ;conteggio bit/caratteri = 8
WRBIN2      MOVLW       '0'         ;W=codice dello "0"
            BTFSC       EL,7        ;controlla bit piu' significativo
            ADDLW       1           ;se settato somma 1 al codice
            MOVWF       DL
            CALL        TX00        ;trasmetti carattere
            RLF         EL,F        ;ruota a sinistra EL
            DECFSZ      CH,F        ;decrementa conteggio bit/caratteri
            GOTO        WRBIN2      ;se non zero torna a WRBIN2
            RETURN                  ;fine subroutine
```

Scrittura di un numero esadecimale a 8 bit

La scrittura di un numero in formato esadecimale si compone di due fasi, nella prima vanno considerati i 4 bit più significativi del valore per ricavare il primo digit, poi i 4 meno significativi. La subroutine seguente va chiamata caricando in EL il valore da visualizzare. I due nibbles (gruppi di 4 bit) del valore vengono scambiati (SWAP) e i 4 bit più significativi vengono azzerati con l'AND 0FH. Si chiama WRHDIG che somma il valore 0..15 del nibble al codice dello «0». Si controlla che il valore ottenuto non superi il 57 (che è il codice del carattere «9»), nel qual caso si aggiunge 7 per arrivare ai codici delle lettere maiuscole, che nella tabella ASCII iniziano 7 valori dopo la fine dei codici dei numeri.

Una volta scritto il primo digit vengono considerati i 4 bit meno significativi del valore originario e si applica lo stesso procedimento. Alla routine si può passare un valore a 8 bit compreso tra 0 e 255, che verrà visualizzato in forma esadecimale da 00 a FF.

```
WRHEX       SWAPF       EL,W          ;scambia i nibbles
            ANDLW       0FH           ;azzera i 4 bit superiori
            CALL        WRHDIG        ;scrivi il primo digit
            MOVF        EL,W          ;riprendi valore EL
            ANDLW       0FH           ;azzera 4 bit superiori
WRHDIG      MOVWF       DH            ;salva in DH
            MOVLW       '0'           ;prepara in W il codice dello "0"
            ADDWF       DH,F          ;sommalo a DH
            MOVF        DH,W
            SUBLW       57            ;controlla se risultato>57
            MOVLW       7
            BTFSS       STATUS,C      ;se no skip
            ADDWF       DH,F          ;altrimenti somma 7
            MOVF        DH,W          ;metti il codice in W
            MOVWF       DL
            CALL        TX00          ;trasmetti carattere
            RETURN                    ;fine subroutine
```

Seguendo attentamente le istruzioni si può notare un uso un po'strano della CALL, infatti la subroutine ad un certo punto richiama come ulteriore subroutine un pezzo di se stessa. Questo è un piccolo trucco usabile in assembly per ridurre di qualche istruzione la lunghezza del programma, nei particolari casi come questo in cui esistono delle istruzioni da eseguire più volte e che l'ultima volta coincidono con una semplice continuazione lineare del programma. La RETURN finale in questo caso rimanda la prima volta alla quinta istruzione della subroutine WRHEX, e la seconda volta ritorna al chiamante della WRHEX. La forma «classica estesa» sarebbe stata la seguente di identico funzionamento:

```
WRHEX       SWAPF       EL,W          ;scambia i nibbles
            ANDLW       0FH           ;azzera i 4 bit superiori
            CALL        WRHDIG        ;scrivi il primo digit
            MOVF        EL,W          ;riprendi valore EL
            ANDLW       0FH           ;azzera 4 bit superiori
            CALL        WRHDIG        ;scrivi il secondo digit
            RETURN

