Università Politecnica Delle Marche

FACOLTÀ DI INGEGNERIA ELETTRONICA

DII Dipartimento Ingegneria Informazione

RELAZIONE PROGETTO

Architetture e Programmazione di Sistemi Digitali

Studente	**Matricola**
Owabor Vivien	1103872
Rossetti Mario	1103356

Docente

Falaschetti Laura

Architetture e Programmazione di Sistemi Digitali,
vengono studiate le fondamentali architetture di microcontrollori
e gli elementi di programmazione in linguaggio assembly.

Indice

1 Relazione progetto 7
1.1 Obiettivo 7
2 Introduzione 7
2.1 Linguaggio Assembly 7
2.2 MPLAB X IDE 7
2.3 Hardware PIC16F887 8
3 Programma Assembly per PIC 9
3.1 Struttura base 9
3.2 Direttive 9
3.2.1 PSECT 9
3.2.10 EQU 11
3.2.11 END 11
3.2.2 UDATA 10
3.2.3 DS 10
3.2.4 UDATA_SHR 10
3.2.5 BANKSEL 10
3.2.6 PAGESEL 10
3.2.7 PROCESSOR 10
3.2.8 #include 10
3.2.9 CONFIG 10
3.3 Set di istruzioni 11
3.3.1 Istruzioni Byte-oriented 11
3.3.2 Istruzioni Bit-oriented 12
3.3.3 Istruzioni su costante e di controllo 12
3.4 Indirizzamento indiretto 12
4. Diagramma di flusso 13
5. Il nostro codice 15
5.1 Commento iniziale 15
5.2 Primo blocco di codice 15
5.3 Secondo blocco di codice 16
5.4 Terzo blocco di codice 17
5.5 Quarto blocco di codice 18
5.6 Quinto blocco di codice 21
5.7 Sesto blocco di codice 22
5.8 Settimo blocco di codice 24
5.9 Ottavo blocco di codice 26

1 Relazione progetto

1.1 Obiettivo

Realizzare un firmware che riceva dal computer tramite porta seriale (USART) una parola, come sequenza di codici ASCII dei singoli caratteri. La parola è terminata da un punto ed è di lunghezza massima fissata a priori. Dopo aver ricevuto la parola, il programma deve convertire tutti i caratteri in minuscolo (solo quelli nel range 'A'..'Z') e reinviarla sulla porta seriale (USART).

2 Introduzione

2.1 Linguaggio Assembly

L'Assembly è un linguaggio di programmazione a basso livello che, analogamente ai linguaggi ad alto livello, richiede un processo di traduzione. A differenza di questi ultimi, l'Assembly consente una traduzione particolarmente semplice che trasforma ogni istruzione di codice, in modo univoco, in un'istruzione in linguaggio macchina.

In questo progetto il nostro compito è quello di sviluppare un codice Assembly al PC in ambiente MPLAB X IDE per PIC16F887 seguendo specifiche assegnate dal docente.

2.2 MPLAB X IDE

La realizzazione di un programma per il PIC di solito si compone delle seguenti fasi:

1. Scrittura del programma in linguaggio Assembly;
2. Compilazione del programma per generare un eseguibile in linguaggio macchina (file Hex);
3. Simulazione e debugging del programma con un opportuno programma di simulazione;
4. Scrittura dell'eseguibile della memoria FLASH del PIC (questa fase viene detta programmazione).

Le prime tre fasi dello sviluppo di un programma possono essere svolte ricorrendo a un ambiente integrato di sviluppo (IDE) espressamente dedicato al PIC.
La casa costruttrice Microchip fornisce un pacchetto integrato chiamato MPLAB®X IDE, liberamente scaricabile.
Esso comprende:

- **MPLAB Editor**: editor di testo per scrivere il testo del programma;

- **MPLAB XC8 PIC-AS**: il compilatore che traduce il testo in codice eseguibile.

2.3 Hardware PIC16F887

Un microcontrollore (MCU) è un microcalcolatore integrato su un singolo chip. Esso è utilizzato principalmente per realizzare sistemi embedded ovvero sistemi di controllo digitale.
I PIC (Peripheral Interface Controller) costituiscono una delle famiglie più diffuse e usate di microcontrollori. Si tratta di microcontrollori con architettura Harvard prodotti da Microchip Technology. Sono molto popolari sia in ambito industriale che nelle applicazioni hobbistiche per il basso costo e la grande disponibilità di software e di strumenti di sviluppo disponibili.

Con questo firmware viene utilizzato il PIC16F887, di cui i primi due digit identificano la famiglia di microcontrollori, la 'F' indica l'inserimento di una Flash memory technology e gli altri caratteri in coda corrispondono ad una sigla numerica.

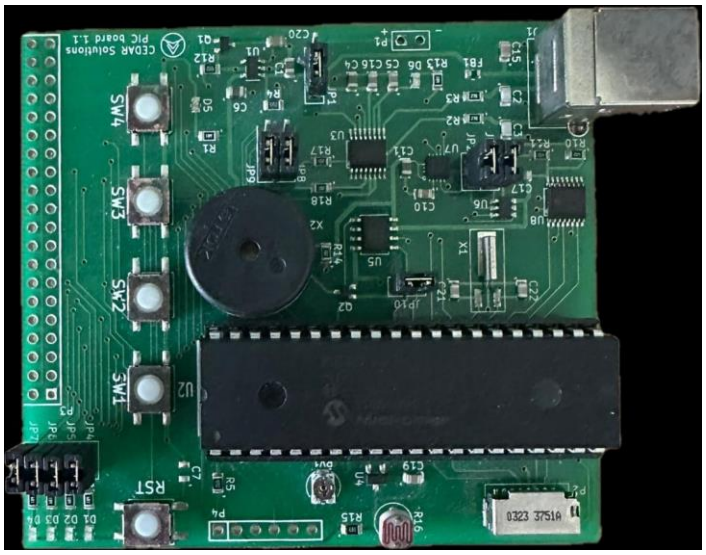

Figura 1: PIC Board

3 Programma Assembly per PIC

3.1 Struttura base

Il nostro codice segue la seguente struttura tipica:

1) un **commento** iniziale[1];
2) una direttiva **PROCESSOR**;
3) una direttiva **#include**;
4) una direttiva **CONFIG**;
5) una o più direttive **EQU**;
6) una o più direttive **PSECT**;
7) una sequenza di **istruzioni**;
8) un'istruzione **GOTO**;
9) una direttiva **END.**

3.2 Direttive

Le direttive sono comandi diretti all'Assembler[2], inserite nel codice al posto delle istruzioni Assembly vere e proprie. In sostanza esse non sono caricate sul PIC, non producono codice eseguibile e non occupano memoria sul microcontrollore.

