Semplice introduzione all'affascinante tecnica dei computer

Volume 2

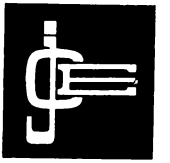
JUNIOR COMPUTER

elektor

Junior-Computer 2

La semplice introduzione
nell'affascinante tecnica
dei computer

A. Nachtmann
G.H. Nachbar



JACOPO CASTELFRANCHI EDITORE
Via dei Lavoratori, 124
20092 Cinisello B. (MI)

Copyright © Uitgeversmaatschappij Elektuur B.V. 6190 AB Beek-1981

Ogni riproduzione o copia, anche parziale, di questo libro, è strettamente vietata se non vi è il permesso scritto dell'editore.

Diritti d'autore

La protezione del diritto d'autore si estende non solamente al contenuto, ma anche alle illustrazioni, circuiti stampati compresi, nonché ai progetti e dettagli relativi. Conformemente alla legge sui Brevetti n° 1127 del 29-6-39, i circuiti riportati non possono essere realizzati altro che ai fini privati e scientifici e comunque non commerciali. L'utilizzazione degli schemi e le relative applicazioni non comportano alcuna responsabilità da parte della Società editrice.
Editore JCE - Divisione Elektor - Via dei Lavoratori, 124 - 20092 Cinisello B.
Prima edizione in lingua italiana 1982
Stampato da S.p.A. Alberto Matarelli - Milano

Interludio ... tra i pasti!

Come sono andati i vostri primi passi nel mondo dei computer assieme allo Junior-Computer? Bene, sicuramente?! Ottima cosa, perchè vuol dire che siete ormai pronti ad affrontare nuove imprese. Vi diremo quindi in poche parole come potrete saziare la vostra fame di progresso.
Perciò: buon appetito!
Dopo aver montato lo Junior-Computer ed aver effettuato le prime prove di marcia, descritte nei capitoli 1 ... 4 (Volume I) in questo nuovo libro si va al nocciolo. Ossia: vi renderete padroni dei corretti ferri del mestiere per lavorare con lo Junior Computer. E gli utensili, per un computer, vogliono dire la programmazione di appropriati programmi.
Il capitolo 5 giunge proprio a proposito. L'"Editor" e l'"Assembler" costituiscono importanti ausili per il programmatore. Diventa facile correggere errori d'impostazione. Potete inoltre inserire in un secondo momento delle istruzioni mancanti in qualsiasi zona della memoria di lavoro, senza dover reimpostare tutto il programma. Infine, non risulta più necessario calcolare preventivamente gli "offset" dei salti condizionati nè gli indirizzi assoluti delle istruzioni di salto: questo stupido genere di lavoro viene assolto dall'Assembler della EPROM dello Junior-Computer standard.
Nel capitolo 6 gli argomenti divengono altisonanti. Questo termine risulta appropriato, perchè vedremo come con pochi circuiti accessori e il Peripheric Interface Adapter (abbrev. PIA), lo Junior-Computer si trasforma in un "carillon". In fin dei conti, per il Computer non fa differenza azionare tramite il PIA un altoparlante od una stampante. D'altra parte la sezione che tratta la stampante è prevista solo nel 3° volume.
Quella che si può chiamare l'"intelligenza" della EPROM dello Junior-Computer è descritta nei capitoli 7 (Monitor), 8 (Editor) e 9 (Assembler). Quando avrete appreso il modo di pensare dello Junior, sarete in grado di stendere vostri programmi.
Le ben note subroutine ora disponibili vi saranno di grande ausilio per la "Do-it-yourself-Software". Nelle appendici, infine, troverete i "listing" (elenchi di istruzioni) di tutte le subroutine ed altri programmi.

E poi? Ora che la programmazione è stata chiarita, quale ne è l'utilità pratica? Sono due domande giustificate, per le quali vi è una risposta unica: con il 3° volume, potrete sviluppare il vostro Junior-Computer ad un completo Personal-Computer, che porterà sì il nome "Junior", ma a cui però non calzano più ormai le scarpe da bambino.

Gli Autori

P.S. Non c'è nulla di perfetto al mondo: perciò saremo grati a tutti coloro che vorranno suggerirci per iscritto aggiunte o varianti allo Junior ed ai suoi tre volumi.

Indice

Capitolo 5 1
Come si lavora con Editor ed Assembler - *L'editing e l'"assemblaggio" facilitano l'impostazione di programmi esenti da errori.*

Capitolo 6 29
Il PIA o Peripheric Interface Adapter - *L'organo di collegamento fra il computer ed il mondo esterno.*

Capitolo 7 97
Il programma Monitor - *Una Software indispensabile.*

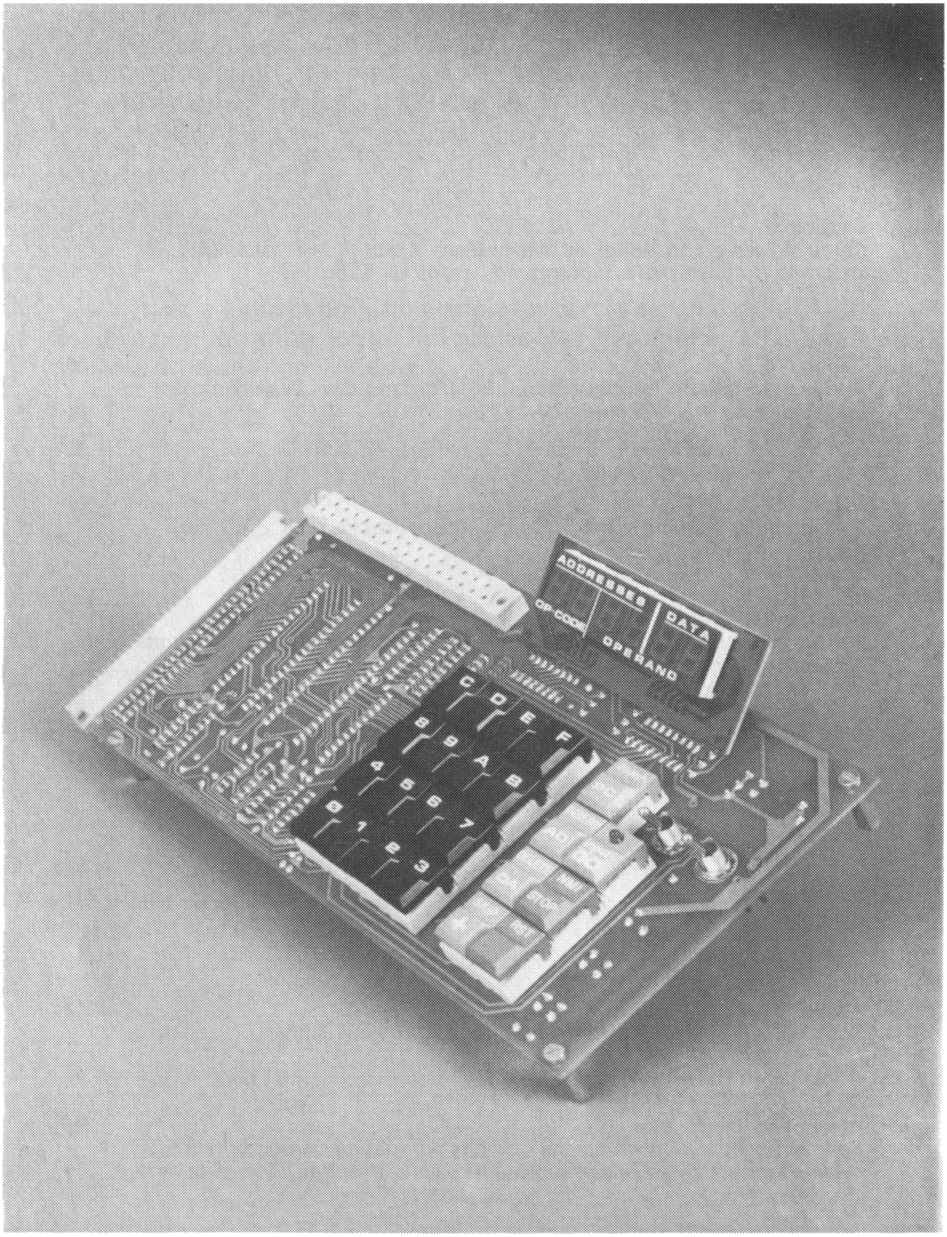
Capitolo 8 145
Il programma Editor - *È la "gomma per cancellare" per programmi contenenti errori di impostazione.*

Capitolo 9 185
Il programma Assembler - *Adatta alla CPU i programmi introdotti tramite l'Editor.*

Appendice 1 211
Sommario delle Subroutine.

Appendice 2 215
Source Listing - Editor, Assembler, Monitor, routine Branch.

Appendice 3 226
Source Listing - Binary-Decimal Conversion, routine Demo, routine Play, routine Input, routine Repeat; sobroutine diverse e routine d'Interrupt varie.


ADDRESSES
DATA
OP-CODE
OPERAND
C D E F
8 9 A B
4 5 6 7
0 1 2 3

5

Come si lavora con Editor ed Assembler

L'introduzione da tastiera di un programma è un lavoro lungo e noioso. Come abbiamo visto nel 1° volume, le singole istruzioni devono venire introdotte nel computer byte per byte. Prima di passare all'impostazione di un programma, si rendono necessari vari preparativi su carta:

- **— Calcolo degli indirizzi di partenza di subroutine**
- **— Calcolo degli "offset" per i salti condizionati di programma**
- **— Calcolo degli indirizzi assoluti per i salti di programma non condizionati.**

Quando finalmente il programma è stato interamente introdotto nel computer, occorre verificare l'esistenza di eventuali errori di digitazione. Che fare, ad esempio, quando proprio nella parte iniziale non è stato impostato un dato byte: ridigitare di bel nuovo tutto il programma?! No, non è più necessario!

Nell'EPROM dello Junior-Computer sono situati un Editor ed un Assembler. Grazie all'Editor è possibile introdurre in un secondo tempo istruzioni in un programma; ricercare determinate istruzioni; ed eliminare determinate istruzioni da un programma. La tastiera e l'Editor costituiscono quindi "il lapis e la gomma per cancellare" utilizzati per introdurre un programma nel computer. Mediante l'Editor è pure possibile introdurre nel computer indirizzi simbolici, i cosiddetti Label. In tal modo, con l'Assembler, il computer può calcolare automaticamente gli indirizzi di partenza delle subroutine, gli offset dei salti condizionati e gli indirizzi assoluti per i salti incondizionati. Gli errori di programmazione vengono così ridotti al minimo, dato che è lo stesso Junior-Computer ad intervenire in aiuto del programmatore durante lo sviluppo di un programma. In questo capitolo descriveremo come si lavora con Editor ed Assembler.

EDITOR

La maggior parte delle istruzioni e dei tipi di indirizzamento della CPU 6502, sono noti dal 1° volume sullo Junior-Computer.
Diversi programmi didattici hanno dimostrato quanto risulti semplice programmare lo Junior-Computer. Sin qui, i dati sono stati introdotti solo in forma di numeri esadecimali o byte.
Il computer interpreta i dati introdotti dopo aver premuto il tasto AD come indirizzi; la pressione del tasto DA fa sì, invece, che il computer depositi i dati introdotti agli indirizzi previamente indicati.
Per programmi brevi, quali ad esempio i programmi didattici del 1° volume, questo sistema di introduzione dei dati è interamente sufficiente. Per programmi maggiori, della lunghezza di varie centinaia di byte, accade spesso nell'introduzione dei dati che capitino degli errori di digitazione. Che fare se ad esempio si sono scordati un paio di byte verso la metà del programma, per errore? Sinora conoscevamo solo una possibilità per rimediare, ossia ribattere interamente il programma a partire dal punto in cui era intervenuto l'errore. Un lavoro noioso e perditempo! Esiste bensì la possibilità di eliminare eventuali istruzioni inutili sovrascrivendole con istruzioni NOP (cod. esadec. EA): ma ciò costituisce un cattivo modo di impiego della memoria del nostro Junior-Computer. Particolarmente fastidioso è il caso in cui, dopo aver impostato un lungo programma, ci si accorge che alla fine mancano un paio di byte di spazio di memoria, proprio per uno sfruttamento non ottimale della sua capacità.
D'ora in poi tutto ciò non dobbiamo più temerlo, perchè il programma Monitor dello Junior-Computer contiene un EDITOR. Mediante l'Editor è possibile inserire od estrarre byte in qualsiasi locazione della memoria. Quando inserisce dei byte, il computer sposta verso il basso tutto il blocco di dati a partire dalla posizione nella quale si vuole inserire un dato numero di byte. I programmi con i quali si può eseguire il trasferimento di blocchi di dati (Block-Transfer) ci sono già noti dal 1° volume, alle voci relative agli indirizzamenti indicizzati ed indiretti.
L'Editor ci solleva quindi da un bel po' di lavoro quando occorre correggere programmi dopo la loro impostazione. Questo capitolo illustrerà compiutamente quanto risulti facile introdurre programmi nello Junior-Computer e correggerli se necessario.