WRHDIG      MOVWF       DH            ;salva in DH
            MOVLW       '0'           ;prepara in W il codice dello "0"
            ADDWF       DH,F          ;sommalo a DH
            MOVF        DH,W
            SUBLW       57            ;controlla se risultato>57
            MOVLW       7
            BTFSS       STATUS,C      ;se no skip
            ADDWF       DH,F          ;altrimenti somma 7
            MOVF        DH,W          ;metti il codice in W
            MOVWF       DL
            CALL        TX00          ;trasmetti carattere
            RETURN                    ;fine subroutine
```

Uno stack per i dati

Se si guardano le ultime due subroutines e gli esempi con la trasmissione seriale software (senza usare la USART interna), si può notare che per ogni subroutine occorre definire un certo numero di registri di lavoro. Per esempio quella per scrivere i numeri binari richiede un registro EL in cui riceve il valore, un CH come contatore

dei caratteri, e usa DL per passare i codici ASCII alla subroutine di trasmissione. Quest'ultima riceve il valore in DL, e usa BL e CL rispettivamente come contatore dei bit e contatore per i cicli di ritardo.
La subroutine per scrivere in esadecimale usa invece un registro di lavoro chiamato DH. Se li contiamo siamo già a 6 diversi registri, la cosa non è grave visto che possiamo definirne tanti di nome diverso quante sono le celle della RAM disponibili (Nel PIC16F628 sono 96 solo nel banco 0), ma se si usano molte subroutines che si chiamano l'un l'altra è facile perdere di vista quali registri vengono usati in una e quali nell'altra. Per esempio può venire comodo chiamare sempre con il nome CL un registro che serve per fare un conteggio, o DL quello in cui impostare un valore da elaborare. Il problema è che se non si fa attenzione una subroutine chiamata potrebbe alterare il valore di un registro usato anche nella routine chiamante producendo risultati imprevedibili.

Le possibilità sono soltanto due, o si fa estrema attenzione a dare un nome univoco ad ogni registro usato in ciascuna subroutine, o ogni subroutine deve incaricarsi di salvare da qualche parte i valori dei registri «comuni» e ripristinarne il valore alla fine.

Nei microprocessori più evoluti lo stack può essere usato proprio per questo scopo. Solitamente dispongono di pochi registri da usare per tutto il programma, e pertanto viene comodo salvarne il contenuto (con operazioni di PUSH) e recuperarlo successivamente (con operazioni di POP). Questo modo di procedere permette di evitare molte «interferenze» tra le diverse parti del programma, e se non ci sono vincoli strettissimi di velocità da rispettare è un buon sistema di programmare.

Sfortunatamente sui PIC lo stack può contenere solo gli indirizzi di ritorno delle subroutine, è gestito in hardware ed è molto piccolo. Esiste però la possibilità di simulare uno stack dati usando una porzione della memoria RAM a cui accedere tramite il registro puntatore FSR.

Nel registro FSR si può scrivere l'indirizzo di una cella della RAM, a cui si può accedere (per leggerla o scriverci) tramite il registro INDF. Ad esempio:

```
        MOVLW   32          ;indirizzo inizio area dati
        MOVWF   FSR         ;lo scrive nel puntatore
        MOVF    INDF,W      ;legge il contenuto della cella
                            ;puntata da FSR e lo mette in W

        MOVLW   32          ;indirizzo inizio area dati
        MOVWF   FSR         ;lo scrive nel puntatore
        CLRF    INDF        ;azzera la cella puntata da FSR
```

Per creare uno stack dati simulato a software si possono riservare ad esempio i primi 16 byte della RAM per questo scopo, e puntare al primo di essi tramite FSR.

```
        ORG     32          ;indirizzo inizio area dati
S_STACK RES     16          ;16 byte usati come stack simulato

        ORG     0           ;indirizzo inizio programma
```

```
MOVLW       S_STACK      ;W=indirizzo di partenza stack
MOVWF       FSR          ;FSR punta all'inizio dello stack
```

Supponiamo di voler salvare il valore di un generico registro chiamato DL, richiamare una subroutine, e successivamente recuperare il valore di DL originale:

```
MOVF        DL,W         ;W = valore registro DL
MOVWF       INDF         ;Lo scrive nella cella puntata da FSR
INCF        FSR,F        ;Incrementa indirizzo puntato da FSR

CALL        ........    ;chiama la subroutine che può
                         ;tranquillamente modificare il valore di D


