*DISPENSA (di Riepilogo ed Autoverifica) relativa ad argomenti svolti in "Sistemi e Reti"*

# *PROGRAMMAZIONE ASSEMBLY NEL MICROPROCESSORE ZILOG Z80*

## **A)** NOZIONI E CONCETTI DI BASE SUI SISTEMI A MICROPROCESSORE

Per comprendere ed utilizzare un qualsiasi linguaggio assembly è essenziale avere idee chiare su alcune tecniche e concetti, in particolare:
1) La struttura interna ed il funzionamento di un generico microprocessore.
2) Il modello di Von Neumann di un sistema a microprocessore ed i relativi protocolli di comunicazione.
3) La struttura di base e l'utilizzo delle porte di I/O e delle memorie RAM e ROM.

Autoverifica:
- conoscere le porte logiche, le reti combinatorie e sequenziali;
- disegnare lo schema interno di un microprocessore;
- comprendere qual è la funzione della parte controllo;
- conoscere la funzione di ogni componente la parte operativa;
- essere in grado di seguire mentalmente cosa accade all'esecuzione di un programma LM di alcune istruzioni;
- sapere cos'è un ciclo di clock ed un ciclo macchina;
- sapere cos'è la fase di fetch e la fase di execute;
- disegnare il modello di Von Neumann compresi i collegamenti e conoscere il protocollo master-slave;
- conoscere a grandi linee la struttura interna di un circuito di memoria;
- conoscere la differenza tra RAM, ROM ed EPROM;
- aver chiaro che la memoria è utilizzata per contenere sia dati che programmi;
- sapere che le periferiche di un sistema a microprocessore sono collegate a questo tramite dei registri buffer di interfaccia o porte di I/O, ciascuna dotata di un proprio indirizzo;

## **B)** PROGRAMMAZIONE ASSEMBLY

Per capire un linguaggio assembly in generale occorre saper utilizzare un linguaggio di programmazione e, tenendo a mente la struttura interna di un microprocessore, considerare quali sono le risorse in questo caso disponibili (in termini di variabili -in questo caso registri interni- strutture di controllo e operazioni eseguibili).
Occorre quindi studiare e comprendere i metodi di indirizzamento tipici di un linguaggio assembly e la tipica struttura di un'istruzione assembly.
Ricordare che il termine "Assembler", spesso confuso con Assembly, indica il SW traduttore (tipicamente compila in due passate) da codice sorgente in linguaggio Assembly a codice oggetto in Linguaggio Macchina.

Verifica:
- indicare quali sono le fasi da seguire nello sviluppo di un software e sapere cos'è la metodologia TOP-DOWN;
- indicare qualè la differenza tra assembly e linguaggio macchina;
- indicare cosa significa compilare, interpretare ed assemblare;
- sapere cosa si intende per sottoprogramma o subroutine e relativi parametri di In, Out, In/Out);
- aver chiaro l'uso delle parentesi per indicare il contenuto di una cella di memoria che ha l'indirizzo indicato;
- conoscere la struttura di una generica istruzione linguaggio macchina ed i "campi" che logicamente dovrebbero comporla: codice operativo, operandi, risultato, indirizzo istruzione successiva;
- conoscere i seguenti modi di indirizzamento: implicito, immediato, diretto registro, indiretto registro, diretto memoria, indiretto memoria, indicizzato (con displacement o offset), relativo (quest'ultimo si usa nei salti);
- sapere come impostare l'editing di un programma assembly: etichetta, codice operativo, operandi, commenti;
- conoscere e descrivere le principali pseudoistruzioni: ORG, DEFB, END, MACRO ...;

**C)** PROGRAMMAZIONE ASSEMBLY NEL LINGUAGGIO MACCHINA DEL μP Z80

A questo punto diviene essenziale focalizzare lo studio sul microprocessore in esame e studiarne le caratteristiche interne e funzionali, in particolare:

1) Elenco dei registri interni e loro eventuali utilizzi specifici.

2) Dimensione dei bus (dati ed indirizzi) e altre linee di controllo.

3) Set assembly e tipologie di istruzioni (divise per gruppi funzionali).