ASSEMBLER

Un aiuto impagabile per introdurre i programmi nel computer in modo semplice, veloce e soprattutto senza errori, ci è offerto dall'Assembler. Riportiamoci alla memoria ancora una volta il 1° volume! Ivi sono state spesso impiegate istruzioni di salto (Branch). Per calcolare gli offset di questi salti, occorre cono-

scere gli indirizzi di partenza e di arrivo dei relativi salti. Solo dopo, mediante l'impiego della routine BRANCH contenuta nel Monitor (indirizzo iniziale 1FD5), è possibile calcolare l'offset. Lo stesso accade per le istruzioni JSR e JMP: anche in questo caso è necessario conoscere l'effettivo indirizzo del salto. Prima di introdurre il programma, quindi, bisogna che siano noti gli indirizzi assoluti di tutte le istruzioni di salto, condizionato e non. Prima di ricavare con gli opportuni conteggi gli indirizzi corretti, è necessario un bel po' di calcoli scritti.
Questo compito, invece, può essere svolto in modo molto semplice, ma soprattutto senza errori tramite l'Assembler contenuto nel programma Monitor. Basta la pressione del tasto GO, ed il computer calcola da sè tutti gli offset, gli indirizzi assoluti delle istruzioni JMP e gli indirizzi di partenza delle subroutine. Può apparire incredibile, ma lo Junior-Computer solleva il programmatore da questo scabroso incarico, svolgendolo in frazioni di secondo. Sia detto per inciso: notare che Assembler ed Editor, oltre ad alcuni altri programmi, sono contenuti in una EPROM da 1 solo kbyte! Ciò è reso possibile dal fatto che la CPU 6502 possiede istruzioni molto "roubuste" ed il massimo numero di modalità di indirizzamento fra i microprocessori esistenti sul mercato.
Ricapitolando, si può dire che mediante l'Editor e l'Assembler si rende possibile sviluppare presto e bene i programmi. Lo Junior-Computer assume in proprio molto lavoro di calcolo, sollevandone il programmatore.

"Editing" ed "Assembling" dall'A alla Z

Prima di esaminare in dettaglio come si utilizzano Editor ed Assembler, procediamo a sviluppare un programma che successivamente verrà introdotto nello Junior-Computer. Questo programma deve provvedere a:
— convertire un numero binario di 8 bit in numero decimale,
— visualizzare il numero binario citato, sotto forma esadecimale, accanto al numero decimale corrispondente, sul display dello Junior-Computer,
— il numero esadecimale va introdotto tramite la tastiera,
— per la scansione del display a 6 cifre si impiegano routine del Monitor.
Il problema è dunque quello di introdurre velocemente e senza errori un programma che risponda ai requisiti sopra citati. Cominciamo con lo sviluppo di un algoritmo che converte un numero binario di 8 bit (00 ... FF) in un numero decimale (0_{10} ... 255_{10}). Consideriamo la fig. 1, dove è tradotto in parole l'algoritmo che esegue la conversione desiderata.
Per inciso, è sempre bene, quando si deve stendere un program-

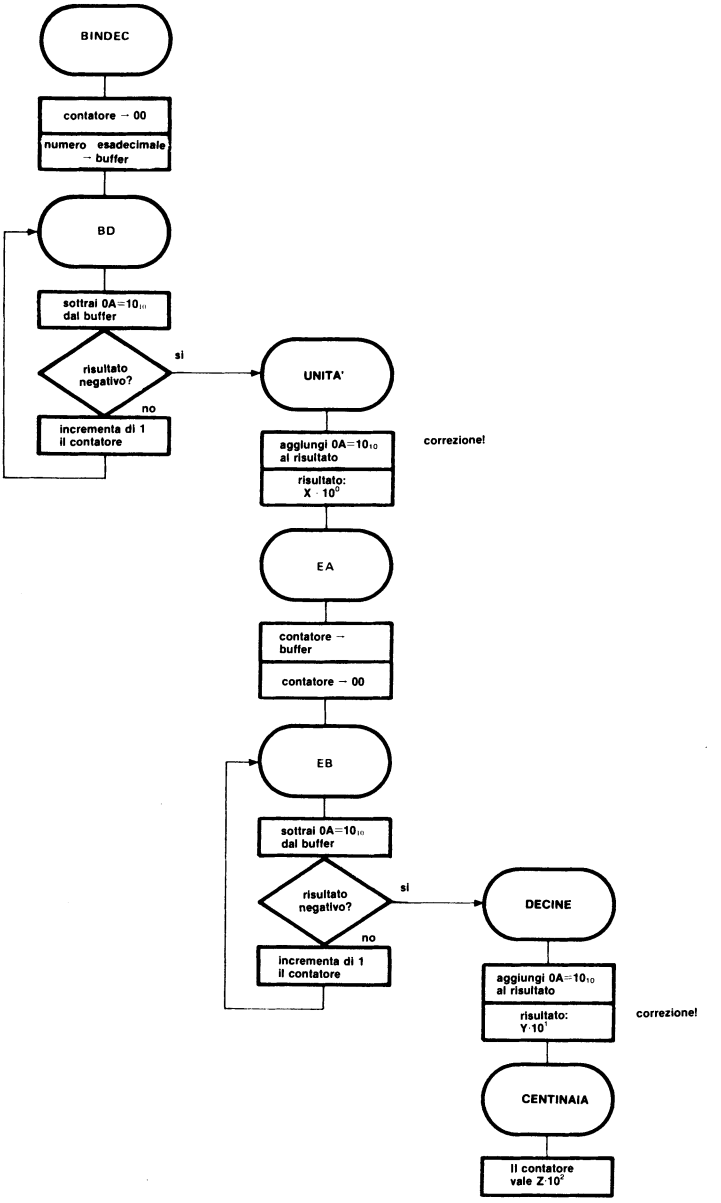


Figura 1. Prima di incominciare a lavorare con Editor ed Assembler, sviluppiamo un programma che converte un numero esadecimale in numero decimale. In questo modo risulterà chiaro come è possibile in seguito utilizzare Editor ed Assembler quali "strumenti per la programmazione" nella stesura dei programmi. Il diagramma di flusso illustra l'algoritmo di conversione numerica citato.

ma complicato, tradurre preventivamente in parole il problema da risolvere! Successivamente sarà facile stendere una "Flow Chart" generale che infine porterà al programma di dettaglio.
Torniamo al nostro programma. Nella sezione di programma BIN-DEC, che corrisponde alla conversione binario-decimale, per prima cosa si azzera un opportuno contatore. Successivamente, bisogna depositare in un buffer il numero esadecimale da convertire in decimale. Vediamone i motivi nel paragrafo che segue:

1. Dal numero esadecimale nel buffer si sottrae tante volte il numero $0A=10_{10}$ sin che il contenuto del buffer è minore di zero. Ad ogni sottrazione si incrementa il contatore di 1. Il contenuto del contatore, al termine, rappresenta il numero di volte che il computer ha eseguito le sottrazioni sino a che il risultato è divenuto negativo.
2. Per poter rappresentare un numero negativo di 8 bit, con un processore da 8 bit, ci serve un buffer numerico di 16 bit, che occupa 2 celle di memoria.
3. Se il bit di posizione più alta di questo numero di 16 bit vale 1, significa che il risultato della sottrazione è negativo. Il programma effettua un salto al Label EINER, e per prima cosa aggiunge 0A al valore negativo del buffer. Il risultato è un numero da 0 a 9, che rappresenta le unità del numero decimale.
4. Il valore contenuto nel contatore viene trasferito nel buffer di 16 bit, e poi il contatore viene riazzerato. Quindi il processore torna a sottrarre 0A dal contenuto del buffer, sin quando il risultato è negativo. Anche qui il contatore indica il numero delle corrispondenti sottrazioni.
5. Dopo aver nuovamente aggiunto 0A al valore negativo del buffer, otteniamo un numero da 0 a 9, che rappresenta le decine del numero decimale.
6. Il valore del contatore a questo punto è un terzo numero da 0 a 2, che rappresenta le centinaia del numero decimale.

Un esempio pratico dimostra l'effettivo funzionamento di questo algoritmo. Vogliamo convertire il numero esadecimale 0091 in decimale, utilizzando il programma sviluppato a tale scopo (fig. 1)

```
Il buffer numerico vale:  0091 Il contatore vale 00
                        —   0A Il contatore vale 01
                          ----
Il buffer numerico vale:  0087
                        —   0A Il contatore vale 02
                          ----
Il buffer numerico vale:  007D
                        —   0A Il contatore vale 03
                          ----
Il buffer numerico vale:  0073
                        —   0A Il contatore vale 04
                          ----
Il buffer numerico vale:  0069
                        —   0A Il contatore vale 05
                          ----
Il buffer numerico vale:  005F
                        —   0A Il contatore vale 06
                          ----
```

Il buffer numerico vale: 0055
— 0A Il contatore vale 07
Il buffer numerico vale: 004B
— 0A Il contatore vale 08
Il buffer numerico vale: 0041
— 0A Il contatore vale 09
Il buffer numerico vale: 0037
— 0A Il contatore vale 0A
Il buffer numerico vale: 002D
— 0A Il contatore vale 0B
Il buffer numerico vale: 0023
— 0A Il contatore vale 0C
Il buffer numerico vale: 0019
— 0A Il contatore vale 0D
Il buffer numerico vale: 000F
— 0A Il contatore vale 0E
Il buffer numerico vale: 0005
— 0A Contatore inalterato

Il risultato è negativo, e torna positivo sommandovi 0A, passando a 5. Si è così calcolato:

$$\$91 = XY5_{10}$$

Il contatore vale ora 0E, che viene trasferito nel buffer numerico. Quindi il programma riazzera il contatore. Lo sviluppo successivo è:

Il buffer numerico vale: 000E Il contatore vale 00
- 0A Il contatore vale 01
Il buffer numerico vale: 0004
- 0A Contatore invariato

raggiungendo ancora un valore negativo, che sommando 0A viene riportato a 4. Si è così calcolato:

$$\$91 = X45_{10}$$

Il contenuto del contatore è adesso 1, che rappresenta le centinaia del numero decimale. Tale numero risulta quindi:

$$91 = 145_{10}$$

Dal volume 1° già sappiamo come le singole cifre del numero decimale 145 si "impaccano" nel buffer di display dello Junior Computer.
Abbiamo così sviluppato l'algoritmo di conversione binario-decimale, e possiamo tracciare il diagramma di flusso "grezzo" del nostro programma, illustrato in fig. 2. Inizialmente il buffer dei display viene riempito di zeri. Quindi il computer verifica lo stato della tastiera e gestisce il display. La relativa subroutine del programma Monitor ci è già nota dal volume 1°: GETBYT con indirizzo iniziale 1D6F. Se risulta premuto un tasto comandi, il processo-

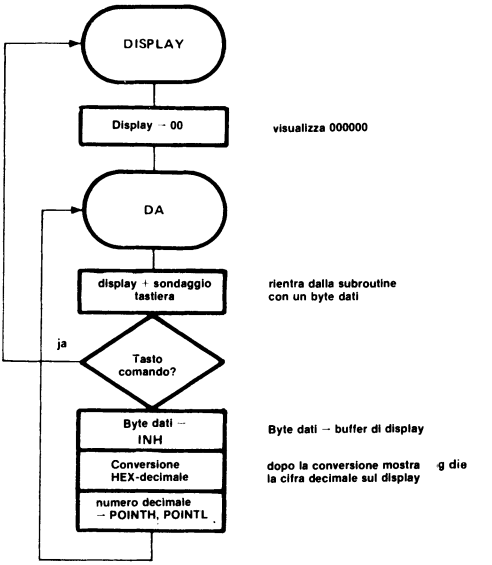


Figura 2. Questo è il diagramma di flusso globale del programma per la gestione del display ed il sondaggio della tastiera nella conversione di un numero esadecimale.

