

SINCLAIR ZX SPECTRUM ASSEMBLER E LINGUAGGIO MACCHINA PER PRINCIPIANTI

di William Tang

Traduzione a cura della JCE

con la consulenza di
Rita Bonelli



Via dei Lavoratori, 124
CINISELLO BALSAMO (MI)

Pubblicato in Inghilterra da:
Melbourne House (Publishers) Ltd.,
Glebe Cottage, Glebe House,
Station Road, Cheddington,
Leighton Buzzard, Bedfordshire, LU7 7NA,
ISBN 0 86161 110 1

Pubblicato in Australia da:
Melbourne House (Australia) Pty. Ltd.,
Suite 4, 75 Palmerston Crescent,
South Melbourne, Victoria, 3205,
National Library of Australia Card Number and
ISBN 0 86759 113 7

Pubblicato negli Stati Uniti d'America da:
Melbourne House Software Inc.,
347 Reedwood Drive,
Nashville NT 37217

Copyright (c) 1982 Beam Software
Copyright (a) per l'edizione italiana: Edizioni JCE, 1984

Tutti i diritti sono riservati. Nessuna parte di questo libro puo' essere riprodotta, posta in sistemi di archiviazione, trasmessa in qualsiasi forma o mezzo, elettronico, meccanico, fotocopiatura, ecc., senza l'autorizzazione scritta dell'editore.

Prima edizione: Marzo 1984

Stampato in Italia da:
Gemm Grafica S.r.l.
Via Magretti - Paderno Dugnano (MI)

I N D I C E

=====

PARTE PRIMA: Alla scoperta del linguaggio macchina

Introduzione	7
Concetti base per la programmazione in linguaggio macchina ..	13
L'aritmetica del calcolatore	20
Come vengono rappresentate le informazioni	27
Uno sguardo all'interno della CPU	34
Tutto questo e' bellissimo ma	43
Come la CPU usa i propri arti (registri)	47
Assegnare valori ad una mano	54
I flag ed il loro uso	63
Contare in piu' od in meno	70
Aritmetica su una mano	74
Operatori logici	80
Precisazione sui numeri a 16 bit	85
Come manipolare i numeri con due mani	88
Operazioni sullo stack	94
Aritmetica dei numeri a 16 bit	98
Cicli e salti	103
Uso dei sottoprogrammi	110
Operazioni su blocchi di dati	113

PARTE SECONDA: Le istruzioni usate meno frequentemente

Scambio tra registri	119
BIT, SET e RESET	121
ROTATE e SHIFT	123
IN e OUT	127
Rappresentazione BCD	132
Interruzioni	134
Istruzioni di RESTART	135

PARTE TERZA: Programmazione dello Spectrum

Progettazione di un programma in linguaggio macchina	137
Struttura del sistema ZX Spectrum	143

PARTE QUARTA: Programmi monitor

Programma EZ-CODE	153
Programma HEXLOAD	164

PARTE QUINTA: Il programma FREEWAY FROG

Progetto del programma	171
Fase 1: database	174
Fase 2: inizializzazione	183
Fase 3: traffico regolare	187
Fase 4: auto della polizia	192
Fase 5: la rana	196
Fase 6: controllo	201
Listato del programma Assembler	202

APPENDICI

Tabella dei tasti dello Spectrum	238
Mappa del video	239
Tabella dei caratteri dello Spectrum	240
Tabella di conversione decimale-esadecimale	241
Complemento a due	243
Tavola di addizione esadecimale	244
Operazioni sui flag	245
Istruzioni dello Z80 ordinate per codice mnemonico	247
Istruzioni dello Z80 ordinate per codice macchina	251
Istruzioni per l'uso della cassetta	255

Sono sicuro che non vi meravigliarete nell'apprendere che la CPU non capisce una sola parola di BASIC! Infatti non e' stata progettata per colloquiare direttamente con noi.

Pensandoci su un po' capirete che si incontrerebbero serie difficoltà (ed in taluni casi sarebbe addirittura impossibile) a dare ad un chip, posto in un computer, istruzioni in una forma che mantenga qualche significato anche per l'uomo. Provate ad aprire il vostro Sinclair (se ne avete il coraggio) e date un'occhiata all'interno: sicuramente, tra i vari componenti, riconoscerete il chip Z80A: e' quello posto vicino all'altoparlante. Ovviamente questo chip, posto all'interno del computer, puo' rispondere solo a segnali di tipo elettrico trasmessigli dai componenti circuitali rimanenti!

Che cosa e' il linguaggio macchina?

Il microprocessore Z80 e' stato progettato in modo da poter accettare contemporaneamente i segnali che giungono su otto dei suoi piedini di connessione.

I progettisti dello Z80 hanno inoltre costruito il microprocessore in modo tale che ogni combinazione di segnali giungente sui suoi piedini "istruisca" la CPU per svolgere una diversa funzione.

Pur ricordando che in realta' vengono utilizzati sempre segnali elettrici, per comodita' utilizzeremo una convenzione che ci permetta di rappresentare con dei numeri le configurazioni dei segnali presenti sugli otto piedini. In particolare associeremo a ciascun piedino il valore 1, se su di esso e' presente un segnale, ed il valore 0 in caso contrario.

Un' istruzione per la CPU potrebbe quindi essere rappresentata nel modo seguente:

0 0 1 1 1 1 0 0

E' evidente l'enorme differenza esistente tra istruzioni di questo tipo e le piu' note istruzioni BASIC, come

LET A = A + 1

Orbene, tutte le istruzioni del linguaggio macchina hanno questa nuova forma. Il nome stesso lo dice: e' un linguaggio per la macchina!

Va inoltre notato che ogni costruttore di microprocessori ha ideato per i propri prodotti un linguaggio "ad hoc" e questo non ne facilita certo lo studio.

A questo punto vi chiederete: - Per quale motivo dobbiamo impegnarci nello studio di un linguaggio cosi' strano, se abbiamo la possibilita' di utilizzare per i nostri programmi dei linguaggi di gran lunga piu' simili a quello umano, quali il BASIC, il COBOL o altri?

La ragione e' molto semplice: l'uso del linguaggio macchina comporta diversi vantaggi di portata non trascurabile, quali:

- * MAGGIORE RAPIDITA' D'ESECUZIONE DEL PROGRAMMA
- * USO PIU' EFFICIENTE DELLA MEMORIA
- * MINOR OCCUPAZIONE DELLA MEMORIA (da parte del programma)
- * INDIPENDENZA DAL SISTEMA OPERATIVO

Tutti questi vantaggi derivano dal fatto che le istruzioni del linguaggio macchina sono direttamente interpretabili dalla CPU e non richiedono una preventiva trasformazione.

In realta' anche quando viene mandato in esecuzione un programma BASIC viene eseguito un particolare programma scritto in linguaggio macchina detto sistema operativo.

Questo programma opera logicamente nel seguente modo:

Inizio Ciclo	Leggi la prossima istruzione BASIC Trasformala in una serie di istruzioni in linguaggio macchina Esegui ognuna di queste istruzioni Memorizza il risultato, ove richiesto Ripeti da Inizio Ciclo
--------------	--

Sicuramente vi starete chiedendo dove e' memorizzato questo importante programma. Esso e' gia' posto all'interno del vostro Spectrum, in una particolare area di memoria sulla quale puo' essere effettuata la sola operazione di lettura, il cui contenuto non viene modificato nel tempo. Questo particolare tipo di memoria prende il nome di memoria ROM (Read Only Memory).

L'esecuzione di un programma BASIC puo' richiedere un tempo di 60 volte superiore a quello necessario a eseguire il corrispondente programma scritto direttamente in linguaggio macchina. Questa notevole lentezza e' dovuta sicuramente ai tempi necessari per la traduzione operata dal Sistema Operativo, ma anche al fatto che le istruzioni in codice macchina generate in questa fase non sono sempre le piu' efficienti. Per fare un paragone banale, la differenza tra un programma scritto in linguaggio macchina dal programmatore e lo stesso, scritto pero' dal sistema operativo, e' simile a quella esistente tra il percorso scelto da un autista privato e quello scelto dal responsabile dei servizi pubblici urbani. Nel primo caso infatti l'autista segue, tra i percorsi che conosce, quello piu' idoneo; nel secondo caso invece la scelta e' condizionata non solo dalla velocita' ma anche dalla necessita' di soddisfare le esigenze GENERALI di tutti gli utenti possibili.

Il linguaggio macchina, pero', non presenta solo vantaggi.

L'uso di questo tipo di linguaggio infatti

- * RENDE I PROGRAMMI PIU' DIFFICILI DA INTERPRETARE E DA CORREGGERE
- * LEGA IL PROGRAMMA AL PARTICOLARE SISTEMA USATO
- * RENDE ELEVATO IL NUMERO DELLE ISTRUZIONI
- * RENDE DIFFICOLTOSI I CALCOLI

La scelta del linguaggio non e' dunque una cosa semplice ed e' sempre legata alla tipologia dei problemi che si intendono affrontare. Per realizzare ad esempio un programma articolato per applicazioni gestionali (contabilita', analisi finanziaria od altro), converra' usare un linguaggio che faciliti i calcoli e che all'occorrenza permetta di operare velocemente modifiche sul programma.

Se invece si vuoi costruire un programma interattivo per un video gioco, che richiede tempi di risposta molto brevi, non converra' scegliere un linguaggio, come il BASIC, che richiede tempi "lungi" per la esecuzione.

Sulla scelta di un linguaggio possono influire molti fattori: la quantita' di memoria a disposizione, i tempi di risposta richiesti, il tempo disponibile per la stesura del programma, e cosi' via.

Tornando al linguaggio macchina, possiamo riassumere quanto detto finora affermando che esso e' formato da un insieme di istruzioni interpretabili direttamente da parte della CPU, che verranno descritte per convenzione con dei valori numerici.

Che cosa e' il linguaggio Assembler?

Se l'unico modo di rappresentare il codice macchina fosse quello indicato, vi sarebbero ovviamente ben poche persone in grado di scrivere programmi in tale linguaggio. Risulterebbe poi molto difficile dare un senso ad un programma che si presenti all'incirca cosi':

```
0 0 1 0 0 0 0 1
0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0
. . . .
```

Fortunatamente pero' possiamo inventare dei nomi simbolici da associare a ciascuna di queste istruzioni.

Il linguaggio Assembler e' per l'appunto una simbolizzazione mnemonica del codice macchina piu' facilmente leggibile ed interpretabile.

L' unica vera differenza tra il linguaggio Assembler e quello macchina consiste dunque nella facilita' d'uso e di interpretazione. Questa facilitazione viene pero' pagata a caro prezzo: il linguaggio Assembler infatti, letto piu' facilmente dall'uomo, per contro non puo' essere letto direttamente dalla CPU.

Vi e' comunque un'enorme differenza tra l' Assembler ed il BASIC. Infatti, mentre per quest'ultimo le istruzioni vengono trasformate in una "serie di istruzioni in codice macchina", nel caso dell' Assembler per ogni "istruzione Assembler" esiste una istruzione equivalente in codice macchina e viceversa.

Tra i due linguaggi esiste dunque una corrispondenza UNO A UNO che li rende funzionalmente EQUIVALENTI.

Per rappresentare le istruzioni il linguaggio Assembler utilizza simboli mnemonici (o abbreviazioni). Le istruzioni Assembler si presentano così in forma simile a:

INC HL

A prima vista questa scrittura può sembrare incomprensibile, ma se pensate che INC rappresenta l'abbreviazione di "incrementa" ed HL il nome di una variabile, è sufficiente una sola occhiata per interpretare l'istruzione.

La stessa istruzione in codice macchina si sarebbe presentata nel modo seguente:

0 0 1 0 0 0 1 1

Anche in questo caso ovviamente potete "leggere" l'istruzione, nel senso di riconoscere i numeri in essa rappresentati, ma la sua lettura non vi permette di risalire al significato logico dell'istruzione a meno che non ricorriate ad una tabella di conversione esplicativa o possediate un cervello che ragiona come la CPU di un computer.

Un programma scritto in Assembler può comunque essere trasformato facilmente in codice macchina. Tale lavoro può essere fatto direttamente da voi o tramite un apposito programma. Un programma di conversione siffatto viene generalmente chiamato ASSEMBLATORE.

Nel nostro caso l'assemblatore può essere visto come un programma che esegue il lavoro alquanto noioso di trasformare il vostro programma, scritto in linguaggio Assembler, in una sequenza di istruzioni in codice macchina direttamente interpretabili dallo Spectrum.

A questo punto avrete ormai capito che esiste già un Assemblatore per il sistema ZX Spectrum.

Va tuttavia notato che l'occupazione di memoria dell'Assemblatore è piuttosto elevata (circa 6K) e pertanto il suo utilizzo sui sistemi a 16K risulta piuttosto limitato. Infatti il monitor dello Spectrum occupa circa 7K di memoria e pertanto, dopo aver caricato anche l'assemblatore, rimangono a disposizione dell'utente meno di 4K di memoria, il che equivale a dire che non si possono assemblare programmi il cui codice macchina superi 1/2 K.

L'alternativa al programma assemblatore consiste nell'effettuare da soli la conversione da linguaggio Assembler a codice macchina, usando le tabelle di conversione contenute nell'appendice di questo volume.

Questo lavoro può apparire a prima vista frustrante e molto oneroso, ma è di estrema praticità e vi fornisce un valido strumento per capire il modo in cui lavora la CPU dello Spectrum.

Per questo vi consigliamo, anche nel caso disponiate di un programma assemblatore, di provare a scrivere in Assembler ed a trasformare manualmente in codice macchina almeno qualche piccolo programma, prima di utilizzare il programma di traduzione automatica.

RIEPILOGO

CPU (Central Processing Unit)

E' l'unita' centrale di elaborazione, e' realizzata in un unico chip e svolge all'interno del computer le funzioni di calcolo e di controllo.

Linguaggio Macchina

E' l'unico linguaggio direttamente interpretabile dalla CPU. Per la CPU dello Spectrum, il microprocessore Z80A, il linguaggio macchina e' formato da meno di 200 istruzioni.

Linguaggio BASIC

E' un linguaggio di programmazione ideato per essere facilmente interpretato dall'uomo. Un'istruzione BASIC non puo' essere eseguita direttamente dalla CPU, ma deve essere prima trasformata in un'opportuna sequenza di istruzioni in codice macchina. Per questo motivo in genere i programmi in BASIC richiedono tempi d' esecuzione piu' alti rispetto ai programmi scritti in linguaggio macchina, ma minor tempo per la loro realizzazione.

Linguaggio Assembler

Puo' essere considerato come una rappresentazione simbolica del codice macchina, piu' facilmente leggibile ed interpretabile. Ad esempio l'istruzione Assembler HALT corrisponde al codice macchina 0110110 .

Assemblatore

E' un programma che trasforma le istruzioni Assembler, leggibili ed interpretabili dall' uomo, nelle istruzioni corrispondenti in linguaggio macchina, interpretabili dalla CPU del vostro computer.

Memoria ROM (Read Only Memory)

E' una particolare memoria destinata a contenere un programma complesso scritto in linguaggio macchina, definito di solito come FIRMWARE. Il nome deriva dal fatto che esso e' stato direttamente inserito nello hardware all'atto della produzione della ROM e pertanto non viene cancellato togliendo tensione al sistema. Nel caso dello Spectrum la ROM e' stata programmata con il codice macchina dello Z80 e contiene un programma scritto appositamente per quel tipo di microprocessore. La ROM dello Spectrum occupa le locazioni ad indirizzo piu' basso e precisamente quelle che vanno da 0 a 16383. Inoltre, contrariamente a quanto accade per le altre memorie non potete cambiare il contenuto delle singole locazioni di una memoria ROM, ma solo leggere cio' che vi e' stato scritto dal costruttore.

CONCETTI BASE PER LA PROGRAMMAZIONE IN LINGUAGGIO MACCHINA

CHE COSA E' LA CPU ?

Per comunicare con il computer dobbiamo conoscere sia il tipo di comandi che puo' accettare, sia il linguaggio che il cervello della macchina (la CPU) capisce.

Infatti solo conoscendo il tipo di informazioni che la CPU sa riconoscere ed interpretare si puo' veramente istruire il computer a svolgere le mansioni piu' disparate, in modo da farlo diventare ad esempio un buon giocatore di scacchi o un esperto contabile.

La CPU non e' un grosso mistero. Si potrebbe ad esempio paragonarla ad un omino solitario che vive all'interno dello Spectrum e non sta mai senza far niente.

E non vi e' niente che gli piaccia di piu' del far calcoli.

Ma questo povero omino non possiede ne' carta ne' penna per prendere appunti nel corso del proprio lavoro. Come puo' dunque operare?

Lo schema della CPU

A questo punto vi aspetterete che vi parliamo della struttura della CPU e del perche' essa sia stata progettata in questo modo. Cominciamo col dire che la CPU e' stata progettata per fare solo cose molto semplici ma molto velocemente.

Inoltre, come abbiamo gia' ricordato, la CPU non e' stata dotata dai progettisti ne' di penna ne' di carta. Essa quindi non puo' ricordare i numeri, e per mantenere traccia di cio' che sta facendo deve pertanto utilizzare dei contenitori esterni, per conservare quei valori che dovra' eventualmente riutilizzare.

Per proseguire ora il nostro discorso facciamo un esempio concreto: supponiamo di voler far calcolare alla CPU l'ora di New York, conoscendo l'ora attuale di Londra.

Ora, dato che la CPU non conosce nulla, dobbiamo innanzitutto comunicarle l'ora di Londra (ad esempio le ore 10). La CPU pero' non ha a disposizione alcun posto per conservare questa informazione e inoltre non sa neppure cosa avete intenzione di farle fare in seguito. Pertanto si limita a conservare questa informazione in uno degli appositi contenitori (ad esempio nel contenitore numero 1)

Dobbiamo inoltre comunicarle la differenza in ore tra New York e Londra (5 ore), informazione che la CPU mette ad esempio nel contenitore numero 2.

E' arrivato ora il momento di fare i calcoli. La CPU corre ad aprire il contenitore 1, prende il numero in esso contenuto, fa la stessa cosa

con il contenitore 2, esegue l'operazione richiesta e conserva il risultato in un terzo contenitore (ad esempio nel contenitore 3):

$$10 - 5 = 5$$

Alla fine di questa corsa troviamo nel contenitore 3 l'ora di New York e cioè 5.

Tutto questo correre avanti ed indietro tra i contenitori, facendo calcoli aritmetici od altro, sarebbe piuttosto stressante se la CPU fosse costretta ad eseguire i calcoli mentalmente: fortunatamente però essa può utilizzare, per contare, le dita delle mani e dei piedi, come del resto facciamo anche noi.

Le mani ed i piedi della CPU vengono chiamati REGISTRI.

Come vedremo in seguito, il chip Z80A del vostro Spectrum è riconoscibile proprio per il numero di mani e di piedi che possiede.

Per illustrare con maggior precisione le operazioni che la CPU esegue nel calcolo dell'ora di New York, chiameremo una mano della CPU "MANO A". La sequenza di operazioni che ora vi mostriamo è limitata, per semplicità, solo all'esecuzione del calcolo.

- * Rappresenta il valore contenuto nel contenitore 1 sulle dita della mano A;
- * Sottrai, contando sulle dita, il valore contenuto nel contenitore 2;
- * Guarda ora il valore rimasto sulle dita della mano A e conservalo nel contenitore 3.

Ora, se la CPU funzionasse in tal modo, potremmo giungere alle seguenti conclusioni, alquanto sconcertanti:

1. La CPU sa operare solo con numeri interi, non è cioè in grado di riconoscere valori decimali come 11.53.
2. La CPU può eseguire solo quei calcoli che prevedono valori rappresentabili sulle dita delle sue mani.

Orbene, anche se può sembrare strano, questa è la realtà!

L'unica consolazione sta nel fatto che la CPU ha tante mani e tanti piedi e che con una mano di 8 dita riesce a rappresentare un valore qualsiasi tra 0 e 255.

Non entreremo qui nei dettagli sul come la CPU riesca a rappresentare ben 256 valori distinti utilizzando solo le 8 dita di una mano, in quanto ciò sarà oggetto di studio nel prossimo capitolo; vogliamo solo sottolineare che ciò rappresenta una grossa potenzialità: basta pensare al fatto che l'uomo, utilizzando le dita di entrambe le mani, sa contare solo fino a 10! Per ora quindi ci limiteremo a ricordare che la CPU utilizza ciascuna mano per contare fino a 255 e ciascun

piede, fornito di un numero doppio di dita, per contare fino ad oltre 64000 !

Riprendiamo ora il discorso relativo alle operazioni che la CPU svolge per calcolare l'ora di New York. Finora ci siamo limitati a descrivere in modo informale il processo risolutivo, senza però rappresentarlo in un linguaggio interpretabile dalla CPU.

Per farvi avere un primo approccio con la programmazione in linguaggio macchina, utilizziamo ora le istruzioni dell'Assembler simbolico per descrivere formalmente i vari passi compiuti dalla CPU.

INIZIALIZZAZIONE:

```
LD (CONTENITORE1), 10 ;metti nel contenitore 1 il valore 10
LD (CONTENITORE2), 5 ;metti nel contenitore 2 il valore 5
```

CALCOLO:

```
LD A, (CONTENITORE1) ;carica in A il valore contenuto nel
;nel contenitore 1
SUB (CONTENITORE2) ;sottrai il valore contenuto nel con-
;tenitore 2
```

MEMORIZZAZIONE DEL RISULTATO:

```
LD (CONTENITORE3), A ;metti nel contenitore 3 il valore
; di A
```

Queste istruzioni possono sembrare a prima vista un vero rompicapo, ma dopotutto i simboli mnemonici sono abbreviazioni.

"LD" e' l'abbreviazione del verbo inglese LOAD (carica), per cui la scrittura

```
LD A,1
```

sta ad indicare che si vuol caricare il valore 1 in A, il che equivale a dire che si conta fino ad "uno" sulle dita della mano A.

Le parentesi invece servono per specificare l' oggetto su cui si deve operare e distinguerlo dal suo contenitore.

QUANDO VENGONO UTILIZZATE LE PARENTESI, CIO' CHE IN ESSE E' CONTENUTO RAPPRESENTA IL NOME O L'INDIRIZZO DEL CONTENITORE, E NON IL DATO DA ELABORARE.

Ricordare questa convenzione non dovrebbe risultarvi difficile, dato che le parentesi richiamano anche visivamente il concetto di contenitore.

Così', nell' esecuzione del nostro programma il valore 10 viene posto

nel contenitore di nome "SCATOLA1", il valore 5 nel contenitore di nome "SCATOLA2", ed il risultato finale 5 viene caricato nel contenitore di nome "SCATOLA3".

Tutto cio' e' concettualmente molto semplice e sono sicuro che non avrete neppure difficolta' a capire che mentre nel corso di questi calcoli la mano A e' stata utilizzata per rappresentare le ore, un istante dopo le dita della stessa mano potrebbero essere usate per rappresentare il numero di impiegati di una ditta, e poco dopo ancora il numero di monete che avete in tasca.

Se avete avuto occasione di realizzare programmi in BASIC, sicuramente questo discorso vi richiamera' alla mente il concetto di variabile. Fate pero' attenzione: vi e' molta differenza tra le variabili BASIC e le dita delle mani della CPU. Infatti la CPU usa in genere le mani solo per contare.

UNA DELLE GROSSE DIFFERENZE ESISTENTI TRA LA PROGRAMMAZIONE IN BASIC E QUELLA IN LINGUAGGIO MACCHINA CONSISTE PROPRIO NEL MODO DI TRATTARE VARIABILI.

A questo punto vi potrebbe venire pero' il dubbio che i contenitori da noi utilizzati precedentemente possano invece essere paragonati, dando loro un nome, alle variabili di tipo BASIC.

Il ragionamento di per se' e' corretto, ma attenzione: neppure queste sono variabili vere e proprie. Infatti i contenitori possono essere utili per realizzare la funzione tipica delle variabili BASIC, ma dovette sempre ricordare che non sono altro che delle locazioni di memoria utilizzate in quel momento per un particolare scopo.

Ad esempio, il modo di rappresentare i valori negativi e' diverso da quello solito, come vedremo meglio in seguito.

E se la CPU finisce le mani?

Voglio premettere che se per caso vi capitasse di incontrare una CPU per strada la trovereste un tipo alquanto strano.

Essa infatti e' dotata di otto mani con otto dita ciascuna e di due piedi forniti di sedici dita: ma, ciononostante, vedeste come si muove con estrema agilita'!

Disporre di un numero cosi' elevato di arti facilita sicuramente la CPU nell'esecuzione dei calcoli; ciononostante puo' capitare ugualmente che, nel corso di un calcolo, essa si trovi con un numero insufficiente di mani oppure che il programmatore chieda di sospendere temporaneamente l'esecuzione di un programma per far eseguire alla CPU qualcos'altro di piu' urgente.

In entrambi i casi la CPU si trova nella necessita' di conservare il contenuto delle mani e dei piedi per poterlo riutilizzare in seguito, ma non puo' usare a questo scopo i contenitori perche' in tal caso dovrebbe comunque utilizzare alcune mani per ricordarsi in quale scatola ha posto le informazioni.

La CPU Z80 aggira l'ostacolo utilizzando uno "stack", paragonabile a quegli spilloni, spesso presenti sulle scrivanie, in cui vengono infilati appunti, bollette, fatture, ecc.. Sono sicuro che anche voi avrete talvolta utilizzato o visto utilizzare questi oggetti, in cui i vari fogli vengono disposti uno sopra l'altro. Essi risultano di estrema praticita' quando i fogli vengono poi consultati dall'alto verso il basso, ma pensate quale complicazione comporterebbe la ricerca e l'estrazione di un foglio posto in posizione intermedia. In tal caso, infatti, per non rompere i vari fogli sovrastanti quello desiderato sarebbe necessario sfilarli tutti.

Un simile strumento risulta comunque comodo per la CPU, che riutilizza sempre le informazioni contenute sui foglietti nell'ordine inverso a quello in cui sono state memorizzate.

Infatti un'interruzione genera una sospensione dell'attivita' della CPU la quale provvede a porre, uno per uno, i valori rappresentati sulle mani nello "stack" e, terminata la causa dell'interruzione, li preleva nell'ordine inverso, ripristinando sulle dita delle mani la situazione precedente.

In termini informatici chiameremo pila o STACK il supporto di memorizzazione, PUSH l'operazione necessaria per aggiungere un elemento in cima alla pila e POP l'operazione che consente invece di estrarre dalla pila l'elemento di testa.

Ovviamente lo "stack" puo' essere usato per memorizzare svariati tipi di informazione: ad esempio, durante un calcolo complesso la CPU puo' "salvare nello stack" alcuni risultati intermedi per riutilizzarli poi in seguito. In questo caso, per ciascun dato da salvare occorrera' fare un'operazione di PUSH, mentre bisognera' eseguire un'operazione di POP per poterlo riutilizzare in seguito.

Per ragioni note solo ai suoi progettisti, la CPU Z80 utilizza uno "stack" capovolto, attaccato al soffitto anziche' essere posto sulla scrivania. La pila si sviluppa pertanto dall'alto al basso ed i dati vengono quindi inseriti nello spillone dal di sotto.

L'uso dello "stack" come strumento di memorizzazione temporanea di informazioni porta alla CPU un notevole vantaggio in quanto non e' piu' costretta a ricordarsi gli indirizzi dei contenitori in cui le stesse vengono conservate.

Per recuperare le informazioni e' sufficiente infatti prelevarle dalla testa della pila.

Naturalmente e' necessario avere un piccolo segnalatore che identifichi il numero di elementi memorizzati nella pila oppure l'elemento di testa della stessa, per assicurare un controllo sulle operazioni di PUSH e POP.

Che cosa puo' fare la CPU?

E' possibile a questo punto definire il tipo di operazioni che i progettisti hanno previsto per la CPU Z80.

Osserviamo innanzitutto che, poiche' la CPU utilizza per i propri cal-

coli le dita delle mani e dei piedi, essa puo' operare solo con due tipi di dati:

- * quelli rappresentabili con otto dita
- * quelli rappresentabili con sedici dita

Anche se la cosa puo' sembrarvi strana, la CPU puo' operare soltanto con valori che stiano su una o tutt'al piu' su due mani!

Anche le operazioni che essa e' in grado di fare sono molto limitate. La CPU sa solo:

- * rappresentare valori su di una mano
- * rappresentare valori su due mani
- * eseguire addizioni, sottrazioni, incrementi e decrementi su valori rappresentabili su di una mano
- * eseguire addizioni, sottrazioni, incrementi e decrementi su valori rappresentabili su due mani
- * eseguire particolari operazioni, di cui parleremo diffusamente in seguito, su valori rappresentabili su di una mano
- * saltare a richiesta da un punto all'altro del programma
- * cercare di comunicare o ricevere dall'esterno valori rappresentabili su una sola mano.

Sono sicuro che questo insieme di istruzioni vi sembrera' molto limitato e che stenterete a credere che possa essere sufficiente per insegnare alla CPU a giocare a scacchi oppure a calcolare il vostro stipendio. Notiamo a tale proposito che tra le istruzioni elencate non esiste neppure la moltiplicazione. Cio' significa che in linguaggio macchina anche per eseguire una moltiplicazione tra due numeri occorre costruire un programma vero e proprio.

Questo e' il motivo per cui, scrivendo un programma in linguaggio macchina si procede molto piu' lentamente che con il BASIC, in quanto si puo' fare solo un piccolissimo passo per volta.

RIEPILOGO

Registri

La CPU utilizza per i suoi calcoli un certo numero di registri. Tra questi 8 vengono considerati come delle mani e 2 come dei piedi. Ciascuna mano della CPU ha 8 dita mentre ciascun piede ne ha 16.

Locazioni di memoria

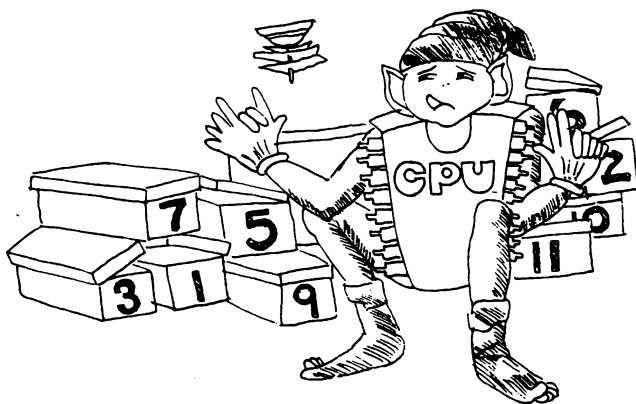
La CPU e' in grado di trasferire informazioni da una mano all'altra, da una mano alla memoria e viceversa. Spetta al programmatore definire di volta in volta la specifica locazione di memoria interessata al trasferimento dell'informazione.

Lo stack

La CPU utilizza lo stack per memorizzare temporaneamente informazioni. L'informazione viene trasferita nello stack mediante un'operazione di PUSH e viene poi prelevata dallo stesso mediante un'operazione di POP.

Set di istruzioni

L'insieme delle operazioni che la CPU e' abilitata ad eseguire e' molto limitato e comprende solo le piu' elementari operazioni aritmetiche ed il trasferimento di dati. Tutti i programmi in linguaggio macchina vengono scritti utilizzando solo queste istruzioni.



L'ARITMETICA DEL CALCOLATORE





Abbiamo gia' detto che la CPU riesce a contare fino a 255 utilizzando solo otto dita. Ma questo come e' possibile, visto che noi possiamo contare solo fino a 10, utilizzando dieci dita?

Non succede sicuramente perche' il computer e' piu' intelligente di noi (e non lo e' davvero), ma per il fatto che la CPU sa organizzare meglio le informazioni: non c'e' infatti alcun motivo per cui, alzando il dito indice si debba rappresentare lo stesso valore (uno) di quando si alza il mignolo.

E' indubbio infatti che, volendo, si possono rappresentare in tal modo due valori diversi: e' un po' come dire che il numero 001 e' diverso da 100. La pura verita' e' dunque che l'uomo non usa nel modo piu' efficiente le dita che ha a disposizione per contare.

La CPU sa trarre valide informazioni non soltanto dal numero di dita alzate, ma anche dalla loro posizione.

Operando in questo modo e' possibile, ad esempio, contare fino a 4 usando solo due dita:

	00 = 0	Abbiamo utilizzato il simbolo 0 per le dita abbassate
	01 = 1	ed il simbolo 1 per quelle alzate
	10 = 2	Abbiamo usato ancora un dito, ma diverso dal precedente
	11 = 3	Abbiamo utilizzato la configurazione con entrambe le dita alzate per rappresentare il valore 3.

Naturalmente, avremmo anche potuto scegliere una rappresentazione diversa per i quattro valori.

Osserviamo comunque che vi e' una stretta relazione tra la rappresentazione da noi scelta e la Rappresentazione Binaria. Le dita della CPU non sono altro che porzioni di memoria e sono state costruite in modo che possano assumere due soli stati: "on" e "off" (o, come si e' soliti dire, 1 e 0).

Se ora aggiungete un terzo dito nel nostro esempio, vi accorgete che potete rappresentare tutti i valori da 0 a 7. Tre dita soltanto per tutti i numeri da 0 a 7!

E con quattro dita si possono addirittura rappresentare tutti i numeri da 0 a 15 ! Se non ci credete, provate a scrivere per esercizio tutte le configurazioni che si possono ottenere con quattro dita, codificando con 0 il dito abbassato e con 1 il dito alzato.

Per semplificare questo tipo di notazione ed evitare la confusione che potrebbe nascere rappresentando numeri di due cifre, adotteremo parti-

colari simboli per indicare i valori compresi tra 10 e 15. Seguendo una convenzione ormai universalmente accettata, utilizzeremo le lettere latine maiuscole da A ad F per rappresentare, come già detto, i numeri da 10 a 15.

Decimale	10 =	A
	11 =	B
	12 =	C
	13 =	D
	14 =	E
	15 =	F

I valori decimali compresi tra 0 e 15, seguendo questa convenzione, vengono quindi rappresentati come segue:

0 1 2 3 4 5 6 7 8 9 A B C D E F

Semplice, no?

Questo modo di rappresentare i numeri prende il nome di NOTAZIONE ESADECIMALE.

Per evitare confusione, molti pospongono al numero esadecimale la lettera "H": ad esempio 10H. La lettera "H" non rappresenta comunque alcun valore, ma serve solo ad indicare che il numero si deve intendere espresso in forma esadecimale.

L'uso della notazione esadecimale presenta notevoli vantaggi nella programmazione in linguaggio macchina in quanto:

1. E' piu' facilmente convertibile nella corrispondente forma binaria, che evidenzia il modo in cui vengono utilizzati i singoli bit (o dita).
2. Fornisce un ottimo strumento per distinguere i valori rappresentabili su di una mano da quelli rappresentabili su due; vale a dire che permette di distinguere i valori a 8 bit da quelli a 16 bit.
3. Permette di rappresentare agevolmente numeri esadecimali di due sole cifre e il calcolatore lavora solo su quelli.
4. E' una notazione standard che ritroverete spesso leggendo libri o manuali di informatica.
5. Poiche' la CPU e' stata costruita in modo da elaborare solo informazioni rappresentabili con numeri binari, che sono di difficile lettura per l'uomo, la notazione esadecimale ci fornisce una rappresentazione sicuramente piu' leggibile.

Questa comunque e' solo una convenzione e non una regola sacra.

Il sistema di numerazione esadecimale permette di rappresentare i numeri da 0 a 15 utilizzando solo 4 bit. Ciascuna locazione di memoria

(formata da 8 bit) e ciascun registro a 8 bit puo' essere allora descritto utilizzando due blocchi di 4 bit ciascuno, allo stesso modo in cui vengono usate due mani di cinque dita ciascuna per contare fino a dieci.

LA RAGIONE CHE CI HA INDOTTO AD ACCENTRARE L' ATTENZIONE SULLE MEMORIE E SUI REGISTRI A 8 BIT RISIEDA NEL FATTO CHE LO SPECTRUM LI HA PROPRIO DI QUESTO TIPO.

Tutte le locazioni di memoria e la maggior parte dei registri dello Spectrum sono infatti formati da 8 bit. Cio' non dovrebbe generare alcuna sorpresa: e' come dire che tutti gli uomini hanno cinque dita per ogni mano.

A questo punto, prima di proseguire, vi proponiamo alcuni esempi che vi consentiranno di familiarizzarvi con questo nuovo modo di contare (due asterischi vicini significano "elevato a").



$$\begin{aligned} 1\ 1\ 1\ 1 &= 2^{**3} + 2^{**2} + 2^{**1} + 2^{**0} \\ &= 8 + 4 + 2 + 1 \\ &= 15 \text{ (notazione decimale)} \\ &= F \text{ (notazione esadecimale)} \end{aligned}$$

Per coloro che non hanno molta dimestichezza con la matematica specifichiamo che, spostandosi da destra a sinistra, il valore rappresentato dal singolo dito viene di volta in volta moltiplicato per 2. Se numeriamo le dita come illustrato nella figura seguente:



possiamo dire che il valore di ciascun dito e' dato dalla potenza ennesima di 2, dove n indica il numero assegnato al dito stesso.

Per semplicita' ci riferiremo ad una mano di 4 dita con il termine manina per distinguerla dalla mano reale, formata da 8 dita.

Esercizio

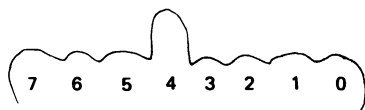
Trovare il valore esadecimale ed il valore decimale corrispondenti alle seguenti configurazioni di bit (o di dita):

Decimale Esadecimale

0010
0110
1001
1010
1100

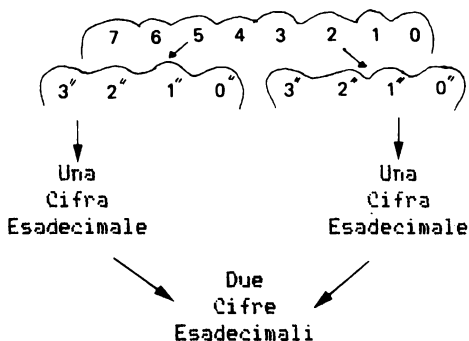
E' molto importante, per andare avanti, che abbiate ben chiari questi concetti e che sappiate usare con una certa disinvoltura la notazione esadecimale. Percio' vi consigliamo, nel caso aveste ancora dei dubbi, di rilegervi le pagine precedenti e di meditarci sopra prima di proseguire.

Esaminiamo ora il caso in cui il numero decimale da rappresentare sia maggiore di 15. Vogliamo provare con 16? Per rappresentare questo valore dobbiamo utilizzare un nuovo dito posto sulla sinistra dei primi quattro:



= 16 decimale = 10H esadecimale

Abbiamo utilizzato la notazione esadecimale 10H in quanto abbiamo pensato la mano suddivisa in "due manine di 4 dita ciascuna". Abbiamo cioe' associato una cifra esadecimale (0-9 & A-F) a ciascuna manina con quattro dita.



La "manina" di sinistra serve a rappresentare valori 16 volte maggiori rispetto a quelli rappresentati su quella di destra. Si opera quindi piu' o meno nello stesso modo in cui si operava con la notazione decimale: la colonna delle "decine" rappresentava valori 10 volte maggiori

di quelli della colonna delle "unita'".

L'esempio mostra l'operazione logica che viene eseguita per passare dalla notazione decimale al suo valore reale:

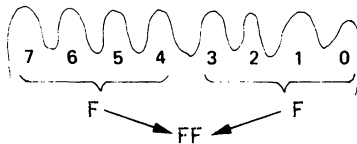
$$15 = (1 \times 10) + 5$$

Solitamente questa operazione e' talmente automatica che non ci accorgiamo neppure di farla.

Utilizzando la notazione esadecimale dobbiamo operare esattamente nello stesso modo. Infatti per convertire un numero esadecimale di due cifre nel corrispondente valore decimale e' sufficiente moltiplicare la prima cifra per 16, come mostra l'esempio seguente:

$$\begin{aligned} 10H &= (1 \times 16) + 0 \\ &= 16 \text{ decimale} \end{aligned}$$

Utilizzando mani di 8 dita possiamo allora rappresentare tutti i valori da 0 a 255. Il valore piu' grande rappresentabile e' infatti:



$$\begin{aligned} FFH &= (F \times 16) + F \\ &= (15 \times 16) + 15 \text{ (in decimale)} \\ &= 255 \text{ decimale} \end{aligned}$$

mentre il valore piu' piccolo e':

$$00H = 0 \text{ decimale}$$

Notiamo ancora una volta che tutti questi valori, dal maggiore al minore, vengono rappresentati usando sempre due e solo due cifre esadecimali.

Provate ora, per esercizio, a rappresentare sia in notazione esadecimale, sia in notazione decimale tutte le combinazioni di 8 cifre binarie (dita).

Questo esercizio, pur essendo molto simile al precedente, vi risultera' utilissimo per la comprensione degli argomenti successivi.

Osserviamo infine che, utilizzando la notazione esadecimale, cambia anche il modo di contare. Infatti contando in decimale o in esadecimale si ha:

Decimale: 26 27 28 29 30 ...

Esadecimale: 26 27 28 29 2A 2B 2C 2D
2E 2F 30 ...

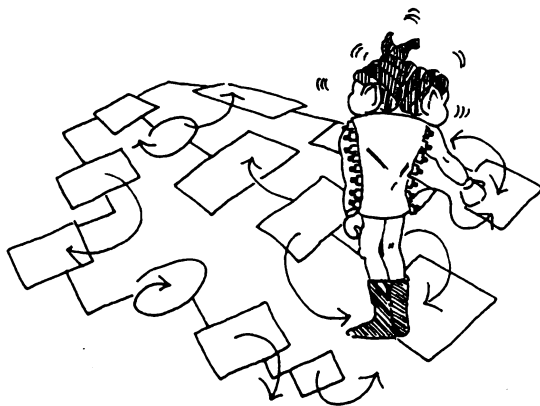
Come potete voi stessi constatare, la serie di valori nelle due rappresentazioni e' totalmente diversa. Nella notazione esadecimale il successivo di 29 e' 2A ed il 30 compare solo cinque posti dopo!

```
100 REM conversione da decimale
110 REM a esadecimale
120 PRINT "inserite un numero i
n formato "decimale": INPUT n:
PRINT n
130 LET s$=""
140 LET n2=INT (n/16): LET n1=I
NT (n-n2*16)
150 LET s$=CHR$ ((n1<=9)*(n1+48
)+(n1>9)*(55+n1))+s$
160 IF n2=0 THEN PRINT "esadeci
male = 0";s$;"H": FOR i=1 TO 200
: NEXT i: RUN
170 LET n=n2: GO TO 140
```

Per concludere, vi presentiamo un programma BASIC, facilmente utilizzabile sul vostro Spectrum, che permette di convertire un valore decimale nella corrispondente notazione esadecimale. Il listato di questo programma, come tutti quelli degli altri programmi Basic presentati nel libro e' stato ottenuto con una stampante SEIKOSHA GP-505.

Provate ora a convertire manualmente i seguenti numeri nel corrispondente valore esadecimale, usando poi il programma BASIC per controllare i risultati ottenuti:

- a. 16484 indirizzo di partenza del "display file" dello Spectrum
- b. 22528 indirizzo di partenza del "file degli attributi" dello Spectrum
- c. 15360 indirizzo di partenza del "character set" dello Spectrum
- d. 15616 indirizzo di partenza dei caratteri ASCII nello Spectrum



RIEPILOGO

Decimale

La notazione decimale e' una convenzione che permette di contare utilizzando gruppi di 10 unita' per volta. Le unita' sono rappresentate con le cifre: 0, 1, 2, 3, 4, 5, 6, 7, 8, e 9.

Esadecimale

La notazione esadecimale e' una convenzione che permette di contare utilizzando gruppi di 16 unita' per volta. Le unita' sono rappresentate dalle cifre: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, ed F. Spesso in coda ad un numero esadecimale viene aggiunta la lettera H per ricordare che stiamo usando questo formato (ad esempio 1800H).

Locazioni di memoria a 8 bit

Il sistema ZX Spectrum e' stato costruito in modo tale che ciascuna locazione di memoria sia formata da 8 bit ("dita"). Una locazione di memoria puo' dunque contenere valori compresi tra 0 e 255. Il valore contenuto in memoria per convenzione e' indicato con la notazione esadecimale utilizzando sempre due cifre.

COME VENGONO RAPPRESENTATE LE INFORMAZIONI

=====

Vi e' un'enorme differenza tra i modi di rappresentare le informazioni dell'uomo e del computer. Un'informazione, per l'uomo, e' per lo piu' composta da numeri e lettere (informazione alfanumerica) mentre in un computer tutte le informazioni sono rappresentate da gruppi di bit. La parola bit deriva dalla fusione delle parole inglesi Binary digIT (cifra binaria); nel microprocessore Z80A questi bit sono organizzati in gruppi di 8, ciascuno dei quali viene denominato BYTE.

Questo modo di rappresentare le informazioni tramite cifre binarie viene definito FORMATO BINARIO e rappresenta la struttura portante dei linguaggi utilizzati dallo Z80 e dagli altri microprocessori.

Fondamentalmente si possono riconoscere due categorie di informazioni rappresentate all'interno di un calcolatore: le ISTRUZIONI ed i DATI. Le prime rappresentano il programma che deve essere eseguito. I dati invece rappresentano gli "oggetti" su cui il programma deve operare.

In particolare, questi ultimi possono essere valori numerici o stringhe di caratteri. Vediamo ora in dettaglio come possono essere rappresentati nel computer i PROGRAMMI, i NUMERI ed i CARATTERI.

Rappresentazione di programmi

Un programma e' una sequenza di istruzioni che la CPU deve eseguire per compiere un certo lavoro, e che puo' essere logicamente suddivisa in piu' parti dette sottoprogrammi.

Nello Z80 tutte le istruzioni sono rappresentate usando uno o piu' bytes. Le istruzioni vengono anche definite tenendo conto del numero di bytes necessari per rappresentarle: si hanno cosi' istruzioni ad 8 bit, a 16 bit e cosi' via.

Poiche' lo Z80 e' un microprocessore ad 8 bit, esso puo' operare solo su un byte per volta. Pertanto, quando un'istruzione ne prevede piu' di uno, i singoli bytes dovranno essere prelevati dalla memoria in tempi successivi. Per questo motivo le istruzioni ad 8 bit richiedono generalmente per l'esecuzione tempi inferiori rispetto alle altre. E' evidente quindi che, scrivendo un programma in linguaggio macchina, sara' buona norma utilizzare il maggior numero possibile di istruzioni ad 8 bit.

Nell'appendice del presente volume potrete trovare l'elenco completo delle istruzioni, suddivise in istruzioni ad un byte ed istruzioni a piu' bytes. Non preoccupatevi comunque eccessivamente, se non avete capito questa sottile differenza, dato che tutte le istruzioni dello Z80 verranno spiegate dettagliatamente in seguito.

Rappresentazione di dati numerici

* Dati di tipo intero

Abbiamo già parlato diffusamente del motivo per cui la CPU non è in grado di operare direttamente su valori decimali del tipo 11.53, ma sa utilizzare solo numeri interi. Non solo; abbiamo anche detto che, dovendo utilizzare mani di 8 dita (vale a dire 8 bit) per contare, la CPU può lavorare solamente con valori compresi tra 0 e 255.

Ad esempio il valore decimale 255, che in notazione esadecimale diventa FFH, verrà rappresentato all'interno del calcolatore nella sua configurazione binaria 1 1 1 1 1 1 1 1.

Ma allora non vi è alcun modo per rappresentare valori negativi?

* Rappresentazione di numeri interi negativi

Ricordiamo che un byte è una mano con 8 dita e che ogni valore viene rappresentato alzando opportunamente alcune di esse.

Ovviamente, se vogliamo rappresentare in notazione binaria dei valori interi negativi dobbiamo trovare il modo di distinguere tra loro i numeri negativi e quelli positivi. Adotteremo la seguente convenzione per rappresentare i numeri con segno:

UN NUMERO SULLA MANO DELLA CPU VIENE CONSIDERATO NEGATIVO SE È
ALZATO IL DITO PIÙ A SINISTRA

(in termini informatici, se il bit più alto, bit numero 7, è 1)

Seguendo però questa convenzione rimangono solo sette dita (bit) per rappresentare il valore del numero e pertanto il numero più alto che potrà essere rappresentato non sarà più 255. Infatti metà dei numeri rappresentabili su di una mano (byte) sarà negativa e l'altra metà positiva. Il segno dipenderà dalla posizione del dito più a sinistra (alzato o abbassato).

Il nuovo insieme di variabilità sarà dunque da -128 a +127. Notiamo a tale proposito che il totale dei numeri rappresentabili rimane ancora 256.

A questo punto però sorge un piccolo problema: quando su una mano il dito più a sinistra (pollice) è alzato, come si può sapere se il numero rappresentato è negativo oppure se è un numero positivo maggiore di 128?

La risposta è molto semplice: scegliete quello che preferite. Il cam-

po di variabilita' puo' essere scelto indifferentemente tra 0 e 255 o tra -128 e +127; pero' non possono essere utilizzati entrambi contemporaneamente. Spettera' al programmatore, e cioe' a voi, decidere quale convenzione adottare in un particolare momento.

Nella scelta di un'opportuna rappresentazione per i valori negativi bisogna pero' tener conto ora di un fatto estremamente importante: tutte le istruzioni devono operare correttamente sia nel caso in cui il programmatore scelga di rappresentare in memoria e nei registri solo valori positivi, sia nel caso in cui la scelta cada invece su numeri relativi.

* Scelta di un'opportuna rappresentazione per i valori negativi

Abbiamo gia' stabilito che un numero negativo sara' caratterizzato da una rappresentazione con il pollice alzato. Ma questo e' sufficiente per scegliere la rappresentazione stessa?

No, perche' non abbiamo ancora deciso quale tra le 128 possibili configurazioni delle sette dita rimanenti verra' utilizzata per rappresentare -1, quale per -2, e cosi' via.

La scelta della rappresentazione deve garantire che, addizionando due numeri opposti, il risultato dia zero. Proviamo per esercizio a cercare la configurazione idonea per il valore -1. Essa dovra' essere un numero binario (con il primo bit uguale ad 1) che addizionato ad 1 dia come risultato zero.

0 0 0 0 0 0 0 1	0 0 0 0 0 0 0 1
1 ? ? ? ? ? ? ?	1 0 0 0 0 0 0 1
-----	-----
0 0 0 0 0 0 0 0	1 0 0 0 0 0 1 0

potrebbe essere questo?=>

Osserviamo che, utilizzando per -1 la stessa configurazione (sugli ultimi sette bit) di +1, si giunge ad un risultato errato in quanto la loro somma non da' zero! Come e' possibile allora trovare due numeri binari diversi da zero, la cui somma dia zero? Il secondo numero, ovviamente, deve essere scelto in modo che converta in zero tutti i bit facendo di volta in volta i riporti.

Attenendovi a questo criterio potreste cercare da soli la configurazione idonea per -1. Vi accorgete cosi' che l'unica configurazione che raggiunge lo scopo e' la seguente:

1 1 1 1 1 1 1 1 (FFH in esadecimale)

Infatti:

	0 0 0 0 0 0 0 1
	1 1 1 1 1 1 1 1

(riporto) (---)	0 0 0 0 0 0 0 0

Ora ci si potrebbe chiedere: esiste una regola generale che permetta di individuare velocemente la configurazione idonea per i valori negativi oppure ogni volta occorre fare questa laboriosa ricerca? La regola effettivamente esiste: basta prendere la configurazione binaria del corrispondente numero positivo, trasformare gli 0 in 1 e viceversa, ed aggiungere al risultato 1.

Applichiamo questa regola per trovare, ad esempio, la configurazione di -3.

```
3=          0 0 0 0 0 0 1 1
scambio    1 1 1 1 1 1 0 0
aggiungo 1 => 1 1 1 1 1 1 0 1      (FDH)
```

Verifichiamo che la somma tra questo numero e 3 dia zero:

```
          0 0 0 0 0 0 1 1
          1 1 1 1 1 1 0 1
          -----
(riporto) (-- 0 0 0 0 0 0 0 0
```

Osserviamo che, operando in questo modo, si ottengono le seguenti configurazioni per i numeri negativi:

```
- 1 =>  FF
- 2 =>  FE
- 3 =>  FD e cosi' via.
```

Il piu' grande numero positivo e'

```
0 1 1 1 1 1 1 1 = 7FH => 127 Decimale
```

ed il corrispondente negativo e'

```
1 0 0 0 0 0 0 1 = 81H => -127 Decimale
```

Questa regola di conversione ha carattere generale e puo' essere utilizzata indifferentemente per passare da valori positivi a valori negativi o viceversa.

Proviamo ad esempio ad applicarla per passare da -3 a 3:

```
Numero      1 1 1 1 1 1 0 1
Opposto    0 0 0 0 0 0 1 0
+1         0 0 0 0 0 0 1 1      => 3
```

Abbiamo dunque trovato un modo corretto per rappresentare i valori negativi e una regola pratica che pone in corrispondenza i valori opposti.

Numeri negativi a 16 bit

Ragionando in modo analogo, possiamo trovare un'opportuna rappresenta-

zione per i valori negativi a 16 bit, quelli rappresentabili cioè su due mani. Logicamente, in questo caso per la determinazione del segno verra' utilizzato soltanto il pollice della prima mano (il bit numero 7 del byte "alto").

Convenzione:

Questo modo di rappresentare i valori negativi viene definito in termini informatici notazione in COMPLEMENTO A DUE. Nell'Appendice troverete la tabella di conversione per i valori decimali negativi.

Ricordiamo ancora che questa e' soltanto una convenzione! Di volta in volta, a seconda delle necessita', sarete voi a decidere se i numeri adoperati saranno compresi tra 0 e 255 oppure tra -128 e +127.

Esercizi

-
- a) se 127 (0 1 1 1 1 1 1 1) e' il piu' grande numero positivo rappresentabile con questa convenzione, quale e' la rappresentazione di -128?
 - b) Trovare l'intervallo di variabilita' per i numeri formati da 16 bit.
 - c) Dopo aver individuato il valore decimale negativo corrispondente a 8000H, trovarne il complemento a 2.

Rappresentazione di dati alfanumerici

Puo' sorgere a volte la necessita' di rappresentare in linguaggio macchina delle informazioni, che non sono ne' istruzioni ne' tantomeno dati numerici, su cui operare. Ad esempio, puo' capitare di dover memorizzare in codice binario i caratteri che compongono il titolo del nostro programma: "IL MIGLIOR PROGRAMMA DEL MONDO".

Per rappresentare i dati alfanumerici adotteremo una convenzione semplicissima: ogni carattere verra' rappresentato in un byte, cioè con una configurazione di otto bit.

Per rappresentare caratteri nel computer vengono utilizzati due codici standard: il codice ASCII ed il codice EBCDIC. Il codice ASCII (American Standard Code for Information Interchange) rappresenta ormai uno standard a livello mondiale per i sistemi a microprocessore, mentre il codice EBCDIC e' una particolare elaborazione del precedente, utilizzata quasi esclusivamente sui computer della IBM.

Nello ZX Spectrum, i codici dei caratteri alfanumerici sono conformi al codice ASCII standard, fatta eccezione per i simboli di Lira sterlina (61H) e di copyright (7FH). Nell' Appendice C e' rappresentata la tabella completa dei codici ASCII utilizzati nello Spectrum.

Provate a dare allo Spectrum il comando: PRINT CHR% 33 e vedrete comparire sullo schermo un punto esclamativo. Il carattere "!" e' infatti rappresentato nel computer dal codice esadecimale 21H.

ATTENZIONE : Abbiamo appena visto che una mano della CPU puo' essere
----- utilizzata indifferentemente per rappresentare svariati
tipi di informazioni. Il contenuto di una mano puo' essere infatti:

- un'istruzione per la CPU
- un numero compreso tra 0 e 255
- un numero compreso tra -128 e +127
- una parte di un valore a 16 bit
- un carattere

E' dunque necessario che il programmatore si ricordi cosa rappresentano di volta in volta le mani della CPU.

RIEPILOGO:

----- Contenuto della memoria

La memoria dello Spectrum puo' contenere indifferentemente istruzioni, numeri e caratteri. Ma e' impossibile risalire al tipo di dato rappresentato, analizzando soltanto il valore binario contenuto in una singola locazione.

----- Programma

Le istruzioni di un programma sono memorizzate in una sequenza di bytes. Alcune istruzioni richiedono un solo byte, mentre altre ne richiedono di piu', fino ad un massimo di quattro.

----- Numeri

Ciascuna locazione di memoria puo' essere utilizzata per rappresentare numeri naturali o numeri relativi. Nei due casi i campi di variabilita' sono rispettivamente tra 0 e 255 e tra -128 e +127.

----- Numeri negativi

Per rappresentare i valori negativi abbiamo adottato una convenzione, riassumibile nel modo seguente:

- Se il bit numero 7 e' posto ad 1 il valore rappresentato e' negativo
- Se il bit numero 7 e' posto a 0 il valore rappresentato e' positivo

Per ottenere l'opposto di un numero e' sufficiente fare il suo complemento a 2 ed aggiungere 1 al risultato.

----- Complemento a 2

Il complemento a 2 di un numero si ottiene cambiando il valore di ogni bit della sua configurazione binaria: ogni 0 diventa 1 e viceversa.

=====

Introduzione

Abbiamo detto che il cervello dello Spectrum, cioè la CPU, è il microprocessore Z80A. Esso è un derivato più veloce del classico microprocessore Z80 brevettato dalla Zilog Inc.

L'unica differenza esistente tra lo Z80 e lo Z80A consiste proprio nella velocità d'esecuzione: il primo infatti opera con una frequenza di clock di 2 Mhz/s (2 Megahertz al secondo), mentre il secondo opera con una frequenza di 3.5 Mhz/s. La "frequenza di clock" rappresenta una misura della velocità con cui opera la CPU. Nello Spectrum quindi vengono generati 3.5 milioni di impulsi di clock al secondo il che equivale a dire che c'è una pulsazione ogni 0.000000286 secondi.

La CPU consuma 4 impulsi di clock per eseguire l'istruzione più veloce e 21 per la più lenta. Pertanto lo Z80 lavora comunque ad una velocità superiore alle 160000 operazioni al secondo!

Un cenno alla struttura fisica della CPU

Il microprocessore è costituito nello Spectrum da un chip di silicio dotato di quaranta piedini di connessione, numerati da 1 a 40. Essi rappresentano le linee di comunicazione tra il microprocessore ed il resto del calcolatore. In particolare, il processore è collegato allo alimentatore attraverso il piedino numero 11, riceve l'impulso di clock dal piedino numero 6, trasmette e riceve indirizzi tramite il gruppo di piedini da 1 a 5 e da 30 a 40, trasmette e riceve dati tramite il gruppo di piedini da 7 a 15, fatta eccezione per il piedino 11, mentre tutti gli altri piedini vengono utilizzati per la trasmissione dei segnali di controllo.

A questo punto vi sentirete certamente molto confusi. Ma non preoccupatevi: per usare il computer non è necessario conoscere fin nei minimi particolari la sua struttura interna e il modo in cui la CPU sfrutta le sue capacità.

La struttura fisica della macchina è per l'utente "trasparente", egli cioè non riesce a vederla. È invece importante conoscere almeno la struttura logica della CPU, nel nostro caso il microprocessore Z80.

La struttura logica della CPU

Lo Z80 è logicamente diviso in cinque blocchi:

- 1) UNITA' DI CONTROLLO
- 2) REGISTRO ISTRUZIONI
- 3) PROGRAM COUNTER
- 4) UNITA' ARITMETICO-LOGICA
- 5) 24 REGISTRI UTENTE (le mani ed i piedi della CPU)

* UNITA' DI CONTROLLO

Possiamo considerare l'UNITA' DI CONTROLLO alla stregua di un SUPERVISORE dell'attivita' della CPU. Il suo compito infatti consiste nello scandire temporalmente e nel coordinare le operazioni (input, output e elaborazione interna) che la CPU compie, seguendo le istruzioni di un programma utente o di sistema.

* REGISTRO DELLE ISTRUZIONI

E' la mano che la CPU usa per ricordarsi le istruzioni da eseguire. Tutte le parti che compongono un programma, cioe' la sequenza delle istruzioni da eseguire, devono risiedere, durante il tempo d'esecuzione, in memoria centrale, sia essa ROM o RAM (Random Access Memory). Quindi l'UNITA' DI CONTROLLO deve provvedere, per compiere il proprio lavoro, a prelevare di volta in volta dalla memoria (ROM o RAM) l'istruzione da eseguire ed a porla in un registro interno detto appunto REGISTRO DELLE ISTRUZIONI.

* PROGRAM COUNTER

E' uno dei piedi dello Z80 che la CPU utilizza per individuare dove si trova il resto del programma che deve essere eseguito: in particolare contiene l'indirizzo di memoria relativo alla prossima istruzione che l'UNITA' DI CONTROLLO deve prelevare.

* UNITA' ARITMETICO-LOGICA

E' la parte della CPU preposta ai calcoli. Essa puo' eseguire sia operazioni aritmetiche, sia operazioni logiche, tutte quelle cioe' che noi definiamo come aritmetiche di base. Questa unita' sa eseguire addizioni e sottrazioni, sa effettuare incrementi (aggiungere 1) e decrementi (togliere 1), ma non moltiplicazioni e divisioni. L'unita' aritmetico-logica sa inoltre effettuare il confronto tra due numeri di 8 bit ed e' in grado di fare alcune particolari operazioni sulle dita di una mano, ad esempio alzare od abbassare un dito specifico, scambiare tra loro le posizioni delle singole dita, e cosi' via. L'unita' aritmetico-logica (ALU) eseguendo tutte queste operazioni modifica anche il valore di certi FLAG, contenuti in un registro apposito chiamato appunto registro dei FLAG, della cui funzione parleremo in seguito.

* REGISTRI UTENTE

Essi sono le mani ed i piedi della CPU che possono essere controllati direttamente dal programmatore. Nello Z80 ci sono ben 24 registri utente, alcuni dei quali sono "mani" mentre altri "piedi".

Le immagini che abbiamo fin qui utilizzato per spiegare le funzioni

dei vari dispositivi di memorizzazione (mani, piedi, ...) sono state utili per capire il modo di operare della CPU, ma qualcuno vi potrebbe guardare male, sentendovi asserire: "... e poi la CPU ha spostato questa informazione dalla mano destra alla sinistra ...".

Pertanto ora vi daremo la terminologia corretta per far riferimento alle mani ed ai piedi della CPU, in modo che, parlando della situazione precedente, vi possiate esprimere così: "LD A,B".

Partiamo, riferendoci alle mani ed ai piedi della CPU con il termine "registri".

Precedentemente abbiamo detto che la CPU ha otto mani, che ora indicheremo con le otto lettere dell'alfabeto: A, B, C, D, E, F, ... e chiameremo registri ad 8 bit.

Abbiamo anche detto che la CPU ha due piedi, che ora indicheremo con le sigle IX ed IY e chiameremo registri a 16 bit.

La corrispondenza tra i nomi usati ed il tipo di registro che rappresentano e' piuttosto facile da ricordare: se il nome del registro e' composto da una sola lettera si riferisce ad una "mano" (registro ad 8 bit), mentre se ne contiene due si riferisce ad un "piede" (registro a 16 bit).

Avete visto come e' stato facile passare dalle dita delle mani e dei piedi ai bit? Probabilmente non sarebbe stato così se avessimo introdotto fin dall'inizio i termini informatici.

Rimangono ora da definire solo due registri ad 8 bit che pero' non chiameremo "G" ed "H" come vi sareste aspettati, bensì "I" ed "L".

L'insieme di tutti i registri della CPU viene convenzionalmente rappresentato nel modo seguente:

!	A	!	F	!
!	B	!	C	!
!	D	!	E	!
!	H	!	L	!
!		IX		!
!		IY		!

Come potete constatare, esclusi i registri A ed H, tutti gli altri ad 8 bit sono accoppiati in modo naturale. Il motivo per cui viene adottato questo tipo di rappresentazione per i registri, consiste nel fat-

to che e' possibile a volte ottenere un piede unendo due mani!
Dopotutto, se un piede e' qualcosa con 16 bit, non vi e' motivo alcuno per non utilizzare due mani da 8 bit al suo posto. Per ottenere questo effetto vengono utilizzati in coppia i registri BC, DE, HL.

La scelta delle lettere H e L anziche' G ed H, come sarebbe stato piu' naturale, non e' casuale. Queste lettere infatti aiutano a ricordare che quando i due registri vengono usati in coppia il registro H contiene la parte piu' significativa (high = alto) del valore a 16 bit ed L quella meno significativa (low = basso).

E' un po' come se volesse rappresentare i numeri tra 0 e 100 utilizzando le vostre dita delle mani e dei piedi. Agevolmente, potete rappresentare i numeri da 0 a 10 utilizzando le dita delle mani; similmente, se foste sufficientemente agili, potreste fare la stessa cosa con le dita dei piedi. Con questo sistema potreste rappresentare il numero 37 contando fino a 3 sulle dita delle mani e fino a 7 sulle dita dei piedi.