4) Modi di indirizzamento ammessi.

Verifica Z80:

- elencare tutti i registri interni accessibili a basso livello (istruzioni Linguaggio Macchina o assembly);

- conoscere l'esistenza del banco alternativo dei registri anche se non si utilizzano normalmente;

- aver compreso chiaramente il funzionamento della struttura stack e relativo registro SP;

- leggere senza dubbi di interpretazione il libretto che elenca e descrive il set assembly in ogni riga corrispondente ad istruzioni di base ed in particolare conoscere la funzione dei dati indicati da ogni colonna;

- saper assemblare (a "mano", senza l'ausilio di un assembler) un semplice programma di alcune righe;

- saper calcolare l'offset da inserire in una JR come differenza tra l'indirizzo a cui deve passare e quello a cui punta attualmente il PC (che punta sempre all'istruzione successiva a quella in esecuzione);

- conoscere l'uso del registro dei flag ed in particolare di Z, C, P/V, S;

- aver chiaro che per eseguire un salto condizionato (cioè un controllo del flusso del programma) è necessario usare un flag ovvero è necessario settare il flag in base ai dati attuali tramite un'istruzione aritmetica o logica;

- indicare le principali tipologie di istruzioni assembly del set Z80;

- conoscere tutte le istruzioni assembly di uso comune ed in particolare: LD, PUSH, POP, IN, OUT, ADD, ADC, SUB, SBC, INC, DEC, NEG, CPL, AND, OR, XOR, BIT, SET, RES, <varie istruzioni di shift & rotate>, CCF, SCF, JP, JR, CALL, RET, NOP, HALT; - Conoscere tutti i possibili operandi per le precedenti istruzioni e quali agiscono anche su dati a 16 bit e non solo su dati ad 8 bit;

- sapere cosa significa il termine interruzione e saper individuare quali istruzioni sono addette alla loro gestione;

- essere in grado di determinare "ad occhio" anche se con una certa approssimazione lo spazio occupato da un'istruzione ed il numero di cicli di clock richiesti per eseguirla;

- sapere desumere da un manuale la funzione di ciascuna istruzione del set assembly (anche quelle non citate) e saperla utilizzare se necessario;

**D)** NOTE **(IMPORTANTISSIME)** SULLA PROGRAMMAZIONE ASSEMBLY/ASSEMBLER Z80

- ➔ nelle istruzioni LD x,y avviene l'assegnamento x:=y;
- ➔ nelle istruzioni aritmetiche ad 8 bit a due operandi uno di questi è sempre il registro A (Accumulatore);
- ➔ nelle istruzioni aritmetiche ad 8 bit il risultato va sempre in A;
- ➔ solo le istruzioni aritmetiche e logiche modificano i flag;
- ➔ le istruzioni aritmetiche su dati a 16 bit non modificano i flag;
- ➔ le PUSH e le POP operano solo su dati a 16 bit;
- ➔ le istruzioni composte (LDIR etc.), pur se più veloci e compatte non sono da utilizzare perché a livello didattico è molto più utile conoscere bene il set Z80 di base;
- ➔ i registri IX ed IY servono a contenere indirizzi (puntatori alla memoria) in particolare se è utile lavorare con spiazzamenti (ad esempio per risolvere problemi con vettori vicini in memoria (± 128 byte));
- ➔ è più efficiente utilizzare il registro HL come puntatore alla memoria anziché DE, BC, IX, IY;
- ➔ il registro A è una risorsa spesso indispensabile (molte istruzioni lavorano solo con A come operando) quindi da utilizzare frequentemente ma da liberare velocemente;
- ➔ occorre inizializzare il registro SP ogni volta che si usa lo stack o anche solo se si utilizzano delle CALL;
- ➔ i salti assoluti JP permettono di saltare a qualsiasi indirizzo da 0000h a FFFFh mentre i salti relativi JR permettono solo di avanzare di 127 byte o indietreggiarne 128 rispetto all'attuale valore del PC;
- ➔ salvare con PUSH e recuperare con POP le risorse (registri interni e flag) utilizzati all'interno di una routine rispettivamente all'ingresso ed all'uscita da questa;
- ➔ ricordare che in assembly il ";" indica l'inizio di un commento fino a fine riga e non la fine istruzione;

**E)** FASI DELLA SOLUZIONE DI UN PROBLEMA DI PROGRAMMAZIONE ASSEMBLY/ASSEMBLER

1) Esame del testo e analisi del problema.