re rientra da questa subroutine nel programma principale, avendo posto a 0 il Flag N. In tal modo il display viene azzerato ogni volta che si preme un tasto comandi. Dopo che sono stati premuti due tasti dati, il microprocessore esce dalla subroutine GETBYT. Il dato impostato (1 byte) si trova nell'Accumulatore (abbrev. Accu) della CPU e viene portato al buffer di display INH. Quindi, in una subroutine, viene eseguita la conversione binario-decimale secondo l'algoritmo descritto. Il numero decimale così ottenuto viene trasferito dal processore nei buffer di display POINTH e POINTL. Successivamente il computer torna ad interessarsi dello stato della tastiera, e colla subroutine GETBYT visualizza il numero decimale e quello esadecimale.
Il diagramma di flusso dettagliato è illustrato in fig. 3a.
Il buffer di display ha gli indirizzi 00F9...00FB. Il buffer dati o numerico interessa invece gli indirizzi 00D7 e 00D8. Le locazioni di memoria riservate al programma sono le seguenti:

```
INH     *  $00F9 DISPLAY BUFFERS
POINTL  *  $00FA
POINTH  *  $00FB
HEXL    *  $00D7 DATA BUFFERS
HEXH    *  $00D8
GETBYT  *  $1D6F MONITOR SUBROUTINE
```

Quale contatore del numero di sottrazioni non impieghiamo locazioni della memoria, bensì il registro Y.
Come si vede in fig. 3, il programma DISPL include le due subroutine GETBYT e HEXDEC. Esse differiscono fra loro per il fatto che è noto l'indirizzo di partenza per GETBYT, ma non quello di HEXDEC. D'ora in poi non ce ne dovremo preoccupare, perché sarà il calcolatore a calcolarsi in modo automatico gli indirizzi di partenza incogniti. Non è quindi necessario assegnare un indirizzo a HEXDEC. Inoltre, il programma principale prevede pure un istruzione JMP ed un istruzione di salto condizionato. Il computer provvede autonomamente a calcolare a quale indirizzi effettuare i

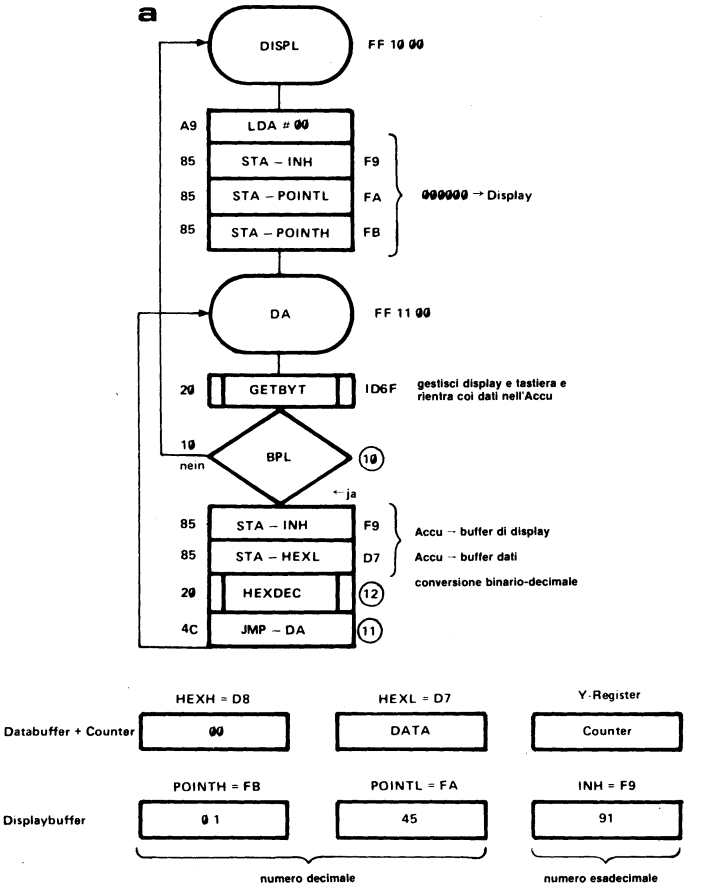


Figura 3a. Il diagramma di flusso dettagliato per fig. 2. Il numero esadecimale, da convertire in numero decimale, è posto nel buffer di display INH. A conversione eseguita il numero decimale si trova nei due buffer di display POINTH e POINTL. Le locazioni HEXH e HEXL nonché il registro Y fungono da memorie dati temporanee.

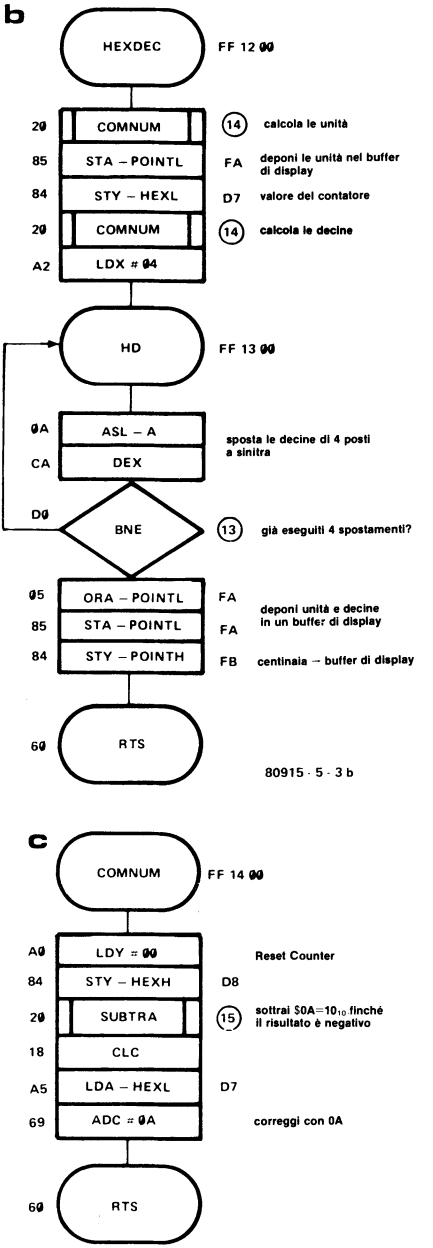


Figura 3b e Fig 3c. La conversione del numero esadecimale impostato in decimale avviene nella subroutine HEXDEC. Prima vengono calcolate mediante la subroutine COMNUM le unità, poi le decine e infine le centinaia del numero decimale.

relativi salti, senza che il programmatore debba più rompersi la testa al riguardo.
La subroutine HEXDEC (Fig. 3b), che esegue in pratica l'algoritmo che abbiamo descritto prima, include le due subroutine COMNUM e SUBTRA (Fig. 3c e 3b), nonché istruzioni di tipo JMP e Branch.
Anche qui non ci dobbiamo più curare di sapere quali siano gli indirizzi iniziali delle subroutine o gli offset dei salti. Questi compiti lunghi e noiosi li lasciamo al computer, che non commette errori, perché il suo programma Monitor è esente da ogni pecca!
La sezione del programma Monitor che provvede a calcolare gli offset delle istruzioni di salto e gli indirizzi assoluti dei salti incondizionati si chiama Assembler. Il suo indirizzo iniziale è: 1F51. La pressione del tasto GO inizializza l'Assembler. Affinché l'Assembler possa funzionare, il computer necessita di un "congegno" mediante il quale si possano introdurre programmi "su misura per

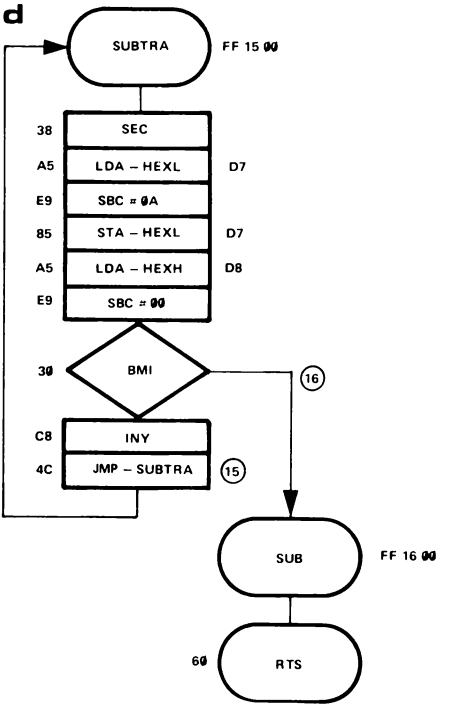


Figura 3d. Nella subroutine SUBTRA il processore esegue una sottrazione di 16 bit. Si toglie 0A dal numero esadecimale ripetutamente sino ad ottenere un risultato negativo. Il registro Y viene impiegato come contatore per le sottrazioni.

l'Assembler". Questo "congegno" è l'Editor, che provvede a rendere un programma introdotto da tastiera "secondo i gusti" del computer. Come già detto, il computer deve ora calcolarsi indirizzi ed offset in modo autonomo. Come fa a sapere dove deve effettuare i salti, se il programmatore non ha indicato né l'indirizzo di partenza né quello di arrivo? La risposta è questa: il programmatore introduce nel computer *indirizzi simbolici*. D'ora in poi non scriveremo più quindi JSR-023B, bensì JSR-SUBTRA. Entrambe queste scritture per il computer si equivalgono! Il compito dell'Assembler è ora proprio quello di assegnare all'indirizzo simbolico SUBTRA l'indirizzo assoluto 023B. Non ci è tuttavia possibile introdurre (per ora) la parola SUBTRA nell'Junior-Computer, dato che disponiamo solo di una tastiera esadecimale e non di una alfa-numerica. Quindi, un Label può contenere solo simboli numerici esadecimali. In memoria possiamo trovare come operandi sia numeri esadecimali che codici operativi (OP). Il computer non potrebbe quindi distinguere da solo se si tratta di un Label o di un codice OP. È dunque, come si possono distinguere fra loro un Label ed un codice OP? Diamo un'occhiata all'appendice del 1° volume, a pag. 167. Ivi sono riportati in ordine esadecimale i codici OP della CPU 6502.

Come si vede da questa tabella, non a tutti i numeri esadecimali 00...FF sono associati dei codici OP: la tabella presenta delle lacune. Neanche al valore FF è associato un codice OP. Pertanto associeremo a questo numero esadecimale un pseudo-codice OP, che il processore non riconosce. Così risulta possibile impiegare il numero esadecimale FF quale indicatore d'un Label. Avremmo naturalmente potuto impiegare a questo scopo altri numeri ricavati dalla tabella, ad es. 04, D3, F7 ecc.: ma FF è un valore facile da ricordare e da impostare, e questo è il motivo della nostra scelta di tale numero esadecimale quale indicatore di Label.

Restiamo ancora un momento presso il Label SUBTRA in fig. 3d. SUBTRA costituisce il punto di partenza, o meglio, l'indirizzo simbolico d'inizio di una subroutine. Quando vogliamo richiamare tale subroutine, lo faremo con: JSR-SUBTRA (in parole: salta alla subroutine SUBTRA). Dato che con l'Editor dello Junior-Computer si possono impiegare solo simboli numerici esadecimali, dobbiamo assegnare un numero alla subroutine SUBTRA: tale numero potrà essere da 00 a FF. In tal modo risulta possibile assegnare 256 Label diversi in un programma. A questo punto i concetti dell'Editor e dell'Assembler dovrebbero risultare chiari. Alla luce di un paio di esempi pratici cercheremo di far sì che la teoria risulti completamente assorbita, perché d'ora in poi per l'introduzione dei programmi impiegheremo esclusivamente l'Editor e l'Assembler del programma Monitor dello Junior-Computer.

1) *Struttura di un Label:*

esempio	FF	15	00
in generale	FF	XX	00
	indicatore	numero	limitatore del Label

Un Label è dunque lungo 3 byte. Per impostare un Label nello Junior-Computer si devono premere in tutto 6 tasti. Come si vede in fig. 3, il Label SUBTRA è stato sostituito da FF 15 00. In tal modo possiamo, dopo l'inizializzazione dell'Editor, introdurre i Label nel computer: non ci serve (per ora) una tastiera ASCII (tastiera alfa-numerica da macchina per scrivere).
Altri esempi di Label *validi* sono:

Tasti	Display	
F F 3 7 0 0	FF 37 00	il numero di Label è 37
F F F A 0 0	FF FA 00	il numero di Label è FA
F F 0 0 0 0	FF 00 00	il numero di Label è 00

Esempi di Label *non validi:*

Tasti	Display	
F F 2 1	FF 21 XX	manca il limitatore
F F 5 6 F F	FF 56 FF	limitatore errato
F F	FF 41 00	benché sul display dopo l'introduzione dell'indicatore appaia il corretto limitatore, occorre che esso venga soprascritto da 00. Solo dopo l'impostazione del Label completo il buffer di display viene copiato in memoria.