E' necessario, comunque, definire dove convenzionalmente viene rappresentata la cifra piu' significativa, per non confondere la rappresentazione di 37 con quella di 73.

Nella coppia "HL" H sta per HIGH ed L per LOW: in questo modo dovrebbe essere evitata qualsiasi possibilita' di confusione!

Il diagramma precedente evidenzia, per ogni coppia, quale registro e' destinato a contenere la parte alta del valore a 16 bit e precisamente

B in BC
D in DE

poiche' in esso l'ordine con cui compaiono i registri rispetta la posizione nella coppia.

I piedi (IX e IY) hanno un altro particolare nome: vengono chiamati anche "registri indice". Tale nome deriva dal fatto che essi possono essere utilizzati anche per organizzare le informazioni, come piu' o meno succede per l'indice di un libro. Potete alternativamente usarli anche come puntatori all'interno di una tabella.

Ora che abbiamo appreso una corretta terminologia, affrontiamo alcuni argomenti particolari:

ACCUMULATORE (registro A)

Questo registro ad 8 bit (un solo byte) e' il piu' importante tra quelli dello Z80. Il suo nome risale al periodo in cui i calcolatori avevano un solo registro, utilizzato per "accumulare" i risultati intermedi.

Pertanto, nonostante vi sia stato un progresso nelle generazioni dei computer, l'accumulatore viene ancora utilizzato, sebbene in modo estensivo, per le operazioni aritmetiche e logiche. Inoltre molti com-

puter vengono progettati in modo tale da permettere l'esecuzione di alcune operazioni solo sul contenuto del registro A.

Questo vale anche per il microprocessore Z80, in cui il registro A e' un registro privilegiato. Volendo potete pensare al registro A come alla mano destra della CPU, e questa, come del resto molti uomini, sa fare piu' cose con la mano destra che con le altre.

Flag:

Avrete gia' notato che non abbiamo mai accennato al fatto di usare in coppia i registri A ed F, per rappresentare valori a 16 bit. Infatti, come succede per il registro A, anche F e' un particolare registro, e prende il nome di registro dei flag. Esso e' una mano ad 8 dita, in cui ciascun dito indica il verificarsi o meno di una determinata condizione. Ad esso dedicheremo piu' avanti un intero capitolo.

Coppia di registri HL

Tra le coppie di registri (BC, DE ed HL) questa e' probabilmente la piu' importante. Infatti, i progettisti dello Z80, oltre a fornire all'utente la possibilita' di utilizzarli singolarmente od in coppia, hanno previsto che certe operazioni aritmetiche su valori a 16 bit possano essere effettuate solo utilizzando questa coppia di registri. Utilizzando la coppia di registri HL, infatti, proprio per questo particolare privilegio di tipo hardware, l'esecuzione delle operazioni su coppie di registri e' piu' veloce.

Registri alternativi

Penso che questo sia il momento opportuno per dire che la CPU si avvale di un insieme di mani di scorta.

In realta' sarebbe meglio parlare di guanti da lavoro, di ricambio (o, utilizzando la terminologia corretta, di un insieme di registri alternativi).

In altre parole, e' come se utilizzaste, per contare, dei guanti di plastica talmente rigida da mantenere la forma datagli dalle dita anche dopo averli sfilati. Così' se dopo aver conteto su di una mano fino a 3 vi togliete il guanto, esso mantiene la stessa forma che aveva la vostra mano, rappresentando il valore 3!

Penso non abbiate alcun dubbio riguardo all'utilita' di questo tipo di guanti: essi infatti permettono, usandone un paio, di prender nota di un certo valore, che verra' conservato cambiando i guanti, lasciandovi peraltro la possibilita' di lavorare con un nuovo paio.

L'altro guanto rimane a vostra disposizione, conservando la forma che aveva la mano quando lo avete tolto. Sfortunatamente pero' non baste-

ra' dargli un'occhiata per riconoscere il numero rappresentatovi, ne' potrete effettuare dei calcoli sul quanto senza averlo prima infilato ancora su una mano.

Per poter quindi riutilizzare le informazioni dovete cambiarvi nuovamente i guanti!

La CPU ha un paio di guanti di ricambio per ogni coppia di mani (ma non per i piedi, del resto quando mai avete visto guanti per piedi?), e ciascun quanto va bene solo per una mano, come del resto per voi il quanto destro o quello sinistro.

La rappresentazione di tutti i registri diventa quindi la seguente:

A - F	<===>	A' - F'
B - C	<===>	B' - C'
D - E	<===>	D' - E'
H - L	<===>	H' - L'

IX

IV

Notate che, mentre per i guanti sono state usate le stesse lettere che definiscono le mani, per quelli di ricambio alle lettere e' stato aggiunto un apice.

Le istruzioni invece fanno sempre riferimento alle mani e non al particolare tipo di guanti usato in quel momento. Pertanto, anche se abbiamo definito i guanti di ricambio con un apice, non possiamo utilizzare istruzioni del tipo LD A',1. La CPU opera sulle mani e non sui guanti!

L'unica istruzione che coinvolge i registri alternativi e' quella che permette alla CPU di "cambiare i guanti". Ad esempio:

1. LD A, (BOX 1) ;Carica in A il contenuto del
;BOX 1
2. EX AF, AF' ;EX e' un'abbreviazione di exchange
;Scambia i guanti AF con quelli AF'
3. LD A, (BOX 2) ;
4. EX AF, AF' ;Nuovo scambio
5. LD A, (BOX 3) ;

Notiamo che tra queste istruzioni non ve ne e' neppure una che operi direttamente sul contenuto dei registri alternativi, eppure il loro valore risulta mutato alla fine della procedura.

Sviluppiamo ora l'esempio per illustrare il modo di operare con questi nuovi registri. Supponiamo per semplicita' che la situazione in memoria sia inizialmente questa:

(BOX 1) = 1
(BOX 2) = 2
(BOX 3) = 3

Seguiamo ora, istruzione dopo istruzione, le variazioni subite dai registri A ed A'.

Lo schema seguente mostra appunto il contenuto dei due registri alla fine di ogni istruzione:

	Registro A	Registro A'
1.	1	Non noto
2.	Non noto	1
3.	2	1
4.	1	2
5.	3	2

Abbastanza semplice, vero?

Questo scambio di registri risulta particolarmente utile quando, durante un'elaborazione, i registri disponibili non sono sufficienti e non e' possibile salvare il loro contenuto ne' nello stack, ne' in memoria. Torneremo comunque ancora su questo argomento.

Vi sono ancora altri registri?

La risposta e' affermativa, ma molto probabilmente nei primi tempi non avrete occasione di utilizzarne altri.

STACK POINTER

Lo STACK POINTER puo' essere considerato come un altro piede ancora della CPU (registro indirizzi a 16 bit). Esso punta sempre all'elemento di testa dello stack. Poiche' lo stack si sviluppa verso il basso, anche lo stack pointer procede dalle locazioni ad indirizzo piu' alto a quelle ad indirizzo piu' basso.

Programmando, in genere, non si opera direttamente sullo stack pointer poiche' la CPU e' in grado di controllare automaticamente il suo contenuto, aggiornandolo ad ogni operazione di PUSH e di POP.

Notiamo solo che nell'uso diretto dello stack si incorre frequentemente nell'errore di dimenticare qualche dato nello stack durante l'operazione di POP e cio' causa sicuramente il fallimento del programma.

Registro I

E' il registro base del Vettore delle Interruzioni. Negli altri siste-

mi basati sullo Z80 questo registro e' usato generalmente per ricordare l'indirizzo base di un vettore di indirizzi, utili per il trattamento delle interruzioni, come ad esempio le richieste di Input/Output.

Nello Spectrum invece quest'uso non e' contemplato, e il registro I e' coinvolto nella generazione dei segnali T.V. , Pertanto e' improbabile che vi troviate nella necessita' di utilizzarlo.

Registro R

Questo registro viene utilizzato per l'operazione di "memory-refresh". Lo Z80 infatti e' predisposto per effettuare in modo automatico il "rinfresco" delle memorie dinamiche. Mentre il processore Z80 lavora, le informazioni contenute in quelle parti della memoria che non sono state recentemente interessate ad operazioni di lettura / scrittura "sbiadiscono", essendo rimaste per troppo tempo senza una goccia di voltaggio. Per questo, non volendo perdere le informazioni in esse contenute, tali locazioni devono essere rinfrescate (ricaricate).

Il registro R e' un semplice contatore che si incrementa ad ogni ciclo di recupero di informazioni in memoria. Il suo valore varia ciclicamente tra 0 e 255.

Questo registro viene utilizzato via hardware per assicurare la corretta esecuzione del rinfresco su tutta la memoria. Non preoccupatevi, comunque, in quanto non siete tenuti a conoscere tutto cio'. Se ne e' preoccupato a sufficienza il Signor Sinclair quando ha progettato lo Spectrum. Noi dobbiamo solo imparare ad usare correttamente questo computer, senza pensare a "rinfreschi" o cose similari.

Dal punto di vista della programmazione il registro R va pensato come qualcosa collegata alla parte hardware ed utilizzata esclusivamente dal sistema. A volte pero' esso puo' risultare utile per la generazione di numeri casuali compresi tra 0 e 255, come vedremo piu' avanti.

Registri Utente

Nella CPU vi sono otto registri principali ad 8 bit (A, B, C, D, E, F, H, L) e due a 16 bit (IX ed IV). I registri ad 8 bit hanno il nome formato da una sola lettera, mentre per quelli a 16 bit se ne usano due.

Coppie di registri

Sei tra gli otto registri ad 8 bit possono talvolta essere accoppiati tra di loro in modo da formare un valore a 16 bit. Si ottengono così le coppie di registri BC, DE ed HL. Il nome HL può servire a ricordare quale tra i due registri contiene il byte più significativo (High byte) e quale quello meno significativo (Low byte).

Registri privilegiati

La CPU Z80 è stata costruita in modo che alcune particolari funzioni ad 8 bit possano operare solo sul registro A. Allo stesso modo vi sono alcune istruzioni a 16 bit che possono essere eseguite solo utilizzando la coppia di registri HL.

Registri alternativi

I registri principali ad 8 bit possono essere scambiati nel corso del programma con un altro insieme di registri, detti appunto registri alternativi.

I valori contenuti nei registri principali vengono così conservati, mentre la CPU per il suo lavoro utilizza i registri alternativi. In questo lasso di tempo, però, i valori salvati non sono accessibili. Per poter operare ancora sui vecchi valori occorre effettuare un nuovo scambio tra i registri.



TUTTO QUESTO E' BELLISSIMO, MA

=====

COME SI ESEGUE UN PROGRAMMA IN LINGUAGGIO MACCHINA?

=====

Probabilmente abbiamo parlato a sufficienza della CPU e della notazione esadecimale, ma non abbiamo ancora spiegato come realmente si lancia l'esecuzione di un programma in linguaggio macchina.

In realta' lo Spectrum esegue programmi in linguaggio macchina in ogni momento (quando e' acceso)! Questa e' la verita', anche se stentate a rendervene conto. Anche quando voi non fate niente e guardate lo schermo cercando qualcosa da inserire come prima linea del vostro rivoluzionario programma BASIC, lo Spectrum e' occupato ad operare sotto il controllo di un programma in linguaggio macchina.

Questo programma, caricato in ROM, e' detto "sistema operativo".

Ad esempio, mentre voi continuate a guardare lo schermo, viene eseguita una parte di questo programma che puo' essere cosi' riassunta:

Scandisci la tastiera di input
Controlla che nessun tasto sia stato premuto
Visualizza l'attuale contenuto del video (vuoto)

La CPU opera sotto il controllo di un programma in linguaggio macchina anche quando lanciate l'esecuzione di un programma BASIC. Abbiamo gia' parlato altrove di questo programma, detto "interprete": esso legge una per volta le istruzioni del programma BASIC, le converte in linguaggio macchina, esegue poi questa parte di programma e ritorna ad analizzare l'istruzione successiva.

Questo non succede piu' quando voi stessi lanciate l'esecuzione di un vostro programma in linguaggio macchina!

Il vostro programma gira in completa autonomia rispetto al sistema operativo! Usando la funzione `USR` il controllo completo della CPU passa al comando che avete caricato all'indirizzo argomento della `USR`. Qualsiasi cosa sia scritta nella locazione indicata, viene interpretata dalla CPU come un'istruzione espressa in linguaggio macchina.

Tutto cio' e' bello, ma anche molto pericoloso: se infatti perdete il controllo del programma, rischiate di perdere anche tutto quello che e' contenuto in memoria. E' sufficiente un errore, un carattere battuto male e sarete costretti a spegnere lo Spectrum e ricominciare tutto da capo.

Non essendovi messaggi che vi segnalino eventuali errori, ne' una preventiva analisi sintattica sull'istruzione, anche l'errore piu' piccolo rende vane tutte le ore di lavoro impiegate per digitare il vostro programma!

Alla fine di questo libro troverete un programma BASIC che vi aiuterà ad introdurre ed a correggere i programmi in linguaggio macchina. Vi consigliamo, dopo aver caricato il vostro programma sullo Spectrum, di salvarlo su di una cassetta, in modo da assicurarvi che, in caso di errore, il danno sia minimo.

D'altronde un simile esperimento non deve impaurirvi, in quanto non potete assolutamente danneggiare il vostro computer. Il peggio che vi possa capitare è di dover spegnere lo Spectrum per riaccenderlo nuovamente.

Vogliamo ora stuzzicarvi l'appetito proponendovi un esperimento con un programma molto semplice.

Caricate il programma BASIC "EZ monitor per codice macchina" che trovate in fondo al libro, e lanciatene l'esecuzione.

Il programma vi chiederà di caricare un indirizzo corrispondente alla posizione in memoria da cui volete che venga memorizzato il vostro codice macchina. Poiché con il programma EZ non si possono usare indirizzi inferiori a 31500, scegliamo come indirizzo di partenza 32000. Battete dunque il numero 32000 ed il tasto <ENTER>.

Sul video apparirà ora la scritta:

Comando o Linea ...:

che sta ad indicare che il programma aspetta l'inserimento di un comando oppure di una linea di programma in linguaggio macchina.

Battete ora di seguito i quattro tasti "1", "SPACE", "c" e "9". In tal modo avete scritto la prima riga del nostro programma in linguaggio macchina, esattamente come succedeva per i programmi BASIC. Se va tutto bene, premete il tasto <ENTER>. Sul video appariranno tutte le linee di programma inserite finora:

1 c9

ed in basso a sinistra la scritta

Comando o Linea ...:

A questo punto non dovete più inserire linee di programma, bensì un comando. Battete pertanto la parola "dump" e il tasto <ENTER>. Con questo comando il programma in codice macchina da voi inserito viene trasferito nell'area di memoria prescelta all'inizio (nel nostro caso a partire dalla locazione 32000).

Congratulazioni: avete memorizzato un'istruzione di un programma in linguaggio macchina! Verifichiamo ora se l'operazione è avvenuta in modo corretto. Battete pertanto il comando "mem" e poi <ENTER>. Questo comando permette all'utente di esaminare il contenuto di una parte della memoria. Il programma vi chiederà dunque l'indirizzo di partenza: battete 32000 <ENTER>.

Vengono così visualizzati i valori contenuti nelle locazioni di memoria che vanno dalla 32000 alla 32087. Potete constatare che in questa area di memoria vi sono tutti 0, eccezion fatta per la locazione 32000 dove vi è il valore esadecimale c9. Per tornare al menu' principale non dovete far altro che battere il tasto "m".

L'istruzione inserita "c9" ha il seguente significato: RETURN!

È un po' come la prima volta che si va in bicicletta: si parte spediti e sciolti nei movimenti, ma si sente il bisogno ben presto di "ritornare" con i piedi per terra (nel nostro caso, di ripassare il controllo al sistema operativo).

Ora facciamo eseguire il nostro programma in linguaggio macchina. Per lanciare il programma, caricato in memoria, battete il comando "run" seguito da <ENTER>.

Che cosa è successo? Perché è comparso un 32000 in fondo al video?

Questo è l'indirizzo che all'inizio avete utilizzato come indirizzo di caricamento.

Non dimentichiamo che la funzione di "USR" lancia l'esecuzione di una subroutine in linguaggio macchina. Questa funzione, tra l'altro, fa sì che il valore di USR al ritorno dal programma in linguaggio macchina da voi memorizzato venga posto uguale a quello contenuto nella coppia di registri BC.

La risposta risiede dunque nel modo in cui opera il sistema operativo dello Spectrum (sì, ancora lui) incontrando la funzione "USR".

Quando il sistema operativo incontra la funzione "USR", carica l'indirizzo da voi specificato (in questo caso 32000) nella coppia di registri BC.

La funzione USR, usata come in

```
Let A = USR 32000
```

ha fornito naturalmente il valore 32000!

Questa caratteristica della funzione USR può divenire un mezzo efficace per controllare cosa succede durante l'esecuzione di un programma in linguaggio macchina.

Provate ora a caricare il seguente programma, in linguaggio macchina:

```
0B  
C9
```

Per il caricamento, operate nel seguente modo:

per inserire la linea 1 battete "1", <SPACE>, "0", "b", <ENTER> e usate poi la stessa tecnica per la seconda riga. Dal listato del programma potrete controllare di aver battuto correttamente le due istruzioni.

Date ora successivamente i due comandi "dump" e "run".

Questa volta la funzione USR restituisce il valore 31999! Cio' e' dovuto al fatto che l'istruzione DB (in Assembler DEC BC) decrementa di 1 il valore contenuto nella coppia di registri BC.

Esercizio

Fate qualche esperimento con istruzioni che operino sulla coppia di registri BC e che potete trovare nell'appendice. In questa ricerca potete operare solo utilizzando le notazioni abbreviate dell'Assembler.

Fate molta attenzione: l'ultima istruzione del programma deve sempre essere "c9"! Essa infatti rappresenta un'istruzione di ritorno al sistema operativo, per cui, dimenticandola, il vostro programma non potra' mai terminare.

Se comunque vi capitasse una cosa simile, nessuna preoccupazione: il vostro computer non ne rimarra' assolutamente danneggiato. Dovrete solo spegnerlo, riaccenderlo e ricaricare tutto.

Esercizio

Provate ad usare il comando "mem" per esaminare una particolare zona di memoria. Scegliete naturalmente degli indirizzi di locazioni in cui pensate di poter trovare qualcosa di interessante.

COME LA CPU USA I PROPRI ARTI (REGISTRI)

Introduzione

Abbiamo visto che tra mani e piedi la CPU puo' contare su 24 arti. Abbiamo inoltre descritto le operazioni che si possono eseguire su questi e come le stesse vengano fatte dalla CPU, dandovi cosi' una chiave per programmare lo Spectrum in linguaggio macchina.

Immaginate ora per un attimo di essere la CPU.

Probabilmente, come la maggior parte delle persone, non siete mancini, quindi vi riesce piu' facile usare le dita della mano destra piuttosto che quelle della mano sinistra. Vi sono poi delle azioni che fate piu' agevolmente in un modo e piu' difficilmente in un altro: ad esempio vi crea maggiori difficolta' prendere qualcosa da uno scaffale piuttosto alto con il piede sinistro, per poi passarla nella mano destra, che non utilizzando al posto del piede la mano sinistra.

La stessa cosa avviene nella programmazione in linguaggio macchina: per realizzare una certa procedura potete trovare un modo agevole, oppure un modo piu' complesso, o addirittura un modo cosi' complicato da renderla inattuabile. Saper combinare in maniera ottimale le varie operazioni rappresenta la chiave del successo.

Nella CPU l'equivalente della vostra mano destra e' l'ACCUMULATORE. Ricordate? L'ACCUMULATORE e' la mano che puo' essere considerata come un'eredita' genetica derivata dai primi computer.

D'altra parte potete temporaneamente memorizzare cio' che avete sulla mano destra su qualsiasi altra mano o piede, e viceversa.

Il linguaggio informatico si riferisce a questo tipo di operazione utilizzando la frase "indirizzamento di registri".

Ma questo e' sicuramente un grosso nome per un'operazione semplice come quella di trasferire un dato da un registro all'altro.

Esempi di questo tipo di operazione sono:

```
LD A, B
LD H, E
```

Pensando che LD e' un'abbreviazione del verbo inglese LOAD (caricare) e che la virgola deve essere letta come se fosse la parola "con" (in inglese with), la frase assembler LD A, B viene letta nel modo seguente:

"LOAD A WITH B" (carica A con B)

Un'istruzione assembler dunque viene letta nello stesso ordine in cui viene letta una normale frase: da sinistra a destra.

Vi sono anche altri modi per indirizzare i registri, cioè per dire che un'informazione deve passare da un registro all'altro o da un registro alla memoria e viceversa.

I modi in cui potete usare gli arti della CPU

Uno dei vantaggi del microprocessore Z80 è rappresentato dal numero elevato di mani e di piedi che mette a disposizione e dagli svariati modi in cui si possono trasferire informazioni tra di loro (metodi di indirizzamento).

I metodi di indirizzamento offerti dallo Z80 sono i seguenti:

- * Indirizzamento immediato
- * Indirizzamento diretto tra registri
- * Indirizzamento indiretto tramite registri
- * Indirizzamento esteso
- * Indirizzamento indicizzato

Che cosa rappresenta questa lista di nomi? Per ora, non preoccupatevi, e consideratela come una confidenza. Riprenderemo qui di seguito il discorso, spiegandovi adeguatamente il significato di ogni tipo di indirizzamento.

Inoltre l'elenco dato precedentemente non ricopre tutte le combinazioni ma soltanto quelle utilizzabili con valori ad 8 bit!

Vediamo ora cosa implica ognuna di esse.

* Indirizzamento immediato

La forma generale per questo tipo di indirizzamento è la seguente:

LD r, n
(oppure un'altra istruzione; abbiamo usato LD come esempio)

Nella notazione sopra, come nelle seguenti, "r" indica un registro ad 8 bit ed "n" un valore binario ad 8 bit.

L'indirizzamento immediato è una tecnica che coinvolge solo una singola mano. Il dato attuale viene definito direttamente nell'istruzione; in tal modo la CPU può eseguirla IMMEDIATAMENTE, non dovendo andare a guardare in memoria per cercare il dato necessario per eseguire l'istruzione stessa.

Per esempio: conta fino a 215 sulla mano "A". Sicuramente conoscete ormai sufficientemente i codici dell'Assembler per poter scrivere que-

sta istruzione in codice mnemonico:

LD A, 215 oppure LD A, D7H

Ricordiamo ancora una volta che in questo tipo di istruzione puo' comparire un qualsiasi registro ed un qualsiasi valore.

Il formato di un'istruzione ad indirizzamento immediato e' dunque il seguente:

byte 1	codice operativo	(dice al computer che cosa deve fare)
byte 2	n	(e' il valore che deve essere usato per l'esecuzione dell'istruzione)

Poiche' per il dato viene usato un solo byte, il campo di variabilita' del valore che voi potete specificare e' dato dall'intervallo 0 - 255. Se cio' non vi e' chiaro, rileggete il capitolo "L'aritmetica del calcolatore".

In genere l'indirizzamento immediato viene usato per inizializzare dei contatori e per definire le costanti utilizzate nei calcoli.

L'indirizzamento immediato si usa agevolmente nella programmazione in linguaggio macchina. Tra tutti i metodi di indirizzamento esso e' pero' il meno flessibile, poiche' richiede che il registro ed il dato siano fissati nel momento in cui si scrive il programma.

In BASIC un'istruzione di questo tipo potrebbe essere:

```
LET A = 5
```

Utilizzeremo ovviamente questo tipo di istruzioni, ma non potremo scrivere solo con esse un intero programma!

L'indirizzamento immediato e' dunque conveniente da utilizzare, ma non risolve la maggior parte dei problemi.

Facciamo comunque un passo alla volta: come programmatori siamo per ora in grado di specificare quale valore vogliamo far caricare in un certo registro.

* Indirizzamento tra registri

Il formato generale di questo metodo di indirizzamento e' il seguente:

```
LD r, r  
(o un'altra istruzione)
```

Questa tecnica coinvolge due mani; brevemente, si tratta di passare un'informazione da una mano all'altra.

La CPU consente il passaggio di informazioni da una mano all'altra, fatta eccezione per la mano F, che abbiamo già detto essere una mano un po' diversa dalle altre, trattandosi del registro dei FLAG: esso non viene usato per memorizzare numeri nel senso normale del termine.

Le istruzioni di indirizzamento tra registri occupano solo un byte.

Istruzioni di questo tipo non sono solo corte (un byte) ma anche veloci. Infatti il tempo necessario per la loro esecuzione è limitato a 4 impulsi di clock, che nello Spectrum corrispondono a meno di un microsecondo.

Nella programmazione in linguaggio macchina è buona "regola" utilizzare questo metodo di indirizzamento (trasferimento di informazioni da registro a registro) ogni volta che risulta possibile, in quanto migliora l'efficienza del programma sia rispetto al tempo d'esecuzione, sia rispetto alla memoria utilizzata.

* Indirizzamento indiretto tramite registri

```
LD (rr), A oppure LD A, (rr)
LD (HL), n
```

Questo potente tipo di istruzione permette di trasferire dati dalla CPU ad una locazione di memoria, puntata dal contenuto di una coppia di registri (piede) e viceversa.

L'indirizzamento indiretto tramite registri è più veloce rispetto a quello indiretto ordinario in quanto la CPU non deve prelevare l'indirizzo dalla memoria.

Occorre però caricare nella coppia di registri prescelta l'indirizzo voluto e quindi tale tipo di indirizzamento risulta vantaggioso solo se nel corso del programma l'indirizzo stesso o qualche indirizzo adiacente vengono utilizzati più volte.

```
Ad esempio,      LD HL,ARCHIVIO ; carica in HL l'indirizzo di
                  ; partenza dell'archivio
CICLO            LD A,(HL)   ; carica il dato
                  INC HL     ; sposta il puntatore
                  continua il CICLO
                  finche' e' finito l'archivio
```

* Indirizzamento indiretto esteso

LD A, (nn) oppure LD (nn),A

Questo e' un altro metodo per trasferire informazioni dalla memoria ad un registro (mano o piede) e viceversa.

Nell'indirizzamento indiretto esteso l'istruzione fornisce direttamente da programma l'indirizzo tramite due bytes.

Se il trasferimento e' da o verso l'accumulatore, l'informazione e' rappresentata solo dal contenuto della locazione individuata dai due bytes di indirizzo.

Se invece il trasferimento interessa una coppia di registri, l'informazione e' rappresentata dal contenuto della locazione di memoria individuata dai due bytes di indirizzo e di quella successiva.

Il formato di questo tipo di istruzione e' il seguente:

byte 1	codice operativo
byte 2	(eventuale codice operativo supplementare)
byte 3	parte bassa dell'indirizzo a 16 bit
byte 4	parte alta dell'indirizzo a 16 bit

Con questo tipo di indirizzamento occorre usare un indirizzo assoluto posto direttamente sul programma; in altre parole, il programma che si ottiene usando questo tipo di istruzione non puo' essere rilocato a meno di rilocare anche l'indirizzo a cui fa riferimento l'istruzione stessa.

Esempio:

```
ARCHIVIO DB n,n,n,... ;archivio di dati
          ;
          ;
          LD A,(ARCHIVIO) ;carica nell'accumulatore il primo
                        ;byte dell'archivio
```

* Indirizzamento indicizzato

LD r, (IX/IY + d) oppure LD (IX/IY + d), r
(od altra istruzione)

Questo tipo di operazione coinvolge uno dei piedi della CPU, vale a dire uno dei due registri indice IX e IY.

La CPU somma al contenuto del registro indice il valore del parametro "d" fornito dall'istruzione, in modo da ottenere l'indirizzo effettivo

del dato.

Questa e' una delle tipiche istruzioni dello Z80 che usano un codice operativo di due bytes. Un'altra istruzione di questo tipo e' la LDIR (load increment and repeat) che serve per il caricamento di un blocco di dati.

L'indirizzamento indicizzato viene generalmente utilizzato nelle operazioni su tabelle o vettori.

I Registri Indice possono essere utilizzati come puntatori all'inizio di una tabella di dati. Il parametro *d*, che compare direttamente nella istruzione, determina la posizione all'interno della tabella a cui si vuole accedere.

```
Esempio: LD IX, INIZIOTABELLA ;posiziona il puntatore
          ;all'inizio della tabella
          LD A, (IX + 3) ;ci si riferisce al terzo byte
          ;dall'inizio della tabella
```

Il formato di questo tipo di istruzioni e' il seguente:

byte 1	codice operativo
byte 2	codice operativo
byte 3	parametro di scostamento "d"

Il numero "d" e' un numero binario di 8 bit che deve essere specificato insieme all'istruzione e non puo' essere una variabile. Cio' significa che con questo metodo si possono indicizzare locazioni il cui indirizzo si scosta da quello posto nel registro indice, di un valore compreso tra -128 e +127.

L'indirizzamento indicizzato e' piuttosto lento poiche' la CPU, per ottenere l'indirizzo effettivo del dato, deve eseguire anche un'addizione. Notiamo infine che esso rappresenta un metodo molto flessibile per indirizzare locazioni di memoria, dato che con la stessa istruzione e' possibile accedere a tutti gli elementi di un vettore o di una tabella.



RIEPILOGO

Per prelevare informazioni di 8 bit o trasferire informazioni dai registri alla memoria, la CPU puo' operare in molti modi.

Indirizzamento immediato

Il valore da assegnare a un certo registro viene definito nel programma.

Indirizzamento tra registri

Trasferimento dati da un registro ad un altro.

Indirizzamento indiretto tramite registri

Una coppia di registri specifica l'indirizzo ed A contiene il valore da trasferire. Se usate i registri HL potete anche definire direttamente nel programma il dato da trasferire.

Indirizzamento indiretto esteso

L'indirizzo viene specificato nel programma ed A e' utilizzato per contenere il dato.

Indirizzamento indicizzato

Attraverso i registri IX e IY viene definita la locazione iniziale di una tabella in memoria, ed ogni registro ad 8 bit puo' essere usato per il dato. La posizione rispetto all'inizio della tabella viene definita da programma. Volendo, si puo' anche definire direttamente nel programma il valore da trasferire in memoria.

Questi modi sono gli unici utilizzabili per trasferire informazioni da ed in memoria; non ne sono permessi altri!

ASSEGNARE VALORI AD UNA MANO

Istruzioni per il caricamento di dati ad 8 bit

Mnemonico	Byte	Tempo	Effetto sui flag						
			Richiesto	C	Z	PV	S	N	H
LD Registro, Registro	1	4	-	-	-	-	-	-	
LD Registro, Numero	2	7	-	-	-	-	-	-	
LD A, (Indirizzo)	3	13	-	-	-	-	-	-	
LD (Indirizzo), A	3	13	-	-	-	-	-	-	
LD Registro, (HL)	1	7	-	-	-	-	-	-	
LD A, (BC)	1	7	-	-	-	-	-	-	
LD A, (DE)	1	7	-	-	-	-	-	-	
LD (HL), Registro	1	7	-	-	-	-	-	-	
LD (BC), A	1	7	-	-	-	-	-	-	
LD (DE), A	1	7	-	-	-	-	-	-	
LD Registro, (IX + d)	3	19	-	-	-	-	-	-	
LD Registro, (IY + d)	3	19	-	-	-	-	-	-	
LD (IX + d), Registro	3	19	-	-	-	-	-	-	
LD (IY + d), Registro	3	19	-	-	-	-	-	-	
LD (HL), Numero	2	10	-	-	-	-	-	-	
LD (IX + d), Numero	4	19	-	-	-	-	-	-	
LD (IY + d), Numero	4	19	-	-	-	-	-	-	

Notazione per i flag:

indica che il flag e' alterato dall'operazione

0 indica che il flag viene posto a 0

1 indica che il flag viene posto ad 1

- indica che il flag non viene alterato dall'operazione

Poiche' nello Spectrum la maggior parte delle attivita' coinvolge registri ad 8 bit o locazioni di memoria (anch'esse a 8 bit), assume ovviamente notevole importanza per la programmazione imparare i modi per assegnare valori alle mani della CPU.

Nel capitolo precedente abbiamo presentato i metodi di trasferimento dati da una mano all'altra. Ci proponiamo ora di analizzarli piu' in dettaglio.

Iniziamo dal metodo che abbiamo definito "indirizzamento tra registri" di cui riportiamo due esempi:

```
LD A,B
LD H,E
```

Ricordiamo che la notazione simbolica LD rappresenta un'abbreviazione del verbo inglese "Load" e che la virgola va interpretata come la preposizione "with". L'istruzione rappresenta quindi l'abbreviazione di una frase inglese che, se ci riferiamo al primo esempio, suonerebbe cosi':

```
load A with B      (carica A con B)
```

Nel secondo esempio la frase Assembler LD H,E va interpretata come "load H with E" (carica H con E).

Questo tipo di istruzione ci permette di trasferire dati da una mano all'altra, fatta eccezione per il registro F (registro dei flag) che non viene considerato un registro come gli altri. L'istruzione presentata puo' essere utilizzata con una coppia qualsiasi di registri. Pensate che e' possibile utilizzare perfino una forma, a prima vista alquanto inutile, del tipo "LD A,A"!

La notazione generale che si usa per questa istruzione e' la seguente:

```
LD r,r
```

dove r indica un qualsiasi registro ad 8 bit, fatta eccezione per il registro F.

Bene: Sappiamo adesso come trasferire informazioni da una mano all'altra ma cio' non ci permette di fare granche' se non ipotizziamo l'esistenza di qualche informazione su quelle mani.

Un secondo metodo per assegnare un valore ad una mano e' quello di dire direttamente alla CPU quale valore volete rappresentare.

Ad esempio, potete dire alla CPU: conta fino a 215 sulla mano "D". Sicuramente conoscete ormai a sufficienza la simbologia dell'Assembler per rappresentare l'istruzione in tal modo:

LD D, D7

(D7 e' la rappresentazione esadecimale di 215).

Ricordiamo che questo tipo di assegnazione viene chiamato indirizzamento immediato (Molto ovvio, no?).

In un registro potete mettere qualsiasi valore, purché compreso nel campo definito per i numeri ad 8 bit, cioè un valore intero qualsivoglia tra 0 e 255.

La notazione generale per questa istruzione e' la seguente:

LD r,n

dove r indica un registro ed n un numero. Per la precedente convenzione, quando si utilizza una sola lettera minuscola si intende che si sta operando su valori ad 8 bit.

Con i due tipi di istruzioni presentati siamo in grado di assegnare ad un registro un valore voluto e di trasferire dati da un registro ad un altro, ma non abbiamo ancora imparato a trasferire dati in memoria ed a portarli dalla memoria ai registri.

Nell'esercizio sulla differenza di orario abbiamo dato un esempio di questo tipo di "indirizzamento esterno" quando abbiamo scritto:

LD A, (BX 3)

La forma generale per questo tipo di istruzione e' la seguente:

LD A, (nn)

Speriamo di non annoiarvi nel ribadire che le parentesi stanno ad indicare la frase: "il contenuto di".

Notiamo due cose su questa istruzione:

1. Può essere usata solo per il registro A
2. Dobbiamo specificare l'indirizzo della locazione mediante un valore a 16 bit (due bytes).

Esiste anche l'istruzione inversa che permette di trasferire il valore di A in memoria:

LD (nn), A

Osserviamo per inciso che nello Z80 tutto il set di istruzioni garantisce questo tipo di simmetria.

Queste istruzioni, come abbiamo già detto, possono essere usate solo con il registro A. Ovviamente vi sono altre istruzioni per altri registri, però non così semplici. Questo è dovuto al fatto che il registro A è privilegiato.

Facciamo per un nanosecondo una pausa e vediamo come possono essere usate queste due istruzioni.

In primo luogo osserviamo che con due bytes possiamo indirizzare le locazioni comprese tra 0 e 65535. Questo numero corrisponde a 64K e pertanto con questa istruzione si può accedere al massimo a 64K di memoria. Ciò significa che tutta la memoria, ROM, programma, display e memoria libera, deve essere contenuta in 64K.

In uno "Spectrum a 16K" vi sono attualmente 16K di ROM e 16K di RAM per un totale complessivo di 32K. Il nome "Spectrum a 16K" fa dunque solo riferimento alla memoria RAM. In uno "Spectrum a 48K" vi sono dunque 16K di ROM e 48K di RAM per un totale di 64K.

Utilizzando lo Z80 non è possibile accedere direttamente ad una memoria più ampia di quella contenuta nello Spectrum a 48K.

L'istruzione "LD A, (nn)", che si legge come "Load A with the contents of location nn" (carica A con il contenuto della locazione nn), è veramente potente. Ci permette infatti di leggere il contenuto di una qualsiasi locazione di memoria sia ROM sia RAM.

Potete usare questa istruzione per esplorare tutto quello che volete, anche una locazione che non esiste realmente in memoria, e cercare di vedere cosa c'è oltre i 32K anche non disponendo della memoria addizionale. E sarete sorpresi nello scoprire che non sono tutti zeri!

L'istruzione inversa, che serve per caricare un valore dall'accumulatore in memoria, è "LD (nn), A". Nel suo uso occorre stare attenti a rispettare le limitazioni fisiche della memoria a disposizione:

non è possibile infatti scrivere qualcosa in una locazione che non può contenerla oppure in una locazione inesistente, in quanto posta al di fuori dei limiti del vostro sistema.

Questo tipo di istruzione presenta una limitazione, consistente nel fatto che occorre sapere a priori, nel momento in cui si scrive il programma, quali saranno le locazioni di memoria interessate ad operazioni di lettura o scrittura. L'abbreviazione nn indica che è necessario definire un numero (ad esempio 17100) da non variare assolutamente nel corso del programma.

Pertanto questa istruzione non è utilizzabile in un ciclo equivalente al Basic "For - Next" e serve solo per relazionare alcune particolari locazioni di memoria, utilizzate per memorizzare valori variabili.

Ad esempio, in un programma tipo l'atterraggio lunare, si potrebbero

utilizzare alcune locazioni nel modo seguente:

32000 : velocita'
32001 : quota
32002 : carburante

progettando percio' un programma in cui si considera il carburante rimasto, se ne diminuisce il valore e si memorizza il nuovo quantitativo nella locazione prescelta. In tal caso sarebbe pero' necessario conoscere gia', al momento di scrivere il programma, la locazione che verra' utilizzata come contenitore per questa informazione.

Ribadiamo ancora un concetto fondamentale: la locazione 32002 non e' una variabile, ma solo una locazione di memoria utilizzata per contenere informazioni.

Scrivendo il vostro programma in linguaggio Assembler potrete descrivere l'operazione di caricamento nel seguente modo:

```
LD A, (Carburante)
```

specificando poi, all'atto della sua conversione in linguaggio macchina, all'interno di questa istruzione l'indirizzo reale, in codice esadecimale, della locazione prescelta per il "carburante".

E se invece non si e' in grado di definire esattamente la locazione di memoria in cui e' contenuta l'informazione? Supponiamo di poterne calcolare solo da programma l'indirizzo e che il risultato di questo calcolo venga poi memorizzato in un registro a 16 bit (IX o IY) od in una coppia di registri (BC, DE, HL).

Dato che in tal caso l'informazione sull'indirizzo e' contenuta in una coppia di registri e non siamo stati in grado di definire "direttamente" l'indirizzo voluto, definiamo questa forma di indirizzamento "indirizzamento indiretto tramite registri".

La forma mnemonica di queste istruzioni e' la seguente:

```
LD r, (HL)  
LD A, (BC)  
LD A, (DE)
```

Con ovvio significato dei termini queste istruzioni vengono interpretate nel modo seguente:

"Carica il registro r con il contenuto della locazione puntata da HL"

"Carica A con il contenuto della locazione puntata da BC"

"Carica A con il contenuto della locazione puntata da DE"

Notiamo che il contenuto della locazione di memoria puntata puo' essere trasferito in un registro qualsivoglia (anche H od L, sebbene questo possa sembrare strano) soltanto usando la coppia HL, mentre, usando le altre coppie BC e DE, l'informazione puo' essere trasferita solo nel registro A.

Cio' e' dovuto al fatto che, tra le coppie di registri, la coppia HL e' privilegiata come lo era il registro A tra i registri ad 8 bit.

In alternativa alla coppia di registri, per puntare una locazione di memoria possiamo utilizzare i registri indice. In tal caso l'istruzione Assembler ha la seguente forma:

```
LD r, (IX + d)
LD r, (IY + d)
```

dove "r" indica un qualsiasi registro e "d" lo "scostamento" rispetto all'indirizzo puntato da IX od IY (Fate attenzione a non confondere "d" che indica uno spostamento con "D" che indica un registro!). Il numero "d" e' un numero ad 8 bit da specificare all'atto della scrittura del programma, che non puo' essere una variabile. A seguito questa grossa limitazione, tale istruzione viene usata quasi esclusivamente per effettuare operazioni di lettura e scrittura in una tabella di dati.

Le istruzioni simmetriche sono le seguenti:

```
LD (IX + d), r
LD (IY + d), r
```

Non preoccupatevi se questo metodo di indirizzamento vi sembra un po' complicato: sara' improbabile che vi capiti la necessita' di usarlo nei pochi programmi che farete all'inizio.

Il chip Z80 usato nei computer Sinclair e' piuttosto versatile e vi permette di combinare tra loro alcuni dei metodi di indirizzamento sopra descritti.

Per esempio potete combinare l'indirizzamento immediato (in cui viene specificato direttamente il valore da caricare) con l'indirizzamento esterno o indiretto (in cui l'indirizzo viene specificato usando una coppia di registri).

Questo nuovo metodo di indirizzamento viene chiamato (sorpresa!) "Indirizzamento Immediato Esterno".

Sfortunatamente questo tipo di indirizzamento puo' essere usato solo con la coppia privilegiata HL. La forma di questa istruzione e' la seguente:

```
LD (HL),n
```

Essa e' molto utile in quanto ci permette di trasferire dati in memoria senza caricarli preventivamente in un registro.

Una combinazione di questo tipo e' permessa anche con i registri indice e da' origine all'"Indirizzamento Immediato Indicizzato".

Quest'ultimo riscontra un uso molto limitato, ed in Assembler si presenta nella forma:

```
LD (IX + d),n
LD (IY + d),n
```

L'uso di queste istruzioni in linguaggio macchina

Vediamo ora qualche esempio pratico sull'uso delle istruzioni di caricamento in linguaggio macchina.

Nei capitoli precedenti abbiamo visto che al ritorno dal vostro programma in linguaggio macchina (cfr funzione USR) viene visualizzato il contenuto della coppia di registri BC.

Carichiamo il seguente programma:

(ricordiamo che, prima di caricare il vostro programma in linguaggio macchina, dovete caricare e lanciare il programma di Editor "EZ Code Machine Monitor" e caricare l'indirizzo 32000)

```
1 0e 00
2 c9
```

Usiamo poi il comando DUMP per trasferire questo codice in memoria. D'ora in poi non ci dilungheremo piu' nella spiegazione del metodo da usare per caricare in memoria un programma in linguaggio macchina e lanciarne l'esecuzione, poiche' cio' non vi darebbe nessuna informazione supplementare, rendendo peraltro pesante il discorso.

Supponiamo quindi che siate riusciti a familiarizzarvi sufficientemente con il programma di editor e con le tabelle di conversione che vi servono per caricare il vostro programma in linguaggio macchina. Ci riferiremo dunque ai programmi da caricare con la notazione seguente:

```
0E 00    LD C,0
C9       RET
```

in cui nella parte sinistra viene presentato il codice oggetto (linguaggio macchina) e nella parte destra la corrispondente istruzione in Assembler. In questo modo viene esplicitato anche il numero di bytes utilizzato da ogni istruzione: ad esempio l'istruzione RET (return) occupa un solo byte, mentre l'istruzione LD C,0 ne utilizza due. Ricordiamo per inciso che un'istruzione puo' occupare anche 4 bytes.

Inoltre presenteremo dei programmi che non fanno riferimento ad una

determinata locazione di partenza (rilocabili), cosicché non ha importanza definire all'atto della stesura del programma l'indirizzo iniziale.

Infine ricordiamo che per caricare il vostro programma potete usare il programma di Editor da noi proposto in questo volume, od in alternativa un qualsiasi altro programma simile che potreste anche esservi costruiti da soli.

Prima di lanciare l'esecuzione del nostro programma (con il comando "run" del programma EZ Code), provate a pensare al risultato che verterà visualizzato.

Ricordiamo che il programma carica nel registro C della coppia BC il valore zero e restituisce poi il controllo al sistema operativo. Sappiamo inoltre che all'inizio nei registri BC viene posto l'indirizzo di partenza del programma, nel nostro caso 32000.

Cosa rispondete? A. 0000
 B. 32000
 C. 31896

Lanciamo ora l'esecuzione del programma. Il risultato è stato quello che vi aspettavate?

Se la vostra risposta non è stata corretta, rileggete il capitolo "L'aritmetica del calcolatore".

Provate ora a caricare ed eseguire il seguente programma:

```
06 00 LD B,0
0E 00 LD C,0
09    RET
```

Dovrebbe essere visualizzato il valore 0, dato che entrambi i registri B e C sono stati posti a zero.

Esercizio:

Potete provare con un po' di immaginazione a scrivere i programmi per caricare in A un certo valore, trasferirlo in L, porre 0 in H e così via.

Esercizio:

Il file degli attributi inizia alla locazione 5800H. È possibile assegnare un valore ad HL in modo che punti al file degli attributi del video nel modo seguente:

```
26 58      LD  H,58H
2E 00      LD  L,0
```

Usando poi il comando LD (HL), n potete cambiare a piacere il colore del video.

La struttura dell' "attribute file" e' descritta nel manuale dello Spectrum. Se ad esempio scegliete per il primo carattere la carta rossa, l'inchiostro bianco e la luminosita' alta, la configurazione binaria di questa scelta e' la seguente:

```
1 0 1 1 1 0 1 0 = BAH
```

e quindi la terza linea di programma sara':

```
36 BA      LD  (HL),BAH
```

Per concludere il nostro programma, aggiungiamo poi l'istruzione necessaria per ripassare il controllo al sistema

```
C9        RET
```

Lanciate ora questo programma. Avete visto come si lavora?

I FLAG E IL LORO USO

=====

I flag sono delle graziose bandierine che possono essere sventolate in occasione delle feste nazionali - errore!

In linguaggio macchina la parola "flag" definisce degli "indicatori". Un flag e' qualcosa che potete usare per indicare a qualcuno che si e' verificata una certa condizione.

Per fare un esempio banale, potete associare il concetto di flag alle bandiere che in marina vengono usate per indicare la nazionalita', il club di appartenenza od altro.

Il motivo che ha spinto i progettisti dello Z80 (e quelli di molte altre CPU) ad introdurre l'uso dei flag nel linguaggio macchina e' stato quello di fornire al programmatore informazioni sullo stato dell'accumulatore (registro A) o sull'ultimo calcolo eseguito.

Ricordiamo a questo proposito che tra i registri della CPU ve ne e' uno, il registro F, dedicato proprio a rappresentare i flag.

All'inizio dell'ultimo capitolo vi abbiamo presentato una tabella riassuntiva delle varie istruzioni di caricamento, e una parte di questa tabella era dedicata proprio ad esplicitare l'effetto che le varie istruzioni avevano sui flag (Fortunatamente nessuna di quelle istruzioni agiva sul valore dei flag).

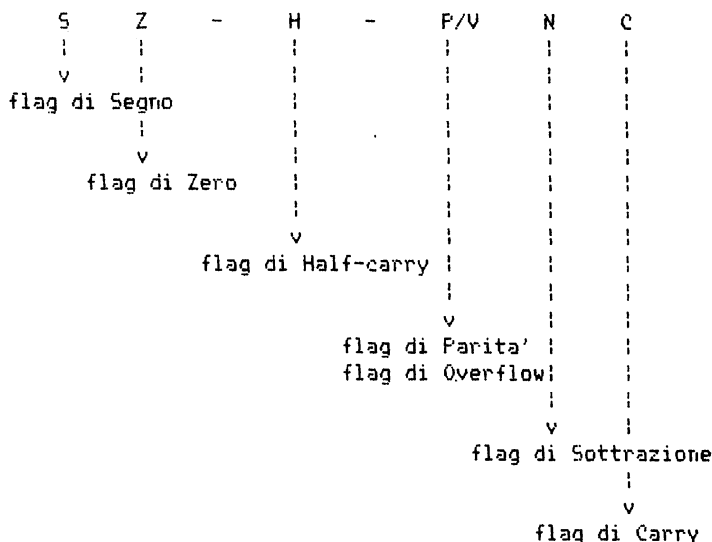
Il flag la cui funzione e' piu' semplice da capire e' il flag di Zero.

Questo flag deve essere "alzato" se il contenuto del registro A e' zero. Cio' e' molto importante, dato che nel corso del programma vengono prese molte decisioni controllando se il valore di A e' zero.

Sottolineiamo che in ogni istante il flag zero e' o alzato o abbassato. Non essendo possibile avere una condizione intermedia (bandiera alzata a meta' per "fare ombra"), per rappresentare questo flag e' sufficiente un solo bit. Cio' vale anche per tutti gli altri flag, i quali in ogni istante sono o alzati o abbassati. E' possibile quindi associare ad ogni flag un singolo bit dell'apposito registro.

I differenti tipi di flag

Il registro F e' un normale registro ad 8 bit e pertanto puo' contenere 8 diversi flag. In pratica pero' i progettisti dello Z80 hanno pensato di utilizzare solo 6 degli 8 bit a disposizione. Nello Z80 vengono infatti usati solo 6 flag:



In realta' i progettisti hanno pensato a ben sette condizioni diverse, ma alla fine hanno preferito rappresentare con un unico flag due condizioni: la parita' e l'overflow.

Vediamo ora in dettaglio il significato di ciascuno di questi flag.

Flag di Zero

Questo e' il flag di cui abbiamo gia' discusso in apertura del capitolo. Il flag di zero ovviamente viene utilizzato per controllare il risultato di un'operazione aritmetica e serve ad indicare se il registro A, alla fine, contiene o no il valore zero.

Notiamo a questo proposito che e' anche possibile avere zero nel registro A senza che cio' comporti di conseguenza che il flag di zero venga posto ad 1 (alzato). Cio' accade se utilizziamo un'istruzione del tipo:

```
LD A,0
```

Abbiamo infatti detto che le istruzioni di caricamento non influenzano il valore dei flag e pertanto l'istruzione LD A,0 NON implica che il flag di zero assuma il valore 1. Esso quindi mantiene il valore che aveva in precedenza.

Il flag di zero e' posto ad 1 anche nel caso in cui il risultato di una delle istruzioni del tipo "rotate" e "shift" porti al valore zero.

Infine notiamo che il flag di zero rappresenta l'unico risultato visi-

bile di alcuni tipi di test, come ad esempio il test sui bit (bit test). In quest'ultimo caso il flag e' posto ad 1 se il bit testato e' a 0 e viceversa.

Flag di Segno

Il flag di segno e' molto simile al flag di zero e viene usato piu' o meno negli stessi casi (fatta eccezione per il gruppo di istruzioni di "bit test" dato che non avrebbe alcun senso parlare di bit di segno).

Flag di Carry

Questo flag e' sicuramente uno dei piu' importanti per la programmazione in linguaggio macchina, dato che senza di esso non si potrebbe dare un senso ai risultati delle operazioni aritmetiche.

E' il momento di ricordare che le istruzioni dell'Assembler fanno sempre riferimento a numeri di 8 bit ed a numeri di 16 bit.

Cio' implica che i valori da utilizzare devono sottostare alle seguenti limitazioni:

8 bit ==> 0 - 255
16 bit ==> 0 - 65535

oppure, includendo il bit di carry,

8 bit ==> 0 - 256
16 bit ==> 0 - 65536

Consideriamo ora la seguente situazione, in cui la sottrazione da' origine ad un "riporto negativo".

200 -
201

Risultato 255 !!!

Il risultato ottenuto e' la diretta conseguenza del fatto che i valori ottenibili devono comunque essere compresi nell'intervallo definito in precedenza. Ovviamente si potrebbe incorrere in situazioni analoghe anche utilizzando numeri a 16 bit.

Abbiamo anche detto che su una mano e' possibile contare solo fino a 255. Che cosa succede allora se in un registro e' contenuto il valore 255 e noi aggiungiamo 1?

Non dovrebbe esservi difficile immaginare il risultato. Pensate al contachilometri della vostra macchina che, raggiunto il valore massimo, ricomincia da zero, registrando eventualmente il completamento del

ciclo.

Nello stesso modo, se il registro o contachilometri e' posizionato a zero e procedete a marcia indietro, vi ritroverete nella condizione precedente, in cui il contachilometri contiene il valore massimo (per i registri ad 8 bit il valore 255).

Ecco il motivo per cui il risultato dell'operazione 200 - 201 e' 255.

Dovendo acquistare un'automobile ci farebbe piacere avere qualche indicazione sull'eventuale completamento del ciclo del contachilometri, dato che, con uno scorrimento in avanti, il valore segnato su di esso sarebbe inferiore al numero effettivo di chilometri percorsi, mentre con uno scorrimento all'indietro il numero di chilometri risulterebbe falsato.

Nel calcolatore un indicatore di questo tipo esiste e, programmando in linguaggio macchina, si fa riferimento ad esso con il nome di flag di carry.

Fortunatamente non abbiamo motivo di preoccuparci se il valore dei registri viene "falsato", tornando indietro. Il flag di carry viene infatti posto ad 1 in seguito ad una sottrazione che ha dato origine ad un fenomeno di "underflow" o ad una addizione che ha originato un "overflow".

E' altresì conveniente pensare al bit di carry come al nono bit del registro A:

Numeri		Bit di carry	Configurazione binaria
132	+	-	1 0 0 0 0 1 0 0
135		-	1 0 0 0 0 1 1 1

267		1	0 0 0 0 1 0 1 1

Furtroppo pero' il registro A non ha nove bit e quindi contiene solo il valore 0BH (11 in decimale). Solo il bit di carry viene settato (posto ad 1) e rappresenta il riporto.

Anche nel caso della sottrazione possiamo immaginare che sia stato preso in prestito un nono bit di valore 1.

L'uso dei flag nel linguaggio macchina e' equivalente

all'uso del costrutto "If ...Then ..." del BASIC

In BASIC possiamo utilizzare il costrutto "IF ... THEN ..." in situazioni del tipo:

If A=0 then ...

ove alla parola chiave then puo' seguire una delle istruzioni:

```
Let ...  
Goto ...  
Gosub ...
```

Lo stesso tipo di istruzione decisionale (fatta eccezione per il "then let ...") puo' essere programmata in linguaggio macchina. Invece di controllare se "A=0" effettueremo un test sul flag di zero: se esso e' settato (posto ad 1) significa che A=0.

I valori dei tre flag che abbiamo considerato rappresentano dei dati, rispetto ai quali puo' venire scelta l'istruzione da eseguire successivamente.

Vediamo ora la forma di ciascuna delle istruzioni condizionali utilizzabili.

Ad esempio:

```
JP cc, End
```

dove JP e' l'abbreviazione di "jump (salta)" ed End e' un conveniente "label", ha il seguente significato:

"salta ad End se la condizione cc e' verificata".

La condizione cc puo' essere una delle seguenti:

```
Z (= Zero)  
NZ (= Non zero)  
P (= Positivo)  
M (= Negativo - minus)  
C (= Carry settato: 1)  
NC (= Carry non settato: 0)
```

Gli altri tre flag, poco usati nella programmazione a livello elementare, sono:

Il flag di Parita'/Overflow

Questo flag viene usato per alcune operazioni come bit di parita' e per altre come bit di overflow, ma raramente si potra' generare confusione, poiche' comunemente i due tipi di condizioni non si verificano assieme.

Come flag di parita', il flag assume significato durante le operazioni

logiche, ed e' settato se nel risultato vi e' un numero pari di bit posti ad 1. Entreremo nei dettagli di questo particolare aspetto del flag quando esamineremo le istruzioni logiche.

L'overflow corrisponde invece ad un avvertimento sul fatto che l'operazione aritmetica eseguita ha dato un risultato che male si adatta ad essere rappresentato con 8 bit. Questo flag non indica che il risultato dell'operazione richiedeva un nono bit, ma piuttosto che l'ottavo bit ha cambiato valore nel corso dell'operazione!

Nell'esempio precedente, sommando 132 e 135, l'ottavo bit era 1 prima dell'operazione, mentre alla fine assume il valore 0, pertanto il flag di overflow viene settato.

Ma lo stesso flag viene settato anche nel seguente caso:

```
  64  +      0 1 0 0  0 0 0 0
  65      0 1 0 0  0 0 0 1
  ---
 129      1 0 0 0  0 0 0 1
```

Il flag di sottrazione

Viene settato se l'ultima operazione e' stata una sottrazione.

Il flag Half-Carry

Questo flag viene settato in condizioni simili a quelle valide per il flag di carry, cioè quando si e' verificata una condizione di overflow o di richiesta di prestito sui primi quattro bit. Esso e' cioè usato come quinto bit anziché come nono bit.

Entrambi i flag, quello di sottrazione e quello di Half-Carry, vengono usati solo nell'aritmetica BCD (Binary coded decimal) a cui in seguito e' dedicato un capitolo.

RIEPILOGO

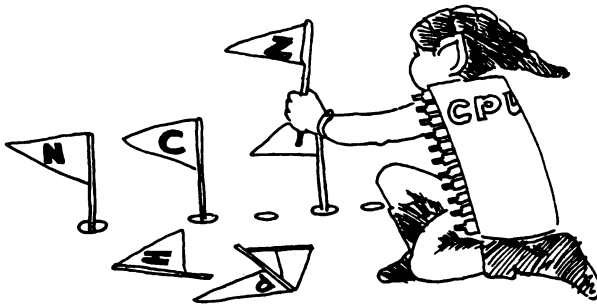
I flag sono usati dalla CPU per indicare certe condizioni verificatesi nell'esecuzione di un'istruzione.

Vi sono 6 flag diversi, ciascuno dei quali puo' essere settato (valore 1) o resettato (valore 0). I bit che rappresentano i flag sono sei degli otto bit del registro F. I due bit rimanenti non sono utilizzati.

Le condizioni indicate dai vari flag sono:

- C - Carry (riporto)
- Z - Zero
- P/V - Parita' od overflow
- S - Segno
- N - Sottrazione
- H - Half Carry (riporto intermedio)

Non tutte le istruzioni agiscono sui flag. Ve ne sono alcune che coinvolgono tutti i flag, altre soltanto alcuni, ed altre ancora neppure uno.



COME CONTARE IN PIU' OD IN MENO SU UNA O DUE MANI

Negli ultimi due capitoli abbiamo potuto esaminare il concetto di flag ed il modo in cui la CPU carica sulle mani e sui piedi i dati.

Vogliamo ora esaminare invece un modo semplicissimo per manipolare questi dati: l'incremento ed il decremento.

Queste due operazioni, benché elementari, costituiscono un passo avanti rispetto al solo caricamento.

L'operazione di incremento risulta infatti essenziale nella programmazione e consiste nell'aggiungere 1 al valore rappresentato.

Questo tipo di operazione può essere usato in situazioni ordinarie come un censimento od il controllo del traffico ad un incrocio.

Incremento

Lo Z80 ha a disposizione un'istruzione che permette di incrementare i valori contenuti nei registri ad 8 bit. La sua forma generale è la seguente:

INC r

ove evidentemente INC rappresenta l'abbreviazione del verbo inglese "increase" (incrementa).

È possibile incrementare anche i registri a 16 bit e le coppie di registri (che, come sappiamo, non sono per la CPU dei veri e propri piedi).

Le istruzioni per eseguire questo tipo di operazioni sono le seguenti:

INC rr
INC IX
INC IY

dove "rr" indica una coppia di registri (BC, DE od HL).

Notiamo ancora una volta che la notazione usata permette di distinguere un registro da 8 bit da uno di 16. Infatti:

i registri o i valori a 8 bit sono denotati con una sola lettera

i registri o i valori a 16 bit sono denotati con due lettere

Ma vi sono delle istruzioni di incremento ancora più potenti di quelle presentate. È infatti possibile incrementare direttamente il contenuto di una locazione di memoria, di cui si conosca l'indirizzo, usando i registri indice oppure la coppia di registri privilegiati HL.

INC (IX+d)
INC (IY+d)
INC (HL)

(attenzione: d e' un parametro di scostamento e non rappresenta il registro D)

Nota importante

Prendiamo spunto dalle istruzioni presentate per ricordare l'importanza delle parentesi:

parentesi ---significa--> "il contenuto della locazione puntata da"

Le due istruzioni

INC HL e
INC (HL)

sono molto simili come forma ma assai diverse come significato. Nel primo caso infatti si "incrementa il valore di HL" mentre nel secondo " si incrementa il contenuto della locazione di memoria il cui indirizzo e' fornito da HL" (o piu' brevemente "la locazione puntata da HL").

Speriamo che, ricordandovi le regole che stanno alla base delle abbreviazioni mnemoniche, riusciate ad evitare una confusione di tale fatta.

Esaminiamo ora il comportamento della CPU nei due casi supponendo che HL contenga il valore 5800H.

INC HL Guarda il valore di HL e lo incrementa di 1. Risultato:

HL = 5801H

INC (HL) Guarda il valore di HL, cerca in memoria la locazione che ha quell'indirizzo e incrementa di 1 il valore in essa contenuto. Risultato:

HL = 5800H
(5800H) = (5800H)+1

Come vedete le due operazioni sono completamente diverse. Notiamo infatti che, mentre INC HL agisce su un valore a 16 bit, l'istruzione INC (HL) agisce su un valore a 8 bit, il numero memorizzato nella locazione 5800H. Ricordando che 5800H e' l'indirizzo iniziale del file degli attributi (vedi manuale dello Spectrum), provate a far eseguire entrambe le istruzioni con piccoli programmi per verificarne l'azione.

Decrementi

Nel set delle istruzioni dello Z80 e' prevista anche l'operazione inversa dell'incremento: il decremento. A ciascuna istruzione INC corrisponde un'identica istruzione DEC, dato che logicamente cio' che si puo' incrementare si puo' anche decrementare.

Le forme generali usate per rappresentare questo nuovo tipo di operazione sono:

```
DEC r
DEC rr
DEC IX
DEC IY
DEC (HL)
DEC (IX+d)
DEC (IY+d)
```

In queste istruzioni DEC rappresenta l'abbreviazione del verbo inglese "decrease" (decrementa); nel loro uso, come abbiamo gia' ripetuto piu' volte, bisogna prestare molta attenzione alle parentesi.

Effetto sui flag

Dato che le istruzioni di incremento e decremento che operano su valori a 8 bit influenzano tutti i flag tranne quello di carry, esaminando l'effetto in dettaglio possiamo effettuare un ripasso sulle funzioni dei flag.

NOTA IMPORTANTE: le istruzioni di incremento e decremento che operano sui registri a 16 bit NON influenzano il valore dei flag, mentre quelle che operano su valori a 8 bit lo influenzano.

- Segno : questo flag e' settato (=1) se il bit piu' significativo
----- numero 7) del risultato dell'operazione e' uguale a 1.
- Zero : questo flag e' settato quando il risultato dell'operazione
----- e' zero.
- Overflow : questo flag e' settato se il valore del bit piu' signifi-
----- cativo viene variato dall'operazione.
- Half-carry: questo flag e' settato se l'istruzione eseguita e' una
----- sottrazione. Pertanto esso assume il valore 0 dopo un INC e 1 dopo un DEC.

Esercizio

Scrivete un programma con una serie di istruzioni LD, INC e DEC e verificate il risultato lanciandone l'esecuzione. Ricordatevi che al termine dell'esecuzione la funzione USR restituisce il valore di BC.

Questo esercizio dovrebbe aiutarvi a prendere dimestichezza con le istruzioni presentate.

RIEPILOGO

Possiamo incrementare o decrementare il contenuto di ciascuno dei registri a 8 bit, di ciascuna coppia di registri e di ciascuno dei registri indice.

Possiamo inoltre incrementare o decrementare il contenuto di una locazione di memoria, il cui indirizzo sia specificato dai registri indice o dalla coppia di registri HL.

Incrementando o decrementando un valore a 16 bit non vengono influenzati i flag, mentre incrementando o decrementando un valore a 8 bit, sia esso contenuto in un registro o in una locazione di memoria, si influenzano tutti i flag tranne quello di carry.



