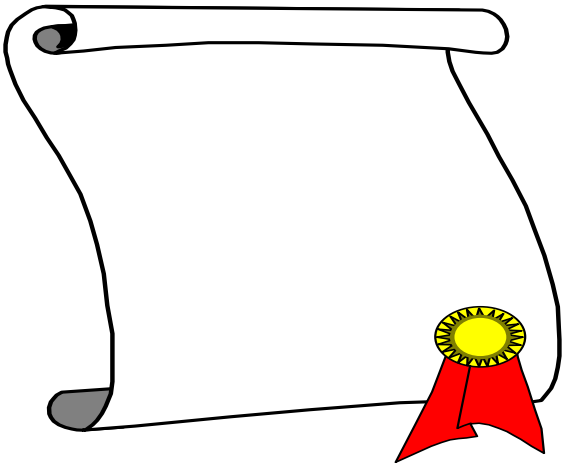


MANUALE D'USO

di

Z80 SIMULATION

Versione 2.00

Realizzato da Fulvio RICCIARDI - LECCE
e-mail: **fricciardi@libero.it** oppure **fricciardi73@hotmail.com**
Tel. 0338/2091766

INTRODUZIONE

Lo Z80 è uno dei microprocessori più utilizzati nel campo dell'automazione. Esso possiede infatti un set di istruzioni molto vasto, la possibilità di essere interfacciato con un elevato numero di dispositivi elettronici, nonché un costo relativamente basso. D'altra parte, spesso, l'uso di microprocessori più evoluti, quali per esempio quelli della famiglia INTEL dall'8086 in su, risulta svantaggioso, poiché le loro potenzialità rimarrebbero in gran parte inutilizzate, portando così a un rapporto costo-prestazioni troppo elevato.
Tuttavia, lo Z80 cede il posto a questi microprocessori nel campo dei microcomputer: qui, infatti, le potenzialità di una CPU non sono mai abbastanza. A conseguenza di ciò, sta il fatto, che per programmare lo Z80, sono necessari strumenti software che girano su macchine che non possiedono tale CPU; basta pensare ad esempio ai cross-assemblatori ovvero a quegli assemblatori, che pur girando su di una macchina con un determinato microprocessore, producono codice per un altro tipo di microprocessore.
Z80 Simulation è un programma che appartiene a questa famiglia di software: esso, infatti, gira sotto il sistema operativo MS-DOS, ma si propone di facilitare la programmazione dello Z80.
Si compone di vari moduli, tra cui:

- l'editor, che permette la stesura del programma in linguaggio assembler;
- l'esecutore, che simula lo Z80, permettendo così l'esecuzione del programma scritto;
- il debugger, che consente una rapida ricerca e correzione degli eventuali errori;
- l'assemblatore, che permette, una volta scritto e testato il programma, di generare il relativo codice macchina direttamente eseguibile dallo Z80.

I vari componenti sono collegati da una interfaccia utente semplice da utilizzare poiché fa uso di menu a tendina. L'applicazione di Z80 Simulation facilità, date le sue caratteristiche, la programmazione di schede in logica programmata facenti uso del µP Z80:

- il programma viene comodamente scritto in linguaggio Assembler Z80;
- viene poi compilato e mandato in esecuzione;

- una volta corretti gli eventuali errori tramite il debugger presente, il programma può essere assemblato, direttamente copiato su EPROM e quindi inserito sulla scheda.

Z80 Simulation, inoltre, si presta bene anche nel campo della didattica, poiché permette l'avvicinamento al linguaggio del microprocessore in modo semplice e graduale, evitando il diretto contatto con i circuiti elettronici in una prima fase, e facilitandolo in un secondo tempo.
Vedremo nei prossimi capitoli come utilizzarlo e quali vantaggi il suo uso può comportare.

INDICE

CAPITOLO 1

CAPITOLO 2

CAPITOLO 3

3.33 CP VAL8
3.34 CP REG8
3.35 CP (PMEM)
3.36 CPD
3.37 CPDR
3.38 CPI
3.39 CPIR
3.40 CPL
3.41 DAA
3.42 DEC REG8
3.43 DEC (PMEM)
3.44 DEC REG16
3.45 DI
3.46 DJNZ VAL8
3.47 EI
3.48 EX AF,AF'
3.49 EX DE,HL
3.50 EX (SP),REG16
3.51 EXX
3.52 HALT
3.53 IM VAL8
3.54 IN REG8,(C)
3.55 IN A,(VAL8)
3.56 INC REG8
3.57 INC (PMEM)
3.58 INC REG16
3.59 IND
3.60 INDR
3.61 INI
3.62 INIR
3.63 JP VAL16
3.64 JP CC,VAL16
3.65 JP (REG16)
3.66 JR VAL8
3.67 JR CC,VAL8
3.68 LD REG16,(VAL16)
3.69 LD REG16,VAL16
3.70 LD REG8,VAL8
3.71 LD REG8,REG8*
3.72 LD REG8,(PMEM)

3.73 LD (PMEM),A
3.74 LD (VAL16),A
3.75 LD (PMEM),VAL8
3.76 LD (PMEM),REG8
3.77 LD (VAL16),REG16
3.78 LD A,(PMEM)
3.79 LD A,(VAL16)
3.80 LD A,I
3.81 LD A,R
3.82 LD I,A
3.83 LD R,A
3.84 LD SP,REG16
3.85 LDD
3.86 LDDR
3.87 LDI
3.88 LDIR
3.89 NEG
3.90 NOP
3.91 OR VAL8
3.92 OR REG8
3.93 OR (PMEM)
3.94 OUT (C),REG8
3.95 OUT (VAL8),A
3.96 OUTD
3.97 OUTI
3.98 OTDR
3.99 OTIR
3.100 POP REG16
3.101 PUSH REG16
3.102 RLCA
3.103 RES nb,REG8
3.104 RES nb,(PMEM)
3.105 RET
3.106 RET CC
3.107 RETI
3.108 RETN
3.109 RL REG8
3.110 RL (PMEM)
3.111 RLA
3.112 RLC REG8

APPENDICE C

CAPITOLO 1

Obiettivi del capitolo:

- Descrizione dell'ambiente integrato
- Le funzioni dell'interfaccia utente

1.1 L'AMBIENTE INTEGRATO

Z80 Simulation è costituito da vari strumenti: l'editor, il compilatore, l'assemblatore, l'esecutore, il debugger e da molte procedure utili all'utente per la generazione e il controllo del programma. Tutti questi moduli sono uniti insieme dall'interfaccia utente e formano così l'ambiente integrato. Vediamo adesso come si presenta tale ambiente.
Subito dopo l'avviamento di Z80 Simulation e la pressione di un tasto (che termina la fase di presentazione), viene attivata la barra menu: questa è il principale dei menu di scelta e la sua attività può essere verificata dal particolare messaggio della riga di stato (fig. 1.1). Per selezionare un comando nella barra menu vi sono due modi:

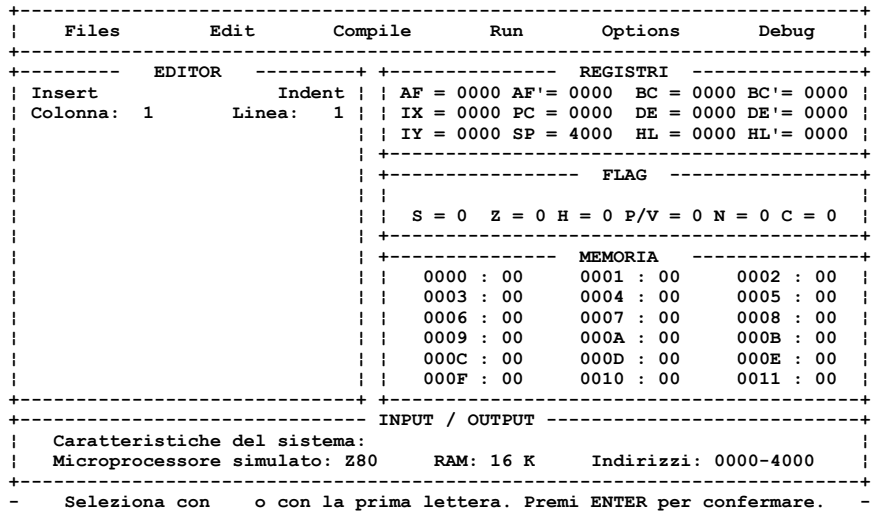
1. Evidenziare la voce interessata usando i tasti freccia. Quindi premere ENTER per confermare.
2. Premere la lettera evidenziata nella voce interessata.

Il primo modo appare più adatto se si utilizza il Mouse (dove quest'ultimo lavori in emulazione dei tasti freccia), mentre il secondo è utile se si utilizza la tastiera.

La barra menu, oltre ad essere la prima per attività, è anche la prima fisicamente, ovvero essa è collocata nella parte alta del video.
Sulla sinistra e sottostante alla barra menu si trova l'editor: tramite questo è possibile inserire e modificare il programma in prova. Le funzioni dell'editor sono molte, pertanto rimandiamo la trattazione di queste al prossimo capitolo interamente dedicato a questo componente.
Nella parte destra del video, accanto all'editor troviamo tre finestre:

- La prima è quella dei registri. In essa è mostrato lo stato di tutti i registri.
- La seconda è quella dei flag. In essa è indicato lo stato di tutti i flag, desumibile d'altra parte dal registro F (o F').
- L'ultima finestra è quella della memoria. In questo caso non si può dire che in essa è visualizzato l'intero stato della memoria, poiché la RAM è di 16 Kbytes. Tuttavia, ogniqualvolta il programma (o direttamente l'utente) modifica un byte di memoria, tale locazione modificata viene mostrata nella finestra.

Si noti comunque, che durante l'esecuzione di un programma, queste finestre possono rispecchiare o meno lo stato reale del sistema, a seconda di come è impostato il debugger. Ciò viene spiegato meglio nella descrizione di quest'ultimo.



<Figura 1.1> L'ambiente integrato.

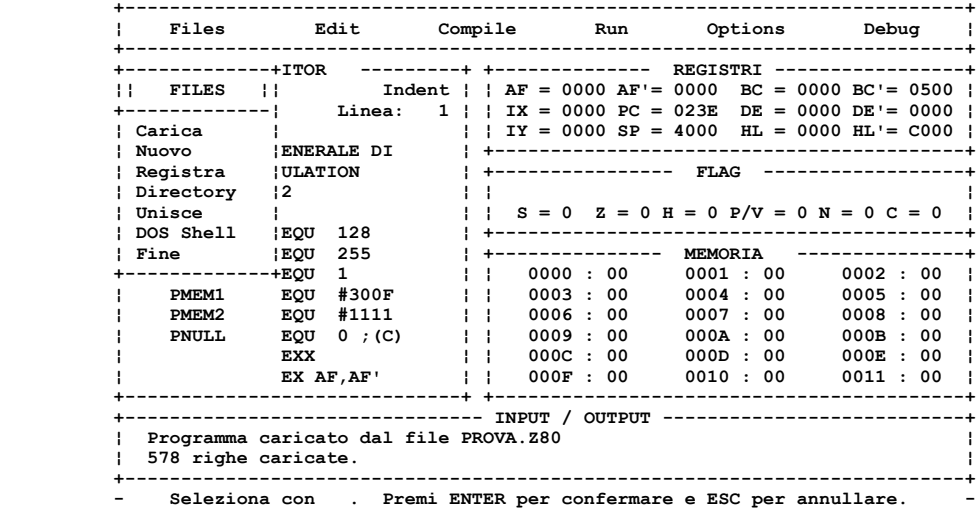
Sotto l'editor e la finestra della memoria è situata la finestra di INPUT/OUTPUT. Essa assolve a due compiti:

- Il primo compito è quello di mostrare le richieste di input (istruzione IN) e di output (istruzione OUT) da parte del programma in esecuzione. Anche in questo caso valgono le considerazioni precedenti riguardo l'impostazione del debugger.
- Il secondo compito è quello di permettere lo scambio di messaggi fra l'utente e l'ambiente integrato. Per esempio: nell'operazione di caricamento di un file, Z80 Simulation manda il messaggio "Nome file:", aspettando dall'utente la risposta. Se eventualmente il file non esistesse, verrebbe visualizzato il messaggio "Il file non è stato trovato.". Si noti che in generale, un messaggio di errore è seguito da un beep.

L'ultima linea dello schermo è occupata dalla riga di stato. In essa sono descritte le azioni che l'utente può intraprendere di situazione in situazione. Molto spesso la riga di stato sopperisce all'assenza di un help in linea. L'aspetto completo dell'ambiente integrato è mostrato in figura 1.1.
Nei prossimi paragrafi vengono descritte in dettaglio le funzioni di ciascun comando dell'ambiente integrato. Si noti la convenzione di intitolare i paragrafi con una stringa del tipo [voce1]/[voce2]/../[vocen], dove voce1 rappresenta la voce selezionata dalla barra menu, voce2 (se esiste) la voce selezionata da un menu sottostante alla barra e così continuando.

1.2 IL MENU FILES [Files]

La prima voce della barra menu, sia per collocazione che per ordine d'uso, è [Files]. Dopo averla selezionata, compare il menu Files (Fig. 1.2) che permette di caricare, salvare, fare il merge, cancellare il contenuto dell'editor, nonché di visualizzare le directory del disco, sospendere momentaneamente l'esecuzione di Z80 Simulation e terminare l'esecuzione dello stesso. Questi comandi vengono descritti di seguito.



<Figura 1.2> Il menu Files.

1.3 CARICAMENTO PROGRAMMA [Files]/[Carica]

Se si è precedentemente salvato un programma su file, questo può essere caricato nell'editor tramite il comando [Files]/[Carica]. Le operazioni da eseguire per caricare un programma sono:

- Attivare il menu Files tramite la voce [Files] della barra menu.
- Attivare il comando di caricamento tramite la voce [Carica] del menu Files.
- Se nell'editor è già presente un programma e questo non è stato salvato dopo l'ultima modifica, Z80 Simulation chiede se è necessario registrarlo: rispondere di conseguenza.
- Inserire il nome del file da cui caricare il programma.

Se le operazioni di caricamento vanno bene, Z80 Simulation visualizza un messaggio indicando il numero di righe caricate, altrimenti mostra un messaggio d'errore. Il vecchio programma, eventualmente presente nell'editor viene cancellato.

Si noti infine, che non necessariamente il programma da caricare, deve essere stato prodotto e registrato da Z80 Simulation: un qualsiasi file, purché di formato ASCII può essere caricato. Questa particolarità permette all'utente di scrivere i suoi programmi con un editor esterno, benché le funzioni dell'editor di Z80 Simulation siano sufficienti. Le righe che superano il limite fisico di 30 caratteri vengono troncate.

1.4 CANCELLA PROGRAMMA [Files]/[Nuovo]

Quando nell'editor è contenuto un programma che almeno momentaneamente non serve più, questo può essere cancellato tramite il comando [Files]/[Nuovo]. Le operazioni da eseguire per inizializzare l'editor sono:

- Attivare il menu Files tramite la voce [Files] della barra menu.
- Attivare il comando di cancellazione tramite la voce [Nuovo] del menu Files.
- Se nell'editor è presente un programma che non è stato salvato dopo l'ultima modifica, Z80 Simulation chiede se lo si vuole cancellare comunque: rispondere di conseguenza.

Se il programma viene cancellato Z80 Simulation visualizza il messaggio "Contenuto dello EDITOR cancellato." e l'editor mostra la prima pagina (ovviamente vuota).
Si noti infine, che se si vuole cambiare programma caricandone uno da disco, non è necessario preventivamente cancellare quello presente, poiché tale operazione è eseguita automaticamente dal comando di caricamento.

1.5 REGISTRA PROGRAMMA [Files]/[Registra]

Molto spesso, lavorando con Z80 Simulation, si ha la necessità di salvare sul disco il programma contenuto nell'editor, per poi riprenderlo in tempi successivi. Ciò è possibile grazie al comando [Files]/[Registra]. Ecco di seguito le operazioni da eseguire per registrare un programma:

- Attivare il menu Files tramite la voce [Files] della barra menu.

- Attivare il comando di registrazione per mezzo della voce [Registra] del menu Files.
- Inserire il nome del file su cui registrare il programma.

Se la registrazione avviene correttamente, viene indicato il numero di righe registrate su file, altrimenti Z80 Simulation visualizza un messaggio d'errore.
Il file prodotto dal comando di registrazione [Files]/[Registra] è un file ASCII. Con ciò si vuol mettere in evidenza, che su tale file può essere applicata qualsiasi operazione legale sui file ASCII, quale per esempio l'editazione con un editor esterno o la stampa su stampante. In particolare per stampare il programma contenuto nell'editor utilizzare il seguente metodo:

- Portare in stato di ON-LINE la stampante.
- Attivare il menu Files tramite la voce [Files] della barra menu.
- Attivare il comando di registrazione per mezzo della voce [Registra] del menu Files.
- Inserire come nome di file PRN o LPT1 (ovviamente a patto che la stampante sia collegata sulla porta parallela).

Infine, è utile notare che il comando [Files]/[Registra], come del resto tutti i comandi di Z80 Simulation che implicano la scrittura su file, può creare o meno i file di riserva con estensione BAK a seconda di come tale opzione è impostata nel SETUP. Ciò è meglio descritto nel paragrafo riguardante il SETUP di Z80 Simulation.

1.6 VISUALIZZA DIRECTORY [Files]/[Directory]

Quando si opera con i file, si presenta frequentemente la necessità di conoscere le directory. Tenendo conto di questa necessità, allo scopo di evitare il continuo ricorso alla Shell del DOS, Z80 Simulation permette di visualizzare le directory tramite il comando [Files]/[Directory]. Di seguito sono indicate le operazioni necessarie alla visualizzazione del contenuto di una directory:

- Attivare il menu Files tramite la voce [Files] della barra menu.
- Attivare il comando di visualizzazione delle directory per mezzo della voce [Directory] del menu Files.

- Inserire il nome della directory e dei file (rispettando le regole del DOS) da visualizzare. Supponiamo per esempio, di voler visualizzare tutti i file della directory Z80DIR immediatamente sottostante alla directory radice del disco. Bisogna inserire la stringa: \Z80DIR*.*. Si noti la presenza di *.* per specificare quali file visualizzare: sarebbe infatti scorretto inserire solo la stringa \Z80DIR.

Se i file specificati esistono, Z80 Simulation apre una finestra e in questa oltre a mostrare i nomi dei file e la loro estensione, mostra anche altre informazioni quali la dimensione di ciascun file, lo spazio totale occupato dai file visualizzati, lo spazio rimanente sull'unità in cui si trova la directory, nonché il nome dell'unità stessa.
Se il numero dei file è troppo elevato, si può scorrere la directory facendo uso dei tasti freccia.
Per chiudere la finestra della directory, e rendere nuovamente attiva la barra menu, è sufficiente premere il tasto ESC.

1.7 FILE MERGE [Files]/[Unisce]

In alcuni casi, sorge la necessità di unire al programma contenuto nell'editor, un programma contenuto in un file (operazione di merge) e che il risultato di tale operazione risieda nell'editor stesso. La soluzione a tale problema è data dal comando di Z80 Simulation [Files]/[Unisce]. Questo comando è molto simile al comando di caricamento di un file [Files]/[Carica] tranne per il fatto che non cancella il contenuto dell'editor e che quindi accoda le righe caricate a quelle già presenti nell'editor. Ecco ora descritti i passi da compiere per eseguire il merge:

- Attivare il menu Files tramite la voce [Files] della barra menu.
- Attivare il comando di file merge tramite la voce [Unisce] del menu Files.
- Inserire il nome del file in cui è contenuto il programma da accodare a quello presente nell'editor.

Se le operazioni di file merge vanno a buon fine, Z80 Simulation visualizza un messaggio indicando il numero di righe accodate, altrimenti, mostra un messaggio d'errore.

1.8 DOS SHELL [Files]/[DOS Shell]

Quando si vuole eseguire un comando DOS, oppure si vuole eseguire un programma esterno, quando insomma si ha bisogno della shell del DOS, non è necessario interrompere definitivamente l'esecuzione di Z80 Simulation.
Grazie al comando [Files]/[DOS Shell] è possibile infatti sospendere solo momentaneamente l'esecuzione e chiamare la Shell del DOS. Per poter eseguire questo comando, Z80 Simulation deve trovare una copia del file COMMAND.COM o nella directory corrente o nella directory specificata dalla variabile d'ambiente COMSPEC, oltre naturalmente a disporre di una quantità di memoria sufficiente. Per chiudere la Shell e tornare in Z80 Simulation è necessario digitare il comando EXIT.