2) *Struttura di un'istruzione JSR:*

esempio	20	14	00
in generale	20	XX	00
	cod. OP di JSR	n° di subroutine	limitatore

Il richiamo di una subroutine è quindi lungo, come sappiamo, 3 byte. Rivediamo la fig. 3b. Nella subroutine HEXDEC viene richiamata due volte la subroutine COMNUM. Normalmente la relativa istruzione sarebbe: JSR-COMNUM. Alla subroutine COMNUM è stato assegnato il numero 14: naturalmente si sarebbe potuto scegliere un altro valore. 20 14 00 per lo Junior-Computer significa: salta all'inizio della subroutine numero 14. O più precisamente: salta alla subroutine che inizia con il numero di Label 14. Si noti

che (ad es.) FF 14 00 e COMNUM hanno il medesimo significato nel diagramma di flusso.
Altri modi *validi* di impostare un'istruzione JSR sono:

Tasti	Display	Significato
2 0 3 9 0 0	20 39 00	salta alla subroutine n° 39
2 0 9 A 0 0	20 9A 00	salta alla subroutine n° 9A
2 0 2 0 0 0	20 20 00	salta alla subroutine n° 20

Esempi di istruzioni JSR *non valide:*

Tasti	Display	Osservazioni
2 0 1 1	20 11 XX	manca il limitatore
2 0 7 1 9 F	20 71 9F	limitatore errato
2 0	20 28 00	benché sul display dopo l'introduzione del codice OP di JSR compaia *casualmente* il corretto n° di subroutine e il corretto limitatore, l'istruzione deve venire impostata completa. Solo dopo l'introduzione dell'istruzione JSR completa il contenuto del buffer di display viene copiato nella memoria del computer.

3) *Struttura di un'istruzione JMP:*

esempio	4C	11	00
in generale	4C	XX	00
	cod. OP di JMP	n° di Label	limitatore

Anche l'istruzione JMP, al solito, è lunga 3 byte. Nella routine principale DISPL si trova alla fine un'istruzione di salto incondizionato: JMP-DA, ossia: salta al Label DA. 4C 11 00 per lo Junior-Computer significa: salta al numero di Label 11. Anche in questo caso per il diagramma di flusso il Label DA e FF 11 00 sono equivalenti.
Altri modi *validi* di impostare un'istruzione JMP sono:

Tasti	Display	Significato
4 C 7 7 0 0	4C 77 00	salta al Label n° 77
4 C 2 9 0 0	4C 29 00	salta al Label n° 29
4 C 0 0 0 0	4C 29 00	salta al Label n° 00

Esempi di istruzioni JMP *non valide*:

Tasti	Display	Osservazioni
4 C 3 2	4C 32 XX	manca il limitatore
4 C 0 8 A A	4C 08 AA	limitatore errato
4C	4C 24 00	benché sul display dopo l'impostazione del codice OP di JMP compaiano

casualmente il corretto n° di Label a cui si deve saltare ed il corretto limitatore, l'istruzione deve venire impostata completa. Solo dopo l'impostazione dell'istruzione JMP completa il buffer di display viene copiato nella memoria del computer.

I numeri esadecimali posti nelle istruzioni JSR e JMP non sono quindi indirizzi assoluti, ma numeri di Label che il programmatore può scegliere liberamente. Gli effettivi indirizzi assoluti ai quali si deve saltare vengono calcolati automaticamente dallo Junior-Computer nella fase di "assemblaggio". Le istruzioni JSR e JMP sono lunghe, come sappiamo, 3 byte e richiedono il byte 00 quale limitatore. Di questa particolarità occorre *assolutamente* tener conto nell'impostazione d'un programma nello Junior-Computer!

4) *Struttura di un'istruzione di Branch*

La CPU 6502 possiede otto tipi di istruzioni Branch:

BPL, BMI, BEQ, BNE, BCC, BCS, BVC, BVS con i relativi codici OP 10, 30, F0, D0, 90, B0, 50, 70. Per tali istruzioni l'Assembler nel programma Monitor dello Junior-Computer calcola in modo automatico la lunghezza degli offset. Un'istruzione di Branch è strutturata come segue:

esempio	30	16
in generale	30	XX
	cod. OP di BMI	n° di Label

Altri esempi di istruzioni di Branch:

10 34	BPL al Label n° 34
30 F6	BMI al Label n° F6
F0 19	BEQ al Label n° 19
D0 56	BNE al Label n° 56
90 9D	BCC al Label n° 9D
B0 21	BCS al label n° 21
ecc	

I numeri al termine delle istruzioni di Branch non sono quindi degli offset, ma numeri di Label, che il programmatore può scegliere liberamente. Gli offset effettivi vengono calcolati dallo Junior-Computer in modo automatico nella fase di assemblaggio. Le istruzioni di Branch sono lunghe 2 byte e *non* richiedono il limitatore 00.

5) *Altre particolarità*

Nell'impostazione di un programma in modo Editor occorre badare a che i Label siano definiti in maniera univoca. Ciò significa che

un dato numero di Label può venire assegnato *per una sola volta* ad un Label. I numeri di Label possono venire introdotti nel computer in un ordine qualsiasi. Nell'introduzione di istruzioni Branch occorre inoltre curare che i salti non si estendano oltre il prescritto campo di -128 ...+127 passi. Altrimenti in questi casi l'assembler *non* emette messaggi di errore e calcola un valore *scorretto* di offset.

6) *JMP e JSR con indirizzo definito*

In vari casi un'istruzione JMP o JSR deve portare ad un ben definito indirizzo: ciò significa che il computer non deve assemblare tali istruzioni. In fig. 3a nel programma principale si effettua un salto alla subroutine GETBYT con l'indirizzo 1D6F. Questo indirizzo è definito nel programma Monitor dello Junior-Computer e non deve venir modificato durante l'assemblaggio! Nell'Assembler è perciò posto un efficiente filtro che impedisce l'assemblaggio di determinate istruzioni JMP e JSR:

- Un'istruzione JMP o JSR viene trattata dall'Assembler solo quando ad essa è assegnato un numero di Label.
- Una istruzione JMP o JSR *non* viene assemblata se ad essa non risulta univocamente assegnato un numero di Label.

Esempio: JSR-GETBYT = 20 6F 1D
cod. OP di JSR ADL ADH

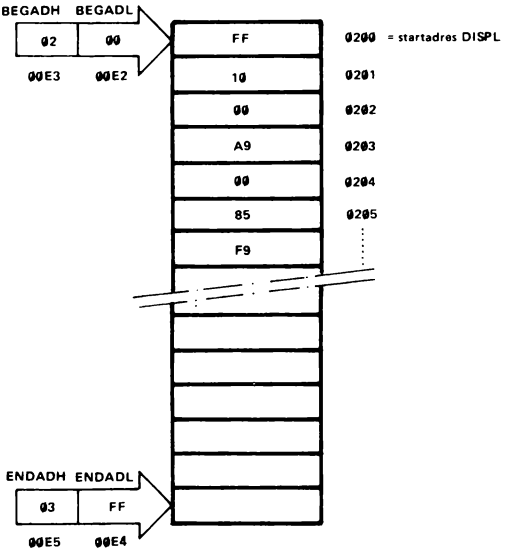


Figura 4. Ecco il modo in cui l'Editor introduce le prime istruzioni del programma DISPL di fig. 3a nella memoria operativa dello Junior-Computer. BEGAD ed ENDAD sono i Pointer che segnano i limiti della memoria operativa.

Il byte basso d'indirizzo, ADL, dell'istruzione JSR è 6F. Nell'esempio di programma di fig. 3 non risulta presente il numero di Label 6F: pertanto l'Assembler non attribuisce un indirizzo alla JSR-GETBYT, ma trascura questa istruzione.

7) *L'introduzione di altre istruzioni*

Prendiamo come esempio l'istruzione LDA.

Tasti	Display	Significato	Modo d'indirizzamento
A 9 7 F	A9 7F	LDA 7F	Immediate
A D 8 2 1 A	AD 82 1A	LDA-1A82	Absolute
A 5 E 6	A5 E6	LDAZ-E6	Zero Page
A 1 F A	A1 FA	LDA-(FA,X)	Indexed Indirect
B 1 F A	B1 FA	LDA-(FA),Y	Indirect Indexed
B D 0 0 0 2	BD 00 02	LDA-0200,X	Absolute Indexed, X
B 9 D 1 2 2	B9 D1 22	LDA-22D1,Y	Absolute Indexed, Y
B 5 3 1	B5 31	LDAZ-31,X	Zero Page Indexed, X

Quello che qui si è indicato per l'istruzione LDA resta naturalmente valido pure per tutte le altre istruzioni della CPU 6502. Per ogni istruzione lo Junior-Computer si calcola autonomamente la lunghezza; a tal fine viene impiegata la subroutine OPLEN del Monitor del sistema. Questa subroutine è quasi identica alla LENACC del volume 1. Mediante essa il computer gestisce il display in funzione della lunghezza delle istruzioni. Le istruzioni con Implied Addressing sono, come si sa, lunghe 1 solo byte. Perciò quando ne viene introdotta una di questo tipo si accende solo il campo del codice OP sul display, mentre resta spento il campo relativo all'operando.
Alle istruzioni che, come LDA, appartengono al punto (7) non è possibile assegnare indirizzi simbolici come per le JMP, JSR e le istruzioni di Branch. Ciò significa che gli indirizzi dai quali leggere o scrivere dati devono essere definiti dal programmatore prima dell'introduzione del programma.

8) *Definizione del campo di memoria*

Prima di far partire l'Editor occorre che sia stato chiaramente fissato il campo di memoria in cui il programma verrà depositato. Si richiedono a tale scopo due Pointer indirizzo, posti in Pagina 0 dello Junior-Computer. Il Pointer BEGADH, BEGADL (=BEGin ADdress High, BEGin ADdress Low) è memorizzato alle seguenti locazioni:

```
BEGADL  *$00E2
BEGADH  *$00E3
```

Il Pointer ENDADH, ENDADL (= END ADdress High, AND AD-

dress Low) è invece memorizzato nelle locazioni:

ENDADL *$00E4
ENDADH *$00E5

BEGAD punta quindi sempre all'indirizzo iniziale ed ENDAD a quello finale della zona di memoria in cui si vuole depositare il programma. Nell'Junior-Computer sono disponibili a tal fine due pagine consecutive, Page 2 e Page 3, con gli indirizzi da 0200 o 03FF. Si dispone così di 1/2 kbyte di memoria, uno spazio più che sufficiente per l'introduzione della maggior parte dei programmi. È quindi utile definire questo grosso campo di memoria consecutiva mediante i Pointer indirizzi, prima di impostare un programma:

BEGADH, BEGADL = 02 00 e
ENDADH, ENDADL = 03 FF

Per l'impostazione da tastiera dei Pointer si precede come segue:

AD 0 0 E 2	
DA 0 0	ADL del Pointer BEGAD
+ 0 2	ADH del Pointer BEGAD
+ F F	ADL del Pointer ENDAD
+ 0 3	ADH del Pointer ENDAD

I Pointer risultano posti in Pagina 0 dello Junior-Computer. Risulta così fissato il campo di memoria in cui sarà il programma: da 0200 a 03FF (Fig. 4).
Con questi Pointer è possibile pure definire altre zone di memoria nel modello standard dello Junior-Computer:

in Page 0	$ 0000 ... 00E0
in Page 1	$ 0100 ... 01F2
in Page 1A	$ 1A00 ... 1A79

In queste pagine tuttavia la memoria disponibile ha dimensioni limitate, perché una parte delle locazioni di memoria è impegnata dal programma Monitor e dallo Stack. Qualora si fuoriesca dal campo di memoria sopra definito, lo Junior-Computer emette un messaggio di errore: EEEEEE.

9) *L'inizializzazione dell'Editor*

Dopo aver definito il campo di memoria per il programma, si può inizializzare l'Editor: il suo indirizzo iniziale è 1CB5. Per "lanciare" l'Editor si premeranno quindi nell'ordine i tasti:

AD 1 C B 5
GO

Sul display, nel campo del codice OP, compare 77; i display dell'operando restano spenti. Il computer comunica in questo modo che l'Editor sta elaborando ed è pronto a depositare le

istruzioni che verranno impostate nel campo di memoria precedentemente definito.