ARITMETICA SU UNA MANO

Istruzioni aritmetiche per numeri di 8 bit

Mnemonico	Byte	Tempo richiesto	Effetto sui flag					
			C	Z	PV	S	N	H
ADD A, registro	1	4	#	#	#	#	0	#
ADD A, numero	2	7	#	#	#	#	0	#
ADD A, (HL)	1	7	#	#	#	#	0	#
ADD A, (IX + d)	3	19	#	#	#	#	0	#
ADD A, ((IY + d)	3	19	#	#	#	#	0	#
ADC A, registro	1	4	#	#	#	#	0	#
ADC A, numero	2	7	#	#	#	#	0	#
ADC A, (HL)	1	7	#	#	#	#	0	#
ADC A, (IX + d)	3	19	#	#	#	#	0	#
ADC A, (IY + d)	3	19	#	#	#	#	0	#
SUB registro	1	4	#	#	#	#	1	#
SUB numero	2	7	#	#	#	#	1	#
SUB (HL)	1	7	#	#	#	#	1	#
SUB (IX + d)	3	19	#	#	#	#	1	#
SUB (IY + d)	3	19	#	#	#	#	1	#
SBC A, registro	1	4	#	#	#	#	1	#
SBC A, numero	2	7	#	#	#	#	1	#
SBC A, (HL)	1	7	#	#	#	#	1	#
SBC A, (IX + d)	3	19	#	#	#	#	1	#
SBC A, (IY + d)	3	19	#	#	#	#	1	#
CP registro	1	4	#	#	#	#	1	#
CP numero	2	7	#	#	#	#	1	#
CP (HL)	1	7	#	#	#	#	1	#
CP (IX + d)	3	19	#	#	#	#	1	#
CP (IY + d)	3	19	#	#	#	#	1	#

Notazione per i flag:

- # indica che il flag viene alterato dall'operazione
- 0 indica che il flag assume il valore 0
- 1 indica che il flag assume il valore 1
- indica che il flag non viene alterato dall'operazione

Abbiamo intitolato questo capitolo "Aritmetica su una mano" proprio per ricordarvi che le operazioni che presenteremo riguarderanno solo numeri a 8 bit e che tali operazioni potranno essere eseguite solo coinvolgendo la mano privilegiata della CPU, il registro A.

Fate dunque attenzione: per effettuare addizioni e sottrazioni potrete usare solo questa mano privilegiata!

Questo fatto e' cosi' implicito nell'uso del linguaggio Assembler dello Z80 che nella forma mnemonica di alcune istruzioni il riferimento al registro A viene sottinteso. Così' ad esempio l'istruzione che permette di sottrarre B da A non ha la forma naturale

SUB A,B

ma la forma abbreviata

SUB B

A parte la limitazione dovuta al fatto che le operazioni aritmetiche possono essere effettuate solo coinvolgendo il registro A, il set di istruzioni aritmetiche nello Z80 e' piuttosto ricco. E' infatti possibile eseguire l'addizione in svariati modi come mostrano le istruzioni seguenti:

ADD A, r	Somma ad A il contenuto di un altro registro a 8 bit
ADD A, n	Somma ad A un numero a 8 bit
ADD A, (HL)	Somma ad A il contenuto della locazione puntata da HL
ADD A, (IX + d)	Somma ad A il contenuto della locazione il cui indirizzo e' dato da IX + d
ADD A, (IY + d)	Somma ad A il contenuto della locazione il cui indirizzo e' dato da IY + d

Come potete vedere la scelta sul secondo operando (il primo e' sempre A) e' piuttosto ampia. Esso puo' essere infatti il contenuto di un altro registro, un numero a 8 bit oppure il contenuto di una locazione di memoria il cui indirizzo puo' essere a sua volta definito in svariati modi.

Manca solo la possibilita' di usare una forma del tipo

ADD A, (nn)

in cui l'indirizzo viene direttamente definito all'interno del programma. Si puo' pero' supplire a questo inconveniente utilizzando due istruzioni del tipo

LD HL, nn
ADD A, (HL)

Notiamo ancora una volta che, tra le coppie di registri, HL risulta privilegiata. Nelle istruzioni aritmetiche infatti per definire l'indirizzo della locazione cercata non possiamo utilizzare le coppie di registri BC e DE ma solo HL.

Un'altra limitazione implicita in questo tipo di istruzione e' quella inerente al fatto che un numero a 8 bit non puo' superare il valore 255.

Così' se ad esempio facciamo:

```
LD A, 80H
ADD A, 81H
```

otteniamo come risultato di avere il valore 1 (uno!) in A. In seguito all'ultima operazione viene settato il flag di carry per indicare che logicamente l'addizione ha dato origine ad un riporto.

Se vi trovate in difficoltà ad operare direttamente con i valori esadecimali potete provare a convertire gli operandi in notazione decimale e verificare che il risultato dell'addizione e' superiore a 255.

Comunque osserviamo che le operazioni di addizione e sottrazione vengono eseguite anche nell'ambito dei valori esadecimali seguendo le regole dell'aritmetica classica.

Contando in esadecimale infatti si ha:

```
1 + 1 = 2
2 + 1 = 3
.....
8 + 1 = 9
```

ed ora attenzione: qual'e' il successivo di 9 ?

```
9 + 1 = A
A + 1 = B
.....
E + 1 = F
```

ed ora attenzione: qual'e' il successivo di F?

```
F + 1 = 10
```

Il risultato mostra che sommando 1 a F si ottiene un "riporto" sulla colonna di sinistra proprio come accadeva sommando 1 a 9 nell'aritmetica decimale.

Il risultato del programma in linguaggio macchina presentato sopra e' giustificato dalla seguente osservazione:

in esadecimale	80 +	
	81 =	

	101	(dato che 8 + 8 = 16 decimale e cioe'
	-----	10 esadecimale)

 Come tener conto del riporto

I progettisti dello Z80 hanno inserito tra le istruzioni valide per la CPU un'istruzione di addizione che tenga conto di un eventuale riporto generatosi nell'operazione precedente. Questa istruzione concettualmente assai simile alla ADD somma al risultato dell'operazione di addizione il valore del flag di carry e per questo viene chiamata "ADDIZIONE CON RIPORTO" e viene indicata con il simbolo ADC (Add with Carry). L'istruzione ADC si presenta nelle stesse forme della ADD e, come abbiamo detto, somma al risultato dell'addizione il flag di carry; pertanto il valore del risultato viene incrementato di 1 nel caso in cui il flag di carry risulti settato al momento dell'operazione.

Questa istruzione permette ad esempio di addizionare tra loro due valori maggiori di 255, concatenando tra loro una serie di addizioni successive.

Il programma seguente mostra come si deve operare per sommare tra loro i due valori 1000 (03E8H) e 2000 (07D0H) e memorizzare il risultato in BC.

```

LD A, E8H           ;carica in A la parte bassa del primo add-
                   ;dendo
ADD A, D0H          ;somma la parte bassa del secondo addendo
LD C, A             ;memorizza il risultato in C
LD A, 03H           ;carica in A la parte alta del primo ad-
                   ;dendo
ADC A, 07H          ;somma la parte alta del secondo addendo
LD B, A             ;memorizza il risultato in C

```

Dopo la prima addizione (E8H+D0H) il flag di carry risulta settato (=1) dato che il risultato dell'operazione superava FFH e il registro A contiene il valore B8 (potete fare la verifica eseguendo voi stessi l'operazione).

Il risultato della seconda addizione non e', come sarebbe ovvio aspettarsi, 0AH (10 decimale) ma bensì 0BH (11 decimale) dato che nell'operazione si e' tenuto conto anche del valore 1 del flag di carry.

Il risultato finale e' dunque 0BB8H che corrisponde proprio a 3000 nella notazione decimale !!

Questo tipo di concatenazione puo' essere usato per sommare numeri anche molto grandi, purché ci si preoccupi di memorizzare il risultato in piu' locazioni di memoria anziché in una coppia di registri.

Sottrazioni

Anche per la sottrazione esistono due tipi di istruzioni: uno per le sottrazioni ordinarie e l'altro per le sottrazioni con riporto.

```
SUB s      ; sottrae s al contenuto di A
SBC s      ; sottrae s al contenuto di A, tenendo
              conto di un eventuale riporto prece-
              dente
```

Il simbolo s e' stato utilizzato per indicare il secondo operando che puo' essere indicato in una qualsiasi delle forme gia' usate per l'addizione.

CONFRONTO TRA DUE NUMERI DI 8 BIT

Sospendiamo per un momento l'analisi del linguaggio macchina e consideriamo cosa si intende esattamente per confronto tra due numeri. Esaminiamo ad esempio che cosa significa confrontare due valori per vedere se sono "uguali". Da un punto di vista matematico dire che due numeri sono uguali equivale a dire che la loro differenza e' zero.

Possiamo verificare se un numero e' maggiore di un altro, operando in tal modo? (Ricordiamo per inciso che un confronto implica sempre una relazione tra due numeri di cui uno e' posto su una delle mani) Si'! Diciamo allora che il secondo numero e' maggiore del primo se il risultato della sottrazione e' un numero negativo. Analogamente, il secondo numero e' minore del primo se il risultato della sottrazione e' positivo.

Usiamo ora questi concetti per realizzare un sistema che ci permetta di confrontare due numeri in linguaggio macchina. In questa operazione ovviamente verranno chiamati in gioco tutti i flag e le istruzioni di sottrazione.

Supponiamo ad esempio di voler confrontare un certo numero con il valore 5. Operiamo quindi nel seguente modo:

```
LD A,5      ;valore noto
SUB N       ;numero che deve essere confrontato
```

ed analizziamo il risultato.

Se $N = 5$ il flag di zero e' settato mentre non lo e' il flag di carry

Se $N < 5$ non sono settati ne' il flag di zero ne' il flag di carry

Se $N > 5$ e' settato il flag di carry mentre non e' settato quello di zero

E' dunque chiaro che il set sull'uguaglianza viene fatto utilizzando il flag di zero, mentre quelli sulla disuguaglianza utilizzando anche il flag di carry. Osserviamo a questo proposito che, mentre per l'espressione "maggiore di" occorre testare entrambi i flag, per quella del tipo "minore di" e' sufficiente un test sul flag di carry.

L'unico inconveniente consiste nel fatto che il contenuto del registro A viene alterato dalla sottrazione.

Fortunatamente pero' vi e' un modo per evitare cio': usare l'istruzione CP ("compare", cioe' confronta), tramite la quale possiamo confrontare il contenuto del registro A con un altro valore, reperibile con uno qualsiasi dei metodi indicati per l'addizione.

"Confronta" viene interpretata dalla CPU come un'istruzione di sottrazione, con la sola differenza che in seguito a quest'operazione il contenuto del registro A resta invariato. Questo tipo di operazione ha dunque effetto solo sui flag.

RIEPILOGO

Le operazioni aritmetiche su valori ad 8 bit, ammesse sullo Z80, sono le seguenti:

- addizione
- sottrazione
- confronto

e possono essere eseguite solo attraverso il registro A.

A parte questa limitazione, lo Z80 offre una vasta gamma di metodi per indirizzare il secondo operando.

Data la natura dei valori ad 8 bit, nel corso di un'operazione si puo' verificare una condizione di overflow. Poiche' il flag di carry (come del resto tutti gli altri) viene alterato dal risultato delle operazioni aritmetiche, possiamo utilizzare in particolare questo flag come segnalatore di overflow.

Vi sono infine delle istruzioni supplementari (addizione e sottrazione con riporto) che permettono di concatenare delle operazioni aritmetiche tenendo conto degli eventuali riporti.

OPERATORI LOGICI

=====

Istruzioni per le operazioni logiche

Mnemonico	Bytes	Tempo Richiesto	Effetto sui flag					
			C	Z	PV	S	N	H
AND Registro	1	4	0	#	#	#	0	1
AND Numero	2	7	0	#	#	#	0	1
AND (HL)	1	7	0	#	#	#	0	1
AND (IX + d)	3	19	0	#	#	#	0	1
AND (IY + d)	3	19	0	#	#	#	0	1
OR Registro	1	4	0	#	#	#	0	0
OR Numero	2	7	0	#	#	#	0	0
OR (HL)	1	7	0	#	#	#	0	0
OR (IX + d)	3	19	0	#	#	#	0	0
OR (IY + d)	3	19	0	#	#	#	0	0
XOR Registro	1	4	0	#	#	#	0	0
XOR Numero	2	7	0	#	#	#	0	0
XOR (HL)	1	7	0	#	#	#	0	0
XOR (IX + d)	3	19	0	#	#	#	0	0
XOR (IY + d)	3	19	0	#	#	#	0	0

Notazioni per i flag:

- # indica che il flag e' alterato dall'operazione
- 0 indica che il flag e' posto a 0
- 1 indica che il flag e' posto a 1
- indica che il flag non e' alterato dall'operazione

Vi sono tre operazioni logiche che nella programmazione in linguaggio macchina (o Assembler) vengono usate con la stessa frequenza con cui vengono usate nell'aritmetica elementare addizione, sottrazione, moltiplicazione e divisione.

Gli operatori che danno origine a queste operazioni vengono generalmente detti operatori Booleani, dal nome dello studioso che ha formulato le regole per le operazioni stesse. Questi operatori sono:

AND
OR
XOR

Finora abbiamo visto operazioni abbastanza familiari, operanti su numeri di 8 bit. Queste nuove operazioni, molto importanti, operano invece sui singoli bit di un numero (o dita della mano della CPU).

Vediamo ora come lavora l'operatore AND:

Bit A	Bit B	Bit A "AND" Bit B
0	0	0
1	0	0
0	1	0
1	1	1

Notiamo che l'operazione di AND restituisce il valore 1 se e solo se i due bit A e B hanno entrambi il valore 1.

In linguaggio macchina, facendo l'operazione AND tra due numeri il risultato si ottiene applicando l'operatore AND a ciascuna delle coppie di bit corrispondenti dei numeri stessi.

Vi potreste chiedere: "Quale e' l'utilita' di questa operazione?"

L'operazione AND e' estremamente utile in quanto ci permette di "mascherare" un byte in modo che venga posto a zero il valore di certi bit.

Se, ad esempio, vogliamo limitare una certa variabile ad assumere solo valori contenuti nell'intervallo 0 - 7, dobbiamo in qualche modo indicare che soltanto i bit 0, 1 e 2 possono assumere valore 1. Infatti, se il bit numero 3 fosse settato, il valore contenuto dalla variabile sarebbe maggiore o uguale ad 8.

0 0 0 0 0 1 0 1 =5
<----->

Questi bit devono essere "0".

Se consideriamo ora un generico valore ad 8 bit ed applichiamo ad esso l'operatore AND con il valore "7", il risultato e' un valore compreso tra 0 e 7.

0 1 1 0 1 0 0 1 = 105
0 0 0 0 0 1 1 1 = 7 => Maschera

risultato di AND 0 0 0 0 0 0 0 1 = 1 => compreso
tra 0 e 7

Notiamo che, anche per l'operazione di AND, lo Z80 prevede l'uso del registro A. Il secondo operando puo' essere un numero ad 8 bit, un altro registro oppure una locazione puntata da HL o da uno dei due registri indice.

```

AND 7      Osserviamo che siccome il primo
AND E      operando e' sempre A, esso viene
AND (HL)   sottinteso nell'istruzione.

```

Le osservazioni fatte sono ugualmente valide per le altre due operazioni Booleane di OR e XOR.

La prima, l'operazione di OR, e' assai simile, concettualmente, a quella di AND:

Bit A	Bit B	Bit A "OR" Bit B
0	0	0
0	1	1
1	0	1
1	1	1

Essa fornisce il valore 1 quando almeno uno dei due bit in esame e' uguale ad 1.

Qui potreste nuovamente interrogarvi sull'utilita' di un'operazione di questo tipo.

L'operazione di OR e' anch'essa molto importante in quanto ci permette di settare alcuni bit in un numero. Se ad esempio vogliamo assicurarci che un numero diventi dispari, dobbiamo far si' che il suo bit meno significativo abbia valore 1 (questo risultato si potrebbe ottenere facilmente usando l'istruzione "SET").

Utilizzando l'operatore OR si puo' fare:

```

LD   A,numero
OR   1           ;il numero e' sicuramente dispari

```

Le due istruzioni precedenti rappresentano una tipica struttura assembler.

Il concetto base dell'operazione di XOR ("Or esclusivo") e' di facile comprensione, tuttavia l'uso di questo operatore nella programmazione e' piu' limitato.

L'operazione di XOR restituisce il valore 1 solo se uno ed uno solo dei bit in esame e' uguale ad 1.

In altri termini, il risultato di uno XOR coincide con quello dell'OR tranne nel caso in cui entrambi i bit siano 1.

XOR => OR - AND

Bit A	Bit B	Bit A "XOR" Bit B
0	0	0
0	1	1
1	0	1
1	1	0

Vediamo ora quali effetti hanno queste operazioni sui flag.

Flag di zero	Questo flag assume il valore 1 se il risultato dell'operazione e' zero
Flag di segno	Questo flag assume il valore 1 se il bit numero 7 del risultato ha valore 1
Flag di carry	Questo flag assume il valore 0 dopo ogni operazione di AND, OR e XOR.
Flag di parita' (Ricordiamo che questo flag e' anche usato come flag di overflow)	Questo flag assume il valore 1 se il numero di bit settati nel risultato e' pari. Esempi: 0 1 1 0 1 1 1 0 => flag di parita': 0 0 1 1 0 1 0 1 0 => flag di parita': 1
Flag di Half-Carry (Entrambi questi flag assumono valori fissi dopo ogni operazione di AND, OR e XOR.
Flag di sottrazione(Questi flag sono usati quasi esclusivamente nell'aritmetica BCD.

Usò degli operatori booleani per operare sui flag

Vi sono alcuni casi particolari in cui, cosa veramente strana, entrambi gli operandi corrispondono al registro A. Vediamo cosa succede:

AND A	A non e' cambiato, il flag di carry diventa zero
OR A	A non e' cambiato, il flag di carry diventa zero
XOR A	A e' uguale a 0, il flag di carry diventa zero.

Queste istruzioni sono usate frequentemente, dato che richiedono un solo byte laddove istruzioni equivalenti ne richiedono due.

Così' ad esempio, per azzerare l'accumulatore e' piu' conveniente usare XOR A anzicché' LD A,0.

E' richiesto di frequente anche l'azzeramento del carry (ad esempio nel caso di routines in cui vengono usate operazioni del tipo ADC -addizione con riporto - ed SBC - sottrazione con riporto -) e cio' puo' essere ottenuto utilizzando l'istruzione AND A che non altera il contenuto dei registri.

RIEPILOGO

I tre operatori logici piu' usati nella programmazione in linguaggio macchina sono:

AND
OR
XOR

Essi possono essere applicati solo a valori di 8 bit ed uno dei due operandi deve essere contenuto nel registro A. Il risultato, alla fine dell'operazione e' posto nel registro A.

Notiamo che il significato dell'operazione di AND dell'Assembler e' diverso da quello di AND del BASIC.

Gli operatori logici esaminano i singoli bit di ciascuno dei due operandi e per questo vengono spesso usati per ottenere particolari "mascheramenti" dei valori in esame o per settare uno o piu' bit.



PRECISAZIONI SUI NUMERI A 16 BIT

Finora abbiamo illustrato solo istruzioni operanti su numeri a 8 bit, ma abbiamo spesso discusso anche sul fatto che la CPU in qualche caso si vede costretta a manipolare valori a 16 bit, come ad esempio quando fa riferimento ai registri indice.

Questi "piedi" infatti sono dotati di 16 dita (16 bit) e possono contenere solo valori a 16 bit.

Abbiamo inoltre detto che vi e' anche un altro modo per memorizzare i numeri a 16 bit: utilizzare delle coppie di mani (registri). Le coppie di registri utilizzabili a questo scopo sono: BC, DE, HL.

La CPU maneggia i valori a 16 bit in modo analogo a quello in cui voi maneggiate oggetti molto pesanti: opera con due mani, e dato che non si trova certo a suo agio a manipolare oggetti simili, le operazioni che puo' compiere sono lente e limitate.

Vogliamo ora esaminare in dettaglio i vari metodi di indirizzamento (possibili contorsioni?) applicabili ai valori a 16 bit.

Indirizzamento immediato esteso

LD rr, nn
(o altra istruzione)

E' l'equivalente dell'indirizzamento immediato descritto per i valori a 8 bit.

Come tutte le istruzioni che coinvolgono numeri a 16 bit, e' piu' lunga della corrispondente istruzione per valori da 8 bit. Infatti, mentre per l'indirizzamento immediato occorreavano due bytes (uno per il codice operativo, l'altro per il numero), ora ne occorrono 3, come mostra lo schema seguente:

Byte 1	Codice operativo
Byte 2	Parte bassa del valore a 16 bit
Byte 3	Parte alta del valore a 16 bit

Questo metodo di indirizzamento viene usato per assegnare un valore (ad esempio un puntatore alla memoria) ad una coppia di registri.

Indirizzamento tra registri

Ricordiamo che in questo caso si tratta di istruzioni che manipolano dati gia' contenuti in uno dei registri.

Nel caso dei valori a 16 bit il numero delle istruzioni di questo tipo presenti nel repertorio della CPU e' piuttosto limitato. Esse sono prevalentemente di tipo aritmetico e possono essere riferite ad un numero molto limitato di coppie di registri.

per esempio: ADD HL, BC

Ricordiamo ancora una volta che la coppia di registri HL e' quella preferita dalla CPU. Questa infatti e' la coppia di mani piu' muscolose e molte istruzioni possono essere eseguite solo usando queste. Cio' e' particolarmente vero per quanto riguarda le istruzioni aritmetiche, come vedremo in dettaglio nel prossimo capitolo.

Indirizzamento indiretto tramite registri

Come abbiamo gia' visto in precedenza, questo tipo di indirizzamento viene usato per istruzioni in cui il valore da manipolare e' in memoria e l'indirizzo e' invece contenuto in una coppia di registri.

Nello Z80 esso fa spesso riferimento alla coppia di registri HL.

Un esempio di istruzione di questo tipo e' il seguente:

 JP (HL)

Indirizzamento esteso

Questo tipo di indirizzamento e' concettualmente simile a quello da noi chiamato indirizzamento indiretto tramite registri, solo che ora il valore da manipolare non e' piu' contenuto in una coppia di registri, ma in due locazioni di memoria.

Per esempio LD HL, (nn)

dove nn deve essere specificato nel programma.

Esercizio

Usando l'EZ Code, caricare i seguenti programmi:

1. Indirizzamento immediato esteso

010F00	LD BC,15	carica in BC il valore 15
C9	RET	return

Alla fine dell'esecuzione di questo programma la funzione USR dovrebbe restituirvi il valore 15.

Da notare che l'uso di questo tipo di indirizzamento e' piuttosto limitato poiche' richiede la definizione del valore direttamente nel programma.

2. Indirizzamento tra registri

Aggiungiamo ora due linee al programma precedente:

```
210040    LD  HL, 4000H    ;carica HL con 16384
010F00    LD  BC, 15      ;carica BC con 15
09        ADD HL, BC     ;somma i due numeri
C9        RET           ;return
```

Lanciando il programma avrete una sorpresa: verra' visualizzato lo stesso valore di prima, e cioe' 15! Ma allora non e' stata fatta l'addizione?

Si', pero' voi non potete accorgervene, dato che il risultato e' rimasto memorizzato nella coppia di registri HL, il cui contenuto non viene visualizzato dalla funzione USR! Per ottenere la visualizzazione del risultato dobbiamo aggiungere al nostro programma alcune linee.

3. Indirizzamento esteso

```
210040    LD  HL, 4000H
010F00    LD  BC, 15
09        ADD HL, BC
22647D    LD  (7D64H), HL ;poni HL in 32100 e 32101
ED4B647D  LD  BC,(7D64H)  ;carica in BC il contenuto delle
                                ;locazioni 32100 e 32101
C9        RET
```

Il metodo utilizzato per trasferire un'informazione da HL a BC non e' il piu' efficiente, dato che si poteva ottenere lo stesso risultato con le istruzioni di PUSH e POP, ma mostra il tipo di difficolta' che si incontra nella programmazione, a causa del numero limitato di metodi di indirizzamento per valori di 16 bit presenti nello Z80.

Potete poi esaminare con il comando "mem" il contenuto delle locazioni di memoria 32100 e 32101 per completare la verifica della correttezza del programma.

COME MANIPOLARE I NUMERI CON DUE MANI

Istruzioni per caricare valori su due mani

Mnemonico	Bytes	Tempo richiesto	Effetto sui flag					
			C	Z	PV	S	N	H
LD Coppia reg. numero	3/4	10	-	-	-	-	-	-
LD IX, numero	4	14	-	-	-	-	-	-
LD IY, numero	4	14	-	-	-	-	-	-
LD (indirizzo), BC o DE	4	20	-	-	-	-	-	-
LD (indirizzo), HL	3	16	-	-	-	-	-	-
LD (indirizzo), IX	4	20	-	-	-	-	-	-
LD (indirizzo), IY	4	20	-	-	-	-	-	-
LD BC o DE, (indirizzo)	4	20	-	-	-	-	-	-
LD HL, (indirizzo)	3	16	-	-	-	-	-	-
LD IX, (indirizzo)	4	20	-	-	-	-	-	-
LD IY, (indirizzo)	4	20	-	-	-	-	-	-

Notazioni per i flag:

- # indica che il flag e' alterato dall'operazione
- 0 indica che il flag e' posto a 0
- 1 indica che il flag e' posto a 1
- indica che il flag non e' alterato dall'operazione

Nei precedenti capitoli abbiamo visto con quanta agilita' la CPU manipola numeri su una mano ed abbiamo presentato il modo in cui maneggia i numeri a 16 bit.

L'abilita' matematica della CPU e' tale che riesce ad effettuare calcoli veramente complessi su numeri anche molto grandi, usando una sola mano. Perche' allora complicare le cose con i numeri a 16 bit?

Perche' vi possono essere casi in cui risulta impossibile specificare cio' che ci occorre utilizzando solo numeri ad 8 bit. Il vostro computer sarebbe una macchina veramente limitata se vi obbligasse a lavorare solo con valori rappresentabili con 8 bit e quindi compresi tra 0 e 255.

Il piu' classico esempio di uso di numeri a 16 bit e' l'indirizzamento di una locazione di memoria. Abbiamo gia' parlato in precedenza dei metodi utilizzabili per indirizzare una locazione di memoria, ed in

quella sede abbiamo considerato implicito il fatto che i valori a 16 bit potessero essere maneggiati dalla CPU.

Come abbiamo visto in alcuni degli esercizi precedenti, un modo per assegnare un valore ad una coppia di registri consiste nell'assegnare separatamente a ciascuno di essi un valore ad 8 bit.

Fortunatamente per noi, lo Z80 mette a disposizione alcune istruzioni (per la verita' un po' pochine) che permettono di manipolare direttamente numeri a 16 bit. In questo capitolo esamineremo i metodi per caricare nei registri od in memoria valori a 16 bit, mentre nel prossimo affronteremo i tipi di istruzioni aritmetiche.

Note sull'uso dei valori a 16 bit per rappresentare indirizzi

Notiamo innanzitutto che tutti gli indirizzi di memoria vengono rappresentati con 16 bit.

Non e' possibile specificare un indirizzo con soli 8 bit anche se esso e' compreso tra 0 e 255, poiche' la CPU riconosce un'informazione come indirizzo solo se e' formata da 2 bytes di 8 bit ciascuno.

Abbiamo implicitamente tenuto conto di cio' quando abbiamo definito la istruzione:

LD A, (nn)

in cui nn rappresentano due valori ad 8 bit. Ricordiamo infine che un numero a 16 bit viene immagazzinato in una coppia di registri in modo che la parte alta del valore sia memorizzata nel primo registro della coppia (vedi il capitolo "Uno sguardo all'interno della CPU" - ... nella coppia HL H sta per "high" ed L per "low"-)

Caricamento in memoria di un valore a 16 bit

Il modo utilizzato dallo Z80 per compiere questo lavoro e' difficile da spiegare e da giustificare.

Quando vogliamo caricare valori a 16 bit in memoria, dobbiamo usare una convenzione opposta a quella appena descritta per le coppie di registri.

In memoria il byte piu' basso viene memorizzato per primo!

Facciamo un esempio: supponiamo di voler trasferire il contenuto di HL in memoria.

Prima dell'operazione la situazione e' la seguente:

H	L	Locazione	Contenuto
01	02	32000	00
		32001	00
		32002	00

e mostra che nella coppia di registri HL e' contenuto il valore 258 (102H) e che le memorie sono tutte azzerate.

Dopo l'operazione di trasferimento la situazione e' la seguente:

H	L	Locazione	Contenuto
01	02	32000	02
		32001	01
		32002	00

Cio' dimostra che i valori a 16 bit vanno posti in memoria (e nel listato del programma) antepo-
nendo la parte bassa a quella alta.

Non vi e' alcuna giustificazione intuitiva a questo modo di operare: lo Z80 e' stato progettato in questo modo, e noi dobbiamo imparare ad usarlo cosi' com'e'.

Per cortesia, leggete molto attentamente le righe seguenti in modo da recepire questa nuova convenzione, dato che la sua interpretazione errata e' di sicuro la maggior fonte di errore nella programmazione:

NEI REGISTRI : E' MEMORIZZATO PRIMA IL BYTE PIU' ALTO
IN MEMORIA E NEI PROGRAMMI: E' MEMORIZZATO PRIMA IL BYTE PIU' BASSO

Questa particolare convenzione non deve essere assolutamente dimenticata o presa in scarsa considerazione, dato ch ogni volta che dovrete rappresentare nel programma, in linguaggio macchina, un numero di 16 bit, dovrete scrivere i due bytes che lo compongono nell'ordine corretto e cioe' prima il byte piu' basso e poi il piu' alto.

Del resto non si poteva rimandare ulteriormente la presentazione di questo argomento: lavorare con lo Z80 senza le istruzioni a 16 bit e' praticamente impossibile e questo e' il prezzo che dobbiamo pagare per poterle usare.

Potete verificare quanto detto per conto vostro, facendo "eseguire" questo tipo di istruzione, caricata usando il programma "EZ Code", ed esaminando poi i contenuti delle locazioni di memoria con il comando "mem".

Caricamento di valori a 16 bit

Il tipo di caricamento piu' usato e' quello che permette di memorizzare un valore a 16 bit in una coppia di registri. La sua forma generale e':

LD rr, nn

Notiamo che vengono usate due lettere sia per indicare una coppia di registri "rr", sia un valore a 16 bit (2 bytes) "nn".

Per coloro che non posseggono un programma traduttore Assembler e che pertanto devono convertire manualmente un programma da codice mnemonico a codice macchina, questa discussione sull'ordine in cui vanno scritti i valori a 16 bit rappresenta un punto molto importante.

Anche se pero' possedete l' Assembler, dovrete prestare attenzione a questo strano modo di memorizzare i valori a 16 bit, dato che e' cosi' che li incontrerete quando vorrete leggere il codice memorizzato.

Specifichiamo meglio quanto detto, in un esempio:

Carica HL con 258

L'istruzione Assembler corrispondente e':

LD HL, 0102H

L'istruzione in codice macchina, come potete constatare nella tabella di conversione posta in fondo al presente volume, e':

21 XX XX

Cio' sta ad indicare che il valore esadecimale deve essere inserito al posto delle "XX XX". Ma attenzione: per la regola sopra citata dovremo inserire 0201 e non 0102.

L'istruzione assume dunque la forma:

21 02 01

La riga seguente mostra chiaramente l'effetto dell'inversione:

21 02 01 LD HL, 0102H (258 dec)

Questo esempio dovrebbe essere abbastanza chiaro e non creare problemi. Ma e' importante che prendiate molta confidenza con questo modo di trattare valori a 16 bit, per non incontrare difficolta' anche quando dovrete scrivere programmi veramente vostri.

Altre istruzioni di caricamento

E' possibile caricare un valore a 16 bit oltre che in una coppia di registri anche in un registro indice (ricordiamo che i registri indice sono i piedi a 16 dita della CPU).

LD IX, nn
LD IY, nn

ed anche trasferire il contenuto da una coppia di registri a due locazioni di memoria adiacenti (questo tipo di operazione e' equivalente a quella che permetteva di trasferire in memoria un numero contenuto in un registro).

La forma generale di questo tipo di istruzione e' la seguente:

```
LD (nn), rr
LD (nn), IX
LD (nn), IY
```

Ricordiamo che l'uso delle parentesi equivale ad individuare un indirizzo. L'ultima istruzione pertanto va letta come: "Carica nella locazione di memoria di indirizzo nn il valore di IY".

Notiamo che per memorizzare un valore a 16 bit contenuto in una coppia di registri od in un registro indice occorrono due locazioni di memoria. Come avete visto, nell'istruzione presentata non e' stato necessario indicare entrambi gli indirizzi (poiche' la CPU e' in grado di immaginare quale sia l'indirizzo della seconda locazione necessaria), ma bisogna fare attenzione a non confondere le operazioni con numeri a 8 bit con quelle con numeri a 16 bit.

Anche in questo caso e' rispettata la simmetria delle istruzioni: vi sono infatti le istruzioni che permettono di trasportare in una coppia di registri o in un registro indice il contenuto di due locazioni di memoria adiacenti.

```
LD rr, (nn)
LD IX, (nn)
LD IY, (nn)
```

Esercizio

Il manuale dello Spectrum ci insegna che l'indirizzo di partenza dello spazio non utilizzato e' contenuto nelle due locazioni di memoria 23653 e 23654.

In BASIC era possibile determinare questo indirizzo con la funzione PEEK, nel modo seguente:

```
PRINT PEEK 23653 + 256 * PEEK 23654
```

Per scrivere l'istruzione corrispondente in linguaggio macchina ricordiamo che il valore esadecimale corrispondente a 23653 e' 5C65H.

Il programma che permette di visualizzare l'indirizzo contenuto nelle locazioni 23653 e 23654 e' il seguente:

```
ED 4B 65 5C      LD BC, (23653)
C9              RET
```


OSSERVIAMO CON MOLTA ATTENZIONE CHE IL VALORE 5C65H E' STATO INSERITO NEL PROGRAMMA SCRIVENDO PRIMA IL VALORE DEL BYTE BASSO E POI QUELLO DEL BYTE ALTO. SAREMMO INCORSI IN UN GRAVE ERRORE SE AVESSIMO INSERITO I DUE BYTES INVERTENDOLI.

Poiche' sullo Spectrum e' il sistema operativo stesso che definisce la posizione dell'area di memoria non utilizzata, noi possiamo determinarla soltanto una volta nel corso dell'esecuzione.

Per raccogliere quest'informazione abbiamo usato i registri BC poiche' la funzione USR, terminato il vostro programma in linguaggio macchina, restituisce proprio il contenuto di questi registri.

Osserviamo infine che l'istruzione LD BC, (nn) richiede ben 4 bytes!

Potete utilizzare programmi simili al precedente per indagare sul contenuto di tutte le altre variabili di sistema contenenti indirizzi a 16 bit, il cui elenco si trova nel manuale dello Spectrum.

RIEPILOGO

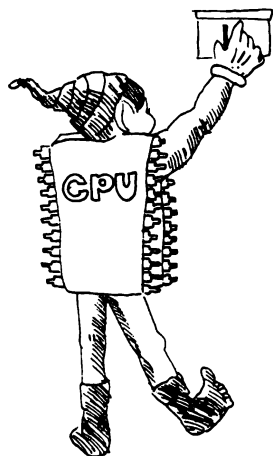
Possiamo assegnare un valore a 16 bit ad un registro indice o ad una coppia di registri, specificando il valore stesso oppure l'indirizzo della locazione di memoria in cui esso e' contenuto.

In modo analogo possiamo trasferire in memoria il valore contenuto in una coppia di registri o in un registro indice.

L'unica cosa da notare e' che l'ordine in cui i bytes di un valore a 16 bit vengono memorizzati dallo Z80 (e quindi in cui devono essere scritti nelle istruzioni di un programma) non e' quello usuale.

E' estremamente importante quindi ricordarsi che lo Z80 memorizza i

valori a 16 bit ponendo prima il byte basso e poi quello alto!!!



OPERAZIONI SULLO STACK

=====

Istruzioni per operazioni sullo stack

Mnemonico	Bytes	Tempo richiesto	Effetto sui flag					
			C	Z	PV	S	N	H
PUSH Coppia reg	1	11	-	-	-	-	-	-
PUSH IX o IY	2	15	-	-	-	-	-	-
POP Coppia reg	1	10	-	-	-	-	-	-
POP IX o IY	2	14	-	-	-	-	-	-
LD SP, indirizzo	3	10	-	-	-	-	-	-
LD SP, (indirizzo)	3	20	-	-	-	-	-	-
LD SP, HL	1	6	-	-	-	-	-	-
LD SP, IX o IY	2	10	-	-	-	-	-	-

Notazioni sui flag:

- # indica che il flag e' alterato dall'operazione
- 0 indica che il flag e' posto a 0
- 1 indica che il flag e' posto a 1
- indica che il flag non e' alterato dall'operazione

Vorremmo ricordare l'immagine utilizzata all'inizio di questo libro per presentarvi lo stack. Esso e' qualcosa che la CPU utilizza per conservare informazioni senza dover contemporaneamente ricordare l'indirizzo della locazione in cui le ha poste.

Uno dei vantaggi che presenta lo stack, anche se a prima vista sembra quasi il contrario, e' quello di coinvolgere solo numeri a 16 bit. Cio' e' dovuto al fatto che esso e' stato ideato per conservare indirizzi, i quali sono rappresentabili solo con numeri a 16 bit.

Le istruzioni che permettono di inserire valori in cima allo stack sono:

```
PUSH rr
PUSH IX
PUSH IY
```

mentre quelle per estrarre un valore dalla cima dello stack sono:

POP rr
POP IX
POP IY

Come vedete, esse sono piuttosto semplici e non richiedono la specifica di alcun indirizzo da parte del programmatore.

Quando fanno riferimento a coppie di registri, e quindi non a registri indice, queste istruzioni occupano un solo byte e pertanto sono molto economiche in termini di occupazione di memoria nell'ambito di un programma.

L'operazione di PUSH, inoltre, non e' distruttiva: il registro a 16 bit interessato mantiene lo stesso valore anche dopo l'operazione di PUSH.

Osserviamo inoltre che, essendo possibile eseguire un'operazione di PUSH o di POP su una qualsiasi coppia di registri, non e' necessario riportare l'indirizzo impilato nello stack nella stessa coppia di registri di partenza.

Ad esempio, e' possibile agire in tal modo:

```
PUSH BC
POP HL
```

L'effetto di queste due istruzioni e' il seguente: il contenuto dei registri BC non e' mutato, mentre la coppia di registri HL ha assunto lo stesso valore che aveva la coppia di registri BC prima dell'operazione di PUSH.

Queste due istruzioni simulano perfettamente l'operazione di

```
LD rr, rr'
```

che non e' prevista per lo Z80.

Poiche' le istruzioni di PUSH e di POP che coinvolgono coppie di registri occupano un solo byte, l'uso di questa struttura non incide in modo notevole sulla quantita' di memoria occupata.

Altro vantaggio delle operazioni suddette e' quello di poter essere applicate anche alla coppia di registri AF! Esse sono infatti due delle poche istruzioni che si possono applicare alla coppia AF, e risultano particolarmente utili, permettendo di conservare il contenuto dei vari flag.

Ad esempio, con un'operazione di PUSH AF possiamo salvare il contenuto dei due registri A ed F, operare un calcolo che ha come indesiderabile effetto quello di modificare il contenuto dei flag, e ripristinare poi

la precedente condizione dei flag con l'operazione POP AF.

Come sappiamo, l'utilita' delle istruzioni di PUSH e di POP consiste nel fatto che non occorre ricordare le locazioni in cui l'operazione di PUSH ha portato il valore oppure da cui l'operazione di POP deve prelevarlo.

Converrete sicuramente con noi che non e' necessario che sia riservata la medesima area di memoria per lo stack sui sistemi a 16K ed a 48K.

La CPU e' infatti in grado di ricordare l'indirizzo attuale della cima dello stack utilizzando un registro a 16 bit, detto per l'appunto stack pointer. Abbiamo accennato brevemente all'esistenza di questo registro nel capitolo in cui descrivevamo la struttura della CPU, ma non ne abbiamo piu' fatto cenno durante la presentazione delle varie istruzioni come LOAD,... per il fatto che questo registro non va trattato come gli altri.

Il compito principale dello stack pointer e' quello di definire la posizione dello stack in memoria, e le istruzioni che permettono di inizializzarlo da programma sono le seguenti:

```
LD SP, nn
LD SP, (nn)
LD SP, IX
LD SP, IY
```

Potete esaminare lo stack dello Spectrum usando il comando "mem" del programma "EZ Code" e guardando il contenuto degli ultimi 30 - 40 byte prima del RAMTOP.

```
*****
Attenzione pero' a non cambiare il contenuto delle locazioni che fanno
parte dello stack.
```

```
*****
Quasi tutte le variazioni causano infatti un blocco del computer: lo
schermo diventa bianco e dovete ricominciare tutto da capo. Cio' e'
dovuto al fatto che il sistema operativo utilizza una serie di infor-
mazioni depositate nello stack e quindi una loro variazione, anche mi-
nima, puo' causare un pasticcio enorme.
```

Per lo stesso motivo e' consigliabile non variare il contenuto dello stack pointer a meno di non essere piu' che sicuri di quel che si sta facendo.

Nota:

In un programma ben fatto il numero di PUSH e POP dovrebbe, alla fine, essere lo stesso indipendentemente dalla via seguita. Un errore in tal senso potrebbe portare a strani risultati.

Esercizio

Potete utilizzare il programma seguente per esaminare l'indirizzo da cui e' stata chiamata la routine USR. Infatti, con l'operazione di POP portiamo l'indirizzo che si trovava in cima allo stack nella coppia di registri BC e con un PUSH ripristiniamo la situazione originale dello stack.

```
C1      POP BC      ;poni l'indirizzo in BC
C5      PUSH BC     ;ricopia l'indirizzo nello stack
C9      RET
```



ARITMETICA DEI NUMERI A 16 BIT

Istruzioni per l'aritmetica dei numeri a 16 bit

Mnemonico	Bytes	Tempo Richiesto	Effetto sui flag					
			C	Z	PV	S	N	H
ADD HL, Coppia reg	1	11	#	-	-	-	0	?
ADD HL, SP	2	11	#	-	-	-	0	?
ADC HL, Coppia reg	2	15	#	#	#	#	0	?
ADC IX, SP	2	15	#	#	#	#	0	?
ADD IX, BC o DE	2	15	#	-	-	-	0	?
ADD IX, IX	2	15	#	-	-	-	0	?
ADD IX, SP	2	15	#	-	-	-	0	?
ADD IY, BC o DE	2	15	#	-	-	-	0	?
ADD IY, IY	2	15	#	-	-	-	0	?
ADD IY, SP	2	15	#	-	-	-	0	?
SBC HL, Coppia reg	2	15	#	#	#	#	1	?
SBC HL, SP	2	15	#	#	#	#	1	?

Notazioni per i flag:

- # indica che il flag e' alterato dall'operazione
- 0 indica che il flag e' posto a 0
- 1 indica che il flag e' posto ad 1
- indica che il flag non e' alterato dall'operazione
- ? indica che l'effetto non e' conosciuto

Uno dei vantaggi derivanti dal fatto che un microprocessore ad 8 bit sia in grado di manipolare anche valori a 16 bit e' che si possono usare questi numeri per specificare indirizzi o per eseguire calcoli su numeri interi compresi tra 0 e 65355 (o, comprendendo i valori negativi, tra -32768 e +32767).

In quest'ottica possiamo capire il motivo per cui in alcuni dei primi microcalcolatori tutta l'aritmetica BASIC fosse limitata solo a numeri interi compresi tra -32000 e + 32000.

Sebbene ci sia data la possibilita' di effettuare alcune operazioni aritmetiche con i numeri a 16 bit, dobbiamo osservare che sono veramente poche, rispetto a quelle per i numeri a 8 bit, le possibilita' che il computer ci offre!

Coppia di registri privilegiata

Come il registro A era privilegiato nelle operazioni aritmetiche con i numeri a 8 bit,cosi' la coppia di registri HL e' privilegiata nelle operazioni aritmetiche con numeri a 16 bit.

Questo favoritismo non e' cosi' pronunciato come nel caso del registro A, ad esempio non e' possibile sottintendere il nome della coppia di registri HL nelle operazioni aritmetiche.

Addizione:

Le addizioni sono molto semplici:

```
ADD HL,BC
ADD HL,DE
ADD HL,HL
ADD HL,SP
```

Ma attenzione: non e' contemplata la possibilita' di sommare direttamente al contenuto di HL un valore noto nn. Sullo Z80 non esiste l'istruzione ADD HL, nn!

Per poter eseguire un calcolo di questo tipo dobbiamo ricorrere ad un artificio:

```
LD DE,nn
ADD HL,DE
```

Ora, se pensate che per fare questa operazione devono essere impegnati ben quattro dei sette registri ad 8 bit che si hanno a disposizione, potete subito capire perche' essa non venga usata molto spesso.

Notiamo inoltre che non e' possibile sommare al contenuto di HL il contenuto di uno dei registri indice; poiche' non vi sono neppure delle istruzioni LOAD che permettano di trasferire il contenuto di IX o IY in BC o DE, l'unica via che possiamo percorrere per fare la somma tra IX ed HL e' la seguente:

```
PUSH IX
POP DE
ADD HL,DE
```

Un caso particolare e' rappresentato dal registro SP (stack pointer). L'addizione e' infatti una delle poche operazioni in cui il registro SP viene trattato come un comune operando, ma, ovviamente, non lo potete usare come una variabile!

Provate a pensare a quello che succederebbe nelle operazioni di PUSH e POP se variaste a vostro piacimento il contenuto di SP!

Effetto sui flag

L'aritmetica a 16 bit rappresenta un campo in cui il flag di carry e' sovrano. Infatti, osservando la tabella posta all'inizio del presente capitolo, noterete che vi e' un solo flag, oltre al flag di carry, influenzato dall'istruzione ADD ed e' il flag di sottrazione (e sappiamo tutti che un'addizione non e' certo una sottrazione!)

Il flag di carry viene settato quando si ha riporto nell'ottavo bit di H, mentre il riporto eventualmente presente in L viene trasferito direttamente in H durante il calcolo.

Addizione con riporto

Date le limitazioni che l'uso di numeri a 16 bit impone, abbiamo la possibilita', come nel caso dei numeri ad 8 bit, di concatenare tra loro piu' addizioni.

L'istruzione di "addizione con riporto" (ADC) opera in modo simile all'istruzione ADD e puo' essere usata con le stesse coppie di registri:

```
ADC HL,BC
ADC HL,DE
ADC HL,HL
ADC HL,SP
```

Sottrazione

Anche la sottrazione e' un'operazione piuttosto semplice, ma fate attenzione, poiche' non ci sono sottrazioni senza riporto. Per questo motivo, se non siete sicuri dello stato del flag di carry, dovete inserire nel vostro programma un'istruzione che azzeri il carry prima dell'operazione di sottrazione.

```
SBC HL,BC
SBC HL,DE
SBC HL,HL
SBC HL,SP
```

(L'ultima istruzione ha una applicazione ovvia: ponete in HL l'indirizzo dell'ultima locazione di memoria usata per il vostro programma per il controllo del video e per le variabili, e sottraete il contenuto di SP; il risultato (negativo) rappresenta lo spazio rimasto libero in memoria).

Effetto delle operazioni con riporto sui flag

Guardando la tabella delle operazioni potete notare che quelle di ADC ed SBC influenzano il valore di tre flag, mentre gli stessi flag non vengono influenzati dalle operazioni di addizione e sottrazione semplice.

I flag interessati sono il flag di zero, il flag di segno ed il flag di overflow, che assumono lo stato di 0 od 1 in accordo con il risultato dell'operazione.

Aritmetica dei registri indice

L'unica operazione ammessa per questi registri e' l'addizione con riporto (ADC) e per di piu' l'insieme dei registri che possono comparire nell'istruzione come secondo operando e' estremamente limitato. Infatti e' possibile:

- Addizionare ad un registro indice una delle coppie di registri scelta tra BC e DE
- Addizionare ad un registro indice se stesso
- Addizionare ad un registro indice lo stack pointer.

Soluzione al problema della memoria libera

Nello Spectrum, per individuare la fine dell'area individuata dal vostro programma vengono utilizzate due particolari locazioni di memoria, dette simbolicamente STKEND: la 23653 e la 23654.

Per risolvere il nostro problema possiamo quindi portare in HL il contenuto di queste due locazioni:

```
LD HL, (STKEND)
```

e sottrarre poi il contenuto dello stack pointer (SBC HL, SP ?)

Ma attenzione: la sottrazione tiene conto del flag di carry, occorre pertanto assicurarsi prima che esso sia posto a zero. Cio' puo' essere ottenuto facilmente utilizzando l'istruzione AND A (vedi capitolo sugli operatori logici):

```
AND A
SBC HL, SP
```

Assegnatevi otto se avete svolto correttamente il programma, sei se vi siete accorti che era necessario azzerare il flag di carry ma non sapevate come farlo, quattro se vi siete completamente dimenticati di

affrontarne il problema.

Dato che lo stack e' posto nella parte alta della memoria, il valore dello stack pointer sara' superiore a quello dell'ultima locazione utilizzata dal vostro programma (in caso contrario, vi siete cacciati in un pasticcio mostruoso) e pertanto il risultato della sottrazione sara' negativo.

Possiamo ora procedere a calcolare il numero di bytes rimasti liberi, trasformando questo valore negativo in un numero positivo ed utilizzando la coppia di registri BC (si poteva usare anche DE).

Per prima cosa dobbiamo spostare il contenuto di HL in BC e, non esistendo l'istruzione diretta LD BC,HL, dobbiamo ricorrere alle operazioni di PUSH e di POP:

```
PUSH HL
POP BC
```

Alla fine di queste operazioni si avra' HL=BC.

Per ottenere ora HL = -BC sottraiamo due volte da HL il valore di BC (non dimentichiamoci pero' che il flag di carry era stato settato alla fine della sottrazione precedente e che pertanto dovra' essere riazzerato).

```
AND A
SBC HL, BC
SBC HL, BC
```

Alla fine di queste operazioni HL conterra' il valore opposto a quello originario, che era negativo, e quindi conterra' esattamente il numero (positivo) di bytes rimasti liberi in memoria.

Non ci resta ora che spostare il contenuto di HL in BC affinche' venga visualizzato dalla funzione USR. Per portare HL in BC eseguiamo:

```
PUSH HL
POP BC
```

e terminiamo il programma con l'ormai familiare istruzione di return

```
RET
```

Visto come e' maneggevole lo stack?
Vi e' stato tutto chiaro?
Speriamo di si'.



CICLI E SALTI

=====

I cicli e i salti sono quelli che danno una reale potenza ai programmi perche' permettono di eseguire parti differenti di programma in funzione di particolari condizioni derivanti dai calcoli precedenti. Quando avrete acquisito l'abilita' necessaria per prendere decisioni e far eseguire di conseguenza diverse istruzioni del programma sarete sulla strada del successo.

Dovete fare pero' attenzione, perche' utilizzando con troppa liberta' le istruzioni di salto rischiate di costruire programmi difficili da seguire e pressoché' impossibili da correggere.

Vi suggeriamo caldamente di progettare i vostri programmi con molta cura prima di codificarli in linguaggio macchina e di seguire le indicazioni da noi fornite nel capitolo sulla progettazione dei programmi in linguaggio macchina.

Vogliamo enfatizzare questo concetto proprio a questo punto, perche' i cicli e i salti sono strutture che facilmente portano ad allontanarsi da una corretta progettazione.

L'equivalente del GOTO in Linguaggio Macchina

In BASIC l'istruzione GOTO, che dovrebbe esservi abbastanza familiare, trasferisce il controllo del programma alla linea da voi specificata.

Anche l'equivalente costruito in linguaggio macchina e' molto semplice: basta specificare nell'istruzione di salto l'indirizzo della locazione in cui la CPU deve andare a cercare la nuova istruzione ed il gioco e' fatto.

Le istruzioni di salto di questo tipo sono:

```
JP xx xx
JP (HL)
JP (IX)
JP (IY)
```

La prima di queste istruzioni puo' essere collegata anche allo stato dei flag, dando origine alle istruzioni di salto condizionato. Ad esempio l'istruzione:

```
JP Z, 0000H
```

deve essere interpretata nel modo seguente: "Salta alla locazione di indirizzo 0000H se il flag di zero e' settato". Ricordiamo per inciso che l'indirizzo 0000H e' quello a cui la CPU dello Spectrum salta

quando accendete il computer e pertanto l'istruzione JP 0000H produce l'effetto di azzerare tutta la memoria e di ricominciare tutto da capo con "K" in fondo al video.

Notiamo infine che, dato che la CPU non ammette errori, se le dite "JUMP", salta comunque senza porsi problemi. Ora, poiché quasi tutti i codici esadecimali possono essere considerati come codici operativi di qualche istruzione, la CPU non si preoccupa se voi la mandate in mezzo ad un blocco di dati o sul secondo byte di un'istruzione: lei legge il contenuto del byte da voi indicato e opera come se fosse il primo byte della nuova istruzione da eseguire.

Il modo in cui la CPU esegue l'istruzione di salto è molto elementare: essa utilizza un piccolo contatore detto "program counter" in cui di volta in volta memorizza l'indirizzo di partenza dell'istruzione che dovrà essere eseguita successivamente. Nel corso normale del programma cioè quando non ci sono salti, la CPU ad ogni passo guarda la istruzione che deve essere eseguita e somma al contenuto del "program counter" il numero di bytes che la compongono; così ad esempio se la istruzione attuale è formata da 2 bytes, somma 2 al "program counter", mentre se è formata da 4 bytes, somma 4 al "program counter".

Quando invece incontra un'istruzione di salto (Jump), la CPU sostituisce al contenuto del "program counter" l'indirizzo da voi specificato.

Per questo motivo dovete fare molta attenzione a non commettere il minimo errore.

Salti lunghi e salti corti

Possiamo definire le precedenti istruzioni come istruzioni di "salto lungo", dato che specificando direttamente l'indirizzo a 16 bit è possibile far saltare la CPU a qualsiasi locazione di memoria sia presente nel vostro computer.

Gli svantaggi di questo tipo di salto consistono nel fatto che:

- A. Spesso non si deve saltare molto lontano ma l'indirizzo di arrivo deve essere sempre espresso con 16 bit e quindi anche per salti corti l'istruzione occupa 3 bytes.
- B. Il programma che contiene questo tipo di istruzione non risulta rilocabile dato che nelle istruzioni di salto viene specificato un indirizzo assoluto.

Per eliminare questi due grossi svantaggi i progettisti dello Z80 hanno pensato di aggiungere delle istruzioni particolari per i "salti corti". Queste istruzioni sono generalmente definite istruzioni di "salto relativo" e permettono di eseguire salti all'indietro ed in a-

vanti con un limite per la lunghezza del salto stesso. E' infatti possibile, partendo dall'indirizzo attualmente presente nel "program counter", saltare in avanti fino a 127 bytes (+127) e indietro fino a 128 bytes (-128). La lunghezza del salto puo' quindi essere espressa con un solo byte!

Istruzioni di salto relativo

Hanno la forma generale

JR d

ove d rappresenta la lunghezza (relativa) del salto.

Possiamo anche in questo caso condizionare il salto al verificarsi di qualche condizione; ad esempio si puo' saltare se il flag di carry ha il valore 1 oppure se il flag di zero ha il valore 0, e cosi' via. L'istruzione di salto condizionato relativo assume la forma seguente:

JR cc,d

ove cc rappresenta il codice della condizione che deve verificarsi per effettuare il salto.

In tutte le istruzioni di salto relativo il parametro d rappresenta il valore che la CPU deve sommare al valore del "program counter" per saltare. Se tale valore e' positivo il salto viene fatto ovviamente in avanti, mentre se e' negativo da' origine ad un salto all'indietro.

Se avete presente quanto abbiamo detto nel capitolo dedicato alla rappresentazione dei numeri relativi, dovrete aver gia' capito il motivo per cui l'intervallo di oscillazione del salto risulta essere tra -128 e +127.

Vogliamo farvi notare che quando la CPU esegue l'istruzione di salto il "program counter" e' gia' posizionato sull'istruzione che nel listato del programma si trova immediatamente dopo l'istruzione di salto stesso e che dovrebbe essere logicamente eseguita se non fossero verificate le condizioni per "saltare".

Infatti la CPU quando giunge all'istruzione JR, prima di eseguirla, riconosce che e' formata da 2 bytes e somma 2 al contenuto del "program counter" che puntera' cosi' l'istruzione successiva.

Per chiarire meglio questo concetto vediamo con un esempio come opera la CPU di fronte ad un'istruzione di salto.

Supponiamo che a partire dalla locazione 32000 sia stato caricato il seguente segmento di programma:

Locazione	Istruzione
32000	ADD A, B
32001	JR Z, 02H
32003	LD B, 00H
32005	LD HL, 4000H

e vediamo come la CPU opera durante la sua esecuzione.

Preleva il valore contenuto della locazione 32000

Dato che esso rappresenta il codice operativo di un'istruzione ad 1 byte, aggiunge 1 al "program counter" che assume il valore 32001

Esegue l'istruzione

Preleva il byte puntato dal "program counter" (32001)

Dato che esso rappresenta il codice operativo di un'istruzione a 2 bytes, aggiunge 2 al "program counter" che assume il valore 32003

Preleva il secondo byte dell'istruzione

Esegue l'istruzione

L'istruzione a 2 bytes appena prelevata e' un'istruzione di salto condizionato. Se il flag di zero e' settato modifica il contenuto del "program counter" altrimenti non fa nulla. Nell'eseguire questa istruzione dunque la CPU opera nel modo seguente:

se il flag di zero vale 1 : somma 2 al "program counter" che passa cosi' a 32005

se il flag di zero vale 0 : non fa nulla e cosi' il valore del "program counter" rimane invariato (32003)

Pertanto dopo la CPU eseguirà l'istruzione che inizia alla locazione 32003 se il flag di zero valeva 0, mentre se il flag valeva 1 eseguirà l'istruzione che inizia alla locazione 32005.

In altre parole con l'istruzione JR otteniamo l'effetto di saltare, se si sono verificate certe condizioni, l'istruzione LD B,00H.

Quanto detto aiuta anche a capire come mai il tempo necessario per la esecuzione dell'istruzione di salto condizionato dipenda dal verificarsi o meno della condizione indicata. E' infatti evidente che occorre meno tempo nel caso in cui la CPU non debba far niente (flag di zero non settato) rispetto a quando invece dovrà calcolare il nuovo valore del "program counter" (flag di zero settato).

Ovviamente anche i salti condizionati possono essere fatti all'indietro assegnando valori negativi al parametro d.

Esercizio

Dato che l'istruzione di salto relativo occupa 2 bytes e che all'atto della sua esecuzione il "program counter" punta già all'istruzione successiva, provate a valutare l'effetto che avrebbe in un programma l'istruzione JR -2.

Realizzazione del costrutto FOR ...NEXT in linguaggio macchina

Sicuramente non troverete difficoltà ad interpretare il seguente programma BASIC:

```
FOR I=1 TO 6
LET C=C+1
NEXT I
```

Ebbene anche in linguaggio macchina, seppure sotto una forma diversa, e' possibile realizzare una struttura ciclica di programma. Per realizzare un ciclo come quello dell'esempio possiamo usare le istruzioni aritmetiche e di salto condizionato :

```
LD B, 1           ;poni il contatore uguale a 1
LD A, 7           ;valore successivo a quello max
CICLO INC C       ;C=C+1
INC B            ;incrementa il contatore
CP B            ;A=B ?
JR NZ, CICLO     ;se A(<)B ripeti il ciclo
```

Notiamo che per poter realizzare la struttura di ciclo, abbiamo dovuto usare ben due registri: il registro B come contatore ed il registro A per ricordare la condizione di fine ciclo. Cio' e' dovuto al fatto che l'istruzione INC B, non influenzando in questo caso i flag, non fornisce nessuna delle condizioni utilizzabili per il salto condizionato. Tenendo conto di cio' si potrebbe pero' elegantemente risolvere il problema utilizzando un contatore che si decrementa. Infatti, sapendo che il ciclo deve essere ripetuto sei volte, possiamo inizializzare il contatore a 6 e decrementarlo di 1 ad ogni passo. In questo modo il ciclo ha termine quando il valore assunto dal contatore dopo il decremento e' zero. Il programma precedente puo' quindi essere modificato nel modo seguente:

```
LD B, 6           ;inizializza il contatore
CICLO INC C       ;C=C+1
DEC B            ;decrementa il contatore
JR NZ, CICLO     ;se non e' 0 ripeti il ciclo
```

Come potete vedere questa nuova versione risulta notevolmente piu' efficiente della precedente.

Per facilitare ancora di piu' questo tipo di operazione, il processore Z80 mette a disposizione un'istruzione particolare che unifica le due funzioni di decremento e salto condizionato.

Questa istruzione si presenta nella forma:

DJNZ d

ed ha il seguente significato: "decrementa B e salta se il risultato non e' zero". Il parametro d indica come al solito la lunghezza del salto.

Anche questa nuova istruzione occupa 2 bytes, di cui il primo rappresenta il codice operativo.

Notiamo che con questa particolare istruzione il registro B assume a tutti gli effetti il ruolo di contatore e pertanto non si possono realizzare cicli con un numero di iterazioni superiori a 256.

Per realizzare cicli piu' lunghi si deve fare uso di piu' cicli annidati tra loro, come illustra il seguente esempio:

```
CICLOEST  LD B, 10H           ;B=16
          PUSH BC          ;salva il valore di B
          LD B, 00H        ;inizializza B per 256 iterazioni
CICLOINT  .....
          .....          ;blocco di istruzioni interne
          .....
          DJNZ CICLOINT    ;svolto il ciclo interno 256 volte?
          POP BC           ;ripristina il valore di B salvato
          DJNZ CICLOEST    ;svolto il ciclo esterno 16 volte?
```

Esercizio

Provate ad eseguire alcune iterazioni annotandovi su un foglio di carta come varia il valore di B dopo ogni istruzione.

Cicli di ritardo

Vi sono delle volte in cui un programma in linguaggio macchina viene eseguito talmente velocemente da richiedere l'introduzione di cicli di ritardo in modo da rallentarne l'esecuzione. Cio' accade ad esempio quando si vogliono registrare dei dati su cassetta (i segnali devono venire trasmessi distanziati tra loro per permettere una corretta registrazione) o stampare su carta (e' impensabile pensare che la vostra stampante possa stampare piu' di 1000 caratteri in un secondo!).

Un metodo abbastanza comodo per realizzare cicli di ritardo consiste nel costruire un ciclo in cui l'unica operazione eseguita e' il decremento del contatore.

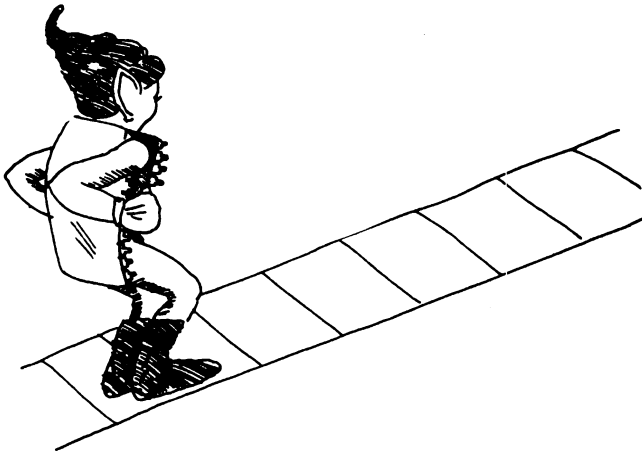

```
LD B, valore
ATTESA DJNZ ATTESA
```

In questo caso l'istruzione DJNZ ATTESA viene eseguita un numero di volte pari al valore specificato in "valore". In pratica si genera una interruzione logica del programma che riprende il suo corso normale solo al termine del ciclo di ritardo.

Provate ora a pensare a cosa sarebbe successo se al posto dell'istruzione DJNZ avessimo usato

```
ATTESA JR ATTESA
```

Lanciando un programma di questo tipo potete aspettare finche' volete per vederlo terminare!



Istruzioni di chiamata e di ritorno

Mnemonico	Byte	Tempo richiesto	Effetto sui flag					
			C	Z	PV	S	N	H
CALL indirizzo	3	17	-	-	-	-	-	-
CALL cc, indirizzo	3	10/17	-	-	-	-	-	-
RET	1	10	-	-	-	-	-	-
RET cc	1	5/11	-	-	-	-	-	-

Nota: cc rappresenta la condizione che si deve verificare affinché venga eseguita la chiamata o il ritorno.

Le condizioni che possono essere usate nelle istruzioni condizionate sono:

Flag	Simbolo	Significato
Carry	C	Flag di carry settato (=1)
	NC	Flag di carry non settato (=0)
Zero	Z	Flag di zero settato (=1)
	NZ	Flag di zero non settato (=0)
Parita'	PE	Parity Even (pari): flag settato (=1)
	PO	Parity Odd (dispari): flag non settato (=0)
Segno	M	Minus (valore negativo): flag settato (=1)
	P	Positive (val positivo): flag non settato (=0)

Effetto sui flag

Nessuna delle istruzioni di chiamata o di ritorno modifica lo stato dei flag.

Tempi

Nei casi in cui sono indicati due tempi, il primo di essi si riferisce al tempo necessario per l'esecuzione dell'istruzione quando la condizione richiesta non si e' verificata.