3.2.1 PSECT

Tutto il codice del programma deve essere inserito in una sezione utilizzando la direttiva PSECT.

Questa direttiva può essere utilizzata senza dover specificare alcun flag.

In alternativa, è possibile definire una propria sezione personale con qualsiasi nome e flag adatti. Se specificato un indirizzo, la sezione inizia a tale indirizzo (codice non rilocabile), altrimenti la sezione viene allocata automaticamente dal linker[3] (codice rilocabile).

[1] Il commento iniziale serve per descrivere brevemente il programma, fornire informazioni sull'autore, sulla data di scrittura, sulle successive versioni etc. I commenti sono costituiti da tutto ciò che segue ';' o '//' e vengono ignorati dall'Assembler.
[2] È un programma che trasforma le istruzioni mnemoniche dell'Assembly in linguaggio macchina.
[3] È un programma che effettua il collegamento tra il programma oggetto, cioè la traduzione del codice sorgente in linguaggio macchina, e le librerie del linguaggio necessarie per l'esecuzione del programma.

3.2.2 UDATA
La direttiva UDATA indica l'inizio di una sezione dati, in cui è possibile riservare locazioni di memoria RAM[4] (Random Access Memory) da utilizzare come variabili. Se specificato un indirizzo, la sezione inizia a tale indirizzo (codice non rilocabile), altrimenti la sezione viene allocata automaticamente dal linker (codice rilocabile).

3.2.3 DS
La direttiva DS riserva dimension byte nella zona UDATA.

3.2.4 UDATA_SHR
Come la direttiva UDATA, ma il linker alloca la sezione dati in una porzione di RAM condivisa tra tutti i banchi. In questo modo le variabili definite possono essere indirizzate senza dover commutare il banco RAM.

3.2.5 BANKSEL
La direttiva BANKSEL genera automaticamente il codice necessario per commutare il banco di RAM a quello in cui si trova l'indirizzo associato alla label indicata.

3.2.6 PAGESEL
La direttiva PAGESEL genera automaticamente il codice necessario per commutare la pagina di ROM[5] (Read Only Memory) a quella in cui si trova l'indirizzo associato alla label indicata. Da utilizzare con le istruzioni CALL e GOTO.

3.2.7 PROCESSOR
Direttiva che definisce il tipo di processore.

3.2.8 #include
La direttiva #include permette di includere un file sorgente aggiuntivo.

3.2.9 CONFIG
I PIC dispongono di un registro di configurazione hardware, che viene scritto una sola volta al momento della programmazione, e che stabilisce il funzionamento di alcuni circuiti interni. Ogni programma per PIC inizia con una intestazione in cui si dichiara, oltre al tipo di micro usato e al formato di default dei numeri (decimale, esadecimale ecc..), anche la Configuration Word che ne determinerà il

[4] È un tipo di memoria volatile in cui vengono immagazzinate le informazioni di cui un programma ha bisogno durante l'esecuzione.
[5] È un tipo di memoria non volatile.

funzionamento.
La direttiva CONFIG serve per settare correttamente i bit della Configuration Word.

3.2.10 EQU
La direttiva EQU permette di definire un valore costante (noto a compile time).

3.2.11 END
La direttiva END segnala all'assemblatore la fine del programma.

3.3 Set di istruzioni
Si possono suddividere in:

- **Byte-oriented**:
  'f' rappresenta il file register a cui è applicata l'istruzione;
  'd' specifica la destinazione del risultato dell'operazione che rappresenta l'istruzione.
- **Bit-oriented**:
  'f' specifica il file register a cui appartengono i bit.
  'b' seleziona i bit coinvolti nell'operazione;
- **Literal & Control**:
  'k' rappresenta il valore in bit su cui agisce.

3.3.1 Istruzioni Byte-oriented
La figura 2 illustra le operazioni di registro file orientate ai byte[6] su W (Working Register[7]) o indirizzo f (File Register).

Mnemonic, Operands		Description	Cycles	14-Bit Opcode MSb	LSb	Status Affected	Notes
BYTE-ORIENTED FILE REGISTER OPERATIONS							
ADDWF	f, d	Add W and f	1	00 0111 dfff ffff		C, DC, Z	1, 2
ANDWF	f, d	AND W with f	1	00 0101 dfff ffff		Z	1, 2
CLRF	f	Clear f	1	00 0001 lfff ffff		Z	2
CLRW	–	Clear W	1	00 0001 0xxx xxxx		Z	
COMF	f, d	Complement f	1	00 1001 dfff ffff		Z	1, 2
DECF	f, d	Decrement f	1	00 0011 dfff ffff		Z	1, 2
DECFSZ	f, d	Decrement f, Skip if 0	1(2)	00 1011 dfff ffff			1, 2, 3
INCF	f, d	Increment f	1	00 1010 dfff ffff		Z	1, 2
INCFSZ	f, d	Increment f, Skip if 0	1(2)	00 1111 dfff ffff			1, 2, 3
IORWF	f, d	Inclusive OR W with f	1	00 0100 dfff ffff		Z	1, 2
MOVF	f, d	Move f	1	00 1000 dfff ffff		Z	1, 2
MOVWF	f	Move W to f	1	00 0000 lfff ffff			
NOP	–	No Operation	1	00 0000 0xx0 0000			
RLF	f, d	Rotate Left f through Carry	1	00 1101 dfff ffff		C	1, 2
RRF	f, d	Rotate Right f through Carry	1	00 1100 dfff ffff		C	1, 2
SUBWF	f, d	Subtract W from f	1	00 0010 dfff ffff		C, DC, Z	1, 2
SWAPF	f, d	Swap nibbles in f	1	00 1110 dfff ffff			1, 2
XORWF	f, d	Exclusive OR W with f	1	00 0110 dfff ffff		Z	1, 2

Figura 2: Istruzioni Byte-oriented

[6] Un byte è composto da 8 bit.
[7] Consiste in un'area di memoria interna alla CPU in cui un dato è temporaneamente salvato, per essere usato come input di un altro blocco o come risultato, oppure per trasferire dati da e verso la memoria.

3.3.2 Istruzioni Bit-oriented

La figura 3 illustra le operazioni di registro file orientate ai bit su indirizzo f.