10) *I tasti comando dell'Editor*

Subito dopo che l'Editor è stato inizializzato, le diciture AD, DA/+, PC e GO sui tasti-comando non sono più valide. Da questo punto in avanti diventano valide le seconde diciture riportate sui tasti-comando: SEARCH, INPUT, INSERT, DELETE e SKIP. Mediante questi comandi è possibile introdurre (INSERT, INPUT), riestrarre (DELETE) od anche ricercare (SEARCH) istruzioni. Inoltre è pure possibile saltare da istruzione ad istruzione: il relativo comando è SKIP. Vediamo ora in successione il significato di questi comandi.

INSERT

Tramite il comando INSERT è possibile inserire un'istruzione nella posizione precedente a quella visualizzata sul display.
Ogni volta che sul display compare il pseudo-codice OP 77 si deve *assolutamente* premere il tasto INSERT, quando si vogliano successivamente introdurre dati in forma di istruzioni nel computer. Perciò è necessario premere sempre per primo il tasto INSERT quando, dopo il lancio dell'Editor, si vuole introdurre un Label od un'istruzione nel computer.

INPUT

Tramite il comando INPUT è possibile inserire un'istruzione nella posizione immediatamente successiva a quella visualizzata sul display. Il programmatore può continuamente alternare, se desidera, i due comandi INPUT ed INSERT. Resta sempre valido l'ultimo tasto premuto.
Considerazioni generali sui comandi INSERT ed INPUT:
Quando si introducono Label od istruzioni mediante i tasti INSERT od INPUT, i successivi dati introdottivi premendo i tasti dati vengono trasferiti byte per byte nei buffer di display. Solo dopo che l'intera istruzione è stata inserita nel computer passa dai buffer display nella zona di memoria previamente definita. In tal modo è possibile "sovra scrivere" eventuali istruzioni digitate non correttamente mediante l'azionamento ripetuto dei tasti INSERT od INPUT. INSERT dispone l'istruzione corretta prima dell'istruzione visualizzata per ultima sul display. INPUT invece la posizione dopo l'ultima istruzione visualizzata. Il funzionamento dei tasti-comando INSERT ed INPUT dovrebbe così essere perfettamente chiaro. Ne deriva un metodo semplice, ma molto efficace, per editare od introdurre programmi nel computer.

DELETE

Mediante il comando DELETE si può eliminare dalla memoria dello Junior-Computer l'istruzione al momento visualizzata sul display. Le locazioni di memoria rese così libere vengono automaticamente riempite dal computer, nel senso che l'intero blocco dati che segue l'istruzione eliminata viene trasportata verso l'alto di un numero di byte pari a quelli eliminati. Quando si elimina un'istruzione dalla memoria non si formano quindi "lacune" nel programma. Dopo premuto il tasto DELETE, sul display compare l'istruzione successiva a quella cancellata. Premendo ripetutamente i tasti INSERT od INPUT, secondo necessità, si possono introdurre istruzioni prima o dopo quella visualizzata sul display.

SEARCH

Mediante il comando SEARCH si può effettuare la ricerca in memoria d'un determinato formato di 2 byte. Se ad esempio si premono i tasti SEARCH F F 1 1, il computer ricerca il Label n° 11 nella memoria e lo presenta sul display. Con il comando SEARCH è pure possibile manipolare il Pointer di display CURAD, anch'esso posto in Pagina 0. Il Pointer di display CURAD indica sempre il codice OP visualizzato al momento sul display dello Junior-Computer. CURADH e CURADL sono definiti come segue:

```
CURADL  *$ 00E6
CURADH  *$ 00E7
```

Il comando SEARCH pone il Pointer display CURAD, gestito in Software, all'indirizzo iniziale della zona di memoria previamente definita. Dato che ogni istruzione ha una determinata lunghezza, che il computer si calcola autonomamente, CURAD viene "trasportato" di codice OP in codice OP (non di locazione in locazione!). Con semplici operazioni di confronto il computer stabilisce se ha rintracciato la configurazione di 2 byte richiesta. In tal modo è possibile ricercare non solo Label, ma anche istruzioni qualsiasi in memoria, tramite il comando SEARCH:

```
SEARCH A 9 0 0          oppure
SEARCH 4 C 1 1          (fig. 3a) oppure
SEARCH 6 9 0 A          (fig. 3c)
```

Osservazione: Se per esempio l'istruzione A9 00 è presente più volte successive una dietro l'altra, con il comando SEARCH si può rintracciare solo la prima istruzione A9 00 a partire dal principio della memoria. Ossia, con il comando SEARCH non è possibile ricercare due istruzioni identiche, una successiva all'altra!

SKIP

Tramite il comando SKIP è possibile saltare di istruzione in istruzione. Questo comando viene utilizzato quando si desidera percorrere passo passo un programma impostato. Col comando SKIP si può compiere facilmente una rapida verifica di eventuali errori di digitazione.
Osservazione: il comando INPUT è una combinazione dei comandi SKIP più INSERT. Si hanno infatti le seguenti analogie:

SKIP INSERT XX	=	INPUT XX	(1 byte)
SKIP INSERT XXXX	=	INPUT XXXX	(2 byte)
SKIP INSERT XXXXXX	=	INPUT XXXXXX	(3 byte)

11) *Cold Start Entry / Warm Start Entry*

L'Editor può iniziare da due diversi indirizzi:

Cold Start Entry = 1CB5 e
Warm Start Entry = 1CCA.

Il Cold Start Entry è quello già noto. Se si inizializza l'Editor a partire da tale indirizzo, esso provvede ad attivare alcuni Pointer in Pagina 0 in funzione dello spazio di memoria richiesto dal programmatore. Prima di introdurre comunque un programma nel computer, occorre far partire l'Editor dall'indirizzo 1CB5.
Il Warm Start Entry consente invece al programmatore, dopo premuto il tasto RST, di rientrare nell'Editor. In questo caso tutti i Pointer in Pagina Zero *non* vengono nuovamente attivati e modificati. Per entrare nell'Editor in questo modo, dopo aver premuto il tasto RST, si azionano nell'ordine i tasti:

AD 1 C C A
GO

Sul display viene ora visualizzata l'ultima istruzione. Nel caso del Warm Start Entry i tasti-comando INSERT, INPUT, DELETE, SEARCH, SKIP restano validi, e funzionano normalmente.

12) *Impostazione da tastiera del programma di Fig. 3*

Premere i tasti:		**Display:**	**Significato**
RST		XX XX XX	Inizializzazione
AD	0 0 E 2	00 E2 XX	Posizionamento del Pointer BEGAD
DA	0 0	00 E2 00	
+	0 2	00 E3 02	
+	F F	00 E4 FF	Posizionamento del Pointer ENDAD
+	0 3	00 E5 03	
AD	1 C B 5	1C B5 20	
GO		77	Lancio dell'Editor (Cold Start Entry)

INSERT	F F 1 0 0 0	FF 10 00	Introduzione del n° di Label 10
INPUT	A 9 0 0	A9 00	LDA # 00
INPUT	8 5 F 9	85 F9	STAZ-INH
INPUT	8 5 F A	85 FA	STAZ-POINTL
INPUT	8 5 F B	85 FB	STAZ-POINTH
INPUT	F F 1 1 0 0	FF 11 00	N° di Label 11
INPUT	2 0 6 F 1 D	20 6F 1D	JSR-GETBYT (6F non è un n° di Label!)
INPUT	1 0 1 0	10 10	BPL al Label n° 10
INPUT	8 5 E	85 XX	Errore di digitazione! L'istruzione non è stata ancora memorizzata
INPUT	8 5 F 9	85 F9	STAZ-INH
INPUT	8 5 D 7	85 D7	STAZ-HEXL
INPUT	2 0 1 2 0 0	20 12 00	JSR-HEXDEC (HEXDEC ha il n° 12)
INPUT	4 C 1 1 0 0	4C 11 00	JMP-DA (DA ha il n° 11)
INPUT	F F 1 2 0 0	FF 12 00	(HEXDEC ha n° di Label 12)
INSERT			Errore di digitazione! (INSERT invece di INPUT)
INPUT	2 0 1 4 0 0	20 14 00	JSR-COMNUM (COMNUM ha il n° 14)
INPUT	8 5 F A	85 FA	STAZ-POINTL
INPUT	8 4 D 7	84 D7	STYZ-HEXL
INPUT	2 0 1 4 0 0	20 14 00	JSR-COMNUM (COMNUM ha il n° 14)
INPUT	A 2 0 4	A2 04	LDX # 04
INPUT	F F 1 3 0 0	FF 13 00	HD (HD ha n° di Label 13)
INPUT	0 A C A	0A EEEEEE	Premere un tasto di comando, altrimenti segnala errore!
INPUT	C A	CA	DEX
INPUT	D 0 1 3	D0 13	BNE al Label n° 13
INPUT	0 5 F A	05 FA	ORAZ-POINTL
INPUT	8 5 F A	85 FA	STAZ-POINTL
INPUT	8 4 F B	84 FB	STYZ-POINTH
INPUT	6 0	60	RTS
SEARCH	F F 1 3	FF 13 00	ricerca il Label n° 13
SKIP		0A	
SKIP		CA	
SKIP		D0 13	
SKIP		05 FA	
SKIP		85 FA	
SKIP		84 FB	
SKIP		60	
SKIP		77	77 significa: premere il tasto INSERT!!!!!!
INSERT	F F 1 4 0 0	FF 14 00	COMNUM ha il n° di Label 14
INPUT	8 4 D 8	84 D8	STYZ-HEXH

Nota: Per errore si è scordata l'istruzione LDA # 00. È facile tuttavia reinserire questa istruzione: a tal fine si impiega il comando INSERT, così:

INSERT	A 9 0 0	A9 00	LDA 00
SKIP		84 D8	STYZ-HEXH Si è così realmente inserita l'istruzione LDA # 00 prima di STYZ-HEXH. Il computer ha dovuto svolgere un bel po' di lavoro, risparmiandolo al programmatore. Ora possiamo proseguire ancora col comando INPUT, perché dopo 84 D8 occorre inserire una nuova istruzione; così:

INPUT	2 0 1 5 0 0	20 15 00	SUBTRA ha il n° 15
INPUT	1 8	18	CLC
INPUT	A 5 D 7	A5 D7	LDAZ-HEXL
INPUT	6 9 0 A	69 0A	ADC # 0A
INPUT	6 0	60	RTS;
INPUT	F F 1 5 0 0	FF 15 00	SUBTRA ha il Label n° 15
INPUT	3 8	38	SEC
INPUT	A 5 D 7	A5 D7	LDAZ-HEXL
INPUT	E 9 0 A	E9 0A	SBC # 0A
INPUT	8 5 D 7	85 D7	STAZ-HEXL
INPUT	A 5 D 8	A5 D8	LDAZ-HEXH
INPUT	E 9 0 0	E9 00	SBC # 00
INPUT	3 0 1 6	30 16	BMI al n° di Label 16
INPUT	C 8	C8	INY
INPUT	4 C 1 5 0 0	4C 15 00	JMP-SUBTRA (SUBTRA ha il Label n° 15)
INPUT	F F 1 6 0 0	FF 16 00	SUB ha il Label n° 16
INPUT	6 0	60	RTS

Così abbiamo introdotto il programma nello Junior-Computer. È consigliabile verificare il programma impostato per eventuali errori di digitazione. Per questo si procede come segue:

SEARCH F F 1 0
SKIP
SKIP
.
.
.
SKIP

Solo dopo aver controllato che il programma impostato corrisponde al diagramma di flusso si può far partire l'Assembler.

L'Assembler

Grazie all'Editor, abbiamo introdotto dunque nello Junior-Computer un programma che contiene indirizzi simbolici. Questi indirizzi simbolici sono numeri di Label, numeri di subroutine o Label a cui occorre effettuare salti condizionati o non.
Un programma con indirizzi simbolici non è idoneo per lo Junior-Computer, dato che il microprocessore non può trattarli. L'Assembler deve quindi conformare il programma impostato in modo che il microprocessore 6502 possa elaborarlo. Le funzioni che l'Assembler deve svolgere possono quindi definirsi come segue:

- Tutti i Label devono venire eliminati dal programma.
- Alle istruzioni JSR occorre assegnare gli effettivi indirizzi assoluti.
- Lo stesso vale per le istruzioni JMP.
- Alle istruzioni di Branch devono essere assegnati gli offset.
- Al termine dell'assemblaggio, il programma deve trovarsi depositato nella memoria di programma del computer in una forma interpretabile dalla CPU 6502.