L'uso dei sottoprogrammi in linguaggio macchina risulta altrettanto agevole di quanto lo e' nella programmazione BASIC, se non addirittura di piu'.

Vogliamo farvi notare a questo proposito che la funzione BASIC "USR" che utilizzate per lanciare i vostri programmi in linguaggio macchina non e' altro che una chiamata ad una subroutine: il vostro programma. Ed e' proprio per questo che ogni vostro programma in linguaggio macchina deve terminare con un'istruzione di RETURN.

La funzione USR vi permette quindi di provare dei sottoprogrammi in modo indipendente e senza che siano inseriti in un programma principale.

Quando poi i sottoprogrammi verranno posti all'interno di un unico programma bisognera' prestare attenzione ad indicare con esattezza nelle istruzioni di chiamata l'indirizzo effettivo di partenza della routine desiderata.

Questo puo' dare origine ad alcuni problemi se voi utilizzate per memorizzare i sottoprogrammi in linguaggio macchina degli array variabili, dato che l'indirizzo di queste variabili non e' necessariamente prefissato.

Notiamo infine che un programma in linguaggio macchina contenente sottoprogrammi, difficilmente puo' essere rilocato dato che le istruzioni di chiamata fanno riferimento a degli indirizzi assoluti.

In linguaggio macchina, come del resto in BASIC, la chiamata ad una subroutine puo' essere vincolata al verificarsi di particolari condizioni; e' possibile quindi realizzare con semplici istruzioni un costrutto del tipo:

```
IF (condizione) THEN GOSUB (linea)
```

A proposito di condizioni, vogliamo far notare che occorre manovrare con molta attenzione sia i flag che i registri all'interno di una subroutine per evitare di modificare involontariamente qualche condizione da testare dopo il ritorno al programma principale. Bisogna comunque evitare di saltare direttamente ad una nuova istruzione di CALL al ritorno al programma principale.

Per condizionare la chiamata ad una subroutine si possono usare solo quattro dei sei flag dello Z80:

```
il flag di Carry  
il flag di Zero  
il flag di Parita' (o di overflow)  
il flag di Segno
```

Ricordiamo che il valore di ciascuno di questi flag risulta in accordo con l'ultima istruzione eseguita che ne ha influenzato il valore.

Per questo e' buona norma porre l'istruzione di CALL o di RETURN con condizione immediatamente dopo l'istruzione che modifica il flag interessato.

Per esempio un programma del tipo:

```
LD A, (numero)
CP 1
CALL Z, Uno
CP 2
CALL Z, Due
CP 3
CALL Z, Tre
```

vi permette di selezionare una delle tre routine Uno, Due e Tre in funzione del valore contenuto in A. Ma attenzione: e' indispensabile per ottenere l'effetto voluto che all'interno delle singole subroutine non venga alterato il contenuto del registro A!!!! Sapreste dire perche'?

Per ottenere lo stesso scopo possiamo operare anche in un altro modo dato che dobbiamo scegliere tra tre possibilita', operando anche sul flag di carry:

```
LD A, (numero)
CP 2
CALL Z, Due           ;A=2
CALL C, Uno           ;A<2 ==> A=1
CP 3
CALL Z, Tre           ;A>2 ==> A=3
```

In questo programma abbiamo tenuto presente che mentre l'istruzione "CP 2" modifica sia il flag di carry che quello di zero, le istruzioni di chiamata non modificano lo stato dei flag.

Anche l'istruzione di Return puo' essere assoggettata a delle condizioni ed e' molto usata anche se teoricamente l'uso di una tale istruzione non rientra nei canoni di una buona programmazione.

OPERAZIONI SU BLOCCHI DI DATI

=====

Istruzioni per confrontare e muovere blocchi di dati

Mnemonico	Byte	• Tempo richiesto	Effetto sui flag					
			C	Z	PV	S	N	H
LDI	2	16	-	-	#	-	0	0
LDD	2	16	-	-	#	-	0	0
LDIR	2	21/16	-	-	0	-	0	0
LDDR	2	21/16	-	-	0	-	0	0
CPI	2	16	-	#	#	#	1	#
CPD	2	16	-	#	#	#	1	#
CPIR	2	21/16	-	#	#	#	1	#
CPDR	2	21/16	-	#	#	#	1	#

Notazioni per i flag:

Indica che il flag e' alterato dall'operazione

0 Indica che il flag assume il valore 0

1 Indica che il flag assume il valore 1

- Indica che il flag non viene alterato

Tempi

Per le istruzioni con ripetizione, i tempi indicati riguardano il singolo passo ed il primo di essi fa riferimento al caso in cui non si sia verificata la condizione di "fine ciclo" (per esempio per CPIR, BC=0 oppure A=(HL)).

Ormai dovrete aver preso sufficiente dimestichezza con il linguaggio macchina da utilizzarlo come una lingua straniera; anzi, se lo avete imparato bene, dovrete cominciare a pensare direttamente in Assembler.

Il presente capitolo illustra l'ultimo blocco di istruzioni che abbiamo catalogato tra quelle piu' usate nel corso della programmazione dello Z80, mentre nei successivi presenteremo, seppure in modo piu' succinto, le istruzioni rimanenti che pur essendo importanti per una programmazione ad alto livello, vengono usate meno frequentemente nella pratica di tutti i giorni e vi sconsigliamo di farlo per lo meno prima di aver letto il capitolo sulla progettazione dei programmi in lin-

guaggio macchina.

Le istruzioni presentate in questo capitolo permettono di abbattere altri ostacoli in un sol colpo e piu' velocemente di una palla da cannone; in altri termini queste sono istruzioni che permettono di manipolare interi blocchi di dati e non un singolo byte per volta.

La piu' semplice di queste istruzioni si presenta nella forma:

CPI

e la vostra conoscenza dell'Assembler dello Z80 vi avra' gia' fatto intuire che questa istruzione ha a che fare con i confronti: ed in effetti proprio di confronti si tratta!

Il significato dell'istruzione CPI e' infatti il seguente: "confronta e incrementa" e non mancheremo di far notare che anche in questo caso uno degli operandi e' contenuto nel registro A anche se cio' appare sottinteso.

Orbene, l'istruzione CPI confronta il valore contenuto in A con quello della locazione puntata da HL ed incrementa automaticamente il valore del puntatore HL. Pertanto al termine dell'operazione la coppia di registri HL punta alla locazione di memoria successiva e quindi ripetendo ciclicamente l'istruzione CPI si possono testare piu' locazioni di memoria tra loro contigue.

Questo tipo di istruzione puo' essere utilizzato, ad esempio, per ricercare all'interno dell'intera memoria un certo valore.

```
Ricerca      CPI
              JR  NZ, Ricerca
```

In questo modo il programma continua a ciclare testando il contenuto delle varie locazioni di memoria finche' non ne trova una in cui e' contenuto il valore desiderato (il flag di zero viene alterato dalla CPI nello stesso modo di tutte le altre operazioni di confronto).

Sfortunatamente per noi questa routine, concettualmente molto semplice, presenta un grosso inconveniente: il programma continua a ciclare finche' non incontra il valore desiderato e pertanto, se questo non esiste il ciclo non ha mai fine. Per nostra fortuna pero' l'istruzione CPI fornisce altri elementi che ci permettono di porre delle condizioni di fine ciclo, dato che con essa viene anche automaticamente decrementato il valore della coppia di registri BC.

Questo fatto puo' essere infatti sfruttato per definire all'inizio del ciclo il numero massimo di volte che questo deve essere eseguito. Ipotizzando ad esempio che il numero di locazioni di memoria da testare sia stato memorizzato nel registro C e che quindi sia inferiore a 256, la routine di ricerca puo' essere sviluppata nel modo seguente:

Ricerca	CPI
	JR Z, Trovato
	INC C
	DEC C
	JR NZ, Ricerca
Non trovato

Trovato

Nel caso in cui invece il numero di iterazioni debba essere superiore a 255, le istruzioni per effettuare il controllo di fine ciclo dovranno essere leggermente modificate. Vogliamo comunque porre in risalto l'uso consecutivo delle due istruzioni INC C e DEC C che permette di verificare se il valore di C e' zero. Queste due istruzioni ad 1 byte infatti influenzano entrambe il flag di zero e, dato che con un incremento ed un immediato decremento si ripristina in C il valore iniziale, alla fine di queste due istruzioni il flag di zero risultera' settato solo se dopo l'istruzione di CPI il registro C conteneva il valore 0. L'uso di questa coppia di istruzioni e' di elevata praticita' dato che permette di controllare il contenuto di un registro senza modificarlo e, cosa ben piu' importante, senza modificare neppure il contenuto degli altri registri.

Lo Z80 mette poi a disposizione un'istruzione assai simile alla CPI che permette di testare un blocco di locazioni partendo dalla fine anziche' dall'inizio.

Questa istruzione che si presenta nella forma

CPD

(confronta e decremента) opera nel modo seguente: esegue il confronto tra il valore di A e quello della locazione puntata da HL, decremента il valore di HL e decremента anche quello di BC.

Ma non basta: vi sono nel linguaggio dello Z80 delle istruzioni di confronto ancora piu' potenti! Esse sono:

CPIR
CPDR

con l'ovvio significato di "confronta, incrementa e ripeti"
e "confronta, decremента e ripeti"

Queste due istruzioni a 2 byte sono incredibilmente potenti: esse infatti autorizzano la CPU a continuare automaticamente la ricerca allo interno del blocco finche' o viene trovato il valore desiderato oppure viene ultimato il blocco. Naturalmente prima di questa istruzione devono essere inizializzati sia A che le coppie di registri HL e BC.

Dato che queste istruzioni prevedono l'uscita dal ciclo in due condizioni diverse (o si e' trovato il valore o si e' giunti alla fine/inizio del blocco) dovremo preoccuparci di verificare quale delle due condizioni si e' verificata.

Un altro fattore di cui bisogna tener conto e' la lentezza con cui queste istruzioni vengono eseguite; lentezza ovviamente in confronto ai tempi del computer. L'istruzione CPIR, ad esempio, richiede ben 21 cicli di clock per effettuare il test su un singolo byte. Supponendo anche che la macchina operi con una frequenza di 3500000 cicli al secondo, la ricerca all'interno di un blocco di 3500 bytes richiederebbe pur sempre un cinquantesimo di secondo.

Parlare in questo caso di lentezza fa un po' sorridere, dato che un cinquantesimo di secondo e' comunque un tempo molto piccolo, ma se pensate che per mantenere visibili i caratteri sullo schermo la CPU deve illuminarlo proprio ogni cinquantesimo di secondo, vi accorgete che tempi di questo genere non sono per nulla trascurabili.

Vi sono infine delle operazioni che permettono di copiare blocchi di dati in altre zone di memoria:

LDI	LDIR
LDD	LDDR

che andranno ovviamente catalogate tra le istruzioni di caricamento. Nell'ordine esse vanno dunque lette nel modo seguente:

"Carica e incrementa"
"Carica, incrementa e ripeti"
"Carica e decrementa"
"Carica, decrementa e ripeti"

Facendo riferimento ora alla prima di esse, vediamo in dettaglio cosa avviene nel corso della sua esecuzione:

Carica (DE) con (HL)
Incrementa DE e HL
Decrementa BC

Notiamo per inciso che questo e' l'unico insieme di istruzioni che permette di spostare direttamente un valore da una locazione di memoria ad un'altra senza passare attraverso un registro.

I progettisti sono stati veramente furbi ad utilizzare la coppia di registri DE per rappresentare l'indirizzo della locazione in cui deve essere copiato il dato, visto che gia' il loro nome richiama il concetto di destinazione!

L'istruzione LDD opera in modo analogo alla precedente solo che anzic-

che' incrementare ad ogni passo DE e HL, li decremen- ta. Questa differenza risulta importante quando si ha a che fare con due aree di memoria (quella di partenza e quella di arrivo) parzialmente sovrapposte.

L'esempio seguente vuole mettere in risalto questo fatto e contemporaneamente illustrare il modo di procedere di queste istruzioni.

Supponiamo di voler utilizzare l'istruzione LDI per un programma di word processing in cui si vuole cancellare una parola da una frase. Consideriamo la frase originale:

Il grosso cane nero salta addosso alla volpe.
1 3 5 7 9 1 3 5 7 9 1 3 5 7 9 1 3 5 7 9 1 3 5

e supponiamo di voler cancellare la parola "nero". Per far cio' dobbiamo spostare indietro di cinque caratteri tutta la parte della frase che segue la parola "nero". Inizializziamo allora i registri DE e HL nel modo seguente:

DE = destinazione = carattere numero 16
HL = partenza = carattere numero 21
BC = contatore = 25 caratteri

Utilizziamo ora l'istruzione LDI. Il suo effetto e' illustrato dallo schema seguente:

Prima : Il grosso cane nero salta addosso alla volpe.
Spostamento: s(---s
Dopo : Il grosso cane sero salta addosso alla volpe.

Eseguido ancora quattro volte l'istruzione LDI si ottiene:

Il grosso cane saltasalta addosso alla volpe.

Alla fine del ciclo completo di spostamenti si otterra':

Il grosso cane salta addosso alla volpe.olpe.

Per eliminare gli ultimi 5 caratteri ormai superflui avremmo potuto aggiungere 5 spazi bianchi in coda alla frase originale ed inizializzare il contatore a 30.

Se ora vogliamo ripristinare la situazione di partenza,ricostruendo lo spazio necessario per inserire la parola "nero", non possiamo utilizzare l'istruzione LDI in modo analogo al precedente, dato che il blocco di arrivo contiene dati che a loro volta dovrebbero poi essere spostati.

Infatti se inizializziamo i registri con i seguenti valori:

HL = partenza = Carattere 16
DE = destinazione = Carattere 21
BC = contatore = 25 caratteri

l'istruzione LDI produrrebbe il seguente effetto:

Prima : Il grosso cane salta addosso alla volpe.olpe.
Spostamento: s--->s
Dopo : Il grosso cane saltasaddosso alla volpe.olpe.

E dopo 5 iterazioni la situazione sarebbe la seguente:

Il grosso cane saltasaltasso alla volpe.olpe.

e cio' a prima vista puo' sembrare anche corretto. Ma dopo altre 5 iterazioni, ahime' il risultato e' il seguente:

Il grosso cane saltasaltasaitalla volpe.olpe.

Questo strano risultato e' dovuto al fatto che spostando i caratteri siamo andati a scrivere sopra altri caratteri prima che questi fossero interessati a loro volta al trasferimento.

Se non siete convinti di cio' provare ad eseguire manualmente le operazioni utilizzando un foglietto per ciascun carattere.

Per essere sicuri, usando l'istruzione LDI di non andare a cancellare dei caratteri significativi, avremmo dovuto fissare il puntatore DE alla fine della frase.

Esercizio

Scrivere una piccola routine che permetta di trasferire sullo schermo il contenuto dei primi 32 bytes della ROM dello Spectrum.

Notate che essi compaiono sul video nello stesso ordine.

Provate ora a ripetere l'esperimento per 256 bytes e per 2048.

PARTE SECONDA

LE ISTRUZIONI DELLO Z80 UTILIZZATE MENO FREQUENTEMENTE

=====

SCAMBIO TRA REGISTRI

Nella prima parte del presente volume abbiamo parlato, seppur brevemente, dei guanti che la CPU utilizza per il proprio lavoro, e di come essa possa conservare delle informazioni in posti piu' accessibili delle normali locazioni di memoria infilandoli e togliendoli.

Dovete ricordare che, mentre le mani della CPU rappresentano i registri normali, non e' possibile manipolare i valori dei registri alternativi rappresentati dai guanti. Da questa considerazione risulta azzeccata l'analogia fatta con gli stessi: infatti i guanti possono mantenere la loro forma ma non possono modificare la posizione delle dita se non sono calzati su di una mano.

Per poter operare sui registri alternativi occorre scambiarli con quelli usati normalmente dalla CPU (e' necessario che la CPU si cambi i guanti per poter operare su quelli di scorta).

La prima istruzione che permette di scambiare coppie di registri e' la seguente:

EX AF,AF'

Questa istruzione va interpretata nel seguente modo: "Scambia (Exchange) la coppia di registri AF con la coppia di registri AF'". Riprendendo l'analogia con i guanti cio' equivale a cambiare i guanti sulla coppia di mani AF. In altre parole, la CPU sfila dalle mani AF il solito paio di guanti ed infila quello di riserva (che vi ricordiamo di aver chiamato AF').

L'istruzione che permette di scambiare tra loro i guanti di tutte le altre mani e':

EXX

Essa scambia i registri nel modo seguente:

B	C		B'	C'
D	E	(==>)	D'	E'
H	L		H'	L'

Questa istruzione, a prima vista molto potente, e' poco usata poiche', operando su tutte le coppie, non permette di utilizzare nessuno dei vecchi valori eccetto quello contenuto nel registro A, non coinvolto

in questa operazione.

Volendo conservare il contenuto di una particolare coppia di registri anche dopo lo scambio, si puo' far ricorso allo stack:

```
PUSH HL
EXX
POP HL
```

Con queste istruzioni abbiamo salvato il contenuto di BC, DE ed HL nei registri alternativi ed abbiamo ripristinato il valore in HL per i lavori futuri.

Esiste un'altra istruzione di scambio, che pero' non puo' essere considerata come un vero e proprio scambio di guanti, ed e':

```
EX DE,HL
```

Con questa istruzione portiamo il contenuto di DE in HL e quello di HL in DE.

Essa risulta piuttosto utile, dato che alcune istruzioni possono essere eseguite utilizzando solo la coppia privilegiata HL, mentre a volte i valori su cui si vuole operare sono contenuti in DE.



BIT, SET E RESET

=====

Tutte le istruzioni che abbiamo presentato sinora operavano su numeri di 8 o di 16 bit.

Il gruppo di istruzioni di "Bit, Set e Reset", che vi presentiamo in questo capitolo, opera invece su un singolo dito di una mano della CPU (singolo bit di un registro) e/o di una locazione di memoria.

Questo gruppo di istruzioni e' poco usato, essendo noioso operare su un singolo bit.

Per di piu' queste istruzioni, che servono a modificare un solo bit all'interno di un registro o di una locazione di memoria, risultano anche piu' lente rispetto a quelle che permettono invece di cambiare od esaminare l'intero contenuto di un registro ad 8 bit o di una locazione.

Nonostante cio', a volte capita di voler sapere se un particolare bit e' settato o no, oppure succede di dover porre a 1 un particolare bit di un registro.

Notiamo pero' che le operazioni di set e reset su particolari bit puo' essere fatta, come abbiamo visto, anche utilizzando gli operatori logici.

Le istruzioni Bit, Set e Reset, in ogni modo, ci permettono di porre a 1 o a 0 un qualsiasi bit oppure di verificarne lo stato.

Le istruzioni di SET, che permettono di porre ad 1 un determinato bit, sono:

```
SET n, r
SET n, (HL)
SET n, (IX + d)
SET n, (IY + d)
```

In esse n indica la posizione del bit da settare (ricordiamo che i bit sono convenzionalmente numerati da destra a sinistra da 0 a 7) in un registro r o in una locazione di memoria.

Queste istruzioni non alterano il registro dei flag.

Le istruzioni di RESET si presentano nella stessa forma delle prece-
ti e permettono di porre a zero un determinato bit di un registro o di una locazione di memoria.

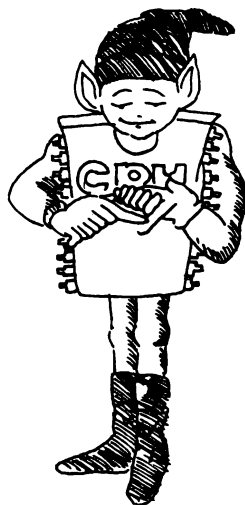
Le istruzioni BIT sono invece istruzioni di test, poiche' servono a specificare in quale stato si trova un determinato bit.
Esse non cambiano il contenuto del registro o della locazione a cui il

bit appartiene ma operano sul flag di zero nel modo seguente:

Se il bit testato e' 0 allora il flag di zero e' posto a 1

Se il bit testato e' 1 allora il flag di zero e' posto a 0

Questo modo di operare puo' a prima vista generare confusione, ma vi apparira' tutto chiaro, se lo descrivete cosi': "se il bit e' zero l'indicatore di zero viene alzato (ricordiamo che abbiamo paragonato il flag ad una bandierina!) mentre se il bit non e' zero il flag rimane abbassato".



ROTATE e SHIFT

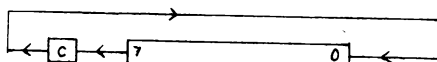
=====

E' possibile spostare i bit di un registro in senso rotatorio da sinistra a destra e da destra a sinistra oppure traslarli semplicemente di un posto in avanti ed indietro.

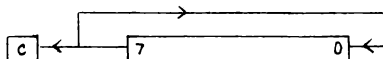
Un modo per classificare le varie istruzioni di rotazione e traslazione consiste nel considerare il modo in cui esse operano con il flag di carry, che in queste operazioni viene considerato un nono bit supplementare (il bit numero 8, secondo la convenzione di numerare i bit da 0 a 7) del registro in esame.

Alcune istruzioni di "rotate" coinvolgono infatti nelle rotazioni anche il flag di carry, di modo che il ciclo completo di rotazioni avviene su 9 bit.

La figura seguente mostra ad esempio come opera l'istruzione RLA (il significato di ogni istruzione dovrebbe esservi chiaro alla fine di questo capitolo):



Altre istruzioni di "rotate" invece coinvolgono nella rotazione solo gli 8 bit del registro. In questo caso il flag di carry assume il valore del bit che "deve fare piu' strada nel corso della rotazione". La figura seguente mostra ad esempio come opera l'istruzione RLCA.



In questa rotazione verso sinistra il bit 0 e' trasferito al posto 1, il bit 1 al posto 2 e cosi' via. Il bit 7 e' trasferito sia nel flag di carry sia nella posizione 0.

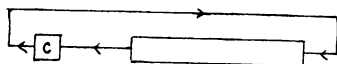
Nell'istruzione RLA invece il bit 7 era trasferito nel carry e nella posizione 0 era trasferito il carry.

Rotazioni a sinistra

Vi sono, come abbiamo visto, due tipi fondamentali di rotazioni:

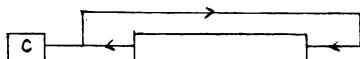
* ROTATE LEFT REGISTER in cui la rotazione avviene su nove bit come illustrato per RLA

RLA - "Rotate Left Accumulator"
RL r - "Rotate Left Register "



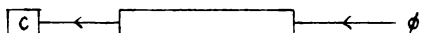
* ROTATE LEFT CIRCULAR - la parola "circular" sta ad indicare che la rotazione avviene solo sugli 8 bit del registro, come illustrato per l'istruzione RLCA.

RLCA - "Rotate Left Circular A"
RLC r - "Rotate Left Circular register"
RLC (HL) - "Rotate Left Circular (HL)"
RLC (IX + d) - "Rotate Left Circular (IX + d)"
RLC (IY + d) - "Rotate Left Circular (IY + d)"



Mentre esistono piu' modi per effettuare la rotazione da destra a sinistra, vi e' una sola istruzione che permette la traslazione a sinistra e puo' essere applicata solo al registro A:

SLA - "Shift Left Accumulator"

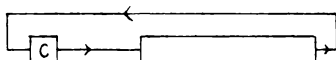


Questa istruzione fa si' che il bit 7 venga trasferito nel carry, che i bit da 0 a 6 vengano traslati in avanti di un posto e che il bit 0 assuma il valore 0. Cio' equivale a moltiplicare per 2 il valore contenuto originariamente nell'accumulatore, assegnando al carry un eventuale riporto. (Provate ad eseguire l'operazione di SLA con il valore 80H).

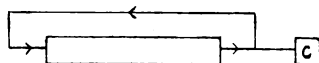
Rotazioni a destra

Anche in questo caso abbiamo due tipi di rotazione, ma questa volta essa avviene in senso antiorario. Le forme in cui si possono presentare le istruzioni di rotazione a destra sono analoghe a quelle per la rotazione a sinistra.

- RRA - "Rotate Right Accumulator"
- RR r - "Rotate Right register"

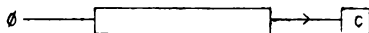


- RRC A - "Rotate Right Circular A"
- RRC r - "Rotate Right Circular register"
- RRC (HL) - "Rotate Right Circular (HL)"
- RRC (IX+d) - "Rotate Right Circular (IX+d)"
- RRC (IY+d) - "Rotate Right Circular (IY+d)"



Esiste anche l'equivalente per la traslazione:

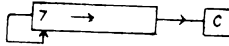
- SRL r - "Shift Right Logical register"



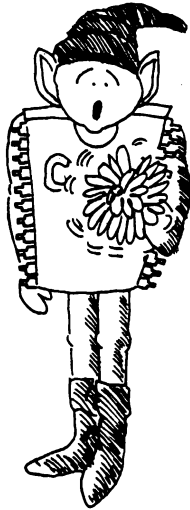
In questo caso l'istruzione rappresenta la divisione per 2 considerando il valore contenuto nel registro come un numero naturale compreso tra 0 e 255.

Dato però che in molte applicazioni si utilizza la convenzione su numeri relativi (con 8 bit quindi si rappresentano i valori tra -128 e +127), lo Z80 mette a disposizione una nuova operazione di shift a destra che permette di conservare il segno:

SRA r - "Shift Right Arithmetic register"



Anche in questo caso viene effettuata la divisione per 2 ma viene conservato il valore del bit di segno.



IN e OUT

=====

Le due istruzioni di IN e OUT rappresentano concettualmente operazioni piuttosto semplici che sicuramente vi troverete a dover usare una volta o l'altra.

Vi sono infatti casi in cui la CPU deve ricevere informazioni dal mondo esterno ("La CPU non vive forse in un'isola?") come ad esempio quando deve ricevere dati dalla tastiera o dal registratore a cassette.

Questi per la CPU sono lontani territori stranieri e purtroppo essa, come del resto tutte le brave CPU, non vuole mai lasciare la propria casa. Pertanto, si limitera' ad aprire una porta affinche' le siano consegnate le informazioni inviatele. La CPU non sa, ne' deve preoccuparsi di sapere, come lavora il registratore. L'unica cosa che deve conoscere e' a quale porta bussera' il postino per consegnare o ricevere informazioni. Il chip Z80 permette di selezionare ben 256 porte, ma il numero di porte realmente esistente e' in genere minore e dipende da come e' stato costruito l'hardware del computer. Nel caso dello Spectrum Sinclair la CPU puo' comunicare solo con tastiera, video, stampante e registratore.

Altro argomento che non deve interessare alla CPU e' il modo in cui i dati vengono trasmessi. Anche in tal caso l'unica cosa che le interessa e' che i dati che entrano od escono dalla porta abbiano tutti la forma di valori binari a 8 bit (1 byte).

La tastiera ed il registratore sono entrambi al di la' della porta FEH (254 in decimale), cosicche', per acquisire un dato da tastiera, useremo l'istruzione

IN A, (FE)

Utilizzando questa istruzione potete ora provare da soli a vedere qual'e' il codice esadecimale usato dallo Spectrum per rappresentare ognuno dei 40 tasti della tastiera.

La risposta sara' ben diversa da quella che vi aspettate: la tastiera fornisce informazioni su al piu' 5 tasti per volta, e questo perche' e' il valore contenuto in A nel momento in cui viene aperta la porta che determina quale blocco di 5 tasti deve essere esaminato!

La tastiera, in effetti, e' divisa in 4 righe, ciascuna delle quali e' a sua volta divisa in due blocchi di 5 tasti ciascuno:

```
3 ==>   1 2 3 4 5   6 7 8 9 0 (<== 4
2 ==>   Q W E R T   Y U I O P (<== 5
1 ==>   A S D F G   H J K L N/L (<== 6
0 ==>   SFT Z X C V   B N M . SPC (<== 7
```

Potete vedere che vi sono 8 blocchi a ciascuno dei quali e' correlato un bit del registro A.

Quando la porta si apre devono essere settati (=1) tutti i bit di A, fatta eccezione per quello che specifica il blocco di tasti che deve essere letto.

Potete pensare a cio' come a qualcosa molto simile ad una stretta di mano tra la CPU e l'amico del cuore, per cui, quando essa apre la porta perche' ha bisogno di informazioni, la stretta di mano fa capire a quest'ultimo da quale parte della tastiera deve prelevare le informazioni da dare alla CPU.

Cosi', se vogliamo leggere i tasti del blocco "1 2 3 4 5", dobbiamo porre a zero il bit numero tre di A:

A = 1 1 1 1 0 1 1 1 = F7H

Le informazioni su questi tasti vengono rinviate dalla tastiera associando ad ogni tasto uno dei bit piu' bassi del registro A.

Tasto "1" => Bit 0 di A
Tasto "2" => Bit 1 di A

e cosi' via.

Se invece fosse stato selezionato il gruppo 4 della tastiera (A = EFH), si sarebbe ottenuto il seguente schema:

Tasto "0" => Bit 0 di A
Tasto "9" => Bit 1 di A

Come avete visto, le informazioni che entrano in A dall'esterno vengono selezionate prima, dato che ad esempio al bit numero 0 del registro A puo' essere associato indifferentemente sia il tasto "0" sia il tasto "1".

In molti giochi interattivi risulta pero' conveniente poter leggere contemporaneamente tutta una riga di tasti anziche' un blocco di 5 tasti alla volta.

Si puo' ottenere cio' aggirando l'amico del cuore della CPU, in modo che trasmetta contemporaneamente due blocchi di informazioni. E' sufficiente cioe', quando si selezionano i blocchi, porre a zero due bit del registro A, e non uno solo:

A = 1 1 1 0 0 1 1 1 = E7H

In questo caso i bit azzerati sono il numero 3 ed il numero 4.

Una stretta di mano di questo tipo dice all'amico del cuore che la CPU

ha bisogno delle informazioni relative a due blocchi. Pertanto corre a prelevarle da entrambi i blocchi interessati, consegnandole pero' mescolate tra loro, per cui non vi e' possibile saper se si tratta del tasto "0" o del tasto "1", essendo entrambi associati allo stesso bit.

"1" o "0" -> Bit 0 di A

"2" o "9" -> Bit 1 di A

e cosi' via.

Nonostante cio', poter testare due blocchi contemporaneamente risulta di notevole utilita' specialmente nei giochi di movimento, in quanto permette ad esempio di utilizzare i tasti "8" e "5", che appartengono a blocchi diversi della tastiera, per rappresentare spostamenti verso destra e verso sinistra.

Se utilizziamo l'istruzione

IN r, (C)

(anziche' IN A, (porta)), dove C sta ad indicare la porta selezionata, e' B a specificare il blocco della tastiera che deve essere letto.

Le altre porte che possono interessarci sono quelle collegate con la trasmissione in ingresso od in uscita di dati da/per il registratore a cassette.

Come abbiamo gia' detto si tratta ancora della porta FE. In questo caso il maggior problema da affrontare e' quello della corretta sincronizzazione per le operazioni di lettura e scrittura. Questo tipo di problema puo' essere affrontato solo dopo aver maturato una buona esperienza di programmazione in linguaggio macchina, poiche' richiede il calcolo preciso dei tempi necessari sia per l'esecuzione delle singole operazioni, sia per le operazioni meccaniche di lettura e scrittura di un dato da cassetta.

L'istruzione OUT e' usata anche per generare suoni o per scegliere il colore dello sfondo.

Nel manuale dello Spectrum viene presentato il comando OUT tra i comandi BASIC. Orbene, la programmazione in Assembler del comando OUT e' esattamente la stessa. In altre parole, i bit 0, 1 e 2 definiscono il colore, il bit 3 manda un segnale agli attacchi per MIC e EAR mentre il bit 4 manda un segnale all'altoparlante interno.

Per cambiare il colore di sfondo basta caricare in A il valore associato al colore voluto e poi eseguire l'istruzione OUT (FE), A. Notiamo che cio' comporta solo un cambiamento TEMPORANEO del colore. Se si vuole ottenere una variazione permanente non solo bisogna usare l'istruzione OUT, ma variare anche il contenuto della locazione di memo-

ria 23624, che e' quella in cui il sistema operativo ha allocato la variabile BORDCR (vedi manuale dello Spectrum).

La ragione di cio' sta nel fatto che e' l'hardware dello Spectrum (il chip ULA) che controlla il colore di sfondo e questo chip riceve le sue informazioni guardando il contenuto di questa particolare locazione di memoria.

E' possibile interrompere l'azione dell'hardware sul colore solo disabilitando tutti gli interrupt (istruzione DI). Notiamo a tale proposito che le interruzioni vengono riabilite automaticamente da alcune delle routines di ROM (istruzione EI).

Generazione di suoni

Sullo Spectrum potete generare delle musiche, ma fate attenzione che vi sono delle limitazioni nei suoni generabili, specie nella versione a 16K, dovute alle caratteristiche dell'hardware.

Poiche' il video deve essere costantemente illuminato, l'hardware interrompe regolarmente lo Z80 per fargli eseguire la routine di visualizzazione di cio' che e' contenuto nel display file, generando cosi' delle routines di ritardo nel programma (interruzione di tipo WAIT).

Cio' produce l'effetto di rendere impossibile la realizzazione di programmi che richiedono una perfetta sincronizzazione, poiche' non e' possibile prevedere l'effetto prodotto da queste interruzioni di tipo WAIT sui tempi.

Fortunatamente lo Spectrum a 48K e' stato progettato in modo tale che lo Z80 possa venir interrotto solo se sta eseguendo un programma contenuto nei primi 16K di memoria, mentre se il programma ed i dati a cui lo Z80 deve accedere sono contenuti su ROM oppure in locazioni disposte oltre i 32K di memoria le interruzioni non intervengono.

Si puo' riassumere tutto cio' in questo modo: se disponete solo di uno Spectrum a 16K, potete produrre musica ugualmente con il comando OUT, ma sappiate che il suono non risultera' perfetto. In alternativa, potete utilizzare la routine di sistema BEEP presentata nel capitolo dedicato all'architettura dello Spectrum.

Per creare musica e' necessario inviare un impulso che azioni l'altoparlante (e/o le connessioni MIC, se disponete di un amplificatore), e dopo un brevissimo intervallo di tempo e' necessario inviarne un altro per disattivarlo. E cosi' di seguito: un impulso per attivarlo ed uno per disattivarlo,

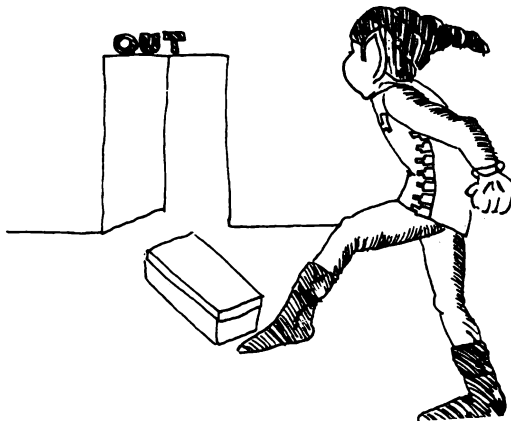
Potete generare in tal modo suoni diversi, poiche' la frequenza del suono stesso dipende dal periodo di tempo intercorrente tra due segnali di attivazione dell'altoparlante. La lunghezza del periodo di tempo

in cui viene lasciato in ON l'altoparlante (che corrisponde poi al tempo intercorrente tra l'impulso ON e quello OFF) vi permette invece di mantenere un grado minimo di controllo sul volume del suono.

Fate attenzione ai valori di A che utilizzate per lanciare i segnali di ON e di OFF, poiché potrebbero anche modificare lo sfondo dello schermo. Il susseguirsi delle note dovete altrimenti vederlo come un insieme di caricamenti di particolari valori in A. Lo spartito diventa quindi un programma!

Esercizio

Scrivete una routine che simuli il suono di una sirena d'ambulanza (alternarsi di frequenze basse ed alte). Fate attenzione a mantenere la stessa frequenza per un certo periodo di tempo, prima di passare alla frequenza successiva.



RAPPRESENTAZIONE BCD

=====

Il codice BCD (Binary Coded Decimal) permette di rappresentare informazioni in formato decimale.

Per codificare le dieci cifre decimali sono sufficienti quattro bit e tra le possibili configurazioni che essi possono assumere solo 10 vengono utilizzate in BCD, mentre le altre sei rappresentano codici non validi.

Da cio' consegue che in un byte possono essere rappresentate due cifre decimali. La rappresentazione BCD per l'appunto associa ad ogni cifra decimale il corrispondente valore binario espresso su quattro bit.

Ad esempio:

00000000 e' la rappresentazione BCD del valore decimale 00.
10011001 e' la rappresentazione BCD del valore decimale 99.

Provate ora ad individuare il codice BCD dei valori decimali 58 e 10. La configurazione binaria 10100000 rappresenta secondo voi un codice utilizzabile in BCD?

Aritmetica BCD

Questo nuovo modo di rappresentare i dati genera numerosi problemi nelle operazioni aritmetiche di addizione e sottrazione. Proviamo ad esempio ad eseguire in BCD la somma tra 8 e 3:

BCD	08	0000 1000	+
BCD	03	0000 0011	=

BCD	11	0000 1011	

Appare evidente che il risultato binario dell'operazione non coincide con il codice BCD del numero 11. Per ottenere il risultato corretto dovremmo utilizzare una particolare istruzione di "correzione" detta DAA (Decimal Adjust Addition) che somma 6 al risultato dell'addizione se il valore degli ultimi 4 bit supera 9.

L'esempio precedente mostra che nella rappresentazione BCD occorre tener conto di un eventuale riporto intermedio (half carry) derivante dall'addizione tra i quattro bit bassi degli operandi, e che tale riporto deve poi influenzare il valore del primo bit del gruppo dei quattro bit piu' significativi.

In altre parole, eseguendo le operazioni in BCD un eventuale riporto sulla prima cifra BCD deve essere addizionato al valore ottenuto per

la seconda.

Per effettuare questo tipo di operazione bisogna tener conto del valore assunto dal flag H (half carry) nel corso della stessa.

```
LD A, 12H      ;carica in A il valore BCD 12
ADD A, 24H     ;somma il valore BCD 24
DAA           ;eventuale correzione del risultato
LD (indirizzo),A;memorizzazione risultato
```

E' molto improbabile che nei vostri programmi vi troviate nella necessita' di operare in codice BCD, ma per completezza abbiamo voluto per lo meno accennare al fatto che lo Z80 supporta questo tipo di rappresentazione e che il suo linguaggio comprende anche istruzioni, quali la DAA, che permettono di manipolare valori espressi in tal modo.

INTERRUZIONI

=====

Una interruzione e' un segnale inviato al microprocessore, il quale generalmente produce una sospensione del programma in esecuzione senza che il programma stesso se ne accorga.

Lo Z80 e' abilitato a ricevere tre tipi di interruzione: richieste provenienti dal bus (BUSRQ), interrupt non mascherabili (NMI) ed interruzioni semplici (INT).

Per quanto riguarda la programmazione, le uniche interruzioni che ci interessano sono quelle di tipo mascherabile (INT).

Esistono due istruzioni che permettono di mascherare questo tipo di interruzione: l'istruzione DI (disable interrupt) che disabilita le interruzioni (mascheramento) e l'istruzione EI (enable interrupt) che effettua l'operazione inversa.

In genere un'interruzione ordinaria genera un salvataggio sullo stack del valore del program counter riferito al programma in esecuzione ed un salto alla pagina zero della ROM con un'azione tipica di RST. La routine di gestione dell'interruzione deve terminare con un'istruzione di RETI (return from interrupt).

Nel corso normale delle operazioni lo Spectrum si trova nello stato EI (abilitazione degli interrupt), infatti, come abbiamo gia' avuto occasione di affermare, un programma utente viene interrotto 50 volte al secondo solo per permettere l'esecuzione della routine di sistema operativo predisposta a mantenere l'illuminazione del video.

Per rendere piu' veloce l'esecuzione del vostro programma potete disabilitare le interruzioni. In tal modo pero' non potete usare la tastiera finche' il programma e' in esecuzione. Non solo, ma dovete anche ricordarvi di riabilitare le interruzioni alla fine del vostro programma, altrimenti il sistema rimane nell'impossibilita' di ricevere segnali dalla tastiera!

ISTRUZIONI DI RESTART (RST)

=====

Questa e' una delle eredita' pervenute dal microprocessore 8080, che e' stata mantenuta per garantire la compatibilita'. Cio' significa che nei vostri programmi utilizzerete ben difficilmente questo tipo di istruzione.

L'istruzione RST puo' essere considerata come una chiamata ad una subroutine, con la precisazione che il sottoprogramma chiamato deve necessariamente iniziare in una delle seguenti locazioni disposte tra le prime 256: 00H, 08H, 10H, 18H, 20H, 28H, 30H, 38H.

Poiche' le routines ad essa collegate sono richiamate molto frequentemente, l'istruzione RST, che occupa un solo byte e viene eseguita piu' celermente dell'istruzione CALL, risulta essere notevolmente vantaggiosa. Uno svantaggio consiste invece nel fatto che essa permette di indirizzare solo 8 locazioni di memoria ben precise.

Inoltre, dato che tutte queste locazioni fanno parte di una memoria ROM, non potete usare le RST per richiamare routines scritte da voi. Potete invece usare questa istruzione ogni qual volta vogliate richiamare, nel corso del vostro programma, una delle 8 routines citate, presenti nella ROM. E' ovvio che per questo occorre sapere esattamente che cosa facciano queste routines.

Per saperne di piu' sull'istruzione RST potete leggere il volume del Dr. Ian Logan dal titolo "UNDERSTANDING YOUR SPECTRUM".

P A R T E T E R Z A

PROGRAMMAZIONE DELLO SPECTRUM

=====

PROGETTAZIONE DI UN PROGRAMMA IN LINGUAGGIO MACCHINA

=====

La programmazione in linguaggio macchina e' estremamente flessibile e permette di fare un po' di tutto.

Inoltre va notato che, poiche' tutti i linguaggi ad alto livello derivano in qualche modo dal linguaggio macchina, non esiste programma Fortran o Cobol o BASIC che non possa essere realizzato anche in linguaggio macchina, con l'ulteriore vantaggio di ottenere un'esecuzione molto piu' veloce.

A volte pero' questa enorme flessibilita' puo' trasformarsi per un programmatore incauto in un trappola, poiche' da' la liberta di operare come meglio credete.

Ad esempio, non vi e' alcun blocco dovuto a test sulla correttezza lessicale e sintattica delle istruzioni, contrariamente a quanto succede invece per l'interprete BASIC dello Spectrum.

Poiche' nella programmazione in linguaggio macchina trasmettete numeri che dovrebbero rappresentare istruzioni di un tipo o di un altro, la CPU ve li elaborera' in ogni caso.

Ma la programmazione in linguaggio macchina, oltre al controllo sulla correttezza sintattica delle istruzioni, non vi pone vincoli sulla impostazione logica del programma; potete infatti usare a piacimento funzioni, salti, ecc. che potrebbero risultare vietati dalla sintassi dei linguaggi ad alto livello.

Per questo motivo e' estremamente importante imporsi un'autodisciplina nella stesura dei programmi in linguaggio macchina. Non insisteremo mai abbastanza nell'invito ad utilizzare un approccio ai problemi di tipo "top-down" non solo nella programmazione in generale, ma anche e soprattutto in quella in linguaggio macchina.

L'approccio "top-down" vi obbliga a suddividere il problema in tante piccole unita', in modo da poter controllare la logica del vostro progetto senza dover passare attraverso lunghi periodi di codifica.

Per illustrare questo metodo facciamo un esempio.

Supponiamo di dover scrivere un programma per un atterraggio lunare

ISTR	Visualizzazione istruzioni per l'utente Torna indietro ad ISTR finche' non e' battuto il tasto (ENTER)
SCENA	Disegna lo scenario lunare e poni il veicolo in posi- zione di partenza
DISCESA	Fa scendere il veicolo Se il carburante e' finito, allora salta a CRASH Se non ha toccato terra, torna a DISCESA
ATTERRATO	Comunica le tue congratulazioni Salta indietro ad ISTR per una nuova partita
CRASH	Comunica la tua commiserazione per l'errato allunaggio Salta indietro ad ISTR per una nuova partita

Notiamo che questo "programma" e' scritto completamente in italiano, poiche' a questo livello di analisi non e' ancora necessario decidere se verra' poi scritto in BASIC od in linguaggio macchina. Infatti la logica generale del programma di atterraggio lunare non dipende in alcun modo dal linguaggio che verra' in seguito scelto per la codifica.

Passiamo ora ad effettuare un test logico su quanto abbiamo scritto. Mettetevi nei panni del computer e vedete se tutte le possibilita' che potrebbero verificarsi sono state contemplate.

Vi sono salti a qualcosa che pensavamo di scrivere e che poi invece abbiamo dimenticato? Abbiamo scritto tutti i blocchi necessari? Vi e' qualche blocco inutile? Dobbiamo aggiungere qualcosa all'interno di qualche subroutine?

Bene, diamo un'occhiata al nostro programma: oh, oh, ci siamo dimenticati di far terminare prima o poi il programma!

Il programma, come e' stato concepito, ripete automaticamente l'esecuzione dopo ogni partita senza darvi la possibilita', quando siete stanchi del gioco, di interromperlo. Per permettere al giocatore, alla fine di ogni partita, di decidere se giocare ancora o no, modifichiamo l'ultima parte del programma:

ATTERRATO	Comunica le tue congratulazioni Salta a FINE
CRASH	Comunica la tua costernazione per l'errato allunaggio
FINE	Chiedi al giocatore se ha finito di giocare Se no, salta ad ISTR Se si', STOP

Vogliamo farvi notare che i riferimenti (label) che abbiamo usato per contrassegnare alcune linee di programma, per quanto abbreviati, permettono di individuare la funzione che il programma compie alla linea stessa. Cio' risulta molto utile sia per la progettazione dei moduli, sia per la loro correzione.

Ora che abbiamo concluso l'analisi a questo primo livello, passiamo al livello successivo sviluppando di volta in volta ciascuno dei moduli individuati nella fase precedente. Questo tipo di analisi parte quindi da un'analisi sommaria e man mano scende nei dettagli, sviluppandosi quindi "dall'alto in basso" (top-down).

Per proseguire nel nostro lavoro ampliamo ad esempio il modulo di FINE:

```
FINE          Cancella il video
              Comunica "Avete finito di giocare?"
              Ricevi la risposta da tastiera
              Se risposta = si', allora STOP
              Salta ad ISTR
```

Osserviamo per inciso che sviluppando l'analisi col metodo top-down potete controllare e provare separatamente ogni singolo modulo non appena e' finito, in modo da trovarlo pronto al momento di costruire il programma completo.

Scendiamo ancora ad un livello piu' basso di astrazione e sviluppiamo la linea:

```
          Cancella il video
```

A questo livello di analisi si rende indispensabile decidere in quale linguaggio vogliamo scrivere il nostro programma (ovviamente noi scegliamo il linguaggio macchina del Sinclair).

Se avessimo scelto l'interprete BASIC, la funzione di cancellazione del video sarebbe stata di semplice realizzazione:

```
900 CLS
```

Per la programmazione in linguaggio macchina, invece, la frase "Cancella il video" e' piuttosto ingannevole. Non si tratta infatti di una vera e propria cancellatura, ma di assegnare ad ogni posizione del video il carattere "spazio bianco" (blank).

La routine di "cancellazione" puo' quindi essere sviluppata come segue:

```
CANCELLA      Cerca l'inizio dell'area di memoria destinata al
              display file
              Riempi le successive 6144 posizioni con dei "blank".
```

Come vedete, non abbiamo ancora fatto alcuna codifica, ma lo sviluppo ha tenuto conto del fatto che la codifica finale avverra' in linguaggio macchina.

Prima di passare alla realizzazione di questa semplice routine richiamiamo alcune informazioni che si possono trovare nel manuale dello Spectrum.

L'insieme di dati che forniscono l'immagine del video in memoria e' formata da 6144 bytes che costituiscono il display file e da altri 768 bytes che costituiscono il file degli attributi e descrivono il colore di stampa, quello di sfondo e cosi' via.

La routine di cancellazione da noi progettata teneva conto del display file ma non agiva in alcun modo nel file degli attributi e quindi, variando all'interno del video il colore di sfondo (paper colour) od attivando le funzioni di flash o di bright (vedi manuale dello Spectrum) per qualche posizione, la cancellazione sarebbe avvenuta in modo imperfetto.

Perche' tale operazione risulti corretta occorre introdurre nella routine di cancellazione anche un blocco che uniformi il file degli attributi. (Vedete come e' complesso realizzare una routine di questo tipo in linguaggio macchina, routine che invece in BASIC risultava addirittura banale).

Tenendo conto di quanto sopra detto, riscriviamo il nostro programma nel seguente modo:

```
CANCELLA      Cerca l'inizio del file display
              Poni nelle successive 6144 locazioni dei "blank"
              Cerca l'inizio del file degli attributi
              Poni nelle successive 768 locazioni il valore relativo al
              colore voluto.
```

E finalmente siamo giunti all'ultimo stadio: quello della codifica. Anche in questo caso sviluppiamo solo una parte della routine di cancellazione: quella che permette di riempire il video di spazi bianchi.

```
CANCELLA      LD   HL,VIDEO          ;inizio del display file
              LD   BC,6144          ;numero di byte da "cancellare"
              LD   D,0              ;D="blank"
CICLO          LD   (HL),D          ;caricamento del "blank"
              INC  HL              ;passaggio alla locazione succes-
              ;siva
              DEC  BC              ;decremento del contatore
              LD   A,B
              OR   C                ;test per vedere se BC = 0
              JR   NZ,CICLO        ;ripeti se il test non ha dato e-
              ;sito positivo
```

Come avete potuto notare questo tipo di analisi permette di scomporre anche problemi molto complessi in moduli abbastanza lineari, che possono essere facilmente codificati in linguaggio macchina.

Da questo breve esempio avrete anche capito il motivo che ha portato ad inventare linguaggi ad alto livello: per rendere molto piu' semplici alcune operazioni che risultano complesse da programmare in linguaggio macchina!

Esercizi:

Vi possono essere modi diversi per realizzare una routine come quella per la cancellazione del video, e pensiamo che realizzare la routine CANCELLA in altro modo possa essere per voi un buon esercizio.

Esercizio 1:

Provate a scrivere il programma per riempire con 6144 blank le posizioni del display file senza utilizzare la coppia di registri BC, ma facendo riferimento solo al registro B ed usando l'istruzione DJNZ.

Esercizio 2:

Provate ora a realizzare lo stesso programma facendo uso dell'istruzione piu' potente LDIR.

Attenzione all'uso dell'istruzione LDIR: infatti non e' necessario, per ottenere l'effetto voluto, che in memoria vi sia gia' da qualche parte un blocco di 6144 bytes contenente blank!

Risposte:

Poiche' vi possono essere piu' soluzioni diverse al problema proposto non e' detto che se la vostra soluzione e' difforme dalla nostra essa sia errata. L'unico test di correttezza possibile consiste nel far eseguire il programma e vedere se fa effettivamente quello che vi eravate proposti! Diamo comunque ora le nostre soluzioni:

Programma con l'uso di DJNZ:

```
CANCELLA    LD  HL,VIDEO          ;
            LD  A,0              ;
            LD  B,24             ;Poni B=24
CICLO EST   PUSH BC              ;salva il valore di B
            LD  B,A              ;Predisponi il contatore per
                                ;256 iterazioni
CICLO INT   LD  (HL),A           ;
            INC HL               ;Carica "blank" in 256 locazio-
                                ;ni
            DJNZ CICLOINT       ;
            POP BC               ;ripristina il valore di B sal-
                                ;vato
            DJNZ CICLOEST       ;ripeti tutto finche' hai fini-
                                ;to
```

Abbiamo cosi' realizzato un ciclo di 6144 iterazioni ripetendo il ciclo interno 24 volte ($24 \times 256 = 6144$).

Note:

Possiamo inizializzare il contatore B a zero per ottenere 256 iterazioni attraverso l'istruzione DJNZ. (Perche'?) Una procedura di questo tipo non e' utilizzata normalmente in un programma; essa infatti risulta conveniente solo se si vuole utilizzare il registro C per altri scopi.

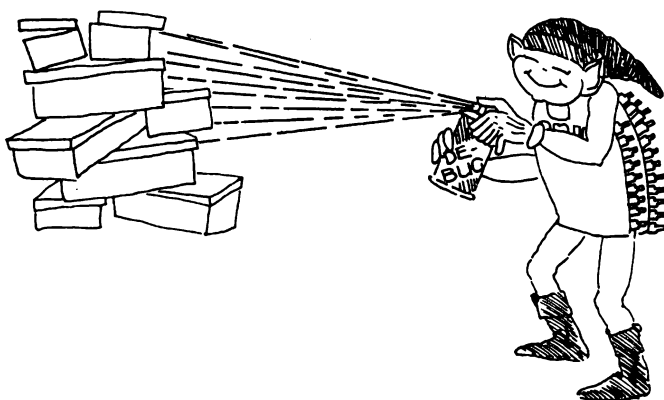
Programma con l'uso di LDIR:

```
CANCELLA    LD HL,VIDEO          ;inizio blocco di origine
             PUSH HL
             POP DE
             INC DE              ;inizio blocco di destinazione
                                 ;DE = HL + 1
             LD BC,6144          ;inizializzazione contatore
             LD (HL),0           ;carica nella prima posizione
                                 ;il "blank"
             LDIR                ;ciclo di spostamento
```

Notiamo che per ottenere $DE = HL + 1$ abbiamo prima posto $DE = HL$ e poi ne abbiamo incrementato il valore. Lo stesso risultato poteva essere raggiunto caricando direttamente in DE l'indirizzo $VIDEO + 1$, ma cio' avrebbe richiesto un byte in piu'!

Questo programma tiene conto del fatto che i due blocchi di origine e di arrivo sono parzialmente sovrapposti e pertanto nel corso dell'iterazione il dato spostato e' sempre un "blank". Abbiamo in pratica usato l'istruzione LDIR come nel problema presentato nel capitolo sui blocchi di dati, traendone pero' questa volta un vantaggio.

Per concludere osserviamo che il primo metodo da noi proposto richiede 14 bytes, il secondo 16 ed il terzo 13.



STRUTTURA DEL SISTEMA ZX SPECTRUM

=====

E' giunto il momento di dare un'occhiata alla struttura interna del nostro calcolatore ZX Spectrum, poiche' da essa possiamo trarre utili informazioni per la programmazione in linguaggio macchina.

Input : Tastiera

Per quanto riguarda l'input, ignoriamo per il momento di proposito la cassetta magnetica e rivolgiamo la nostra attenzione solo alla tastiera.

La tastiera e' l'unico organo di input che garantisca una comunicazione real-time. Essa puo' influire sull'esecuzione di un qualsiasi programma, sia esso di sistema operativo (posto su ROM) oppure utente (posto su RAM).

Logicamente possiamo vedere la tastiera come una matrice bidimensionale formata da 8 righe e 5 colonne, come illustrato nell'appendice A. Ognuna delle quaranta intersezioni rappresenta un tasto della tastiera. Normalmente (quando non vi sono tasti schiacciati) essi si trovano nello stato alto, il che equivale a dire che ciascuna intersezione assume il valore logico 1.

Quando viene premuto un tasto, se non addirittura "pressato", l'intersezione ad esso corrispondente sulla matrice passa allo stato basso, che corrisponde al valore logico 0.

Conoscendo la relazione esistente tra la tastiera e la sua rappresentazione matriciale interna, possiamo trovare un modo logico per controllare lo stato della tastiera, che possa essere utilizzato nella programmazione in linguaggio macchina.

In BASIC per leggere la tastiera si poteva utilizzare la funzione IN, la quale aveva come argomento l'indirizzo della mezza fila di tasti in cui erano compresi quelli da controllare (vedi manuale dello Spectrum, capitolo sulle porte di INPUT/OUTPUT).

In modo analogo, in un programma in linguaggio macchina occorrera' caricare nell'accumulatore il valore corrispondente all'indirizzo della mezza fila di tasti che si vuole controllare. Questi particolari valori, che servono a selezionare blocchi di tasti, sono inseriti nella colonna estrema a sinistra della tabella in appendice A.

Ad esempio, per la mezza riga che va dal tasto H al tasto ENTER il valore da caricare in A e' BFH.

LD A, BFH

Il valore di A viene quindi usato per individuare il byte che contiene lo stato dei tasti di quella particolare mezza riga, e viene trasmesso ad A quando viene eseguita l'istruzione di INPUT. Nel caso della tastiera, la porta utilizzata e' la FEH. L'istruzione di INPUT pertanto risulta essere:

```
IN A, (FEH)
```

Essendoci cinque tasti per ogni mezza riga, risultano significativi solo i cinque bit piu' bassi del valore trasmessa ad A.

Se in quella mezza riga non risulta premuto nessun tasto, il valore dei primi cinque bit di A sara' dato da

$$(2^{*4} + 2^{*3} + 2^{*2} + 2^{*1} + 2^{*0}) = 16 + 8 + 4 + 2 + 1 = 31.$$

```
registro A = xxx11111      quando non sono stati premuti tasti
```

Per verificare se il tasto all'estrema destra e' premuto dovremo vedere se il bit numero zero e' allo stato basso.

Vi sono due modi per effettuare questo controllo:

- 1) Utilizzare l'istruzione BIT (per esempio BIT 0,A)
Se il bit e' allo stato basso il flag di zero viene posto a 1.
- 2) Utilizzare l'istruzione logica AND (per esempio AND 1)
Se il bit e' allo stato basso, il risultato dell'operazione e' 0 e pertanto il flag di zero risulta settato.

Il primo metodo risulta conveniente quando si puo' specificare direttamente nell'istruzione BIT il particolare bit che si vuole controllare. Esso non si puo' utilizzare pero' nel caso si vogliano controllare contemporaneamente gli stati di due bit appartenenti alla stessa mezza riga della tastiera, poiche' non possono essere attivate contemporaneamente due istruzioni di Bit-test e due diversi salti condizionati. Quindi, se vogliamo controllare lo stato dei due bit 0 ed 1 dobbiamo operare nel modo seguente:

```
BIT 0, A      ;il bit 0 di A e' settato o no?  
JR Z, NOSET   ;salta se non e' settato  
BIT 1, A      ;il bit 1 di A e' settato o no?  
JR Z, NOSET   ;salta se non e' settato
```

```
.....  
.....
```

```
blocco di istruzioni da eseguire se entrambi i bit sono  
settati
```

```
.....  
NOSET  
.....  
.....
```

Il secondo metodo presentato richiede un ragionamento logico un po' piu' sottile. Per vedere il valore del bit 0 utilizzeremo l'istruzione

AND 1; per vedere il valore del bit 1 useremo l'istruzione AND 2; per vedere il valore del bit 2 utilizzeremo l'istruzione AND 4, e così via.

Per controllare il valore di due bit contemporaneamente useremo l'istruzione "AND x", dove x rappresenta il valore ottenuto sommando il valore reale dei singoli bit interessati all'operazione.

Per esempio, per testare lo stato dei bit 0 e 1 di A useremo l'istruzione AND 3.

In particolare, per controllare se entrambi i bit 0 ed 1 sono settati (tasti non pressati) useremo le istruzioni:

```
AND    3           ;controllo sui bit 0 ed 1 di A
CP     3           ;sono entrambi settati (=1)
JR     NZ    NO    ;salto se non lo sono
...
...
```

Per vedere invece se almeno uno e' posto sullo stato alto possiamo usare la forma:

```
AND    3           ;test sullo stato dei bit 0 ed 1
JR     Z,NESSUNO   ;salto se neppure uno e' settato
...
...
```

Esercizio

Per ricapitolare quanto abbiamo detto sulla tastiera, potete costruire una subroutine in linguaggio macchina che si blocchi finché non viene premuto il tasto <ENTER>

Questo programma deve contenere le istruzioni che:

- permettano di selezionare la mezza riga in cui e' posto il tasto <ENTER>
- selezionino per l'INPUT la porta FEH
- controllino se il bit associato al tasto <ENTER> ha il valore 0 od il valore 1

Output - Immagine video

Il video e' sicuramente lo strumento principale usato dal computer per comunicare con voi.

Il seguente programma in linguaggio macchina vi permettera' di focalizzare il modo in cui sono organizzate le informazioni relative al video all'interno dello Spectrum.

```

210040 LD HL,4000H ;poni in H l'indirizzo della prima lo-
;cazione assegnata al display file
36FF LD (HL), FFH ;riempi quella locazione del display
;file
110140 LD DE,4001H ;poni in DE l'indirizzo della seconda
;locazione del display file
010100 LD BC,1 ;contatore dei bytes che devono essere
;trasferiti
ED80 LDIR ;trasferisci il blocco di lunghezza BC
;da (HL) a (DE)
C9 RET ;fine del programma

```

Caricate il presente programma sul vostro Spectrum e lanciatene l'esecuzione. Vedrete che, come specificato, un solo byte e' stato trasferito da (HL) a (DE).

Cambiamo ora la quarta linea con la seguente: LD BC, 31 (011F00) cosi' che vengano riempiti i primi 32 byte del display file. Vi aspettereste dunque che venga modificata in toto la forma dei caratteri della prima linea dello schermo, invece il risultato e' diverso: modificando i primi 32 bytes viene sovrapposta una riga continua sulla parte alta dei caratteri presenti nella prima riga del video, poiche' questi primi 32 bytes fanno riferimento ciascuno al primo soltanto degli otto bytes assegnati a ciascun carattere della prima riga del video.

Ora cambiamo nuovamente la quarta linea di programma nel seguente modo: LD BC,255 (01FF00). Ancora una volta sarete sorpresi dal risultato. Il byte successivo al trentaduesimo non rappresenta un segmentino sulla seconda riga di punti dello schermo! Esso infatti rappresenta il primo byte relativo al trentatreesimo carattere! E cosi' di seguito fino al 256esimo carattere.

Siete pronti ora a predire cosa succedera' con il prossimo byte? Cambiate il valore di BC con LD BC,2047 (01FF07) e lanciate poi l'esecuzione del programma. Dovreste ottenere che si riempia completamente un terzo del video, ed in particolare quello posto in alto.

Potete procedere in questo modo, usando cioe' valori diversi di BC, per verificare in che modo lo Spectrum organizza l'immagine del video.

La memoria video dello Spectrum e' suddivisa in tre blocchi:

1. da 4000H a 47FFH (===) prime otto righe
2. da 4800H a 4FFFH (===) righe dalla 9 alla 16
3. da 5000H a 57FFH (===) ultime otto righe

Non solo; nello Spectrum ciascun carattere e' realizzato da una matrice di 64 punti rappresentati in 8 byte.

Il carattere "!", ad esempio, ha la seguente rappresentazione:

0	00000000	00H
16	00010000	10H
16	00010000	10H
16	00010000	10H
16	00010000	10H
0	00000000	00H
16	00010000	10H
0	00000000	00H

L'organizzazione del display file dello Spectrum e' fatta in modo tale che i primi 256 byte (dalla locazione 4000H alla locazione 40FFH) corrispondano al primo byte di ognuno dei 256 caratteri formanti le prime otto righe di video.

I successivi 256 bytes (dalla locazione 4100H alla locazione 41FFH) corrispondono al secondo byte di ognuno dei caratteri precedenti e cosi' via.

Pertanto, gli otto bytes che rappresentano i punti del primo carattere della prima riga di schermo avranno in memoria la seguente posizione:

1^	byte	4000H
2^	byte	4100H
3^	byte	4200H
4^	byte	4300H
5^	byte	4400H
6^	byte	4500H
7^	byte	4600H
8^	byte	4700H

Strano, non vi pare? Ma dobbiamo accettare lo Spectrum cosi' com'e', senza discutere.

Siete ora in grado di individuare gli otto bytes che rappresentano il trentunesimo carattere della terza riga dello schermo? Essi sono:

405EH, 415EH, 425EH, ..., 475EH.

La mappa dello schermo, presentata nell'appendice B, vi fornisce le esatte informazioni sulla rappresentazione in memoria dello schermo. Per ribadire quanto abbiamo appena detto sulla rappresentazione dello schermo in memoria, vi presentiamo la posizione dei bytes che rappresentano il primo carattere del secondo gruppo di 8 righe:

4800H, 4900H, 4A00H, 4B00H, 4C00H, 4D00H, 4E00H, 4F00H.

e, similmente vi presentiamo la posizione dei bytes che rappresentano il primo carattere dell'ultimo gruppo di 8 righe dello schermo:

5000H, 5100H, 5200H, 5300H, 5400H, 5500H, 5600H, 5700H.

Vi e' comunque qualche vantaggio nell'uso del linguaggio macchina. Vale la pena di rimuovere gli ostacoli. Esempio banale: in BASIC, se cercate di scrivere con PRINT nella sezione del video destinata all'input (le ultime due righe in basso), il sistema operativo BASIC protesta violentemente, mentre col linguaggio macchina avete libero accesso all'intero schermo.

Analizzando ora piu' in dettaglio l'organizzazione dell'immagine dello schermo, possiamo vedere che il valore del byte alto dell'indirizzo del primo byte (High Order Byte of First Byte - HOBFB) di ciascun carattere permette di individuare per ognuno di essi il blocco di appartenenza, tra i tre in cui e' diviso il display file.