2) Ideazione algoritmo risolutivo tenendo a mente le risorse disponibili a basso livello (al livello assembly/LM) e le strutture dati da utilizzare (realizzare un diagramma di flusso in metodologia TOP-DOWN).

3) Stesura del listato assembly. Inserire gli opportuni commenti in particolare sull'uso delle variabili (registri o zone di memoria) ed ovunque siano utili a comprendere la soluzione da parte di altri o in futuro (ma non commenti stupidi o inutili).

4) Verificare tramite manuale delle istruzioni che tutte le istruzioni appartengano al set assembly Z80 ed eventualmente svolgere le opportune modifiche-ottimizzazioni.

5) "Assemblare", "Linkare" ed eseguire il programma tramite uno strumento di sviluppo adeguato (generalmente un simulatore/emulatore) e verificarne il corretto funzionamento. Nel caso di errori utilizzare gli strumenti di debug per individuarli (TRACE/Step-by-Step & BREAKPOINTS).

[ovviamente svolgere il punto 4) nel caso si tratti di un esercizio in classe oppure il punto 5) nel caso si tratti di una prova di laboratorio] 6) consegnare/utilizzare quanto svolto

**F)** TRADUZIONE ASSEMBLER Z80 DELLE STRUTTURE CONTROLLO DI FLUSSO PASCAL

N.B.: Con BLOCK si intende una sequenza di istruzioni (nel PASCAL delimitate da una coppia BEGIN END, nell'assembly è sufficiente siano in sequenza). È ovvio che in assembly è necessario che nelle istruzioni precedenti i salti condizionati sia settato il flag utilizzato. L'utilizzo del flag Z (Livello assembly) ovvero condizione booleana Z (PASCAL) è indicativo: in generale quanto espresso è valido per tutti i flag.

| *<Pascal>* | *<Assembly>* |
|---|---|
| 1) IF THEN | |
| <BLOCK0>;<br>IF Z THEN <BLOCK1>;<br><BLOCK2>; | <BLOCK0><br><setta Z o NZ><br>JR NZ,DOPO<br><BLOCK1><br>DOPO <BLOCK2> |
| 2) IF THEN ELSE | |
| <BLOCK0>;<br>IF Z THEN <BLOCK1><br>ELSE <BLOCK2>;<br><BLOCK3>; | <BLOCK0><br><setta Z o NZ><br>JR NZ,ELSE<br>THEN <BLOCK1><br>JR FINEIF<br>ELSE <BLOCK2><br>FINEIF <BLOCK3> |
| 3) WHILE DO | |
| <BLOCK0>;<br>WHILE Z DO <BLOCK1>;<br><BLOCK2>; | <BLOCK0><br>CICLO <setta Z o NZ><br>JR NZ,DOPO<br><BLOCK1><br>JR CICLO<br>DOPO <BLOCK2> |
| 4) REPEAT UNTIL | |
| <BLOCK0>;<br>REPEAT <BLOCK1> UNTIL Z;<br><BLOCK2>; | <BLOCK0><br>ANCORA <BLOCK1><br><setta Z o NZ><br>JR NZ,ANCORA<br><BLOCK2> |
| 5) FOR DO | |
| <BLOCK0>;<br>FOR I:=1 TO n DO <BLOCK1>;<br><BLOCK2>; | <BLOCK0><br>LD C,n<br>FOR <BLOCK1<br>– NB: non usa C<br>Fine Block 1><br>DEC C<br>JR NZ,FOR<br><BLOCK2> |

**G)** ESEMPI DI SEMPLICI ESERCIZI SVOLTI

Da svolgere seguendo le fasi di sviluppo sopra descritte confrontando il risultato ottenuto con le soluzioni proposte (che non sono uniche né ottime). Eventualmente se non si riesce a risolvere l'esercizio limitarsi allo studio della soluzione proposta e disegnarne il diagramma di flusso.