1.9 RITORNO AL DOS [Files]/[Fine]

Per terminare l'esecuzione di Z80 Simulation e quindi restituire il controllo al DOS utilizzare il comando [Files]/[Fine]. Prima di terminare l'esecuzione, si chiede all'utente di confermare la scelta e in particolare se il programma non è stato registrato, viene inviato un messaggio di avvertimento.

1.10 ATTIVA L'EDITOR [Edit]

Per editare un programma, bisogna attivare l'editor tramite il comando [Edit]. Data la numerosità dei comandi dell'editor ne rimandiamo la descrizione dettagliata al capitolo 2 interamente dedicato all'editor.
Si noti solamente, che per uscire dall'editor e restituire il controllo alla barra menu, è necessario premere il tasto ESC.

1.11 COMPILAZIONE [Compile]

Prima di mandare in esecuzione un programma, è necessario compilarlo, ovvero trasformarlo in un codice (si badi a non confonderlo con il codice macchina, che è invece l'output dell'assemblatore) eseguibile dall'esecutore di Z80 Simulation. Per compilare il programma presente nell'editor utilizzare il comando [Compile]. Durante la compilazione, nella finestra di Input/Output è indicato il numero della linea di programma che si sta compilando. La compilazione avviene in due passate: nella prima sono calcolati i valori delle label; nella seconda viene realmente generato il codice. Eventuali errori provocano l'arresto del compilatore con la visualizzazione di un messaggio d'errore.

1.12 ESECUZIONE [Run]

Dopo essere stato compilato, il programma può essere eseguito per mezzo del comando [Run]. Questo comando, prima di attivare l'esecutore, carica nel Program Counter (registro PC) il valore 0001, cosicché l'esecuzione parte dalla prima linea del programma (vedremo che con il comando [Options]/[Gotoline] l'esecuzione parte da una linea specificata dall'utente).
L'esecuzione di un programma può sempre essere arrestata mediante la pressione del tasto ESC che restituisce il controllo alla barra menu.
Se si vuole conoscere la durata dell'esecuzione del programma si deve ricorrere alle informazioni di sistema, ovvero utilizzare il comando del menu Options [Options]/[Informazioni].
L'esecutore si comporta in modo diverso a seconda di come sono impostate le opzioni di debugger:

- Può visualizzare o meno, durante l'esecuzione, lo stato dei registri e dei flag.
- Può visualizzare o meno, durante l'esecuzione, le locazioni di memoria modificate.
- Può visualizzare o meno, durante l'esecuzione, le richieste di Input/Output.
- Può arrestarsi o meno dopo l'esecuzione di una singola istruzione, aspettando che sia premuto il tasto ENTER prima di eseguire la successiva (esecuzione passo passo).

E' comunque ovvio, che se l'esecutore è settato in modo da non visualizzare durante l'esecuzione le modifiche sui registri e sui flag, lo stato reale di questi viene mostrato a fine esecuzione. Il modo con cui impostare il debugger è descritto più avanti nel paragrafo dedicato al comando [Debug].
E' infine opportuno notare, che quando si lancia il comando [Run] e il programma non è stato compilato dopo l'ultima modifica, esso viene automaticamente compilato.

1.13 IL MENU OPTIONS [Options]

Selezionando la voce [Options] compare il menu delle opzioni (Fig. 1.3). Da esso si possono attivare una serie di comandi che, sebbene non siano indispensabili, sono spesso molto utili. Tali comandi permettono di aggiornare il valore dei registri, gestire la memoria, eseguire un programma a partire da una linea qualsiasi, attivare l'assemblatore, ottenere alcune informazioni di sistema, eseguire il setup di Z80 Simulation e di fare una serie di operazioni aritmetiche.

```
+-------------------------------------------------------------------------------+
¦    Files        Edit       Compile        Run        Options       Debug  ¦
+-------------------------------------------------------------------------------+
+---------   EDITOR   --------++---------------- +--------------+-----------+
¦ Insert               Indent ¦¦ AF = 0000 AF'= 0¦¦  OPTIONS   ¦¦ BC'= 0000 ¦
¦ Colonna:  1      Linea:  91 ¦¦ IX = 0000 PC = 0+--------------¦ DE'= 0000 ¦
¦                             ¦¦ IY = 0000 SP = 4¦ Registri     ¦ HL'= 0000 ¦
¦   LOOP10     DEC B          ¦+-----------------¦ Memoria      ¦-----------+
¦              JP NZ,LOOP10   ¦+-----------------¦ Goto line    ¦-----------+
¦              LD C,#1D       ¦¦                 ¦ Assembler    ¦           ¦
¦              SCF            ¦¦  S = 0  Z = 0 H ¦ Informazioni ¦ 0  C = 0  ¦
¦   LOOP       DEC C          ¦+-----------------¦ Setup        ¦-----------+
¦              JP Z,LOOP      ¦+-----------------¦ Arithmetic   ¦-----------+
¦              CCF            ¦¦   0000 : 00     +--------------+002 : 00   ¦
¦              JP NC,LOOP     ¦¦   0003 : 00     0004 : 00      0005 : 00   ¦
¦              JP C,ESCE0     ¦¦   0006 : 00     0007 : 00      0008 : 00   ¦
¦   VALUE      EQU #E5        ¦¦   0009 : 00     000A : 00      000B : 00   ¦
¦   ESCE0      LD D,VALUE     ¦¦   000C : 00     000D : 00      000E : 00   ¦
¦   LOOP2      RL D           ¦¦   000F : 00     0010 : 00      0011 : 00   ¦
+------------------------------++-----------------------------------------+
+------------------------------ INPUT / OUTPUT -------------------------------+
¦  Programma caricato dal file PROVA.Z80                                       ¦
¦  578 righe caricate.                                                         ¦
+-------------------------------------------------------------------------------+
-    Seleziona con   .   Premi ENTER per confermare e ESC per annullare.     -
```

<Figura 1.3> Il menu options.

1.14 MODIFICA REGISTRI [Options]/[Registri]

L'utente può modificare direttamente il valore dei registri tramite il comando [Options]/[Registri]. I passi da compiere per aggiornare i registri sono:

- Attivare il menu options tramite la voce [Options] della barra menu.
- Attivare il comando di aggiornamento registri selezionando la voce [Registri] del menu options.
- Il cursore punta sul primo registro (AF) permettendone l'aggiornamento: si prema il tasto ENTER per passare al registro successivo.
- Una volta aggiornati i registri desiderati premere il tasto ESC per restituire il controllo alla barra menu.

Si noti che tramite questo comando è possibile aggiornare il valore dei flag modificando opportunamente il registro F (o F' se è attiva la coppia di registri AF').

1.15 GESTIONE MEMORIA [Options]/[Memoria]

Con il comando [Options]/[Memoria] si accede al menu della memoria (Fig. 1.4). Tramite tale menu è possibile eseguire una serie di operazioni sulla memoria, quali la visualizzazione, l'aggiornamento, l'azzeramento, la copia, il caricamento da file, la registrazione su file, nonché lo spostamento dei 16K di memoria da un indirizzo ad un altro.
Vediamo adesso uno per uno come operano questi comandi.

1.16 [Options]/[Memoria]/[Visualizza]

Se si vuole conoscere il contenuto di un byte o di più byte consecutivi di memoria si può utilizzare il comando [Options]/[Memoria]/[Visualizza]. Le operazioni da compiere sono:

- Attivare il menu options tramite la voce [Options] della barra menu.
- Attivare il menu della memoria selezionando la voce [Memoria] del menu options.
- Attivare il comando di visualizzazione tramite la voce [Visualizza].

- Inserire l'indirizzo (esadecimale) della prima locazione da visualizzare.
- Premere il tasto ENTER per visualizzare la locazione successiva.
- Premere ESC per terminare la visualizzazione e restituire il controllo alla barra menu.

La visualizzazione avverrà nella finestra della memoria con il solito formato:

hhhh : hh

dove hhhh rappresenta l'indirizzo in esadecimale della locazione, mentre hh è il suo valore, anche questo in esadecimale.

1.17 [Options]/[Memoria]/[Modifica]

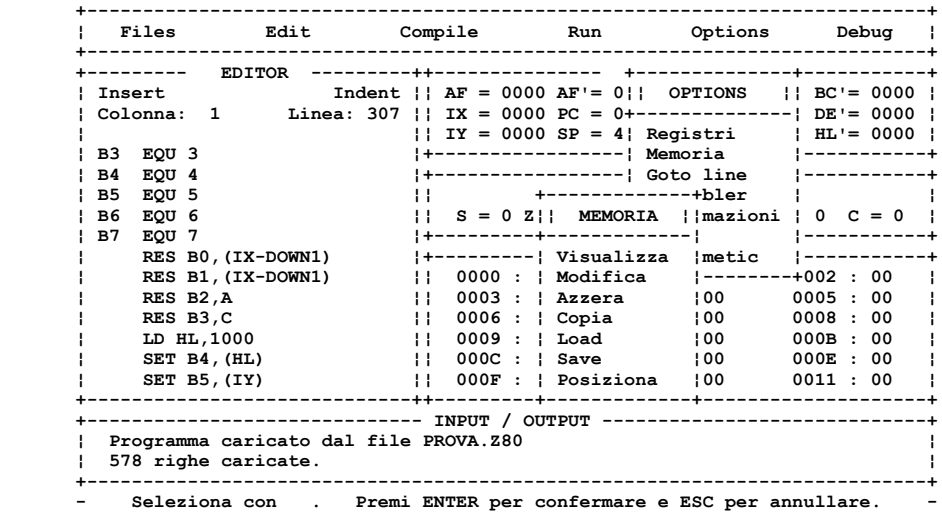
Se l'utente vuole direttamente modificare il contenuto della memoria, deve utilizzare il comando [Options]/[Memoria]/[Modifica]. Le operazioni da eseguire per far ciò sono:

- Attivare il menu options tramite la voce [Options] della barra menu.
- Attivare il menu della memoria selezionando la voce [Memoria] del menu options.
- Attivare il comando di aggiornamento tramite la voce [Modifica].
- Inserire l'indirizzo (esadecimale) della prima locazione di memoria da modificare.
- Modificare la locazione e premere ENTER per passare alla successiva.
- Premere ESC per terminare l'aggiornamento e restituire il controllo alla barra menu.

Anche in questo caso l'aggiornamento avverrà nella finestra della memoria con il solito formato:

hhhh : hh

dove hhhh rappresenta l'indirizzo in esadecimale del byte, mentre hh è il suo valore (modificabile) anche questo in esadecimale.



<Figura 1.4> Il menu della memoria.

1.18 [Options]/[Memoria]/[Azzera]

Può accadere che sia necessario azzerare (assegnare il valore 0) il contenuto dell'intera memoria.
Ciò è reso possibile dal comando [Options]/[Memoria]/[Azzera]. Le operazioni da compiere per inizializzare la memoria sono:

- Attivare il menu options tramite la voce [Options] della barra menu.
- Attivare il menu della memoria selezionando la voce [Memoria] del menu options.
- Attivare il comando di azzeramento tramite la voce [Azzera] del menu della memoria. Z80 Simulation chiede di confermare l'operazione: rispondere di conseguenza.

Se l'azzeramento avviene, Z80 Simulation risponde con un messaggio e visualizza nella finestra della memoria le prime 18 locazioni (ovviamente azzerate).

1.19 [Options]/[Memoria]/[Copia]

Se il contenuto di una zona di memoria deve essere copiato in un'altra zona di memoria, il comando da utilizzare è [Options]/[Memoria]/[Copia]. I passi da compiere per fare la copia di un blocco di memoria sono i seguenti:

- Attivare il menu options tramite la voce [Options] della barra menu.
- Attivare il menu della memoria selezionando la voce [Memoria] del menu options.
- Attivare il comando di copia blocco tramite la voce [Copia] del menu della memoria.
- Inserire i seguenti dati (in esadecimale): indirizzo d'inizio del blocco sorgente; indirizzo d'inizio del blocco destinazione; dimensione (in byte) del blocco.
- Z80 Simulation chiede di confermare l'operazione: rispondere di conseguenza.

Z80 Simulation dopo aver effettuato la copia risponde con il numero di byte copiati.
Si può verificare il caso in cui mentre si immettono i dati, Z80 Simulation automaticamente modifichi tali valori: ciò significa che i dati inseriti dall'utente non erano compatibili con la posizione o la dimensione della memoria.

1.20 [Options]/[Memoria]/[Load]

Se il contenuto della memoria era stato precedentemente salvato su disco con il comando [Options]/[Memoria]/[Save], esso può essere ricaricato tramite il comando [Options]/[Memoria]/[Load]. Le operazioni da compiere per caricare il contenuto della memoria da file sono le seguenti:

- Attivare il menu options tramite la voce [Options] della barra menu.
- Attivare il menu della memoria selezionando la voce [Memoria] del menu options.
- Attivare il comando di caricamento tramite la voce [Load] del menu della memoria.
- Inserire il nome del file da cui caricare il contenuto della memoria.

Se il caricamento della memoria avviene senza errori, Z80 Simulation visualizza un messaggio indicando il numero di byte caricati, e mostra nella finestra della memoria i primi 18 byte. Se invece durante il caricamento si verifica un errore di lettura da disco, Z80 Simulation mostra un messaggio di errore.

1.21 [Options]/[Memoria]/[Save]

Così come è possibile salvare su disco il programma contenuto nell'editor, così è anche possibile salvare il contenuto della memoria. Il comando utile a far ciò è [Options]/[Memoria]/[Save] e le operazioni da compiere sono le seguenti:

- Attivare il menu options tramite la voce [Options] della barra menu.
- Attivare il menu della memoria selezionando la voce [Memoria] del menu options.
- Attivare il comando di registrazione tramite la voce [Save] del menu della memoria.
- Inserire il nome del file su cui registrare il contenuto della memoria.

Se la registrazione avviene senza errori, Z80 Simulation visualizza il numero di byte registrati, altrimenti visualizza un messaggio d'errore.

1.22 [Options]/[Memoria]/[Posiziona]

L'ultimo dei comandi del menu della memoria è [Options]/[Memoria]/ [Posiziona]: tramite questo comando è possibile cambiare l'indirizzo dei 16K di memoria. Ciò significa, che se un particolare programma ha bisogno di accedere alla memoria con indirizzi ad esempio a partire da #B000, è sufficiente utilizzare questo comando e dare come indirizzo iniziale #B000.
Si noti, che il fatto di traslare la memoria, non comporta la perdita dei dati presenti in essa: l'unica differenza è che ad una locazione a cui ci si riferiva con un certo indirizzo, dopo l'esecuzione del comando ci si riferirà con un altro.

L'indirizzo iniziale della RAM può essere settato anche tramite il SETUP e quindi rimanere permanente.

1.23 ESECUZIONE [Options]/[Goto line]

Abbiamo visto, che per eseguire un programma è sufficiente selezionare il comando [Run], che attiva l'esecutore partendo dalla prima linea del codice.
Tuttavia, si potrebbe presentare la necessità di eseguire un programma partendo da una linea particolare scelta dall'utente. Il comando necessario a far ciò è [Options]/[Gotoline] e le operazioni da compiere sono le seguenti:

- Attivare il menu options tramite la voce [Options] della barra menu.
- Attivare il comando di esecuzione tramite la voce [Goto line] del menu Options.
- Inserire il numero della linea da cui iniziare l'esecuzione. Tale valore può essere scritto in decimale, oppure in esadecimale se preceduto dal simbolo #.

Si noti che il comando [Options]/[Goto line] differisce dal comando [Run], solo per il fatto che nel Program Counter (Registro PC) non viene caricato il valore 1, bensì il valore specificato dall'utente come numero di linea da cui partire: ciò vuol dire che tutte le considerazioni fatte in precedenza per il comando [Run] restano valide per questo comando.

1.24 L'ASSEMBLATORE [Options]/[Assembler]

Il codice generato dal compilatore di Z80 Simulation non è accessibile all'esterno. Del resto, tale codice non sarebbe di nessuna utilità all'utente, poiché, è progettato per essere eseguito solo sull'esecutore di Z80 Simulation. E' invece vantaggioso, il poter ricavare, a partire dal programma presente nell'editor, il codice macchina ad esso associato, ovvero quel codice direttamente eseguibile dai microprocessori Z80.
Questa operazione è possibile grazie alla presenza dell'assemblatore che si attiva mediante il comando [Options]/[Assembler]. Non è inoltre difficile, utilizzando l'assemblatore, rendersi conto, che questo componente altro non è se non il compilatore stesso, che attivato in questo modo produce oltre al codice eseguibile dall'esecutore, anche il codice macchina. A dimostrazione di

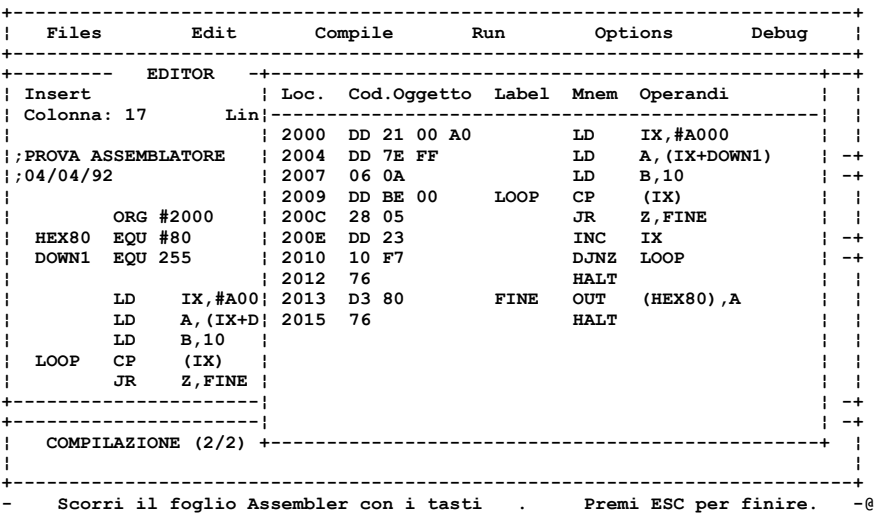
ciò, sta il fatto, che dopo aver assemblato un programma, esso può essere eseguito da Z80 Simulation senza essere preventivamente compilato.
Vediamo di seguito come operare per assemblare un programma:

- Attivare il menu options tramite la voce [Options] della barra menu.
- Attivare l'assemblatore mediante la voce [Assembler] del menu options.
- Inserire l'indirizzo iniziale della memoria in cui dovrà essere posto il codice macchina. Tale operazione non ha effetto se all'inizio del programma è presente la pseudo-istruzione ORG.

A questo punto l'assemblatore inizia ad assemblare il programma. Terminato il lavoro di assemblaggio, Z80 Simulation visualizza un menu da cui è possibile selezionare il formato con cui il codice macchina deve essere emesso.
Discutiamo ora i quattro formati di output disponibili:

1. **Formato video** (Fig. 1.5). Con questo formato il codice macchina viene visualizzato sul video. Se il codice occupa più di una pagina, lo si può scorrere con i tasti freccia. Per concludere la visualizzazione premere il tasto ESC.
2. **Foglio Assembler** (Fig. 1.6). Questo formato si presta bene alla stampa su carta: il codice macchina, unito al programma stesso viene impaginato e registrato su file. La dimensione delle pagine si stabilisce mediante il SETUP.
3. **Modulo continuo** (Fig. 1.7). Questo formato differisce dal precedente per il fatto di non impaginare l'output e di non contenere caratteri grafici. Viene consigliato quindi, in quei casi in cui non si disponga di una stampante con simboli grafici, oppure si abbia bisogno di maggiore velocità di stampa.
4. **Formato binario**. Questo formato prevede la scrittura su file del solo codice macchina. Risulta quindi difficilmente leggibile dall'utente, ma si presta molto bene alla scrittura del codice macchina su EPROM o altro dispositivo.

Si noti infine, che per i formati 2 e 3, se si volesse la stampa direttamente su stampante, è sufficiente inserire come nome di file PRN o LPT1.



<Figura 1.5> Formato video.

```
F O G L I O    A S S E M B L E R    Z 8 0

Pag. N°:  01    Descrizione:
Data:

Loc.  | Cod. Oggetto | Label  | Mnem. | Operandi          | Commento
------+--------------+--------+-------+-------------------+----------
      |              | HEX80  | EQU   | #0080             |
      |              | DOWN1  | EQU   | #00FF             |
2000  | DD 21 00 A0  |        | LD    | IX,#A000          |
2004  | DD 7E FF     |        | LD    | A,(IX+DOWN1)      |
2007  | 06 0A        |        | LD    | B,10              |
2009  | DD BE 00     | LOOP   | CP    | (IX)              |
200C  | 28 05        |        | JR    | Z,FINE            |
200E  | DD 23        |        | INC   | IX                |
2010  | 10 F7        |        | DJNZ  | LOOP              |
2012  | 76           |        | HALT  |                   |
2013  | D3 80        | FINE   | OUT   | (HEX80),A         |
2015  | 76           |        | HALT  |                   |

Autore:                    Osservazioni:

                                                Z80 Simulation
```

<Figura 1.6> Foglio Assembler.