Ad esempio, se 40H (= HOBFB < 41H	il carattere fa parte di una delle prime otto righe dello schermo
se 48H (= HOBFB < 49H	il carattere appartiene alle otto righe intermedie
se 50H (= HOBFB < 51H	il carattere appartiene alle ultime otto righe dello schermo

Non solo, ma i tre bit piu' bassi (bit 0, 1, 2) della parte alta dello indirizzo (High Order Byte - HOB) di un generico byte individuano la posizione del byte stesso tra gli otto che compongono il carattere.

Vogliamo ora cominciare a "macchiare d'inchiostro" il nostro video? Andate all'appendice B e cercate di osservare la relazione che esiste tra la locazione di memoria e lo schermo video.

Proviamo ora a cercarla analizzando il seguente esempio:

supponiamo di avere un indirizzo come 4A36H. La parte alta di questo indirizzo (HOB) e' 4A, quindi:

1. sappiamo che esso rappresenta l'indirizzo di un byte del display file, poiche' e' un valore compreso tra 40H e 58H.
2. la sua rappresentazione binaria e' 01001010
3. dalle ultime tre cifre deduciamo l'informazione che si tratta del terzo byte di un carattere sullo schermo
4. il primo byte dello stesso carattere da' gli ultimi tre byte uguali a zero e pertanto ad esso corrisponde un valore HOB di 48H. Cio' ci permette di stabilire che il carattere in questione appartiene al secondo blocco di memoria e pertanto e' posizionato su una delle otto righe intermedie dello schermo.

Dall'analisi della parte alta dell'indirizzo abbiamo dunque potuto stabilire che il byte selezionato e' il terzo byte di un carattere di una delle otto righe intermedie dello schermo.

Ma di quale carattere si tratta? La risposta a questo quesito la troviamo analizzando la parte bassa dell'indirizzo (Low Order Byte -LOB).

Nel nostro caso infatti il LOB vale 36H. Cio' ci permette di dedurre che il carattere in esame si trova 54 ($36H = 48 + 6 = 54$) posizioni dopo il primo carattere del blocco precedentemente individuato, e poiche' ogni riga e' formata da 32 caratteri, il nostro carattere si trova nella seconda riga del blocco intermedio. Piu' precisamente, esso e' il $(54 - 32 + 1)$ esimo carattere di quella riga.

In conclusione possiamo affermare che l'indirizzo 4A36H individua il terzo byte del ventitreesimo carattere della decima riga dello schermo.

Esercizio

1. A quale byte di quale carattere si riferisce l'indirizzo 564FH?
2. Provate a scrivere una routine che permetta di fornire un punto esclamativo in una determinata posizione del video. Il valore dei singoli bytes che formano questo carattere vi e' stato presentato nelle pagine precedenti.

Output - Attributi dello schermo

Il file attributi dello schermo e' strutturato piu' semplicemente rispetto al display file, poiche' associa ad ogni carattere un solo byte. Il file attributi e' posto nelle locazioni dalla 5800H alla 5AFFH ed e' composto da 768 bytes che corrispondono a 24 righe di 32 caratteri ciascuna. Esiste quindi una relazione uno-a-uno tra i bytes del file ed i caratteri dello schermo. Così' 5800H corrisponde agli attributi del primo carattere della prima riga, 5801H al secondo carattere, 5802H al terzo carattere, ..., 581FH al trentaduesimo carattere della prima riga, 5820H al primo carattere della seconda riga, 5840H al primo carattere della terza riga, ... e 5AE0H al primo carattere dell'ultima riga dello schermo.

Vediamo ora in dettaglio il significato dei singoli bit del byte degli attributi di un carattere:

byte degli attributi b b bbb bbb

bit 0 - 2	rappresentano il codice (da 0 a 7) del colore dell'inchiostro (INK) del carattere
bit 3 - 5	rappresentano il codice (da 0 a 7) del colore della carta (PAPER)
bit 6	parametro BRIGHT di luminosita' (vedi manuale)
bit 7	parametro FLASH di lampeggio (vedi manuale)

Esercizio

Trovate l'indirizzo del byte degli attributi del primo carattere del blocco intermedio e quello del primo carattere dell'ultimo blocco di 8 righe dello schermo. Le risposte sono presentate in questa stessa pagina, ma sarebbe veramente utile che risolveste l'esercizio da soli.

Esercizio

Provate a scrivere una subroutine che converta un indirizzo del file display nel corrispondente indirizzo del file attributi (per esempio l'indirizzo 4529H).

Si tratta in pratica di determinare la posizione del carattere sullo schermo ed aggiungere 5800H.

Il programma seguente presenta un metodo veloce per ottenere cio':

```
LD HL, 4529H      ;carica in HL l'indirizzo dato
LD A, H           ;carica la parte alta in A
AND 18H           ;maschera il byte per individuare il bloc-
                  ;co di appartenenza considerando i bit 3 e
                  ;4
SRA A             ;esegui per tre volte uno shift a destra
SRA A             ;dividendo cosi' per 8
SRA A             ;il risultato puo' essere 0, 1 o 2 a se-
                  ;conda che H contenga un valore compreso
                  ;tra 40H e 48H, oppure tra 48H e 50H, op-
                  ;pure ancora maggiore di 50H
ADD A, 58H        ;passa al file attributi
LD H, A           ;H contiene ora la parte alta dell'indi-
                  ;irizzo del file attributi H=58H, 59H, o
                  ;60H ed L e' rimasto sempre lo stesso!!!
```

Provate a pensarci su un attimo per convincervi!

Il modo di operare del programma e' legato alla risposta al primo esercizio:

```
1^ car. del 1^ blocco = 4000H   Corrispondente attributo = 5800H
1^ car. del 2^ blocco = 4800H   Corrispondente attributo = 5900H
1^ car. del 3^ blocco = 5000H   Corrispondente attributo = 5A00H
```

```
2^ car. del 1^ blocco = 4801H   Corrispondente attributo = 5801H
ecc. ...
ecc. ...
```

Ora dovrebbe esservi tutto chiaro!

Output - il suono

Un altro tipo di output in tempo reale che puo' essere usato sul microcomputer Spectrum e' quello sonoro; sarebbe un vero peccato se non lo usassimo in tutta la sua potenzialita'.

Programmando in linguaggio macchina, vi sono essenzialmente due modi per produrre suoni con lo Spectrum.

- 1) Mandare segnali alla porta 254 per l'output al registratore con una certa frequenza, utilizzando l'istruzione OUT

OUT (254), A

- 2) Inizializzare HL e DE con opportuni valori e chiamare la routine di sistema operativo gia' preposta per generare suoni.
I parametri da assegnare sono:

DE : durata in secondi * frequenza
HL : (437500/frequenza) - 30.125

e l'istruzione di chiamata e'

CALL 03B5H

Il primo dei metodi presentati ha il vantaggio di non essere in alcun modo legato al sistema operativo e garantisce inoltre tempi d'esecuzione piu' brevi. Ma ... vi e' un MA!

Poiche' la ULA lancia spesso delle interruzioni per eseguire la routine di illuminazione del video, il vostro programma subira' delle frequenti interruzioni durante l'esecuzione se e' memorizzato nei primi 16K di memoria. Se poi il programma prevede la generazione di suoni, il suono stesso risultera' spezzettato in parti di durata imprecisata. Per evitare queste interruzioni si dovrebbe memorizzare il programma per la generazione dei suoni nella parte piu' alta della memoria (oltre i 16K) ma cio' risulta possibile solo avendo a disposizione la versione dello Spectrum a 48K.

Se non possedete la versione a 48K, bensì quella a 16K, potete comunque provare a generare suoni con questo metodo, pero' e' bene che sappiate a priori che la musica ottenuta non sara' "pura". Per ottenere dei suoni precisi dovete usare pertanto l'apposita routine di sistema operativo (con una CALL alla ROM).

Notiamo che, mandando valori alla porta di output 254, oltre ad attivare o disattivare l'altoparlante si influenza il colore dello sfondo e si attiva la connessione MIC. Per evitare durante la generazione dei suoni questi effetti collaterali bisognera' mantenere a valori costan-

ti particolari bit del registro A (vedi capitolo sull'OUTPUT del manuale dello Spectrum).

Per quanto riguarda il secondo metodo precisiamo che la chiamata alla ROM non fa altro che attivare il comando BEEP. Il valore contenuto in DE rappresenta la durata del suono mentre il valore di HL ne fornisce la frequenza. Sperimentando valori diversi per DE ed HL potete rendervi conto di quale sia la gamma di suoni realizzabili.

E' evidente che, facendo uso in questa routine del comando BEEP, la gamma di suoni generati sara' compresa tra quelli previsti appunto per il BEEP.

P A R T E Q U A R T A

I PROGRAMMI MONITOR PER IL LINGUAGGIO MACCHINA

=====

Programma "EZ CODE Monitor"

Le funzioni che questo programma mette a disposizione dell'utente sono:

1. INPUT Permette di caricare da tastiera un modulo di programma scritto in esadecimale
 gia' completamente assemblato
 o parzialmente assemblato con i salti relativi ed assoluti espressi sotto forma di numeri di linea.
2. LIST Fornisce il listato del programma sorgente da voi inserito.
3. DUMP Trasferisce il codice oggetto in un'area di memoria specificata dall'utente.
4. EXAMINE Permette di esaminare il contenuto di un blocco di locazioni di memoria.
5. SAVE Trasferisce su cassetta, a vostra scelta
 o il "modulo sorgente"
 o il "programma oggetto" ottenuto precedentemente con un DUMP.
6. LOAD Carica il file specificato da cassetta.
7. RUN Lancia l'esecuzione del programma caricato con DUMP.

PREREQUISITI per l'EZ-Code

Prima di usare questo programma monitor per il caricamento da tastiera del vostro programma dovrete avere gia' tradotto questo da linguaggio Assembler a linguaggio macchina. Nel far cio' pero' ricordate che non e' necessario specificare gli indirizzi dei salti relativi o assoluti, ma e' sufficiente esprimerli come numero di linea!

Il modulo di programma che intendete caricare non deve superare gli 800 bytes o le 200 istruzioni.

Non potete trasferire (DUMP) il programma oggetto partendo da un indirizzo inferiore a 31499 (per non cancellare il programma EZ-Code).

LOGICA del programma EZ-Code

Il programma EZ-Code e' stato realizzato in modo da permettere all'utente di caricare da tastiera delle istruzioni in linguaggio macchina per linee numerate, come avveniva per i programmi BASIC.

Ciascuna linea del "programma sorgente" ha un numero di linea e puo' contenere fino ad un massimo di 4 bytes di codice macchina.

Il grosso vantaggio presentato da questo tipo di programma consiste nel fatto che esso permette di eseguire funzioni di EDIT sulle varie linee. Il "programma sorgente" puo' anche essere salvato per blocchi separati, man mano che il lavoro procede.

Questo programma permette inoltre di inserire nel programma sorgente dei salti relativi od assoluti senza prima averne calcolato gli indirizzi o la lunghezza in bytes, ma facendo solo riferimento al numero della linea del "programma sorgente" nella quale e' contenuta l'istruzione a cui si vuol saltare!

Cio' risulta molto utile, perche' permette di effettuare modifiche al programma, senza per questo dover ricalcolare la lunghezza di tutti i salti relativi.

Il calcolo degli indirizzi relativi ed assoluti viene fatto automaticamente dal monitor nel momento in cui trasferisce il "programma oggetto" in un'area di memoria specificata da voi (DUMP). Il "programma oggetto" cosi' ottenuto e' praticamente "salvato" in memoria.

Sommario delle istruzioni dell'EZ-Code

Notiamo che la prima cosa che il programma chiede e'

"Indirizzo di caricamento".

Pertanto va specificato a questo punto l'indirizzo a partire dal quale dovra' poi essere caricato il "programma oggetto".

Questo indirizzo deve essere superiore a 31500.

**** Inserimento di linee di programma ****

1. per INSERIRE una linea di programma:

<numero linea><spazio><codice esadecimale(al piu' 4 byte)><ENTER>

per esempio:

1 210040 se volete inserire alla riga 1 l'istruzione LD HL,4000H

2. per CORREGGERE una linea di programma:

<numero linea><spazio><nuovo codice macchina><ENTER>

per esempio, se dopo la prima riga precedente battiamo:

1 210140 ne modifichiamo il contenuto, e la linea rappresenta ora l'istruzione LD HL,4001H

3. per CANCELLARE una riga:

<numero linea><ENTER>

per esempio:

1 <ENTER> cancella la prima linea di programma.

4. per specificare i salti RELATIVI e ASSOLUTI:

<numero linea><spazio><codice operativo del salto>

<tasto "elle minuscola"><numero di linea><ENTER>

per esempio:

1 c312 rappresenta un JUMP alla linea 2

2 1811 rappresenta un JR alla linea 1

**** COMANDI ****

1. dump<ENTER>

* trasferisce il codice macchina nell'area di memoria che parte dall'indirizzo specificato all'inizio.

* questo comando deve essere eseguito prima di lanciare l'esecuzione del programma.

Abbreviazione: du

2. exit<ENTER>

* rilascia il programma EZ-Code e fa rientrare nel sistema BASIC.

Abbreviazione: ex

3. list<ENTER>

* visualizza le prime 22 linee del "programma sorgente".

* per far proseguire il listato, battere un tasto qualsiasi, tranne "m" e "BREAK".

Abbreviazione: li

list<numero linea><ENTER>

* visualizza 22 linee a partire da quella specificata.

(<numero linea> deve essere compreso tra 1 e 200)

Abbreviazione: NESSUNA

4. load(ENTER)

- * carica un "programma sorgente" dalla cassetta, rimpiazzando quello attualmente presente in memoria.
(per caricare codice oggetto rimandiamo a HEXLOAD)
Abbreviazione: lo

5. mem(ENTER)

messaggio: Indirizzo di partenza

- * inserisce l'indirizzo di memoria del primo dei byte che devono essere visualizzati.
- * tale indirizzo deve essere compreso tra 0 e 32767 per lo Spectrum a 16K e tra 0 e 65535 per quello a 48K.
- * battere "m" per interrompere l'esame della memoria
Abbreviazione: me

6. new(ENTER)

- * cancella il programma in memoria e rilancia l'esecuzione dello EZ-Code
- * questo comando risulta molto utile quando vogliamo digitare un nuovo programma.
Abbreviazione: ne

7. run(ENTER)

- * lancia l'esecuzione del programma caricato (dump) dalla locazione specificata o all'inizio del lavoro oppure quando avete caricato un nuovo programma sorgente.
Abbreviazione: ru

8. save(ENTER)

- * salva su cassetta il programma sorgente oppure il programma oggetto.

messaggi: nome del programma

inserire il nome che volete dare al programma.

Sorgente o codice macchina: (s or m)

inserire s per salvare il programma sorgente

inserire m per salvare il programma oggetto.

Start tape, then press any key

assicurarsi che il registratore sia attivato e sia collegato correttamente.

battere un tasto quando e' tutto pronto.

Abbreviazione: sa

NOTE

1. Se non volete che alla fine dell'esecuzione venga visualizzato il contenuto della coppia di registri BC, modificate la linea 3090 cosi':

```
3090 IF K$="ru" THEN LET L=USR R
```


2. Per far ripartire il programma EZ-Code:
 - o usate RUN e riinizializzate tutte le variabili
 - o usate GOTO 2020 e ripartite dal messaggio:
 - "Comando o linea ... :".
3. Tutti i dati numerici, fatta eccezione per le istruzioni in codice macchina, devono essere in notazione decimale.
4. Per facilitare le operazioni di inserimento di nuove linee, conviene lasciare gia' tra una riga e l'altra del programma dei numeri di linea liberi.
 - Così', ad esempio, conviene sin dall'inizio numerare le linee come 1, 5, 10, ... anziche' come 1, 2, 3,
 - Cio' rende, come abbiamo gia' detto, piu' flessibile l'operazione di inserimento delle istruzioni.

ESERCIZIO sull'uso di EZ-Code

Inserire in macchina il seguente programma.

```

210040    LD HL,4000H           ;cancellazione video
110140    LD DE,4001H
01FF17    LD BC,6143
3EFF     LD A, 0FFH
77       LD (HL), A
EDB0     LDIR
3E7F     LOOP:LD A, 7FH       ;maschera il tasto BREAK
DBFE     IN A,(0FEH)
E601     AND 1
20F8     JR NZ, LOOP
C9       RET

```

Per caricare il programma usiamo il monitor EZ-Code, che deve essere gia' caricato in memoria:

<RUN>

Indirizzo di caricamento: 31500<ENTER>

Comando o linea...: 1 210040<ENTER>

Comando o linea...: 5 110140<ENTER>

Comando o linea...: 10 01ff17<ENTER>

Comando o linea...: 15 3eff<ENTER>

Comando o linea...: 20 77<ENTER>

Comando o linea...: 25 edb0<ENTER>

Comando o linea...: 30 3e7f<ENTER>

Comando o linea...: 35 dbfe<ENTER>

Comando o linea...: 40 e601<ENTER>

Comando o linea...: 45 20130<ENTER>

(Quest'ultima istruzione: 20 seguito da "L minuscolo" e da 30, ha il significato di JR NZ alla linea 30)

Comando o linea...: 50 c9<ENTER>

Comando o linea...: list<ENTER>

Comando o linea...: dump<ENTER>

Comando o linea...: mem<ENTER>

Indirizzo di partenza: 31500<ENTER>

m (e' il tasto che serve per concludere la fase
di visualizzazione della memoria)

Comando o linea...: run<ENTER>

{BREAK}

Notiamo che tra il numero di linea ed il codice dell'istruzione deve essere lasciato uno spazio.

```

100 REM programma EZCODE
110 REM machine code monitor
120 GO TO 9000
130 DEF FN d(s$)=(s$>"9")*(CODE
s$-55)+(s$<="9")*(CODE s$-48)-
s$>"z")*32
140 DEF FN o(o$)=((O$="ca")+ (O$
="da")+ (O$="ea")+ (O$="fa")+ (O$="
ce")+ (O$="d2")+ (O$="e2")+ (O$="f2
")+ (O$="c3"))-((O$="38")+ (O$="30
")+ (O$="28")+ (O$="20")+ (O$="18")
+(O$="10"))
1000 REM
1010 REM visualizzazione linea
1020 CLS : PRINT AT ze,25; INVER
SE on; FLASH on;"LISTATO"
1030 LET F=ze: PRINT AT ze,ze;
1040 FOR J=pl1 TO pl2
1050 IF C$(J,on)=" " THEN GO TO
1110
1060 PRINT TAB tr-LEN STR$ J;J;T
AB fr;" "
1070 IF C$(J,tw,on TO on)="l" TH
EN PRINT C$(J,on)+" "+C$(J,tw)+C
$(J,tr): GO TO 1090
1080 PRINT C$(J,on);" ";C$(J,tw)
;" ";C$(J,tr);" ";C$(J,fr)
1090 LET F=F+on
1100 IF F=22 THEN GO TO 1120
1110 NEXT J
1120 PRINT AT ze,25;" "
1130 RETURN
2000 REM
2010 REM programma principale
2020 INPUT "Comando o Linea... "
:A$
2030 IF A$( TO fr)=" " THEN G
O TO mr
2040 IF A$(on)>"9" THEN GO TO 30
00
2050 LET k$="": FOR K=on TO fr
2060 IF A$(K TO K)=" " THEN GO T
O 2090
2070 LET k$=k$+A$(K TO K)
2080 NEXT K
2090 IF K=5 OR VAL k$=ze OR VAL
k$>ln THEN GO TO mr
2100 LET J=VAL k$: LET n=J: REM
il numero di linea deve essere
el piu' di 3 cifre
20110 LET A$=A$(K+on TO )
2120 LET k$=""
2130 FOR K=on TO LEN A$
2140 IF A$(K TO K)<>" " THEN LET
k$=k$+A$(K TO K)
2150 NEXT K
2160 LET A$=k$
2162 IF A$(on)="l" THEN GO TO mr
2170 CLS : FOR I=on TO 7 STEP tw
2180 LET K=INT (I/tw+on)
2190 LET C$(J,K)=A$(I TO I+on)

```

```

2200 NEXT I
2210 IF C$(n,on)=" " THEN GO TO
2250
2220 IF n<TP THEN LET TP=n
2230 IF n>BP THEN LET BP=n
2240 GO TO 2320
2250 IF n<>BP THEN GO TO 2260
2260 IF BP=on OR C$(BP,on)<>" "
THEN GO TO 2320
2270 LET BP=BP-on: GO TO 2260
2280 IF n<>TP THEN GO TO 2320
2290 IF C$(TP,on)<>" " THEN GO
TO 2320
2300 IF TP<>BP AND TP<>n THEN L
ET TP=TP+on: GO TO 2290
2310 LET TP=on
2320 LET pp=n
2330 IF n<TP THEN LET pp=TP: GO
TO 2380
2340 LET numlp=ze
2350 IF pp=TP OR numlp=11 THEN G
O TO 2380
2360 IF C$(pp,on)<>" " THEN LET
numlp=numlp+on
2370 LET pp=pp-on: GO TO 2350
2380 LET pl1=pp: LET pl2=BP
2390 GO SUB 1000: REM visualizza
zione di un blocco di linee
2400 GO TO mr
3000 REM
3010 REM Comandi*****
3020 LET k$=A$(TO tw)
3030 IF k$="du" THEN GO TO 5000
3040 IF k$="ex" THEN STOP
3050 IF k$="li" THEN GO TO 4000
3060 IF k$="lo" THEN GO TO 7000
3070 IF k$="me" THEN GO TO 6000
3080 IF k$="ne" THEN RUN
3090 IF k$="ru" THEN PRINT USR R
3100 IF k$="sa" THEN GO TO 8000
3110 GO TO mr
4000 REM
4010 REM List*****
4020 LET pl1=TP: LET pl2=BP
4030 LET n1=CODE A$(6 TO 6)
4040 IF LEN A$>fr AND n1>47 AND
n1<58 THEN LET pl1=VAL A$(5 TO 8
)
4050 GO SUB 1000
4060 GO TO mr
5000 REM
5010 REM CUMF*****
5020 CLS : PRINT AT ze,20; INK 0
n; INVERSE on; FLASH on;"CARICAM
ENTO": LET G=R
5030 PRINT AT on,ze;
5040 FOR J=TP TO BP
5050 IF C$(J,on)=" " THEN GO TO
5470
5060 IF C$(J,tw,on TO on)<>"(" T
HEN GO TO 5380

```

```

5070 POKE G,ze: POKE G+on,ze: PO
KE G+tw,ze: POKE G+tr,ze
5080 LET jl=VAL (C$(J,tw,tw TO t
w)+C$(J,tr))
5090 PRINT TAB tr-LEN STR$ J; IN
VERSE on;J;TAB fr; INVERSE ze;"
";C$(J,on)+" "+C$(J,tw)+C$(J,tr)
";=>";
5100 IF jl<ze OR jl>ln THEN GO T
O 5460
5110 LET CJ=FN O(C$(J,on))
5120 PRINT TAB 17-LEN STR$ J; I
NVERSE on;J;TAB 18; INVERSE ze;"
";C$(J,on);" ";C$(J,tw);" ";
C$(J,tr);" ";C$(J,fr);
5130 IF ABS CJ<>on THEN GO TO 54
60
5140 LET dd=(jl>J)-(jl<J)
5150 LET ja=G: LET dp=ze
5160 IF jl=J THEN GO TO 5270
5170 LET cl=J+dd
5180 LET n1=ze: IF C$(cl,on)="
" THEN GO TO 5220
5190 IF C$(cl,tw,on TO on)<>"L"
THEN LET n1=on+(C$(cl,tw)<>" ")
+(C$(cl,tr)<>" ")+(C$(cl,fr)<>"
"): GO TO 5220
5200 LET TJ=FN O(C$(cl,on))
5210 LET n1=(TJ=on)*tr+(TJ=-on)*
tw
5220 IF cl=JL AND dd>ze THEN GO
TO 5270
5230 LET dp=dp+n1
5240 IF cl=JL THEN GO TO 5270
5250 LET cl=cl+dd
5260 GO TO 5160
5270 IF CJ=on THEN LET ja=ja+dd+
dp+(dd>ze)*tr: GO TO 5310
5280 IF dd>ze THEN LET dp=dp+2
5290 IF dp>126 AND dd<ze THEN GO
TO 5460
5300 IF dp>129 AND dd>ze THEN GO
TO 5460
5310 LET U=16*FN d(C$(J,on,on TO
on))+FN d(C$(J,on,tw TO tw))
5320 POKE G,U: LET G=G+on
5330 IF CJ=on THEN POKE G,ja-INT
(ja/qk)*qk: LET G=G+on: POKE G,
INT (ja/qk): LET G=G+on: GO TO 5
360
5340 IF dd<ze THEN LET dp=-dp
5350 LET dp=dp-tw: POKE G,dp: LE
T G=G+on
5360 PRINT "ok"
5370 GO TO 5470
5380 FOR I=on TO 7 STEP tw
5390 LET K=INT (I/tw+on)
5400 LET U=16*FN d(C$(J,K,on TO
on))+FN d(C$(J,K,tw TO tw))
5410 IF U<ze THEN GO TO 5440
5420 POKE G,U

```

```

5430 LET G=G+on
5440 NEXT I
5450 GO TO 5470
5460 PRINT "***"
5470 NEXT J
5480 PRINT AT ze,20;"
": GO TO mr
6000 REM
6010 REM Vis. memoria*****
6020 INPUT "indirizzo iniziale:"
":dm
6030 CLS : PRINT AT ze,ze;
6040 LET G=dm: LET F=ze
6050 LET F=F+on: PRINT TAB 5-LEN
STR$ G;G;TAB 6;
6060 FOR I=on TO fr
6070 LET V=PEEK G
6080 LET H=INT (V/16)
6090 LET L=V-16*H
6100 PRINT D$(H+on);D$(L+on);" "
:
6110 LET G=G+on
6120 NEXT I
6130 PRINT " "
6140 IF F<>22 THEN GO TO 6050
6150 LET k$=INKEY$: IF k$="" THE
N GO TO 6150
6160 IF k$<>"m" AND k$<>"M" THEN
LET F=ze: POKE 23692,qk-on: GO
TO 6050
6200 POKE 23692,on: PAUSE 20: GO
TO mr
7000 REM
7010 REM Load*****
7020 CLS
7030 INPUT "Caricamento dati: ba
tti un tastoquando sei pronto ";
k$
7040 PRINT AT ze,19; INVERSE on;
FLASH on;"CARICAMENTO"
7050 LOAD "source" DATA C$( )
7060 FOR I=on TO ln
7070 LET TP=I
7080 IF C$(I,on)<>" " THEN GO T
O 7100
7090 NEXT I
7100 FOR I=ln TO on STEP -1
7110 LET BP=I
7120 IF C$(I,on)<>" " THEN GO T
O 7140
7130 NEXT I
7140 PRINT AT ze,19;"
"
7150 GO TO 9150
8000 REM
8010 REM Save*****
8020 INPUT "nome del file";n$
8030 IF n$="" THEN GO TO 8020
8040 INPUT "Sorgente o codice ma
cchina: (s o m)";k$
8050 IF k$<>"s" AND k$<>"m" THEN

```

```

GO TO 8040
8050 IF k$="s" THEN SAVE n$ DATA
C$( ): GO TO mr
8070 INPUT "Indirizzo iniziale:
";ss
8080 INPUT "Indirizzo finale: ";
sf
8090 LET sb=sf-ss+on
8100 SAVE n$CODE ss,sb
8110 GO TO mr
9000 REM
9010 REM inizializzazione
9020 LET ze=PI-PI: LET on=PI/PI:
LET tw=on+on: LET tr=on+tw: LET
fr=tw+tw: LET qk=256: LET mr=20
90: LET ln=200
9025 BORDER 7: PAPER 7: INK on:
INVERSE ze: OVER ze: FLASH ze: B
RIGHT ze: BEEP .25,24: BEEP .25,
12
9030 DIM A$(15): DIM O$(tw)
9040 LET TP=ln: LET BP=on: REM b
offer del numero di linea
9050 DIM C$(ln,fr,tw): REM mem.
codice
9060 PRINT AT ze,16; INVERSE on;
FLASH on;"INIZIALIZZAZIONE"
9070 FOR I=on TO ln
9080 FOR J=on TO fr
9090 LET C$(I,J)=" "
9100 NEXT J
9110 BEEP .01,20
9120 NEXT I
9130 PRINT AT ze,16;"
9140 LET D$="0123456789ABCDEF"
9150 CLS : PRINT "Indirizzo piu'
basso: ";31500
9160 INPUT "Indirizzo di caricam
ento: ";R: PAUSE 20
9170 IF R<31500 THEN GO TO 9160
9180 CLS : GO TO mr
9900 CLS : PRINT "Inizio spazio
libero: ";(PEEK 23653+256*PEEK 2
3654)

```

HexLoad machine code Monitor

Questo programma BASIC e' classificato tra i programmi monitor in quanto puo' essere usato per SCRIVERE un programma in codice esadecimale in memoria, LISTARE il contenuto della memoria, SPOSTARE i blocchi di bytes da un posto all'altro della memoria, SALVARE il contenuto di una parte della memoria su cassetta e CARICARE dalla cassetta il valore di alcuni bytes.

In particolare possiamo usare il programma Hexload per collegare tra loro diverse routines realizzate con il programma EZ-Code. Cio' risulta utilissimo nel caso in cui si debbano scrivere programmi piuttosto complessi, poiche' il monitor EZ-Code non permette di caricare piu' di 800 bytes o 200 istruzioni alla volta.

In altri termini, dovendo costruire programmi molto lunghi, conviene sviluppare separatamente ciascuno dei moduli in cui essi sono suddivisi e salvare il relativo codice oggetto su cassetta. Usando poi il programma Hexload, che e' un programma BASIC molto piu' breve dello EZ-Code, si possono caricare i vari moduli e collegarli tra loro spostandoli appropriatamente all'interno della memoria.

Questo metodo e' stato per l'appunto da noi utilizzato per realizzare il programma FREEWAY FROG (La rana attraversa la strada).

Logica del programma Hexload

La logica del programma Hexload e' molto semplice.

Il programma fissa il RAMTOP del BASIC a 26999. Cio' significa che il codice macchina del vostro programma puo' essere caricato in una qualsiasi locazione di memoria compresa per la versione a 16K tra 27000 e 32578 e per quella a 48K tra 27000 e 65346.

Le funzioni che il programma Hexload monitor offre sono:

- WRITE per scrivere in memoria in formato esadecimale
- SAVE per memorizzare su cassetta
- LOAD per caricare da cassetta
- LIST per visualizzare il contenuto di un blocco di locazioni
- MOVE per spostare bytes da una locazione ad un'altra.

Sommario delle istruzioni

1. WRITE

Scriva un codice esadecimale in memoria.

Procedura:

- a. Inserire l'indirizzo della locazione da cui si vuole iniziare la scrittura in codice decimale.
L'indirizzo deve essere compreso tra 27000 e 32578 per la versione a 16K
tra 27000 e 65346 per la versione a 48K
Per esempio: alla richiesta del sistema inserire: 27000<ENTER>
- b. digitare il programma in formato esadecimale.
- c. Battere "m" per tornare al menu.

2. SAVE

Salva un blocco di bytes su cassetta.

Procedura:

- a. Inserire l'indirizzo di partenza del blocco di memoria da salvare. Tale indirizzo deve essere compreso
- tra 0 e 27000 per la versione a 16K
- tra 0 e 65535 per la versione a 48K
- b. Inserire il numero di bytes da salvare.
- c. Inserire il nome del modulo da salvare.
- d. Premere un tasto dopo aver predisposto il registratore.
- e. (Opzionale) Richiedere la verifica di quanto e' stato salvato. Quest'ultima operazione, anche se non e' obbligatoria, risulta molto utile in quanto permette di verificare che non sia stato alterato qualche byte nel corso della registrazione.

3. LOAD

Carica da cassetta un modulo di programma.

Procedura:

- a. Inserire l'indirizzo di partenza per il caricamento. L'indirizzo ha le stesse limitazioni gia' presentate per l'istruzione WRITE.
- b. Inserire il nome del modulo da caricare. Se non siete sicuri del nome premete solo <ENTER>. Verra' cosi' caricato il primo modulo presente sulla cassetta.

4. LIST

Visualizza il contenuto della memoria partendo dall'indirizzo specificato.

Procedura:

- a. Inserire l'indirizzo di partenza.
(per le limitazioni sull'indirizzo vedi comando SAVE).
- b. Premere un tasto per proseguire l'esame con nuove locazioni
- c. Premere "m" per tornare al menu.

5. MOVE

Sposta un blocco di bytes, di cui e' assegnato l'indirizzo di inizio e fine, in un'altra zona di memoria.

Procedura:

- a. Inserire l'indirizzo di partenza del blocco da spostare.
(Per le limitazioni vedi comando SAVE).
- b. Inserire l'indirizzo di termine del blocco da spostare.
(Per le limitazioni vedi comando SAVE).
- c. Inserire l'indirizzo di inizio del blocco di destinazione.
(Per le limitazioni vedi comando WRITE).
- d. Con questo comando potete anche copiare nella RAM delle routines poste su ROM.

Per esempio:

```
Sposta dall'indirizzo : 0<ENTER>  
fino a: 1000<ENTER>  
all'indirizzo: 32000<ENTER>
```

copia i valori contenuti nelle locazioni di ROM dalla 0 alla 1000 nella RAM a partire dall'indirizzo 32000.

NOTA: In fase di inizializzazione, il programma non accetta indirizzi non ammessi. Se commettete un errore di questo tipo ve li richiede un'altra volta.

ESERCIZIO

Provate ad inserire il programma già caricato con EZ-Code utilizzando ora questo nuovo Monitor.

```

100 REM Programma monitor
110 REM HEXLOAD
120 CLEAR 26999: LET ze=PI-PI:
LET on=PI/PI: LET tw=on+on: LET
qk=256: LET lm=27000: LET mr=140
: LET wl=340:
130 GO SUB 2000
140 CLS: PRINT "inizio area a
disposizione: ";lm
150 PRINT "menu": PRINT: PRINT
" Per scrivere in codice
macchina.....1"
160 PRINT: PRINT " Per salv
are il codice macchina
.....2"
170 PRINT: PRINT " Per cari
care il codice macchina
.....3"
180 PRINT: PRINT " Per list
are il codice macchina
.....4"
190 PRINT: PRINT " Per spos
tare il codice macchina
.....5"
200 PRINT: PRINT "Date il codi
ce della funzione prescelta "
210 LET g$=INKEY$
220 IF g$="m" OR g$="M" THEN ST
OP
230 IF g$="" OR g$<"1" OR g$>"5
" THEN GO TO 210
240 CLS: PRINT "Indirizzo iniz
iale minimo per il codice macchi
na: ";lm
250 GO TO 300*VAL g$
300 REM Write*****
310 INPUT "Scrittura dall'indir
izzo: ";d
320 IF d>mm OR d<lm THEN GO TO
310
330 PRINT: PRINT "Scrivi all'i
ndirizzo: ";d: PRINT "Per tornar
e al menu battere ""M""
340 LET a$=""
350 IF a$="" THEN INPUT "Inseri
re codice esad. ";a$
360 IF a$(on)="m" OR a$(on)="M"
THEN GO TO mr
370 IF LEN a$/tw<>INT (LEN a$/t
w) THEN PRINT "Dato non accettab
ile "; GO TO wl
380 LET c=ze
390 FOR f=16 TO on STEP -15
400 LET a=CODE a$((f=16)+tw*(f=
on))
410 IF a<48 OR a>102 OR (a>57 A
ND a<55) OR (a>70 AND a<97) THEN
PRINT "Dato non accettabile ";
GO TO wl
420 LET c=c+f*((a<58)*(a-48)+(a
>64 AND a<71)*(a-55)+(a>96)*(a-8

```

```

7))
430 NEXT f: POKE d,c: LET d=d+o
n
440 PRINT a$( TO tw);" ";
450 LET a$a$(3 TO )
460 IF d=UDG THEN PRINT "Attenz
ione: state invadendo l'area gra
fica!": GO TO w1
470 IF d=UDG-20 THEN PRINT "Att
enzione: state invadendo una are
a di memoria riservata!": GO TO
w1
480 GO TO w1+on
600 REM Save*****
610 INPUT "salva dall'indirizzo
":a
620 INPUT "Numero di bytes da s
alvare:";n
630 INPUT "Nome del programma:"
;a$
640 SAVE a$CODE a,n
650 PRINT "Volete fare la verif
ica? (s/n)"
660 INPUT v$
670 IF v$<>"s" THEN GO TO mr
680 PRINT "Riavvolgete il nastri
o e mettete su ""PLAY""."
690 VERIFY a$CODE a,n
700 PRINT "O.K.": PAUSE 50
710 GO TO mr
900 REM Load*****
910 INPUT "Carica dall'indirizz
o:";a
920 IF a>mm OR a<lm THEN GO TO
910
930 INPUT "Nome del programma:"
;a$
940 PRINT "Posiziona il registr
atore su ""PLAY""."
950 LOAD a$CODE a: GO TO mr
1200 REM List*****
1210 LET a$="0123456789ABCDEF"
1220 INPUT "Indirizzo di partenz
a:";d
1230 PRINT "Batti ""M"" per torn
are al Menu."
1240 LET a=INT (PEEK d/16): LET
b=PEEK d-16*INT (PEEK d/16)
1250 PRINT d;TAB 7;a$(a+on);a$(b
+on)
1260 LET d=d+on
1270 IF INKEY$="m" OR INKEY$="M"
THEN GO TO mr
1280 GO TO 1240
1500 REM Move*****
1510 INPUT "Sposta dalla locazio
ne:";fm
1520 INPUT "fino alla locazione:"
";Um
1530 INPUT "nell'area che parte
da ";tm

```

```

1540 IF tm>fm THEN GO TO 1610
1550 LET mp=tm
1560 FOR I=fm TO um
1570 POKE mp,PEEK I
1580 LET mp=mp+on
1590 NEXT I
1600 GO TO mr
1610 LET mp=um+tm-fm
1620 FOR I=um TO fm STEP -on
1630 POKE mp,PEEK I
1640 LET mp=mp-on
1650 NEXT I
1660 GO TO mr
2000 LET RT=PEEK 23732+qk*PEEK 2
3733
2010 IF RT=65535 THEN LET mm=653
47: LET UDG=65367
2020 IF RT=32767 THEN LET mm=325
79: LET UDG=32599
2030 LET n1=INT (UDG/qk)
2040 POKE 23675,UDG-n1*qk: POKE
23676,n1
2050 RETURN

```


P A R T E Q U I N T A

PROGRAMMA LA RANA ATTRAVERSA LA STRADA (FREEWAY FROG)

Progetto del programma.

Questo programma affronta il problema di alcune rane che, per tornare alle loro dimore, devono attraversare saltando una strada a traffico intenso.

Sulla strada vi sono camion, auto, moto e biciclette e per di piu' vi transitano spesso vetture della polizia.

Il punteggio viene stabilito in base al numero di salti che la rana compie per attraversare la strada.

Cercate di capire a fondo il gioco, poiche' dovreste poi realizzarlo con un programma.

Il primo livello di analisi consiste per l'appunto nella corretta definizione del problema.

Se non avete capito bene il problema e non siete in grado di definirlo in modo preciso, troverete parecchie difficolta' nel seguirci durante le prossime fasi di sviluppo dell'intero progetto.

Struttura del programma.

Per sviluppare l'intero progetto adotteremo la tecnica di analisi presentatavi nei precedenti capitoli, definita PROGETTAZIONE MODULARE TOP-DOWN. Procederemo partendo da un livello molto alto di astrazione, e suddivideremo l'intero programma in moduli logicamente ben definiti.

I moduli che compongono il nostro programma sono:

1. INIZIALIZZAZIONE
sviluppo delle routines iniziali.
2. FLUSSO DEL TRAFFICO
controllo del traffico sulla strada.
Questo blocco puo' a sua volta essere suddiviso in:
 - a. flusso del traffico regolare (camion, auto, motociclette, ...)
 - b. flusso del traffico privilegiato (auto della polizia, ambulanze, ...)

3. RANA

controllo sul movimento della rana, su eventuali urti con veicoli e sull'arrivo a casa.

4. CONTROLLO SUL PUNTEGGIO

Questa parte del programma e' destinata al calcolo ed alla visualizzazione del punteggio, nonche' al test di fine partita.

5. CONCLUSIONE

E' la parte del programma predisposta a riordinare tutto prima di ritornare il controllo al sistema operativo.

Sviluppo del programma.

Nello sviluppo del programma FREEWAY FROG abbiamo individuato sei fasi distinte che permettono comunque di mantenere la suddivisione logica presentata precedentemente.

Dopo aver ultimato ciascuna di queste fasi, dobbiamo effettuare sul prodotto ottenuto dei test di correttezza, ed essere ben sicuri del risultato prima di affrontare l'analisi della parte successiva.

Le sei fasi individuate sono:

1. Sviluppo del data base iniziale

in cui bisogna definire la forma di ogni oggetto e per ciascuno di essi l'insieme dei dati, su cui operera' il programma.

2. Inizializzazione

in cui bisogna predisporre lo schermo ed inizializzare le singole variabili.

3. Flusso del traffico

in questa fase ci occuperemo solo del flusso del traffico normale, lasciando ad una fase successiva il controllo dei veicoli speciali (auto della polizia), che richiede una logica diversa.

4. Auto della polizia

viene controllato il flusso dei mezzi speciali.

5. Rana

questa e' la fase destinata al controllo dei movimenti della rana, alla sostituzione di una rana morta o giunta al traguardo con un'altra nuova, pronta ad attraversare ancora la strada, ai controlli sul risultato dell'attraversamento (incidenti, arrivo a casa,...), al calcolo del punteggio, ...

6. Controllo

in questa fase si progettano routines che permettono di inizializzare il gioco, di controllare i punteggi, di bloccare il programma, di tornare al BASIC.

Prima di procedere nell'analisi di queste fasi, vogliamo presentarvi un programma BASIC che, sommando tra loro i contenuti di un blocco di locazioni di memoria, genera un parametro di controllo detto per l'appunto "checksum".

Questo tipo di controllo viene spesso usato per testare la correttezza dei dati in ingresso.

```
9000 REM programma di checksum
9010 REM
9020 INPUT "dall'indirizzo: ";f
9030 INPUT "all'indirizzo: ";t
9040 LET s=0
9050 FOR i=f TO t
9060 LET s=s+PEEK i
9070 NEXT i
9080 PRINT "checksum: ";s
9090 GO TO 9020
```

Dati gli indirizzi di inizio e fine blocco che vogliamo controllare, il programma esegue automaticamente il "checksum" e ci fornisce il risultato in notazione decimale.

**** Progetto della sagoma degli oggetti ****

Dato che nel gioco si presuppone che il traffico si svolga nei due sensi, dovremo prevedere due sagome per il camion: quella del camion che si muove verso destra e quella del camion che si muove in senso inverso e cio' naturalmente vale anche per tutti gli altri mezzi di locomozione.

Per la rana invece, visto che essa si puo' muovere in quattro direzioni (alto, basso, destra, sinistra), dovremo prevedere addirittura quattro sagome diverse: una per ciascuna direzione.

Ora, per definire la posizione di ogni oggetto e poi disegnarlo adotteremo le seguenti convenzioni:

Se la sagoma dell'oggetto e' formata dai quattro caratteri:

```
  C  D
  A  B
```

il puntatore di posizione deve fare riferimento al carattere A. Il carattere A viene disegnato per primo, poi viene disegnato B ... fino a completare l'intera riga.

Si passa poi a disegnare la riga superiore, riposizionandosi sul carattere C ed operando in modo analogo a prima.

La sagoma sara' percio' organizzata nel database nella forma seguente:

Sagoma ABCD

Non dimentichiamo che ogni carattere e' formato da otto bytes. Se adottiamo ora il principio di disegnare il singolo carattere dall'alto in basso, il database relativo ad una figura del tipo presentato avra' la seguente configurazione:

```
Sagoma:      a1, a2, a3, a4, a5, a6, a7, a8
              b1, b2, b3, b4, b5, b6, b7, b8
              c1, c2, c3, c4, c5, c6, c7, c8
              d1, d2, d3, d4, d5, d6, d7, d8
```

Per organizzare il database facciamo anche la seguente ipotesi: quando dovremo disegnare una sagoma, prima riempiamo gli appositi bytes del display file e poi assegneremo gli opportuni valori ai relativi bytes del file attributi.

Sotto questa ipotesi nel database i dati relativi ai colori della sagoma verranno inseriti subito dopo quelli necessari per la costruzione della sagoma stessa.

Ricordiamo che mentre per costruire un carattere occorrono otto bytes,

per definire le caratteristiche del colore ne e' sufficiente uno solo. Nel caso di una sagoma formata da quattro caratteri pertanto dovremo far seguire, nel database, ai trentadue bytes che rappresentano la figura i quattro bytes relativi agli attributi dei singoli caratteri.

**** Caricamento dei dati relativi alle sagome ****

label	n.lines	da(H)	a (H)	da(D)	a (D)	checksum
FRGSHP	120	69AFH	6A36H	27055	27190	18085
LBIKE	340	6A37H	6A76H	27191	27254	3647
LBATT	430	6A77H	6A7EH	27255	27262	28
RBIKE	460	6A7FH	6ABEH	27263	27326	3355
RBATT	560	6ABFH	6AC6H	27327	27334	28
LCAR	600	6AC7H	6B26H	27335	27430	5073
LCATT	730	6B27H	6B32H	27431	27442	36
RCAR	770	6B33H	6B92H	27443	27538	4902
RCATT	900	6B93H	6B9EH	27539	27550	12
LTRUCK	940	6B9FH	6C76H	27551	27766	22834
LTATT	1230	6C77H	6C91H	27767	27793	87
RTRUCK	1280	6C92H	6D69H	27794	28009	21834
RTATT	1570	6D6AH	6D84H	28010	28036	87
BLANK	1620	6D85H	6D88H	28037	28040	0

Questo modulo occupa le locazioni da 27055 a 28040 per un totale di 986 bytes ed il checksum totale e' di 79197.

A questo database e' stato assegnato il suggestivo nome di "shapdb" (dall'inglese shape database).

All'interno del database non tutte le sagome sono rappresentate seguite dai bytes degli attributi: fanno eccezione infatti le sagome della rana. Non si e' mantenuto lo stesso formato anche per la rana dato che essa assume il colore VERDE finche' agisce da viva, il colore ROSSO quando muore ed il colore GIALLO quando invece riesce a raggiungere la propria casa.

In questo programma utilizzeremo il colore NERO (0) per lo sfondo (PAPER) mentre i bordi della strada e i punteggi vengono visualizzati in colore BIANCO (7).

Per i mezzi di locomozione che compaiono sulla via il colore dello sfondo (PAPER) sara' ancora nero (0) mentre il colore della sagoma (INK) sara' definito dai bytes del database relativi agli attributi.

Prima di procedere al caricamento dei dati in memoria e alla loro successiva memorizzazione su cassetta, vogliamo sviluppare completamente un esempio per verificare che abbiate ben capito il procedimento per organizzare correttamente il database.

Prendiamo pertanto in considerazione il listato assembler relativo al

caricamento dei dati che rappresentano la sagoma della rana (FROG1) partendo dalla linea 160.

Alla linea 160 trovate:

```
69B7 6F 160 FROG1 DB 111,15,31,159,220,216,120,48
      0F 1F 9F DC D8 78 30
```

69B7 rappresenta un indirizzo di memoria in formato esadecimale

6F e' il valore del primo degli otto bytes che compongono la figura in formato esadecimale

160 e' il numero di linee dell'istruzione nel listato Assembler

FROG1 e' un richiamo (label) che e' stato inserito per facilitarvi la comprensione del modulo

DB e' un simbolo mnemonico che sta ad indicare che i valori che seguono sono delle costanti da caricare in memoria.

111, 15, 31, 159, 220, 216, 120, 48 sono i bytes da caricare in memoria.

La sagoma FROG1 e' rappresentata dai seguenti valori:

00	00000000	00000000	00
01	00000001	10000000	80
23	00100011	11000100	C4
25	00100101	10100100	A4
6F	01101111	11110110	F6
4F	01001111	11110010	F2
0F	11011111	11111011	FB
FF	11111111	11111111	FF
6F	01101111	11110110	F6
0F	00001111	11110000	F0
1F	00011111	11111000	F8
9F	10011111	11111001	F9
DC	11011100	00111011	3B
D8	11011000	00011011	1B
78	01111000	00011110	1E
30	00110000	00001100	0C

Richiami sul modo di operare:

1. disegniamo innanzitutto la prima riga in basso da sinistra a destra.
2. disegniamo poi la riga successiva del primo carattere.
3. Per ciascun carattere disegniamo ben 8 bytes disposti dall'alto in

basso.

4. Per ultimo, assegnamo i valori agli attributi.

FRGSHP definisce dalla linea 120 ciascuno dei quattro puntatori, che puntano rispettivamente al primo byte delle quattro sagome della rana, permettendoci, nel corso del programma, di trovare in qualsiasi momento le sagome relative ai movimenti che la rana deve effettuare.

DEFW e' un simbolo mnemonico, ed indica che vengono definiti valori a 16 bit. Il primo byte rappresenta la parte meno significativa ed il secondo indica invece la piu' significativa di tale valore.

**** Caricamento dei dati per le sagome ****

Per il caricamento dovete usare il programma Hexload e caricare il codice esadecimale relativo alle linee dalla 120 alla 1590. Attenzione: dovete caricare solo i codici esadecimali, divisi per byte, presenti nella seconda colonna del listato assembler!

Ricordatevi anche di verificare la correttezza di quanto avete caricato e di salvarlo poi su cassetta, prima di passare alla fase successiva.

**** Progetto del database degli oggetti ****

Abbiamo deciso di creare durante il gioco un flusso regolare di sei veicoli nei due sensi di marcia e di distribuire i veicoli stessi casualmente nei due sensi.

Il database degli oggetti ci fornira' informazioni sullo stato del traffico in un certo istante.

Per esempio, per ogni oggetto noi abbiamo bisogno di sapere:

se esiste, quanti movimenti ha fatto, la direzione del moto, se e' parzialmente presente sul video o no, il valore del puntatore di posizione, la posizione della sagoma nel database delle sagome, la posizione dei bytes di attributi nel relativo database, il numero di righe e di colonne che formano la sagoma.

Queste informazioni vengono modificate nel database ad ogni ciclo.

I primi sei gruppi di dati che compongono il database (lin. 1710-2040) rappresentano i sei veicoli che stanno percorrendo la strada. Quando un veicolo esce dallo schermo, casualmente ne viene generato un altro.

Un modo abbastanza semplice per gestire il database consiste nel pre-

parare le informazioni iniziali relative ad ogni veicolo possibile e conservarle in memoria. Quando viene generato un nuovo veicolo, ne preleviamo i relativi dati, ristrutturando con essi il database.

Lo stesso principio sara' usato per gestire il movimento delle auto appartenenti alla Polizia e quello della Rana.

Si tratta quindi di un database in cui la parte relativa ai veicoli in movimento (database transitorio) viene definito volta per volta dal programma. Del database transitorio pertanto verra' definita solo la struttura.

* Mappa di memoria del database transitorio *

Formato: per ciascuno dei sei veicoli, per la rana e per la macchina della polizia le informazioni contenute nel database sono:

Esistenza	DEFB	1 byte
Contatore	DEFB	1 byte
Direzione	DEFB	1 byte
Reale/Astratto	DEFB	1 byte
Posizione	DEFW	2 bytes
Puntatore alla sagoma	DEFW	2 bytes
Attributi	DEFW	2 bytes
Righe	DEFB	1 byte
Colonne	DEFB	1 byte

	TOTALE	12 bytes

label	n.linea	da(H)	a(H)	da(D)	a(D)
-----	-----	-----	-----	-----	-----
0B1EXT	1710	6E25H	6E30H	28197	28208
0B2EXT	1800	6E31H	6E3CH	28109	28220
0B3EXT	1850	6E3DH	6E48H	28221	28232
0B4EXT	1900	6E49H	6E54H	28233	28244
0B5EXT	1950	6E55H	6E60H	28245	28256
0B6EXT	2000	6E61H	6E6CH	28257	28268
PCAREXT	2070	6E6DH	6E78H	28269	28280
FRGEXT	2180	6E79H	6E80H	28281	28288

Come abbiamo gia' detto prima, queste sono soltanto delle aree di lavoro temporaneo. Le informazioni in esse contenute variano nel tempo, durante il gioco.

Vi sono anche altre due aree che contengono dati variabili e sono quelle usate per memorizzare quello che rispettivamente c'e' sotto la rana e sotto la macchina della polizia.

label	n.linea	da(H)	a(H)	da(D)	a(D)
FRGSTR	1650	6D89H	6DACH	28041	28076
PCSTR	1660	6DADH	6F24H	28077	28196

Anche in questo caso non dobbiamo definire i valori da inserire in queste locazioni, ma solo le loro posizioni.

Dobbiamo invece definire per intero il seguente insieme di dati:

FRGDB	frog database
DBINDEX	indice del database per i veicoli
RBDB	moto che procede verso destra
LBDB	moto che procede verso sinistra
RCDB	auto che procede verso destra
LCDB	auto che procede verso sinistra
RTDB	camion che procede verso destra
LTDB	camion che procede verso sinistra
LPCDB	auto della polizia che procede verso sinistra
LPCATT	attributi delle auto della polizia verso sinistra
RPCDB	auto della polizia che procede verso destra
RPCATT	attributi delle auto della polizia verso destra

label	n.linea	da(H)	a(H)	da(D)	a(D)	checksum
FRGDB	2260	6E81H	6E88H	28289	28296	561
DBINDEX	2320	6E89H	6E94H	28297	28308	1734
RBDB	2400	6E95H	6EAOH	28309	28320	640
LBDB	2470	6EA1H	6EACH	28321	28332	692
RCDB	2540	6EADH	6EB8H	28333	28344	523
LCDB	2610	6EB9H	6EC4H	28345	28356	760
RTDB	2680	6EC5H	6ED0H	28357	28368	584
LTDB	2750	6ED1H	6EDCH	28369	28380	809
LPCDB	2820	6EDDH	6EE8H	28381	28392	955
LPCATT	2890	6EE9H	6EF4H	28393	28404	30
RPCDB	2930	6EF5H	6F00H	28405	28416	379
RPCATT	3000	6F01H	6F0CH	28417	28428	30

Modulo di memoria tra 28289 e 28428 di 140 bytes con checksum di 7687.

Nome del modulo: "objdb". (object database)

Osserviamo che tutti gli oggetti, fatta eccezione per la rana, hanno a disposizione nel database 12 bytes.

Specifichiamo ora il significato di ognuno di essi:

* Esistenza (1 byte)

- ha il valore zero quando l'oggetto non esiste.
- ha il valore n quando (n-1) e' il numero di cicli che il veicolo deve aspettare tra un movimento e l'altro.

n vale:	2	per le moto
	3	per le auto
	6	per i camion
	1	per le auto della polizia
	8	per la rana

in altre parole, l'auto della polizia si muove ad ogni ciclo, le moto a cicli alternati, ecc. ...

* Contatore di ciclo (1 byte)

- e' inizialmente posto ad 1, per segnalare che il movimento puo' cominciare, e viene decrementato ad ogni ciclo.
- quando assume valore zero l'oggetto compie un movimento, ed il contatore viene riinizializzato con il valore n di cui si e' parlato sopra.

* Direzione (1 byte)

- tutto il traffico che va da sinistra a destra (parte alta dello schermo) ha questo byte posto a zero, mentre quello inverso, che va da destra a sinistra, ha il byte posto ad 1.

* Flag Astratto/Reale (1 byte)

- questo flag definisce se gli oggetti sono in parte fuori dal video.
- per tutti i veicoli che viaggiano da sinistra a destra e' inizializzato a zero (astratto).
- per i veicoli che viaggiano da sinistra a destra assumerà il valore 1 quando il puntatore di posizione assume il valore 4820H.
- per tutti i veicoli che viaggiano da destra a sinistra e' inizializzato ad uno (reale) ed il puntatore di posizione e' posto a 48DFH.
- quando la sagoma esce dallo schermo, cioe' quando il puntatore di posizione passa da 48C0 e 48BF il flag passa da uno a zero.

* Puntatore di posizione (2 bytes)

- questi due bytes vengono utilizzati per memorizzare la posizione corrente del veicolo.

* Puntatore di sagoma (2 bytes)

- questi due bytes individuano l'indirizzo di partenza del blocco di dati nel database delle sagome relative all'oggetto.

* Puntatore agli attributi (2 bytes)

- questi due bytes rappresentano l'indirizzo di partenza del blocco di dati relativo agli attributi di colore dell'oggetto.

* Righe (1 byte)

- rappresenta il numero di righe usato per disegnare la sagoma dello oggetto.

* Colonne (1 byte)

- rappresenta il numero di colonne usato per disegnare la sagoma.
- questo valore include anche due colonne di blank poste alla fine di ogni figura. Cio' permette di distanziare ragionevolmente i vechi tra loro.

Siamo pronti adesso a caricare questo nuovo blocco di programma (linee 2270-3010).

Per caricarlo, potete usare a vostra scelta uno dei due programmi Monitor (EZ-Code ed Hexload).

Se optate per il programma EZ-Code, ricordatevi di salvare su cassetta il programma sorgente, nonche' il programma oggetto ottenuto dopo un DUMP.

**** Database generale ****

Abbiamo per ora utilizzato per il nostro database le locazioni da 69AFH a 6FOCH (27055 a 28428).

Completiamo adesso il database con delle informazioni di carattere generale, relative al suono, al punteggio ed a funzioni generali.

Esse sono organizzate in questo modo:

dalla linea 500 alla 630	SOUND (suono)
dalla linea 660 alla 690	SCORE (punteggio)
dalla linea 710 alla 1210	GENERAL (funzioni generali)

label	n.linea	da(H)	a(H)	da(D)	a(D)	checksum
PCTON1	500	6F0DH	6F10H	28429	28432	282
PCTON2	510	6F11H	6F14H	28433	28436	166
HOMTON	540	6F15H	6F3CH	28437	28476	2565
SCRMS1	660	6F3DH	6F42H	28477	28482	540
SCORE	670	6F43H	6F48H	28483	28488	290
SCRMS2	680	6F49H	6F53H	28489	28499	732
HISCR	690	6F54H	6F58H	28500	28504	243

Questo modulo occupa 76 bytes (da 28429 a 28504) ed ha un checksum 4818.

Nome assegnatogli "gendb". (general database).

E' necessario caricare solo le linee dalla 500 alla 690.

Dalla linea 790 alla 1210, infatti, vengono definite delle variabili (locazioni da 6F59H a 6F82H) che vengono inizializzate direttamente dal programma.

In particolare, le linee dalla 1100 alla 1150 sono istruzioni con codice mnemonico EQU e servono per assegnare un valore al label specificato. Tali istruzioni servirebbero al programma assemblatore per definire gli indirizzi: voi non dovete caricare niente!

Conclusioni

L'intero database occupa le locazioni comprese tra 69AFH e 6F82H (27055 e 28546).

Controllate attentamente tutti i moduli che avete costruito, i loro nomi e le aree di memoria usate, prima di passare alla fase successiva del progetto del programma FREEWAY FROG.

Sono stati sviluppati tre moduli:

nome	da mem	a mem	lung.	checksum
-----	-----	-----	-----	-----
shpdb	27055	28040	986	79197
objdb	28289	28428	140	7697
gendb	28429	28504	76	4818

Notate che il database occupa ben 1400 bytes!!

FREEWAY FROG Fase 2 (inizializzazione)

**** Organizzazione del video ****

In questo modulo dobbiamo predisporre la strada, la visualizzazione del punteggio e delle rane, nonché inizializzare tutte le variabili di controllo.

Suddivideremo il modulo in tre parti.

Primo: cancellazione del video e visualizzazione della strada.

Secondo: predisposizione delle rane.

Terzo: visualizzazione del punteggio.

Questo modulo e' formato dalle seguenti funzioni:

routine	n.linea	da(H)	a(H)	da(H)	a(H)	checksum
INIT	1240	6F83H	700AH	28547	28682	11996
CLRSCR	7060	72D7H	7316H	29399	29462	5236
DRWHWY	1820	700BH	7040H	28685	28736	4609
HIGWY	1980	7038H	7040H	28728	28736	696
FILHWY	2070	7041H	7054H	28737	28756	2609
LINEUP	2290	7055H	7059H	28757	28793	4325
DISASC	7630	7328H	7349H	29480	29513	3580
SCRIMG	14500	776FH	778FH	30575	30598	1855
FINAL	15390	77FEH	781DH	30718	30747	2089

occupa 2201 bytes (da 28547 a 30747) ed e' denominato "init" (initialisation).

Caricate in ordine le funzioni CLRSCR, DRWHWY, FILHWY, FINAL e poi la funzione INIT. Nel fare cio' ponete tre bytes di zeri nelle linee sottoidicate, rappresentanti istruzioni di chiamata a funzioni non ancora sviluppate. Cio' vi permettera' di provare immediatamente la correttezza del modulo inserito.

n.linea	indirizzo(H)	indirizzo(D)
1430	6FAFH	28591
1470	6FBAH	28602
1490	6FC0H	28608
1530	6FCBH	28619
1570	6FD6H	28630
1590	6FDCH	28636
1630	6FE7H	28647

Salvate su cassetta il modulo da 28547 a 30594 (2052 bytes), e caricate a partire dalla locazione 32000 il seguente programma, lanciandone poi l'esecuzione:

```

F3      DI                ;Disabilita gli interrupt
D9      EXX               ;salva HL
E5      PUSH             HL
D9      EXX
CD836F  CALL             INIT
3E7F   KEY LD            A, 7FH      ;maschera per il tasto SPACE
DBFE   IN               A, (FEH)
E601   AND              1
2DF8   JR              NZ,KEY      ;cicla finche' non e' premuto
CDFE77 CALL             FINAL      ;fase finale
D9      EXX               ;ripristina HL
E1      POP              HL
D9      EXX
FB      EI                ;riabilita gli interrupt
C9      RET

```

Dovreste vedere lo schermo annerito e su di esso alcune linee bianche.

Qui di seguito vi forniamo la spiegazione di come opera ciascuna di queste routines.

INIT

```

pone il colore di sfondo nero
inizializza gli indicatori di crash e d'esistenza per la rana, il
numero di rane per partita
inizializza il valore per la funzione di randomizzazione
pone la postazione (o la posizione iniziale della rana) a 50ACH
chiama la procedura di cancellazione video
chiama la procedura di disegno della strada
chiama la procedura di allineamento delle rane (5 di esse)
prepara il messaggio per il punteggio
visualizza il messaggio
inizializza tutti i veicoli come inesistenti
inizializza l'indicatore per il suono della sirena e per il pun-
teggio

```

DRWHWY

```

riempie la riga corrispondente al bordo superiore della strada (32
caratteri da 40A0H)
riempie la linea di separazione tra le corsie (32 caratteri da
4860H)
riempie la riga corrispondente al bordo inferiore della strada (32
caratteri da 5020H)

```

* ricordiamo che gli attributi della strada definiscono il colore bianco per la carta e nero per l'inchiostro.

"vuota" i due bytes bassi di ciascun carattere del bordo alto della strada (che danno luogo ad una riga bianca)

"vuota" i due bytes alti di ciascun carattere del bordo basso della strada (che danno luogo ad un'altra riga bianca)

ridisegna la linea intermedia, modificando i due bytes intermedi di ciascun carattere

FILHWY

inizializza il codice del carattere (FFH)
inizializza il contatore a 32 (32 caratteri su una riga)
disegna un carattere (8 bytes)
sposta ogni volta il puntatore al carattere successivo

FINAL

pone l'attributo BORDER a bianco
cancella il video
pone nel file degli attributi i codici bianco per PAPER e nero per INK

Se tutto funziona a puntino, salvate il modulo contenuto nelle locazioni da 28500 a 30800 (2300 bytes).

Inserite ora le routines LINEUP e DRWFRG. Controllate l'esattezza dei dati con il checksum e salvate ancora una volta l'intero modulo con lo stesso nome e gli stessi indirizzi.

Cambiate ora i bytes da 6FAFH (28591) a 6FB1H (28593) in modo che corrispondano alla linea 1430 del listato (CD 55 70).

Lanciate l'esecuzione. Dovreste a questo punto vedere 5 rane allineate sulla parte bassa a sinistra del video.