BIT-ORIENTED FILE REGISTER OPERATIONS						
BCF	f, b	Bit Clear f	1	01 00bb bfff ffff		1, 2
BSF	f, b	Bit Set f	1	01 01bb bfff ffff		1, 2
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	01 10bb bfff ffff		3
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	01 11bb bfff ffff		3

Figura 3: Istruzioni Bit-oriented

3.3.3 Istruzioni su costante e di controllo

La figura 4 illustra le operazioni letterali e di controllo.

LITERAL AND CONTROL OPERATIONS					
ADDLW	k	Add literal and W	1	11 111x kkkk kkkk	C, DC, Z
ANDLW	k	AND literal with W	1	11 1001 kkkk kkkk	Z
CALL	k	Call Subroutine	2	10 0kkk kkkk kkkk	
CLRWDT	–	Clear Watchdog Timer	1	00 0000 0110 0100	$\overline{TO}$, $\overline{PD}$
GOTO	k	Go to address	2	10 1kkk kkkk kkkk	
IORLW	k	Inclusive OR literal with W	1	11 1000 kkkk kkkk	Z
MOVLW	k	Move literal to W	1	11 00xx kkkk kkkk	
RETFIE	–	Return from interrupt	2	00 0000 0000 1001	
RETLW	k	Return with literal in W	2	11 01xx kkkk kkkk	
RETURN	–	Return from Subroutine	2	00 0000 0000 1000	
SLEEP	–	Go into Standby mode	1	00 0000 0110 0011	$\overline{TO}$, $\overline{PD}$
SUBLW	k	Subtract W from literal	1	11 110x kkkk kkkk	C, DC, Z
XORLW	k	Exclusive OR literal with W	1	11 1010 kkkk kkkk	Z

Figura 4: Istruzioni su costante e di controllo

3.4 Indirizzamento indiretto

Le istruzioni del PIC16 permettono soltanto un accesso diretto alla memoria, cioè specificando un indirizzo esplicito.

In molti casi è necessario un accesso indiretto alla memoria, in cui l'indirizzo non è noto a priori, ma a sua volta contenuto in memoria.

Il PIC16 mette a disposizione due registri per realizzare questa funzione:

- **FSR** (File Select Register): contiene l'indirizzo della locazione RAM da puntare, quindi funge da puntatore.
- **INDF** (Indirect File Register): non è un registro fisico, ma leggendo e scrivendo tale registro si accede in realtà alla locazione il cui indirizzo è contenuto in FSR.

4 Diagramma di flusso

I diagrammi di flusso sono una rappresentazione in forma di grafo del flusso dell'esecuzione di un programma.

L'esecuzione dei programmi Assembly consiste in una successione di trasformazioni dello stato delle memorie della RAM e del Program Counter[8] (PC).

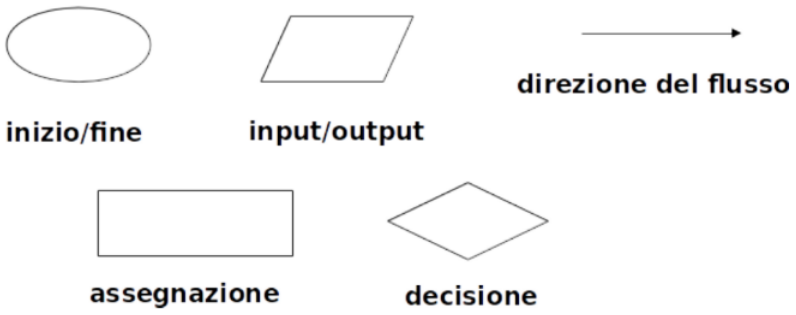

Figura 5: Legenda dei diagrammi di flusso

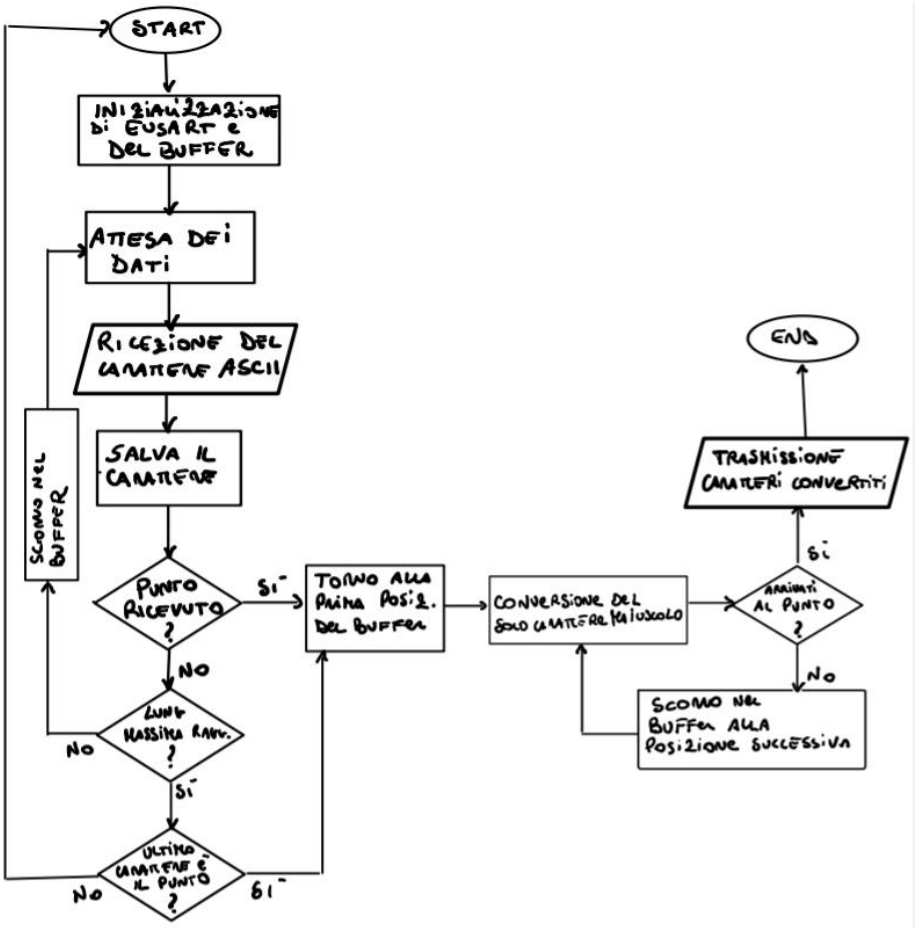

Figura 6: Flow chart del nostro progetto

[8] Registro speciale che contiene l'indirizzo di memoria della successiva istruzione da eseguire.