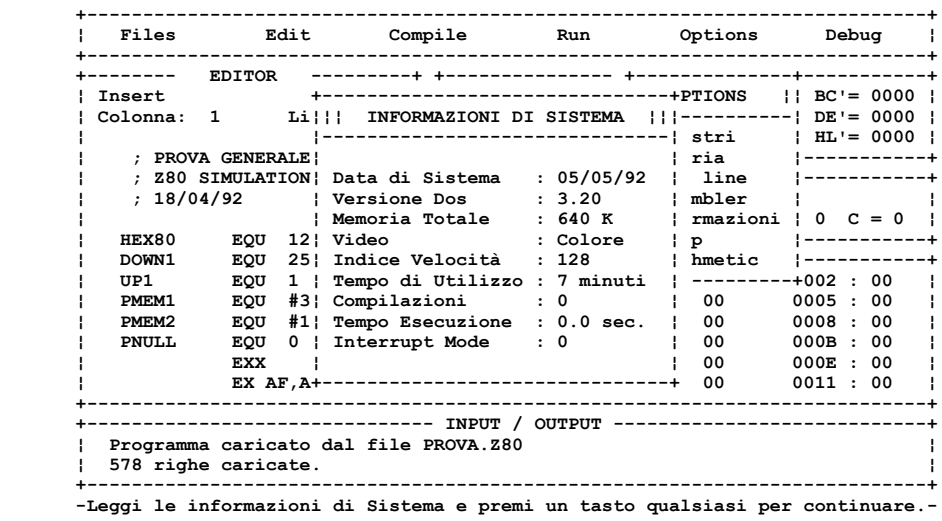
```
Loc.  | Cod. Oggetto | Label  | Mnem. | Operandi          | Commento
------+--------------+--------+-------+-------------------+----------
      |              | HEX80  | EQU   | #0080             |
      |              | DOWN1  | EQU   | #00FF             |
2000  | DD 21 00 A0  |        | LD    | IX,#A000          |
2004  | DD 7E FF     |        | LD    | A,(IX+DOWN1)      |
2007  | 06 0A        |        | LD    | B,10              |
2009  | DD BE 00     | LOOP   | CP    | (IX)              |
200C  | 28 05        |        | JR    | Z,FINE            |
200E  | DD 23        |        | INC   | IX                |
2010  | 10 F7        |        | DJNZ  | LOOP              |
2012  | 76           |        | HALT  |                   |
2013  | D3 80        | FINE   | OUT   | (HEX80),A         |
2015  | 76           |        | HALT  |                   |
      |              |        |       |                   |
                                                Z80 Simulation --
```

<Figura 1.7> Modulo continuo.

1.25 INFORMAZIONI [Options]/[Informazioni]

Tramite il comando [Options]/[Informazioni] è possibile visualizzare una serie di informazioni (Fig. 1.8). Vediamo adesso il significato di questi dati:

1. **Data di Sistema.** Si tratta della data espressa in giorno, mese, anno mantenuta dal DOS.
2. **Versione Dos**. Questa informazione rappresenta la versione del DOS su cui Z80 Simulation sta girando.
3. **Memoria Totale.** E' la quantità totale di memoria, espressa in Kbyte, presente sul sistema.
4. **Video.** Indica se Z80 Simulation sta lavorando in modalità colore o monocromatico. Tale modalità, la si può impostare mediante il SETUP. Tuttavia nel caso in cui Z80 Simulation rilevi la presenza della scheda video HERCULES, lavorerà in modo monocromatico prescindendo dal valore di SETUP.
5. **Indice Velocità.** E' un valore intero che permette di avere un'idea della velocità del sistema su cui Z80 Simulation sta girando.
6. **Tempo di Utilizzo.** Rappresenta il tempo, espresso in minuti, trascorso dal momento in cui si è avviato Z80 Simulation.
7. **Compilazioni.** E' il numero totale di compilazioni effettuate.
8. **Tempo di Esecuzione.** Indica la durata, espressa in secondi, dell'ultima esecuzione di programma.
9. **Interrupt Mode.** Individua il modo di Interruzione (0, 1 o 2) in cui si può trovare l'esecutore in seguito all'esecuzione delle istruzioni IM 0, IM 1, IM 2.



<Figura 1.8> Informazioni di sistema.

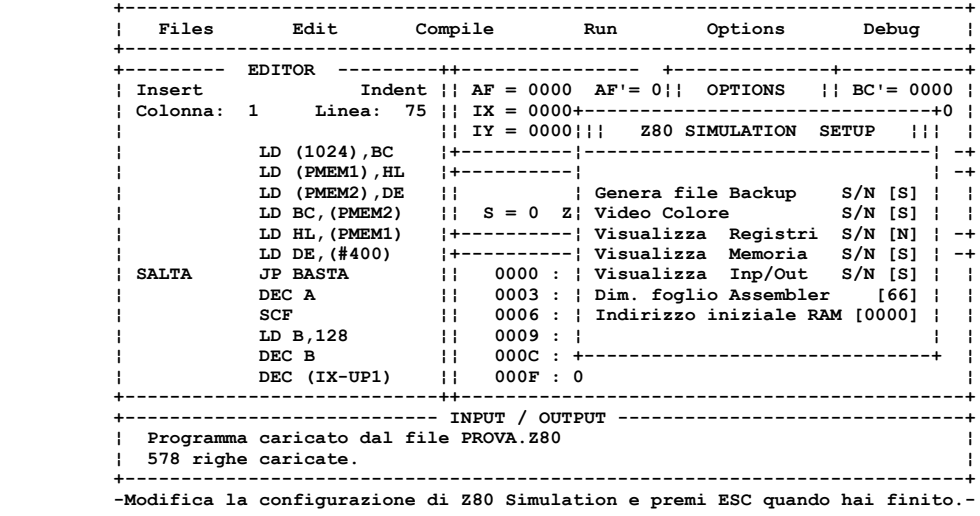
1.26 SETUP [Options]/[Setup]

Alcuni aspetti di Z80 Simulation possono essere configurati mediante il SETUP. Vediamo quindi come operare per modificare la configurazione:

- Attivare il menu options tramite la voce [Options] della barra menu.
- Attivare il SETUP mediante la voce [Setup] del menu options.
- Compare la finestra di SETUP (Fig. 1.9): modificare i vari valori racchiusi tra parentesi quadre; per passare al parametro successivo premere ENTER, mentre quando si ha finito premere ESC.
- Z80 Simulation chiede se la configurazione deve essere salvata su disco o meno: se si preme 'S', la configurazione diviene permanente, ovvero viene salvata sul file Z80.CFG e ricaricata al momento dell'avviamento; se si preme 'N' la nuova configurazione è temporanea, ovvero al prossimo avviamento verrà caricata la configurazione precedente.

Descriviamo di seguito, facendo riferimento alla finestra di SETUP (Fig. 1.9) quali sono gli aspetti configurabili:

1. **Genera file Backup S/N.** Se si specifica 'S', nelle operazioni di scrittura vengono creati i file di riserva con estensione BAK.
2. **Video Colore S/N.** Se si specifica 'N', Z80 Simulation al prossimo avviamento lavora in modalità monocromatica. Se si specifica 'S' la modalità dipenderà dal tipo di scheda grafica presente. Con ciò s'intende dire, che se la scheda è Hercules, la modalità è comunque monocromatica. Si noti che questo valore diviene attivo solo al prossimo avviamento di Z80 Simulation (a patto che si sia registrata la configurazione).



<Figura 1.9> Finestra di SETUP.

3. **Visualizza Registri S/N.**
4. **Visualizza Memoria S/N.**
5. **Visualizza Inp/Out S/N.**

Si tratta delle opzioni del debugger che sono descritte più avanti nel paragrafo dedicato ad esso. Il motivo per cui queste possono essere settate anche da SETUP, oltre che dal debugger, è evidente: permettere che il valore settato rimanga permanente.

6. **Dim. foglio Assembler.** Si tratta del numero di righe presenti in un foglio assembler. Tale numero deve essere compreso tra 35 e 99. In genere ogni foglio è di 66 righe.

7. **Indirizzo iniziale RAM.** E' lo stesso valore settabile mediante il comando [Options]/[Memoria]/[Posiziona], ovvero è l'indirizzo iniziale della memoria. Il vantaggio di settarlo da SETUP è la possibilità di renderlo permanente.

1.27 CALCOLI [Options]/[Arithmetic]

Spesso, durante l'esecuzione di Z80 Simulation, si presenta la necessità di fare delle operazioni matematiche o delle conversioni da decimale a esadecimale e viceversa. Ciò può essere fatto mediante i comandi del menu Arithmetic (Fig.1.10), attivabile con il comando [Options]/[Arithmetic]. Vediamo uno per uno questi comandi:

```
+-------------------------------------------------------------------------------+
¦    Files        Edit        Compile      Run       Options       Debug     ¦
+-------------------------------------------------------------------------------+
+---------  EDITOR  ---------++----------------  +-------------+-----------+
¦ Insert              Indent ¦¦ AF = 0000  AF'= 0¦¦  OPTIONS   ¦¦ BC'= 0000 ¦
¦ Colonna:  1     Linea: 211 ¦¦ IX = 0000  PC = 0+-------------¦ DE'= 0000 ¦
¦                            ¦¦ IY = 0000  SP = 4¦ Registri    ¦ HL'= 0000 ¦
¦         LD A,0             ¦+-----------------¦ Memoria     ¦-----------+
¦         RLCA               ¦+----------+-------------+line   ¦-----------+
¦         SCF                ¦¦          ¦ ARITHMETIC  ¦bler   ¦           ¦
¦         LD A,15            ¦¦  S = 0  Z+-------------¦mazioni ¦ 0  C = 0  ¦
¦         LD (IY-1),#11      ¦+----------¦ N1 + N2      ¦        ¦-----------+
¦  LOOP5  RLC C              ¦+----------¦ N1 - N2      ¦metic   ¦-----------+
¦         RLC (HL)           ¦¦   0000 : ¦ N1 * N2      ¦--------+002 : 00   ¦
¦         RLC (IY+DOWN1)     ¦¦   0003 : ¦ N1 / N2      ¦00      0005 : 00   ¦
¦         RLC A              ¦¦   0006 : ¦ Dec -> Hex   ¦00      0008 : 00   ¦
¦         RRC H              ¦¦   0009 : ¦ Hex -> Dec   ¦00      000B : 00   ¦
¦         RL D               ¦¦   000C : ¦ Cpl2  8 bit  ¦00      000E : 00   ¦
¦         RR (IX)            ¦¦   000F : ¦ Cpl2 16 bit  ¦00      0011 : 00   ¦
+---------------------------++----------+-------------+-------------------+
+------------------------------- INPUT / OUTPUT ---------------------------+
¦    N1: FFFF                                                               ¦
¦    N2: 00FE                    N1 / N2 = 0102   (resto = 0003)            ¦
+-------------------------------------------------------------------------------+
-    Seleziona con   .   Premi ENTER per confermare e ESC per annullare.     -
```

<Figura 1.10> Menu Arithmetic.

1. **[Options]/[Arithmetic]/[N1 + N2]**. Visualizza in esadecimale la somma dei 2 valori N1 e N2 inseriti dall'utente in esadecimale.
2. **[Options]/[Arithmetic]/[N1 - N2]**. Visualizza in esadecimale la differenza dei valori N1 e N2 inseriti dall'utente in esadecimale.
3. **[Options]/[Arithmetic]/[N1 * N2]**. Visualizza in esadecimale il prodotto dei valori N1 e N2 inseriti dall'utente in esadecimale.

4. **[Options]/[Arithmetic]/[N1 / N2]**. Visualizza in esadecimale il quoziente e il resto della divisione tra N1 e N2 inseriti dall'utente in esadecimale.
5. **[Options]/[Arithmetic]/[Dec → Hex]**. Effettua la conversione da decimale a esadecimale del valore inserito.
6. **[Options]/[Arithmetic]/[Hex → Dec]**. Effettua la conversione da esadecimale a decimale del valore inserito.
7. **[Options]/[Arithmetic]/[Cpl2 8 bit]**. Effettua il complemento a 2 a 8 bit del dato inserito dall'utente in esadecimale.
8. **[Options]/[Arithmetic]/[Cpl2 16 bit]**. Effettua il complemento a 2 a 16 bit del dato inserito dall'utente in esadecimale.

1.28 IL DEBUGGER [Debug]

Siamo finalmente giunti all'ultima voce della barra menu: tramite essa possiamo modificare le opzioni del debugger. Molto spesso, fino a questo momento, ci siamo riferiti al debugger come fosse uno strumento a sé stante. In realtà, esso altro non è, che una serie di parametri (chiamati opzioni) che influenzano il modo di funzionamento dell'esecutore. Ecco di seguito come settare le opzioni di debugger:

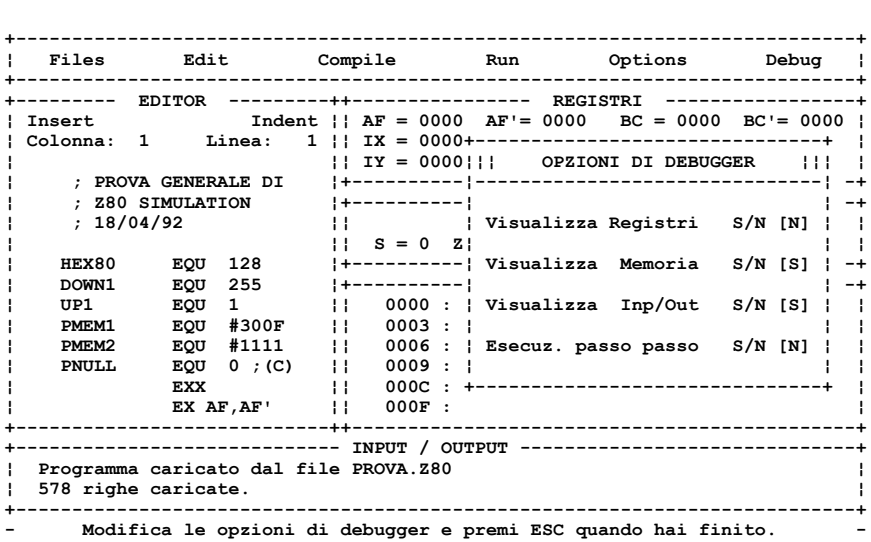
- Attivare il comando di debugger tramite la voce [Debug] della barra menu.
- Compare sul video una finestra (Fig. 1.11) nella quale sono mostrati i valori attuali delle opzioni di debugger: modificarle secondo le necessità. Per passare alle opzioni successive premere ENTER.
- Per chiudere la finestra e quindi restituire il controllo alla barra menu, premere il tasto ESC.

Vediamo ora come ogni opzione di debugger influenzi l'esecutore:

1. **Visualizza Registri S/N.** Se si specifica 'S', durante l'esecuzione del programma vengono visualizzate le modifiche sui registri e sui flag. Se si specifica 'N', il valore reale dei flag e dei registri si può vedere solo quando l'esecuzione del programma termina. Questa opzione settata a 'N' conferisce all'esecutore una maggiore velocità.
2. **Visualizza Memoria S/N.** Se si specifica 'S', durante l'esecuzione del programma, vengono visualizzate le locazioni di memoria modificate.

3. **Visualizza Inp/Out.** Se si specifica 'S', durante l'esecuzione del programma vengono visualizzate le richieste di Input/Output.
4. **Esecuz. passo passo.** Se si specifica 'S', l'esecutore prima di eseguire un'istruzione, aspetta che venga premuto il tasto ENTER. Inoltre l'istruzione da eseguire alla pressione di ENTER, viene evidenziata nell'editor. Si noti che se questa opzione viene settata a 'S', anche le altre opzioni vengono settate automaticamente a 'S'.

Lo stato delle prime tre opzioni può anche essere settato mediante il SETUP e quindi rimanere permanente.



<Figura 1.11> Opzioni di debugger.

CAPITOLO 2

Obiettivi del capitolo:

- Descrizione dei comandi dell'editor

2.1 L'EDITOR

L'editor è uno degli strumenti più importanti di Z80 Simulation: grazie ad esso possiamo scrivere e modificare i programmi da provare. Nella parte superiore della finestra in cui è collocato, vengono indicate la posizione del cursore, nonché lo stato della modalità di inserimento e di identazione. Per attivare l'editor è sufficiente selezionare la voce [Edit] della barra menu. Una volta al suo interno sono disponibili una serie di comandi che facilitano l'editing del programma. Ci occupiamo della descrizione di questi nei paragrafi seguenti.

2.2 TASTI DI MOVIMENTO DEL CURSORE

Il cursore può essere spostato in una delle quattro direzioni possibili mediante l'uso dei tasti freccia o del mouse qualora questo lavori in emulazione dei tasti freccia.
Si possono tuttavia compiere spostamenti più complessi usando i seguenti tasti:

1. **[Home].** Sposta il cursore all'inizio della riga su cui si trova.
2. **[End].** Sposta il cursore alla fine della riga su cui si trova.
3. **[Tab].** Sposta il cursore a destra sulla prima posizione che sia multiplo di 5.
4. **[Shift]+[Tab].** Sposta il cursore a sinistra sulla prima posizione che sia multiplo di 5.
5. **[PgUp].** Sposta il cursore di 10 righe in alto.
6. **[PgDn].** Sposta il cursore di 10 righe in basso.
7. **[Ctrl]+[PgUp].** Sposta il cursore allo inizio del programma.
8. **[Ctrl]+[PgDn].** Sposta il cursore alla fine del programma.

2.3 TASTI DI CANCELLAZIONE

I seguenti tasti sono utilizzati per cancellare singoli caratteri o intere righe:

1. **[Canc]**. Cancella il carattere sotto il cursore. I caratteri alla destra del cursore sono spostati verso sinistra di una posizione, mentre il cursore rimane fermo.
2. **[Ctrl]+[G]**. Come il tasto [Canc].
3. **[BackSpace]**. Cancella il carattere che si trova alla sinistra del cursore. Il cursore e i caratteri alla sua destra sono spostati di una posizione verso sinistra.
4. **[F1]**. Cancella la linea su cui si trova il cursore. Le linee sottostanti vengono portate sopra di una posizione.
5. **[Ctrl]+[Y]**. Come il tasto [F1].

2.4 TASTI DI INSERIMENTO

Il tasto [INS] serve per attivare e disattivare il modo inserimento. Se è attivo, nella parte superiore dell'editor compare la parola "Insert" e quando si digita un carattere, questo viene inserito in mezzo a quelli già presenti, spostando quelli alla destra del cursore di una posizione verso destra. Inoltre se l'inserimento è attivo, premendo il tasto [ENTER], tutte le linee al di sotto di quella su cui si trova il cursore sono spostate verso il basso: viene dunque inserita una linea vuota e il cursore si posiziona su di questa. Per inserire invece una linea sopra quella su cui si trova il cursore, è sufficiente premere il tasto [F2]: la linea corrente e quelle sottostanti sono spostate in basso, mentre il cursore si posiziona (in realtà rimane fermo) sulla nuova linea. Il tasto [Ctrl]+[N] è equivalente al tasto [F2].

2.5 GESTIONE BLOCCHI

Molto spesso, risulta utile compiere delle operazioni su blocchi contigui di linee. Vediamo di seguito quali sono le operazioni sui blocchi:

1. **[F3].** Contrassegna la linea su cui si trova il cursore come inizio del blocco.
2. **[F4].** Contrassegna la linea su cui si trova il cursore come fine del blocco.

Si noti che quando si definisce un blocco le linee in esso contenute hanno un colore diverso rispetto alle altre.

3. **[F5].** Copia il blocco di linee alla posizione del cursore. La linea corrente e le linee a essa sottostanti sono spostate in basso di un numero di righe pari alla dimensione del blocco.
4. **[F6].** Sposta il blocco di linee alla posizione del cursore.
5. **[F7].** Legge un blocco da file e lo copia a partire dalla linea corrente. Il file che contiene il blocco può essere un qualsiasi file ASCII.
6. **[F8].** Scrive il blocco di linee su un file.