Come si presenta il programma in memoria prima dell'assemblaggio?

L'indirizzo di partenza del nostro programma è stato fissato in 0200. La fig. 5 mostra la forma sotto cui l'Editor ha deposto il programma nella memoria dello Junior-Computer. Dato che per ogni Label sono state riservate 3 posizioni di memoria, il programma impostato risulta sensibilmente più lungo del programma dopo assemblato. Ora è il momento di far partire l'Assembler. Il suo indirizzo iniziale è 1F51:

RST	Richiamo del programma Monitor
AD 1 F 5 1	Indirizzo di partenza dell'Assembler
GO	Lancio dell'Assembler

Dopo la pressione del tasto GO il display si spegne per alcuni secondi: è il tempo che necessita allo Junior-Computer per assemblare il programma. Se il programma è lungo, il display può restare spento vari secondi. Appena il programma è stato assemblato, il computer si rifà vivo tramite il programma Monitor. Da questo momento tornano ad essere valide le diciture AD, DA, PC, + e GO dei tasti comandi.

Come si presenta il programma in memoria dopo l'assemblaggio?

Lo Junior-Computer dispone di un "Two-Pass" Assembler. Che vuol dire? L'assemblaggio del programma viene eseguito dal computer non in una, ma in due fasi o meglio richiede due passaggi:

1.a fase

Il computer legge il programma impostato (Fig. 5). In questa fase gli interessano *solo* i Label introdotti dal programmatore. Quando incontra lungo il programma un Label (caratterizzato da FF), il computer memorizza il numero di Label e l'indirizzo a cui il Label si trova su di un Symbol-Stack. Questo non è altro che una Look-Up Table che il computer allestisce da sé, senza che il programmatore debba curarsene.
Dopo aver conservato il n° di Label e l'indirizzo assoluto a cui si trova il Label sul Symbol-Stack, il programma provvede a cancellare tutti i Label presenti nel programma. Come procede a questo? Come sappiamo, un Label ha una lunghezza di 3 byte. Perciò il computer provvede a spostare di 3 byte verso l'alto l'intero programma seguente. Le istruzioni successive al Label perciò lo cancellano per sovrascrittura, ed il Label risulta eliminato dal programma.
Questo processo si ripete in corrispondenza ad ogni Label. Il processore ripercorre nuovamente da cima a fondo il programma,

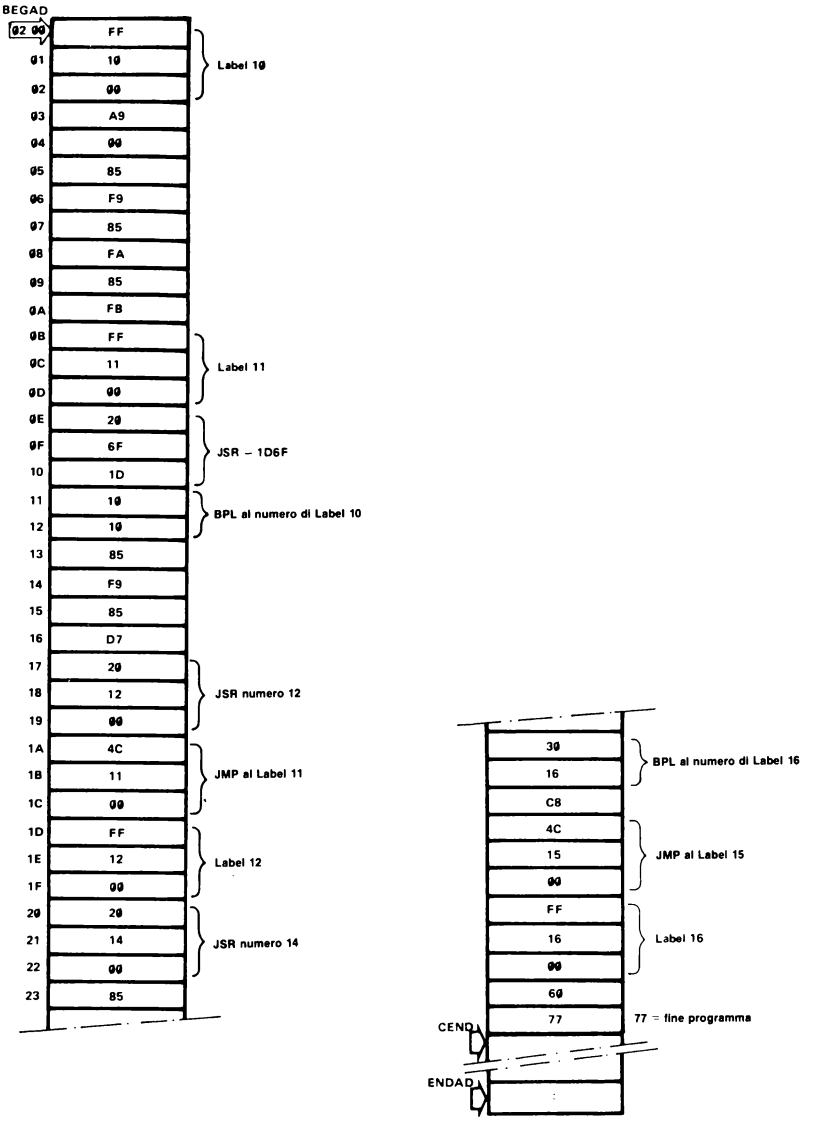


Figura 5. La memoria operativa principia da BEGAD e termina a ENDAD. Lo spazio di memoria fra BEGAD ed ENDAD contiene il programma impostato, che nell'esempio inizia col Label FF 10 00 e termina col carattere EOF 77. La zona di memoria fra BEGAD ed ENDAD si chiama File. Qui il file contiene tutte le istruzioni utilizzate nel programma, anche gli indirizzi col pseudocodice OP FF, i cosiddetti Label. Un programma così non può essere "digerito" dal computer, perché non sono indicati gli offset e gli indirizzi assoluti di salto. Solo il successivo assemblaggio rende il programma introdotto mediante l'Editor "gradito" alla CPU 6502.

partendo ogni volta dall'indirizzo (nel nostro esempio) 0200, sin quando non ritrova un Label. Anche questo viene salvato con n° di Label ed indirizzo assoluto sul Symbol-Stack. Il tutto viene ripetuto fino a quando tutti i Label sono stati eliminati dal programma impostato. Ogni informazione relativa ai Label di cui il computer necessita è adesso deposta sul Symbol-Stack. Il Symbol-Stack è un registro in Software lungo 256 byte, per cui vi si possono memorizzare un massimo di 256:3 = 85 Label. Si possono quindi mediante l'Editor inserire programmi con non più di 85 Label! Un numero normalmente più che sufficiente per l'assemblaggio di un programma.

2.a fase

Ora l'Assembler ripercorre ancora una volta dalla cima, partendo dall'indirizzo 0200, l'intero programma. Come sappiamo, tutti i Label sono stati eliminati. L'Assembler a questo punto si interessa *esclusivamente* delle seguenti istruzioni:

- JMP
- JSR
- Istruzioni di Branch quali BPL, BMI, BEQ, BNE, BCC, BCS, BVC e BVS.

Dopo queste istruzioni *si trovano ancora* i numeri dei Label a cui si deve saltare. Quando il computer incontra un'istruzione JMP o JSR, verifica il numero di Label posto dopo tale istruzione. Ad ogni n° di Label corrisponde un indirizzo assoluto. Questo indirizzo assoluto il computer se lo va a prendere dal Symbol-Stack. In che modo rintraccia questi indirizzi assoluti sul Symbol-Stack? Rammentiamoci che nella prima fase di assemblaggio il computer ha salvato sul Symbol-Stack i numeri di Label e gli indirizzi assoluti dei Label. Adesso il computer ricerca nel Symbol-Stack il numero di Label posto dopo l'istruzione JMP o JSR *in assemblaggio al momento*. Rintracciato sul Symbol-Stack questo n° di Label, estrae dallo stesso Stack i 2 byte indirizzo, alto e basso, relativi al n° di Label, e li pone dopo l'istruzione JMP o JSR *in assemblaggio*. Cioé, il n° di Label ed il limitatore, che seguono tale istruzione, vengono sovrascritti dai byte indirizzo alto e basso del relativo indirizzo assoluto. Questo processo si ripete tante volte quanto necessario perché vengano assemblate tutte le istruzioni JMP e JSR.

Ora restano da calcolare i valori degli offset per le istruzioni di Branch (salti condizionati). Perciò, nella seconda fase, il microcomputer ricerca anche tutte le istruzioni di Branch introdotte dal programmatore. Dopo queste istruzioni di Branch si trovano i numeri di Label a cui il salto deve aver luogo. Il computer può calcolare facilmente l'indirizzo a cui sta l'istruzione di salto. L'indirizzo a cui saltare si trova inoltre sul Symbol-Stack. Per il calcolo dell'offset si ha:

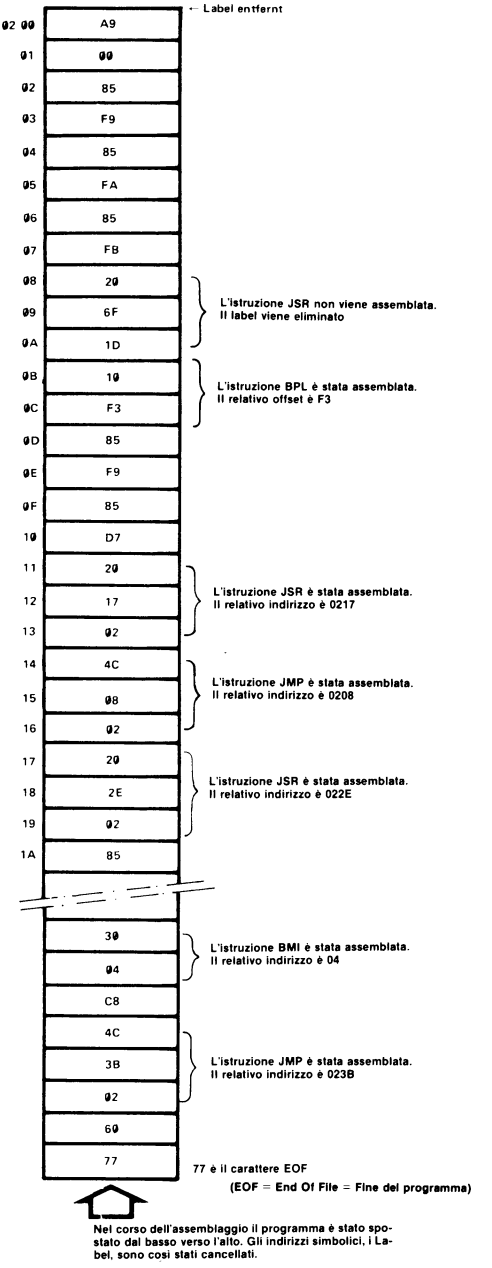


Figura 6. Dopo l'assemblaggio tutti i Label sono stati eliminati dal file. Il file si è corrispondentemente accorciato. Dopo i codici OP delle istruzioni di salto incondizionato ora stanno gli indirizzi assoluti corretti. Anche le istruzioni di Branch recano vicini i relativi offset. In questa forma il programma può essere elaborato dalla CPU 6502.

Destination - Source + 2 = Offset, ovvero traducendo:
Indirizzo del Label dove effettuare il salto *meno* indirizzo a cui si trova l'istruzione Branch *più* due *eguale* valore dell'offset. Il programma di fig. 3 dopo assemblaggio è illustrato in Fig. 6
Così abbiamo descritto anche la gestione dell'assembler ed il suo meccanismo. Un programma assemblato può venire pure (ri)percorso con l'Editor. In tal caso bisogna però badare che:
- l'Editor venga avviato *solo* mediante Warm Start Entry;
- i pointer in Pagina 0 devono essere attivati a mano:
  BEGAD = CURAD
  ENDAD = CEND (CEND = Current END Address)

Rifacendoci all'esempio di fig. 3, ecco i tasti da premere per ripercorrere con l'Editor un programma *già assemblato*:

AD	0	0	E	6	
DA	0	0			disponi il Pointer CURAD a 0200
+	0	2			
+	F	F			disponi il Pointer CEND a 03FF
+	0	3			
AD	1	C	C	A	Warm Start Entry
GO					Lancio dell'Editor

Con questa procedura è possibile percorrere un programma assemblato con il tasto SKIP, da cima a fondo. Tramite il comando SEARCH il programmatore è in grado di ricercare determinate istruzioni nel programma assemblato. I tasti INSERT, INPUT e DELETE dovrebbero venire impiegati solo nel caso in cui, inserendo od eliminando istruzioni dalla memoria del computer, non vengono modificati gli indirizzi assoluti e la lunghezza dei salti calcolati dal computer.