NB: Gli esercizi proposti (e svolti) sono elencati in un ordine (più o meno) di complessità crescente.

1) Caricamento della memoria da 210h a 230h compresa con il dato D7H.

```
        ORG 0100h
        LD HL,0210h     ;HL=puntatore alla memoria
        LD C,21h        ;C=contatore celle da riempire
CICLO   LD (HL),D7h     ;carico locazione
        INC HL          ;incremento puntatore memoria
        DEC C           ;decremento contatore volte e setto flag Z
        JR NZ,CICLO     ;se C<>0 non ho finito e salto a ciclo
        HALT
```

2) Spostamento di 50 byte dalla zona 230h alla 270h

```
        ORG 0100h
        LD B,32h        ;B=contatore celle da spostare
        LD IX,0230h     ;IX=puntatore alla memoria
CICLO   LD A,(IX+0)
        LD (IX+40h),A   ;sposto dato usando temporaneamente A
        INC IX          ;incremento puntatore memoria
        DEC B           ;decremento contatore volte e setto flag Z
        JR NZ, CICLO    ;se B<>0 non ho finito e salto a ciclo
        HALT
```

3) Sommare i dati ad 8 bit presenti in un vettore di 32 locazioni allocato a partire dall'indirizzo 200h con quelli in un analogo vettore a partire da 220h, ottenendo un terzo vettore di risultati allocato dall'indirizzo 350h (lavorare in modulo 256 ovvero trascurare eventuali overflow).

```
        ORG 0100h
        LD IY,0200h     ;IY=puntatore ai vettori in memoria
        LD C,20h        ;C=contatore celle
LOOP    LD A,(IY+0)     ;carico in A il primo operando
        ADD A,(IY+20h)  ;sommo primo e secondo dato
        LD (IY+50h),A   ;salvo risultato nel terzo vettore
        INC IY          ;incremento puntatore memoria
        DEC C           ;decremento contatore volte e setto flag Z
        JR NZ,LOOP      ;salto finché non ho esaurito i vettori
        HALT
```

4) Ciclo di ritardo da circa 1/10 di secondo (cicli annidati).

Considerando che la frequenza del ciclo di clock dello Z80 è di 2,5 Mhz, ne risulta TAU = 0,4 µsec.
Per un ritardo di 1/10 sec = 100000 µsec occorrono quindi 100000/0,4 = 250000 TAU ( = durata ciclo clock).
Supponendo una struttura di ritardo come la seguente:

```
        ORG 0100h
RIT     PUSH AF         ; 11 TAU  salvo lo stato dei flag
        PUSH BC         ; 11 TAU  salvo i registri usati dalla routine
        LD B,n          ;  7 TAU  ritardo di n cicli L1
L1      LD C,FFh        ;  7 TAU
L2      DEC C           ;  4 TAU  ritardo di FFh cicli L2
        JR NZ,L2        ; 12 TAU oppure 7 TAU  se non si salta
        DEC B           ; 4 TAU
        JR NZ,L1        ; 12 TAU  oppure 7 TAU se non si salta
        POP BC          ; 10 TAU
        POP AF          ; 10 TAU ripristino lo stato
        RET             ; 10 TAU
```

Occorre inoltre considerare il tempo impiegato dalla CALL RIT = 17 TAU .

Ne risulta che caricando il registro B con il valore n=X (incognita) si ottiene un ritardo di circa 17+11+11+7+X*(7+255*(4+12)+4+12)+10+10+10 TAU (ho trascurato l'ultima esecuzione di ogni ciclo dove il salto non viene eseguito e occupa solo 7 TAU ). Imponendo quindi 76+4179*X=250000 ne risulta X~60 che caricato nel registro B (LD B,3Ch) ottiene un ritardo che approssima adeguatamente quanto richiesto. (verificare i calcoli, potrebbero esserci errori)