2.6 SOSTITUZIONE DI STRINGHE

Per sostituire automaticamente una stringa con un'altra stringa, si deve utilizzare il comando [Ctrl]+[S]. I passi da compiere sono i seguenti:

1. Premere il tasto [Ctrl]+[S].
2. Inserire la stringa da sostituire e la stringa con cui sostituire.
3. Se Z80 Simulation trova una stringa da sostituire nel testo, la evidenzia e chiede se la sostituzione deve essere effettuata.
4. Il passo 3 viene ripetuto fino all'esaurimento delle stringhe da sostituire.

Alla fine del comando Z80 Simulation mostra il numero di stringhe sostituite. Si noti che le stringhe da sostituire, vengono ricercate a partire dalla posizione del cursore.

2.7 L'AUTOIDENTAZIONE

Per autoidentazione si intende, la capacità dell'editor, di incolonnare la prima parola della linea con quella della linea precedente. In altre parole, quando è attiva l'autoidentazione, la pressione di [ENTER] sposta il cursore alla linea successiva e sotto il primo carattere della prima parola della linea soprastante. Per attivare e disattivare l'autoidentazione premere il tasto [F9]. Se è attiva, nella parte superiore dell'editor, compare il messaggio "Indent".

2.8 L'EDITOR HELP

Tutti i comandi fin qui illustrati, vengono descritti schematicamente, direttamente da Z80 Simulation nell'editor help (Fig. 2.1). Questo può essere

visualizzato tramite la pressione del tasto [F10] (naturalmente a patto che sia attivo l'editor).

2.9 COME USCIRE DALL'EDITOR

L'unico modo per uscire dall'editor e quindi per restituire il controllo alla barra menu, è di premere il tasto [ESC].

```
+-------------------------------------------------------------------------+
| ||       Z 8 0    S I M U L A T I O N    E D I T O R    H E L P       || |
+-------------------------------------------------------------------------+
+-----------------------------------++-----------------------------------+
|                           +-------++-------+                           |
|  Cancella linea           | F1    ||    F2 |  Inserisce linea          |
|  Contrass. inizio blocco  | F3    ||    F4 |  Contrass. fine blocco    |
|  Copia blocco             | F5    ||    F6 |  Sposta blocco            |
|  Legge blocco da file     | F7    ||    F8 |  Scrive blocco su file    |
|  Auto-Indent ON/Off       | F9    ||   F10 |  Visual. Editor Help      |
+---------------------------+-------++-------+---------------------------+
|  Muove inizio programma   | ^PgUp ||  ^PgDn |  Muove fine programma     |
|  Cancella carattere       | Del   || BackSp |  Canc. caratt. sinistra   |
|  Muove sinistra destra    |       ||       |  Muove sopra sotto        |
|  Muove inizio linea       | Home  ||   End |  Muove fine linea         |
|  Cancella linea           | ^Y    ||    ^N |  Inserisce linea          |
|  Muove pagina sopra       | PgUp  ||  PgDn |  Muove pagina sotto       |
|  Tabulazione a destra     | Tab   || Sh+Tab |  Tabulazione a sinistra   |
|  Cancella carattere       | ^G    ||    ^S |  Sostituisce stringa      |
|  Inserimento ON/off       | Ins   ||   Esc |  Esce da Editor           |
|                           +-------++-------+                           |
+-----------------------------------++-----------------------------------+
+-------------------------------------------------------------------------+
| || Consulta Editor HELP e premi un tasto qualsiasi quando hai finito. || |
+-------------------------------------------------------------------------+
```

<Figura 2.1> L'editor help.

CAPITOLO 3

Obiettivi del capitolo:

- Le strutture di memorizzazione
- Il linguaggio di programmazione di Z80 Simulation
- Le istruzioni dell'Assembler Z80

3.1 IL LINGUAGGIO DI Z80 SIMULATION

Per scrivere programmi che possano essere compilati ed eseguiti da Z80 Simulation, è necessario conoscere la sintassi e la semantica di ciascuna istruzione. Tali istruzioni sono identiche a quelle utilizzate dal microprocessore Z80 o al più possono differire da queste solo per alcuni particolari che comunque sono evidenziati nei prossimi paragrafi. Proprio in virtù di questa similitudine, spesso nel resto del testo, si utilizzerà il termine "microprocessore" al posto del termine "esecutore di Z80 Simulation".
Prima di descrivere una per una le istruzioni, vediamo quali sono gli oggetti su cui operano (registri, flag, memoria, stack, e porte di Input/Output) e il formato generale di una linea di programma.

3.2 I REGISTRI

I registri servono per memorizzare in modo temporaneo i dati da elaborare dal microprocessore. Questi si presentano perciò, come fossero delle locazioni di memoria di 8 o 16 bit a seconda che il registro sia semplice o doppio.
I registri semplici di Z80 Simulation sono:

A, B, C, D, E, H, L, F, I, R

mentre quelli doppi sono:

AF, BC, DE, HL, IX, IY, SP, PC.

Si noti comunque, che i registri doppi AF, BC, DE e HL non sono altro che i registri A, F, B, C, D, E, H, L combinati a due a due. Ciò vuol dire che un'operazione del tipo:

```
LD HL,#12FE  ;carica #12FE in HL
```

equivale a due istruzioni del tipo:

```
LD H,#12     ;carica #12 in H
LD L,#FE     ;carica #FE in L
```

Inoltre questi ultimi registri, sono presenti in due blocchi: quello formato dai registri AF, BC, DE e HL, e quello degli omologhi che indicheremo con AF', BC', DE' e HL'. Si passa da un blocco all'altro mediante le istruzioni:

```
EXX
EX AF,AF'
```

Con ciò si mette in evidenza il fatto che i registri normali non possono essere utilizzati contemporaneamente ai rispettivi omologhi e che quindi Z80 Simulation, nella sintassi delle istruzioni non distingue tra normali e omologhi. Sarebbe infatti scorretta l'istruzione:

```
LD HL',#12FE
```

Diamo adesso una breve descrizione dei ruoli dei registri:

1. **Il registro A (8 bit).** E' il più importante dei registri del microprocessore, in quanto partecipa al funzionamento di un grandissimo numero di operazioni e accetta ogni forma di indirizzamento. Molto spesso ci riferiremo ad esso con il nome di accumulatore.
2. **I registri IX e IY (16 bit).** Possono essere utilizzati in tutti quei casi in cui sia lecito usare il registro HL. In più, questi possono realizzare il cosiddetto indirizzamento indicizzato, di cui ci occuperemo nella descrizione della memoria.
3. **I registri B, C, D, E, H, L (8 bit).** Sono semplici registri a 8 bit che vengono sfruttati sia per contenere temporaneamente i dati dell'elaborazione, sia, uniti a coppie, per puntare alla memoria.
4. **Il registro SP (16 bit).** Si tratta dello Stack Pointer, ovvero del puntatore alla cima dello stack. Approfondiremo il discorso su questo registro quando parleremo dello stack.
5. **Il registro PC (16 bit).** Questo è un registro essenziale per il funzionamento del microprocessore, poiché, il suo compito è quello di puntare all'indirizzo della prossima istruzione che verrà eseguita. Proprio agendo su questo registro, le istruzioni di salto riescono a cambiare il flusso del programma. In realtà, in Z80 Simulation, il Program Counter ha

una funzione leggermente diversa da quella che ha nel microprocessore vero e proprio: in esso, non punta a un indirizzo di memoria, bensì a una linea di programma contenuto nell'EDITOR.

6. **Il registro I (8 bit).** E' utilizzato per la gestione del modo 2 d'interruzione. Tuttavia in Z80 Simulation non ha alcun significato, dato che questo non gestisce le interruzioni.
7. **Il registro R (8 bit).** Come il precedente ha un utilizzo molto particolare: contiene un indirizzo in continua evoluzione, utilizzato per il rinfresco della memoria. Anche in questo caso questo registro non ha alcun significato in Z80 Simulation.
8. **Il registro F (8 bit).** E' il registro di stato del microprocessore. I suoi bit sono denominati flag e lo stato di questi è determinato in base al risultato di alcune operazioni.

3.3 I FLAG

Vediamo adesso il significato di ciascun bit del registro F:

1. **Il bit 0 (C).** E' il flag del carry che in genere è settato a 1 quando si verifica un riporto sull'ultimo bit. Inoltre è alterato dalle operazioni di shift e dalle operazioni logiche.
2. **Il bit 1 (N).** E' il flag N che viene settato quando viene effettuata una sottrazione o un decremento. Non può essere testato mediante i salti condizionati, ma è sfruttato dalla operazione DAA.
3. **Il bit 2 (P/V).** E' il flag di parità o di overflow. Quando lavora come flag di parità (in genere nelle operazioni logiche e negli shift) viene posto a uno quando il numero dei bit posti a 1 nel risultato è pari. Se invece lavora come flag di overflow (in genere nelle operazioni aritmetiche) viene settato a 1 quando il segno del risultato è errato. Supponiamo ad esempio di voler sommare -1 a -128. Le istruzioni sono:

```
LD A,255          ; -1=255 in complemento a 2
ADD A,128         ; -128=128 in complemento a 2
```

Il risultato corretto sarebbe -129 ma invece nel registro A troviamo il valore 127. In questo caso il flag P/V sarà posto a 1.

4. **Il bit 3.** Questo bit non è un flag e non ha alcun significato. Il suo valore è casuale.
5. **Il bit 4 (H).** E' il flag dell'half-carry e indica il riporto che avviene al bit 3 dell'accumulatore. Non può essere testato con le istruzioni di salto condizionato, ma è sfruttato dall'operazione DAA.
6. **Il bit 5.** Questo bit non è un flag e non ha alcun significato. Il suo valore è casuale.
7. **Il bit 6 (Z).** E' il flag dello zero: viene posto a 1 nel momento in cui il risultato di un'operazione è 0.
8. **Il bit 7 (S).** E' il flag del segno. Quando viene posto a 1 significa che il risultato dell'operazione è negativo (in altre parole il bit più significativo del risultato è uguale a 1).

3.4 LA MEMORIA

Il microprocessore per poter memorizzare i dati dell'elaborazione, qualora i registri non siano più sufficienti, utilizza la memoria. Z80 Simulation ne ha una di 16 Kbyte, i cui indirizzi sono contigui, ma variabili dall'utente mediante il comando dell'ambiente integrato [Options]/[Memoria]/[Posiziona] il cui funzionamento è stato già descritto.
Per accedere ad un byte di memoria ci sono più modi di indirizzamento:

1. **Indirizzamento diretto.** In questo modo di indirizzamento, viene specificato direttamente, mediante una costante, l'indirizzo della locazione a cui si vuole accedere. Se si vuole ad esempio caricare il contenuto del registro A nella locazione di indirizzo #2FCB si scrive:

   ```
   LD (#2FCB),A
   ```

2. **Indirizzamento indiretto.** In questo caso l'indirizzo della locazione a cui si vuole accedere, viene specificato mediante un doppio registro (registro puntatore). L'equivalente dell'esempio precedente è:

   ```
   LD HL,#2FCB
   LD (HL),A
   ```

3. **Indirizzamento indicizzato**. E' una tecnica di indirizzamento molto simile alla precedente: l'indirizzo è ottenuto come somma (algebrica) tra uno dei registri IX o IY ed una costante a 8 bit che chiameremo spiazzamento. Si noti comunque, che la costante viene rappresentata in complemento a 2 e che quindi lo spiazzamento deve essere compreso tra -128 e 127. Le seguenti istruzioni caricano il valore di A nel byte di indirizzo pari a quello contenuto in IX diminuito di uno:

```
LD (IX+255),A o anche LD (IX-1),A
```

mentre queste altre caricano il valore #A5 nella locazione di indirizzo 300:

```
LD IY,200
LD (IY+100),#A5
```

Per concludere il discorso sulla memoria, si noti che un qualsiasi tipo di accesso ad una locazione non presente, implica l'arresto del programma con la visualizzazione di un messaggio d'errore.

3.5 LO STACK

Una delle difficoltà della programmazione dei microprocessori, consiste nella gestione dello spazio della memoria. Bisogna infatti tener conto degli indirizzi in cui si collocano i dati. Con lo stack è invece tutto più semplice. Questo è una struttura LIFO (comunque residente in memoria), in cui i dati vengono immessi (istruzione PUSH) e poi prelevati (istruzione POP) in ordine inverso da quello con cui sono stati inseriti, senza dover tenere conto dell'indirizzo di memoria in cui saranno destinati.
Lo stack gioca un ruolo importante, anche nella gestione delle chiamate dei sottoprogrammi. Infatti: quando si esegue una chiamata a sottoprogramma (istruzione CALL), il microprocessore salva il contenuto del registro PC nello stack; quando bisogna ritornare all'istruzione successiva alla chiamata (istruzione RET), il microprocessore carica nel PC il dato prelevato dalla cima dello stack. La cima dello stack viene puntata dal registro SP. Tenendo conto che lo stack cresce da indirizzi alti a indirizzi bassi, possiamo dire che: ad ogni

deposito nello stack, corrisponde il decremento di SP; ad ogni prelievo corrisponde l'incremento di SP.
All'avviamento di Z80 Simulation, la cima dello stack viene automaticamente posta alla fine della memoria, ma può essere riposizionata all'indirizzo nnnn mediante l'istruzione LD SP,nnnn.

3.6 LE PORTE DI INPUT/OUTPUT

Nel microprocessore Z80, le porte di INPUT/OUTPUT sono 256 con indirizzi che vanno da 0 a 255. Tramite queste, il microprocessore comunica con i dispositivi periferici, ed in particolare scrive su di un dispositivo con un'istruzione di output, e legge da esso con un'istruzione di input.
Tuttavia, Z80 Simulation non può comunicare con l'esterno. In esso le istruzioni di input provocano (qualora il debugger lo permetta), la visualizzazione nella finestra di INPUT/OUTPUT del messaggio:

```
INP (porta #pp): 00
```

L'utente può quindi digitare un valore che sarà considerato come input dalla porta pp. Le istruzioni di output causano invece la visualizzazione del messaggio:

```
OUT (porta #pp): nn
```

dove nn è il valore scritto sulla porta pp.

3.7 IL FORMATO DELLE COSTANTI

Abbiamo avuto già modo di vedere, specialmente negli esempi, che le costanti numeriche possono essere scritte in 2 formati:

1. **Formato decimale**. Si specifica il numero con cifre decimali. Per esempio:

```
LD A,15
LD HL,30215
SUB 47
```

2. **Formato esadecimale**. Si specifica il numero con cifre esadecimali precedute dal simbolo #. Per esempio:

```
        LD D,#2F
LD BC,#C1ED
CP #42
```

Si possono anche definire delle costanti simboliche tramite la pseudo-istruzione EQU il cui funzionamento è descritto più avanti.

3.8 I COMMENTI

Spesso, all'interno del programma, si ha la necessità di inserire dei commenti. In Z80 Simulation questi iniziano con il simbolo ";" e possono occupare un'intera linea o condividerla con un'istruzione. Sono esempi di commento i seguenti:

```
; SALVATAGGIO NELLO STACK

        PUSH HL         ;SALVA HL
        PUSH AF         ;SALVA AF

; FINE SALVATAGGIO

        LD IX,0         ;AZZERAMENTO DI IX
        CALL STAMPA     ;CHIAMA STAMPA
```

3.9 LE LABEL

Si è già osservato, che mentre nel microprocessore Z80 il registro PC contiene l'indirizzo di memoria della prossima istruzione da eseguire, in Z80 Simulation, questo registro contiene il numero della prossima riga di programma da eseguire. Si deduce perciò, che l'operando delle istruzioni di salto non è un indirizzo, ma un numero di riga: se per esempio si vuole saltare alla linea 315, basta scrivere:

```
JP 315
```

Tuttavia, questo metodo è abbastanza scomodo, poiché non solo è necessario tener conto della posizione in cui si collocano le istruzioni, ma, anche un semplice inserimento di una riga, può comportare la modifica di molte istruzioni di salto.
Per evitare questi problemi, il compilatore di Z80 Simulation gestisce le cosiddette label. Queste, altro non sono, che delle costanti, che assumono il valore del numero della linea in cui vengono specificate. Una label deve essere composta da lettere e cifre, non può superare 6 caratteri di lunghezza e non deve iniziare con una cifra.
Vediamo adesso l'uso delle label nel seguente programma che inizializza la memoria dall'indirizzo #0000 all'indirizzo #00FF, con il valore 32:

```
                ;INIZIALIZZA MEMORIA (0000-00FF)
                ;CON IL VALORE 32

                ORG #B000

VALUE           EQU 32              ; VALUE ← 32

                LD HL,0             ; HL ← 0
LOOP            LD (HL),VALUE       ; (HL) ← value
                INC L               ; incrementa L e quindi HL
                JP NZ,LOOP          ; se L<>0 torna a LOOP
                HALT
```

Si noti infine, che le label non possono essere usate in luogo delle costanti nelle istruzioni che non siano di salto, ovvero sarebbe scorretta l'istruzione

```
LD A,LOOP
```

dove LOOP sia una label.

3.10 LE CONDIZIONI

Il microprocessore può eseguire dei salti condizionati basandosi sullo stato dei flag. Vediamo adesso quali sono le condizioni:

1. Condizione **NZ** (Z=0). Questa condizione è vera quando il flag dello zero è 0.
2. Condizione **Z** (Z=1). Questa condizione è vera quando il flag dello zero è 1.
3. Condizione **NC** (C=0). Questa condizione è vera quando il flag del carry è 0.
4. Condizione **C** (C=1). Questa condizione è vera quando il flag del carry è 1.
5. Condizione **PO** (P/V=0). Questa condizione è vera quando il flag della parità/overflow è 0.
6. Condizione **PE** (P/V=1). Questa condizione è vera quando il flag della parità/overflow è 1.
7. Condizione **P** (S=0). Questa condizione è vera quando il flag del segno è 0.
8. Condizione **M** (S=1). Questa condizione è vera quando il flag del segno è 1.

Ecco alcuni esempi di salto condizionato:

```
JP Z,LOOP          ; salta se Z=1
CALL PE,STAMPA     ; chiama se P/V=1
RET NC             ; ritorna se C=0
JR C,LOOP          ; salta se C=1
JP M,LOOP          ; salta se S=1
```

3.11 I SALTI ASSOLUTI E RELATIVI

Si è detto in precedenza, che quando il microprocessore esegue un'istruzione di salto, in realtà altro non fa se non caricare l'indirizzo di salto nel registro PC.
Questo modo di saltare lo diciamo assoluto, per distinguerlo da quello relativo, in cui l'operando dell'istruzione di salto, non è l'indirizzo a cui saltare, bensì un valore compreso fra -128 e 127, che verrà sommato al valore del PC.

In particolare poi, in Z80 Simulation, dove non si parla di indirizzi, ma di numeri di linea, un'istruzione del tipo JR 20, fa saltare l'esecutore 20 linee più avanti della linea successiva a quella che contiene la JR. Un'istruzione del tipo JR 253 fa tornare indietro l'esecutore di 3 linee (-3 in complemento a 2 è 253) rispetto alla linea successiva a quella contenente la JR. Si noti comunque, che se l'operando di un'istruzione di salto relativo, è una label, questo non vuol dire che l'esecutore salterà avanti o indietro per un numero di linee pari al valore della label, bensì salterà (come per il salto assoluto) sulla linea in cui è stata definita la label.

3.12 LE PSEUDO-ISTRUZIONI

Le pseudo-istruzioni sono delle istruzioni che non producono codice eseguibile, ma hanno particolari funzioni durante la compilazione del programma. Z80 Simulation ne ha soltanto due, vediamole di seguito:

1. **ORG nnnn**. E' una pseudo-istruzione che influenza solo l'assemblatore: essa fissa l'indirizzo a partire dal quale verrà posto il codice macchina. Se per esempio si specifica nel programma ORG #A200, il codice macchina inizia dall'indirizzo #A200. La pseudo-istruzione ORG può essere ripetuta più volte nel programma.
2. **cost EQU nnnn**. Questa pseudo-istruzione serve per definire una costante simbolica. Vediamo alcuni esempi:

```
HEX80     EQU #80          ; HEX80 ← #80
BUFFER    EQU #FDE1        ; BUFFER ← #FDE1
```

Anche le costanti simboliche, come le label, devono essere composte da lettere e cifre, non possono superare 6 caratteri di lunghezza e non devono iniziare con una cifra.

3.13 LE ISTRUZIONI

Nei prossimi paragrafi sono descritte tutte le istruzioni del microprocessore Z80. Il titolo di ciascun paragrafo è l'istruzione stessa, mentre all'interno di questo viene descritta la sintassi, lo scopo, l'influenza sui flag. Spesso sono

presenti alcune note sul comportamento dell'istruzione e sulle eventuali differenze tra microprocessore vero e proprio e Z80 Simulation.