Diamo ora la descrizione delle due nuove routines utilizzate:

LINEUP

pone il byte di direzione della rana ad 1 (verso destra)
punta la sagoma FROG2
pone a 2 l'attributo di colore della sagoma (verde)
se non vi sono piu' rane
 allora return
altrimenti
 per ogni rana
 salva BC, DE ed HL nello stack
 disegna la rana (routine DRWFRG)
 recupera BC, DE ed HL dallo stack
 si posiziona per un nuovo disegno

DRWFRG

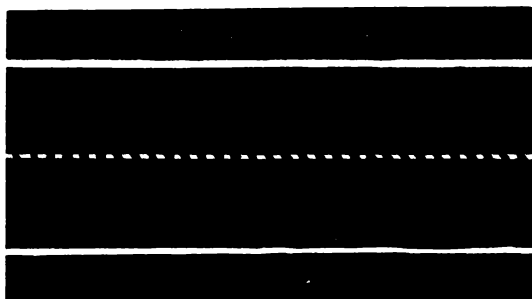
disegna la figura usando le convenzioni citate nelle pagine precedenti
calcola il puntatore agli attributi
colora la rana

Caricate ora le routines DISASC e SCRIMG, eseguite un test sul checksum e salvate ancora una volta l'intero modulo, come prima.

Ora modificate i bytes relativi alle linee 1470, 1490, 1530, 1570, 1590 e 1630 del listato in modo che contengano i valori corretti.

Rilanciate l'esecuzione e vedrete finalmente l'intero schermo predisposto con il tracciato della strada, le cinque rane e la visualizzazione del punteggio.

Se tutto funziona alla perfezione, salvate su cassetta l'intero modulo "init".



FREEWAY FROG Fase 3 (traffico regolare)

In questa fase viene sviluppato il modulo che realizza il traffico regolare dei veicoli, ad eccezione delle auto della polizia.
Tale modulo si sviluppa nel modo seguente:

- Controllo del traffico (inclusa la rigenerazione di veicoli)
 - predispone il blocco dei veicoli nei due sensi
- Movimento dei mezzi
 - controlla il movimento
 - disegna i veicoli
 - sceglie le opportune sagome

Le routine usate in questo modulo sono:

nome	n.linea	da(H)	a(H)	da(H)	a(H)	checksum
TFCTRL	3090	70BDH	7008H	28861	28888	2587
REGEN	3320	70D9H	710EH	28889	28942	5673
MOVTRF	3700	710FH	71AEH	28943	29102	14831
MVCTRL	4720	71AFH	7208H	29103	29192	9222
DRAW	5560	7209H	7295H	29193	29333	13923
RSHAPE	6630	7296H	72D6H	29334	29398	6803
RANDNO	15050	77CCH	77DDH	30668	30685	2194

Il modulo, chiamato "regtrf" (regular traffic), occupa 1824 bytes (da 28861 a 30685).

Ancora una volta voglio raccomandarvi di non limitarvi ad eseguire un checksum solo sull'intero modulo, bensì di caricare e testare singolarmente ciascuna routine.

Sviluppiamo il modulo dividendolo in due parti: una relativa al disegno dei mezzi in movimento, l'altra relativa al controllo.

Caricate le routines DRAW e RSHAPE nelle zone di memoria a loro assegnate, eseguite poi per ciascuna il checksum e salvatele.

Vi presentiamo ora il programma che permette di controllare la correttezza delle due routine.

```

F3          DI
D9          EXX
E5          PUSH    HL
D9          EXX
C0836F     CALL    INIT
3E03       LD      A, 3          ;contatore di righe
32606F     LD      (ROW), A      ;memorizza in ROW
3E09       LD      A, 9          ;contatore di colonne
325F6F     LD      (COLUMN), A   ;memorizza in COLUMN
11926C     LD      DE, RTRUCK    ;sagoma del camion a destra
216A6D     LD      HL, RTATT     ;attributi
226A6F     LD      (ATTPTR), HL  ;memorizza in ATTPTR
3E01       LD      A, 1          ;pos: reale
212248     LD      HL, 4822H     ;corsia in alto
C00972     CALL    DRAW         ;disegna la sagoma
3E7F     KEY  LD      A, 7FH     ;maschera per la tastiera
DBFE     IN      A, (OFEH)
E601     AND     1
20F8     JR      NZ, KEY
CDFE77     CALL    FINAL
D9          EXX
E1          POP     HL
D9          EXX
FB          EI
C9          RET

```

Caricate i moduli di database nell'ordine in cui sono stati creati.

Caricate il modulo init.

Caricate le due routines sviluppate in questo modulo.

Avete generato così un modulo più ampio, a cui assegnamo il nome di "frog". Tale modulo occupa a partire dalla locazione 27000, circa 4000 bytes. Lanciandone l'esecuzione, sarete in grado di provare tutte le routines fin qui sviluppate.

Caricate, a partire dalla locazione 32000, il programma presentato qui sopra e lanciatene l'esecuzione.

Dovrebbe comparire il video inizializzato, come nella prova precedente, con l'aggiunta però della sagoma di un camion che va verso destra nella corsia in alto.

Se volete provare le routines DRAW ed RSHAPE per altri oggetti, potete modificare i parametri di ingresso prima delle CALL INIT e CALL DRAW.

Diamo ora una breve descrizione delle due nuove routine utilizzate.

DRAW

Ha una logica simile a quella di DRWFRG

RSHAPE

maschera i 5 bit meno significativi del byte meno significativo
del parametro di posizione
sottrae il risultato da 1FH ed aggiunge 1
maschera ancora gli ultimi 5 bit
determina SKIP e FILL tenendo conto dell'indicatore reale/astratto
ricerca la posizione nel file attributi e la memorizza in ATTP05

Caricate ora le routines di controllo TFCTRL, REGEN e MVCTRL, collegandole con il resto del modulo e salvate il modulo ottenuto.

Per testarne la correttezza, usate poi il seguente programma di prova:

```
DI
EXX
PUSH    HL
EXX
CALL    INIT
CDBD7D  MOVE  CALL    TFCTRL
CDDF71  CALL    MOVTRF
3E7F    LD     A, 7FH
DBFE    IN     A, (OFEH)
E601    AND    1
2DF2    JR     NZ, MOVE
CALL    FINAL
EXX
POP     HL
EXX
EI
RET
```

D'ora in poi, ogni volta che concluderemo un modulo, non lo salveremo piu' separatamente su cassetta, ma lo collegheremo con quelli sviluppati precedentemente e lo salveremo come "frog". Il modulo "frog", cosi', nel corso dell'analisi si sviluppera' diventando via via sempre piu' ampio.

In tal modo, ogniqualvolta verra' completato e collegato ai precedenti un nuovo modulo, si potra' testare il nuovo modulo "frog" con piccoli programmi di prova che partono dalla locazione 32000.

Nel caso in esame, se tutto e' stato fatto correttamente, vedremo lo intero video come nella precedente prova, ed in aggiunta il traffico in movimento ad una velocita' supersonica sulle due corsie di marcia. La velocita' eccessiva e' motivata col fatto che non abbiamo ancora introdotto nel programma degli elementi di ritardo tra un ciclo e lo altro.

Diamo ora una descrizione sommaria delle varie routines costituenti il modulo in oggetto.

TFCTRL

```
controlla l'indicatore di generazione
se non segnala necessita' di rigenerazione
  decrementa il contatore
  return
altrimenti
  rigenera il primo oggetto "non esistente" chiamando la routine
    REGEN
return
```

REGEN

```
salva il puntatore ai dati relativo all'esistenza
genera un numero casuale tra 0 e 5
controlla i primi due caratteri della posizione del video in cui
dovrebbe essere generato il nuovo oggetto
se la somma degli attributi di questi due caratteri non e' uguale
a zero
  allora return (vi e' gia' un altro oggetto)
altrimenti
  determina i dati per l'inizializzazione
  li carica nel database temporaneo di lavoro
  pone il contatore di cicli uguale a 2
  return
```

MOVTRF

```
se e' stato raggiunto il bordo del video
  cambia il valore dell'indicatore reale/astratto
se e' un movimento verso sinistra
  se e' raggiunto il bordo iniziale dello schermo (posizione con
  la parte bassa dell'indirizzo 1FH)
    se il flag astratto/reale segnala astratto
      pone la condizione di non esistenza, return
    altrimenti
      goto l1
  altrimenti
    goto l1
altrimenti
  cerca la fine dell'oggetto
  se ha raggiunto la fine dello schermo (parte bassa dell'indi-
  rizzo 0COH)
    pone la condizione di non esistenza, return
l1: riinizializza il contatore di cicli
ripristina il puntatore alla sagoma
memorizza il puntatore agli attributi in ATTPTR
ripristina righe (ROW) e colonne (COLUMN) della sagoma
disegna (DRAW) le sagome nella nuova posizione
```

RANDNO

salva nello stack HL e BC

preleva il valore puntato dal puntatore di numeri casuali in ROM

aggiorna il puntatore

score 0 HIGH SCORE 0



score 0 HIGH SCORE 0



FREEWAY FROG Fase 4 (auto della polizia)

Vogliamo ora introdurre nel traffico anche le auto della polizia. L'auto della polizia (POLICE) e' generata casualmente ed entra in scena con il suono della sirena. Essa si muove ad ogni ciclo e non vi e' nulla che fermi la sua corsa: sorpassa qualsiasi mezzo trovi davanti a se' cancellandolo letteralmente.

Per questo il programma deve ricordare cosa vi era nei singoli caratteri che vengono coperti dalla sagoma dell'auto della polizia in fase di sorpasso, in modo da poterlo ripristinare dopo il passaggio della auto stessa.

Il modulo e' praticamente composto da tre routines:

nome	n.lines	da(H)	a (H)	da(D)	a (D)	checksum
RESPC	9560	7450H	74C1H	29776	29889	11011
POLICE	7930	734AH	73DEH	29514	29662	15769
STRPC	8830	73DFH	744FH	29663	29775	10615

Nello sviluppo del modulo verranno poi richiamate altre routines sviluppate in precedenza.

Il modulo, a cui abbiamo assegnato il nome "police" occupa solo 376 bytes (da 29514 a 29889)

Caricate ora le nuove routines POLICE e STRPC, testatene il checksum e unitele in un unico modulo di nome "police" che salverete su cassetta. Caricate poi il seguente programma di prova:

```

DI
EXX
PUSH HL
EXX
CALL INIT
MOVE      CALL TFCTRL
          CALL MOVTRF
CD4A73    MOVE1  CALL POLICE
LD        A, 7FH
IN        A, (FEH)
AND       1
20F5     JR      NZ, MOVE1
          CALL FINAL
EXX
POP      HL
EXX
EI
RET
    
```

Caricate ora da cassetta i moduli "frog" e "police" e lanciatene l'esecuzione dalla locazione 32000: dovrete vedere sfrecciare un'auto della polizia sulla strada. Se volete poi far muovere anche gli altri mezzi presenti sulle corsie, cambiate il salto relativo in JR NZ,MOVE. Cio' equivale a trasformare il codice di salto ricalcolando il parametro d che diventa EFH.

Questa volta dovrete vedere l'auto della polizia sorpassare i vari mezzi, cancellandone gradatamente la sagoma. Ma quest'operazione avverra' talmente velocemente che non vi accorgete neppure di questa gradualita'.

Potrete comunque sempre raccontare di aver visto un'auto della polizia sfrecciare sulla strada eliminando tutti i veicoli che avevano la sfortuna di trovarsi davanti.

Diamo ora una breve descrizione della struttura logica delle nuove routines introdotte.

POLICE

- se l'auto della polizia "non esiste"
 - genera un numero casuale
 - se non e' un multiplo di 31
 - return
 - altrimenti
 - pone a 1 l'indicatore di inseguimento
 - determina casualmente la corsia
 - preleva dal database i dati per l'inizializzazione
- determina la direzione
- memorizza il puntatore di posizione
- ricalcola la posizione
- conserva il valore ottenuto in NEWPOS
- determina il valore di ROW, COLUMN, POS e dell'indicatore reale /astratto prima di chiamare la routine RSHAPE
- controlla se ATTPOS punta a una posizione con attributo di colore pari a verde
- se e' verde
 - pone a 1 l'indicatore di crash
 - attribuisce l'attributo bianco alla posizione davanti alla macchina
- chiama la routine STRPC (per memorizzare i dati relativi alla nuova posizione occupata dall'auto)
- chiama la routine MVCTRL (per controllare la posizione sullo schermo)
- se e' verificata la condizione "non esiste" pone a zero l'indicatore di inseguimento

STRPC

- pone in HL l'indirizzo di NEWPOS
- pone in DE l'indirizzo di PCSTR (police car store)
- conserva il valore di ROW e i successivi 5 bytes di informazione
- conserva i dati del display file e del file attributi per poter ricostruire il quadro video

Caricate ora la routine RESPC e collegatela con le altre che compongono il modulo "police" e verificate la correttezza lanciando l'esecuzione di un programma di prova simile al precedente.

Notiamo per inciso che la prova fatta sul precedente modulo "police" non ha permesso di verificare la correttezza della routine STRPC dato che i dati da essa memorizzati non venivano piu' riutilizzati. Con questa nuova prova quindi otteniamo il duplice scopo di verificare la correttezza della routine di ripristino RESPC e della routine di salvataggio STRPC.

Per effettuare la prova del nuovo modulo "police" modificate il programma di prova nel modo seguente:

```

                .....
                .....
CD5074      MOVE   CALL   TFCRTL
                CALL   RESPC   (-----)
                CALL   MOVTR
                CALL   POLICE
                LD     A,7FH
                IN    A,(FEH)
                AND   1
                JR    NZ, MOVE
                .....
                .....
```

Ancora una volta si e' reso necessario modificare il valore del parametro nell'istruzione di salto relativo.

Questa volta lanciando l'esecuzione vedrete l'auto della polizia sorpassare i vari mezzi senza pero' cancellarli.

La logica con cui opera la routine RESPC e' la seguente:

```
RESPC
  se l'auto della polizia "non esiste" return
  ripristina il valore di posizione e i 5 bytes salvati partendo da
  ROW
  ripristina i valori del display file e del file attributi secondo
  il formato SKIP/FILL
  return
```

Caricate ora la routine SIREN per azionare la sirena collegandola con le precedenti.

Salvate per sicurezza la nuova versione del modulo "police" e lanciate l'esecuzione modificando il programma di prova come segue:

```

      . . . . .
      . . . . .
      CALL  INIT
MOVE   CALL  TFCTRL
      CALL  RESPC
      CALL  MOVTRF
      CALL  POLICE
CD8777 CALL  SIREN      (-----)
      LD    A, 7FH
      IN   A, (FEH)
      AND  1
      JR   NZ, MOVE
      . . . . .
      . . . . .

```

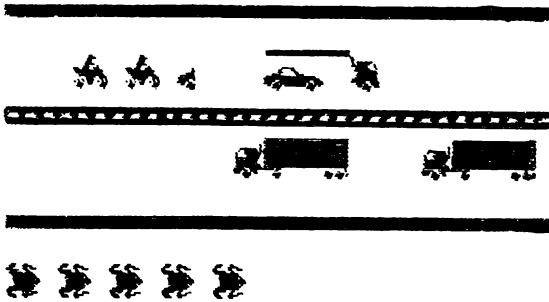
Nel corso dell'esecuzione vedrete passare i veicoli piu' lentamente delle altre volte. Cio' e' dovuto al fatto che la routine per la generazione del suono introduce un elemento di ritardo supplementare allo interno di ciascun ciclo.

SIREN

- controllo sullo stato del tasto <ENTER>
- se <ENTER> e' premuto
 - cambia "suona" in "non suona" o viceversa
- se "non suona"
 - salta a DELAY (ciclo di ritardo)
- altrimenti
 - determina il tono del suono
 - carica i dati necessari in HL e DE
 - chiama la routine di sistema 03B5H
 - return

DELAY conta fino a 6144

Collegate il modulo "police" con il modulo "frog" e salvate tutto come un nuovo modulo "frog".



FREEWAY FROG Fase 5 (la rana)

In questo modulo svilupperemo le routines relative ai movimenti della rana.

Innanzitutto dovremo preoccuparci di predisporre per il movimento una nuova rana ogni volta che quella in attivita' muore.

Occorrera' poi sviluppare delle opportune routines che permettano allo utente di governare da tastiera i movimenti della rana, che conservino i dati delle locazioni occupate temporaneamente dalla rana durante il suo moto e che ripristino la situazione esistente prima dell'arrivo della rana, dopo il suo passaggio.

E bisognera' infine preoccuparsi di costruire una routine che gestisca gli scontri tra la rana e gli automezzi, gli arrivi a casa e che calcoli i punteggi.

Data la complessita' del modulo affronteremo separatamente i diversi problemi suddividendo l'analisi in tre parti:

- inizializzazione e movimento della rana
- gestione del video durante il passaggio della rana
- gestione degli incidenti, degli arrivi a casa e del punteggio.

Il modulo completo prendera' il nome di "frgrtn" (frog routine), occupera' 685 bytes (da 29890 a 30574) e sara' composto dalle seguenti routines:

nome	n.linea	da(H)	a (H)	da(D)	a (D)	checksum
FROG	10280	74C2H	74E2H	29890	29922	3818
REGFRG	10520	74E3H	750FH	29923	29967	4079
MOVFRG	10770	7510H	75D5H	29968	30165	19943
RESFRG	11870	75D6H	7627H	30166	30247	8492
STRFRG	12440	7628H	7690H	30248	30352	10136
CRASH	13160	7691H	76A6H	30353	30374	2767
FRGDIE	13280	76A7H	7707H	30375	30471	9965
FRGTON	13890	7708H	771CH	30472	30492	2435
CALSCR	14040	771DH	776EH	30493	30574	8106

Caricate ora le routines FROG, REFRG, MOVFRG, RESFRG, STRFRG e CRASH e modificate il programma di prova come segue:


```

      . . . . .
      . . . . .
MOVE   CALL  INIT
      CALL  TFCTRL
      CALL  RESPC
      CALL  MOVTRF
      CALL  POLICE
      CALL  FROG      (-----)
      CALL  SIREN
      LD    A, 7FH
      IN   A, A, (FEH)
      AND  1
      JR   NZ, MOVE
      . . . . .
      . . . . .

```

Dato che non abbiamo ancora caricato la routine FRGDIE, sostituiamo il codice della linea 13190 con

```
00 00 00
```

Lanciando l'esecuzione potrete far muovere a vostro piacimento la rana usando i quattro tasti l,a,i,p come segue:

"l" = su "a" = giu' "i" = sinistra "p" = destra

In caso di incidente la rana verra' solo eliminata dallo schermo dato che la routine per la gestione della morte della rana non e' ancora stata realizzata.

Descriviamo ora le routines di questo modulo fin qui utilizzate.

FROG

questa routine controlla lo sviluppo dell'intero modulo.

se la rana ha avuto un incidente

 salta a CRH

altrimenti

 poni l'indicatore di punteggio a zero

 chiama la routine REGFR

 decrementa il contatore di cicli

 se il contatore non e' zero

 return

altrimenti

 riinizializza il contatore di cicli

 chiama la routine MOVFRG

 se non ha provocato un incidente

 return

CRH chiama la routine CRASH

 return

REGFRG

se la rana "non esiste"
carica nel database di lavoro i dati necessari prelevandoli
dal database iniziale
posiziona la rana al punto di partenza
inizializza OLDFRG e NEWFRG con FRGPOS
inizializza l'area di memoria destinata alla rana a zero
return

MOVFRG

inizializza i registri come segue:
C - movimento
B - direzione della rana
DE - puntatore alla sagoma della rana
controlla il movimento scelto per la rana:
i - su, a - giu', l - sinistra, p - destra
memorizza i dati relativi alla direzione e alla sagoma
se C=0
return
altrimenti
considera la vecchia posizione della rana
calcola la nuova posizione e memorizzala
controlla la validita' della posizione rispetto allo schermo
se e' accettabile
memorizzala in NEWFRG
poni a 1 l'indicatore di punteggio
controlla la vecchia posizione
se OLDFRG e' uguale a NEWFRG
return
altrimenti
chiama la routine RESFRG
poni OLDFRG=NEWFRG
realizza il movimento tenendo conto dei dati fin qui acquisiti
chiama la routine STRFRG
return

RESFRG

ripristina sul video nella posizione OLDFRG la situazione esi-
stente prima dell'arrivo della rana ricostruendo il display
file e il file degli attributi

STRFRG

conserva i dati del display file e degli attributi relativi alle
nuove posizioni occupate dalla rana per poterle riutilizzare
dopo il suo passaggio

CRASH

azzerà l'indicatore di crash
poni l'indicatore di esistenza su "non esiste"
chiama la routine FRGDIE (per la morte della rana)
chiama la routine RES'XG (per ripristinare la situazione sul video)
decrementa il contatore di rane

Dopo aver salvato queste routines, caricate CALSCR, FRGDIE e FRGTON
Sostituite la linea 13190 con

```
7698   CDA776   CALL FRGDIE   (30360)
```

e modificate il programma di prova come segue:

```
.....  
.....  
CALL   FROG  
CALL   CALSCR  (-----)  
CALL   SIREN  
.....  
.....
```

Collegate tra loro le varie routines, salvate il tutto per sicurezza e lanciate l'esecuzione.

Quando la rana subisce un incidente, si illumina di rosso e svanisce.

FRGDIE

controlla se la rana ha raggiunto la casa o se e' morta
seleziona il tono triste ed il colore rosso
se la rana ha raggiunto la casa
aggiungi uno alla terza cifra del punteggio
(bonus di 100 punti)
chiama la routine DISSCR (display score)
seleziona il tono allegro ed il colore giallo
disegna la rana tenendo conto di OLDFRG, FROGSH e degli attributi chiamando la routine DRWFRG
illumina la rana con il colore prescelto per cinque volte

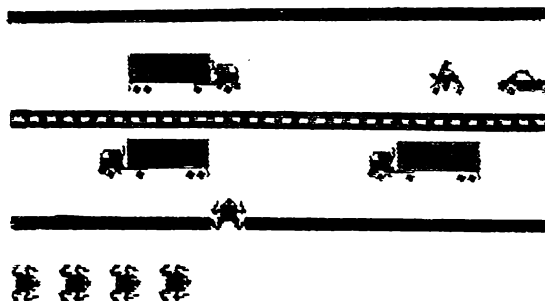
FRGTON

chiama la routine TONE1 (tono codificato dalla routine SIREN)
sposta il puntatore sulla scala dei toni in accordo con il colore selezionato per la rana (allegro se giallo)
(cupò se rosso)

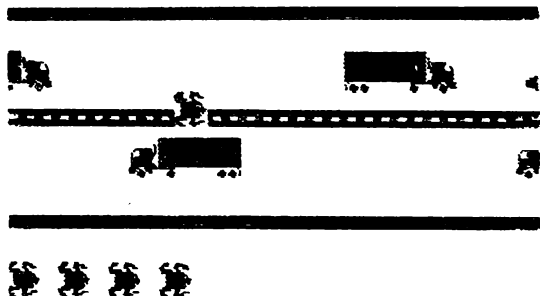
CALSCR

```
se la rana "non esiste"  
    return  
altrimenti  
    se l'indicatore di punteggio non e' settato  
        return  
    altrimenti  
        se la rana e' salita  
            aggiungi uno alla seconda cifra del punteggio  
                ( bonus di 10 punti)  
        altrimenti  
            se la rana non e' sulla strada  
                return  
            altrimenti  
                aggiungi uno alla seconda cifra del punteg-  
                    gio  
                    ( bonus di 10 punti)  
            rileggi ad una ad una le cifre del punteggio  
            predisponi i dati necessari per la sua visualiz-  
                zazione  
            visualizza il nuovo punteggio.
```

Score 0 HIGH SCORE 0



Score 100 HIGH SCORE 0



FREEWAY FROG fase 6 (controllo)

In questa fase verranno sviluppate le routines preposte al controllo dell'intero programma.

Il controllo consiste nel verificare alla fine di ogni partita se deve essere aggiornato il valore del punteggio massimo raggiunto e nel far ripartire automaticamente il programma per una nuova partita.

Per far terminare il programma sarà sufficiente battere in un momento qualsiasi il tasto (SPACE).

La struttura di questo modulo di controllo, che si sviluppa dalla linea 180 alla linea 440, è assai simile a quella dei programmi di prova utilizzati per testare le singole parti del programma.

Esso usa essenzialmente queste ultime due routines:

nome	n.linea	da(H)	a (H)	da(D)	a (D)	checksum
START	180	6978H	69AEH	27000	27054	8427
OVER	15200	77DEH	77FDH	30686	30717	2491

Caricate queste nuove routines, collegatele con il vecchio modulo "frog" e avrete finalmente ultimato il programma!

Per provarne la correttezza non vi resta altro che lanciarne l'esecuzione. Ma fate attenzione: questa volta dovrete lanciarla a partire dalla locazione 27000 e non dalla 32000!

Per concludere vi presentiamo la struttura della routine OVER:

OVER

```
confronta tutte le cifre di HISCR e SCORE+1
per la prima cifra diversa
    se la cifra di HISCR è minore di quella di SCORE+1
        sostituisci HISCR con SCORE+1
    altrimenti
        return
return
```

Congratulazioni: avete finito il vostro lavoro ed io spero proprio che siate soddisfatti per aver sviluppato con noi il programma FREEWAY FROG.

```

00100 ;***** FREENWAY FROG *****
00110 ;
00120 ;
00130 ;
00140 ;
00150 ;
6978          00160          ORG          27000
          00170 ;
6978 F3      00180 START   DI              ;DISABILITA IL CONTROLLO DELLA
6979 D9      00190        EXX              ;TASTIERA DA PARTE DEL BASIC
697A E5      00200        PUSH             HL              ;SALVA IL CONTENUTO DI HL DA
697B D9      00210        EXX              ;RIPRISTINARE AL RETURN
697C CD836F  00220 AGAIN   CALL           INIT           ;INIZIALIZZAZIONE
697F CDBD70  00230 MOVE    CALL           TFCTRL        ;CONTROLLO TRAFFICO
6982 CD5074  00240        CALL           RESPC        ;RISTRUTTURAZIONE VIDEO
6985 CD0F71  00250        CALL           MOVTRF       ;MOVIM.TRAFFICO
6988 CD4A73  00260        CALL           POLICE       ;AUTO DELLA POLIZIA
698B CDC274  00270        CALL           FROG         ;MODULO FROG
698E CD1D77  00280        CALL           CALSCR      ;CALCOLO E VIS. PUNTI
6991 CD8777  00290        CALL           SIREN       ;SIRENA O MORTE RANA
6994 3A776F  00300        LD             A,(GAMFLG) ;RANE FINITE?
6997 A7      00310        AND              A
6998 2005    00320        JR              NZ,CONTIN
699A CDDE77  00330        CALL           OVER        ;GESTIONE PT.MAX
699D 18DD    00340        JR              AGAIN     ;NUOVA PARTITA
699F 3E7F    00350 CONTIN LD             A,7FH    ;MASCHERA TASTO SPACE
69A1 DBFE    00360        IN              A,(0FEH) ; SCANS.TASTIERA
69A3 E601    00370        AND              1
69A5 20DB    00380        JR              NZ,MOVE
69A7 CDFE77  00390        CALL           FINAL      ;CANCELLA VIDEO
69AA D9      00400        EXX
69AB E1      00410        POP             HL        ;RIPRISTINO HL
69AC D9      00420        EXX
69AD FB      00430        EI              ;ABILITAZIONE INTEPPUZIONI
69AE C9      00440        RET             ;RITORNO AL BASIC
          00450 ;
          00460 ;
69AF          00470
          00100 ;***** FROGDB/ASM *****
          00110 ;
69AF B769    00120 FRGSHP  DEFW     FROG1   ;RANA SU
69B1 D769    00130        DEFW     FROG2   ;RANA DESTRA
69B3 F769    00140        DEFW     FROG3   ;RANA GIU
69B5 176A    00150        DEFW     FROG4   ;RANA SINISTRA
69B7 6F      00160 FROG1   DB       111,15,31,159,220,216,120,48
          0F 1F 9F DC DB 7B 30
69BF F6      00170        DB       246,240,248,249,59,27,30,12
          F0 FB F9 3B 1B 1E 0C
69C7 00      00180        DB       0,1,35,37,111,79,223,255
          01 23 25 6F 4F DF FF
69CF 00      00190        DB       0,128,196,164,246,242,251,255
          80 C4 A4 F6 F2 FB FF
69D7 1F      00200 FROG2   DB       31,31,31,127,252,193,113,56
          1F 1F 7F FC C1 71 38
69DF FE      00210        DB       254,244,248,240,192,156,240,192
          F4 FB F0 C0 9C F0 C0
69E7 38      00220        DB       56,113,193,252,127,31,31,31

```

71 C1 FC 7F 1F 1F 1F		
69EF C0 00230	DB	192, 240, 156, 192, 240, 248, 244, 254
F0 9C C0 F0 FB F4 FE		
69F7 FF 00240 FR0G3	DB	255, 223, 79, 111, 37, 35, 1, 0
DF 4F 6F 25 23 01 00		
69FF FF 00250	DB	255, 251, 242, 246, 164, 196, 128, 0
FB F2 F6 A4 C4 80 00		
6A07 30 00260	DB	48, 120, 216, 220, 159, 31, 15, 111
7B DB DC 9F 1F 0F 6F		
6A0F 0C 00270	DB	12, 30, 27, 59, 249, 248, 240, 246
1E 1B 3B F9 FB F0 F6		
6A17 7F 00280 FR0G4	DB	127, 47, 31, 15, 3, 57, 15, 3
2F 1F 0F 03 39 0F 03		
6A1F F0 00290	DB	240, 240, 248, 254, 63, 131, 142, 28
F0 FB FE 3F 83 BE 1C		
6A27 03 00300	DB	3, 15, 57, 3, 15, 31, 47, 127
0F 39 03 0F 1F 2F 7F		
6A2F 1C 00310	DB	28, 142, 131, 63, 254, 248, 240, 240
BE 83 3F FE FB F0 F0		
00320 ;		
00330 ;		
6A37 00 00340 LBIKE	DB	0, 0, 0, 0, 0, 0, 0, 0
00 00 00 00 00 00 00		
6A3F 1F 00350	DB	31, 63, 115, 81, 169, 112, 112, 32
3F 73 51 A9 70 70 20		
6A47 FE 00360	DB	254, 252, 252, 234, 213, 206, 14, 4
FC FC EA D5 CE OE 04		
6A4F 00 00370	DB	0, 0, 0, 0, 0, 0, 0, 0
00 00 00 00 00 00 00		
6A57 00 00380	DB	0, 0, 0, 0, 0, 0, 0, 0
00 00 00 00 00 00 00		
6A5F 01 00390	DB	1, 3, 1, 0, 3, 4, 14, 31
03 01 00 03 04 OE 1F		
6A67 80 00400	DB	128, 192, 192, 224, 224, 112, 119, 255
C0 C0 E0 E0 70 77 FF		
6A6F 00 00410	DB	0, 0, 0, 0, 0, 0, 0, 0
00 00 00 00 00 00 00		
00420 ;		
6A77 00 00430 LBATT	DB	0, 7, 7, 0
07 07 00		
6A7B 00 00440	DB	0, 7, 7, 0
07 07 00		
00450 ;		
6A7F 00 00460 RBIKE	DB	0, 0, 0, 0, 0, 0, 0, 0
00 00 00 00 00 00 00		
6A87 7F 00470	DB	127, 63, 63, 87, 171, 115, 112, 32
3F 3F 57 AB 73 70 20		
6A8F FB 00480	DB	248, 252, 206, 138, 149, 14, 14, 4
FC CE BA 95 0E OE 04		
6A97 00 00490	DB	0, 0, 0, 0, 0, 0, 0, 0
00 00 00 00 00 00 00		
6A9F 00 00500	DB	0, 0, 0, 0, 0, 0, 0, 0
00 00 00 00 00 00 00		
6AA7 01 00510	DB	1, 3, 3, 7, 7, 14, 238, 255
03 03 07 07 0E EE FF		
6AAF 80 00520	DB	128, 192, 128, 0, 192, 32, 112, 248

6AB7	C0 B0 00 C0 20 70 F8		
	00 00 00 00530	DB	0,0,0,0,0,0,0,0
	00 00 00 00 00 00 00		
	00540 ;		
	00550 ;		
6ABF	00 00560 RBATT	DB	0,7,7,0
	07 07 00		
6AC3	00 00570	DB	0,7,7,0
	07 07 00		
	00580 ;		
	00590 ;		
6AC7	00 00600 LCAR	DB	0,0,0,0,0,0,0,0
	00 00 00 00 00 00 00		
6ACF	00 00610	DB	0,0,3,7,15,2,0,0
	00 03 07 0F 02 00 00		
6AD7	07 00620	DB	7,255,255,159,111,247,240,96
	FF FF 9F 6F F7 F0 60		
6ADF	80 00630	DB	128,255,255,255,255,254,0,0
	FF FF FF FF FE 00 00		
6AE7	F0 00640	DB	240,254,255,159,111,246,240,96
	FE FF 9F 6F F6 F0 60		
6AEF	00 00650	DB	0,0,0,0,0,0,0,0
	00 00 00 00 00 00 00		
6AF7	00 00660	DB	0,0,0,0,0,0,0,0
	00 00 00 00 00 00 00		
6AFF	00 00670	DB	0,0,0,0,0,0,0,0
	00 00 00 00 00 00 00		
6B07	00 00680	DB	0,0,0,0,0,0,0,0
	00 00 00 00 00 00 00		
6B0F	00 00690	DB	0,0,0,0,0,63,97,193
	00 00 00 00 3F 61 C1		
6B17	00 00700	DB	0,0,0,0,0,0,128,192
	00 00 00 00 00 B0 C0		
6B1F	00 00710	DB	0,0,0,0,0,0,0,0
	00 00 00 00 00 00 00		
	00720 ;		
6B27	00 00730 LCATT	DB	0,6,6,6,6,0
	06 06 06 06 00		
6B2D	00 00740	DB	0,0,0,6,6,0
	00 00 06 06 00		
	00750 ;		
	00760 ;		
6B33	00 00770 RCAR	DB	0,0,0,0,0,0,0,0
	00 00 00 00 00 00 00		
6B3B	0F 00780	DB	15,127,255,249,246,111,15,6
	7F FF F9 F6 6F 0F 06		
6B43	01 00790	DB	1,255,255,255,255,127,0,0
	FF FF FF FF 7F 00 00		
6B4B	E0 00800	DB	224,255,255,249,246,239,15,6
	FF FF F9 F6 EF 0F 06		
6B53	00 00810	DB	0,0,192,224,240,64,0,0
	00 C0 E0 F0 40 00 00		
6B5B	00 00820	DB	0,0,0,0,0,0,0,0
	00 00 00 00 00 00 00		
6B63	00 00830	DB	0,0,0,0,0,0,0,0
	00 00 00 00 00 00 00		

6B66	00		00240	DB	0,0,0,0,0,0,1,3
	00	00	00 00 01 03		
6B73	00		00850	DB	0,0,0,0,0,252,134,131
	00	00	00 00 FC B6 B3		
6B7E	00		00860	DB	0,0,0,0,0,0,0,0
	00	00	00 00 00 00		
6BB3	00		00870	DB	0,0,0,0,0,0,0,0
	00	00	00 00 00 00		
6BB8	00		00880	DB	0,0,0,0,0,0,0,0
	00	00	00 00 00 00		
			00890 ;		
6B93	00		00900 RCATT	DB	0,2,2,2,2,0
	02	02	02 02 00		
6B99	00		00910	DB	0,2,2,0,0,0
	02	02	00 00		
			00920 ;		
			00930 ;		
6B9F	00		00940 LTRUCK	DB	0,0,0,0,0,0,0,0
	00	00	00 00 00 00		
6BA7	1F		00950	DB	31,31,31,62,61,59,3,1
	1F	1F	3E 3D 3B 03 01		
6BAF	F8		00960	DB	248,252,254,127,184,216,192,128
	FC	FE	7F B8 DB C0 80		
6BB7	FF		00970	DB	255,255,255,255,6,15,15,6
	FF	FF	06 0F 0F 06		
6BBF	FF		00980	DB	255,255,255,0,0,0,0,0
	FF	FF	00 00 00 00		
6BC7	FF		00990	DB	255,255,255,0,0,0,0,0
	FF	FF	00 00 00 00		
6BCF	FF		01000	DB	255,255,255,0,6,15,15,6
	FF	FF	00 06 0F 0F 06		
6BD7	FE		01010	DB	254,254,254,4,50,122,122,48
	FE	FE	04 32 7A 7A 30		
6BDF	00		01020	DB	0,0,0,0,0,0,0,0
	00	00	00 00 00 00		
6BE7	00		01030	DB	0,0,0,0,0,0,0,0
	00	00	00 00 00 00		
6BEF	00		01040	DB	0,0,7,9,17,17,31,31
	00	07	09 11 11 1F 1F		
6BF7	02		01050	DB	2,2,250,250,254,252,252,248
	02	FA	FA FE FC FC FB		
6BFF	FF		01060	DB	255,255,255,255,255,255,255,255
	FF	FF	FF FF FF FF		
6C07	FF		01070	DB	255,255,255,255,255,255,255,255
	FF	FF	FF FF FF FF		
6C0F	FF		01080	DB	255,255,255,255,255,255,255,255
	FF	FF	FF FF FF FF		
6C17	FF		01090	DB	255,255,255,255,255,255,255,255
	FF	FF	FF FF FF FF		
6C1F	FE		01100	DB	254,254,254,254,254,254,254,254
	FE	FE	FE FE FE FE		
6C27	00		01110	DB	0,0,0,0,0,0,0,0
	00	00	00 00 00 00		
6C2F	00		01120	DB	0,0,0,0,0,0,0,0
	00	00	00 00 00 00		
6C37	00		01130	DB	0,0,0,0,0,0,0,0

6C3F	00 00 00 00 00 00	00 00	01140	DB	0, 0, 0, 0, 0, 0, 0
	00 00 00 00 00 00	00 00			
6C47	00 00 00 00 00 00	00 00	01150	DB	0, 0, 0, 0, 0, 255, 255, 255
	00 00 00 00 00 00	00 FF FF FF			
6C4F	00 00 00 00 00 00	00 00	01160	DB	0, 0, 0, 0, 0, 255, 255, 255
	00 00 00 00 00 00	00 FF FF FF			
6C57	00 00 00 00 00 00	00 00	01170	DB	0, 0, 0, 0, 0, 255, 255, 255
	00 00 00 00 00 00	00 FF FF FF			
6C5F	00 00 00 00 00 00	00 00	01180	DB	0, 0, 0, 0, 0, 255, 255, 255
	00 00 00 00 00 00	00 FF FF FF			
6C67	00 00 00 00 00 00	00 00	01190	DB	0, 0, 0, 0, 0, 254, 254, 254
	00 00 00 00 00 00	00 FE FE FE			
6C6F	00 00 00 00 00 00	00 00	01200	DB	0, 0, 0, 0, 0, 0, 0
	00 00 00 00 00 00	00 00			
			01210 ;		
			01220 ;		
6C77	00 03 03 05 05 05	05 05 05 05 00	01230	LTATT DB	0, 3, 3, 5, 5, 5, 5, 5, 0
	00 03 03 05 05 05	05 05 05 05 00			
6C80	00 03 03 05 05 05	05 05 05 05 00	01240	DB	0, 3, 3, 5, 5, 5, 5, 5, 0
	00 03 03 05 05 05	05 05 05 05 00			
6C89	00 00 05 05 05 05	05 05 05 05 00	01250	DB	0, 0, 0, 5, 5, 5, 5, 5, 0
	00 00 05 05 05 05	05 05 05 05 00			
			01260 ;		
			01270 ;		
6C92	00 00 00 00 00 00	00 00 00 00	01280	RTRUCK DB	0, 0, 0, 0, 0, 0, 0, 0
	00 00 00 00 00 00	00 00 00 00			
6C9A	7F 7F 7F 20 4C 5E 5E 0C		01290	DB	127, 127, 127, 32, 76, 94, 94, 12
	7F 7F 7F 20 4C 5E 5E 0C				
6CA2	FF FF 00 60 F0 F0 60		01300	DB	255, 255, 255, 0, 96, 240, 240, 96
	FF FF 00 60 F0 F0 60				
6CAA	FF FF 00 00 00 00		01310	DB	255, 255, 255, 0, 0, 0, 0, 0
	FF FF 00 00 00 00				
6CB2	FF FF 00 00 00 00		01320	DB	255, 255, 255, 0, 0, 0, 0, 0
	FF FF 00 00 00 00				
6CBA	FF FF FF 60 F0 F0 60		01330	DB	255, 255, 255, 255, 96, 240, 240, 96
	FF FF FF 60 F0 F0 60				
6CC2	1F 3F 7F FE 1D 1B 03 01		01340	DB	31, 63, 127, 254, 29, 27, 3, 1
	3F 7F FE 1D 1B 03 01				
6CCA	F8 F8 7C BC DC C0 80		01350	DB	248, 248, 248, 124, 188, 220, 192, 128
	F8 F8 7C BC DC C0 80				
6CD2	00 00 00 00 00 00		01360	DB	0, 0, 0, 0, 0, 0, 0, 0
	00 00 00 00 00 00				
6CDA	00 00 00 00 00 00		01370	DB	0, 0, 0, 0, 0, 0, 0, 0
	00 00 00 00 00 00				
6CE2	7F 7F 7F 7F 7F 7F		01380	DB	127, 127, 127, 127, 127, 127, 127, 127
	7F 7F 7F 7F 7F 7F				
6CEA	FF FF FF FF FF FF		01390	DB	255, 255, 255, 255, 255, 255, 255, 255
	FF FF FF FF FF FF				
6CF2	FF FF FF FF FF FF		01400	DB	255, 255, 255, 255, 255, 255, 255, 255
	FF FF FF FF FF FF				
6CFA	FF FF FF FF FF FF		01410	DB	255, 255, 255, 255, 255, 255, 255, 255
	FF FF FF FF FF FF				
6D02	FF FF FF FF FF FF		01420	DB	255, 255, 255, 255, 255, 255, 255, 255
	FF FF FF FF FF FF				
6D0A	40 01430			DB	64, 64, 95, 95, 127, 63, 63, 31

6D12	40 5F 5F 7F 3F 3F 1F 00 01440	DB	0,0,224,144,136,136,248,248
6D1A	00 E0 90 88 88 FB FB 00 01450	DB	0,0,0,0,0,0,0,0
6D22	00 00 00 00 00 00 00 01460	DB	0,0,0,0,0,0,0,0
6D2A	00 00 00 00 7F 7F 7F 00 01470	DB	0,0,0,0,0,127,127,127
6D32	00 00 00 00 FF FF FF 00 01480	DB	0,0,0,0,0,255,255,255
6D3A	00 00 00 00 FF FF FF 00 01490	DB	0,0,0,0,0,255,255,255
6D42	00 00 00 00 FF FF FF 00 01500	DB	0,0,0,0,0,255,255,255
6D4A	00 00 00 00 FF FF FF 00 01510	DB	0,0,0,0,0,255,255,255
6D52	00 00 00 00 00 00 00 01520	DB	0,0,0,0,0,0,0,0
6D5A	00 00 00 00 00 00 00 01530	DB	0,0,0,0,0,0,0,0
6D62	00 00 00 00 00 00 00 01540	DB	0,0,0,0,0,0,0,0
6D6A	00 05 05 05 05 05 03 03 00 00 01550 ; 01560 ; 01570 RTATT	DB	0,5,5,5,5,5,3,3,0
6D73	00 05 05 05 05 05 03 03 00 00 01580	DB	0,5,5,5,5,5,3,3,0
6D7C	00 05 05 05 05 05 00 00 00 00 01590	DB	0,5,5,5,5,5,0,0,0
6D85	00 00 00 00 01600 ; 01610 ; 01620 BLANK	DB	0,0,0,0
0024	01630 ; 01640 ; 01650 FRGSTR	DS	36 ;4*8+4
007B	01660 PCSTR 01670 ; 01680 ; 01690 ;***** DATA BASE ***** 01700 ;	DS	120 ;12*8+12+7
6E25	00 01710	OB1EXT	DEFB 0 ;ESISTENZA OGGETTO
6E26	00 01720	DEFB 0 ;CONTATORE	
6E27	00 01730	DEFB 0 ;DIR. 8 => DESTRA	
6E28	00 01740	DEFB 0 ;IND. POS. OGGETTO1	
6E29	0000 01750	DEFW 0 ;CONT. POS.	
6E2B	0000 01760	DEFW 0 ;PUNT. BAGOMA	
6E2D	0000 01770	DEFW 0 ;PUNT. ATTRIBUTI	
6E2F	00 01780	DEFB 0 ;CONT. RIGHE	
6E30	00 01790	DEFB 0 ;CONT. COLONNE	
6E31	00 00 00 00 01800	OB2EXT	DB 0,0,0,0
6E35	0000 01810	DEFW 0 ;INDIC.REALE/ABSTR. OGG2	
6E37	0000 01820	DEFW 0	

6E39	0000	01830	DEFW	0	
6E3B	00	01840	DB	0,0	
	00				
6E3D	00	01850	OBSEXT	DB	0,0,0,0
	00 00 00				
6E41	0000	01860	DEFW	0	
6E43	0000	01870	DEFW	0	
6E45	0000	01880	DEFW	0	
6E47	00	01890	DB	0,0	
	00				
6E49	00	01900	OB4EXT	DB	0,0,0,0
	00 00 00				
6E4D	0000	01910	DEFW	0	
6E4F	0000	01920	DEFW	0	
6E51	0000	01930	DEFW	0	
6E53	00	01940	DB	0,0	
	00				
6E55	00	01950	OBSEXT	DB	0,0,0,0
	00 00 00				
6E59	0000	01960	DEFW	0	
6E5B	0000	01970	DEFW	0	
6E5D	0000	01980	DEFW	0	
6E5F	00	01990	DB	0,0	
	00				
6E61	00	02000	OB6EXT	DB	0,0,0,0
	00 00 00				
6E65	0000	02010	DEFW	0	
6E67	0000	02020	DEFW	0	
6E69	0000	02030	DEFW	0	
6E6B	00	02040	DB	0,0	
	00				
		02050	;		
		02060	;		
6E6D	00	02070	PCAREXT	DEFB	0
6E6E	00	02080	PCARCYC	DEFB	0
6E6F	00	02090	PCARDIR	DEFB	0
6E70	00	02100	PCARRAP	DEFB	0
6E71	0000	02110	PCARPOS	DEFW	0
6E73	0000	02120	PCARSHP	DEFW	0
6E75	0000	02130	PCARATT	DEFW	0
6E77	02	02140	PCARROW	DEFB	2
6E78	06	02150	PCARCOL	DEFB	6
		02160	;		
		02170	;		
6E79	00	02180	FRGEXT	DEFB	0
6E7A	00	02190	FRGCYC	DEFB	0
6E7B	00	02200	FRGDIR	DEFB	0
6E7C	0000	02210	FRGPOS	DEFW	0
6E7E	0000	02220	FROGSH	DEFW	0
6E80	00	02230	FRGATR	DEFB	0
		02240	;		
		02250	;		
6E81	08	02260	FRGDB	DB	8,8,1
	08 01				
6E84	AC30	02270	FRGSTN	DEFW	50ACH
6E86	B769	02280	DEFW	FROG1	

; DATABASE AUTO POLIZIA

; DATABASE RAMA

; I0-SU 1-DES 2-0IU 3-9IN

; POS. INIZ. RAMA

6E88 04	02290	DB	4		;ATTRIBUTI 8 CARAT.
	02300 ;				
	02310 ;				
6E89 956E	02320	DBINDEX	DEFW	RBDB	;DB BICI DESTRA
6E8B A16E	02330	DEFW		LBDB	;DB BICI SIN.
6E8D AD6E	02340	DEFW		RCDB	;DB AUTO DESTRA
6EBF B96E	02350	DEFW		LCDB	;DB AUTO SIN.
6E91 C56E	02360	DEFW		RTDB	;DB CAMION DESTRA
6E93 D16E	02370	DEFW		LTDB	;DB CAMION SIN.
	02380 ;				
	02390 ;				
6E95 02	02400	RBDB	DB	2,1,0,0	;ES. CONT. DIR. IND. R/A
01 00 00					
6E99 1D4B	02410	DEFW		4B1DH	;POS
6E9B 7F6A	02420	DEFW		RBIKE	;BICI DESTRA
6E9D BF6A	02430	DEFW		RBATT	;ATTRIBUTO
6E9F 02	02440	DB		2,4	;RIGA COLONNA
04					
	02450 ;				
	02460 ;				
6EA1 02	02470	LBDB	DB	2,1,1,1	
01 01 01					
6EA5 DF4B	02480	DEFW		4BDFH	
6EA7 376A	02490	DEFW		LBIKE	
6EA9 776A	02500	DEFW		LBATT	
6EAB 02	02510	DB		2,4	
04					
	02520 ;				
	02530 ;				
6EAD 03	02540	RCDB	DB	3,1,0,0	
01 00 00					
6EB1 1B4B	02550	DEFW		4B1BH	
6EB3 336B	02560	DEFW		RCAR	
6EB5 936B	02570	DEFW		RCATT	
6EB7 02	02580	DB		2,6	
06					
	02590 ;				
	02600 ;				
6EB9 03	02610	LCDB	DB	3,1,1,1	
01 01 01					
6EBD DF4B	02620	DEFW		4BDFH	
6EBF C76A	02630	DEFW		LCAR	
6EC1 276B	02640	DEFW		LCATT	
6EC3 02	02650	DB		2,6	
06					
	02660 ;				
	02670 ;				
6EC5 06	02680	RTDB	DB	6,1,0,0	
01 00 00					
6EC9 1B4B	02690	DEFW		4B1BH	
6ECB 926C	02700	DEFW		RTRUCK	
6ECD 6A6D	02710	DEFW		RTATT	
6ECF 03	02720	DB		3,9	
09					
	02730 ;				
	02740 ;				

6ED1	06		02750	LTDB	DB	6,1,1,1	
	01	01	01				
6ED5	DF4B		02760		DEFW	4BDFH	
6ED7	9F6B		02770		DEFW	LTRUCK	
6ED9	776C		02780		DEFW	LTATT	
6EDB	03		02790		DB	3,9	
	09						
			02800				
			02810				
6EDD	01		02820	LPCDB	DB	1,1,1,1	
	01	01	01				
6EE1	DF4B		02830		DEFW	4BDFH	
6EE3	C76A		02840		DEFW	LCAR	
6EE5	E96E		02850		DEFW	LPCATT	
6EE7	02		02860		DB	2,6	
	06						
			02870				
			02880				
6EE9	00		02890	LPCATT	DB	0,5,5,5,5,0	
	05	05	05	00			
6EEF	00		02900		DB	0,0,0,5,5,0	
	00	00	05	00			
			02910				
			02920				
6EF5	01		02930	RPCDB	DB	1,1,0,0	
	01	00	00				
6EF9	1B4B		02940		DEFW	4B1BH	
6EFB	336B		02950		DEFW	RCAR	
6EFD	016F		02960		DEFW	RPCATT	
6EFF	02		02970		DB	2,6	
	06						
			02980				
			02990				
6F01	00		03000	RPCATT	DB	0,5,5,5,5,0	
	05	05	05	00			
6F07	00		03010		DB	0,5,5,0,0,0	
	05	05	00	00			
			03020				
			03030				
			00480				
			00490				
6F0D	29		00500	PCTON1	DB	41,0,0F0H,1	PRIMO TONO SIRENA
	00	F0	01				
6F11	17		00510	PCTON2	DB	23,0,8CH,3	SECONDO TONO
	00	BC	03				
			00520				
			00530				
6F15	46		00540	HOMTON	DB	46H,0,0C7H,4	TONO ALLEGRO
	00	C7	04				
6F19	5D		00550		DB	5DH,0,8CH,3	
	00	BC	03				
6F1D	7C		00560		DB	7CH,0,0A1H,2	
	00	A1	02				
6F21	AA		00570		DB	0AAH,0,0F1H,1	
	00	F1	01				
6F25	DE		00580		DB	0DEH,0,6DH,1	

	00 6D 01				
6F29	28	00590	DB	28H, 1, 9, 1	
	01 09 01				
6F2D	8B	00600	DB	8BH, 1, 0BFH, 0	
	01 BF 00				
6F31	0F	00610	DB	0FH, 2, 8BH, 0	
	02 8B 00				
6F35	C0	00620	DB	0C0H, 2, 5EH, 0	
	02 5E 00				
6F39	84	00630	DIETON DB	84H, 3, 43H, 0	; TONO TRISTE
	03 43 00				
		00640 ;			
		00650 ;			
6F3D	53	00660	SCRMS1 DM	'Score '	
	63 6F 72	65 20			
6F43	30	00670	SCORE DB	30H, 30H, 30H, 30H, 30H, 30H	
	30 30 30	30 30			
6F49	4B	00680	SCRMS2 DM	'HIGH SCORE '	
	49 47 4B	20 53 43 4F 52			
	45 20				
6F54	30	00690	HISCR DB	30H, 30H, 30H, 30H, 30H	
	30 30 30				
		00700 ;			
		00710 ;			
0005		00720	IMAGE DS	5	; VIS. PUNTEGGIO
6F5E	00	00730	UPDWN DEFB	0	; RANA IN SU O IN GIU?
		00740 ;			
		00750 ;			
6F5F	00	00760	COLUMN DB	0	; VAR. COLONNA
6F60	00	00770	ROW DB	0	; VAR. RIGA
6F61	00	00780	SKIP DEFB	0	; CAP. SALTATO
6F62	00	00790	FILL DEFB	0	; CAR. DISEGNATO
6F63	0000	00800	ATTPOS DEFW	0	; MEM. PUNT. FILE ATTP.
6F65	00	00810	ATTR DB	0	; ATTRIBUTI DISEGNO
6F66	0000	00820	DRWPOS DEFW	0	; POS. DISEGNO
6F68	0000	00830	STRPOS DEFW	0	; POS. MEM.
		00840 ;			
		00850 ;			
6F6A	0000	00860	ATTPTR DEFW	0	
6F6C	0000	00870	NEWPOS DEFW	0	; POS. NUOVO OGG.
6F6E	0000	00880	POSPTR DEFW	0	; PUNT. D. B.
6F70	00	00890	GENFLG DEFB	0	; IND. GENER. TRAFF.
		00900 ;			
		00910 ;			
6F71	00	00920	JAMFLG DEFB	0	; POSTO A 1 SE VI E' MOV.
		00930 ;			
		00940 ;			
6F72	00	00950	CHASE DEFB	0	; POSTO A 1 SE APPARE POLIZIA
6F73	00	00960	SOUNDF DEFB	0	; POSTO A 1 SE SUONA SIRENA
6F74	00	00970	TONFLG DEFB	0	; DETERM. TONO SIRENA
6F75	0000	00980	RND DEFW	0	; GEN. NUM. CASUALE
		00990 ;			
		01000 ;			
6F77	01	01010	GAMFLG DEFB	1	; FINE SE ZERO
6F7B	0000	01020	DLDFRG DEFW	0	; VECCHIA POS. RANA
6F7A	0000	01030	NEWFRG DEFW	0	; NUOVA POS. RANA

6F7C 00	01040	CRHFLG	DEFB	0	;	POSTO A 1 SE INCIDENTE
6F7D 00	01050	TEMDIR	DEFB	0	;	INNOVA DIR. RANA
6F7E 0000	01060	TEMPOS	DEFW	0	;	INNOVA POS. RANA
6F80 0000	01070	TEMSPH	DEFW	0	;	INNOVA SAGOMA RANA
	01080	;				
	01090	;				
5020	01100	BOTHY1	EQU	5020H	;	0,38. 0,39
5120	01110	BOTHY2	EQU	5120H		
46A0	01120	TOPHY1	EQU	46A0H	;	0,128. 0,129
47A0	01130	TOPHY2	EQU	47A0H		
4B60	01140	MIDHY1	EQU	4B60H	;	x,83. x,84
4C60	01150	MIDHY2	EQU	4C60H		
	01160	;				
	01170	;				
3C00	01180	CHRSET	EQU	3C00H	;	PRIMO DEI 256 BYTES VUOTI
	01190	;				
	01200	;				
6F82 05	01210	NUMFRG	DEFB	5	;	NUMERO RANE
	01220	;				
	01230	;				
6F83 AF	01240	INIT	XOR	A	;	000 PER D2 D1 D0
6F84 D3FE	01250		OUT	(0FEH),A	;	COLORE SFONDO
6F86 32485C	01260		LD	(23624),A	;	MERO
6F89 327C6F	01270		LD	(CRHFLG),A		
6F8C 32796E	01280		LD	(FRGEXT),A	;	RANA NON ES.
6F8F 3C	01290		INC	A		
6F90 32776F	01300		LD	(GAMFLG),A	;	IND. GIOCO
6F93 3E05	01310		LD	A,5	;	INIZ. NUM.RANE
6F95 32826F	01320		LD	(NUMFRG),A		
6F98 ED5F	01330		LD	A,R	;	GEN. NUM. CASUALE
6F9A E63F	01340		AND	3FH	;	PER QUESTO CICLO
6F9C 67	01350		LD	H,A	;	PUNT. ALLA ROM
6F9D ED5F	01360		LD	A,R		
6F9F 6F	01370		LD	L,A		
6FA0 22756F	01380		LD	(RND),HL		
6FA3 21AC50	01390		LD	HL,50ACH	;	POS.INIZ. RANA
6FA6 22846E	01400		LD	(FRGSTN),HL		
6FA9 CDD772	01410		CALL	CLRSCR	;	CANCELL. VIDEO
6FAC CD0B70	01420		CALL	DRWHWY	;	DISEGNO
6FAF CD5570	01430		CALL	LINEUP	;	LIMITE PER ES. RANA
6FB2 210040	01440		LD	HL,4000H	;	MESSAGGIO PUNT.
6FB5 113D6F	01450		LD	DE,SCRMS1		
6FB8 0606	01460		LD	B,6		
6FBA CD2B73	01470		CALL	DISASC	;	VIS.CARATTERE A9CII
6FBD 21446F	01480		LD	HL,SCORE+1	;	VIS. PUNTEGGIO
6FC0 CD6F77	01490		CALL	SCRIMG	;	CONVERTITO IN IMMAGINE
MB						
6FC3 210640	01500		LD	HL,4006H		
6FC6 11596F	01510		LD	DE,IMAGE		
6FC9 0605	01520		LD	B,5		
6FCB CD2B73	01530		CALL	DISASC		
6FCE 210E40	01540		LD	HL,400EH	;	MESSAGGIO PUNT.MAX.
6FD1 11496F	01550		LD	DE,SCRMS2		
6FD4 060B	01560		LD	B,11		
6FD6 CD2B73	01570		CALL	DISASC		
6FD9 21546F	01580		LD	HL,HISCR		
6FDC CD6F77	01590		CALL	SCRIMG		

6FDF 211940	01600		LD	HL, 4019H	
6FE2 11596F	01610		LD	DE, IMAGE	
6FE5 0605	01620		LD	B, 5	
6FE7 CD2873	01630		CALL	DISASC	
6FEA 21256E	01640		LD	HL, DB1EXT	
6FED 110C00	01650		LD	DE, 12	
6FF0 0607	01660		LD	B, 7	
6FF2 AF	01670		XOR	A	
6FF3 77	01680	INTLP1	LD	(HL), A	
6FF4 19	01690		ADD	HL, DE	
6FF5 10FC	01700		DJNZ	INTLP1	
6FF7 32726F	01710		LD	(CHASE), A	
6FFA 3C	01720		INC	A	
6FFB 32736F	01730		LD	(SOUNDF), A	
6FFE 21436F	01740		LD	HL, SCORE	
7001 11446F	01750		LD	DE, SCORE+1	
7004 0E05	01760		LD	C, 5	
7006 3630	01770		LD	(HL), 30H	
7008 EDB0	01780		LDIR		
700A C9	01790		RET		
	01800		;		
	01810		;		
700B 21A040	01820	DRHWY	LD	HL, 40A0H	
700E CD4170	01830		CALL	FILHWY	
7011 216048	01840		LD	HL, 4860H	
7014 CD4170	01850		CALL	FILHWY	
7017 212050	01860		LD	HL, 5020H	
701A CD4170	01870		CALL	FILHWY	
701D 21A046	01880		LD	HL, TOPHY1	
7020 11A047	01890		LD	DE, TOPHY2	
7023 AF	01900		XOR	A	
7024 CD3870	01910		CALL	HIGHWY	
7027 212050	01920		LD	HL, BOTHY1	
702A 112051	01930		LD	DE, BOTHY2	
702D CD3870	01940		CALL	HIGHWY	
7030 21604B	01950		LD	HL, MIDHY1	
7033 11604C	01960		LD	DE, MIDHY2	
7036 3EC3	01970		LD	A, 195	
7038 0620	01980	HIGHWY	LD	B, 32	
703A 77	01990	HWYLOP	LD	(HL), A	
703B 12	02000		LD	(DE), A	
703C 23	02010		INC	HL	
703D 13	02020		INC	DE	
703E 10FA	02030		DJNZ	HWYLOP	
7040 C9	02040		RET		
	02050		;		
	02060		;		
7041 3EFF	02070	FILHWY	LD	A, OFFH	
7043 D9	02080		EXX		
7044 0620	02090		LD	B, 32	
7046 D9	02100	FILHYL	EXX		
7047 E5	02110		PUSH	HL	
7048 0608	02120		LD	B, B	
704A 77	02130	FILCHR	LD	(HL), A	
704B 24	02140		INC	H	
704C 10FC	02150		DJNZ	FILCHR	

; NESSUN OGG. ESISTENTE

; POLIZIA NON ESIST.