5) Ricerca e conteggio del dato 88h in memoria tra la cella 300h e la 6FFh compresa e salvataggio del numero di ricorrenze alla cella 700-701h.

```
        ORG 0100h
        LD HL,300h      ;HL=puntatore alla memoria
        LD BC,0000h     ;azzero BC=contatore volte che trovo 88h
CICLO   LD A,88h        ;predispongo A per confronto tramite CP
        CP (HL)         ;confronto (HL) con 88h
        JR NZ,DOPO      ;se dato <> 88h salto a DOPO
        INC BC          ;incremento contatore #celle uguali ad 88h
DOPO    INC HL          ;incremento puntatore memoria
        LD A,7
        CP H            ;setto flag Z se H=7 ovvero HL=700h
        JR NZ,CICLO     ;salto a ciclo se non ho finito
        LD (0700h),BC   ;salvo alle locazioni 700-701h numero celle=88h
        HALT
```

6) Considerando i dati come numeri rappresentati in complemento a due, ricercare il massimo valore positivo presente in memoria tra la cella 400h e la 7FFh e salvarlo alla locazione 330h lasciandone l'indirizzo sullo stack (solo se esiste almeno un numero > 0 nella zona considerata).

```
            ORG 0100h
            LD SP,0900h         ;se devo usare lo stack occorre inizializzare SP
                                ;notare che lo stack "mangia in basso" ovvero occupa 900h,8FFh,8FEh,etc.
            LD HL,3FFh          ;HL = puntatore memoria
            LD B,00h            ;B = contenitore del dato maggiore trovato sinora
            LD C,00h            ;C viene usato come flag C = 0 --> no dato > 0
NEWDAT      INC HL              ;incremento puntatore memoria
            LD A,H
            CP 08h
            JR Z,FINEDAT        ;se H = 8 significa che ho esaurito i dati
            LD A,(HL)           ;carico in A contenuto cella puntata da HL
            ADD A,00h           ;setto i flag in base al valore di A
            JP M,NEWDAT         ;se A < 0 passo al prossimo - N.B. uso JP in quanto non esiste JR col flag S
            CP B                ;setto i flag facendo A - B, se B > A si setta C
            JR C,NEWDAT         ;se B > A A non mi interessa e passo al prossimo
NEWMAX      LD B,A              ;se sono qui significa A > B --> B:=A
            PUSH HL
            POP DE              ;eseguo LD DE,HL che non esiste usando lo stack
            INC C               ;ho trovato un nuovo max --> incremento il flag C
            JR NEWDAT           ;passo al prossimo
FINEDAT     LD A,0h
            ADD C               ;setto il registro F in base al flag C
            JR Z,DOPO           ;se c = 0 non salvo il massimo perché non esiste
SALVMAX     LD A,B
            LD (0330h),A        ;salvo il massimo nella cella di indirizzo 0330h
                                ;usando A perché nello Z80 non esiste LD (nn),B
            PUSH DE             ;salvo l'indirizzo del massimo sullo stack
DOPO        HALT                ;ho finito
```

Pensare ad una condizione particolare in cui il programma non è corretto.

Provare a capire il programma senza leggere i commenti, ovvero a scriverli e poi confrontarli con i miei.

Disegnare un diagramma di flusso del programma precedente.

7) Moltiplicazione di due dati positivi ad 8 bit allocati in B ed in C, salvando il risultato su HL (metodo delle addizioni successive: per moltiplicare X*Y addiziono X a partire da 0 per Y volte).

```
            ORG 0100h           ;NB: I registri B e C devono essere precaricati coi valori da moltiplicare
            LD DE,0000          ;azzero DE in quanto devo operare su 16 bit
            LD HL,0000          ;azzero HL = l'accumulatore a 16 bit
            LD A,C              ;carica C in Acc (trucchetto – vedi sotto – per verificare se Reg C=0)
            ADD 0               ;setto il flag Z se C = moltiplicatore = 0
            JR Z,FINEMOLT       ;se C = 0 ho finito e HL contiene già risultato
CNOZERO     LD E,B              ;scrivo B su 16 bit dentro DE
            ADD HL,DE           ;svolgo l'addizione successiva
            DEC C               ;decremento C e verifico se = 0
            JR NZ,CNOZERO       ;se C > 0 eseguo altra somma
FINEMOLT    HALT                ;ho finito, HL contiene il risultato
```

Provare ad utilizzare un algoritmo più efficiente per risolvere il medesimo esercizio (vedi algoritmo del contadino russo, in assembly super-efficiente grazie alle istruzioni di shift) e calcolare il tempo impiegato (in cicli di clock) per eseguire questo frammento di programma e quello alternativo di minore complessità computazionale.