Il nostro progetto, il cui diagramma di flusso è in figura 6, è sviluppato nel seguente modo:

1. **Start**: inizio del programma;
2. **Inizializzazione dell'EUSART[9] e del buffer**: per gestire le comunicazioni I/O seriali e memorizzare i caratteri;
3. **Attesa dei dati**;
4. **Ricezione del carattere ASCII**;
5. **Salva il carattere**;
6. **Punto ricevuto?**
   -Sì, vado al punto 7;
   -No, vado al punto 12;
7. **Torno alla prima posizione del buffer**;
8. **Conversione del solo carattere maiuscolo**;
9. **Arrivati al punto?**
   -Sì, vado al punto 10;
   -No, vado al punto 11;
10. **Trasmissione dei caratteri convertiti** e successivamente termino il programma;
11. **Scorro nel buffer alla posizione successiva** ed in seguito torno al punto 8;
12. **Lunghezza massima raggiunta?**
    -Sì, vado al punto 13;
    -No, vado al punto 14;
13. **Ultimo carattere è il punto?**
    -Sì, vado al punto 7;
    -No, comincia nuovamente il programma;
14. **Scorro nel buffer** e successivamente torno al punto 3;
15. **End**: Fine del programma.

[9] L'Enhanced/Addressable Universal Asynchronous Receiver Transceiver (EUSART/AUSART) è una periferica per la gestione delle comunicazioni I/O seriali. Contiene tutti i generatori di clock, i registri a scorrimento e i buffer di dati necessari per eseguire un trasferimento di dati seriali in ingresso o in uscita indipendentemente dall'esecuzione del programma principale. Esso è noto anche come Serial Communications Interface (SCI).

5 Il nostro codice

È fondamentale leggere anche le immagini dei blocchi di codice per capire a meglio come è stato raggiunto l'obiettivo posto dalla traccia.

5.1 Commento iniziale

Come detto in precedenza, il codice inizia con un commento che serve a descrivere l'obiettivo del programma.

```
;-------------------------------------------------------------------------------
;
;   Esame di Architetture e Programmazione di Sistemi Digitali/Sistemi Elettronici
;   Autori Vivien Owabor e Mario Rossetti
;   Corso Ingegneria Elettronica / Ingegneria Elettronica e delle Tecnologie Digitali
;   Università Politecnica delle Marche, Ancona
;
;   Data consegna progetto 26/11/2024
;
;   Traccia 56:
;   Si realizzi un firmware che riceva dal computer tramite porta seriale (USART) una
;   parola, come sequenza di codici ASCII dei singoli caratteri. La parola è terminata da
;   un punto ed è di lunghezza massima fissata a priori. Dopo aver ricevuto la parola, il
;   programma deve convertire tutti i caratteri in minuscolo (solo quelli nel range 'A'..'Z')
;   e reinviarla sulla porta seriale (USART).
;
;-------------------------------------------------------------------------------
```

Figura 7: Commento iniziale

In seguito, troviamo i nostri blocchi di codice.

5.2 Primo blocco di codice

Il primo blocco di codice espone il tipo di processore usato, i file da includere, le allocazioni della memoria e configura i bit di configurazione.

Per approfondire:

- il **Watchdog Timer** è un timer interno che può resettare il microcontrollore in caso di blocco;
- il **Brown-out Reset** è un sistema che resetta il microcontrollore se la tensione di alimentazione scende sotto una soglia critica;
- il **Power-up Timer** è un ritardo iniziale che consente di stabilizzare l'alimentazione prima di avviare il microcontrollore;
- il **Program Flash Memory** (PFM) è un tipo di memoria non volatile, ovvero mantiene i dati anche in assenza di alimentazione. Nei microcontrollori è utilizzata per salvare il codice sorgente compilato (il programma che il microcontrollore eseguirà).

```
;-------------------------------------------------------------------------------

PROCESSOR 16F887        ;direttiva che definisce il tipo di processore
#include <xc.inc>       ;file che contiene le definizioni dei simboli (nomi registri, nomi bit dei registri, ecc).
#include "macro.inc"    ;definizione di macro utili

;CONFIGURATION BITS
    CONFIG "FOSC = INTRC_NOCLKOUT"      ;configura l'oscillatore del microcontrollore per utilizzare il clock interno (INTRC)
                                        //il segnale di clock non viene inviato all'esterno (NOCLKOUT)
    CONFIG "CP = OFF"                   ;PFM (Program Flash Memory) and Data EEPROM code protection disabled
                                        //disabilita la protezione del codice memorizzato nel microcontrollore
                                        //il codice può essere letto esternamente
    CONFIG "CPD = OFF"                  ;Data memory code protection is disabled
                                        //disabilita la protezione del codice nella memoria dati EEPROM
                                        //la memoria può essere letta o scritta esternamente
    CONFIG "WDTE = OFF"                 ;disabilita il Watchdog Timer
    CONFIG "BOREN = OFF"                ;disabilita il Brown-out Reset
    CONFIG "PWRTE = OFF"                ;disabilita il Power-up Timer
    CONFIG "LVP = OFF"                  ;Low voltage programming disabled
                                        //disabilita la Low Voltage Programming, altrimenti il pin RB3 (porta B)
                                        //non può essere utilizzato come I/O generico.
                                        //(_LVP_ON -> RB3/PGM come PGM pin cioè per ICSP = In-Circuit Serial Programming)
    CONFIG "DEBUG = OFF"                ;Background debugger disabled
                                        //disabilita la modalità di debug del microcontrollore
```

Figura 8: Primo blocco di codice (parte 1)

```
;CONFIG2
    CONFIG  "BOR4V = BOR21V"    ;Brown-out Reset Selection bit (Configura la soglia del Brown-out Reset a 2,1 V)
    CONFIG  "WRT = OFF"         ;disabilita la protezione in scrittura della memoria Flash

;Variabili in RAM (shared RAM)
    PSECT udata_shr

tmp:
    DS 1    ;variabile temporanea per calcoli intermedi
```

Figura 9: Primo blocco di codice (parte 2)

5.3 Secondo blocco di codice

Il secondo blocco di codice rappresenta l'inizio del nostro programma:

- il **Reset Vector** è l'indirizzo di memoria in cui il microcontrollore cerca l'istruzione iniziale da eseguire quando viene avviato o resettato.
  Il flag delta utilizzato nella direttiva *PSECT* ed impostato uguale a 2 indica che l'indirizzamento della memoria utilizza 2 byte.