; SIRENA ACCESA
; INIZ. PUNTEGGIO COM
; LO ZERO ASCII (30H)

; INIZ. PUNTI A 30H

; LINEA ALTA

; LINEA MEDIA

; LINEA BASSA

; INV. COLORE

; BIN 11000011
; 32#8 BITS

```

704E E1      02160      POP      HL
704F 23      02170      INC      HL
7050 D9      02180      EXX
7051 10F3    02190      DJNZ    FILHYL
7053 D9      02200      EXX
7054 C9      02210      RET

02220 ;
02230 ;
02240 ;***** LINEUP *****
02250 ;
02260 ;
02270 ;
02280 ;
02290 ;
02300 ;
02310 ;
02320 ;
02330 ;
02340 ;
02350 ;
02360 ;
02370 ;
02380 ;
02390 ;
02400 ;
02410 ;
02420 ;
02430 ;
02440 ;
02450 ;
02460 ;
02470 ;
02480 ;
02490 ;
02500 ;
02510 ;
02520 ;
02530 ;***** DRWFRG *****
02540 ;
02550 ;
02560 ;
02570 ;
02580 ;
02590 ;
02600 ;
02610 ;
02620 ;
02630 ;
02640 ;
02650 ;
02660 ;
02670 ;
02680 ;
02690 ;
02700 ;
02710 ;

```

DISEGNA TUTTE LE RANE IN BASSO A SINISTRA DELLO SCHERMO

```

7055 3E01    02290      LINEUP  LD      A,1          ; RANA A DESTRA
7057 327B6E 02300      LD      (FRGDIR),A
705A 11D769 02310      LD      DE,FR0G2        ; SACOMA RANA
705D 2A846E 02320      LD      HL,(FRGSTN)    ; POS. RANA
7060 3E04     02330      LD      A,4            ; (PAPER 0)*8+(INK*4)
7062 32656F 02340      LD      (ATTR),A
7065 3A826F 02350      LD      A,(NUMFRG)    ; NUMERO RANE
7068 A7      02360      AND
7069 C8      02370      RET      Z
706A 47      02380      LD      B,A          ; CONT.TEMPO
706B C5      02390      DRAWLN  PUSH   BC
706C D5      02400      PUSH   DE
706D E5      02410      PUSH   HL
706E CD7A70 02420      CALL   DRWFRG        ; DISEGNO DELLA RANA
7071 E1      02430      POP     HL
7072 D1      02440      POP     DE
7073 2B      02450      DEC     HL
7074 2B      02460      DEC     HL
7075 2B      02470      DEC     HL
7076 C1      02480      POP     BC
7077 10F2    02490      DJNZ    DRAWLN
7079 C9      02500      RET

02510 ;
02520 ;
02530 ;***** DRWFRG *****
02540 ;
02550 ;
02560 ;
02570 ;
02580 ;
02590 ;
02600 ;
02610 ;
02620 ;
02630 ;
02640 ;
02650 ;
02660 ;
02670 ;
02680 ;
02690 ;
02700 ;
02710 ;

```

SIMILE ALLA ROUTINE DRAW

```

707A 3E02    02570      DRWFRG  LD      A,2          ; SACOMA SU DUE RIGHE
707C 08      02580      EX
707D E5      02590      PUSH   HL          ; SALVA PUNT.POS.
707E E5      02600      FRGLP0  PUSH   HL
707F 0E02    02610      LD      C,2        ; CONT.COLOMNE
7081 E5      02620      FRGLP1  PUSH   HL
7082 0608    02630      LD      B,B        ; CAR.DISEGNATO
7084 1A      02640      FRGLP2  LD      A,(DE)
7085 77      02650      LD      (HL),A
7086 13      02660      INC     DE
7087 24      02670      INC     H          ; BYTE SUCCESSIVO
708B 10FA    02680      DJNZ    FRGLP2
708A E1      02690      POP     HL          ; PUNT.CORRENTE
708B 23      02700      INC     HL          ; INCR.PUNT.
708C 0D      02710      DEC     C          ; DECR. CONT. COLOMNE

```

708D 20F2	02720	JR	NZ,FRGLP1	
708F E1	02730	POP	HL	: PUNT. RIGHE
7090 08	02740	EX	AF,AF'	
7091 3D	02750	DEC	A	: DECR. LINEA DI CAR.
7092 0E20	02760	LD	C,32	
7094 280E	02770	JR	Z,FRGATT	: ATTRIBUTO RANA
7096 08	02780	EX	AF,AF'	
7097 A7	02790	AND	A	
7098 ED42	02800	SBC	HL,BC	: SPOST. 32 CAR.
709A CB44	02810	BIT	0,H	: CONTROLLO VIDEO
709C 28E0	02820	JR	Z,FRGLP0	
709E 7C	02830	LD	A,H	
709F D607	02840	SUB	7	: SEL.SEZ.VIDEO
70A1 67	02850	LD	H,A	
70A2 18DA	02860	JR	FRGLP0	
70A4 E1	02870	FRGATT POP	HL	: PUNT.POS
70A5 7C	02880	LD	A,H	: CONV.PUNT.ATTR.
70A6 E618	02890	AND	18H	
70AB CB2F	02900	SRA	A	
70AA CB2F	02910	SRA	A	
70AC CB2F	02920	SRA	A	
70AE C658	02930	ADD	A,58H	
70B0 67	02940	LD	H,A	
70B1 3A656F	02950	LD	A,(ATTR)	: ATTR. SAGOMA RANA
70B4 77	02960	LD	(HL),A	
70B5 23	02970	INC	HL	: CAR. SUCCESSIVO
70B6 77	02980	LD	(HL),A	
70B7 ED42	02990	SBC	HL,BC	: UNA LINEA IN SU
70B9 77	03000	LD	(HL),A	
70BA 2B	03010	DEC	HL	: UN CAR. A SINISTRA
70BB 77	03020	LD	(HL),A	
70BC C9	03030	RET		
	03040	;		
	03050	;	***** TFCTRL *****	
	03060	;		
	03070	;	ROUTINE DI CONTROLLO DEL TRAFFICO	
	03080	;		
70BD 21706F	03090	TFCTRL LD	HL,GENFLG	: CONTROLLO INDIC. GEN.
70C0 AF	03100	XDR	A	
70C1 BE	03110	CP	(HL)	
70C2 2802	03120	JR	Z,GENER	: SE 8 GENERA
70C4 35	03130	DEC	(HL)	: DECR. INDIC. GEN.
70C5 C9	03140	RET		
70C6 21256E	03150	GENER LD	HL,OB1EXT	: INIZIO D.B. TRAFFICO
70C9 110C00	03160	LD	DE,12	: 12 BYTE D.B.
70CC 0606	03170	LD	B,6	: 6 COPPIE D.B.
70CE BE	03180	TCTRLP CP	(HL)	: TEST ESISTENZA
70CF 2004	03190	JR	NZ,NSPACE	
70D1 CDD970	03200	CALL	REGEN	: ROUTINE DI GEN.
70D4 C9	03210	RET		
70D5 19	03220	NSPACE ADD	HL,DE	
70D6 10F6	03230	DJNZ	TCTRLP	
70DB C9	03240	RET		
	03250	;		
	03260	;		
	03270	;	***** REGEN *****	

```

03280 ;
03290 ;
03300 ;
03310 ;
RIGENERAZIONE DEL TRAFFICO
INPUT: HL=> D.B. COPPIE

7009 E5 03320 REGEN PUSH HL
700A CDDC77 03330 RAND1 CALL RANDNO ;ROUTINE PER GENERARE
700D E607 03340 AND 7 ; UN NUM. CASUALE
700F FE06 03350 CP 6 ; TRA 0 E 5
70E1 30F7 03360 JR NC,RAND1
70E3 012159 03370 LD BC,5921H ; TEST SU DUE CAR.
70E6 212059 03380 LD HL,5920H ; TEST JAM
70E9 CB47 03390 BIT 0,A ; SE DISPARI A BIN
70EB 2804 03400 JR Z,RTRAF ; SE PARI A DES.
70ED 2EDF 03410 LD L,0DFH
70EF 0EDE 03420 LD C,0DEH
70F1 37 03430 RTRAF ADD A,A ; PONI IN DE PUNT. AL D.B.
70F2 5F 03440 LD E,A
70F3 0A 03450 LD A,(BC) ; TEST SU DUE CAR.
70F4 86 03460 ADD A,(HL)
70F5 A7 03470 AND A ; PAPER 0 E INK 0
70F6 2802 03480 JR Z,LOADDB ; SE 0 INIZ. UN NUOVO OGG.
70F8 E1 03490 POP HL ; ALTRIMENTI RETURN
70F9 C9 03500 RET
70FA 57 03510 LOADDB LD D,A ; A=0
70FB 21896E 03520 LD HL,DBINDEX ; CARICA D.B.
70FE 19 03530 ADD HL,DE
70FF 5E 03540 LD E,(HL) ; GENERA IL D.B. DI LAV.
7100 23 03550 INC HL
7101 56 03560 LD D,(HL)
7102 EB 03570 EX -DE,HL ; SORGENTE
7103 D1 03580 POP DE ; DESTINAZIONE
7104 010C00 03590 LD BC,12
7107 EDB0 03600 LDIR
7109 3E02 03610 LD A,2 ; SETTA INDIC. GEN.
710B 32706F 03620 LD (GENFLG),A ; SALTA PER DUE CICLI
710E C9 03630 RET
03640 ;
03650 ;
03660 ;***** MOVTRF *****
03670 ;
03680 ; ROUTINE PER IL TRAFFICO
03690 ;
710F D9 03700 MOVTRF EXX
7110 21256E 03710 LD HL,OB1EXT
7113 110C00 03720 LD DE,12
7116 0606 03730 LD B,6
7118 E5 03740 MTRFLP PUSH HL
7119 D9 03750 EXX
711A E1 03760 POP HL ; ESISTENZA
711B 7E 03770 LD A,(HL) ; SALTA SE NON ESISTE
711C A7 03780 AND A
711D CAA771 03790 JP Z,NXTMOV
7120 23 03800 INC HL ; CONT. DI CICLI
7121 35 03810 DEC (HL)
7122 C2A771 03820 JP NZ,NXTMOV
7125 23 03830 INC HL ; DIREZIONE

```

7126	7E	03B40		LD	A, (HL)	
7127	23	03B50		INC	HL	;:0 DA 9 A D, 1 DA D A 8
7128	23	03B60		INC	HL	
7129	226E6F	03B70		LD	(POSPTR), HL	;:PUNT.POS.
712C	5E	03B80		LD	E, (HL)	;:RIPRISTINA POS.
712D	23	03B90		INC	HL	
712E	56	03900		LD	D, (HL)	
712F	1C	03910		INC	E	;:MOV. A DESTRA
7130	A7	03920		AND	A	
7131	2B02	03930		JR	Z, LDPOS	
7133	1D	03940		DEC	E	;:MOV. A SIN.
7134	1D	03950		DEC	E	;:MOV. A SIN
7135	ED536C6F	03960	LDPOS	LD	(NEWPOS), DE	
7139	08	03970		EX	AF, AF'	
713A	010500	03980		LD	BC, 5	;:RIP. LUM. 000.
713D	09	03990		ADD	HL, BC	
713E	7E	04000		LD	A, (HL)	;:RIGA
713F	32606F	04010		LD	(ROW), A	
7142	23	04020		INC	HL	
7143	7E	04030		LD	A, (HL)	;:COLONNA
7144	325F6F	04040		LD	(COLUMN), A	
7147	3D	04050		DEC	A	
7148	4F	04060		LD	C, A	
7149	08	04070		EX	AF, AF'	
714A	A7	04080		AND	A	;:CONTROLLO DIREZ.
714B	EB	04090		EX	DE, HL	
714C	2008	04100		JR	NZ, RTOL	;:SIN. O DESTRA
714E	09	04110		ADD	HL, BC	;:RICERCA INIZIO VEIC.
714F	7D	04120		LD	A, L	;:ILOS
7150	FE40	04130		CP	40H	;:TEST SU BORDO DES.
7152	3046	04140		JR	NC, MOVEOK	;:SALTA PROX.TEST SE OFF
7154	1805	04150		JR	TESTAH	;:TEST SU POS. DAVANTI
7156	7D	04160	RTOL	LD	A, L	;:LA NUOVA POS. E' ACC.
7157	FEC0	04170		CP	0C0H	;:TEST SU BORDO SIN.
7159	3B3F	04180		JR	C, MOVEOK	;:SALTA TEST SU POS. DAVANTI
715B	7C	04190	TESTAH	LD	A, H	;:ATTRIB. COPERTO
715C	E618	04200		AND	18H	
715E	CB2F	04210		SRA	A	
7160	CB2F	04220		SRA	A	
7162	CB2F	04230		SRA	A	
7164	C658	04240		ADD	A, 5BH	
7166	67	04250		LD	H, A	
7167	012000	04260		LD	BC, 32	
716A	AF	04270		XOR	A	
716B	32716F	04280		LD	(JAMFLG), A	;:INIZ. IND. OSTACOLO
716E	3A606F	04290		LD	A, (ROW)	
7171	08	04300	TAHLOP	EX	AF, AF'	
7172	7E	04310		LD	A, (HL)	;:RIPRIST.ATTRIB.
7173	E607	04320		AND	Z	
7175	2B0E	04330		JR	Z, TFROG1	;:SALTA SE INK E' NERO
7177	FE04	04340		CP	4	;:TEST SUL COLORE VERDE, RANA
7179	2007	04350		JR	NZ, JAM1	;:IFERMA SE NON E' UNA RANA
717B	3E01	04360		LD	A, 1	;:MUOVI SE E' UNA RANA
717D	327C6F	04370		LD	(CRHFLG), A	;:BETTA INDIC. DI CRASH
7180	1803	04380		JR	TFROG1	
7182	32716F	04390	JAM1	LD	(JAMFLG), A	;:PONI JAMFLG DIVERSO DA 0

```

7185 A7      04400 TFR0G1 AND      A
7186 ED42    04410          SBC      HL, BC
7188 08      04420          EX       AF, AF'
7189 3D      04430          DEC      A
718A 20E5    04440          JR       NZ, TAHLOP
718C 3A716F  04450          LD       A, (JAMFLG)
718F A7      04460          AND      A
7190 2808    04470          JR       Z, MOVEOK
7192 D9      04480          EXX     HL
7193 23      04490          INC      HL
7194 34      04500          INC      (HL)
7195 34      04510          INC      (HL)
7196 2B      04520          DEC      HL
7197 D9      04530          EXX     HL
7198 1B0D    04540          JR       NXTMOV
719A 2A6E6F  04550 MOVEOK LD       HL, (POSPTR)
719D ED5B6C6F 04560          LD       DE, (NEWPOS)
71A1 73      04570          LD       (HL), E
71A2 23      04580          INC      HL
71A3 72      04590          LD       (HL), D
71A4 CDAF71  04600 CALL    MVCTRL
71A7 D9      04610 NXTMOV EXX     HL, DE
71A8 19      04620          ADD      B
71A9 05      04630          DEC      B
71AA C21871  04640          JP       NZ, MTRFLP
71AD D9      04650          EXX     HL
71AE C9      04660          RET

04670 ;
04680 ;***** MVCTRL *****
04690 ;
04700 ;
04710 ;
ROUTINE DI CONTROLLO SUL TRAFFICO IN MOTO
71AF 2B      04720 MVCTRL DEC      HL
71B0 2B      04730          DEC      HL
71B1 7B      04740          LD       A, E
71B2 E61F    04750          AND      1FH
71B4 2005    04760          JR       NZ, CHGRAF
71B6 7E      04770          LD       A, (HL)
71B7 3C      04780          INC      A
71B8 E601    04790          AND      1
71BA 77      04800          LD       (HL), A
71BB 2B      04810 CHGRAF DEC      HL
71BC 7E      04820          LD       A, (HL)
71BD A7      04830          AND      A
71BE 200F    04840          JR       NZ, TOLEFT
71C0 7B      04850          LD       A, E
71C1 E61F    04860          AND      1FH
71C3 201B    04870          JR       NZ, DRWOBJ
71C5 23      04880          INC      HL
71C6 7E      04890          LD       A, (HL)
71C7 2B      04900          DEC      HL
71CB A7      04910          AND      A
71C9 2015    04920          JR       NZ, DRWOBJ
71CB D9      04930          EXX     HL
71CC 77      04940          LD       (HL), A
71CD D9      04950          EXX     HL

```

```

710E C9      04960      RET
710F 3A5F6F 04970      TOLEFT LD      A, (COLUMN)
7102 4F      04980      LD      C, A
7103 EB      04990      EX      DE, HL
7104 09      05000      ADD     HL, BC      ; CONT. SE FINE OGG.
7105 7D      05010      LD      A, L      ; RAGGIUNTO BORDO SIN.
7106 FE00    05020      CP
710B EB      05030      EX      DE, HL
7109 2005    05040      JR      NZ, DRWOBJ
710B D9      05050      EXX
710C 3600    05060      LD      (HL), 0      ; OGG. NON ESISTENTE
710E D9      05070      EXX      ; SI MUOVE FUORI DAL VIDEO
710F C9      05080      RET
71E0 D9      05090      DRWOBJ EXX
71E1 7E      05100      LD      A, (HL)
71E2 23      05110      INC     HL
71E3 77      05120      LD      (HL), A
71E4 2B      05130      DEC     HL      ; AGGIORNA CONT.CICLI
71E5 D9      05140      EXX
71E6 23      05150      INC     HL
71E7 E5      05160      PUSH   HL      ; PUNT. IND. R A
71E8 23      05170      INC     HL
71E9 23      05180      INC     HL
71EA 23      05190      INC     HL
71EB 5E      05200      LD      E, (HL)      ; RIPRISTINA PUNT. SAGOMA
71EC 23      05210      INC     HL
71ED 56      05220      LD      D, (HL)
71EE 23      05230      INC     HL
71EF 4E      05240      LD      C, (HL)      ; RIPRISTINA PUNT.ATTRIB.
71F0 23      05250      INC     HL
71F1 46      05260      LD      B, (HL)
71F2 ED436A6F 05270      LD      (ATTPTR), BC
71F6 23      05280      INC     HL
71F7 7E      05290      LD      A, (HL)
71F8 32606F 05300      LD      (ROW), A
71FB 23      05310      INC     HL
71FC 7E      05320      LD      A, (HL)
71FD 325F6F 05330      LD      (COLUMN), A
7200 E1      05340      POP    HL
7201 7E      05350      LD      A, (HL)      ; IND. S G
7202 2A6C6F 05360      LD      HL, (NEWPOS)
7205 CD0972 05370      CALL   DRAW
7208 C9      05380      RET
05390 ;
05400 ;
05410 ;
05420 ;***** DRAW *****
05430 ;
05440 ; INPUT HI = INIZIO POS. DISPLAY
05450 ; DE = PUNT. AL D.R. SAGOME
05460 ; A = INDIC. REALE/ABSTRATTO
05470 ; C = N. COLONNE DA VIS.
05480 ; COL. DATO COME VAR
05490 ;
05500 ; VAR COLUMN, ROW, ATTR, DRWPOS
05510 ; SKIP, FILL

```

```

05520 ;
05530 ;
05540 ; REG : A,BC,DE,HL,A'
05550 ;
7209 CD9672          05560 DRAW CALL RSHAPE ;RENDE PUNT. RIGA/COL.
720C 3A606F          05570 LD A,(ROW)
720F 08              05580 EX AF,AF'
7210 D5              05590 LPO PUSH DE
7211 E5              05600 PUSH HL ;MEM. PUNT.LINEA
7212 3A616F          05610 LD A,(SKIP)
7215 4F              05620 LD C,A
7216 0600            05630 LD B,0
7218 09              05640 ADD HL,BC ;SPOSTA PUNT.POS.
7219 87              05650 ADD A,A ;PER MULT.DI 8 BYTES
721A 87              05660 ADD A,A
721B 87              05670 ADD A,A
721C 4F              05680 LD C,A ;SPOSTA PUNT. SAGOMA
721D EB              05690 EX DE,HL
721E 09              05700 ADD HL,BC
721F EB              05710 EX DE,HL
7220 CB44            05720 BIT 0,H ;CONTROLLO SU VIDEO
7222 2804            05730 JR Z,NOSKIP
7224 3E07            05740 LD A,7 ;SE POSSIBILE MUOVI IN SU
7226 84              05750 ADD A,H
7227 67              05760 LD H,A
7228 3A626F          05770 NOSKIP LD A,(FILL)
722B A7              05780 AND A
722C 2B11            05790 JR Z,NXT
722E 4F              05800 LD C,A ;ICOLONNA DA RIEMPIRE
722F E5              05810 LP1 PUSH HL ;CARATTERE PIENO
7230 0608            05820 LD B,B
7232 1A              05830 LP2 LD A,(DE)
7233 77              05840 LD (HL),A ;ICOMPLET.BYTES DEL CAR.
7234 13              05850 INC DE
7235 24              05860 INC H
7236 10FA            05870 DJNZ LP2
7238 E1              05880 POP HL
7239 0D              05890 DEC C
723A 2B03            05900 JR Z,NXT
723C 23              05910 INC HL ;CARATTERE SUCC.
723D 18F0            05920 JR LP1
723F 08              05930 NXT EX AF,AF'
7240 E1              05940 POP HL ;RIPRIST. PUNT. LINEA
7241 D1              05950 POP DE ;PUNT D.B. SAGOMA
7242 3D              05960 DEC A ;CONT.RIGHE
7243 2B1A            05970 JR Z,LDATTR
7245 08              05980 EX AF,AF'
7246 A7              05990 AND A ;AZZERA IL CARRY
7247 0E20            06000 LD C,20H
7249 ED42            06010 SRC HL,BC ;UNA LINEA IN SU
724B CB44            06020 BIT 0,H ;CONTROLLO SUL VIDEO
724D 2B04            06030 JR Z,MODDB
724F 7C              06040 LD A,H
7250 D607            06050 SUB 7
7252 67              06060 LD H,A
7253 3A5F6F          06070 MODDB LD A,(COLUMN)

```



```

7256 B7      06080      ADD      A,A
7257 B7      06090      ADD      A,A
7258 B7      06100      ADD      A,A      ;EL.INIZ. D.B. SAGOME
7259 4F      06110      LD       C,A
725A EB      06120      EX       DE,HL
725B 09      06130      ADD      HL,BC
725C EB      06140      EX       DE,HL
725D 18B1    06150      JR       LPO
725F 2A636F  06160      LDATTR  LD      HL,(ATTPOS)
7262 ED5B6A6F 06170      LD      DE,(ATTPTR)
7266 3A606F  06180      LD      A,(ROW)
7269 08      06190      ATROW   EX       AF,AF'
726A 05      06200      PUSH    DE
726B E5      06210      PUSH    HL
726C 3A616F  06220      LD      A,(SKIP)
726F 4F      06230      LD      C,A
7270 0600    06240      LD      B,0
7272 09      06250      ADD      HL,BC      ;SPOSTATI NEL FILE ATTRIB.
7273 EB      06260      EX       DE,HL
7274 09      06270      ADD      HL,BC      ;SPOSTATI NEL D.B. DEGLI ATTR.
7275 EB      06280      EX       DE,HL
7276 3A626F  06290      LD      A,(FILL)
7279 A7      06300      AND     A
727A 2B07    06310      JR      Z,SKIPAT   ;SALTA ATTRIB
727C 47      06320      LD      B,A        ;ASSEGNA ATTR.
727D 1A      06330      ATTR2  LD      A,(DE)
727E 77      06340      LD      (HL),A
727F 23      06350      INC     HL
7280 13      06360      INC     DE
7281 10FA    06370      DJNZ   ATTR2
7283 E1      06380      SKIPAT POP    HL
7284 D1      06390      POP     DE
7285 3A5F6F  06400      LD      A,(COLUMN)
7288 A7      06410      AND     A        ;AZZERA IL CARRY
7289 0E20    06420      LD      C,20H
728B ED42   06430      SBC    HL,BC     ;PROSS. ATTR. LINEA SOPRA
728D 4F      06440      LD      C,A
728E EB      06450      EX     DE,HL
728F 09      06460      ADD    HL,BC
7290 EB      06470      EX     DE,HL     ;RIPPOS. D.B. ATTR.
7291 08      06480      EX     AF,AF'
7292 3D      06490      DEC    A
7293 20D4    06500      JR     NZ,ATROW
7295 C9      06510      RET
06520 ;
06530 ;
06540 ;***** RSHAPE *****
06550 ;
06560 ;      INPUT HL=>POSIZIONE
06570 ;      A =>IND. REALE/ASTRATTO
06580 ;      DE=>PUNT. SAGOMA
06590 ;      COLONNA
06600 ;
06610 ;      OUTPUT SKIP.FILL.ATTPOS
06620 ;
7296 E5      06630      RSHAPE PUSH   HL

```

7297 08	06640	EX	AF,AF'	J:SAGOMA REALE
7298 261F	06650	LD	H,1FH	
729A 7C	06660	LD	A,H	
729B A5	06670	AND	L	J:MASCHERA PER I PRIMI 5 BIT
729C 6F	06680	LD	L,A	
729D 7C	06690	LD	A,H	
729E 95	06700	SUB	L	J:BOTTRAI DA 1FH
729F 3C	06710	INC	A	
72A0 A4	06720	AND	H	J:ADEQUA PER DIFF. ZERO
72A1 6F	06730	LD	L,A	
72A2 08	06740	EX	AF,AF'	
72A3 A7	06750	AND	A	J:0->ABSTRATTO, 1->REALE
72A4 3A5F6F	06760	LD	A, (COLUMN)	
72A7 200A	06770	JR	NZ,REAL	
72A9 95	06780	SUB	L	
72AA 32626F	06790	LD	(FILL),A	
72AD 7D	06800	LD	A,L	J:RIP. DIFF. ASS.
72AE 32616F	06810	LD	(SKIP),A	
72B1 1811	06820	JR	CALATT	
72B3 BD	06830	REAL CP	L	J:CERCA MIN TRA PIEN/COL
72B4 3807	06840	JR	C,TOOBIG	J:PIEN>COL
72B6 7D	06850	LD	A,L	
72B7 A7	06860	AND	A	
72B8 2003	06870	JR	NZ,TOOBIG	
72BA 3A5F6F	06880	LD	A, (COLUMN)	
72BD 32626F	06890	TOOBIG LD	(FILL),A	
72C0 AF	06900	XOR	A	
72C1 32616F	06910	LD	(SKIP),A	
72C4 E1	06920	CALATT POP	HL	J:CALCOLO PUNT. ATTR.
72C5 E5	06930	PUSH	HL	
72C6 7C	06940	LD	A,H	
72C7 E618	06950	AND	18H	
72C9 CB2F	06960	SRA	A	
72CB CB2F	06970	SRA	A	
72CD CB2F	06980	SRA	A	
72CF C658	06990	ADD	A,58H	
72D1 67	07000	LD	H,A	
72D2 22636F	07010	LD	(ATTPOS),HL	
72D5 E1	07020	POP	HL	
72D6 C9	07030	RET		
	07040	;		
	07050	;		
72D7 210040	07060	CLRSCR LD	HL,4000H	J:HL->INIZIO VIDEO
72DA 110140	07070	LD	DE,4001H	
72DD 01FF17	07080	LD	BC,6143	J:DIM.VIDEO 17FFH
72E0 AF	07090	XOR	A	J:VIDEO BIANCO
72E1 77	07100	LD	(HL),A	
72E2 EDB0	07110	LDIR		
72E4 210058	07120	LD	HL,5800H	J:SEL.PRIMA LINEA PER
72E7 110158	07130	LD	DE,5801H	J:MEM. FILE ATTRIB.
72EA 011F00	07140	LD	BC,31	
72ED 3607	07150	LD	(HL),7	J:INK SETTE
72EF EDB0	07160	LDIR		
72F1 212058	07170	LD	HL,5820H	J:SEL. ATTRIB. A PARTIRE
72F4 112158	07180	LD	DE,5821H	J:DALLA SECONDA LINEA
72F7 01DF02	07190	LD	BC,735	

```

72FA 77      07200      LD      (HL),A      ;(PAPER 0)*8+(INK 0)
72FB EDB0    07210      LDIR
72FD 21A05B  07220      LD      HL,5BA0H    ;SETTA I BORDI
7300 116059  07230      LD      DE,5960H    ;ALTO,MEDIO,BASSO
7303 01205A  07240      LD      BC,5A20H
7306 3E38    07250      LD      A,56        ;(PAPER 7)*8+(INK 0)
7308 D9      07260      EXX
7309 0620    07270      LD      B,32        ;RICOPRI UNA RIGA
730B D9      07280      EXX      HWYATT
730C 77      07290      LD      (HL),A
730D 12      07300      LD      (DE),A
730E 02      07310      LD      (BC),A
730F 23      07320      INC     HL
7310 13      07330      INC     DE
7311 03      07340      INC     BC
7312 D9      07350      EXX
7313 10F6    07360      DJNZ   HWYATT
7315 D9      07370      EXX
7316 C9      07380      RET
          07390      ;
          07400      ;
7317 E5      07410      SHAPE  PUSH     HL      ;SALVA HL
7318 3A7B6E  07420      LD      A,(FRGDIR)
731B 87      07430      ADD     A,A
731C 21AF69  07440      LD      HL,FRGSHP
731F 1600    07450      LD      D,0
7321 5F      07460      LD      E,A
7322 19      07470      ADD     HL,DE      ;PUNT.ALLA POS. SAGOMA
7323 5E      07480      LD      E,(HL)    ;DE RESTITUISCE PUNT. A SAGOMA
7324 23      07490      INC     HL
7325 56      07500      LD      D,(HL)
7326 E1      07510      POP     HL
7327 C9      07520      RET
          07530      ;
          07540      ;
          07550      ;***** DISASC *****
          07560      ;
          07570      ;      VIS.IL VALORE ASCII DAL CHARACTER SET
          07580      ;      ND---CONSERVA DE.IL PUNTATORE HL
          07590      ;      CONSERVA LO STESSO VALORE DOPO
          07600      ;      LA VISUALIZZAZIONE SE SI E'
          07610      ;      USATA BENE LA COPPIA DI REG. BC
          07620      ;
7328 C5      07630      DISASC  PUSH     BC
7329 D5      07640      PUSH     DE
732A E5      07650      PUSH     HL
732B 1A      07660      LD      A,(DE)    ;CARICA IL CARATT. ASCII
732C 6F      07670      LD      L,A
732D 2600    07680      LD      H,0
732F 29      07690      ADD     HL,HL      ;MULTIPLI DI 8 BYTES
7330 29      07700      ADD     HL,HL
7331 29      07710      ADD     HL,HL
7332 EB      07720      EX      DE,HL
7333 21003C  07730      LD      HL,CHRSET ;INIZIO CHARACTER SET
7336 19      07740      ADD     HL,DE
7337 EB      07750      EX      DE,HL

```

7338	E1	07760	POP	HL	
7339	0608	07770	DRWCHR	LD	B,B
733B	E5	07780	PUSH	HL	
733C	1A	07790	CHARLP	LD	A,(DE)
733D	77	07800		LD	(HL),A
733E	13	07810	INC	DE	
733F	24	07820	INC	H	
7340	10FA	07830	DJNZ	CHARLP	
7342	E1	07840	POP	HL	
7343	D1	07850	POP	DE	
7344	23	07860	INC	HL	!PUNT.POS.
7345	13	07870	INC	DE	!PUNT.MESSAGGIO
7346	C1	07880	POP	BC	
7347	10DF	07890	DJNZ	DISASC	
7349	C9	07900	RET		
		07910	;		
		07920	;		
734A	D9	07930	POLICE	EXX	
734B	216D6E	07940		LD	HL,PCAREXT
734E	7E	07950		LD	A,(HL)
734F	E5	07960	PUSH	HL	!TEST ESISTENZA POLIZIA
7350	D9	07970	EXX		
7351	A7	07980	AND	A	
7352	2023	07990	JR	NZ,MOVPC	!MOV. AUTO POLIZIA
7354	D1	08000	POP	DE	!PUNT. D.B. ESIST.
7355	CDCC77	08010	CALL	RANDNO	!MUOVI SE MULT. DI
7358	E61F	08020	AND	1FH	!31
735A	FE1F	08030	CP	1FH	
735C	C0	08040	RET	NZ	
735D	3E01	08050	LD	A,1	!SETTA INDIC. DI INS.
735F	32726F	08060	LD	(CHASE),A	
7362	21F56E	08070	LD	HL,RPCDB	!DESTRA
7365	CDCC77	08080	CALL	RANDNO	
7368	E601	08090	AND	1	
736A	2803	08100	JR	Z,RHTPC	
736C	21DD6E	08110	LD	HL,LPCDB	
736F	010C00	08120	RHTPC	LD	BC,12
7372	EDB0	08130		LDIR	
7374	D9	08140	EXX		
7375	E5	08150	PUSH	HL	
7376	D9	08160	EXX		
7377	E1	08170	MOVPC	POP	HL
7378	23	08180	INC	HL	!PUNT. ESIST.
7379	23	08190	INC	HL	
737A	7E	08200	LD	A,(HL)	!DIREZIONE
737B	47	08210	LD	B,A	!MEM.DIR.
737C	23	08220	INC	HL	
737D	23	08230	INC	HL	!PUNT.POS.
737E	226E6F	08240	LD	(POSPTR),HL	
7381	5E	08250	LD	E,(HL)	
7382	23	08260	INC	HL	
7383	56	08270	LD	D,(HL)	
7384	1C	08280	INC	E	!MOV. VERSO DESTRA
7385	A7	08290	AND	A	
7386	2B02	08300	JR	Z,PCMRHT	!AUTO MUOVE A DESTRA
7388	1D	08310	DEC	E	

73B9 1D	08320	DEC	E	
73BA ED536C6F	08330	FCMRHT	LD	(NEWPOS), DE
73BE 3E02	08340	LD	A, 2	
7390 32606F	08350	LD	(ROW), A	;DUE RIGHE
7393 3E06	08360	LD	A, 6	
7395 325F6F	08370	LD	(COLUMN), A	
7398 C5	08380	PUSH	BC	;DIREZIONE
7399 3A706E	08390	LD	A, (PCARRAP)	;IND. REALE/ASTR.
739C EB	08400	EX	DE, HL	
739D CD9672	08410	CALL	RSHAPE	;RIPRIET. BAGOMA
73A0 2A636F	08420	LD	HL, (ATTPOS)	
73A3 F1	08430	POP	AF	
73A4 A7	08440	AND	A	
73A5 2004	08450	JR	NZ, PCTAH	;SE 1 ALLORA OK
73A7 010500	08460	LD	BC, 5	;CONTROLLO INIZIO AUTO POL.
73AA 09	08470	ADD	HL, BC	
73AB 7E	08480	PCTAH	LD	A, (HL)
73AC E607	08490	AND	7	
73AE 012000	08500	LD	BC, 32	
73B1 A7	08510	AND	A	
73B2 ED42	08520	SBC	HL, BC	
73B4 FE04	08530	CP	4	
73B6 2B07	08540	JR	Z, ISFRG2	
73B8 7E	08550	LD	A, (HL)	
73B9 E607	08560	AND	7	
73BB FE04	08570	CP	4	
73BD 2009	08580	JR	NZ, NFROG2	
73BF 3E01	08590	ISFRG2	LD	A, 1
73C1 327C6F	08600	LD	(CRHFLG), A	;SETTA IND. CRASH
73C4 3D	08610	DEC	A	;COLORE BIANCO
73C5 77	08620	LD	(HL), A	;BIANCO DAVANTI A A.P.
73C6 09	08630	ADD	HL, BC	
73C7 77	08640	LD	(HL), A	
73C8 CDDF73	08650	NFROG2	CALL	STRPC
73CB 2A6E6F	08660	LD	HL, (POSPTR)	
73CE ED5B6C6F	08670	LD	DE, (NEWPOS)	
73D2 73	08680	LD	(HL), E	
73D3 23	08690	INC	HL	
73D4 72	08700	LD	(HL), D	
73D5 CDAF71	08710	CALL	MVCTRL	
73D8 D9	08720	EXX		
73D9 7E	08730	LD	A, (HL)	
73DA 32726F	08740	LD	(CHASE), A	
73DD D9	08750	EXX		
73DE C9	08760	RET		
	08770	;		
	08780	;		
	08790	;		
	08800	;		
	08810	;		
	08820	;		
	08830	STRPC		
	08840	LD	HL, (NEWPOS)	;PUNT. POS.
	08850	EX	DE, PCSTR	;MEM. LOC.
	08860	LD	(HL), E	
	08870	INC	HL	;MEM. POS.

CONSERVAZ. DI CIO' CHE COPRE AUTO POLIZIA

73EB 72	08880	LD	(HL),D	
73E9 23	08890	INC	HL	
73EA EB	08900	EX	DE,HL	
73EB 21606F	08910	LD	HL,ROW	
73EE 7E	08920	LD	A,(HL)	;CARICA 5 BYTES DI IMP.
73EF 010500	08930	LD	BC,5	
73F2 EDB0	08940	LDIR		
73F4 08	08950	EX	AF,AF'	
73F5 2A6C6F	08960	LD	HL,(NEWPOS)	
73F8 E5	08970	SPCLP1	PUSH	HL
73F9 3A616F	08980	LD	A,(SKIP)	
73FC 4F	08990	LD	C,A	
73FD 09	09000	ADD	HL,BC	
73FE CB44	09010	BIT	O,H	
7400 2B04	09020	JR	Z,NSSPS	
7402 7C	09030	LD	A,H	
7403 C607	09040	ADD	A,7	
7405 67	09050	LD	H,A	
7406 3A626F	09060	NSSPS	LD	A,(FILL)
7409 A7	09070	AND	A	
740A 2B0F	09080	JR	Z,NXTSPC	
740C 4F	09090	LD	C,A	
740D E5	09100	SPCLP2	PUSH	HL
740E 0608	09110	LD	B,B	
7410 7E	09120	SPCLP3	LD	A,(HL)
7411 12	09130	LD	(DE),A	
7412 13	09140	INC	DE	
7413 24	09150	INC	H	
7414 10FA	09160	DJNZ	SPCLP3	
7416 E1	09170	POP	HL	
7417 23	09180	INC	HL	
7418 0D	09190	DEC	C	;CAR. SUCC.
7419 20F2	09200	JR	NZ,SPCLP2	
741B E1	09210	NXTSPC	POP	HL
741C 08	09220	EX	AF,AF'	;RIP. CONT. RIGHE
741D 3D	09230	DEC	A	
741E 2B0F	09240	JR	Z,SPCATR	;RIPR. ATTR. POLIZIA
7420 08	09250	EX	AF,AF'	
7421 0E20	09260	LD	C,32	
7423 ED42	09270	SBC	HL,BC	;SALI DI UNA LINEA
7425 CB44	09280	BIT	O,H	
7427 2BCF	09290	JR	Z,SPCLP1	;FINE VIDEO?
7429 7C	09300	LD	A,H	
742A D607	09310	SUB	7	
742C 67	09320	LD	H,A	
742D 18C9	09330	JR	SPCLP1	
742F 2A636F	09340	SPCATR	LD	HL,(ATTPOS)
7432 3A606F	09350	LD	A,(ROW)	
7435 08	09360	EX	AF,AF'	
7436 E5	09370	SPCAT1	PUSH	HL
7437 3A616F	09380	LD	A,(SKIP)	
743A 4F	09390	LD	C,A	
743B 09	09400	ADD	HL,BC	
743C 3A626F	09410	LD	A,(FILL)	
743F A7	09420	AND	A	
7440 2B03	09430	JR	Z,NXTSPA	

7442	4F	09440	LD	C, A	
7443	EDB0	09450	LDIR		
7445	E1	09460	POP	HL	
7446	08	09470	EX	AF, AF'	
7447	3D	09480	DEC	A	
7448	CB	09490	RET	Z	
7449	08	09500	EX	AF, AF'	
744A	0E20	09510	LD	C, 32	
744C	ED42	09520	SBC	HL, BC	
744E	18E6	09530	JR	SPCAT1	
		09540	;		
		09550	;		
7450	3A6D6E	09560	RESPC	LD	A, (PCAREXT)
7453	A7	09570	AND	A	;TEST ESISTENZA POL.
7454	CB	09580	RET	Z	
7455	11606F	09590	LD	DE, ROW	
7458	21AF6D	09600	LD	HL, PCSTR+2	
745B	010500	09610	LD	BC, 5	
745E	EDB0	09620	LDIR		;RIP. 5 BYTES INF.
7460	EB	09630	EX	DE, HL	;DE CONTIENE PUNT.
7461	2AAD6D	09640	LD	HL, (PCSTR)	;CARICA POS.
7464	3A606F	09650	LD	A, (ROW)	
7467	08	09660	EX	AF, AF'	
7468	E5	09670	RPCLP1	PUSH	HL
7469	3A616F	09680	LD	A, (SKIP)	;SALVA POS
746C	4F	09690	LD	C, A	
746D	09	09700	ADD	HL, BC	
746E	CB44	09710	BIT	O, H	
7470	2804	09720	JR	Z, NSRPS	
7472	3E07	09730	LD	A, 7	
7474	B4	09740	ADD	A, H	
7475	67	09750	LD	H, A	
7476	3A626F	09760	NSRPS	LD	A, (FILL)
7479	A7	09770	AND	A	
747A	280F	09780	JR	Z, NXTRPC	
747C	4F	09790	LD	C, A	
747D	E5	09800	RPCLP2	PUSH	HL
747E	0608	09810	LD	B, 8	
7480	1A	09820	RPCLP3	LD	A, (DE)
7481	77	09830	LD	(HL), A	;RIP. CARATT.
7482	13	09840	INC	DE	
7483	24	09850	INC	H	
7484	10FA	09860	DJNZ	RPCLP3	
7486	E1	09870	POP	HL	
7487	23	09880	INC	HL	
7488	0D	09890	DEC	C	
7489	20F2	09900	JR	NZ, RPCLP2	
748B	E1	09910	NXTRPC	POP	HL
748C	08	09920	EX	AF, AF'	
748D	3D	09930	DEC	A	;AQQ. CONT. RIOME
748E	280F	09940	JR	Z, RPCATR	;RIP. AUTO POL.
7490	08	09950	EX	AF, AF'	
7491	0E20	09960	LD	C, 32	
7493	ED42	09970	SBC	HL, BC	;SALI DI UNA RIGA
7495	CB44	09980	BIT	O, H	
7497	2BCF	09990	JR	Z, RPCLP1	

7499	7C	10000	LD	A, H		
749A	D607	10010	SUB	7	; VAL. LIMITE	
749C	67	10020	LD	H, A		
749D	18C9	10030	JR	RPCLP1		
749F	2A636F	10040	LD	HL, (ATTPOS)	; CAR. ATTR. INIZ. POS.	
74A2	3A606F	10050	LD	A, (RDW)		
74A5	08	10060	EX	AF, AF'		
74A6	E5	10070	RPCAT1	PUSH	HL	
74A7	3A616F	10080	LD	A, (SKIP)		
74AA	4F	10090	LD	C, A		
74AB	09	10100	ADD	HL, BC		
74AC	3A626F	10110	LD	A, (FILL)		
74AF	A7	10120	AND	A		
74B0	2B05	10130	JR	Z, NXTRPA		
74B2	EB	10140	EX	DE, HL		
74B3	4F	10150	LD	C, A		
74B4	EDB0	10160	LDIR			
74B6	EB	10170	EX	DE, HL		
74B7	E1	10180	NXTRPA	POP	HL	
74B8	08	10190	EX	AF, AF'		
74B9	3D	10200	DEC	A		
74BA	C8	10210	RET	Z		
74BB	08	10220	EX	AF, AF'		
74BC	0E20	10230	LD	C, 32		
74BE	ED42	10240	SBC	HL, BC		
74C0	1BE4	10250	JR	RPCAT1		
		10260	;			
		10270	;			
74C2	3A7C6F	10280	FR0G	LD	A, (CRHFLG)	; IND. CRASH
74C5	A7	10290	AND	A		
74C6	2017	10300	JR	NZ, FRGCRH		; INCIDENTE
74C8	325E6F	10310	LD	(UPDWN), A		; PONI A 0 IND. PUNT.
74CB	CDE374	10320	CALL	REGFRG		; RIGEN RAMA
74CE	217A6E	10330	LD	HL, FRGCYC		; TEST MOV
74D1	35	10340	DEC	(HL)		
74D2	C0	10350	RET	NZ		
74D3	2B	10360	DEC	HL		
74D4	7E	10370	LD	A, (HL)		; AZZERA CONT. CICLI
74D5	23	10380	INC	HL		
74D6	77	10390	LD	(HL), A		
74D7	CD1075	10400	CALL	MOVFRG		
74DA	3A7C6F	10410	LD	A, (CRHFLG)		
74DD	A7	10420	AND	A		
74DE	C8	10430	RET	Z		
74DF	CD9176	10440	FRGCRH	CALL	CRASH	
74E2	C9	10450	RET			
		10460	;			
		10470	;	***** REGFRG *****		
		10480	;			
		10490	;			
		10500	;	RIGENERA RAMA SE VE NE SONO		
		10510	;	ALTRIMENTI PONI A 0 QAMFL		
		10520	;			
74E3	3A796E	10520	REGFRG	LD	A, (FRGEXT)	
74E6	A7	10530	AND	A		
74E7	C0	10540	RET	NZ		; RETURN SE ESISTE
74E8	21816E	10550	LD	HL, FRGDB		

74EB	11796E	10560	LD	DE, FRGEXT	
74EE	010800	10570	LD	BC, 8	
74F1	EDB0	10580	LDIR		
74F3	21846E	10590	LD	HL, FRGSTN	;CARICA POS. RANA
74F6	35	10600	DEC	(HL)	;SPOSTATI DI 3 CAR. A SIN.
74F7	35	10610	DEC	(HL)	
74F8	35	10620	DEC	(HL)	
74F9	2A7C6E	10630	LD	HL, (FRGPOS)	
74FC	22786F	10640	LD	(OLDFRG), HL	
74FF	227A6F	10650	LD	(NEWFRG), HL	
7502	21896D	10660	LD	HL, FRGSTR	;IN MEM. PER RIP.
7505	118A6D	10670	LD	DE, FRGSTR+1	;BIANCO POS. RANA
7508	012300	10680	LD	BC, 35	
750B	3600	10690	LD	(HL), 0	
750D	EDB0	10700	LDIR		
750F	C9	10710	RET		
		10720	;		
		10730	;	***** MOVFRG *****	
		10740	;		
		10750	;	MOVIMENTI RANA, SALVAT. E RIPRIS. FIGURE	
		10760	;		
7510	AF	10770	MOVFRG XOR	A	
7511	2120E0	10780	LD	HL, 0E020H	;M=-32, L=32
7514	4F	10790	LD	C, A	;C->MOV. ASS.
7515	08	10800	EX	AF, AF?	
7516	3EDF	10810	LD	A, 0DFH	;TEST A DESTRA
7518	DBFE	10820	IN	A, (OFEH)	
751A	E601	10830	AND	1	
751C	2006	10840	JR	NZ, LEFT	
751E	0C	10850	INC	C	
751F	11D769	10860	LD	DE, FROG2	
7522	0601	10870	LD	B, 1	
7524	3EDF	10880	LEFT LD	A, 0DFH	;TEST IN BASSO
7526	DBFE	10890	IN	A, (OFEH)	
7528	E604	10900	AND	4	
752A	2006	10910	JR	NZ, DOWN	
752C	0D	10920	DEC	C	
752D	11176A	10930	LD	DE, FROG4	
7530	0403	10940	LD	B, 3	
7532	3EFD	10950	DOWN LD	A, 0DFH	;ADD 32
7534	DBFE	10960	IN	A, (OFEH)	
7536	E601	10970	AND	1	
7538	200B	10980	JR	NZ, UP	
753A	79	10990	LD	A, C	
753B	85	11000	ADD	A, L	;ADD 32
753C	4F	11010	LD	C, A	
753D	08	11020	EX	AF, AF?	
753E	3D	11030	DEC	A	
753F	08	11040	EX	AF, AF?	;DECR. IND. ALTO/BASSO
7540	11F769	11050	LD	DE, FROG3	
7543	0602	11060	LD	B, 2	
7545	3EF7	11070	UP LD	A, 0F7H	;TEST IN ALTO
7547	DBFE	11080	IN	A, (OFEH)	
7549	E601	11090	AND	1	
754B	200B	11100	JR	NZ, VALID	
754D	79	11110	LD	A, C	

754E	B4	11120	ADD	A, H	
754F	4F	11130	LD	C, A	J'ADD -32
7550	08	11140	EX	AF, AF'	
7551	3C	11150	INC	A	
7552	08	11160	EX	AF, AF'	
7553	11B769	11170	LD	DE, FROG1	
7556	0600	11180	LD	B, 0	
7558	78	11190	LD	A, B	J'MEM. DIR. TEMP.
7559	327D6F	11200	LD	(TEMDIR), A	
755C	ED53806F	11210	LD	(TEMSPH), DE	J'MEM. SAGOMA TEMP.
7560	AF	11220	XOR	A	
7561	B9	11230	CP	C	
7562	C8	11240	RET	Z	J'SE FERMO TORNA
7563	2A786F	11250	LD	HL, (OLDFRG)	
7566	CB79	11260	BIT	7, C	J'TEST -VE
7568	47	11270	LD	B, A	
7569	1E07	11280	LD	E, 7	J'PER LIMITE VIDEO
756B	2803	11290	JR	Z, NETDWN	J'ADEGUA MOV.
756D	05	11300	DEC	B	
756E	1EF9	11310	LD	E, -7	
7570	09	11320	ADD	HL, BC	
7571	CB44	11330	BIT	0, H	
7573	2803	11340	JR	Z, VALID1	J'LIMITE NON RAGG.
7575	7C	11350	LD	A, H	
7576	83	11360	ADD	A, E	
7577	67	11370	LD	H, A	J'AGG. MOB
7578	227E6F	11380	LD	(TEMPOS), HL	
757B	EB	11390	EX	DE, HL	
757C	3E40	11400	LD	A, 40H	J'TEST SU VIDEO SUP.
757E	BA	11410	CP	D	
757F	7E	11420	LD	A, E	
7580	2004	11430	JR	NZ, VALID2	
7582	FE20	11440	CP	20H	
7584	382F	11450	JR	C, NVALID	
7586	E61F	11460	AND	1FH	J'TEST SU LIM. DESTRO
7588	FE1F	11470	CP	1FH	
758A	2829	11480	JR	Z, NVALID	
758C	21BE50	11490	LD	HL, 50BEH	J'TEST SU LIM. INF.
758F	A7	11500	AND	A	
7590	ED52	11510	SBC	HL, DE	
7592	3821	11520	JR	C, NVALID	
7594	217E50	11530	LD	HL, 507EH	J'TEST SU POS. PANA
7597	ED52	11540	SBC	HL, DE	
7599	3011	11550	JR	NC, YVALID	
759B	7B	11560	LD	A, E	J'TEST SU CASA
759C	E61F	11570	AND	1FH	
759E	67	11580	LD	H, A	
759F	3A846E	11590	LD	A, (FRGSTN)	J'TEST ULTIMA PANA
75A2	FEA0	11600	CP	0A0H	J'NON POSIZ.
75A4	3806	11610	JR	C, YVALID	J'SE NON VE NE SONOULTIMA PANA
75A6	3C	11620	INC	A	J'NON POSIZ. SE
75A7	E61F	11630	AND	1FH	J'NON VE NE SONO
75A9	94	11640	SUB	H	
75AA	3009	11650	JR	NC, NVALID	
75AC	ED537A6F	11660	LD	(NEWFRG), DE	J'MEM. NUOVA POR.
75B0	08	11670	EX	AF, AF'	

75B1	325E6F	11680	LD	(UPDOWN),A	
75B4	08	11690	EX	AF,AF'	
75B5	2A786F	11700	NVALID	LD	HL,(OLDFRG)
75B8	A7	11710	AND	A	;OLDFRG=NEWFRGT
75B9	ED52	11720	SBC	HL,DE	
75BB	7D	11730	LD	A,L	
75BC	B4	11740	OR	H	
75BD	CB	11750	RET	Z	;RETURN SE UGUALE
75BE	CDD675	11760	CALL	RESFRG	;RIP.RAMA
75C1	2A7A6F	11770	LD	HL,(NEWFRG)	;AGG.VECCHIA POS.
75C4	22786F	11780	LD	(OLDFRG),HL	
75C7	217D6F	11790	LD	HL,TEMDIR	
75CA	117B6E	11800	LD	DE,FRGDIR	
75CD	010500	11810	LD	BC,5	
75D0	EDB0	11820	LDIR		
75D2	CD2B76	11830	CALL	STRFRG	
75D5	C9	11840	RET		
		11850	;		
		11860	;		
75D6	11896D	11870	RESFRG	LD	DE,FRGSTR
75D9	2A786F	11880	LD	HL,(OLDFRG)	;PUNT.MEM.
75DC	E5	11890	PUSH	HL	;DA OLOPOS
75DD	3E02	11900	LD	A,2	
75DF	08	11910	EX	AF,AF'	;CONT.RIGHE
75E0	E5	11920	RFRLP1	PUSH	
75E1	0E02	11930	LD	C,2	;CONT. COL.
75E3	E5	11940	RFRLP2	PUSH	
75E4	0608	11950	LD	B,8	
75E6	1A	11960	RFRLP3	LD	A,(DE)
75E7	77	11970	LD	(HL),A	;RIP. DA D.B.
75E8	13	11980	INC	DE	;NELLO SCHERMO
75E9	24	11990	INC	H	
75EA	10FA	12000	DJNZ	RFRLP3	;BYTE SUCC. DEL CAR.
75EC	E1	12010	POP	HL	
75ED	23	12020	INC	HL	
75EE	0D	12030	DEC	C	;CONT.COL.
75EF	20F2	12040	JR	NZ,RFRLP2	
75F1	E1	12050	POP	HL	
75F2	08	12060	EX	AF,AF'	
75F3	3D	12070	DEC	A	;CONT. RIGHE
75F4	2B10	12080	JR	Z,RFRATR	
75F6	08	12090	EX	AF,AF'	
75F7	A7	12100	AND	A	
75F8	0E20	12110	LD	C,32	;SALI DI UNA LINEA
75FA	ED42	12120	SBC	HL,BC	
75FC	CB44	12130	BIT	O,H	
75FE	28E0	12140	JR	Z,RFRLP1	
7600	7C	12150	LD	A,H	
7601	D607	12160	SUB	7	
7603	67	12170	LD	H,A	
7604	18DA	12180	JR	RFRLP1	
7606	E1	12190	RFRATR	POP	HL
7607	7C	12200	LD	A,H	
7608	E618	12210	AND	18H	
760A	CB2F	12220	SRA	A	
760C	CB2F	12230	SRA	A	

760E	CB2F	12240	SRA	A	
7610	C658	12250	ADD	A,58H	
7612	67	12260	LD	H,A	
7613	3E02	12270	LD	A,2	J CONT.RIGA
7615	08	12280	EX	AF,AF'	
7616	E5	12290	RFRAT1 PUSH	HL	
7617	EB	12300	EX	DE,HL	
7618	0E02	12310	LD	C,2	J RIP.ATTR.
761A	EDB0	12320	LDIR		
761C	EB	12330	EX	DE,HL	
761D	E1	12340	POP	HL	
761E	08	12350	EX	AF,AF'	
761F	3D	12360	DEC	A	J AGG.CONT.RIGHE
7620	C8	12370	RET	Z	
7621	08	12380	EX	AF,AF'	
7622	0E20	12390	LD	C,32	
7624	ED42	12400	SBC	HL,BC	
7626	18EE	12410	JR	RFRAT1	
		12420 ;			
		12430 ;			
7628	11896D	12440	STRFRG LD	DE,FRGSTR	
762B	2A7A6F	12450	LD	HL, (NEWFRG)	J MEM. SECONDO N.POS
762E	D9	12460	EXX		
762F	2A7E6E	12470	LD	HL, (FROGSH)	J CARICA SAGOMA
7632	D9	12480	EXX		
7633	E5	12490	PUSH	HL	
7634	3E02	12500	LD	A,2	
7636	08	12510	EX	AF,AF'	
7637	E5	12520	SFRLP1 PUSH	HL	
7638	0E02	12530	LD	C,2	
763A	E5	12540	SFRLP2 PUSH	HL	
763B	0608	12550	LD	B,B	J MEM.E RIP. UN CAR. AR
763D	7E	12560	SFRLP3 LD	A, (HL)	
763E	12	12570	LD	(DE),A	
763F	D9	12580	EXX		
7640	7E	12590	LD	A, (HL)	
7641	23	12600	INC	HL	
7642	D9	12610	EXX		
7643	77	12620	LD	(HL),A	
7644	13	12630	INC	DE	
7645	24	12640	INC	H	
7646	10F5	12650	DJNZ	SFRLP3	
7648	E1	12660	POP	HL	
7649	23	12670	INC	HL	J CAP.3UCC.
764A	0D	12680	DEC	C	
764B	20ED	12690	JR	NZ,SFRLP2	
764D	E1	12700	POP	HL	
764E	08	12710	EX	AF,AF'	
764F	3D	12720	DEC	A	
7650	2810	12730	JR	Z,SFRATR	
7652	08	12740	EX	AF,AF'	
7653	A7	12750	AND	A	
7654	0E20	12760	LD	C,32	
7656	ED42	12770	SBC	HL,BC	J PROSSIMA RIGA
7658	CB44	12780	BIT	O,H	
765A	28DB	12790	JR	Z,SFRLP1	

765C	7C	12800	LD	A,H		
765D	D607	12810	SUB	7		
765F	67	12820	LD	H,A		
7660	18D5	12830	JR	SFRLP1		
7662	E1	12840	SFRATR	POP	HL	
7663	7C	12850	LD	A,H	J CAL.COLO ATTP.POS.	
7664	E618	12860	AND	18H		
7666	CB2F	12870	SRA	A		
7668	CB2F	12880	SRA	A		
766A	CB2F	12890	SRA	A		
766C	C65B	12900	ADD	A,5BH		
766E	67	12910	LD	H,A		
766F	3E02	12920	LD	A,2		
7671	08	12930	EX	AF,AF'		
7672	0602	12940	SFRAT1	LD	B,2	
7674	E5	12950	PUSH	HL		
7675	7E	12960	SFRATLP	LD	A,(HL)	
7676	12	12970	LD	(DE),A		
7677	3604	12980	LD	(HL),4	J IACC.ATTR. PANA	
7679	23	12990	INC	HL		
767A	13	13000	INC	DE		
767B	E607	13010	AND	7	J TEST SU INCIDENTE	
767D	2805	13020	JR	Z,NFR0G3		
767F	3E01	13030	LD	A,1		
7681	327C6F	13040	LD	(CRHFLG),A		
7684	10EF	13050	NFR0G3	DJNZ	SFRATLP	
7686	E1	13060	POP	HL		
7687	08	13070	EX	AF,AF'		
7688	3D	13080	DEC	A		
7689	08	13090	RET	Z		
768A	08	13100	EX	AF,AF'		
768B	0E20	13110	LD	C,32		
768D	ED42	13120	SBC	HL,BC		
768F	18E1	13130	JR	SFRAT1		
		13140	;			
		13150	;			
7691	AF	13160	CRASH	XDR	A	
7692	327C6F	13170	LD	(CRHFLG),A	J AZZERA IND. DI INCID.	
7695	32796E	13180	LD	(FRGEXT),A	J AZZERA IND. ERIQT.	
7698	CDA776	13190	CALL	FRGDIE	J ROUTINE MORTE	
769B	CDD675	13200	CALL	RESFRG		
769E	21826F	13210	LD	HL,NUMFRG		
76A1	35	13220	DEC	(HL)	J DECR. NUM. RANE	
76A2	C0	13230	RET	NZ		
76A3	32776F	13240	LD	(GAMFLG),A	J AZZERA IND.GIOCO	
76A6	C9	13250	RET		J SE NON ESIST. RANE	
		13260	;			
		13270	;			
76A7	2A786F	13280	FRGDIE	LD	HL,(OLDFRG)	J VECCHIA POS.
76AA	010240	13290		LD	BC,4002H	J COLORE ROSSO
76AD	D9	13300	EXX			
76AE	21396F	13310	LD	HL,DIETON		J SEL.TONO TRISTE
76B1	D9	13320	EXX			
76B2	7C	13330	LD	A,H		J TEST SU FINE VIAGGIO
76B3	8B	13340	CP	B		
76B4	2016	13350	JR	NZ,NOTEND		

76B6	7D	13360	LD	A, L	
76B7	8B	13370	CP	B	
76BB	3012	13380	JR	NC, NOTEND	
76BA	11466F	13390	LD	DE, SCORE+3	: BONUS 100 PT
76BD	EB	13400	EX	DE, HL	
76BE	34	13410	INC	(HL)	
76BF	21476F	13420	LD	HL, SCORE+4	
76C2	CD4B77	13430	CALL	DISSCR	
76C5	0E06	13440	LD	C, 6	: GIALLO
76C7	D9	13450	EXX		
76CB	21156F	13460	LD	HL, HOMTON	
76CB	D9	13470	EXX		
76CC	79	13480	LD	A, C	
76CD	32656F	13490	LD	(ATTR), A	
76D0	2A7B6F	13500	LD	HL, (OLDFRG)	
76D3	ED5B7E6E	13510	LD	DE, (FROGSH)	
76D7	CD7A70	13520	CALL	DRWFRG	
76DA	112000	13530	LD	DE, 32	: ADATT. LINEA
76DD	19	13540	ADD	HL, DE	
76DE	08	13550	EX	AF, AF'	
76DF	3A656F	13560	LD	A, (ATTR)	
76E2	08	13570	EX	AF, AF'	
76E3	0605	13580	LD	B, 5	
76E5	C5	13590	PUSH	BC	
76E6	E5	13600	PUSH	HL	: PUNT. ATTR.
76E7	AF	13610	XOR	A	: INK E PAPER NERI
76EB	77	13620	LD	(HL), A	
76E9	23	13630	INC	HL	
76EA	77	13640	LD	(HL), A	
76EB	ED52	13650	SBC	HL, DE	
76ED	77	13660	LD	(HL), A	
76EE	2B	13670	DEC	HL	
76EF	77	13680	LD	(HL), A	
76F0	CD0B77	13690	CALL	FRGTON	: GENERA TONO
76F3	E1	13700	POP	HL	
76F4	E5	13710	PUSH	HL	
76F5	08	13720	EX	AF, AF'	
76F6	77	13730	LD	(HL), A	: PAPER NERO
76F7	23	13740	INC	HL	: ROSSO O GIALLO
76F8	77	13750	LD	(HL), A	
76F9	A7	13760	AND	A	
76FA	ED52	13770	SBC	HL, DE	
76FC	77	13780	LD	(HL), A	
76FD	2B	13790	DEC	HL	
76FE	77	13800	LD	(HL), A	
76FF	08	13810	EX	AF, AF'	
7700	CD0B77	13820	CALL	FRGTON	
7703	E1	13830	POP	HL	
7704	C1	13840	POP	BC	
7705	10DE	13850	DJNZ	FLASLP	
7707	C9	13860	RET		
		13870	;		
		13880	;		
7708	D9	13890	FRGTON	EXX	
7709	E5	13900	PUSH	HL	
770A	CDB577	13910	CALL	TONE1	

770D E1	13920		POP	HL	
770E 010400	13930		LD	BC, 4	; DB MOV. IN BASSO
7711 08	13940		EX	AF, AF'	
7712 FE06	13950		CP	6	
7714 2B03	13960		JR	Z, HOME	
7716 01FCFF	13970		LD	BC, -4	; DB MOV. IN ALTO
7719 09	13980	HOME	ADD	HL, BC	
771A D9	13990		EXX		
771B 08	14000		EX	AF, AF'	
771C C9	14010		RET		
	14020 ;				
	14030 ;				
771D 3A796E	14040	CALSCR	LD	A, (FRGEXT)	; TEST ESIST.
7720 A7	14050		AND	A	
7721 C8	14060		RET	Z	; NESSUN PUNT.
7722 3A5E6F	14070		LD	A, (UPDOWN)	; TEST MOV. SU/GIU
7725 A7	14080		AND	A	; TEST PER PUNT.
7726 C8	14090		RET	Z	
7727 21476F	14100		LD	HL, SCORE+4	; BONUS 10 PT
772A CB7F	14110		BIT	7, A	; TEST MOV. IN GIU'
772C 2003	14120		JR	NZ, DWNSCR	; PUNT. RELATIVO
772E 34	14130		INC	(HL)	
772F 1B1A	14140		JR	DISSCR	; VIS.PUNTI
7731 3A796F	14150	DWNSCR	LD	A, (OLDFRG+1)	; TEST MOB
7734 FE40	14160		CP	40H	; TEST PRIMO BLOCCO
7736 2009	14170		JR	NZ, TLHWY	; TEST LIMITE STRADA
7738 3A7B6F	14180		LD	A, (OLDFRG)	
773B FEC0	14190		CP	0C0H	; PASSO DIV. SU LIM. STRADA
773D DB	14200		RET	C	
773E 34	14210		INC	(HL)	
773F 1B0A	14220		JR	DISSCR	
7741 FE50	14230	TLHWY	CP	50H	; TEST SU LIMITESTRADA
7743 C0	14240		RET	NZ	
7744 3A7B6F	14250		LD	A, (OLDFRG)	
7747 FE20	14260		CP	20H	
7749 D0	14270		RET	NC	; NESSUN PUNT.
774A 34	14280		INC	(HL)	
774B 0604	14290	DISSCR	LD	B, 4	; HL=>POS. DECINE
774D 7E	14300	ADDLOP	LD	A, (HL)	
774E FE3A	14310	CRYLOP	CP	3AH	; CICLO DI RIPORTO
7750 3B07	14320		JR	C, UPDDIG	; NUOVA CIFRA
7752 D60A	14330		SUB	10	
7754 2B	14340		DEC	HL	
7755 34	14350		INC	(HL)	; RIPORTO
7756 23	14360		INC	HL	
7757 1B5F	14370		JR	CRYLOP	
7759 77	14380	UPDDIG	LD	(HL), A	
775A 2B	14390		DEC	HL	
775B 10F0	14400		DJNZ	ADDLOP	
775D 21446F	14410		LD	HL, SCORE+1	
7760 CD6F77	14420		CALL	SCRIMG	; IMMAGINE PUNTI
7763 210640	14430		LD	HL, 4006H	
7766 11596F	14440		LD	DE, IMAGE	
7769 0605	14450		LD	B, 5	
776B CD2B73	14460		CALL	DISASC	
776E C9	14470		RET		

	14480 ;				
	14490 ;				
776F	11596F	14500	SCRIMG	LD	DE, IMAGE
7772	010500	14510		LD	BC, 5
7775	EDB0	14520		LDIR	
7777	21596F	14530		LD	HL, IMAGE
777A	013004	14540		LD	BC, 0430H
777D	79	14550	PREZER	LD	A, C
777E	BE	14560		CP	(HL)
777F	2005	14570		JR	NZ, PREZEX
7781	3620	14580		LD	(HL), 20H
7783	23	14590		INC	HL
7784	10F7	14600		DJNZ	PREZER
7786	C9	14610	PREZEX	RET	
		14620 ;			
		14630 ;			
7787	3EBF	14640	SIREN	LD	A, 0BFH
7789	DBFE	14650		IN	A, (OFEH)
778B	E601	14660		AND	1
778D	2009	14670		JR	NZ, NSOUND
778F	3A736F	14680		LD	A, (SOUNDF)
7792	3C	14690		INC	A
7793	E601	14700		AND	1
7795	32736F	14710		LD	(SOUNDF), A
7798	3A736F	14720	NSOUND	LD	A, (SOUNDF)
779B	A7	14730		AND	A
779C	2825	14740		JR	Z, DELAY
779E	3A726F	14750		LD	A, (CHASE)
77A1	A7	14760		AND	A
77A2	2B1F	14770		JR	Z, DELAY
77A4	3A746F	14780		LD	A, (TONFLG)
77A7	3C	14790		INC	A
77AB	E601	14800		AND	1
77AA	32746F	14810		LD	(TONFLG), A
77AD	210D6F	14820		LD	HL, PCTON1
77B0	2803	14830		JR	Z, TONE1
77B2	21116F	14840		LD	HL, PCTON2
77B5	5E	14850	TONE1	LD	E, (HL)
77B6	23	14860		INC	HL
77B7	56	14870		LD	D, (HL)
77B8	23	14880		INC	HL
77B9	4E	14890		LD	C, (HL)
77BA	23	14900		INC	HL
77BB	46	14910		LD	B, (HL)
77BC	C5	14920		PUSH	BC
77BD	E1	14930		POP	HL
77BE	CDB503	14940		CALL	03B5H
77C1	F3	14950		DI	
77C2	C9	14960		RET	
77C3	01001B	14970	DELAY	LD	BC, 6144
77C6	0B	14980	WAIT	DEC	BC
77C7	7B	14990		LD	A, B
77C8	B1	15000		OR	C
77C9	20FB	15010		JR	NZ, WAIT
77CB	C9	15020		RET	
		15030 ;			

/ TEST SU 30H

/ SPAZIO PIENO

/ AZZ. COND. BUONO

/ SE POLIZIA SELEZ.