**Cenni algoritmo**: per moltiplicare due numeri A e B ottenendo R = A * B potremmo:
anziché fare come nell'esempio sopra: R = A + A + A ……B volte
calcolare: R = (A * 2) * (B / 2) …… Raddoppiando e dimezzando A e B finchè B vale 1
attenzione perché dobbiamo lavorare con numeri interi e quindi l'idea dell'algoritmo diviene:
**R = repeat [ 2A * (B div 2) + A** *solo se (B mod 2) <>0* **] until B = 1;**

8) Sommare due numeri binari interi positivi su 32 bit posti in memoria sequenzialmente a partire dall'indirizzo 200h e scrivere in sequenza il risultato. Se si ha overflow (risultato > $2^{32}$ - 1) lanciare una subroutine di nome ERRORE.

```
            ORG 0100h
            LD SP,0800h         ;devo usare lo stack --> inizializzo SP
            LD IX,0200h         ;IX=ind I° op, IX+4=ind II° op, IX+8=ind risultato
            LD C,04h            ;C = contatore # byte che compongono gli operandi
            SCF                 ;setto flag C=1
            CCF                 ;complemento flag C e quindi lo setto = 0
SOMBYTE     LD A,(IX+0)         ;carico byte I° operando in A
            ADC A,(IX+04)       ;addiziono byte II° op,0 considerando bit di carry
            LD (IX+08),A        ;salvo byte risultato
            DEC C               ;notare che la DEC modifica il registro F
                                ;ma lascia inalterato il flag C
            JR NZ,SOMBYTE       ;salto finché non ho sommato i 4 byte
FINESOM     CALL C,ERRORE       ;nel caso il flag C sia settato ho avuto overflow
            . . .               ;etc.
```

9) Utilizzare la chiamata annidata di subroutine per realizzare un ritardo di un minuto.

Considerato che 1 minuto = 60 secondi = 600 decimi di secondo è sufficiente chiamare 600 volte la routine RIT (vedi esercizio n.4).

```
            ORG 0200h
            LD SP,0800h
RMIN        LD C,C8h            ;carico il numero 200 = C8h nel reg C
            CALL RIT
            CALL RIT
            CALL RIT            ;ogni CALL RIT = 1/10 sec
            DEC C
            JR NZ,RMIN          ;salta ad RMIN finchè C<>0
            . . .
```

Notare che ho utilizzato C anche dentro RIT. Questo non crea problemi avendo provveduto a salvare lo stato e le risorse sullo stack.

Provare a seguire cosa avviene sullo stack durante l'esecuzione del programma.

10) Produrre in uscita su di un banco di 8 led interfacciati ad un registro latch di uscita (port di OUT) il cui indirizzo è 15h lo scorrimento di un solo led acceso impiegando 4 secondi per ogni ciclo completo (1/2 sec per ogni led). Ricordare che con degli ambienti di simulazione Z80 non è possibile utilizzare le istr. IN… e OUT...!

```
            ORG 0300h
            LD SP,0800h
            LD A,01h            ;configurazione iniziale dei led da visualizzare
SCORRI      OUT (15h),A         ;accendo i led a seconda del contenuto di A
            LD C,05h
ATTESA      CALL RIT            ;attesa di 1/2 sec (5*1/10 sec)
            DEC C
            JR NZ,ATTESA
            RLCA                ;rotazione circolare a sinistra di A
            JR SCORRI           ;loop infinito
            . . .
```

H) BUON LAVORO! Ricordare che se si usa un ambiente di sviluppo che richiede la partenza del programma dalla locazione 0000h (posizione di avvio dello Z80 in fase di reset) è sufficiente anteporre le seguenti istruzioni:

```
            ORG 0000h
            JP <posizione Hex dove è posizionato il codice LM>
```

Chiedere con l'insegnante per qualsiasi dubbio sul contenuto della dispensa.