3.14 I SIMBOLI UTILIZZATI

Nel descrivere le varie istruzioni utilizzeremo i seguenti simboli:

1. **"REG8" (registri a 8 bit).** Con questo simbolo indicheremo i registri a 8 bit.
2. **"REG16" (registri a 16 bit).** Con questo simbolo indicheremo i registri a 16 bit.
3. **"PMEM" (puntatore alla memoria).** Con questo simbolo indicheremo i puntatori alla memoria del tipo HL, IX+DD e IY+DD, ove "DD" sia un valore compreso tra -128 e 127.
4. **"VAL8" (costante a 8 bit).** Con questo simbolo indichiamo una costante a 8 bit.
5. **"VAL16" (costante a 16 bit).** Con questo simbolo indichiamo una costante a 16 bit.
6. **"•" (flag non modificato).** Il simbolo "•" indica, che il flag sotto cui è posto, non viene modificato dalla istruzione.
7. **"≈" (flag modificato).** Il simbolo "≈" indica, che il flag sotto cui è posto, viene modificato dall'istruzione.
8. **"X" (flag modificato in modo casuale)**. Il simbolo "X" indica, che il flag sotto cui è posto, viene modificato dall'istruzione, ma in modo casuale.
9. **"nb" (numero di bit).** Con "nb" indicheremo un bit all'interno di un byte. Il bit meno significativo viene indicato con 0.
10. **(condizione).** Il Simbolo CC indica le condizioni NZ, Z, NC, C, PO, PE, P, M.

3.15 ADC A,VAL8

SINTASSI: ADC A,VAL8

Dove VAL8 è una costante a 8 bit.

SCOPO: Sommare al contenuto del registro A, la costante VAL8 e il carry, e memorizzare la somma in A stesso.

A ← A + VAL8 + Carry

FLAG:

S	Z	H	P/V	N	C
≈	≈	≈	≈	0	≈

Il flag P/V lavora come flag di overflow.

ESEMPI:

```
ADC A,12  ; A ← A+12+Carry
ADC A,0   ; A ← A+Carry
ADC A,#FF ; A ← A+255+Carry
```

3.16 ADC A,REG8

SINTASSI: ADC A,REG8

Dove REG8 rappresenta i registri A, B, C, D, E, H, L.

SCOPO: Sommare al contenuto del registro A, il registro REG8 e il carry, e memorizzare la somma in A stesso.

A ← A + REG8 + Carry

FLAG:

S	Z	H	P/V	N	C
≈	≈	≈	≈	0	≈

Il flag P/V lavora come flag di overflow.

ESEMPI:

```
ADC A,B ; A ← A+B+Carry
ADC A,C ; A ← A+C+Carry
ADC A,L ; A ← A+L+Carry
```

3.17 ADC A,(PMEM)

SINTASSI: ADC A,(PMEM)

Dove PMEM è un puntatore alla memoria del tipo HL, IX+DD, IY+DD.

SCOPO: Sommare al contenuto del registro A, il contenuto della locazione di memoria puntata da PMEM e il carry. Memorizzare la somma in A stesso.

A ← A + (PMEM) + Carry

FLAG:

S	Z	H	P/V	N	C
≈	≈	≈	≈	0	≈

Il flag P/V lavora come flag di overflow.

ESEMPI:

```
ADC A,(HL)     ;A ←A+(HL)+Carry
ADC A,(IX+9)   ;A ←A+(IX+9)+Carry
ADC A,(IY-16)  ;A ←A+(IY-16)+Carry
ADC A,(IX)     ;A ←A+(IX+00)+Carry
```

3.18 ADC HL,REG16

SINTASSI: ADC HL,REG16

Dove REG16 rappresenta i registri BC, DE, HL, SP.

SCOPO: Sommare al contenuto del registro HL, il registro REG16 e il carry, e memorizzare la somma in HL.

HL ← HL + REG16 + Carry

FLAG:

S	Z	H	P/V	N	C
≈	≈	≈	≈	0	≈

Il flag P/V lavora come flag di overflow. H è posto a 1 se si verifica un riporto dal bit 11.

ESEMPI:

```
ADC HL,BC ; HL ← HL+BC+Carry
ADC HL,DE ; HL ← HL+DE+Carry
ADC HL,HL ; HL ← HL+HL+Carry
ADC HL,SP ; HL ← HL+SP+Carry
```

3.19 ADD A,VAL8

SINTASSI: ADD A,VAL8

Dove VAL8 è una costante a 8 bit.

SCOPO: Sommare al contenuto del registro A, la costante VAL8 e memorizzare la somma in A stesso.

A ← A + VAL8

FLAG:

S	Z	H	P/V	N	C
≈	≈	≈	≈	0	≈

Il flag P/V lavora come flag di overflow.

ESEMPI:

```
ADD A,125 ; A ← A+125
ADD A,7   ; A ← A+7
ADD A,#10 ; A ← A+16
```

3.20 ADD A,REG8

SINTASSI: ADD A,REG8

Dove REG8 rappresenta i registri A, B, C, D, E, H, L.

SCOPO: Sommare al contenuto del registro A, il registro REG8 e memorizzare la somma in A stesso.

A ← A + REG8

FLAG:

S	Z	H	P/V	N	C
≈	≈	≈	≈	0	≈

Il flag P/V lavora come flag di overflow.

ESEMPI:

```
ADD A,E ; A ← A+E
ADD A,H ; A ← A+H
ADD A,D ; A ← A+D
```

3.21 ADD A,(PMEM)

SINTASSI: ADD A,(PMEM)

Dove PMEM è un puntatore alla memoria del tipo HL, IX+DD, IY+DD.

SCOPO: Sommare al contenuto del registro A, il contenuto della locazione di memoria puntata da PMEM. Memorizzare la somma in A stesso.

A ← A + (PMEM)

FLAG:

S	Z	H	P/V	N	C
≈	≈	≈	≈	0	≈

Il flag P/V lavora come flag di overflow.

ESEMPI:

```
ADD A,(HL)      ;A ← A+(HL)
ADD A,(IX+#F1)  ;A ← A+(IX+#F1)
ADD A,(IY-99)   ;A ← A+(IY-99)
ADD A,(IX)      ;A ← A+(IX+00)
```

3.22 ADD HL,REG16

SINTASSI: ADD HL,REG16

Dove REG16 rappresenta i registri BC, DE, HL, SP.

SCOPO: Sommare al contenuto del registro HL, il registro REG16 e memorizzare la somma in HL.

HL ← HL + REG16

FLAG:

S	Z	H	P/V	N	C
•	•	≈	•	0	≈

H è posto a 1 se si verifica un riporto dal bit 11.

ESEMPI:

```
ADD HL,BC ; HL ← HL+BC
ADD HL,DE ; HL ← HL+DE
ADD HL,HL ; HL ← HL+HL
ADD HL,SP ; HL ← HL+SP
```

3.23 ADD IX,REG16

SINTASSI: ADD IX,REG16

Dove REG16 rappresenta i registri BC, DE, IX, SP.

SCOPO: Sommare al contenuto del registro IX, il registro REG16 e memorizzare la somma in IX.

IX ← IX + REG16

FLAG:

S	Z	H	P/V	N	C
•	•	≈	•	0	≈

H è posto a 1 se si verifica un riporto dal bit 11.

ESEMPI:

```
ADD IX,BC ; IX ← IX+BC
ADD IX,DE ; IX ← IX+DE
ADD IX,IX ; IX ← IX+IX
ADD IX,SP ; IX ← IX+SP
```

3.24 ADD IY,REG16

SINTASSI: ADD IY,REG16

Dove REG16 rappresenta i registri BC, DE, IY, SP.

SCOPO: Sommare al contenuto del registro IY, il registro REG16 e memorizzare la somma in IY.

IY ← IY + REG16

FLAG:

S	Z	H	P/V	N	C
•	•	≈	•	0	≈

H è posto a 1 se si verifica un riporto dal bit 11.

ESEMPI:

```
ADD IY,BC ; IY ← IY+BC
ADD IY,DE ; IY ← IY+DE
ADD IY,IY ; IY ← IY+IY
ADD IY,SP ; IY ← IY+SP
```

3.25 AND VAL8

SINTASSI: AND VAL8

Dove VAL8 è una costante a 8 bit.

SCOPO: Fare l'operazione logica di AND fra i bit del registro A e i bit della costante VAL8. Memorizzare il risultato in A stesso.

A ← A ∩ VAL8

FLAG:

S	Z	H	P/V	N	C
≈	≈	1	≈	0	0

Il flag P/V lavora come flag di parità

ESEMPI:

```
AND #B2   ; A ← A ∩ #B2
AND 1     ; A ← A ∩ 1
AND 100   ; A ← A ∩ 100
```

3.26 AND REG8

SINTASSI: AND REG8

Dove REG8 rappresenta i registri A, B, C, D, E, H, L.

SCOPO: Fare l'operazione logica di AND fra i bit del registro A e i bit del registro REG8. Memorizzare il risultato in A stesso.

$A \leftarrow A \cap REG8$

FLAG:

S	Z	H	P/V	N	C
≈	≈	1	≈	0	0

Il flag P/V lavora come flag di parità

ESEMPI:

```
AND A  ; A ← A ∩ A
AND L  ; A ← A ∩ L
AND C  ; A ← A ∩ C
```

3.27 AND (PMEM)

SINTASSI: AND (PMEM)

Dove PMEM è un puntatore alla memoria del tipo HL, IX+DD, IY+DD.

SCOPO: Fare l'operazione logica di AND fra i bit del registro A e i bit del byte di memoria puntato da PMEM. Memorizzare il risultato in A stesso.

A ← A ∩ (PMEM)

FLAG:

S	Z	H	P/V	N	C
≈	≈	1	≈	0	0

Il flag P/V lavora come flag di parità

ESEMPI:

```
AND (HL)     ; A ← A ∩ (HL)
AND (IX+15)  ; A ← A ∩ (IX+15)
AND (IY-10)  ; A ← A ∩ (IY-10)
```

3.28 BIT nb,REG8

SINTASSI: BIT nb,REG8

Dove REG8 rappresenta i registri A, B, C, D, E, H, L, mentre nb il numero del bit (quindi è una costante compresa tra 0 e 7).

SCOPO: Testare il contenuto del bit nb del registro REG8. Se il bit nb è 0 il flag dello zero viene posto a 1.

$$Z \leftarrow \overline{REG8\_nb}$$

FLAG:

S	Z	H	P/V	N	C
X	≈	1	X	0	•

ESEMPI:

```
BIT 0,A    ; testa LSB di A
BIT 3,B    ; testa bit 3 di B
BIT 7,H    ; testa MSB di H
```

3.29 BIT nb,(PMEM)

SINTASSI: BIT nb,(PMEM)

Dove PMEM è un puntatore alla memoria del tipo HL, IX+DD, IY+DD, mentre nb il numero del bit.

SCOPO: Testare il contenuto del bit nb del byte di memoria puntato da PMEM. Se il bit nb è 0 il flag dello zero viene posto a 1.

$$Z \leftarrow \overline{(PMEM)\_nb}$$

FLAG:

S	Z	H	P/V	N	C
X	≈	1	X	0	•

ESEMPI:

```
BIT 0,(HL)    ;testa LSB di (HL)
BIT 5,(IX)    ;testa bit 5 di (IX)
BIT 7,(IY+9)  ;testa MSB di (IY+9)
```

3.30 CALL VAL16

SINTASSI: CALL VAL16

Dove VAL16 è una costante a 16 bit (in generale si tratta di una label).

SCOPO: Chiamare il sottoprogramma che si trova all'indirizzo VAL16.

(SP-1) ← PC_alto
(SP-2) ← PC_basso
SP ← SP - 2
PC ← VAL16

FLAG:

S	Z	H	P/V	N	C
•	•	•	•	•	•

NOTE: In Z80 Simulation la costante VAL16 non è un indirizzo di memoria, bensì un numero di linea. Si consiglia perciò, di utilizzare sempre le label, almeno in quei casi che sia possibile.

ESEMPI:

```
                ; QUESTO PROGRAMMA CARICA IN OGNI
                ; BYTE DEL BUFFER CHE SI TROVA
                ; AGLI INDIRIZZI #F000 ÷ #F050 IL
                ; VALORE #E5

                BUFF    EQU #F000
                VALUE   EQU #E5

                ORG #B000

                LD HL,BUFF
                LD C,#51
    LOOP        CALL CARICA
                JR NZ,LOOP
                HALT

                ; ROUTINE DI CARICAMENTO DI UN
                ; BYTE

                ORG #BF00

    CARICA      LD (HL),VALUE
                INC HL
                DEC C
                RET
```

3.31 CALL CC,VAL16

SINTASSI: CALL CC,VAL16

Dove VAL16 è una costante a 16 bit (in generale si tratta di una label), mentre CC è una delle condizioni:
NZ, Z, NC, C, PO, PE, P, M.

SCOPO: Se la condizione CC è vera, chiamare il sottoprogramma che si trova all'indirizzo VAL16.

Se CC è vera:
(SP-1) ← PC_alto
(SP-2) ← PC_basso
SP ← SP - 2
PC ← VAL16

FLAG:

S	Z	H	P/V	N	C
•	•	•	•	•	•

NOTE: Vedi la nota dell'istruzione precedente.

ESEMPI:

```
                ;QUESTO PROGRAMMA  COPIA  I  BYTE
                ;CON PARITA' PARI NEL BUFFER ALLO
                ;INDIRIZZO #2000,  MENTRE I  BYTE
                ;CON PARITA' DISPARI ALL'INDIRIZ-
                ;ZO #3000. I DATI SONO LETTI DAL-
                ;LA PORTA DI INPUT #35

     PORTA      EQU #35
     BPARI      EQU #2000
     BDISP      EQU #3000

                ORG #1000
                LD C,PORTA
                LD HL,BPARI
                LD DE,BDISP
     LOOP       IN A,(C)
                CALL PE,MPARI
                CALL PO,MDISP
                JR LOOP

                ;MEMORIZZA DATI CON PARITA' PARI
     MPARI      LD (HL),A
                INC HL
                RET

                ;MEMORIZZA DATI CON PARITA'
                ;DISPARI
     MDISP      LD (DE),A
                INC DE
                RET
```

3.32 CCF

SINTASSI: CCF

SCOPO: Complementare lo stato del carry.

$$Carry \leftarrow \overline{Carry}$$

FLAG:

S	Z	H	P/V	N	C
•	•	≈	•	0	≈

Il flag H assume lo stato del carry precedente l'operazione.

ESEMPI:
```
SCF ; Carry ← 1
CCF ; Carry ← 0
```

3.33 CP VAL8

SINTASSI: CP VAL8

Dove VAL8 è una costante a 8 bit.

SCOPO: Confrontare il contenuto del registro A con la costante VAL8.
In realtà viene eseguita l'operazione di sottrazione A - VAL8, il cui risultato viene perso, ma che comunque modifica i flag.

A - VAL8

FLAG:

S	Z	H	P/V	N	C
≈	≈	≈	≈	1	≈

Il flag P/V lavora come flag di overflow.

ESEMPI:

```
CP 10       ; A - 10
CP #FF      ; A - #FF
CP 100      ; A - 100
```

3.34 CP REG8

SINTASSI: CP REG8

Dove REG8 rappresenta i registri A, B, C, D, E, H, L.

SCOPO: Confrontare il contenuto del registro A con il contenuto del registro REG8.
In realtà viene eseguita una opezione di sottraz. A - REG8, il cui risultato viene perso, ma che comunque modifica i flag.

A - REG8

FLAG:

S	Z	H	P/V	N	C
≈	≈	≈	≈	1	≈

Il flag P/V lavora come flag di overflow.

ESEMPI:

```
CP B   ; A - B
CP L   ; A - L
```

3.35 CP (PMEM)

SINTASSI: CP (PMEM)

Dove PMEM è un puntatore alla memoria del tipo HL, IX+DD, IY+DD.

SCOPO: Confrontare il contenuto del registro A con il contenuto della locazione puntata da PMEM.
In realtà viene eseguita l'operazione di sottrazione A - (PMEM) il cui risultato viene perso, ma che comunque modifica i flag.

A - (PMEM)

FLAG:

S	Z	H	P/V	N	C
≈	≈	≈	≈	1	≈

Il flag P/V lavora come flag di overflow.

ESEMPI:
```
CP (HL)      ; A - (HL)
CP (IX+10)   ; A - (IX+10)
```

3.36 CPD

SINTASSI: CPD

SCOPO: Confrontare il contenuto del registro A con il contenuto della locazione puntata da HL. Decrementare poi HL e BC.

A - (HL)
HL ← HL - 1
BC ← BC - 1

FLAG:

S	Z	H	P/V	N	C
≈	≈	≈	≈	1	•

Il flag P/V è 0 se alla fine della istruzione, BC è 0. Il flag Z è 1 se A=(HL).

ESEMPI:

```
LOOP    CPD
        JR NZ,LOOP
```

3.37 CPDR

SINTASSI: CPDR

SCOPO: Confrontare il contenuto del registro A con il contenuto del byte puntato da HL. Decrementare poi HL e BC, e ripetere l'istruzione finché BC=0, oppure A=(HL).

Ripeti

A - (HL)
HL ← HL - 1
BC ← BC - 1

finché BC=0 o A=(HL)

FLAG:

S	Z	H	P/V	N	C
≈	≈	≈	≈	1	•

Il flag P/V è 0 se alla fine della istruzione, BC è 0. Il flag Z è 1 se A=(HL).

3.38 CPI

SINTASSI: CPI

SCOPO: Confrontare il contenuto del registro A con il contenuto del byte puntato da HL. Incrementare poi HL e decrementare BC.

A - (HL)
HL ← HL + 1
BC ← BC - 1

FLAG:

S	Z	H	P/V	N	C
≈	≈	≈	≈	1	•

Il flag P/V è 0 se alla fine della istruzione, BC è 0. Il flag Z è 1 se A=(HL).

ESEMPI:

```
LOOP    CPI
        JR NZ,LOOP
```

3.39 CPIR

SINTASSI: CPIR

SCOPO: Confrontare il contenuto del registro A con il contenuto del byte puntato da HL. Incrementare poi HL e decrementare BC, e ripetere l'istruzione finché BC=0, oppure A=(HL).

ripeti

A - (HL)
HL ← HL + 1
BC ← BC - 1

finché BC=0 o A=(HL)

FLAG:

S	Z	H	P/V	N	C
≈	≈	≈	≈	1	•

Il flag P/V è 0 se alla fine della istruzione, BC è 0. Il flag Z è 1 se A=(HL).

3.40 CPL

SINTASSI: CPL

SCOPO: Complementare i bit dell'accumulatore.

$A \leftarrow \overline{A}$

FLAG:

S	Z	H	P/V	N	C
•	•	1	•	1	•

ESEMPI:

```
LD A,#FF ; A ← 255
CPL      ; A ← 0
LD A,#C1 ; A ← #C1
CPL      ; A ← #3E
```

3.41 DAA

SINTASSI: DAA

SCOPO: Eseguire l'aggiustamento decimale dell'accumulatore.

A ← BCD di A

FLAG:

S	Z	H	P/V	N	C
≈	≈	≈	≈	1	≈

Il flag P/V lavora come flag di parità

NOTE: La conversione binario-BCD dipende dal tipo di operazione aritmetica fatta prima della DAA, dallo stato dei flag H e C: data perciò la complessità dell'istruzione, si consiglia la consultazione del manuale del µP Z80.

ESEMPI:

```
LD A,15 ; A ← 15
ADD A,0 ; A ← 15
DAA     ; A ← #15
```

3.42 DEC REG8

SINTASSI: DEC REG8

Dove REG8 rappresenta i registri A, B, C, D, E, H, L.

SCOPO: Decrementare il registro REG8.

REG8 ← REG8 - 1

FLAG:

S	Z	H	P/V	N	C
≈	≈	≈	≈	1	•

Il flag P/V lavora come flag di overflow.

ESEMPI:

```
DEC A  ; A ← A - 1
DEC L  ; L ← L - 1
DEC E  ; E ← E - 1
DEC B  ; B ← B - 1
```

3.43 DEC (PMEM)

SINTASSI: DEC (PMEM)

Dove PMEM è un puntatore alla memoria del tipo HL, IX+DD, IY+DD.

SCOPO: Decrementare il contenuto della locazione di memoria puntata da PMEM.

(PMEM) ← (PMEM) - 1

FLAG:

S	Z	H	P/V	N	C
≈	≈	≈	≈	1	•

Il flag P/V lavora come flag di overflow.