DECF        FSR,F        ;Decrementa indirizzo puntato da FSR
MOVF        INDF,W       ;Legge valore della cella puntata da FSR
MOVWF       DL           ;Lo riscrive in DL
```

Certo si sarebbe anche potuto salvare DL in un altro registro con un altro nome, ma una struttura come lo stack permette di riutilizzare per compiti diversi sempre le stesse celle di RAM e soprattutto senza bisogno di definire nuovi nomi e e senza assegnare loro un compito che viene svolto magari una sola volta in tutto il programma. Semplicemente con quelle prime 3 istruzioni «salviamo» il nostro registro lasciando il compito al puntatore FSR di tenere traccia di dove viene salvato, e quando ci riserve quel valore usiamo le ultime 3 istruzioni che lo recuperano.

E'ovviamente possibile effettuare più di un salvataggio uno dietro l'altro (in questo caso fino a riempire tutti i 16 byte dello stack), ricordando però che il recupero dei dati avviene sempre in ordine inverso. Lo stack infatti è una struttura «LIFO» (last in first out), come una catasta di piatti, l'ultimo piatto che si appoggia sulla cima della catasta è il primo che si riprende. Quindi se si salvano in sequenza i registri AL BL CL DL, si recupera per primo il valore di DL per finire con AL.

Da questo si può anche capire che alla fine il numero dei recuperi deve essere uguale a quello dei salvataggi, altrimenti si rischia di lasciare qualche dato nello stack, o far sconfinare l'indice dello stack al di fuori dello spazio previsto ottenendo i soliti risultati imprevedibili.

Usando poi la direttiva MACRO è anche possibile racchiudere i gruppi di istruzioni di salvataggio/recupero in due comodi nomi facili da ricordare, per esempio PUSH e POP come negli assembly di micro più evoluti:

```
PUSH        MACRO        registro
            MOVF         registro,W
            MOVWF        INDF
            INCF         FSR,F
            ENDM

POP         MACRO        registro
            DECF         FSR,F
            MOVF         INDF,W
```

```
            MOVWF       registro
            ENDM
```

Con queste due macro è possibile salvare un registro semplicemente scrivendo: PUSH nomeregistro, e recuperarne il valore con POP nomeregistro. Bisogna comunque ricordare che le PUSH e POP scritte nel programma sono delle macro, e quindi al momento della compilazione verranno automaticamente sostituite con le istruzioni corrispondenti. Ciascuna PUSH e POP occupa perciò lo spazio di 3 istruzioni, ed è solo una forma mnemonica abbreviata per identificarle. **Inoltre ricordare sempre che ad ogni PUSH e POP viene alterato il valore dell'accumulatore W e del flag Z!**

Per tornare alla visualizzazione su terminale supponiamo allora di voler usare la subroutine di trasmissione software e voler usare meno nomi possibili per i registri di lavoro salvandoli sullo stack quando necessario.
Vogliamo scrivere in binario e in esadecimale il valore 207 separati da due spazi. Il seguente programma completo svolge questa funzione e usa lo stack simulato per salvare il valore di alcuni registri prima di riutilizzarli all'interno delle subroutines.

```
            PROCESSOR   16F628
            RADIX       DEC
            INCLUDE     "P16F628.INC"
            __CONFIG    11110100010000B
            ORG         32
S_STACK     RES         16          ;16 byte usati come stack simulato
BL          RES         1
CL          RES         1
DL          RES         1
#DEFINE     TXOUT       PORTB,2     ;Uscita seriale
;-----------------------------------------------------
PUSH        MACRO       registro
            MOVF        registro,W
            MOVWF       INDF
            INCF        FSR,F
            ENDM

POP         MACRO       registro
            DECF        FSR,F
            MOVF        INDF,W
            MOVWF       registro
            ENDM
;-----------------------------------------------------
            ORG         0
            BSF         STATUS,RP0  ;Attiva banco 1
            BCF         TRISB,2     ;PORTB=uscita
            BCF         STATUS,RP0  ;Ritorna al banco 0
            MOVLW       7
            MOVWF       CMCON       ;PORTA=I/O digitali

            MOVLW       S_STACK     ;W=indirizzo di partenza stack
            MOVWF       FSR         ;FSR punta all'inizio dello stack

            MOVLW       207
            MOVWF       DL
            CALL        WRBIN       ;scrive 207 in binario
            MOVLW       ' '
            MOVWF       DL
            CALL        TX00        ;scrive uno spazio
            CALL        TX00        ;un altro spazio
```

```
            MOVLW       207
            MOVWF       DL
            CALL        WRHEX       ;scrive 207 in esadecimale