La direttiva *PAGESEL* e l'istruzione *GOTO* ci aiutano a saltare alla prima istruzione utile, ossia ***start***.

```
;--------------------------------INIZIO PROGRAMMA-----------------------------------
;Reset vector
    PSECT resetVec,class=CODE,delta=2

resetVec:
    PAGESEL start
    GOTO start
```

Figura 10: Secondo blocco di codice

5.4 Terzo blocco di codice

Il terzo blocco di codice rappresenta il nostro loop principale in cui vengono sviluppate le sezioni ***start***, ***RX*** e ***RIC***.

La sezione start richiama l'inizializzazione dell'hardware e del buffer, che saranno spiegati in seguito, ed effettua la pulizia dell'inizio del buffer con l'istruzione ***CLRF***.

```
59      ;------------------------------CICLO PRINCIPALE----------------
60      PSECT MainCode,global,class=CODE,delta=2    ;codice rilocabile
61
62      start:
63          PAGESEL INIT
64          CALL INIT       ;inizializzazione hardware
65          PAGESEL BUFF
66          CALL BUFF       ;inizializzazione del buffer
67          CLRF BUF_START  ;pulizia dell'inizio del buffer
68
69          ;va a capo
70          PAGESEL STAMPA_CAPO
71          CALL STAMPA_CAPO
72
73          ;stampa "SCRIVI" all'avvio
74          PAGESEL STAMPA_SCRIVI
75          CALL STAMPA_SCRIVI
```

Figura 11: Terzo blocco (parte 1)

Nella sezione RX avviene l'attesa dei dati in cui comincia la ricezione dei caratteri.

Il registro ***PIR1*** contiene i flag bits delle interruzioni periferiche. Questi ultimi sono settati quando occorre una interrupt condition, indipendentemente dallo stato del suo corrispondente enable bit o global enable bit.

Il bit test ***BTFSS*** avviene sul bit ***RCIF*** [10]del registro ***PIR1***.

```
77      RX:
78          ;ATTESA DEI DATI: inizio ricezione dei caratteri
79          BANKSEL PIR1                    ;il registro PIR1 contiene i flag bits delle interruzioni periferiche
80          BTFSS PIR1,PIR1_RCIF_POSITION   ;BTFSS è un bit test che controlla se è stato ricevuto un carattere
81          GOTO RX                         ;se RCIF=1, salta il GOTO e va in RIC, altrimenti continua ad aspettare
82
```

Figura 12: Terzo blocco (parte 2)

Nella sezione RIC viene sviluppata la ricezione del carattere.

Il bit test ***BTFSC*** avviene sul bit Zero del registro ***STATUS***:

- Il bit Zero (***Z***) viene settato quando un'operazione precedente ha avuto come risultato 0;

[10] EUSART Receive Interrupt Flag bit

- Il registro STATUS racchiude lo stato delle operazioni aritmetiche, la selezione del banco RAM e le informazioni dello stato del microcontrollore.

```
;RICEZIONE CARATTERE
RIC:
        BANKSEL RCREG   ;EUSART Receive Data Register
        MOVF RCREG,W    ;prendo il dato ricevuto dal registro e lo metto su W
        MOVWF TEMP      ;salvo il dato da W a TEMP

        ;SALVA IL CARATTERE nel buffer tramite indirizzamento indiretto
        MOVF TEMP,W ;copio TEMP in W
        MOVWF INDF  ;salva il carattere nel buffer tramite FSR

        ;visualizzazione parola immessa dall'utente
        BANKSEL TXREG   ;EUSART Transmit Data Register
        MOVF INDF,W     ;prendo il carattere dal buffer e lo metto su W
        MOVWF TXREG     ;carica il carattere nel registro di trasmissione per stamparlo a schermo

        ;CONTROLLO PUNTO: primo controllo per far partire la conversione prima della lunghezza massima
        ;prendo il carattere dal buffer e lo confronto con '.'
        MOVF INDF,W                     ;prendo il carattere dal buffer e lo metto su W
        MOVWF TEMP                      ;carica il carattere in TEMP
        MOVLW '.'                       ;metto il carattere '.' in W
        SUBWF TEMP,W                    ;sottrazione di W da TEMP, il risultato va in W.
                                        //se W=TEMP='.', Z diventa 1
        BTFSC STATUS,STATUS_Z_POSITION  ;BIT-TEST: se Z=0, salta la prossima istruzione e continua la ricezione,
                                        //altrimenti si va al GOTO CONVERT
        GOTO CONVERT                    ;sezione in cui avviene la conversione
```

Figura 13: Terzo blocco (parte 3)

```
          INCF FSR,F          ;incremento l'FSR per salvare il carattere nel buffer
          DECFSZ LUNG_MAX,F   ;decremento di 1 il valore della lunghezza massima
          GOTO RX             ;se LUNG_MAX è diverso da 0, vado a ricevere il prossimo carattere
          GOTO CONTROLLO      ;se LUNG_MAX è uguale a 0, vado a controllare il carattere appena immesso
```

Figura 14: Terzo blocco (parte 4)

5.5 Quarto blocco di codice

Nel quarto blocco vi è il codice del salvataggio dei caratteri nel buffer attraverso l'indirizzamento indiretto spiegato nel sottocapitolo 3.4.

Esso si suddivide in cinque sezioni: ***CONTROLLO***, ***CONVERT***, ***CICLO_CONVERT***, ***INCREMENTO*** e ***MAIUSCOLO***.

Nella sezione CONTROLLO si verifica che l'ultimo carattere immesso sia il punto.

```
;---------------------- SALVA NEL BUFFER (Indirizzamento indiretto) ------------------------------
CONTROLLO:
    DECF FSR,F  ;decremento l'FSR perchè si trova in una posizione successiva all'ultimo carattere

    ;CONTROLLO PUNTO: secondo controllo per verificare che l'ultimo carattere immesso sia il punto
    ;prendo il carattere dal buffer e lo confronto con '.'
    MOVF INDF,W                     ;prendo il carattere dal buffer e lo metto su W
    MOVWF TEMP                      ;carica il carattere in TEMP
    MOVLW '.'                       ;metto il carattere '.' in W
    SUBWF TEMP,W                    ;sottrazione di W da TEMP, il risultato va in W.
                                    //se W=TEMP='.', Z diventa 1
    BTFSC STATUS,STATUS_Z_POSITION  ;BIT-TEST: se Z=0 si va al GOTO start, altrimenti si va al GOTO CONVERT
    GOTO CONVERT                    ;sezione in cui avviene la conversione
    GOTO start
```

Figura 15: Quarto blocco (parte 1)

Nella sezione CONVERT riconfiguro il buffer e la lunghezza massima tramite la sezione BUFF che si trova più avanti.

```
CONVERT:
    PAGESEL BUFF
    CALL BUFF       ;riconfiguro buffer e lung_max
```

Figura 16: Quarto blocco (parte 2)

Nella sezione CICLO_CONVERT avviene la routine di conversione di tutti i caratteri tranne il punto.