ESEMPI:

```
DEC (HL)     ;(HL) ← (HL)-1
DEC (IX-1)   ;(IX-1) ← (IX-1)-1
DEC (IY+20)  ;(IY+20) ← (IY+20)-1
DEC (IX)     ;(IX+0) ← (IX+0)-1
```

3.44 DEC REG16

SINTASSI: DEC REG16

Dove REG16 rappresenta i registri BC, DE, HL, IX, IY, SP.

SCOPO: Decrementare il registro REG16.

REG16 ← REG16 - 1

FLAG:

S	Z	H	P/V	N	C
•	•	•	•	•	•

ESEMPI:

```
DEC HL ; HL ← HL-1
DEC DE ; DE ← DE-1
DEC IX ; IX ← IX-1
DEC SP ; SP ← SP-1
```

3.45 DI

SINTASSI: DI

SCOPO: Disabilitare le interruzioni mascherabili caricando 0 nel flip-flop IFF.

IFF ← 0

FLAG:

S	Z	H	P/V	N	C
•	•	•	•	•	•

NOTE: Poiché in Z80 Simulation le interruzioni non vengono simulate, l'istruzione DI non ha alcun effetto.

3.46 DJNZ VAL8

SINTASSI: DJNZ VAL8

Dove VAL8 è una costante a 8 bit in complemento a 2.

SCOPO: Drecrementare il registro B. Se dopo il decremento il risultato non è 0, saltare in modo relativo.

```
B ← B - 1
Se B<>0
     PC ← PC + VAL8
```

FLAG:

S	Z	H	P/V	N	C
•	•	•	•	•	•

ESEMPI:

```
          LD B,100
LOOP      LD (DE),A
          INC DE
          DJNZ LOOP
```

3.47 EI

SINTASSI: EI

SCOPO: Abilitare le interruzioni mascherabili caricando 1 nel flip-flop IFF.

IFF ← 1

FLAG:

S	Z	H	P/V	N	C
•	•	•	•	•	•

NOTE: Poiché in Z80 Simulation le interruzioni non vengono simulate, l'istruzione EI non ha alcun effetto.

3.48 EX AF,AF'

SINTASSI: EX AF,AF'

SCOPO: Scambiare il registro AF con il registro AF'.

AF ←→ AF'

FLAG:

S	Z	H	P/V	N	C
•	•	•	•	•	•

3.49 EX DE,HL

SINTASSI: EX DE,HL

SCOPO: Scambiare il contenuto del registro DE con il contenuto del registro HL

DE ←→ HL

FLAG:

S	Z	H	P/V	N	C
•	•	•	•	•	•

3.50 EX (SP),REG16

SINTASSI: EX (SP),REG16

Dove REG16 rappresenta i registri HL, IX, IY.

SCOPO: Scambiare il registro REG16 con la cima dello stack.

(SP) ←→ REG16_basso
(SP+1) ←→ REG16_alto

FLAG:

S	Z	H	P/V	N	C
•	•	•	•	•	•

3.51 EXX

SINTASSI: EXX

SCOPO: Scambiare i registri BC, DE, HL, con i rispettivi omologhi BC', DE', HL'.

BC ←→ BC'
DE ←→ DE'
HL ←→ HL'

FLAG:

S	Z	H	P/V	N	C
•	•	•	•	•	•

3.52 HALT

SINTASSI: HALT

SCOPO: Sospendere il funzionamento della CPU finché non si opera un RESET o si riceve un'interruzione. I cicli di rinfresco della RAM vengono comunque eseguiti.

FLAG:

S	Z	H	P/V	N	C
•	•	•	•	•	•

NOTE: In Z80 Simulation l'istruzione HALT arresta il programma e restituisce il controllo alla barra menu.

3.53 IM VAL8

SINTASSI: IM VAL8

Dove VAL8 vale 0, 1 o 2.

SCOPO: Predisporre il modo d'interruzione 0, 1, o 2 a seconda del valore di VAL8.

FLAG:

S	Z	H	P/V	N	C
•	•	•	•	•	•

ESEMPI:

```
IM 0 ;Imposta modo 0 d'interr.
IM 1 ;Imposta modo 1 d'interr.
IM 2 ;Imposta modo 2 d'interr.
```

NOTE: In Z80 Simulation, questa istruzione non ha particolari funzioni. Tuttavia, nelle informazioni di sistema, si può vedere quale è il modo d'interruzione attivo.

3.54 IN REG8,(C)

SINTASSI: IN REG8,(C)

Dove REG8 rappresenta i registri A, B, C, D, E, H, L, mentre il simbolo (C) indica la porta di input puntata dal registro C.

SCOPO: Caricare nel registro REG8, il dato letto dalla porta di input puntata dal registro C.

REG8 ← (C)

FLAG:

S	Z	H	P/V	N	C
≈	≈	≈	≈	0	•

Il flag P/V lavora come flag di parità

ESEMPI:

```
LD C,#32 ; C ← #32
IN H,(C) ; carica in H il dato
         ; letto dalla porta #32
```

3.55 IN A,(VAL8)

SINTASSI: IN A,(VAL8)

Dove con (VAL8) si indica la porta di input numero VAL8.

SCOPO: Caricare nel registro A, il dato letto dalla porta di input numero VAL8.

A ← (VAL8)

FLAG:

S	Z	H	P/V	N	C
•	•	•	•	•	•

ESEMPI:

```
IN A,(99) ; carica in A il dato
          ; letto dalla porta 99
```

3.56 INC REG8

SINTASSI: INC REG8

Dove REG8 rappresenta i registri A, B, C, D, E, H, L.

SCOPO: Incrementare il registro REG8.

REG8 ← REG8 + 1

FLAG:

S	Z	H	P/V	N	C
≈	≈	≈	≈	0	•

Il flag P/V lavora come flag di overflow.

ESEMPI:

```
INC A  ; A ← A + 1
INC L  ; L ← L + 1
INC E  ; E ← E + 1
INC B  ; B ← B + 1
```

3.57 INC (PMEM)

SINTASSI: INC (PMEM)

Dove PMEM è un puntatore alla memoria del tipo HL, IX+DD, IY+DD.

SCOPO: Incrementare il contenuto della locazione di memoria puntata da PMEM.

(PMEM) ← (PMEM) + 1

FLAG:

S	Z	H	P/V	N	C
≈	≈	≈	≈	0	•

Il flag P/V lavora come flag di overflow.

ESEMPI:

```
INC (HL)     ;(HL) ← (HL)+1
INC (IX-1)   ;(IX-1) ← (IX-1)+1
INC (IY+20)  ;(IY+20) ← (IY+20)+1
INC (IX)     ;(IX+0) ← (IX+0)+1
```

3.58 INC REG16

SINTASSI: INC REG16

Dove REG16 rappresenta i registri BC, DE, HL, IX, IY, SP.

SCOPO: Incrementare il registro REG16.

REG16 ← REG16 + 1

FLAG:

S	Z	H	P/V	N	C
•	•	•	•	•	•

ESEMPI:

```
INC HL ; HL ← HL+1
INC DE ; DE ← DE+1
INC IX ; IX ← IX+1
INC SP ; SP ← SP+1
```

3.59 IND

SINTASSI: IND

SCOPO: Caricare nella locazione puntata da HL, il dato letto dalla porta di valore pari al contenuto del registro C. Decrementare poi HL e B.

(HL) ← (C)
HL ← HL - 1
B ← B - 1

FLAG:

S	Z	H	P/V	N	C
X	≈	X	X	1	X

Il flag Z è posto a 1 se il registro B è 0 dopo l'esecuzione.

ESEMPI:

```
LOOP    IND
        JR NZ,LOOP
```

3.60 INDR

SINTASSI: INDR

SCOPO: Caricare nella locazione puntata da HL, il dato letto dalla porta di valore pari al contenuto del registro C. Decrementare poi HL e B, e ripetere l'operazione finché B non diventa 0.

ripeti

(HL) ← (C)
HL ← HL - 1
B ← B - 1

finché B=0

FLAG:

S	Z	H	P/V	N	C
X	≈	X	X	1	X

Il flag Z è posto a 1 se il registro B è 0 dopo l'esecuzione.

3.61 INI

SINTASSI: INI

SCOPO: Caricare nella locazione puntata da HL, il dato letto dalla porta di valore pari al contenuto del registro C. Incrementare poi HL e decrementare B.

(HL) ← (C)
HL ← HL + 1
B ← B - 1

FLAG:

S	Z	H	P/V	N	C
X	≈	X	X	1	X

Il flag Z è posto a 1 se il registro B è 0 dopo l'esecuzione.

ESEMPI:

```
LOOP    INI
        JR NZ,LOOP
```

3.62 INIR

SINTASSI: INIR

SCOPO: Caricare nella locazione puntata da HL, il dato letto dalla porta di valore pari al contenuto del registro C. Incrementare poi HL e decrementare B, e ripetere l'operazione finché B non diventa 0.

ripeti

(HL) ← (C)
HL ← HL + 1
B ← B - 1

finché B=0

FLAG:

S	Z	H	P/V	N	C
X	≈	X	X	1	X

Il flag Z è posto a 1 se il registro B è 0 dopo l'esecuzione.

3.63 JP VAL16

SINTASSI: JP VAL16

Dove VAL16 è una costante a 16 bit (in generale si tratta di una label).

SCOPO: Saltare all'istruzione che si trova all'indirizzo VAL16.

PC ← VAL16

FLAG:

S	Z	H	P/V	N	C
•	•	•	•	•	•

NOTE: In Z80 Simulation la costante VAL16 non è un indirizzo di memoria, bensì un numero di linea. Si consiglia perciò, di utilizzare sempre le label, almeno in quei casi in cui sia possibile.

3.64 JP CC,VAL16

SINTASSI: JP CC,VAL16

Dove VAL16 è una costante a 16 bit (in generale si tratta di una label), mentre CC è una delle condizioni:
NZ, Z, NC, C, PO, PE, P, M.

SCOPO: Se la condizione CC è vera, saltare all'istruzione che si trova all'indirizzo VAL16.

Se CC è vera:

PC ← VAL16

FLAG:

S	Z	H	P/V	N	C
•	•	•	•	•	•

ESEMPI:
```
JP NC,#F2
JP PO,FINE
```

NOTE: Vedi la nota dell'istruzione precedente.

3.65 JP (REG16)

SINTASSI: JP (REG16)

Dove REG16 rappresenta i registri HL, IX, IY.

SCOPO: Saltare all'istruzione che si trova all'indirizzo contenuto nel registro REG16.

PC ← REG16

FLAG:

S	Z	H	P/V	N	C
•	•	•	•	•	•

ESEMPI:

```
JP (HL) ; PC ← HL
JP (IX) ; PC ← IX
JP (IY) ; PC ← IY
```

NOTE: In Z80 Simulation il registro REG16 deve contenere un numero di linea.

3.66 JR VAL8

SINTASSI: JR VAL8

Dove VAL8 è una costante a 8 bit in complemento a 2.

SCOPO: Saltare in modo relativo.

PC ← PC + VAL8

FLAG:

S	Z	H	P/V	N	C
•	•	•	•	•	•

NOTE: In Z80 Simulation VAL8 non rappresenta lo spostamento in byte, bensì lo spostamento in righe.
Si noti che se si specifica una label l'esecutore non salta avanti o indietro di un numero di righe pari al valore della label, ma salta alla linea in cui si è definita la label.

3.67 JR CC,VAL8

SINTASSI: JR CC,VAL8

Dove VAL8 è una costante a 8 bit in complemento a 2, mentre CC è una delle condizioni:
NZ, Z, NC, C.

SCOPO: Se la condizione CC è vera, saltare in modo relativo.

Se CC è vera:
PC ← PC + VAL8

FLAG:

S	Z	H	P/V	N	C
•	•	•	•	•	•

NOTE: Vedi la nota dell'istruzione precedente.

3.68 LD REG16,(VAL16)

SINTASSI: LD REG16,(VAL16)

Dove REG16 rappresenta i registri BC, DE, HL, IX, IY, SP, mentre VAL16 è un indirizzo di memoria.

SCOPO: Caricare nel registro REG16 il contenuto delle due locazioni di memoria di indirizzo VAL16 e VAL16+1.

REG16_basso ← (VAL16)
REG16_alto ← (VAL16+1)

FLAG:

S	Z	H	P/V	N	C
•	•	•	•	•	•

ESEMPI:

```
LD BC,(1240)    ; BC ← (1240)
LD SP,(#1234)   ; SP ← (#1234)
LD IX,(0)       ; IX ← (0)
```

3.69 LD REG16,VAL16

SINTASSI: LD REG16,VAL16

Dove REG16 rappresenta i registri BC, DE, HL, IX, IY, SP, mentre VAL16 rappresenta una costante a 16 bit.

SCOPO: Caricare nel registro REG16 la costante VAL16.

REG16 ← VAL16

FLAG:

S	Z	H	P/V	N	C
•	•	•	•	•	•

ESEMPI:

```
LD DE,1940    ; DE ← 1940
LD IY,#FFFF   ; IY ← #FFFF
LD SP,500     ; SP ← 500
```

3.70 LD REG8,VAL8

SINTASSI: LD REG8,VAL8

Dove REG8 rappresenta i registri A, B, C, D, E, H, L, mentre VAL8 rappresenta una costante a 8 bit.

SCOPO: Caricare nel registro REG8 la costante VAL8.

REG8 ← VAL8

FLAG:

S	Z	H	P/V	N	C
•	•	•	•	•	•

ESEMPI:

```
LD E,40    ; E ← 40
LD C,#FF   ; C ← #FF
LD A,13    ; A ← 13
```

3.71 LD REG8,REG8*

SINTASSI: LD REG8,REG8*

Dove REG8 e REG8* rappresentano i registri A, B, C, D, E, H, L.

SCOPO: Caricare nel registro REG8 il contenuto del registro REG8*.

REG8 ← REG8*

FLAG:

S	Z	H	P/V	N	C
•	•	•	•	•	•

ESEMPI:

```
LD B,D   ; B ← D
LD C,E   ; C ← E
LD A,L   ; A ← L
```

3.72 LD REG8,(PMEM)

SINTASSI: LD REG8,(PMEM)

Dove PMEM è un puntatore alla memoria del tipo HL, IX+DD, IY+DD, mentre REG8 è uno dei registri A, B, C, D, E, H, L.

SCOPO: Caricare nel registro REG8 il contenuto della locazione di memoria puntata da PMEM.

REG8 ← (PMEM)

FLAG:

S	Z	H	P/V	N	C
•	•	•	•	•	•

ESEMPI:

```
LD E,(HL)      ; E ← (HL)
LD L,(IX+1)    ; L ← (IX+1)
LD A,(IY-1)    ; A ← (IY-1)
```

3.73 LD (PMEM),A

SINTASSI: LD (PMEM),A

Dove PMEM rappresenta i registri BC, DE.

SCOPO: Caricare nella locazione di memoria puntata da PMEM il contenuto del registro A.

(PMEM) ← A

FLAG:

S	Z	H	P/V	N	C
•	•	•	•	•	•

ESEMPI:

```
LD (BC),A   ; (BC) ← A
LD (DE),A   ; (DE) ← A
```

3.74 LD (VAL16),A

SINTASSI: LD (VAL16),A

Dove VAL16 rappresenta un indirizzo a 16 bit.

SCOPO: Caricare nella locazione di memoria di indirizzo VAL16 il contenuto del registro A.

(VAL16) ← A

FLAG:

S	Z	H	P/V	N	C
•	•	•	•	•	•

ESEMPI:

```
LD (#15FF),A    ; (#15FF) ← A
LD (10),A       ; (10) ← A
```

3.75 LD (PMEM),VAL8

SINTASSI: LD (PMEM),VAL8

Dove PMEM è un puntatore alla memoria del tipo HL, IX+DD, IY+DD.

SCOPO: Caricare nella locazione puntata da PMEM la costante VAL8.

(PMEM) ← VAL8

FLAG:

S	Z	H	P/V	N	C
•	•	•	•	•	•

ESEMPI:

```
LD (HL),102      ; (HL) ← 102
LD (IX+1),#0F    ; (IX+1) ← #0F
LD (IY-1),#1F    ; (IY-1) ← #1F
```

3.76 LD (PMEM),REG8

SINTASSI: LD (PMEM),REG8

Dove PMEM è un puntatore alla memoria del tipo HL, IX+DD, IY+DD, mentre REG8 è uno dei registri A, B, C, D, E, H, L.

SCOPO: Caricare nella locazione di memoria puntata da PMEM il contenuto del registro REG8.

(PMEM) ← REG8

FLAG:

S	Z	H	P/V	N	C
•	•	•	•	•	•

ESEMPI:

```
LD (HL),C      ; (HL) ← C
LD (IX+1),D    ; (IX+1) ← D
LD (IY-1),E    ; (IY-1) ← E
```

3.77 LD (VAL16),REG16

SINTASSI: LD (VAL16),REG16

Dove REG16 rappresenta i registri BC, DE, HL, IX, IY, SP, mentre VAL16 è un indirizzo di memoria.

SCOPO: Caricare nelle locazioni di memoria di indirizzo VAL16 e VAL16+1, il contenuto del registro REG16.

(VAL16) ← REG16_basso
(VAL16+1) ← REG16_alto

FLAG:

S	Z	H	P/V	N	C
•	•	•	•	•	•

ESEMPI:

```
LD (1240),DE   ; (1240) ← DE
LD (#1234),SP  ; (#1234) ← SP
LD (0),IY      ; (0) ← IY
```

3.78 LD A,(PMEM)

SINTASSI: LD A,(PMEM)

Dove PMEM rappresenta i registri BC, DE.

SCOPO: Caricare nell'accumulatore il contenuto della locazione di memoria puntata da PMEM.

A ← (PMEM)

FLAG:

S	Z	H	P/V	N	C
•	•	•	•	•	•

ESEMPI:

```
LD A,(BC)    ; A ← (BC)
LD A,(DE)    ; A ← (DE)
```

3.79 LD A,(VAL16)

SINTASSI: LD A,(VAL16)

Dove VAL16 rappresenta un indirizzo a 16 bit.

SCOPO: Caricare nell'accumulatore il contenuto della locazione di memoria di indirizzo VAL16.

A ← (VAL16)

FLAG:

S	Z	H	P/V	N	C
•	•	•	•	•	•

ESEMPI:

```
LD A,(#15FF)    ; A ← (#15FF)
LD A,(10)       ; A ← (10)
```

3.80 LD A,I

SINTASSI: LD A,I

SCOPO: Caricare nell'accumulatore il contenuto del registro I.

A ← I

FLAG:

S	Z	H	P/V	N	C
≈	≈	0	≈	0	•

Il flag P/V indica lo stato del flip-flop IFF2.

NOTE: Poiché Z80 Simulation non simula le interruzioni, questa istruzione non ha alcun effetto.

3.81 LD A,R

SINTASSI: LD A,R

SCOPO: Caricare nell'accumulatore il contenuto del registro R.

A ← R

FLAG:

S	Z	H	P/V	N	C
≈	≈	0	≈	0	•

Il flag P/V indica lo stato del flip-flop IFF2.

NOTE: Poiché Z80 Simulation non rinfresca la memoria, questa istruzione non ha alcun effetto.

3.82 LD I,A

SINTASSI: LD I,A

SCOPO: Caricare nel registro I il contenuto dell'accumulatore.

I ← A

FLAG:

S	Z	H	P/V	N	C
•	•	•	•	•	•

NOTE: Poiché Z80 Simulation non simula le interruzioni, questa istruzione non ha alcun effetto.

3.83 LD R,A

SINTASSI: LD R,A

SCOPO: Caricare nel registro R il contenuto dell'accumulatore.

R ← A

FLAG:

S	Z	H	P/V	N	C
•	•	•	•	•	•

NOTE Poiché Z80 Simulation non rinfresca la memoria, questa istruzione non ha alcun effetto.

3.84 LD SP,REG16

SINTASSI: LD SP,REG16

Dove REG16 rappresenta i registri HL, IX, IY.

SCOPO: Caricare nel registro SP il contenuto del registro REG16.

SP ← REG16

FLAG:

S	Z	H	P/V	N	C
•	•	•	•	•	•

ESEMPI:

```
LD SP,HL  ; SP ← HL
LD SP,IX  ; SP ← IX
LD SP,IY  ; SP ← IY
```

3.85 LDD

SINTASSI: LDD

SCOPO: Caricare nella locaz. puntata da DE, il contenuto della locazione puntata da HL. Decrementare poi HL, DE e BC.