Indirizzi importanti per l'Editor e l'Assembler

1) *Editor*

Cold Start Entry : $1CB5
Warm Start Entry : $1CCA

I pointer dell'Editor:

BEGADL	*	$00E2	Begin Address Pointer
BEGADH	*	$00E3	
ENDADL	*	$00E4	End Address Pointer
ENDADH	*	$00E5	
CURADL	*	$00E6	Current Address Pointer
CURADH	*	$00E7	
CENDL	*	$00E8	Current End Address Pointer
CENDH	*	$00E9	

2) *Assembler*

Indirizzo iniziale : $1F51

3) *Altre possibilità dell'Editor ed Assembler:*
Il tasto ST della tastiera dello Junior-Computer è collegato alla linea NMI. Questa linea è riferita al vettore NMI che può essere liberamente scelto dal programmatore. Il vettore NMI è posto nelle seguenti locazioni di memoria:

NMIL	* $1A7A	byte basso d'indirizzo
NMIH	* $1A7B	byte alto d'indirizzo

Caricando gli indirizzi di partenza in queste posizioni di memoria, il programmatore può lanciare l'Editore *oppure* l'Assembler premendo il tasto ST:

```
AD 1 A 7 A      Cold Start Entry via tasto ST
DA B 5
+  1 C
```

oppure:

```
AD 1 A 7 A
DA C A          Warm Start Entry via tasto ST
+  1 C
```

oppure

```
AD 1 A 7 A      Lancio dell'Assembler via tasto ST
DA 5 1
+  1 E
```

Il quinto capitolo ci ha dunque mostrato come si lavora con l'Editor e l'Assembler dello Junior-Computer. Grazie a queste particolarità del programma Monitor è possibile introdurre programmi nel computer velocemente ed in modo corretto. Il computer allevia il programmatore di un bel po' di lavoro. Solo tramite l'Editor e l'Assembler è possibile introdurre in memoria nello Junior-Computer programmi senza lacune. Tutto questo ha particolare importanza quando è disponibile solo un limitato spazio di memoria.

6

PIA

Il Peripheric interface Adapter

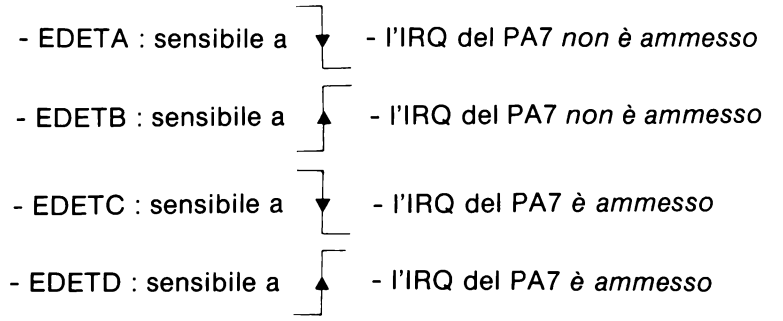
Lo Junior-Computer dispone di un Peripheric Interface Adapter sistemato, come la CPU, in un circuito integrato a 40 pin. Attraverso questo adattatore, abbreviato in PIA, corre tutto lo scambio di dati col mondo esterno dello Junior-Computer. Questo mondo esterno, da cui il computer riceve dei dati o a cui deve trasmetterne, può assumere diverse facce: una tastiera esadecimale, un display a più cifre a 7 segmenti, una tastiera ASCII, un videodisplay come l'ELEKTERMINAL, una stampante od anche il servomeccanismo di un modello di aereo. Questo capitolo descrive il modo in cui, tramite il PIA, il computer gestisce i dispositivi esterni collegati oppure riceve ed elabora dati.

Un computer privo di ingressi (input) ed uscite (output) è una macchina priva di utilità. La persona a cui il computer dovrebbe facilitare il lavoro non può partecipare né intervenire sullo svolgimento del programma nella macchina, se non ha a disposizione input ed output. Il computer non è in grado di comunicare. È una situazione analoga a quella di insegnanti o professori dotati di grande sapere, ma che non sono capaci o non vogliono trasmettere le loro conoscenze agli allievi o studenti. Per i computer è la stessa cosa: quanto meno un computer è in grado di tenere in conto l'inesperienza o l'ignoranza del programmatore, tanto meno risulta utile agli uomini.
In che modo dunque un computer può comunicare con il mondo esterno? Lo scambio di dati fra uomo e computer copia esattamente lo scambio di informazioni fra due persone. Per comunicare pensieri o idee memorizzate nel cervello l'uomo si serve della parola. Anche in un computer si presentano "pensieri" ed "idee" depositati in memoria sotto forma di dati elaborati. Per poter comunicare questi dati al mondo esterno, il computer si serve di una stampante, o in modo più semplificato di un display a 6 cifre, come lo Junior-Computer.
Un colloquio fra due persone ha senso solo se oltre a parlare si ascolta: le due funzioni hanno entrambe la stessa importanza! Ascoltare, d'altra parte, significa nient'altro che ricevere informa-

zioni dal compagno di discorso. Al fine di poter ricevere dati dal mondo esterno, un computer si serve di una tastiera: per grossi calcolatori si tratta di una tastiera alfa-numerica ASCII; per lo Junior-Computer una tastiera esadecimale.
In modo analogo all'uomo che parla con la bocca ed ascolta con l'orecchio, lo Junior-Computer trasmette e riceve tramite il PIA. Trasmissione e ricezione implicano che il PIA (Peripheric Interface Adapter) deve possedere degli ingressi e delle uscite: ingressi per ricevere dati, uscite per trasmettere dati. Per dati si devono intendere come al solito tensioni elettriche corrispondenti agli stati logici 0 od 1, e che possono variare nel tempo. Il PIA dello Junior-Computer è costituito da un CI 6532. Questo elemento, di tipo LSI, contiene oltre al PIA 128 celle di RAM, un timer, programmabile in modo universale, un registro-Flag ed un rivelatore di fronti, che reagisce ai fronti positivi e negativi degli impulsi. Questi registri del 6532 si presentano al programmatore come normali celle di memoria, in cui il microprocessore può effettuare operazioni sia di lettura che di scrittura. Per completezza, citeremo che il CI 6532, oltre alle 128 locazioni di memoria, possiede 19 *registri speciali*, i cui modi di programmazione e di funzionamento verranno esaminati meglio nel corso di questo capitolo.
La tabella seguente dà una prima informazione sulle denominazioni di questi registri particolari:

1. Porta A e Porta B
   - PAD : Port A Data Register (Registro dati)
   - PADD : Port A Data Direction Register (registro direzionamento dati)
   - PBD : Port B Data Register
   - PBDD : Port B Data Direction Register
2. Avvio del Timer (NB: *non è ammesso* l'IRQ del timer)
   - CNTA : CLK1T (Fattore di divisione ÷ 1)
   - CNTB : CLK8T (Fattore di divisione ÷ 8)
   - CNTC : CLK64T (Fattore di divisione ÷ 64)
   - CNTD : CLK1KT (Fattore di divisione ÷ 1024)
3. Il registro-Flag e le locazioni del Timer:
   - RDFLAG : leggi il registro-Flag
   - RDTEN : leggi la cella del timer, enable (= abilita) IRQ del timer
   - RDTIS : leggi la cella del timer, disable (= inibisci) IRQ del timer
   - CNTE : CLK1T (Fattore di divisione ÷ 1)
   - CNTF : CLK8T (Fattore di divisione ÷ 8)
   - CNTG : CLK64T (Fattore di divisione ÷ 64)
   - CNTH : CLK1KT (Fattore di divisione ÷ 1024)

4. PA7 rivelatore dei fronti



Anche se la maggior parte dei concetti di questa tabella non sono ancora noti, è possibile ricavarne le denominazioni di queste particolari celle della memoria e la struttura di questo capitolo. (Osservazione: alcune locazioni di memoria possono risultare sovrapposte; ossia, ad uno stesso indirizzo possono essere riferite due celle di memoria di diversa denominazione. Ciò può provocare imbarazzo. Pertanto, ad ogni nome simbolico d'una cella di memoria è assegnato un diverso indirizzo: vengono così esclusi scambi fra locazioni di memoria e possibili doppi indirizzamenti).

Le Porte A e B

Il CI 6532 dispone di due porte: la Porta A e la Porta B. Le 2 porte insieme formano il Peripheric Interface Adapter, PIA. Questo organo consiste di 16 collegamenti che escono da 16 piedini del CI 6532. Il programmatore può connettere al PIA altri dispositivi, ad esempio una stampante. Nello Junior-Computer è previsto un connettore a spine sul quale giungono le 16 linee del PIA: il connettore delle porte. In fig. 1 si può vedere come siano collegati la tastiera ed il display al PIA. Non tutte le 16 linee del PIA risultano accessibili all'utente, quando si devono impiegare la tastiera ed il display. Restano in tal caso liberamente disponibili le linee da PB7 e PB5 nonché PB0 e PA7. Se invece la tastiera ed il display dello Junior-Computer non vengono utilizzate nei programmi, restano disponibili all'utente tutte le linee delle porte.

Il modello di programmazione del PIA

Analogamente al modello di programmazione del microprocessore illustrato nel 1° volume (capitolo 3, fig. 1b), è possibile dare anche un modello di programmazione del PIA. Si veda la fig. 2. Il PIA è collegato al microprocessore tramite tre Bus: il Bus indirizzi, il Bus dati ed il Bus di controllo. Esso è quindi collegato alla CPU come un normale elemento di memoria. I 3 bus citati funzionano nel modo seguente:

- Il bus indirizzi seleziona le locazioni interne di memoria.

- Sul Bus dati si svolge, in due direzioni, lo scambio di dati fra la CPU ed il PIA. La direzione dello scambio dati è fissata dalla linea R/$\overline{W}$ del processore.
- Sul bus di controllo la CPU invia al PIA i seguenti segnali:
  * il segnale R/$\overline{W}$
  * il segnale CS
  * il segnale di clock Φ 2
  * ed il segnale RES (Reset)