Il bit test ***BTFSS*** avviene sul bit Carry del registro ***STATUS***. Quest'ultimo bit viene utilizzato per operazioni di addizione e sottrazione.

```
CICLO_CONVERT: ;ciclo di conversione di tutti i caratteri tranne il '.'

    ;CONTROLLO PUNTO: faccio la conversione fino al carattere prima del punto
    MOVF  INDF,W                    ;prendo il carattere dal buffer e lo metto su W
    MOVWF TEMP                      ;carica il carattere in TEMP
    MOVLW '.'                       ;metto il carattere '.' in W
    SUBWF TEMP,W                    ;sottrazione di W da TEMP, il risultato va in W.
                                    //se W=TEMP='.', Z diventa 1
    BTFSC STATUS,STATUS_Z_POSITION  ;BIT-TEST: se Z=0, salta la prossima istruzione e continua il ciclo di conversione,
                                    //altrimenti va al GOTO INIT_TRANS
    GOTO INIT_TRANS                 ;sezione in cui avviene il ciclo di trasmissione

    ;recupera il carattere dal buffer
    MOVF INDF,W                   ;prendo il carattere dal buffer e lo metto su W
    MOVWF TEMP                      ;salva il carattere in TEMP
    MOVLW 'Z'                       ;carica il carattere 'Z' in W
    SUBWF TEMP,W                    ;sottrazione di 'Z' da TEMP. Se 'Z'<=TEMP, allora C=1
    BTFSS STATUS,STATUS_C_POSITION  ;BIT-TEST: se C=1, significa che il carattere è minuscolo e va al GOTO INCREMENTO,
                                    //altrimenti si va al GOTO MINUSCOLO
    GOTO MINUSCOLO                  ;vai alla sezione MINUSCOLO
    GOTO INCREMENTO                 ;vai alla sezione INCREMENTO
```

Figura 17: Quarto blocco (parte 3)

La sezione INCREMENTO viene utilizzata quando incontriamo un carattere già minuscolo, di conseguenza dobbiamo saltare la conversione.

```
155     INCREMENTO: ;se il carattere è minuscolo, non lo devo convertire e quindi passo alla valutazione del prossimo carattere
156         INCF FSR,F          ;incrementa il puntatore FSR
157         DECFSZ LUNG_MAX,F   ;decremento di 1 il valore della lunghezza massima
158         GOTO CICLO_CONVERT  ;se LUNG_MAX è diverso da 0, continuo la conversione
159         GOTO INIT_TRANS     ;se LUNG_MAX=0, inizia la trasmissione
```

Figura 18: Quarto blocco (parte 4)

Nella sezione MINUSCOLO avviene la conversione da maiuscolo a minuscolo aggiungendo la costante numerica esadecimale ***0x20*** all'Indirect File Register. Il risultato di quest'ultima operazione viene salvata nuovamente in ***INDF***.

Un modo semplice per effettuare la conversione da maiuscolo a minuscolo è aggiungere il decimale 32 (che corrisponde all'esadecimale ***0x20***) alla posizione in cui ci troviamo nella tabella dei caratteri ASCII.

```
161     MINUSCOLO:
162         ;CONVERSIONE DA MAIUSCOLO A MINUSCOLO (aggiunge 0x20)
163         MOVLW 0x20          ;aggiunge 0x20 per convertire da maiuscolo a minuscolo
164         ADDWF INDF,F        ;aggiorna il carattere nel buffer
165         GOTO INCREMENTO     ;vai a incrementare l'FSR e poi, a seconda di LUNG_MAX, si sceglie come proseguire
```

Figura 19: Quarto blocco (parte 5)

65	A	97	a
66	B	98	b
67	C	99	c
68	D	100	d
69	E	101	e
70	F	102	f
71	G	103	g
72	H	104	h
73	I	105	i
74	J	106	j
75	K	107	k
76	L	108	l
77	M	109	m
78	N	110	n
79	O	111	o
80	P	112	p
81	Q	113	q
82	R	114	r
83	S	115	s
84	T	116	t
85	U	117	u
86	V	118	v
87	W	119	w
88	X	120	x
89	Y	121	y
90	Z	122	z

Figura 20: Tabella caratteri ASCII

In esadecimale, le cifre sono espresse in base di 16, quindi il calcolo è il seguente:

$$2 \cdot 16^1 + 0 \cdot 16^0 = 32$$

5.6 Quinto blocco di codice

Nel quinto blocco vi è il ciclo di trasmissione a cui appartengo le subroutine ***INIT_TRANS***, relativa all'inizializzazione della trasmissione, e ***TRANS***, relativa alla trasmissione stessa.

La sezione INIT_TRANS ha al suo interno le subroutine ***STAMPA_CAPO*** per mandare a capo e ***STAMPA_CONV*** per stampare la parola "CONVERSIONE:".

```
167     ;---------------------- ROUTINE DI TRASMISSIONE ----------------
168     INIT_TRANS: ;inizializzazione della trasmissione
169         ;stampa "CONVERSIONE:" all'avvio della trasmissione
170         PAGESEL STAMPA_CAPO
171         CALL STAMPA_CAPO        ;sezione che manda a capo
172         PAGESEL STAMPA_CONV
173         CALL STAMPA_CONV        ;sezione che stampa "CONVERSIONE:"
174         PAGESEL BUFF
175         CALL BUFF               ;riconfigurazione buffer e lung_max
```