/ DE=DURATA&FREQ.

/ HL=437500/FREQ=30.125

/ 03B5H DISAB. INTERR.


```

15040 ;
77CC E5 15050 RANDNO PUSH HL
77CD C5 15060 PUSH BC
77CE 2A756F 15070 LD HL, (RND)
77D1 46 15080 LD B, (HL)
77D2 23 15090 INC HL
77D3 3E3F 15100 LD A,3FH ;PUNT A ROM
77D5 A4 15110 AND H
77D6 67 15120 LD H,A
77D7 78 15130 LD A,B
77D8 22756F 15140 LD (RND),HL
77DB C1 15150 POP BC
77DC E1 15160 POP HL
77DD C9 15170 RET

15180 ;
15190 ;
77DE 21446F 15200 OVER LD HL,SCORE+1 ;PT. MAX.
77E1 11546F 15210 LD DE,HISCR
77E4 0605 15220 LD B,5
77E6 1A 15230 SORTLP LD A,(DE)
77E7 BE 15240 CP (HL)
77E8 2803 15250 JR Z,SAMSCR ;RICER. PRIMO BIT DIV.
77EA D0 15260 RET NC
77EB 1805 15270 JR SCRGT ;AGG.PT.MAX.
77ED 13 15280 SAMSCR INC DE
77EE 23 15290 INC HL
77EF 10F5 15300 DJNZ SORTLP
77F1 C9 15310 RET
77F2 21446F 15320 SCRGT LD HL,SCORE+1
77F5 11546F 15330 LD DE,HISCR
77F8 010500 15340 LD BC,5
77FB EDB0 15350 LDIR
77FD C9 15360 RET
15370 ;
15380 ;
77FE 3E38 15390 FINAL LD A,56 ;SEL. BORDO BIANCO
7800 32485C 15400 LD (23624),A
7803 210040 15410 LD HL,4000H ;INIZIO VIDEO
7806 110140 15420 LD DE,4001H
7809 01FF17 15430 LD BC,6143 ;DIM. VIDEO
780C 3600 15440 LD (HL),0
780E EDB0 15450 LDIR
7810 210058 15460 LD HL,5800H ;INIZIO FILE ATTR.
7813 110158 15470 LD DE,5801H
7816 01FF02 15480 LD BC,767
7819 3638 15490 LD (HL),56 ;PAPER BIANCO,INK NERO
781B EDB0 15500 LDIR
781D C9 15510 RET
15520 ;
15530 ;
6978 15540 END START
00000 Total errors

```

APPENDICE A
TABELLA DEI TASTI DELLO SPECTRUM

Valore di
INPUT per
A da OFE

	D4	D3	D2	D1	D0
OFEH	V	C	X	Z	CAP SHIFT
OFDH	G	F	D	S	A
OFBH	7	R	E	W	Q
OF7H	5	4	3	2	1
DEFH	6	7	8	9	0
ODFH	Y	U	I	0	P
OBFH	H	J	K	L	ENTER
O7FH	B	N	M	SYM SHIFT	BREAK SPACE

	↓	↓	↓	↓	↓
X	16	8	4	2	1

NB: Per controllare un tasto

1. Caricare in A il valore di INPUT della riga corrispondente.
LD A, 07FH ;ultima riga
2. Preparare la porta OFEH per l'acquisizione dati
IN A, (OFEH)
3. Test su Dx per vedere se il tasto e' allo stato basso.
AND 1 ;controlla il tasto BREAK/SPACE
4. Se zero, il tasto e' allo stato alto.
JR Z,tasto allo stato logico 1;lo stato normale
;e' alto

APPENDICE B

MEMORY ATTRIBUTE		LINE	MEMORY ATTRIBUTE	
IN HEX	IN HEX		IN HEX	IN HEX
4000	5800	0	401F	581F
4020	5820	1	403F	583F
4040	5840	2	405F	585F
4060	5860	3	407F	587F
4080	5880	4	409F	589F
40A0	58A0	5	40BF	58BF
40C0	58C0	6	40DF	58DF
40E0	58E0	7	40FF	58FF
4800	5900	8	481F	591F
4820	5920	9	483F	593F
4840	5940	10	485F	595F
4860	5960	11	487F	597F
4880	5980	12	489F	599F
48A0	59A0	13	48BF	59BF
48C0	59C0	14	48DF	59DF
48E0	59E0	15	48FF	59FF
5000	5A00	16	501F	5A1F
5020	5A20	17	503F	5A3F
5040	5A40	18	505F	5A5F
5060	5A60	19	507F	5A7F
5080	5A80	20	509F	5A9F
50A0	5AA0	21	50BF	5ABF
50C0	5AC0	22	50DF	5ADF
50E0	5AE0	23	50FF	5AFF

APPENDICE C

TABELLA DEI CARATTERI DELLO SPECTRUM

HEX LOB	HOB BITS	0	1	2	3	4	5	6	7
0	0000	000	001	010	011	100	101	110	111
/	0001	NU	INK ctrl	SPACE	0	@	P	f	p
2	0010	NU	PAPER ctrl	!	1	A	Q	a	q
3	0011	NU	FLASH ctrl	"	2	B	R	b	r
4	0100	NU	BRIGHT ctrl	#	3	C	S	c	s
5	0101	Nu	INVERSE ctrl	\$	4	D	T	d	t
6	0110	PRINT	OVER ctrl	%	5	E	U	e	u
7	0111	EDIT	AT ctrl	&	6	F	V	f	v
8	1000	cursor left	TAB ctrl	'	7	G	W	g	w
9	1001	cursor right	NU	(8	H	X	h	x
A	1010	cursor down	NU)	9	I	Y	i	y
B	1011	cursor up	NU	*	:	J	Z	j	z
C	1100	DELETE	NU	+	;	K	[k	{
D	1101	ENTER	NU	,	<	L	/	l	
E	1110	number	NU	-	=	M]	m	}
F	1111	NU	NU	/	>	N	↑	n	~
					?	O	—	o	☺

NON PRINTABLE * PRINTABLE

NB: NU = Not Used.

APPENDICE D

TABELLA DI CONVERSIONE DECIMALE-ESADECIMALE

HEX	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	00XX	XX00
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	0
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	256	4096
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	512	8192
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	768	12288
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	1024	16384
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	1280	20480
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	1536	24576
7	112	113	114	115	116	117	118	119	120	121	122	123	124	135	126	127	1792	28672
8	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	2048	32768
9	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	2304	36864
A	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	2560	40960
B	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	2816	45056
C	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	3072	49152
D	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	3328	53248
E	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	3584	57344
F	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	3840	61440

Appendice D

Vogliamo mostrarvi con un esempio come si lavora con questa tabella.

Supponiamo di voler trovare il codice esadecimale equivalente al valore 6200.

Dobbiamo determinare un valore a 16 bit in cui i primi otto bit formino la parte alta e gli altri la parte bassa del numero:

bbbbbbbb	bbbbbbbb
HOB	LOB

1. Confrontando il valore decimale da convertire con quello contenuto nell'ultima colonna della tabella (xx00) vediamo che 6200 e' compreso tra 4096 e 8192. Scegliamo il valore piu' basso, e cioe' 4096, ed estraiamo dalla riga corrispondente sulla prima colonna il valore dei primi quattro bit della parte alta: nel nostro caso 1.

0001bbbb	bbbbbbbb
HOB	LOB

2. Il secondo passo consiste nel determinare gli altri 4 bit di HOB. Calcoliamo la differenza tra 6200 e 4096 e troviamo 2104. Dato che essa e' maggiore di 255, ricerchiamo nella penultima colonna (00xx) tra quali valori essa e' compresa, trovando 2048 e 2304. Ancora una volta prendiamo il piu' piccolo e nella prima colonna troviamo che il valore corrispondente e' 8, cioe' 1000 binario. Si ottiene cosi' per HOB la configurazione

00011000	bbbbbbbb
HOB	LOB

3. Il terzo passo consiste nella ricerca del valore di LOB. Osserviamo che la differenza tra 2104 e 2048 e' 56. Cerchiamo nella parte interna della tabella questo valore e vediamo che esso corrisponde alla terza riga ed alla ottava colonna. Il valore esadecimale di LOB risulta cosi' 38H.

00011000	00111000
HOB	LOB

Il valore esadecimale di 6200 e' dunque 1838H.

APPENDICE E

TABELLA DI CONVERSIONE IN COMPLEMENTO A DUE

HEX	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
8	-128	-127	-126	-125	-124	-123	-122	-121	-120	-119	-118	-117	-116	-115	-114	-113
9	-112	-111	-110	-109	-108	-107	-106	-105	-104	-103	-102	-101	-100	-99	-98	-97
A	-96	-95	-94	-93	-92	-91	-90	-89	-88	-87	-86	-85	-84	-83	-82	-81
B	-80	-79	-78	-77	-76	-75	-74	-73	-72	-71	-70	-69	-68	-67	-66	-65
C	-64	-63	-62	-61	-60	-59	-58	-57	-56	-55	-54	-53	-52	-51	-50	-49
D	-48	-47	-46	-45	-44	-43	-42	-41	-40	-39	-38	-37	-36	-35	-34	-33
E	-32	-31	-30	-29	-28	-27	-26	-25	-24	-23	-22	-21	-20	-19	-18	-17
F	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

APPENDICE F

TAVOLA DI ADDIZIONE ESADECIMALE

+	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10
2	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11
3	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12
4	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13
5	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14
6	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15
7	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16
8	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17
9	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18
A	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19
B	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A
C	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B
D	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C
E	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
F	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E

APPENDICE G

TABELLA RIASSUNTIVA DELLE OPERAZIONI SUI FLAG

ISTRUZIONE	C	Z	P/V	S	N	H	COMMENTI
ADC HL, SS	#	#	V	#	0	X	Add. a 16 bit con riporto
ADX s; ADD s	#	#	V	#	0	#	Add. a 8 bit con o senza riporto
ADD DL, SS	#	-	-	-	0	X	Add. a 16 bit
AND s	0	#	P	0	#	1	Operazione logica
BIT b, s	-	#	X	X	0	1	Stato del bit b della locazione s (nel flag Z)
CCF	#	-	-	-	0	X	Operazione di riporto
CPD; CPDR; CPI; CPIR	-	#	#	X	1	X	Ricerca in un blocco Z=1 se A=(HL) se no Z=0 P/V=1 se BC<>0 se no P/V=0
CP s	#	#	V	#	1	#	Confronto con l'accumul.
CPL	-	-	-	-	1	1	Complemento accumulatore
DAA	#	#	P	#	-	#	Accum. in forma decimale
DEC s	-	#	V	#	1	#	Decremento a 8 bit
IN r, (C)	-	#	P	#	0	0	Input indiretto
INC s	-	#	V	#	0	#	Incremento a 8 bit
IND; INI	-	#	X	X	1	X	Blocco di input: Z=0 se B<>0 se no Z=1
INDR; INIR	-	1	X	X	1	X	Blocco di input: Z=0 se B<>0 se no Z=1
LD A, I; LD A, R	-	#	IFF	#	0	0	Contenuto dell'interruzione disabilitata (nel flag P/V)
LDD; LDI	-	X	#	X	0	0	Istr. transf. blocchi
LDDR; LDIR	-	X	0	X	0	0	P/V=1 se BC<>0 se no P/V=0
NEG	#	#	V	#	1	#	Accumulatore negato
OR s	0	#	P	#	0	0	Operazione logica
OTDR; OTIR	-	1	X	X	1	X	Blocco di output: Z=0 se B<>0 altrimenti Z=1
OUTD; OUTI	-	#	X	X	1	X	Blocco di output
RLA; RLCA; RRA; RRCA	#	-	-	-	0	0	Operazione di rotate
RLD; RRD	-	#	P	#	0	/	Rotate da sin. a destra
RLS; RLC s; RR s; RRC s SLA s; SRA s; SRL s	#	#	P	#	0	0	Rotate and shift
SBC HL, SS	#	#	V	#	1	X	Sottrazione con riporto su 16 bit
SCF	1	-	-	-	0	0	Carry settato
SBC s; SUB s			V		1		Sottrazione con riporto su 8 bit
XOR x	0		P		0	0	OR esclusivo

SIMBOLO	OPERAZIONE
C	Flag di carry. C=1 se l'ultima operazione eseguita ha prodotto un riporto dal bit piu' significativo dell'operando o del risultato.
Z	Flag di Zero. Z=1 se il risultato dell'ultima operazione e' stato 0.
S	Flag di Segno. S=1 se il bit piu' significativo del risultato e' 1, cioe' se e' un numero negativo.
P/V	Flag di Parita' (P)/Overflow (O). Le operazioni logiche influenzano questo flag con la parita' del risultato, mentre le operazioni aritmetiche lo influenzano come overflow. Se P/V ricorda la parita': P/V=1 quando il numero di bit settati nel risultato e' pari, 0 se e' dispari. Se P/V ricorda l'overflow, P/V=1 quando il risultato dell'operazione e' andato in overflow.
H	Flag di Half-carry. H=1 se l'operazione di addizione o sottrazione produce riporto (positivo o negativo) sul bit numero 4 dell'accumulatore.
N	Flag di Sottrazione. N=1 se l'operazione eseguita e' una sottrazione.
<p>I flag H e N sono in genere usati con le istruzioni di aggiornamento (DAA), che servono per fornire in BCD il corretto risultato decimale di una addizione o sottrazione.</p>	
#	Il flag e' influenzato dall'operazione ed il suo valore dipende dal risultato.
-	Il flag non e' influenzato dall'operazione.
0	Il flag e' posto a 0 dopo l'operazione.
1	Il flag e' posto a 1 dopo l'operazione.
X	Il valore del flag non e' prevedibile.
V	Il flag P/V e' considerato di overflow.
P	Il flag P/V e' considerato di parita'.
r	Un qualsiasi registro della CPU: A, B, C, D, E, H, L.
s	Una qualsiasi locazione di memoria ad 8 bit, indirizzata in uno qualsiasi dei modi premessi.
SS	Una qualsiasi coppia di locazioni (16 bit) indirizzata in uno dei modi premessi.
R	Registro di refresh.
n	Valore ad 8 bit compreso tra 0 e 255.
nn	Valore a 16 bit compreso tra 0 e 65535.

APPENDICE H
ISTRUZIONI DELLO Z80 ORDINATE PER CODICE ABBONICO

MNEMONIC	HEXADECIMAL	MNEMONIC	HEXADECIMAL	MNEMONIC	HEXADECIMAL
ADC A, (HL)	8E	BIT 2,B	CB 50	CP n	FE XX
ADC A, (IX+dis)	DD 8E XX	BIT 2,C	CB 51	CP E	8B
ADC A, (IY+dis)	FD 8E xx	BIT 2,D	CB 52	CP H	8C
ADC A,A	8F	BIT 2,E	CB 53	CP L	8D
ADC A,B	88	BIT 2,H	CB 54	CPD	ED A9
ADC A,C	89	BIT 2,L	CB 55	CPDR	ED B9
ADC A,D	8A	BIT 3,(HL)	CB 5E	CPI	ED A1
ADC A,n	CE XX	BIT 3,(IX+dis)	DD CB XX 5E	CPIR	ED B1
ADC A,E	8E	BIT 3,(IY+dis)	FD CB XX 5E	CPA	2F
ADC A,H	8C	BIT 3,A	CB 5F	DAA	27
ADC A,L	8D	BIT 3,B	CB 58	DEC (HL)	35
ADC HL,BC	ED 4A	BIT 3,C	CB 59	DEC (IX+dis)	DD 35 XX
ADC HL,DE	ED 5A	BIT 3,D	CB 5A	DEC (IY+dis)	FD 35 XX
ADC HL,HL	ED 6A	BIT 3,E	CB 5B	DEC A	3D
ADC HL,SP	ED 7A	BIT 3,H	CB 5C	DEC B	05
ADD A, (HL)	86	BIT 3,L	CB 5D	DEC BC	0B
ADD A, (IX+dis)	DD 86XX	BIT 4,(HL)	CB 66	DECC	08
ADD A, (IY+dis)	FD 86XX	BIT 4,(IX+dis)	DD CB XX 66	DECD	15
ADD A,A	87	BIT 4,(IY+dis)	FD CB XX 66	DEC DE	1B
ADD A,B	80	BIT 4,A	CB 67	DEC E	1D
ADD A,C	81	BIT 4,B	CB 60	DEC H	25
ADD A,D	82	BIT 4,C	CB 61	DEC HL	2B
ADD A,n	C6 XX	BIT 4,D	CB 62	DEC IX	DD 2B
ADD A,E	83	BIT 4,E	CB 63	DEC IY	FD 2B
ADD A,H	84	BIT 4,H	CB 64	DEC L	2D
ADD A,L	85	BIT 4,L	CB 65	DEC SP	3B
ADD HL,BC	09	BIT 5,(HL)	CB 6E	DI	F3
ADD HL,DE	19	BIT 5,(IX+dis)	DD CB XX 6E	DJNZ,dis	10 XX
ADD HL,HL	29	BIT 5,(IY+dis)	FD CB XX 6E	EI	FB
ADD HL,SP	39	BIT 5,A	CB 6F	EX (SP),HL	E3
ADD IX,BC	DD 09	BIT 5,B	CB 68	EX (SP),IX	DD E3
ADD IX,DE	DD 19	BIT 5,C	CB 65	EX (SP),IY	FD E3
ADD IX,IX	DD 29	BIT 5,D	CB 6A	EX AF,AF'	0B
ADD IX,SP	DD 39	BIT 5,E	CB 6B	EX DE,HL	E8
ADD IY,BC	FD 09	BIT 5,H	CB 6C	EXX	D9
ADD IY,DE	FD 19	BIT 5,L	CB 6D	HALT	76
ADD IY,IY	FD 29	BIT 6,(HL)	CB 76	IM 0	ED 46
ADD IY,SP	FD 39	BIT 6,(IX+dis)	DD CB XX 76	IM 1	ED 56
AND (HL)	A6	BIT 6,(IY+dis)	FD CB XX 76	IM 2	ED 5E
AND (IX+dis)	DD A6 XX	BIT 6,A	CB 77	IN A, (C)	ED 78
AND (IY+dis)	FD A6 XX	BIT 6,B	CB 70	IN A,port	DB XX
AND A	A7	BIT 6,C	CB 71	IN B, (C)	ED 40
AND B	A8	BIT 6,D	CB 72	IN C, (C)	ED 48
AND C	A9	BIT 6,E	CB 73	IN D, (C)	ED 50
AND D	AA	BIT 6,H	CB 74	IN E, (C)	ED 58
AND n	E6 XX	BIT 6,L	CB 75	IN H, (C)	ED 60
AND E	AB	BIT 7,(HL)	CB 7E	IN L, (C)	ED 68
AND H	A4	BIT 7,(IX+dis)	DD CB XX 7E	INC (HL)	34
AND L	A5	BIT 7,(IY+dis)	FD CB XX 7E	INC (IX+dis)	DD 34 XX
BIT 0,(HL)	CB 46	BIT 7,A	CB 7F	INC (IY+dis)	FD 34 XX
BIT 0,(IX+dis)	DD CB XX 46	BIT 7,B	CB 78	INC A	34
BIT 0,(IY+dis)	FD CB XX 46	BIT 7,C	CB 79	INC B	0C
BIT 0,A	CB 47	BIT 7,D	CB 7A	INC BC	03
BIT 0,B	CB 40	BIT 7,E	CB 7B	INC C	0C
BIT 0,C	CB 41	BIT 7,H	CB 7C	INC D	14
BIT 0,D	CB 42	BIT 7,L	CB 7D	INC DE	13
BIT 0,E	CB 43	CALL ADDR	CD XX XX	INC E	1C
BIT 0,H	CB 44	CALL C,ADDR	DC XX XX	INC H	24
BIT 0,L	CB 45	CALL M,ADDR	FC XX XX	INC HL	23
BIT 1,(HL)	CB 4E	CALL NC,ADDR	D4 XX XX	INC IX	DD 23
BIT 1,(IX+dis)	DD CB XX 4E	CALL NZ,ADDR	C4 XX XX	INC IY	FD 23
BIT 1,(IY+dis)	FD CB XX 4E	CALL P,ADDR	F4 XX XX	INC L	2C
BIT 1,A	CB 4F	CALL PE,ADDR	EC XX XX	INC SP	33
BIT 1,B	CB 48	CALL PO,ADDR	E4 XX XX	IND	ED AA
BIT 1,C	CB 49	CALL Z,ADDR	CC XX XX	INCR	ED BA
BIT 1,D	CB 4A	CCF	3F	INI	ED A2
BIT 1,E	CB 4B	CP (HL)	BE	INIR	ED B2
BIT 1,H	CB 4C	CP (IX+dis)	DD BE XX	JP (HL)	E9
BIT 1,L	CB 4D	CP (IY+dis)	FD BE XX	JP (IX)	ED E9
BIT 2,(HL)	CB 56	CP A	BF	JP (IY)	FD E9
BIT 2,(IX+dis)	DD CB XX 56	CP B	B8	JP ADDR	C3 XX XX
BIT 2,(IY+dis)	FD CB XX 56	CP C	B9	JP C,ADDR	DA XX XX
BIT 2,A	CB 57	CP D	BA	JP M,ADDR	FA XX XX

MNEMONIC	HEXADECIMAL	MNEMONIC	HEXADECIMAL	MNEMONIC	HEXADECIMAL
JP NC,ADDR	D2 XX XX	LD BC,nn	01 XX XX	LDDR	ED B8
JP NZ,ADDR	C2 XX XX	LD C, (HL)	4E	LDI	ED A0
JP P,ADDR	F2 XX XX	LD C, (IX+dis)	DD 4E xx	LDIR	ED B0
JP PE,ADDR	EA XX XX	LD C, (IY+dis)	FD 4E XX	NEG	ED 44
JP PO,ADDR	E2 XX XX	LD C,A	4F	NOP	00
JP Z,ADDR	CA XX XX	LD C,B	48	OR (HL)	B6
JR C,dis	38 XX	LD C,C	49	OR (IX+dis)	DD B6 XX
JR dis	18 XX	LD C,D	4A	OR (IY+dis)	FD B6 xx
JR NC,dis	30 XX	LD C,n	0E XX	OR A	B7
JR NZ,dis	20 XX	LD C,E	4B	OR B	B0
JR Z,dis	28 XX	LD C,H	4C	OR C	B1
LD (ADDR) ,A	32 XX XX	LD C,L	4D	OR D	B2
LD (ADDR) ,BC	ED 43 XX XX	LD D, (HL)	56	OR n	F6 XX
LD (ADDR) ,DE	ED 53 XX XX	LD D, (IX+dis)	DD 56 XX	OR E	B3
LD (ADDR) ,HL	ED 63 XX XX	LD D, (IY+dis)	FD 56 XX	OR H	B4
LD (ADDR) ,HL	22 XX XX	LD D,A	57	OR L	B5
LD (ADDR) ,IX	DD 22 XX XX	LD D,B	50	OTDR	ED B8
LD (ADDR) ,IY	FD 22 XX XX	LD D,C	51	OTIR	ED B3
LD (ADDR) ,SP	ED 73 XX XX	LD D,D	52	OUT (C) ,A	ED 79
LD (BC) ,A	02	LD D,n	16 XX	OUT (C) ,B	ED 41
LD (DE) ,A	12	LD D,E	53	OUT (C) ,C	ED 49
LD (HL) ,A	77	LD D,H	54	OUT (C) ,D	ED 51
LD (HL) ,B	70	LD D,L	55	OUT (C) ,E	ED 59
LD (HL) ,C	71	LD DE, (ADDR)	ED 5B XX XX	OUT (C) ,H	ED 61
LD (HL) ,D	72	LD DE,nn	11 XX XX	OUT (C) ,L	ED 69
LD (HL) ,n	36 XX	LD E, (HL)	5E	OUT part,A	D3 port
LD (HL) ,E	73	LD E, (IX+dis)	DD 5E XX	OUTD	ED AB
LD (HL) ,H	74	LD E, (IY+dis)	FD 5E XX	OUTI	ED A3
LD (HL) ,L	75	LD E,A	5F	POP AF	F1
LD (IX+dis) ,A	DD 77 XX	LD E,B	58	POP BC	C1
LD (IX+dis) ,B	DD 70 XX	LD E,C	59	POP DE	D1
LD (IX+dis) ,C	DD 71 XX	LD E,D	5A	POP HL	E1
LD (IX+dis) ,D	DD 72 XX	LD E,n	1E XX	POP IX	DD E1
LD (IX+dis) ,n	DD 36 XX XX	LD E,E	5B	POP IY	FD E1
LD (IX+dis) ,E	DD 73 XX	LD E,H	5C	PUSH AF	F5
LD (IX+dis) ,H	DD 74 XX	LD E,L	5D	PUSH BC	C5
LD (IX+dis) ,L	DD 75 XX	LD H, (HL)	66	PUSH DE	D5
LD (IY+dis) ,A	FD 77 XX	LD H, (IX+dis)	DD 66 XX	PUSH HL	E5
LD (IY+dis) ,B	FD 70 XX	LD H, (IY+dis)	FD 66 XX	PUSH IX	DD E5
LD (IY+dis) ,C	FD 71 XX	LD H,A	67	PUSH IY	FD E5
LD (IY+dis) ,D	FD 72 XX	LD H,B	60	RES 0, (HL)	CB 86
LD (IY+dis) ,n	FD 36 XX XX	LD H,C	61	RES 0, (IX+dis)	DD CB XX 86
LD (IY+dis) ,E	FD 73 XX	LD H,D	62	RES 0, (IY+dis)	FD CB XX 86
LD (IY+dis) ,H	FD 74 XX	LD H,n	26 XX	RES 0,A	CB 87
LD (IY+dis) ,L	FD 75 XX	LD H,E	63	RES 0,B	CB 80
LD A, (ADDR)	3A XX XX	LD H,H	64	RES 0,C	CB 81
LD A, (BC)	0A	LD H,L	65	RES 0,D	CB 82
LD A, (DE)	1A	LD HL, (ADDR)	ED 68 XX XX	RES 0,E	CB 83
LD A, (HL)	7E	LD HL, (ADDR)	2A XX XX	RES 0,H	CB 84
LD A, (IX+dis)	DD 7E XX	LD HL,nn	21 XX XX	RES 0,L	CB 85
LD A, (IY+dis)	FD 7E XX	LD I,A	ED 47	RES 1, (HL)	CB 8E
LD A,A	7F	LD IX, (ADDR)	DD 2A XX XX	RES 1, (IX+dis)	DD CB XX 8E
LD A,B	78	LD IX,nn	DD 21 XX XX	RES 1, (IY+dis)	FD CB XX 8E
LD A,C	79	LD IY, (ADDR)	FD 2A XX XX	RES 1,A	CB 8F
LD A,D	7A	LD IY,nn	FD 21 XX XX	RES 1,B	CB 88
LD A,n	3E XX	LD L,A	6F	RES 1,C	CB 89
LD A,E	7B	LD L,B	68	RES 1,D	CB 8A
LD A,H	7C	LD L,C	69	RES 1,E	CB 8B
LD A,I	ED 57	LD L,D	6A	RES 1,H	CB 8C
LD A,L	7D	LD L,n	2E XX	RES 1,L	CB 8D
LD A,R	ED 5F	LD L,E	6B	RES 2, (HL)	CB 96
LD B, (HL)	46	LD L, (HL)	6E	RES 2, (IX+dis)	DD CB XX 96
LD B, (IX+dis)	DD 46 XX	LD L, (IX+dis)	DD 6E XX	RES 2, (IY+dis)	FD CB XX 96
LD B, (IY+dis)	FD 46 XX	LD L, (IY+dis)	FD 6E XX	RES 2,A	CB 97
LD B,A	47	LD L,H	6C	RES 2,B	CB 90
LD B,B	40	LD L,L	6D	RES 2,C	CB 91
LD B,C	41	LD R,A	ED 4F	RES 2,D	CB 92
LD B,D	42	LD SP, (ADDR)	ED 7B XX XX	RES 2,E	CB 93
LD B,n	06 XX	LD SP,nn	31 XX XX	RES 2,H	CB 94
LD B,E	43	LD SP,HL	F9	RES 2,L	CB 95
LD B,H	44	LD SP,IX	DD F9	RES 3, (HL)	CB 9E
LD B,L	45	LD SP,IY	FD F9	RES 3, (IX+dis)	DD CB XX 9E
LD BC, (ADDR)	ED 4B XX XX	LDD	ED A8	RES 3, (IY+dis)	FD CB XX 9E
				RES 3,A	CB 9F

MNEMONIC	HEXADECIMAL	MNEMONIC	HEXADECIMAL	MNEMONIC	HEXADECIMAL
RES 3,B	CB 98	RLC C	CB 01	SET 1,L	CB CD
RES 3,C	CB 99	RLC D	CB 02	SET 2, (HL)	CB DE
RES 3,D	CB 9A	RLC E	CB 03	SET 2, (IX+dis)	DD CB XX D6
RES 3,E	CB 9B	RLC H	CB 04	SET 2, (IY+dis)	FD CB XX D6
RES 3,H	CB 9C	RLC L	CB 05	SET 2,A	CB D7
RES 3,L	CB 9D	RLCA	07	SET 2,B	CB D0
RES 4, (HL)	CB A6	RLD	ED 6F	SET 2,C	CB D1
RES 4, (IX+dis)	DD CB XX A6	RR (HL)	CB 1E	SET 2,D	CB D2
RES 4, (IY+dis)	FD CB XX A6	RR (IX+dis)	DD CB XX 1E	SET 2,E	CB D3
RES 4,A	CB A7	RR (IY+dis)	FD CB XX 1E	SET 2,H	CB D4
RES 4,B	CB A0	RR A	CB 1F	SET 2,L	CB D5
RES 4,C	CB A1	RR B	CB 18	SET 3, (HL)	CB DE
RES 4,D	CB A2	RR C	CB 19	SET 3, (IX+dis)	DD CB XX DE
RES 4,E	CB A3	RR D	CB 1A	SET 3, (IY+dis)	FD CB XX DE
RES 4,H	CB A4	RR E	CB 1B	SET 3,A	CB DF
RES 4,L	CB A5	RR H	CB 1C	SET 3,B	CB D8
RES 5 (HL)	CB AE	RR L	CB 1D	SET 3,C	CB D9
RES 5, (IX+dis)	DD CB XX AE	RR A	1F	SET 3,D	CB DA
RES 5, (IY+dis)	FD CB XX AE	RRC (HL)	CB 0E	SET 3,E	CB DB
RES 5,A	CB AF	RRC (IX+dis)	DD CB XX 0E	SET 3,H	CB DC
RES 5,B	CB A8	RRC (IY+dis)	FD CB XX 0E	SET 3,L	CB DD
RES 5,C	CB A9	RRC A	CB 0F	SET 4, (HL)	CBE6
RES 5,D	CB AA	RRC B	CB 08	SET 4, (IX+dis)	DD CB XX E6
RES 5,E	CB AB	RRC C	CB 09	SET 4, (IY+dis)	FD CB XX E6
RES 5,H	CB AC	RRC D	CB 0A	SET 4,A	CB E7
RES 5,L	CB AD	RRC E	CB 0B	SET 4,B	CB E0
RES 6, (HL)	CB B6	RRC H	CB 0C	SET 4,C	CB E1
RES 6, (IX+dis)	DD CB XX B6	RRC L	CB 0D	SET 4,D	CB E2
RES 6, (IY+dis)	FD CB XX B6	RRCA	0F	SET 4,E	CB E3
RES 6,A	CB B7	RRD	ED 67	SET 4,H	CB E4
RES 6,B	CB B0	RST 00	C7	SET 4,L	CB E5
RES 6,C	CB B1	RST 08	CF	SET 5, (HL)	CB EE
RES 6,D	CB B2	RST 10	D7	SET 5, (IX+dis)	DD CB XX EE
RES 6,E	CB B3	RST 18	DF	SET 5, (IY+dis)	FD CB XX EE
RES 6,H	CB B4	RST 20	E7	SET 5,A	CB EF
RES 6,L	CB B5	RST 28	EF	SET 5,B	CB E8
RES 7, (HL)	CB BE	RST 30	F7	SET 5,C	CB E9
RES 7, (IX+dis)	DD CB XX BE	RST 38	FF	SET 5,D	CB EA
RES 7, (IY+dis)	FD CB XX BE	SBC A, (HL)	9E	SET 5,E	CB EB
RES 7,A	CB BF	SBC A, (IX+dis)	DD 9E XX	SET 5,H	CB EC
RES 7,B	CB B8	SBC A, (IY+dis)	FD 9E XX	SET 5,L	CB ED
RES 7,C	CB B9	SBC A,A	9F	SET 6, (HL)	CB FE
RES 7,D	CB BA	SBC A,B	98	SET 6, (IX+dis)	DD CB XX F6
RES 7,E	CB BB	SBC A,C	99	SET 6, (IY+dis)	FD CB XX F6
RES 7,H	CB BC	SBC A,D	9A	SET 6,A	CB F7
RES 7,L	CB BD	SBC A,n	DE XX	SET 6,B	CB F0
RET	C9	SBC A,E	9B	SET 6,C	CB F1
RET C	D8	SBC A,H	9C	SET 6,D	CB F2
RET M	F8	SBC A,L	9D	SET 6,E	CB F3
RET NC	D0	SBC HL,BC	ED 42	SET 6,H	CB F4
RET NZ	C0	SBC HL,DE	ED 52	SET 6,L	CB F5
RET P	F0	SBC HL,HL	ED 62	SET 7, (HL)	CB FE
RET PE	E8	SBC HL,SP	ED 72	SET 7, (IX+dis)	DD CB XX FE
RET PO	E0	SCF	37	SET 7, (IY+dis)	FD CB XX FE
RET Z	C8	SET 0, (HL)	CB C6	SET 7,A	CB FF
RETI	ED 4D	SET 0, (IX+dis)	DD CB XX C6	SET 7,B	CB F8
RETN	ED 45	SET 0, (IY+dis)	FD CB XX C6	SET 7,C	CB F9
RL (HL)	CB 16	SET 0,A	CB C7	SET 7,D	CB FA
RL (IX+dis)	DD CB XX 16	SET 0,B	CB C0	SET 7,E	CB FB
RL (IY+dis)	FD CB XX 16	SET 0,C	CB C1	SET 7,H	CB FC
RL A	CB 17	SET 0,D	CB C2	SET 7,L	CB FD
RL B	CB 10	SET 0,E	CB C3	SLA (HL)	CB 26
RL C	CB 11	SET 0,H	CB C4	SLA (IX+dis)	DD CB XX 26
RL D	CB 12	SET 0,L	CB C5	SLA (IY+dis)	FD CB XX 26
RL E	CB 13	SET 1, (HL)	CB CE	SLA A	CB 27
RL H	CB 14	SET 1, (IX+dis)	DD CB XX CE	SLA B	CB 20
RL L	CB 15	SET 1, (IY+dis)	FD CB XX CE	SLA C	CB 21
RLA	17	SET 1,A	CB CF	SLA D	CB 22
RLC (HL)	CB 06	SET 1,B	CB C8	SLA E	CB 23
RLC (IX+dis)	DD CB XX 06	SET 1,C	CB C9	SLA H	CB 24
RLC (IY+dis)	FD CB XX 06	SET 1,D	CB CA	SLA L	CB 25
RLC A	CB 07	SET 1,E	CB CB	SRA (HL)	CB 2E
RLC B	CB 00	SET 1,H	CB CC	SRA (IX+dis)	DD CB XX 2E

MNEMONIC	HEXADECIMAL	MNEMONIC	HEXADECIMAL	MNEMONIC	HEXADECIMAL
SRA (IY+dis)	FD CB XX 2E				
SRA A	CB 2F				
SRA B	CB 28				
SRA C	CB 29				
SRA D	CB 2A				
SRA E	CB 2B				
SRA H	CB 2C				
SRA L	CB 2D				
SRL (HL)	CB 3E				
SRL (IX+dis)	DD CB XX 3E				
SRL (IY+dis)	FD CB XX 3E				
SRL A	CB 3F				
SRL B	CB 38				
SRL C	CB 39				
SRL D	CB 3A				
SRL E	CB 3B				
SRL H	CB 3C				
SRL L	CB 3D				
SUB (HL)	96				
SUB (IX+dis)	DD 96 XX				
SUB (IY+dis)	FD 96 XX				
SUB A	97				
SUB B	90				
SUB C	91				
SUB D	92				
SUB E	D6 XX				
SUB n	93				
SUB H	94				
SUB L	95				
XOR (HL)	AE				
XOR (IX+dis)	DD AE XX				
XOR (IY+dis)	FD AE XX				
XOR A	AF				
XOR B	A9				
XOR C	A9				
XOR D	AA				
XOR n	EE XX				
XOR E	AB				
XSOR H	AC				
XOR L	AD				

APPENDICE I
ISTRUZIONI DELLO Z80 ORDINATE PER CODICE ESADECIMALE

HEXADECIMAL	MNEMONIC	HEXADECIMAL	MNEMONIC	HEXADECIMAL	MNEMONIC
00	NOP	49	LD C,C	92	SUB D
01 XXXX	LD BC,NN	4A	LD C,D	93	SUB E
02	LD (BC),A	4B	LD C,E	94	SUB H
03	INC BC	4C	LD C,H	95	SUB L
04	INC B	4D	LD C,L	96	SUB (HL)
05	DEC B	4E	LD C,(HL)	97	SUB A
06XX	LD B,N	4F	LD C,A	98	SBC A,B
07	RLCA	50	LD D,B	99	SBC A,C
08	EX AF, AF'	51	LD D,C	9A	SBC A,D
09	ADD HL,BC	52	LD D,D	9B	SBC A,E
0A	LD A, (BC)	53	LD D,E	9C	SBC A,H
0B	DEC BC	54	LD D,H	9D	SBC A,L
0C	INC C	55	LD D,L	9E	SBC A,(HL)
0D	DEC C	56	LD D,(HL)	9F	SBC A,A
0EXX	LD C,N	57	LD D,A	A0	AND B
0F	RRCA	58	LD E,B	A1	AND C
10XX	DJNZ DIS	59	LD E,C	A2	AND C
11XXXX	LD DE,NN	5A	LD E,D	A3	AND E
12	LD (DE),A	5B	LD E,E	A4	AND H
13	INC DE	5C	LD E,H	A5	AND L
14	INC D	5D	LD E,L	A6	AND (HL)
15	DEC D	5E	LD E,(HL)	A7	AND A
16XX	LD D,N	5F	LD E,A	A8	XOR B
17	RLA	60	LD H,B	A9	XOR C
18XX	JR DIS	61	LD H,C	AA	SOR D
19	ADD HL,DE	62	LD H,D	AB	XOR E
1A	LD A,(DE)	63	LD H,E	AC	SOR H
1B	DEC DE	64	LD H,H	AD	SOR L
1C	INC E	65	LD H,L	AE	XOR (HL)
1D	DEC E	66	LD H,(HL)	AF	XOR A
1EXX	LD E,N	67	LD H,A	B0	OR B
1F	RRA	68	LD L,B	B1	OR C
20XX	JR NZ,DIS	69	LD L,C	B2	OR D
21XXXX	LD HL,NN	6A	LD L,D	B3	OR E
22XXXX	LD (NN),HL	6B	LD L,E	B4	OR H
23	INC HL	6C	LD L,H	B5	OR L
24	INC H	6D	LD L,L	B6	OR (HL)
25	DEC H	6E	LD L,(HL)	B7	OR A
26XX	LD H,N	6F	LD L,A	B8	CP B
27	DAA	70	LD (HL),B	B9	CP C
28XX	JR Z,DIS	71	LD (HL),C	BA	CP D
29	ADD HL,HL	72	LD (HL),D	BB	CP E
2AXXXX	LD HL,(NN)	73	LD (HL),E	BC	CP H
2B	DEC HL	74	LD (HL),H	BD	CP L
2C	INC L	75	LD (HL),L	BE	CP (HL)
2D	DEC L	76	HALT	BF	CP A
2EXX	LD L,N	77	LD (HL),A	C0	RET NZ
2F	CPL	78	LD A,B	C1	POP BC
30XX	JR NC,DIS	79	LD A,C	C2XXXX	JP NZ,NM
31XXXX	LD SP,NN	7A	LD A,D	C3XXXX	JP NM
32XXXX	LD (NN),A	7B	LD A,E	C4XXXX	CALL NZ,NM
33	INC SP	7C	LD A,H	C5	PUSH BC
34	INC (HL)	7D	LD A,L	C6XX	ADD A,N
35	DEC (HL)	7E	LD A,(HL)	C7	RST 0
3620XX	LD (HL),N	7F	LD A,A	C8	RET Z
37	SCF	80	ADD A,B	C9	RET
38XX	JR C,DIS	81	ADD A,C	CAXXXX	JP Z,NM
39	ADD HL,SP	82	ADD A,D	CCXXXX	CALL Z,NN
3AXXXX	LD A,(NN)	83	ADD A,E	CDXXXX	CALL NN
3B	DEC SP	84	ADD A,H	CEXX	ADC A,N
3C	INC A	85	ADD A,L	CF	RST 8
3D	DEC A	86	ADD A,(HL)	D0	RET NC
3EXXXX	LD A	87	ADD A,A	D1	POP DE
3F	CCF	88	ADC A,B	D2XXXX	JP NC,NN
40	LD B,B	89	ADC A,C	D3XX	OUT (N),A
41	LD B,C	8A	ADC A,D	D4XXXX	CALL NC,NN
42	LD B,D	8B	ADC A,E	D5	PUSH DE
43	LD B,H	8C	ADC A,H	D6XX	SUB N
44	LD B,L	8D	ADC A,L	D7	RST 10H
45	LD B,(HL)	8E	ADC A,(HL)	D8	RET C
46	LD B,A	8F	ADC A,A	D9	EXX
47	LD B,A	90	SUB B	DAXXXX	JP C,NN
48	LD C,B	91	SUB C	DBXX	IN A,(N)

HEXADECIMAL	MNEMONIC	HEXADECIMAL	MNEMONIC	HEXADECIMAL	MNEMONIC
DCXXXX	CALL C,NN	CB28	SRA B	CB79	BIT 7,C
DEXX	SBC A,N	CB29	SRA C	CB7A	BIT 7,D
DF	RST 18H	CB2A	SRA D	CB7B	BIT 7,E
E0	RET P0	CB2B	SRA E	CB7C	BIT 7,H
E1	POP HL	CB2C	SRA H	CB7D	BIT 7,L
E2XXXX	JP P0,NN	CB2D	SRA L	CB7E	BIT 7,(HL)
E3	EX (SP),HL	CB2E	SRA (HL)	CB7F	BIT 7,A
E4XXXX	CALL P0,NN	CB2F	SRA A	CB80	RES 0,B
E5	PUSH HL	CB38	SRL B	CB81	RES 0,C
E6XX	AND N	CB39	SRL C	CB82	RES 0,D
E7	RST 20 H	CB3A	SRL D	CB83	RES 0,E
E8	RET PE	CB3B	SRL E	CB84	RES 0,H
E9	JP (HL)	CB3C	SRL H	CB85	RES 0,L
EAXXXX	JE PE NN	CB3D	SRL L	CB86	RES 0,(HL)
EB	EX DE,HL	CB3E	SRL (HL)	CB87	RES 0,A
ECXXXX	CALL PE,NN	CB3F	SRL A	CB88	RES 1,B
EEXX	XOR N	CB40	BIT 0,B	CB89	RES 1,C
EF	RST 28H	CB41	BIT 0,C	CB9A	RES 1,D
F0	RET P	CB42	BIT 0,D	CB8B	RES 1,E
F1	POP AF	CB43	BIT 0,E	CB8C	RES 1,H
F2XXXX	JR P,NN	CB44	bit 0,H	CB8D	RES 1,L
F3	D1	CB45	BIT 0,L	CB8E	RES 1,(HL)
F4XXXX	CALL P,NN	CB46	BIT 0,(HL)	CB8F	RES 1,A
F5	PUSH AF	CB47	BIT 0,A	CB90	RES 2,B
F620XX	OR N	CB48	Bit 1,B	CB91	RES 2,C
F7	RST 30H	CB49	BIT 1,C	CB92	RES 2,D
F8	RET N	CB4A	BIT 1,D	CB93	RES 2,E
F9	LD,SP,HL	CB4B	BIT 1,E	CB94	RES 2,H
FAXXXX	JP N,NN	CB4C	BIT 1,H	CB95	RES 2,L
FB	E1	CB4D	BIT 1,L	CB96	RES 2,(HL)
FCXXXX	CALL M,NN	CB4E	BIT 1,(HL)	CB97	RES 2,A
FE20XX	CP N	CB4F	BIT 1,A	CB98	RES 3,B
FF	RST 38H	CB50	BIT 2,B	CB99	RES 3,C
CB00	RLC B	CB51	BIT 2,C	CB9A	RES 3,D
CB01	RLC C	CB52	BIT 2,D	CB9B	RES 3,E
CB02	RLC D	CB53	BIT 2,E	CB9C	RES e,H
CB03	RLC E	CB54	BIT 2,H	CB9D	RES 3,L
CB04	RLC H	CB55	BIT 2,L	CB9E	RES 3,(HL)
CB05	RLC L	CB56	BIT 2,(HL)	CB9F	RES 3,A
CB06	RLC (HL)	CB57	BIT 2,A	CA0A	RES 4,B
CB07	RLC A	CB58	BIT 3,B	CA1	RES 4,C
CB08	RRC B	CB59	BIT 3,C	CA2	RES 4,D
CB09	RRC C	CB5A	BIT 3,D	CA3	RES e,E
CB0A	RRC D	CB5B	BIT 3,E	CA4	RES e,H
CB0B	RRC E	CB5C	BIT 3,H	CA5	RES 4,L
CB0C	RRC H	CB5D	BIT 3,L	CA6	RES 4,(HL)
CB0D	RRC L	CB5E	BIT 3,(HL)	CA7	RES 4,A
CB0E	RRC (HL)	CB5F	BIT 3,A	CA8	RES 5,B
CB0F	RRC A	CB60	BIT 4,B	CA9	RES 5,C
CB10	RL B	CB61	BIT 4,C	CAA	RES 5,D
CB11	RL C	CB62	BIT 4,D	CBAB	RES 5,E
CB12	RL D	CB63	BIT 4,E	CBAC	RES 5,H
CB13	RL E	CB64	BIT 4,H	CBAD	RES 5,L
CB14	RL H	CB65	BIT 4,L	CAE	RES 5,(HL)
CB15	RL L	CB66	BIT 4,(HL)	CBAF	RES 5,A
CB16	RL (HL)	CB67	BIT 4,A	CB80	RES 6,B
CB17	RL A	CB68	BIT 5,B	CB81	RES 6,C
CB18	RR B	CB69	BIT 5,C	CB82	RES 6,D
CB19	RR C	CB6A	BIT 5,D	CB83	RES 6,E
CB1A	RR D	CB6B	BIT 5,E	CB84	RES 6,H
CB1B	RR E	CB6C	BIT 5,H	CB85	RES 6,L
CB1C	RR H	CB6D	BIT 5,L	CB86	RES 6,(HL)
CB1D	RR L	CB6E	BIT 5,(HL)	CB87	RES 7,A
CB1E	RR (HL)	CB6F	BIT 5,A	CB88	RES 7,B
CB1F	RR A	CB70	BIT 6,B	CB89	RES 7,C
CB20	SLA B	CB71	BIT 6,C	CB8A	RES 7,D
CB21	SLA C	CB72	BIT 6,D	CB8B	RES 7,E
CB22	SKA D	CB73	BIT 6,E	CB8C	RES 7,H
CB23	SLA E	CB74	BIT 6,H	CB8D	RES 7,L
CB24	SLA H	CB75	BIT 6,L	CB8E	RES 7,(HL)
CB25	SLA L	CB76	BIT 6,(HL)	CB8F	RES 7,A
CB26	SLA (HL)	CB77	BIT 6,A	CB80	SET 0,B
CB27	SLA A	CB78	BIT 7,B	CB81	SET 0,C
				CB82	SET 0,D

HEXADECIMAL	MNEMONIC	HEXADECIMAL	MNEMONIC	HEXADECIMAL	MNEMONIC
CBC3	SET 0,E	DD4EXX	LD C,(IX+d)	ED56	IN 1
CBC4	SET 0,H	DD56XX	LD D,(IX+d)	ED57	LD A,1
CBC5	SET 0,L	DD5EXX	LD E,(IX+d)	ED58	IN E,(C)
CBC6	SET 0,(HL)	DD66XX	LD H,(IX+d)	ED59	OUT(C),E
CBC7	SET 0,A	DD6EXX	LD L,(IX+d)	ED5A	ADC HL,DE
CBC8	SET 1,B	DD70XX	LD (IX+d),B	ED5BXXXX	LD DE,(NN)
CBC9	SET 1,C	DD71XX	LD (IC+d),C	ED5E	IM 2
CBCA	SET 1,D	DD72XX	LD (IX+d),D	ED60	IN H,(C)
CBCB	SET 1,E	DD73XX	LD (IX+d),E	ED61	OUT(C),H
CBCC	SET 1,H	DD74XX	LD (IX+d),H	ED62	SBC HL,HL
CBCD	SET 1,L	DD75XX	LD (IX+d),L	ED67	RRD
CBCE	SET 1,(HL)	DD77XX	LD (IX+d),A	ED68	IN L,(C)
CBCF	SET 1,A	DD7EXX	LD A,(IX+d)	ED69	OUT(C),L
CBD0	SET 2,B	DD86XX	ADD A,(IX+d)	ED6A	ADC HL,HL
CBD1	SET 2,C	DD8EXX	ADC A,(IX+d)	ED6F	RLD
CBD2	SET 2,D	DD96XX	SUB(IX+d)	ED72	SBC HL,SP
CBD3	SET 2,E	DD9EXX	SBC A,(IX+d)	ED73XXXX	LD(NN),SP
CBD4	SET 2,H	DDA6XX	AND(IX+d)	ED78	IN A,(C)
CBD5	*SET 2,L	DDAEXX	XOR(IX+d)	ED79	OUT(C),A
CBD6	SET 2,(HL)	DDB6XX	OR(IX+d)	ED7A	ADC HL,SP
CBD7	SET 2,A	DDBEXX	CP(IX+d)	ED7BXXXX	LD SP,(NN)
CBD8	SET 3,B	DDE1	POP IX	EDA0	LDI
CBD9	SET 3,C	DDE3	EX(SP),IX	EDA1	CPI
CBD A	SET 3,D	DDE5	PUSH IX	EDA2	INI
CBD B	SET 3,E	DDE9	JP(IX)	EDA3	OUTI
CBD C	SET 3,H	DDF9	LD SP,IX	EDA8	LDD
CBD D	SET 3,L	DDCBXX06	RLC(IX+d)	EDA9	CP0
CBD E	SET 3,(HL)	DDCBXX0E	RRC(IX+d)	EDAA	IND
CBD F	SET 3,A	DDCBXX16	RL(IX+d)	EDAB	OUTD
CBE0	SET 4,B	DDCBXX1E	RR(IX+d)	ED80	LDIR
CBE1	SET 4,C	DDCBXX26	SLA(IX+d)	ED81	CPIR
CBE2	SET 4,D	DDCBXX2E	SRA(IX+d)	ED82	INIR
CBE3	SET 4,E	DDCBXX3E	SRL(IX+d)	ED83	OTIR
CBE4	SET 4,H	DDCBXX46	BIT 0,(IX+d)	ED88	LDDR
CBE5	SET 4,L	DDCBXX4E	BIT 1,(IX+d)	ED89	CPDR
CBE6	SET 4,(HL)	DDCBXX56	BIT 2,(IX+d)	ED8A	INDR
CBE7	SET 4,A	DDCBXX5E	BIT 3,(IX+d)	ED8B	OTDR
CBE8	SET 5,B	DDCBXX66	BIT 4,(IX+d)	ED09	ADD IV,BC
CBE9	SET 5,C	DDCBXX6E	BIT 5,(IX+d)	ED19	ADD IV,DC
CBEA	SET 5,D	DDCBXX76	BIT 6,(IX+d)	ED21XXXX	LD IV,NN
CBE B	SET 5,E	DDCBXX7E	BIT 7,(IX+d)	FD22XXXX	LD(NN),IV
CBE C	SET 5,H	DDCBXX86	RES 0,(IX+d)	FD23	INC IX
CBE D	SET 5,L	DDCBXX8E	RES 1,(IX+d)	FD29	ADD IV,IY
CBE E	SET 5,(HL)	DDCBXX96	RES 2,(IX+d)	FD2AXXXX	LD IV,(NN)
CBE F	SET 5,A	DDCBXX9E	RES 3,(IX+d)	FD2B	DEC IY
CBF0	SET 6,B	DDCBXXA6	RES 4,(IX+d)	FD34XX	INC(IY+d)
CBF1	SET 6,C	DDCBXXAE	RES 5,(IX+d)	FD35XX	DEC(IY+d)
CBF2	SET 6,D	DDCBXXB6	RES 6,(IX+d)	FD36XX20	LD(IY+d),N
CBF3	SET 6,E	DDCBXXBE	RES 7,(IX+d)	FD39	ADD IY,SP
CBF4	SET 6,H	DDCBXXC6	SET 0,(IX+d)	FD46XX	LD B,(IY+d)
CBF5	SET 6,L	DDCBXXCE	SET 1,(IX+d)	FD3EXX	LD C,(IY+d)
CBF6	SET 6,(HL)	DDCBXXD6	SET 2,(IX+d)	FD56XX	LD D,(IY+d)
CBF7	SET 6,A	DDCBXXDE	SET 3,(IX+d)	FD5EXX	LD E,(IY+d)
CBF8	SET 7,B	DDCBXXE6	SET 4,(IX+d)	FD66XX	LD H,(IY+d)
CBF9	SET 7,C	DDCBXXEE	SET 5,(IX+d)	FD6EXX	LD L,(IY+d)
CBFA	SET 7,D	DDCBXXF6	SET 6,(IX+d)	FD70XX	LD (IY+d),B
CBFB	SET 7,E	DDCBXXFE	SET 7,(IX+d)	FD71XX	LD (IY+d),C
CBFC	SET 7,H	ED40	IN B,(C)	FD72XX	LD (IY+d),D
CBFD	SET 7,L	ED41	OUT(C),B	FD73XX	LD (IY+d),E
CBFE	SET 7,(HL)	ED42	SBC HL,BC	FD74XX	LD (IY+d),H
CBFF	SET 7,A	ED43XXXX	LD(NN),BC	FD75XX	LD (IY+d),L
DD09	ADD IX,BC	ED44	NEG	FD77XX	LD (IY+d),A
DD19	ADD IX,DE	ED45	RETN	FD7EXX	LD A,(IY+d)
DD21XXXX	LD IX,NN	ED46	IM 0	FD86XX	ADD A,(IY+d)
DD22XXXX	LD(NN),IX	ED47	LD 1,A	FD8EXX	ADC A,(IY+d)
DD23	INC IX	ED48	IN C,(C)	FD96XX	SUB(IY+d)
DD29	ADD IX,IX	ED49	OUT(C),C	FD9EXX	SBC A,(IY+d)
DD2AXXXX	LD IX,(NN)	ED4A	ADC HL,BC	FDA6XX	AND (IY+d)
DD2B	DEC IX	ED4BXXXX	LD BC,(NN)	FDAXXX	XOR (IY+d)
DD34XX	INC(IX+d)	ED4D	RET1	FD86XX	OR (IY+d)
DD35XX	DEC(IX+d)	ED50	IN D,(C)	FD8EXX	CP (IY+d)
DD36XX20	LD(IX+d),N	ED51	OUT(C),D	FDE1	POP IY
DD39	ADD IX,SP	ED52	SBC HL,DE	FDE3	EX (SP), IY
DD46XX	LD B,(IX+d)	ED53XXXX	LD(NN),DE		

HEXADECIMAL	MNEMONIC	HEXADECIMAL	MNEMONIC	HEXADECIMAL	MNEMONIC
FDE5	PUSH IY				
FDE9	JP (IY)				
FD F9	LD SP,IY				
FDCBXX06	RLC(IY+d)				
FDCBXX0E	RRC(IY+d)				
FDCBXX16	RL(IY+d)				
FDCBXX1E	RR(IY+d)				
FDCBXX26	SLA(IY+d)				
FDCBXX2E	SRA(IY+d)				
FDCBXX3E	SRL(IY+d)				
FDCBXX46	BIT 0,(IY+d)				
FDCBXX4E	BIT 1,(IY+d)				
FDCBXX56	BIT 2,(IY+d)				
FDCBXX5E	BIT 3,(IY+d)				
FDCBXX66	BIT 4,(IY+d)				
FDCBXX6E	BIT 5,(IY+d)				
FDCBXX76	BIT 6,(IY+d)				
FDCBXX7E	BIT 7,(IY+d)				
FDCBXX86	RES 0,(IY+d)				
FDCBXX8E	RES 1,(IY+d)				
FDCBXX96	RES 2,(IY+d)				
FDCBXX9E	RES 3,(IY+d)				
FDCBXXA6	RES 4,(IY+d)				
FDCBXXAE	RES 5,(IY+d)				
FDCBXXB6	RES 6,(IY+d)				
FDCBXXBE	RES 7,(IY+d)				
FDCBXXC6	SET 0,(IY+d)				
FDCBXXCE	SET 1,(IY+d)				
FDCBXXD6	SET 2,(IY+d)				
FDCBXXDE	SET 3,(IY+d)				
FDCBXXE6	SET 4,(IY+d)				
FDCBXXEE	SET 5,(IY+d)				
FDCBXXF6	SET 6,(IY+d)				
FDCBXXFE	SET 7,(IY+d)				

APPENDICE L

=====

ISTRUZIONI PER L'USO DELLA CASSETTA

La cassetta allegata contiene i due programmi monitor presentati nel libro (EZCODE e HEXLOAD) ed il programma FROG completo.

A questi programmi sono poi stati aggiunti i vari moduli separati, che compongono il programma FROG, con i relativi programmi di prova in modo che, volendo, possiate utilizzarli nel corso della progettazione.

Vi consigliamo comunque di usare solo i programmi monitor "EZCODE" e "HEXLOAD" e di provare invece a costruire da soli, seguendo il libro e utilizzando delle vostre cassette vergini, il programma FROG passo dopo passo. Avrete così la soddisfazione, terminato il lavoro, di giocare con un programma costruito con le vostre mani.

Vi diamo ora una breve descrizione di ciascuno dei programmi contenuti nella cassetta con le indicazioni per il loro uso corretto.

Programma monitor EZCODE

Con questo programma potete costruire moduli composti al più da 800 bytes. L'indirizzo di caricamento deve essere superiore a 31499.

Edit di linea: ogni linea deve iniziare con (numero linea) e spazio. Il contenuto di una linea deve essere scritto in formato esadecimale e contenere al più 4 bytes. Una linea vuota provoca la cancellazione di quanto scritto precedentemente nella linea stessa. Nelle istruzioni di SALTO (relativo o assoluto) potete riferirvi direttamente al numero di linea voluto (cfr. il libro per ulteriori chiarimenti). Ad esempio, JR LINEA 4 verrà scritto come 18 14: notate l'uso della lettera "l" per indicare che 4 è un numero di linea.

Comandi: "dump" costruisce il programma oggetto, che verrà posto in memoria nell'area da voi specificata pronto per l'esecuzione, che potete lanciare con "run"; "exit" rilascia il programma monitor e restituisce il controllo al BASIC; "list" seguito da un numero di linea visualizza venti linee del programma sorgente, a partire da quella indicata. Se non viene indicato nessun numero di linea, la visualizzazione parte dalla prima linea; "load" carica un modulo da cassetta; "mem" permette di visualizzare il contenuto delle locazioni di memoria desiderate; "new" cancella il modulo corrente e riinizializza il programma monitor; "save" permette di salvare su cassetta, a scelta, o il programma sorgente o il programma oggetto.

Per far ripartire il programma EZCODE potete usare o RUN (nel qual caso vengono riinizializzate tutte le variabili) o il comando GOTO 2020. I numeri di linea ed i riferimenti ad essi devono essere in formato decimale, mentre le istruzioni devono essere in formato esadecimale.

Programma monitor HEXLOAD

E' un programma piu' piccolo, rispetto a EZCODE, e posiziona il RAMTOP a 26999. Pertanto potrete caricare i vostri moduli a partire da 27000. Le sue funzioni sono: "SCRITTURA" in memoria di codici esadecimali; "CARICAMENTO" di moduli da cassetta; "CONSERVAZIONE" di moduli su cassetta; "VISUALIZZAZIONE" di locazioni di memoria; "SPOSTAMENTO" di blocchi di dati all'interno della memoria stessa. Per far partire il programma battere RUN (ENTER).

Programma FROG

Contiene l'intero codice macchina relativo al gioco della rana che attraversa la strada. Il programma occupa le locazioni da 6978H a 7810H (in decimale da 27000 a 31005).

Per caricare il programma da cassetta battere: LOAD "" CODE (ENTER).
Per lanciarne l'esecuzione battere: LET L=USR 27000.

Controlli: "I" per salire, "A" per scendere, "J" per muoversi da destra a sinistra e "P" da sinistra a destra. Per far suonare la sirena della polizia battere (ENTER). Lo stesso tasto disabilita la sirena se questa e' in funzione. Per far interrompere il programma battere (BREAK).

MODULI DEL PROGRAMMA FROG

Dato che tutti i moduli sono scritti in codice esadecimale, usate il programma HEXLOAD per caricarli.

Fase 1

I dati relativi a questa fase sono contenuti nel modulo "frog" che deve essere caricato a partire da 27055 fino a 28504.

Il modulo e' formato dai seguenti data base:

"shpdb" da 27055 a 28040, 986 bytes, checksum 79197.

"objdb" da 28289 a 28428, 140 bytes, checksum 7697.

"gendb" da 28429 a 28504, 76 bytes, checksum 4818.

Fase 2

Caricare "init" da 28547 a 30747 (2201 bytes).

Caricare "test2" a partire da 32000 e battere LET L=USR 32000 (ENTER) per testare il modulo.

Dovreste vedere i bordi della strada e 5 rane allineate in basso a sinistra.

Salvate il contenuto delle locazioni da 27005 a 30747 su una vostra cassetta, sotto il nome di "frog".

Fase 3

Caricate "regtrf" da 28861 a 30685 (1824 bytes).
Caricate "test3" e battete LET L=USR 32000 (ENTER) per provare il modulo. Vedrete il traffico scorrere velocemente nei due sensi.
Salvate da 27055 a 30747 su una cassetta come "frog".

Fase 4

Caricate "police" da 29514 a 29889 (376 bytes).
Caricate "test4" e battete LET L=USR 32000 (ENTER) per testare il modulo. Vedrete ora anche l'auto della polizia con la sirena.
Salvate da 27055 a 30747 come "frog".

Fase 5

Caricate "frgrtn" da 29890 a 30574 (685 bytes).
Testate il programma caricando "test5" e battendo LET L=USR 32000. Dovreste riuscire a far muovere la rana e a vedere il punteggio sul video.
Salvate da 27055 a 30747 come "frog".

Fase 6

Caricate "frgctrl" e collegatelo a "frog" a partire da 27000. Avete così completato il programma. Salvate tutto (da 27000 a 30747) sotto il nome di "frog" e otterrete lo stesso programma da voi registrato sotto il nome di "FROG". Buon divertimento!

Se vi sentite frustrati per le limitazioni del BASIC e vorreste scrivere programmi o routines più veloci, più potenti e che non occupino troppo spazio in memoria, questo libro fa per voi. Anche se non avete avuto finora alcuna esperienza nell'uso di linguaggi di tipo Assembler, questo libro vi metterà in grado di apprezzare e utilizzare vantaggiosamente le potenzialità del linguaggio macchina dello Spectrum.

Ogni capitolo contiene esempi esplicativi sull'uso di tutte le istruzioni del linguaggio macchina e semplici esercizi che potrete risolvere costruendo semplici programmi da caricare sul vostro Spectrum.

In particolare l'ultima parte del volume è dedicata alla progettazione e alla realizzazione di un divertente programma: il gioco della rana che attraversa la strada ("FREEWAY FROG"). Di tale programma viene presentato nel libro il listato completo sia del codice macchina che del codice Assembler.

Al volume infine è allegata una cassetta contenente due programmi BASIC ("EZCODE" e "HEXLOAD") che vi permetteranno di scrivere, caricare, salvare ed eseguire programmi in linguaggio macchina. Sulla stessa cassetta inoltre è stato registrato il codice oggetto completo del programma Freeway frog ("FROG") e di tutti i moduli che lo compongono.

SIMCLAIR ZX SPECTRUM ASSSEMBLER ELMCGUAGCIOMACCHINA PER PRIMICIPIANTI