(DE) ← (HL)
HL ← HL - 1
DE ← DE - 1
BC ← BC - 1

FLAG:

S	Z	H	P/V	N	C
•	•	0	≈	0	•

Il flag P/V è posto a 0 se il registro BC è 0 dopo l'esecuzione.

ESEMPI:

```
LOOP    LDD
        JP PE,LOOP
```

3.86 LDDR

SINTASSI: LDDR

SCOPO: Caricare nella locaz. puntata da DE, il contenuto della cella puntata da HL. Decrementare poi HL, DE, BC e ripetere l'operazione finché BC non diventa 0.

ripeti

(DE) ← (HL)
HL ← HL - 1
DE ← DE - 1
BC ← BC - 1

finché BC=0

FLAG:

S	Z	H	P/V	N	C
•	•	0	≈	0	•

Il flag P/V è posto a 0 se il registro BC è 0 dopo l'esecuzione.

3.87 LDI

SINTASSI: LDI

SCOPO: Caricare nella locaz. puntata da DE, il contenuto della cella puntata da HL. Incrementare poi HL e DE, e decrementare BC.

(DE) ← (HL)
HL ← HL + 1
DE ← DE + 1
BC ← BC - 1

FLAG:

S	Z	H	P/V	N	C
•	•	0	≈	0	•

Il flag P/V è posto a 0 se il registro BC è 0 dopo l'esecuzione.

ESEMPI:

```
LOOP    LDI
        JP PE,LOOP
```

3.88 LDIR

SINTASSI: LDIR

SCOPO: Caricare nella locaz. puntata da DE, il contenuto della cella puntata da HL. Incrementare poi HL e DE, decrementare BC e ripetere l'operazione finché BC non diventa 0.

Ripeti

(DE) ← (HL)
HL ← HL + 1
DE ← DE + 1
BC ← BC - 1

finché BC=0

FLAG:

S	Z	H	P/V	N	C
•	•	0	≈	0	•

Il flag P/V è posto a 0 se il registro BC è 0 dopo l'esecuzione.

3.89 NEG

SINTASSI: NEG

SCOPO: Eseguire il complemento a 2 dello accumulatore.

A ← 0 - A

FLAG:

S	Z	H	P/V	N	C
≈	≈	≈	≈	1	≈

Il flag C viene posto a 1 se A era 0 prima dell'istruzione.
P/V viene posto a 1 se A era #80 prima dell'istruzione.

ESEMPI:

```
A,#FF ; A ← 255
NEG   ;A ← 1
```

3.90 NOP

SINTASSI: NOP

SCOPO: Eseguire un ciclo di ritardo. Durante l'esecuzione di questa istruzione lo Z80 non fa niente per un ciclo macchina.

FLAG:

S	Z	H	P/V	N	C
•	•	•	•	•	•

3.91 OR VAL8

SINTASSI: OR VAL8

Dove VAL8 è una costante a 8 bit.

SCOPO: Fare l'operazione logica di OR fra i bit del registro A e i bit della costante VAL8. Memorizzare il risultato in A stesso.

A ← A U VAL8

FLAG:

S	Z	H	P/V	N	C
≈	≈	0	≈	0	0

Il flag P/V lavora come flag di parità

ESEMPI:

```
OR #B2   ; A ← A U #B2
OR 1     ; A ← A U 1
OR 100   ; A ← A U 100
```

3.92 OR REG8

SINTASSI: OR REG8

Dove REG8 rappresenta i registri A, B, C, D, E, H, L.

SCOPO: Fare l'operazione logica di OR fra i bit del registro A e i bit del registro REG8. Memorizzare il risultato in A stesso.

A ← A U REG8

FLAG:

S	Z	H	P/V	N	C
≈	≈	0	≈	0	0

Il flag P/V lavora come flag di parità

ESEMPI:

```
OR A  ; A ← A U A
OR L  ; A ← A U L
OR C  ; A ← A U C
```

3.93 OR (PMEM)

SINTASSI: OR (PMEM)

Dove PMEM è un puntatore alla memoria del tipo HL, IX+DD, IY+DD.

SCOPO: Fare l'operazione logica di OR fra i bit del registro A e i bit del byte di memoria puntato da PMEM. Memorizzare il risultato in A stesso.

A ← A U (PMEM)

FLAG:

S	Z	H	P/V	N	C
≈	≈	0	≈	0	0

Il flag P/V lavora come flag di parità.

ESEMPI:

```
OR (HL)     ; A ← A U (HL)
OR (IX+15)  ; A ← A U (IX+15)
OR (IY-10)  ; A ← A U (IY-10)
```

3.94 OUT (C),REG8

SINTASSI: OUT (C),REG8

Dove REG8 rappresenta i registri A, B, C, D, E, H, L, mentre il simbolo (C) indica la porta di output puntata dal registro C.

SCOPO: Scrivere sulla porta di output puntata dal registro C, il contenuto del registro REG8.

(C) ← REG8

FLAG:

S	Z	H	P/V	N	C
•	•	•	•	•	•

ESEMPI:

```
LD C,#4F  ; C ← #4F
LD L,20   ; L ← 20
OUT (C),L ; Scrive 20 sulla
          ; porta #4F
```

3.95 OUT (VAL8),A

SINTASSI: OUT (VAL8),A

Dove con (VAL8) si indica la porta di output numero VAL8.

SCOPO: Scrivere sulla porta di output numero VAL8, il contenuto del registro A.

(VAL8) ← A

FLAG:

S	Z	H	P/V	N	C
•	•	•	•	•	•

ESEMPI:

```
LD A,220        ; A ← 220
OUT (#78),A     ; scrive 220 sulla
                ; porta #78.
```

3.96 OUTD

SINTASSI: OUTD

SCOPO: Scrivere sulla porta puntata dal registro C, il contenuto della locazione puntata da HL. Decrementare poi HL e B.

(C) ← (HL)
HL ← HL - 1
B ← B - 1

FLAG:

S	Z	H	P/V	N	C
X	≈	X	X	1	X

Il flag Z è posto a 1 se il registro B è 0 dopo l'esecuzione.

ESEMPI:

```
LOOP    OUTD
        JR NZ,LOOP
```

3.97 OUTI

SINTASSI: OUTI

SCOPO: Scrivere sulla porta puntata dal registro C, il contenuto della locazione puntata da HL. Incrementare poi HL e decrementare B.

(C) ← (HL)
HL ← HL + 1
B ← B - 1

FLAG:

S	Z	H	P/V	N	C
X	≈	X	X	1	X

Il flag Z è posto a 1 se il registro B è 0 dopo l'esecuzione.

ESEMPI:

```
LOOP    OUTI
        JR NZ,LOOP
```

3.98 OTDR

SINTASSI: OTDR

SCOPO: Scrivere sulla porta puntata dal registro C, il contenuto della locazione puntata da HL. Decrementare poi HL, decrementare B e ripetere l'operazione finché B non diventa 0.

Ripeti

(C) ← (HL)
HL ← HL - 1
B ← B - 1

finché B=0

FLAG:

S	Z	H	P/V	N	C
X	≈	X	X	1	X

Il flag Z è posto a 1 se il registro B è 0 dopo l'esecuzione.

3.99 OTIR

SINTASSI: OTIR

SCOPO: Scrivere sulla porta puntata dal registro C, il contenuto della locazione puntata da HL. Incrementare poi HL, decrementare B e ripetere l'operazione finché B non diventa 0.

Ripeti

(C) ← (HL)
HL ← HL + 1
B ← B - 1

finché B=0

FLAG:

S	Z	H	P/V	N	C
X	≈	X	X	1	X

Il flag Z è posto a 1 se il registro B è 0 dopo l'esecuzione.

3.100 POP REG16

SINTASSI: POP REG16

Dove REG16 rappresenta i registri AF, BC, DE, HL, IX, IY.

SCOPO: Prelevare il registro REG16 dalla cima dello stack.

REG16_basso ← (SP)
REG16_alto ← (SP+1)
SP ← SP + 2

FLAG:

S	Z	H	P/V	N	C
•	•	•	•	•	•

ESEMPI:

```
POP HL ; Prelev. HL dallo stack.
POP IX ; Prelev. IX dallo stack.
POP AF ; Prelev. AF dallo stack.
```

3.101 PUSH REG16

SINTASSI: PUSH REG16

Dove REG16 rappresenta i registri AF, BC, DE, HL, IX, IY.

SCOPO: Depositare il registro REG16 nella cima dello stack.

(SP-1) ← REG16_alto
(SP-2) ← REG16_basso
SP ← SP - 2

FLAG:

S	Z	H	P/V	N	C
•	•	•	•	•	•

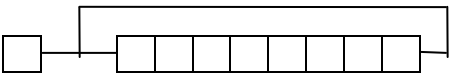
ESEMPI:

```
PUSH DE ;Deposita DE nello stack.
PUSH IY ;Deposita IY nello stack.
PUSH AF ;Deposita AF nello stack.
```

3.102 RLCA

SINTASSI: RLCA

SCOPO: Ruotare gli 8 bit dell'accumulatore a sinistra di una posizione. Il contenuto del bit 7 viene spostato sia nel bit 0, sia nel flag del carry.



FLAG:

S	Z	H	P/V	N	C
•	•	0	•	0	≈

Il flag C viene posizionato dal bit 7 di A.

3.103 RES nb,REG8

SINTASSI: RES nb,REG8

Dove REG8 rappresenta i registri A, B, C, D, E, H, L, mentre nb il numero del bit (quindi è una costante compresa tra 0 e 7).

SCOPO: Settare a 0 il bit numero nb del registro REG8.

REG8_nb ← 0

FLAG:

S	Z	H	P/V	N	C
•	•	•	•	•	•

ESEMPI:

```
RES 0,D  ; pone a 0 il LSB di D
RES 4,E  ; pone a 0 il bit 4 di E
RES 7,C  ; pone a 0 il MSB di C
```

3.104 RES nb,(PMEM)

SINTASSI: RES nb,(PMEM)

Dove PMEM è un puntatore alla memoria del tipo HL, IX+DD, IY+DD.

SCOPO: Settare a 0 il bit numero nb della locazione di memoria puntata da PMEM.

(PMEM)_nb ← 0

FLAG:

S	Z	H	P/V	N	C
•	•	•	•	•	•

ESEMPI:
```
RES 2,(HL)
RES 4,(IX+2)
RES 1,(IY-3)
```

3.105 RET

SINTASSI: RET

SCOPO: Ritornare da un sottoprogramma. La prossima istruzione ad essere eseguita è quella successiva alla CALL che ha fatto la chiamata.

PC_basso ← (SP)
PC_alto ← (SP+1)
SP ← SP + 2

FLAG:

S	Z	H	P/V	N	C
•	•	•	•	•	•

3.106 RET CC

SINTASSI: RET CC

Dove CC è una delle condizioni:
NZ, Z, NC, C, PO, PE, P, M.

SCOPO: Se la condizione è vera, ritornare da un sottoprogramma.
La prossima istruzione (sempre se la condizione è vera) ad essere eseguita è quella successiva alla CALL che ha fatto la chiamata.

Se CC è vera:

PC_basso ← (SP)
PC_alto ← (SP+1)
SP ← SP + 2

FLAG:

S	Z	H	P/V	N	C
•	•	•	•	•	•

3.107 RETI

SINTASSI: RETI

SCOPO: Ritornare dalla routine di gestione dell'interrupt. La prossima istruzione eseguita è quella successiva a quella in cui lo Z80 ha accettato l'interruzione.

PC_basso ← (SP)
PC_alto ← (SP+1)
SP ← SP + 2

FLAG:

S	Z	H	P/V	N	C
•	•	•	•	•	•

NOTE: Il codice macchina della RETI è riconosciuto dai periferici come parte finale di una routine d'interrupt. In Z80 Simulation questa istruzione si comporta come una normale RET.

3.108 RETN

SINTASSI: RETN

SCOPO: Ritornare dalla routine di gestione dell'interrupt non mascherabile. La prossima istruzione eseguita è quella successiva a quella in cui lo Z80 ha accettato l'interruzione.

PC_basso ← (SP)
PC_alto ← (SP+1)
SP ← SP + 2
IFF1 ← IFF2

FLAG:

S	Z	H	P/V	N	C
•	•	•	•	•	•

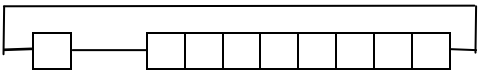
NOTE: In Z80 Simulation questa istruzione si comporta come una normale RET.

3.109 RL REG8

SINTASSI: RL REG8

Dove REG8 rappresenta i registri A, B, C, D, E, H, L.

SCOPO: Ruotare gli 8 bit del registro REG8 a sinistra di una posizione attraverso il carry.



FLAG:

S	Z	H	P/V	N	C
≈	≈	0	≈	0	≈

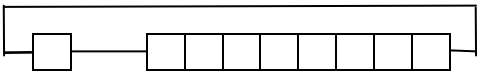
Il flag C viene posizionato dal bit 7 di REG8.
Il flag P/V lavora come flag di parità.

3.110 RL (PMEM)

SINTASSI: RL (PMEM)

Dove PMEM è un puntatore alla memoria del tipo HL, IX+DD, IY+DD.

SCOPO: Ruotare gli 8 bit della locazione di memoria puntata da PMEM a sinistra di una posizione attraverso il carry.



Carry Registro (PMEM)

FLAG:

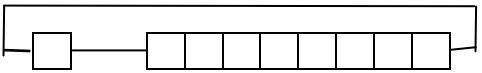
S	Z	H	P/V	N	C
≈	≈	0	≈	0	≈

Il flag C viene posizionato dal bit 7 di (PMEM).
Il flag P/V lavora come flag di parità.

3.111 RLA

SINTASSI: RLA

SCOPO: Ruotare gli 8 bit dell'accumulatore a sinistra di una posizione attraverso il carry.



FLAG:

S	Z	H	P/V	N	C
•	•	0	•	0	≈

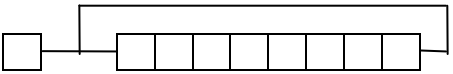
Il flag C viene posizionato dal bit 7 di A.

3.112 RLC REG8

SINTASSI: RLC REG8

Dove REG8 rappresenta i registri A, B, C, D, E, H, L.

SCOPO: Ruotare gli 8 bit del registro REG8 a sinistra di una posizione. Il contenuto del bit 7 viene spostato sia nel bit 0, sia nel flag del carry.



FLAG:

S	Z	H	P/V	N	C
≈	≈	0	≈	0	≈

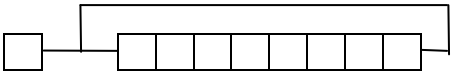
Il flag C viene posizionato dal bit 7 di REG8.
Il flag P/V lavora come flag di parità.

3.113 RLC (PMEM)

SINTASSI: RLC (PMEM)

Dove PMEM è un puntatore alla memoria del tipo HL, IX+DD, IY+DD.

SCOPO: Ruotare gli 8 bit della locazione di memoria puntata da PMEM a sinistra di una posizione. Il contenuto del bit 7 viene spostato sia nel bit 0, sia nel flag del carry.



Carry Locazione (PMEM)

FLAG:

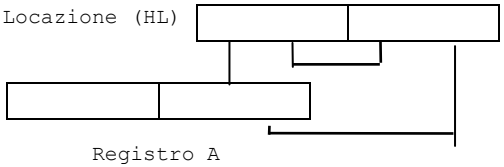
S	Z	H	P/V	N	C
≈	≈	0	≈	0	≈

Il flag C viene posizionato dal bit 7 di (PMEM).
Il flag P/V lavora come flag di parità.

3.114 RLD

SINTASSI: RLD

SCOPO: Spostare i 4 bit di ordine inferiore del byte indirizzato da HL, nelle posizioni dei 4 bit di ordine superiore della locazione stessa. I 4 bit di ordine superiore del byte vengono spostati nella parte bassa dell'accumulatore, mentre la parte bassa di A viene spostata nei bit di ordine inferiore della locazione di memoria.



FLAG:

S	Z	H	P/V	N	C
≈	≈	0	≈	0	•

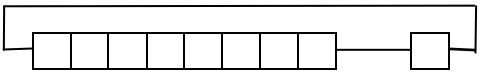
Il flag P/V indica la parità.

3.115 RR REG8

SINTASSI: RR REG8

Dove REG8 rappresenta i registri A, B, C, D, E, H, L.

SCOPO: Ruotare gli 8 bit del registro REG8 a destra di una posizione attraverso il carry.



FLAG:

S	Z	H	P/V	N	C
≈	≈	0	≈	0	≈

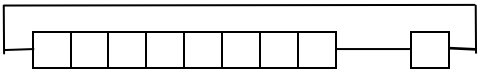
Il flag C viene posizionato dal bit 0 di REG8.
Il flag P/V lavora come flag di parità.

3.116 RR (PMEM)

SINTASSI: RR (PMEM)

Dove PMEM è un puntatore alla memoria del tipo HL, IX+DD, IY+DD.

SCOPO: Ruotare gli 8 bit della locazione di memoria puntata da PMEM a destra di una posizione attraverso il carry.



FLAG:

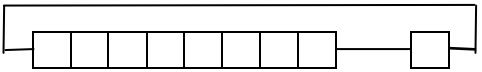
S	Z	H	P/V	N	C
≈	≈	0	≈	0	≈

Il flag C viene posizionato dal bit 0 di (PMEM).
Il flag P/V lavora come flag di parità.

3.117 RRA

SINTASSI: RRA

SCOPO: Ruotare gli 8 bit dell'accumulatore a destra di una posizione attraverso il carry.



FLAG:

S	Z	H	P/V	N	C
•	•	0	•	0	≈

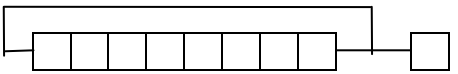
Il flag C viene posizionato dal bit 0 di A.

3.118 RRC REG8

SINTASSI: RRC REG8

Dove REG8 rappresenta i registri A, B, C, D, E, H, L.

SCOPO: Ruotare gli 8 bit del registro REG8 a destra di una posizione. Il contenuto del bit 0 viene spostato sia nel bit 7, sia nel flag del carry.



FLAG:

S	Z	H	P/V	N	C
≈	≈	0	≈	0	≈

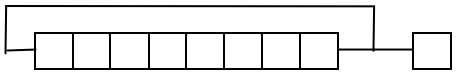
Il flag C viene posizionato dal bit 0 di REG8.
Il flag P/V lavora come flag di parità.

3.119 RRC (PMEM)

SINTASSI: RRC (PMEM)

Dove PMEM è un puntatore alla memoria del tipo HL, IX+DD, IY+DD.

SCOPO: Ruotare gli 8 bit della locazione di memoria puntata da PMEM a destra di una posizione. Il contenuto del bit 0 viene spostato sia nel bit 7, sia nel flag del carry.



FLAG:

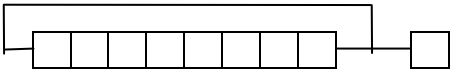
S	Z	H	P/V	N	C
≈	≈	0	≈	0	≈

Il flag C viene posizionato dal bit 0 di (PMEM).
Il flag P/V lavora come flag di parità.

3.120 RRCA

SINTASSI: RRCA

SCOPO: Ruotare gli 8 bit dell'accumulatore a destra di una posizione. Il contenuto del bit 0 viene spostato sia nel bit 7, sia nel flag del carry.



FLAG:

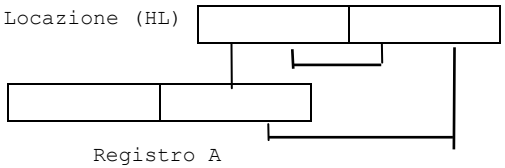
S	Z	H	P/V	N	C
•	•	0	•	0	≈

Il flag C viene posizionato dal bit 0 di A.

3.121 RRD

SINTASSI: RRD

SCOPO: Spostare i 4 bit di ordine superiore del byte indirizzato da HL, nelle posizioni dei 4 bit di ordine inferiore della locazione stessa. I 4 bit di ordine inferiore del byte vengono spostati nella parte bassa dell'accumulatore, mentre la parte bassa di A viene spostata nei bit di ordine superiore della locazione di memoria.



FLAG:

S	Z	H	P/V	N	C
≈	≈	0	≈	0	•

Il flag P/V indica la parità.

3.122 RST VAL8

SINTASSI: RST VAL8

Dove VAL8 è uno dei seguenti valori: #00, #08, #10, #18, #20, #28, #30, #38.

SCOPO: Chiamare il sottoprogramma che si trova all'indirizzo di memoria VAL8.