Figura 21: Quinto blocco (parte 1)

```
177     TRANS: ;trasmissione
178         BANKSEL PIR1                    ;il registro PIR1 contiene i flag bits delle interruzioni periferiche
179         BTFSS PIR1,PIR1_TXIF_POSITION   ;aspetta finché il registro di trasmissione non è pronto
180         GOTO TRANS                      ;se TXIF=1, salta l'istruzione successiva, altrimenti continua ad aspettare
181
182         BANKSEL TXREG   ;EUSART Transmit Data Register
183         MOVF INDF,W     ;prendo il carattere dal buffer e lo metto su W
184         MOVWF TXREG     ;carica il carattere nel registro di trasmissione
185
186         ;CONTROLLO PUNTO: Terzo controllo per trasmettere i caratteri fino al punto
187         ;prendo il carattere dal buffer e lo confronto con '.'
188         MOVF INDF,W                         ;prendo il carattere dal buffer e lo metto su W
189         MOVWF TEMP                          ;salva il carattere in TEMP
190         MOVLW '.'                           ;;carica il carattere '.' in W
191         SUBWF TEMP,W                        ;sottrazione di W da TEMP, il risultato va in W.
192                                             //se W=TEMP='.', Z diventa 1
193         BTFSC STATUS,STATUS_Z_POSITION      ;BIT-TEST: se Z=0, l'istruzione successiva viene saltata e
194                                             //quindi continua il ciclo di trasmissione, altrimenti si va al GOTO start
195         GOTO start
196
197         INCF FSR,F          ;incremento per trasmettere il carattere successivo
198         DECFSZ LUNG_MAX,F   ;decremento di 1 il valore della lunghezza massima
199         GOTO TRANS          ;se LUNG_MAX è diverso da 0, continuo la trasmissione
200         GOTO start          ;se LUNG_MAX=0, ritorno allo start
```

Figura 22: Quinto blocco (parte 2)

5.7 Sesto blocco di codice

In questo blocco avvengono le inizializzazioni dell'hardware, dell'EUSART e del buffer.

Nella sezione INIT avvengono le prime due inizializzazioni: quella dell'hardware e quella dell'EUSART ed in aggiunta avviene anche l'inizializzazione dei LED utilizzati come verifica durante la progettazione del codice.

Focalizziamoci sui LED e l'EUSART con cui esaminiamo nuovi registri.

Nell'inizializzazione dei LED analizziamo i registri ***PORTD*** e ***TRISD***: il PORTD è una porta bidirezionale larga 8 bit, e il TRISD è il suo corrispondente registro di direzione dati.

Impostando un bit TRISD (= 1) il pin PORTD corrispondente diventerà un input.

Cancellando un bit TRISD (= 0) il pin PORTD corrispondente diventerà un output.

L'inizializzazione dell'EUSART avviene tramite simboli binari (distinguibili dal carattere "***B***" dopo i numeri). Queste costanti numeriche servono a configurare i registri ***TXSTA***[11], ***RCSTA***[12] e ***OSCCON***[13].

I registri TXSTA e RCSTA, insieme al registro ***BAUDCTL***[14], servono a controllare il funzionamento del modulo EUSART.

In questa parte di blocco viene calcolato anche il Baud Rate[15]. Quest'ultimo viene impostato al valore desiderato usando i bits ***BRGH***[16] (dal registro TXSTA) e ***BRG16***[17] (dal registro BAUDCTL) ed i registri SPBRGH e ***SPBRG***.

La coppia di registri SPBRGH e SPBRG determina il periodo del timer del Baud Rate libero.

In modalità asincrona la velocità di trasmissione risultante è la seguente, dove ***[SPBRG]*** è il valore nel registro con lo stesso nome:

$$\text{For } \mathbf{BRGH} = 0 \qquad \text{Baud rate} = \frac{f_{osc}}{64([\mathbf{SPBRG}] + 1)}$$

$$\text{For } \mathbf{BRGH} = 1 \qquad \text{Baud rate} = \frac{f_{osc}}{16([\mathbf{SPBRG}] + 1)}$$

[11] EUSART Transmit Status and Control Register
[12] EUSART Receive Status and Control Register
[13] Oscillator Control Register
[14] Baud Rate Control Register
[15] Indica il numero di transizioni al secondo che avvengono sulla linea (velocità di trasmissione)
[16] High Baud Rate Select bit
[17] 16-bit Baud Rate Generator bit

Il bit BRGH è uguale a 1, la frequenza dell'oscillatore è stata impostata ad 8MHz nel registro OSCCON e SPBRG vale 25, quindi il Baud Rate sarà circa pari a 19200.

```
;------------------------------INIZIALIZZAZIONI---------------------------------
INIT: ;inizializzazione dell'hardware
    BUF_START EQU 0x30  ;indirizzo di partenza del buffer per la ricezione
    LUNG_MAX EQU 0x70   ;lunghezza max della parola (posizionata lontana dalle prime posizioni
                        //per non incorrere in errori)
    TEMP EQU 0x20       ;variabile temporanea per la conversione
    CLRF TEMP           ;azzera il contenuto del registro di indirizzo 0x20

    ;inizializzazione dei LED utilizzati come verifica durante la progettazione
    BANKSEL TRISD   ;PORTD Tri-State Control bit
    BCF TRISD,0     ;azzera il bit TRISD0, quindi configura il Pin PORTD come uscita
    BANKSEL PORTD   ;PORTD General Purpose I/O Pin bit
    BCF PORTD,0     ;azzerra il bit RD0, quindi spegne il LED

    ;EUSART
    ;INIZIALIZZAZIONE EUSART per trasmissione
    BANKSEL TXSTA       ;EUSART Transmit Status and Control Register
    MOVLW 00100100B     ;TXEN=1 (Transmit enabled), SYNC=0 (Asynchronous mode), BRGH=1 (High speed asynchronous mode)
    MOVWF TXSTA         ;imposta il registro TXSTA

    BANKSEL RCSTA       ;EUSART Receive Status and Control Register
    MOVLW 10010000B     ;SPEN=1 (porta seriale abilitata), CREN=1 (ricezione abilitata)
    MOVWF RCSTA         ;imposta il registro RCSTA

    BANKSEL OSCCON      ;Oscillator Control Register
    MOVLW 01110001B     ;frequenza dell'oscillatore interno impostata a 8MHz
    MOVWF OSCCON        ;scrive il valore in OSCCON
```

Figura 23: Sesto blocco (parte 1)

```
    BANKSEL BAUDCTL     ;Baud Rate Control Register
    CLRF BAUDCTL        ;azzera il registro, allora BRG16=0 (8-bit Baud Rate Generator is used)

    BANKSEL SPBRG   ;insieme al bit BRGH determinano il Baud Rate
    MOVLW 25        ;muovo il 25 in W
    MOVWF SPBRG     ;sposto il 25 nel registro SPBRG, così impostando il Baud Rate a 19200
```

Figura 24: Sesto blocco (parte 2)

```
BUFF: ;INIZIALIZZAZIONE DEL BUFFER
    ;imposto il valore in LUNG_MAX=8
    CLRF LUNG_MAX       ;azzera il registro
    MOVLW 0x08          ;muove l'esadecimale(=8 decimale) in W
    MOVWF LUNG_MAX      ;muove l'esadecimale in W
    MOVLW BUF_START     ;carica l'inizio del buffer
    MOVWF FSR           ;FSR punta all'inizio del buffer
    RETURN              ;return from subroutine
```

Figura 25: Sesto blocco (parte 3)

5.8 Settimo blocco di codice

Nel penultimo blocco sono descritte le routine di scrittura a schermo.