            SLEEP                   ;Stop programma
;-----------------------------------------------------
; Scrive numero in binario 8 bit (8 digit)
; DL=valore da scrivere. BL=copia di DL.
; CL=contatore cicli. Non altera i registri.
;-----------------------------------------------------
WRBIN       PUSH        BL
            PUSH        CL
            PUSH        DL
            MOVF        DL,W
            MOVWF       BL          ;copia DL in BL
            MOVLW       8
            MOVWF       CL          ;conteggio bit/caratteri = 8
WRBIN2      MOVLW       '0'         ;W=codice dello "0"
            MOVWF       DL
            BTFSC       BL,7        ;controlla bit piu' significativo
            INCF        DL,F        ;se settato somma 1 al codice
            CALL        TX00        ;trasmetti carattere
            RLF         BL,F        ;ruota a sinistra BL
            DECFSZ      CL,F        ;decrementa conteggio bit/caratteri
            GOTO        WRBIN2      ;se non zero torna a WRBIN2
            POP         DL
            POP         CL
            POP         BL
            RETURN                  ;fine subroutine
;-----------------------------------------------------
; Scrive numero in esadecimale 8 bit (2 digit)
; DL=valore da scrivere.  Non altera i registri.
;-----------------------------------------------------
WRHEX       PUSH        DL          ;salva 2 copie di DL
            PUSH        DL
            SWAPF       DL,W        ;scambia i nibbles
            ANDLW       0FH         ;azzera i 4 bit superiori
            MOVWF       DL
            CALL        WRHDIG      ;scrivi il primo digit
            POP         DL          ;riprendi copia di DL
            MOVLW       0FH
            ANDWF       DL,F        ;azzera 4 bit superiori
            CALL        WRHDIG      ;scrivi il secondo digit
            POP         DL
            RETURN

WRHDIG      MOVLW       '0'         ;prepara in W il codice dello "0"
            ADDWF       DL,F        ;sommalo a DL
            MOVF        DL,W
            SUBLW       57          ;controlla se risultato>57
            MOVLW       7
            BTFSS       STATUS,C    ;se no skip
            ADDWF       DL,F        ;altrimenti somma 7
            CALL        TX00        ;trasmetti carattere
            RETURN
;-----------------------------------------------------
; Trasmissione seriale di un byte in formato
; 9600 8-N-1 dal pin "TXOUT" (per clock 4MHz)
; DL=valore da trasmettere, BL=contatore dei bit,
; CL=contatore cicli ritardo. Non altera i registri.
;-----------------------------------------------------
TX00        PUSH        BL
            PUSH        CL
            PUSH        DL
            MOVLW       10
            MOVWF       BL          ;Contatore dei bit=10
            BCF         STATUS,C    ;Azzera flag C
TX01        BTFSS       STATUS,C    ;Se flag C=1 skip
            GOTO        TX02        ;altrimenti salta a TX02
```

```
            NOP
            BSF         TXOUT       ;Manda a 1 il pin TXOUT
            GOTO        TX03        ;e salta a TX03
TX02        BCF         TXOUT       ;Manda a 0 il pin TXOUT
            NOP
            NOP
TX03        MOVLW       30
            MOVWF       CL          ;CL=30, ritardo durata bit
            DECFSZ      CL,F        ;Decrementa CL, skip se zero
            GOTO        $-1         ;Altrimenti nuovo decremento
            BSF         STATUS,C    ;Setta flag C
            RRF         DL,F        ;Ruota a destra DL
            DECFSZ      BL,F        ;Decrem.contat.bit, skip se 0
            GOTO        TX01        ;Altrimenti torna a TX01
            POP         DL
            POP         CL
            POP         BL
            RETURN
;-------------------------------------------------
            END
```

In questo programma si possono notare alcune cose. Intanto il salvataggio dei registri all'inizio di ogni subroutine e il ripristino alla fine, questo permette ad esempio di impostare il codice in DL e, senza rischiare che il suo valore cambi, richiamare due o piu' volte di fila la TX00 (cosa che viene fatta nella sezione principale del programma quando si scrivono i due spazi di fila). Poi si può notare che il recupero dei dati con le POP avviene in ordine inverso a quello della PUSH. Ed infine nella subroutine WRHEX si nota che DL viene salvato due volte. Questo è perfettamente lecito, e in questo caso permette di evitare l'uso di un registro di copia temporanea.

E' pure possibile usare lo stack per scambiare il valore tra due registri, anche se è una pessima scelta come velocità di esecuzione e occupazione di memoria programma (12 istruzioni effettive quando uno scambio si può fare con 4), ma serve ad illustrare le possibilità che si hanno a disposizione con uno stack. Alla fine CL e DL si sono scambiati i valori:

```
            PUSH        CL
            PUSH        DL
            POP         CL
            POP         DL
```