(SP-1) ← PC_alto
(SP-2) ← PC_basso
SP ← SP - 2
PC_alto ← 0
PC_basso ← VAL8

FLAG:

S	Z	H	P/V	N	C
•	•	•	•	•	•

NOTE: In Z80 Simulation, questa istruzione non ha alcun effetto, poiché è quasi sempre utilizzata per gestire le interruzioni.

3.123 SBC A,VAL8

SINTASSI: SBC A,VAL8

Dove VAL8 è una costante a 8 bit.

SCOPO: Sottrarre al reg. accumulatore la costante VAL8 e il carry, e memorizzare la differenza in A stesso.

A ← A - VAL8 - Carry

FLAG:

S	Z	H	P/V	N	C
≈	≈	≈	≈	1	≈

Il flag P/V lavora come flag di overflow.

ESEMPI:

```
SBC A,12  ; A ← A-12-Carry
SBC A,0   ; A ← A-Carry
SBC A,#FF ; A ← A-255-Carry
```

3.124 SBC A,REG8

SINTASSI: SBC A,REG8

Dove REG8 rappresenta i registri A, B, C, D, E, H, L.

SCOPO: Sottrarre al reg. accumulatore il registro REG8 e il carry, e memorizzare la differenza in A stesso.

A ← A - REG8 - Carry

FLAG:

S	Z	H	P/V	N	C
≈	≈	≈	≈	1	≈

Il flag P/V lavora come flag di overflow.

ESEMPI:

```
SBC A,B ; A ← A-B-Carry
SBC A,C ; A ← A-C-Carry
SBC A,L ; A ← A-L-Carry
```

3.125 SBC A,(PMEM)

SINTASSI: SBC A,(PMEM)

Dove PMEM è un puntatore alla memoria del tipo HL, IX+DD, IY+DD.

SCOPO: Sottrarre al reg. accumulatore il contenuto della locazione di memoria puntata da PMEM e il carry. Memorizzare la differenza in A stesso.

A ← A - (PMEM) - Carry

FLAG:

S	Z	H	P/V	N	C
≈	≈	≈	≈	1	≈

Il flag P/V lavora come flag di overflow.

ESEMPI:

```
SBC A,(HL)     ;A←A-(HL)-Carry
SBC A,(IX+12)  ;A←A-(IX+12)-Carry
SBC A,(IY-16)  ;A←A-(IY-16)-Carry
SBC A,(IX)     ;A←A-(IX+00)-Carry
```

3.126 SBC HL,REG16

SINTASSI: SBC HL,REG16

Dove REG16 rappresenta i registri BC, DE, HL, SP.

SCOPO: Sottrarre al registro HL, il registro REG16 e il carry, e memorizzare la differenza in HL.

HL ← HL - REG16 - Carry

FLAG:

S	Z	H	P/V	N	C
≈	≈	≈	≈	1	≈

Il flag P/V lavora come flag di overflow. H è posto a 1 se si verifica un prestito al bit 11.

ESEMPI:

```
SBC HL,BC ; HL ← HL-BC-Carry
SBC HL,DE ; HL ← HL-DE-Carry
SBC HL,HL ; HL ← HL-HL-Carry
SBC HL,SP ; HL ← HL-SP-Carry
```

3.127 SCF

SINTASSI: SCF

SCOPO: Settare a 1 il flag del carry.

Carry ← 1

FLAG:

S	Z	H	P/V	N	C
•	•	0	•	0	1

ESEMPI:
```
SCF ; Carry ← 1
CCF ; Carry ← 0
```

3.128 SET nb,REG8

SINTASSI: SET nb,REG8

Dove REG8 rappresenta i registri A, B, C, D, E, H, L, mentre nb il numero del bit (quindi è una costante compresa tra 0 e 7).

SCOPO: Settare a 1 il bit numero nb del registro REG8.

REG8_nb ← 1

FLAG:

S	Z	H	P/V	N	C
•	•	•	•	•	•

ESEMPI:

```
SET 0,D  ; pone a 1 il LSB di D
SET 4,E  ; pone a 1 il bit 4 di E
SET 7,C  ; pone a 1 il MSB di C
```

3.129 SET nb,(PMEM)

SINTASSI: SET nb,(PMEM)

Dove PMEM è un puntatore alla memoria del tipo HL, IX+DD, IY+DD.

SCOPO: Settare a 1 il bit numero nb della locazione di memoria puntata da PMEM.

(PMEM)_nb ← 1

FLAG:

S	Z	H	P/V	N	C
•	•	•	•	•	•

ESEMPI:
```
SET 3,(HL)
SET 2,(IX+35)
SET 6,(IY-10)
```

3.130 SLA REG8

SINTASSI: SLA REG8

Dove REG8 rappresenta i registri A, B, C, D, E, H, L.

SCOPO: Spostare aritmeticamente a sinistra di una posizione, gli 8 bit di REG8. Nel bit meno significativo entra uno 0, mentre nel carry entra il bit più significativo.

FLAG:

S	Z	H	P/V	N	C
≈	≈	0	≈	0	≈

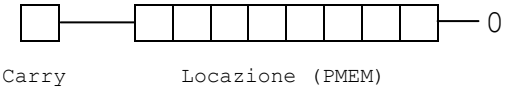
Il flag C viene posizionato dal bit 7 di REG8.
Il flag P/V lavora come flag di parità.

3.131 SLA (PMEM)

SINTASSI: SLA (PMEM)

Dove PMEM è un puntatore alla memoria del tipo HL, IX+DD, IY+DD.

SCOPO: Spostare in modo aritmetico, gli 8 bit della locazione puntata da PMEM di una posizione a sinistra. Nel bit meno significativo entra uno 0, mentre nel carry entra il bit più significativo del byte.



FLAG:

S	Z	H	P/V	N	C
≈	≈	0	≈	0	≈

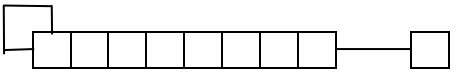
Il flag C viene posizionato dal bit 7 di (PMEM).
Il flag P/V lavora come flag di parità.

3.132 SRA REG8

SINTASSI: SRA REG8

Dove REG8 rappresenta i registri A, B, C, D, E, H, L.

SCOPO: Spostare in modo aritmetico, gli 8 bit di REG8, di una posizione a destra. Il bit meno significativo entra nel carry, mentre il bit più significativo rimane invariato per conservare il segno.



Registro REG8 Carry

FLAG:

S	Z	H	P/V	N	C
≈	≈	0	≈	0	≈

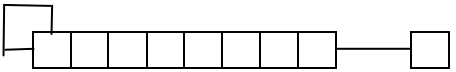
Il flag C viene posizionato dal bit 0 di REG8.
Il flag P/V lavora come flag di parità.

3.133 SRA (PMEM)

SINTASSI: SRA (PMEM)

Dove PMEM è un puntatore alla memoria del tipo HL, IX+DD, IY+DD.

SCOPO: Spostare in modo aritmetico, gli 8 bit del byte puntato da PMEM di una posizione a destra. Il bit meno significativo entra nel carry, mentre il bit più significativo rimane invariato per conservare il segno.



Locazione (PMEM) Carry

FLAG:

S	Z	H	P/V	N	C
≈	≈	0	≈	0	≈

Il flag C viene posizionato dal bit 0 di (PMEM).
Il flag P/V lavora come flag di parità.

3.134 SRL REG8

SINTASSI: SRL REG8

Dove REG8 rappresenta i registri A, B, C, D, E, H, L.

SCOPO: Spostare in modo logico gli 8 bit di REG8, di una posizione a destra. Il bit meno significativo entra nel carry, mentre in quello più significativo entra uno 0.

FLAG:

S	Z	H	P/V	N	C
≈	≈	0	≈	0	≈

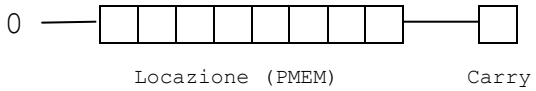
Il flag C viene posizionato dal bit 0 di REG8.
Il flag P/V lavora come flag di parità.

3.135 SRL (PMEM)

SINTASSI: SRL (PMEM)

Dove PMEM è un puntatore alla memoria del tipo HL, IX+DD, IY+DD.

SCOPO: Spostare in modo logico gli 8 bit del byte puntato da PMEM di una posizione a destra. Il bit meno significativo entra nel carry, mentre in quello più significativo entra uno 0.



FLAG:

S	Z	H	P/V	N	C
≈	≈	0	≈	0	≈

Il flag C viene posizionato dal bit 0 di (PMEM).
Il flag P/V lavora come flag di parità.

3.136 SUB VAL8

SINTASSI: SUB VAL8

Dove VAL8 è una costante a 8 bit.

SCOPO: Sottrarre al reg. accumulatore la costante VAL8 e memorizzare la differenza in A stesso.

A ← A - VAL8

FLAG:

S	Z	H	P/V	N	C
≈	≈	≈	≈	1	≈

Il flag P/V lavora come flag di overflow.

ESEMPI:

```
SUB 12  ; A ← A-12
SUB 2   ; A ← A-2
SUB #FF ; A ← A-255
```

3.137 SUB REG8

SINTASSI: SUB REG8

Dove REG8 rappresenta i registri A, B, C, D, E, H, L.

SCOPO: Sottrarre al reg. accumulatore il registro REG8 e memorizzare la differenza in A stesso.

A ← A - REG8

FLAG:

S	Z	H	P/V	N	C
≈	≈	≈	≈	1	≈

Il flag P/V lavora come flag di overflow.

ESEMPI:

```
SUB B ; A ← A-B
SUB C ; A ← A-C
SUB L ; A ← A-L
```

3.138 SUB (PMEM)

SINTASSI: SUB (PMEM)

Dove PMEM è un puntatore alla memoria del tipo HL, IX+DD, IY+DD.

SCOPO: Sottrarre al reg. accumulatore il contenuto della locazione di memoria puntata da PMEM, e memorizzare la differenza in A.

A ← A - (PMEM)

FLAG:

S	Z	H	P/V	N	C
≈	≈	≈	≈	1	≈

Il flag P/V lavora come flag di overflow.

ESEMPI:

```
SUB (HL)     ; A← A-(HL)
SUB (IX+12)  ; A← A-(IX+12)
SUB (IY-16)  ; A← A-(IY-16)
SUB (IX)     ; A← A-(IX+00)
```

3.139 XOR VAL8

SINTASSI: XOR VAL8

Dove VAL8 è una costante a 8 bit.

SCOPO: Fare l'operazione logica di XOR fra i bit del registro A e i bit della costante VAL8. Memorizzare il risultato in A stesso.

A ← A XOR VAL8

FLAG:

S	Z	H	P/V	N	C
≈	≈	0	≈	0	0

Il flag P/V lavora come flag di parità.

ESEMPI:

```
XOR #B2   ; A ← A XOR #B2
XOR 1     ; A ← A XOR 1
XOR 100   ; A ← A XOR 100
```

3.140 XOR REG8

SINTASSI: XOR REG8

Dove REG8 rappresenta i registri A, B, C, D, E, H, L.

SCOPO: Fare l'operazione logica di XOR fra i bit del registro A e i bit del registro REG8. Memorizzare il risultato in A stesso.

A ← A XOR REG8

FLAG:

S	Z	H	P/V	N	C
≈	≈	0	≈	0	0

Il flag P/V lavora come flag di parità.

ESEMPI:

```
XOR A  ; A ← A XOR A
XOR L  ; A ← A XOR L
XOR C  ; A ← A XOR C
```

3.141 XOR (PMEM)

SINTASSI: XOR (PMEM)

Dove PMEM è un puntatore alla memoria del tipo HL, IX+DD, IY+DD.

SCOPO: Fare l'operazione logica di XOR fra i bit del registro A e i bit del byte di memoria puntato da PMEM. Memorizzare il risultato in A stesso.

A ← A XOR (PMEM)

FLAG:

S	Z	H	P/V	N	C
≈	≈	0	≈	0	0

Il flag P/V lavora come flag di parità.

ESEMPI:

```
XOR (HL)     ; A ← A XOR (HL)
XOR (IX+15)  ; A ← A XOR (IX+15)
XOR (IY-10)  ; A ← A XOR (IY-10)
```

APPENDICE A

Obiettivi dell'appendice:

- Descrizione dei messaggi d'errore

A.1 I MESSAGGI D'ERRORE

Abbiamo avuto già modo di vedere, che quando si presenta una situazione anomala, Z80 Simulation visualizza un messaggio d'errore nella finestra di Input/Output ed emette un beep.

A.2 DESCRIZIONE DEI MESSAGGI D'ERRORE

Vediamo adesso il significato di ciascun messaggio d'errore:

1. **"Il file non è stato trovato.".** Si è tentato di leggere un file che non poteva essere aperto.
2. **"Errore durante la lettura del file.".** Durante la lettura di un file, si è verificato un errore che ha reso impossibile il completamento dell'operazione.
3. **"Impossibile scrivere il file.".** Si è tentato di scrivere su di un file che non poteva essere aperto in scrittura.
4. **"Errore durante la scrittura del file.".** Durante la scrittura di un file, si è verificato un errore che ha reso impossibile il completamento dell'operazione.
5. **"Files non trovati.".** Con il comando di visualizzazione delle directory, si sono specificati dei file che non esistono.
6. **"Errore durante la lettura della directory.".** Durante la lettura della directory, si è verificato un errore che ha reso impossibile il completamento dell'operazione.
7. **"ERRORE: Impossibile caricare. COMMAND.COM. File non trovato.".** L'operazione di DOS shell non è stata completata poiché né nella directory corrente né in quella specificata dalla variabile d'ambiente COMSPEC, era presente una copia del file COMMAND.COM.
8. **"ERRORE: Impossibile caricare COMMAND.COM. Memoria insufficiente.".** L'operazione di DOS shell non è stata completata poiché non vi era abbastanza memoria.
9. **"ERRORE: il testo non contiene la stringa xxxxxxxxx".** Si è tentato di sostituire con il comando [Ctrl]+[S] una stringa non presente nell'editor.
10. **"Codice non presente.".** Si è attivato il compilatore quando nell'editor non vi era nessun programma.

11. **"ERRORE: nel sorgente non ci sono istruzioni.".** Si è attivato l'assemblatore, ma il sorgente conteneva solo pseudo-istruzioni e/o messaggi.
12. **"ERRORE alla linea nnnn. Spiazzamento non compreso nel range -128..127.".** Un'istruzione JR (salto relativo) ha tentato di saltare ad un indirizzo troppo lontano.
13. **"ERRORE alla linea nnnn: Label già utilizzata.".** Si è cercato di definire una label già definita.
14. **"ERRORE alla linea nnnn: Costante o label già utilizzata.".** Si è cercato di definire con EQU una costante già definita o con lo stesso nome di una label.
15. **"Errore di sintassi alla linea nnnn".** L'istruzione non è stata riconosciuta come tale o qualche operando non è corretto.
16. **"ERRORE di Lettura/Scrittura in MEMORIA. Indirizzo fuori intervallo.".** Durante l'esecuzione del programma, l'istruzione evidenziata nell'editor, ha tentato di leggere o scrivere ad un indirizzo in cui non vi è memoria.
17. **"ERRORE: Valore di PC non valido.".** Un'istruzione di salto ha caricato nel PC il numero di una linea che nel programma non esiste.
18. **"Troppe label.".** Si sono definite più di 250 label.
19. **"Troppe definizioni di costanti.".** Si sono definite più di 250 costanti con la pseudo-istruzione EQU.

APPENDICE B

Obiettivi dell'appendice:

- Descrizione dei file di Z80 Simulation

B.1 COSA TROVIAMO SUL FLOPPY DISK

Il floppy disk che contiene Z80 Simulation include due file:

1. Il file "**Z80.EXE**". Si tratta del programma eseguibile (l'unico veramente essenziale). Per avviare Z80 Simulation è infatti necessario (dopo aver avviato il sistema con il DOS):

   - Inserire il floppy nel drive A (o B);
   - Digitare al prompt:

     A:\>Z80 (o B:\>Z80)

   Si noti che si può fare l'avviamento anche da un drive che non è quello corrente, digitando ad esempio:

     A:\>B:\Z80

   Tuttavia, in questo caso, Z80 Simulation potrebbe non riuscire a caricare il file di configurazione.

1. Il file "**Z80.CFG**". E' il file che contiene l'ultima configurazione salvata con il SETUP. Se questo file non esiste, Z80 Simulation si configura con la configurazione di default.

B.2 I TIPI DI FILE

Oltre ai file precedentemente descritti, man mano che si lavora con Z80 Simulation, si creano nuovi file. Vediamo di che file si tratta:

1. **File "programma".** Questi file contengono programmi. Sono stati creati mediante il comando dell'ambiente [Files]/[Registra], oppure salvando un blocco con il comando [F8] dell'editor. Sono file di formato ASCII e l'estensione di default è ".Z80".

2. **File "memoria".** Sono i file che contengono la memoria salvata mediante il comando dell'ambiente [Options]/[Memoria]/[Save]. Sono file di formato binario, di lunghezza fissa (16K) e la loro estensione di default è ".RAM".
3. **File "foglio assembler".** Sono generati dall'assemblatore, e contengono uno dei due tipi di foglio assembler. Il loro formato è ASCII e la loro estensione di default è ".TXT".
4. **File "formato binario".** Anche questi sono generati dall'assemblatore e contengono il codice macchina. Sono adatti per il trasferimento del codice su sistemi con microprocessore Z80, o per la programmazione di EPROM. Il loro formato è binario, e la loro estensione di default è ".BIN".
5. **File di riserva.** Quando Z80 Simulation crea un file, esso controlla se ne esiste già uno con lo stesso nome. In caso affermativo, il file già presente viene rinominato, ovvero conserva lo stesso nome, ma la sua estensione diventa ".BAK". Ciò permette all'utente di ripristinare il file qualora si presentasse la necessità. La produzione dei file di riserva può essere inibita mediante il SETUP.

APPENDICE C

Obiettivi dell'appendice:

- Descrizione della gestione delle interruzioni nel microprocessore Z80.

C.1 LE INTERRUZIONI IN Z80 SIMULATION

Si è visto nel capitolo 3, che le istruzioni che gestiscono le interruzioni, benché accettate dal compilatore (e quindi dall'assemblatore), in fase di esecuzione non hanno alcun effetto. Questo a causa del fatto che Z80 Simulation non simula il meccanismo di interruzione presente nello Z80 vero e proprio. Si è comunque ritenuto utile, descrivere, anche se superficialmente, tale meccanismo.

C.2 LE INTERRUZIONI MASCHERABILI

Le interruzioni mascherabili sono caratterizzate dal fatto di poter essere attivate e disattivate dal programmatore stesso. La richiesta d'interruzione di questo tipo viene effettuata dai dispositivi esterni, tramite il piedino INT dello Z80, e quest'ultimo la prende in considerazione solo se il flip-flop IFF è posto a 1. Per settare il flip-flop IFF (quindi per abilitare o disabilitare le interruzioni) si utilizzano le istruzione EI (Enable Interrupt) che lo pone a 1 e l'istruzione DI (Disable Interrupt) che lo pone a 0.
Se lo Z80 accetta l'interruzione, interrompe il programma che sta eseguendo, salva il PC nello stack per poi poter riprendere l'esecuzione dal punto in cui l'ha interrotta, e poi risponde in 3 modi diversi a seconda che sia attivo il modo d'interruzione 0, 1 o 2. Vediamo cosa succede nei vari casi:

1. Se lo Z80 si trova in modo 0 (istruzione IM 0) preleva la prossima istruzione dal BUS DATI, in modo tale che sia il dispositivo periferico a emettere il codice macchina (e non la memoria). Generalmente si tratta di istruzioni di restart (RST).
2. Se lo Z80 si trova in modo 1 (istruzio ne IM 1) esegue il sottoprogramma all'indirizzo #0038.
3. Se lo Z80 si trova in modo 2 (istruzione IM 2), legge dal BUS DATI il codice di riconoscimento emesso dal dispositivo che ha richiesto l'interruzione, lo concatena con il registro I formando così l'indirizzo a 16 bit (di cui I è la parte alta) in cui preleverà un altro indirizzo dove troverà la prima istruzione della routine di gestione dell'interruzione.

Dei tre modi di risposta all'interruzione, l'ultimo appare chiaramente il più potente, poiché permette, senza alcuna tecnica di polling, di gestire più dispositivi.

C.3 LE INTERRUZIONI NON MASCHERABILI

Le interruzioni non mascherabili, contrariamente a quelle mascherabili, non possono essere disattivate, ovvero lo Z80 non può fare a meno di rispondere a un'interruzione di questo tipo.
La richiesta avviene tramite il piedino NMI, che quando portato a livello basso impone allo Z80 di salvare il PC nello stack e di eseguire la routine che si trova all'indirizzo #0066.