Esso si suddivide in quattro subroutine: ***STAMPA_CAPO***, ***STAMPA_SCRIVI***, ***STAMPA_CONV*** e ***INVIA_CARATTERE***.

La sezione STAMPA_CAPO va a capo.

```
248     ;------------------------------ROUTINE SCRITTURA----------------------
249     STAMPA_CAPO: ;va a capo
250         MOVLW 0x0A              ;muove l'esadecimale(=10 decimale) in W
251         CALL INVIA_CARATTERE    ;chiama la sezione che invia il carattere
252         RETURN                  ;return from subroutine
```

Figura 26: Settimo blocco (parte 1)

La sezione STAMPA_SCRIVI mostra a schermo la parola “SCRIVI”,

```
254     STAMPA_SCRIVI: ;routine per inviare "SCRIVI"
255         MOVLW 'S'               ;carattere 'S'
256         CALL INVIA_CARATTERE    ;chiama la subroutine per inviare il carattere
257         MOVLW 'C'               ;carattere 'C'
258         CALL INVIA_CARATTERE    ;chiama la subroutine per inviare il carattere
259         MOVLW 'R'               ;carattere 'R'
260         CALL INVIA_CARATTERE    ;chiama la subroutine per inviare il carattere
261         MOVLW 'I'               ;carattere 'I'
262         CALL INVIA_CARATTERE    ;chiama la subroutine per inviare il carattere
263         MOVLW 'V'               ;carattere 'V'
264         CALL INVIA_CARATTERE    ;chiama la subroutine per inviare il carattere
265         MOVLW 'I'               ;carattere 'I'
266         CALL INVIA_CARATTERE    ;chiama la subroutine per inviare il carattere
267         MOVLW ':'               ;carattere ':'
268         CALL INVIA_CARATTERE    ;chiama la subroutine per inviare il carattere
269         MOVLW 0x0A              ;va a capo
270         CALL INVIA_CARATTERE    ;chiama la subroutine per inviare il carattere
271         RETURN                  ;return from subroutine
```

Figura 27: Settimo blocco (parte 2)

La sezione STAMPA_CONV mostra a schermo il termine “CONVERSIONE:” e poi va a capo.

```
273     STAMPA_CONV: ;stampa "CONVERSIONE:"
274         MOVLW 'C'               ;carattere 'C'
275         CALL INVIA_CARATTERE    ;chiama la subroutine per inviare il carattere
276         MOVLW 'O'               ;carattere 'O'
277         CALL INVIA_CARATTERE    ;chiama la subroutine per inviare il carattere
278         MOVLW 'N'               ;carattere 'N'
279         CALL INVIA_CARATTERE    ;chiama la subroutine per inviare il carattere
280         MOVLW 'V'               ;carattere 'V'
281         CALL INVIA_CARATTERE    ;chiama la subroutine per inviare il carattere
282         MOVLW 'E'               ;carattere 'E'
283         CALL INVIA_CARATTERE    ;chiama la subroutine per inviare il carattere
284         MOVLW 'R'               ;carattere 'R'
285         CALL INVIA_CARATTERE    ;chiama la subroutine per inviare il carattere
286         MOVLW 'S'               ;carattere 'S'
287         CALL INVIA_CARATTERE    ;chiama la subroutine per inviare il carattere
288         MOVLW 'I'               ;carattere 'I'
289         CALL INVIA_CARATTERE    ;chiama la subroutine per inviare il carattere
290         MOVLW 'O'               ;carattere 'O'
291         CALL INVIA_CARATTERE    ;chiama la subroutine per inviare il carattere
292         MOVLW 'N'               ;carattere 'N'
293         CALL INVIA_CARATTERE    ;chiama la subroutine per inviare il carattere
294         MOVLW 'E'               ;carattere 'E'
295         CALL INVIA_CARATTERE    ;chiama la subroutine per inviare il carattere
296         MOVLW ':'               ;carattere ':'
297         CALL INVIA CARATTERE    ;chiama la subroutine per inviare il carattere
```

Figura 28: Settimo blocco (parte 3)

```
298         MOVLW 0x0A              ;va a capo
299         CALL INVIA_CARATTERE    ;chiama la subroutine per inviare il carattere
300         RETURN                  ;return from subroutine
```

Figura 29: Settimo blocco (parte 4)

La sezione INVIA_CARATTERE utilizza l'USART[18] per inviare i caratteri.

```
302     INVIA_CARATTERE: ;routine per inviare un singolo carattere via USART
303         BANKSEL PIR1                    ;il registro PIR1 contiene i flag bits delle interruzioni periferiche
304         BTFSS PIR1, PIR1_TXIF_POSITION  ;aspetta finché il registro di trasmissione non è pronto
305         GOTO INVIA_CARATTERE            ;se TXIF=1, salta questa istruzione e continua la routine,
306                                         //altrimenti continua ad aspettare
307         BANKSEL TXREG   ;EUSART Transmit Data Register
308         MOVWF TXREG     ;carica il carattere da inviare nel registro TXREG
309         RETURN          ;return from subroutine
```

Figura 30: Settimo blocco (parte 5)

[18] Un USART (Universal Synchronous/Asynchronous Receiver/Transmitter) è un hardware che consente a un dispositivo di comunicare tramite protocolli seriali. Può operare sia in modalità sincrona che asincrona e può ricevere e trasmettere.

5.9 Ottavo blocco di codice

Nell'ultimo blocco è presente la subroutine ***FINE_PROG*** incaricata a termine il programma con la direttiva ***END***.

```
311     ;---------------------- TERMINAZIONE DEL PROGRAMMA ------------------
312     FINE_PROG:
313         END resetVec    ;direttiva che segnala all'assemblatore la fine del programma
```

Figura 31: Ottavo blocco