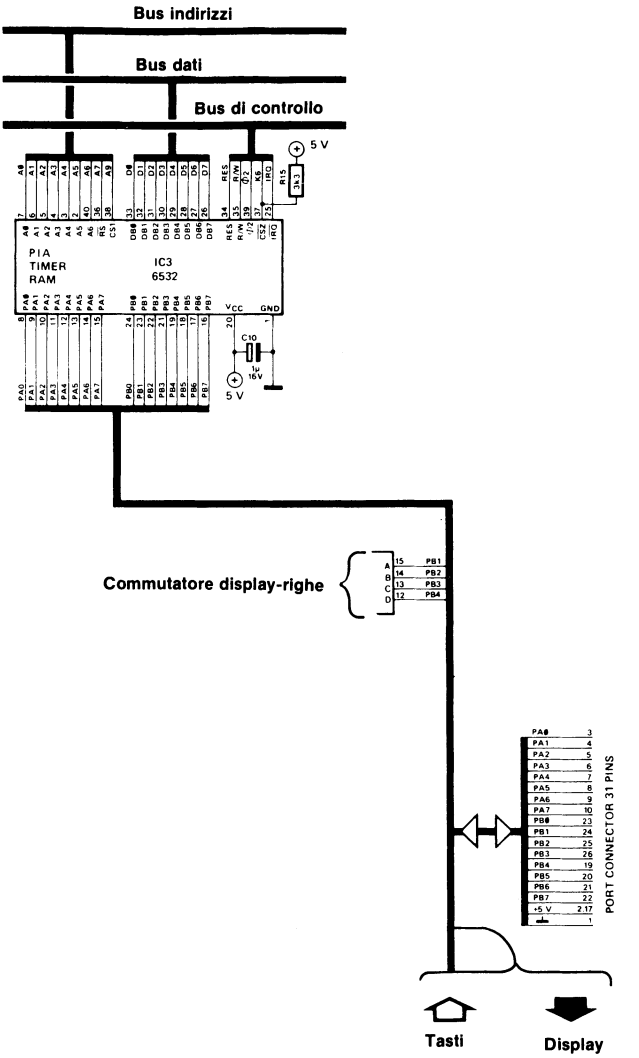
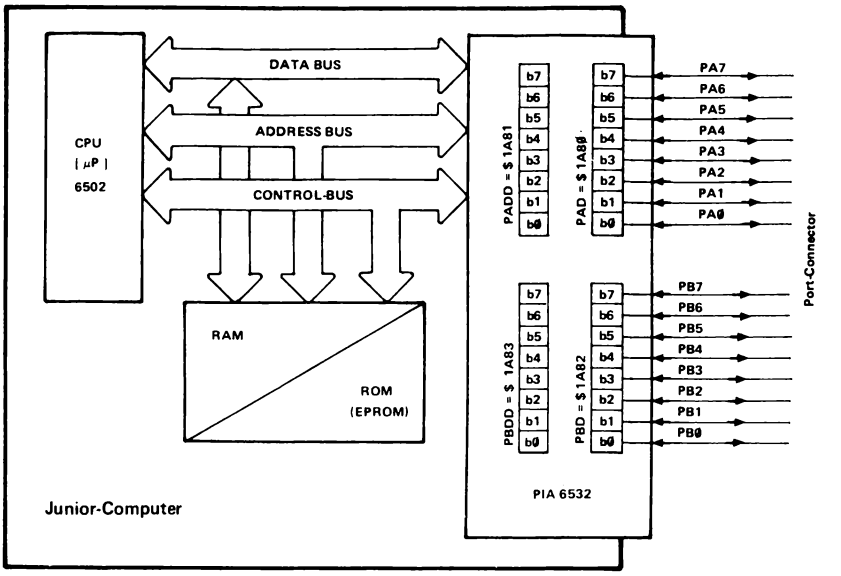


Figura 1. Il PIA è collegato alla CPU tramite i bus indirizzi, dati e di controllo. Il bus indirizzi seleziona diverse locazioni di memoria del CI 6532. Non tutte le linee di porta sono liberamente disponibili all'utente: alla Porta A e Porta B sono collegati la tastiera ed il display. Tutte e 16 le linee di porta sono collegate al connettore delle porte.



PAD = Port A Data Register

PADD = Port A Data Direction Register

PBD = Port B Data Register

PBDD = Port B Data Direction Register (Übersetzung siehe Text)

Figura 2. Il PIA (=Peripheric Interface Adapter) consta di quattro registri e le due porte PORT A e PORT B. Una Porta è composta da un fascio di 8 collegamenti, tramite i quali il microcomputer trasmette dati al mondo esterno o ne riceve. Ogni linea di porta può venir programmata, indipendentemente dalle altre, come ingresso o come uscita. È la configurazione di bit nei registri direzionamento dati PADD e PBDD a determinare se una linea di porta è definita ingresso od uscita. Uno "0" in una data posizione di bit nel registro direzionamento dati definisce la corrispondente linea di porta quale ingresso (input), un "1" quale uscita (output). Tramite il bus dati il processore può leggere i registri dati o scrivere in essi. In una operazione di lettura il processore legge la configurazione di bit sulle linee di porta. Se la CPU scrive un formato di bit nei registri dati PAD e PBD, lo stesso formato di bit compare sulle linee di porta definite uscite. Dopo l'operazione di scrittura il formata di bit è mantenuto sulle linee di porta (funzione di "latch" di una linea di porta definita output). Il PIA è collegato alla CPU tramite i Bus indirizzi, dati e di controllo.

Tramite il bus di controllo risultano pure collegati fra loro la linea di IRQ del PIA e la CPU dello Junior-Computer. Il PIA tramite la linea IRQ è quindi in grado di generare un IRQ. Come ciò avvenga in dettaglio, lo spiegheremo parlando del Timer e della programmazione di PA7 (punti 2,4 e 5).
Ci sono così noti tutti i collegamenti elettrici fra PIA e microprocessore, e nulla più osta alla descrizione del modello di programmazione. Descrivendo il Peripheral Interface Adapter non è più possibile separare Hardware e Software. Al programmatore quindi il PIA si presenta nel modo seguente:

- *Hardware:* 16 linee, programmabili indipendentemente come ingressi o come uscite. Queste 16 linee si suddividono su 2 Porte: Porta A e Porta B. Ciascuna Porta comprende 8 linee. Si usa anche dire che le Porte A, B sono "larghe" 8 bit ciascuna.

Le linee delle porte recano i nomi da PA0 a PA7 e da PB0 a PB7. Dalle singole linee di porta esce il Bus dati. Dato che sono disponibili 2 porte, il bus dati è commutabile a scelta sulla Porta A o sulla B.
Poiché il bus dati è di tipo bidirezionale, la CPU tramite il bus dati può scrivere dati sulle linee di porta, ovvero leggere dati, presenti sotto forma di tensioni elettriche (allo stato logico 0 od 1) sulle linee di porta. Quando il microprocessore "scrive" su linee di porta, previamente definite quali uscite, una data configurazione di bit, tale formato di bit si mantiene su dette linee output di porta anche dopo il termine dell'operazione di scrittura (*funzione "Latch"* di una linea di uscita).
Quando il microprocessore "legge" una o più linee di porta, definite previamente quali ingressi, esso misura in effetti il livello istantaneo di tensione sulle linee di porta. Se la tensione su una data linea di porta ha un livello minore di 0,4 V, la CPU legge "0"; se la tensione sulla linea di porta è maggiore di 2,4 V, per la CPU vale "1" logico (*non* vi è una *funzione* di *Latch* per le linee di ingresso).

- *Software:* 4 registri nei quali è possibile sia scrivere che leggere dati.
  Le denominazioni di questi quattro registri interni sono:

PAD	= Port A Data Register (registro dati)
PADD	= Port A Data Direction Register (registro direzionamento dati)
PBD	= Port B Data Register
PBDD	= Port B Data Direction Register

Come lascia intendere la denominazione di questi registri, ogni Porta consta di 2 celle di memoria:

1. un registro dati, e
2. un registro di direzionamento dei dati.

Il registro di direzionamento dei dati governa per ogni Porta la direzione del flusso dei dati attraverso le linee della porta. Poiché questo registro ha la capacità ("larghezza") di 8 bit, la CPU può, tramite il bus dati, scrivere una "parola" nel registro direzionamento dati. Questa parola è una data configurazione di zeri e di uno. Tale configurazione di 0 ed 1 nel registro direzionamento dati agisce sulle linee di porta nel modo che segue:
- Uno 0 o in un dato bit del registro direzionamento dati programma la corrispondente linea di porta quale input.
- Un 1 in un dato bit del registro direzionamento dati programma la corrispondente linea di porta quale output.

Ad entrambi i registri di una porta sono assegnati indirizzi, che per lo Junior-Computer sono i seguenti:

PAD	ha l'indirizzo 1A80
PADD	ha l'indirizzo 1A81
PBD	ha l'indirizzo 1A82
PBDD	ha l'indirizzo 1A83

Gli esempi che seguono serviranno ad illustrare in pratica la programmazione delle porte. Per prima cosa vediamo come programmare quali ingressi o quali uscite alcune linee di porta:
1. Tutte le linee della Porta A devono risultare ingressi e tutte le linee della Porta B quali uscite. Ecco il programma:

LDA # 00	tutti i bit dell'Accu vengono azzerati
STA-PADD	le linee PA0...PA7 sono dichiarate ingressi
LDA # FF	tutti i bit dell'Accu sono 1
STA-PBDD	le linee PB0...PB7 sono dichiarate uscite

Come già detto, sono i singoli bit nel registro direzionamento dati a determinare se una linea di porta viene definita input od output. Nel registro direzionamento dati PADD tutti i bit sono 0. Perciò tutte le linee della Porta A sono programmate quali input. A questo punto la CPU può leggere nell'Accu, nel registro X o in quello Y i segnali, in conformazione di 1 e 0, presenti sulle singole linee di porta. Ne riparleremo comunque più avanti. Nel registro di direzionamento dati PBDD tutti i bit sono 1. Perciò tutte le linee della porta B sono programmate quali output. La CPU, tramite il bus dati, può così scrivere una determinata configurazione di bit sulle linee di questa porta. Rimandiamo anche questo tipo di descrizione a più oltre.
2. Vogliamo che le linee di porta PA4 e PB0 vengano definite quali uscite, mentre tutte le altre linee di porta devono agire da input. Scriviamo su una riga la disposizione necessaria dei bit, così come devono essere introdotti nei due registri direzionamento dati:

PADD	b7	b6	b5	b4	b3	b2	b1	b0	
	0	0	0	1	0	0	0	0	= $10
PBDD	b7	b6	b5	b4	b3	b2	b1	b0	
	0	0	0	0	0	0	0	1	= $01

Il programma prende il seguente aspetto:

LDA # 10	carica il formato di bit nell'Accu
STA-PADD	la linea PA4 è dichiarata output
LDA # 01	carica il formato di bit nell'Accu
STA-PBDD	la linea PB0 è dichiarata input; tutte le rimanenti linee delle Porte A e B sono programmate quali input.

Lettura e scrittura sulle linee di porta

Sinora abbiamo imparato che le linee di porta possono venire programmate quali ingressi o quali uscite con l'ausilio dei registri direzionamento dati. Come fa in effetti il microprocessore a leggere od a scrivere sulle porte i dati (sotto forma di 0 ed 1)? A tale scopo sono adibiti i registri-dati PAD e PBD. L'indirizzo di PAD è 1A80, quello di PBD è 1A82.

Lettura delle linee di porta PA0...PA7 e PB0...PB7 (Input Mode "I")

Quando vengono letti i registri-dati PAD e PBD, il formato di bit giacente sulle linee di porta viene trasferito, tramite il bus dati, in uno dei registri interni della CPU. Ossia, è come se le linee di porta costituissero un prolungamento del bus dati. *Se il processore vuole effettuare la lettura di una o più linee di porta, queste devono previamente essere programmate quali ingressi.* Lo si vede bene con il seguente esempio:

```
LDA # 00
STA-PADD     le linee PA0...PA7 sono programmate quali input
LDX-PAD      il formato di bit presente sulle linee di porta viene
             depositato nel registro X della CPU.
```

Se sulle linee di porta A è presente il formato di bit 10101110

PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0	linee della Porta A
1	0	1	0	1	1	1	0	formato bit su PA7... PA0

allora nel registro X si trova AE. Se la tensione su una linea di porta è inferiore a 0,4 V, il processore la interpreta come uno 0 logico; se è superiore a 2,4 V come un 1 logico.
Osservazione: se una linea di porta programmata quale input viene pilotata da un gate, un transistor o un altro circuito logico, questi elementi debbono essere in grado di fornire una corrente di almeno 1,6 mA. Ciò vale particolarmente nel caso in cui vengano usati circuiti integrati CMOS. Quando invece il microprocessore legge sulle linee di porta, il formato di bit *presente al momento* viene portato in un registro interno della CPU: le linee di porta programmate quali input *non* godono della funzione "latch".

Scrittura sulle linee di porta PA0...PA7 o PB0...PB7 (Output Mode "0")

Quando si scrive nei registri-dati PAD o PBD, il formato di bit presente in uno dei registri della CPU viene deposto sulle linee di

porta. Anche nell'operazione di scrittura le linee di porta sono un prolungamento del bus dati. Se il processore vuole scrivere su di una o più linee di porta, queste devono essere previamente dichiarate quali output.

```
LDA # FF
STA-PADD      le linee PA0...PA7 sono programmate quali output
LDX # C3
STX-PAD       il formato di bit nel registro X viene scritto sulle linee della porta A.
```

Il contenuto del registro X qui è C3= 11000011. Questa configurazione la ritroviamo sulle linee PA7...PA0, ossia:

PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0	linee della Porta A
1	1	0	0	0	0	1	1	

Il numero esadecimale C3 viene mantenuto sulle linee di porta anche dopo l'operazione di scrittura (funzione "latch").
Osservazione: Se una linea di porta programmata da output deve pilotare un gate, un transistor od altro circuito logico, questi elementi non devono assorbire più di 1,6 mA in ingresso. Le linee di porta possono quindi tranquillamente avere per carico un normale circuito TTL. Se si trascura questa compatibilità TTL e si vogliono pilotare con le linee PB0...PB7 dei transistor, tali linee possono fornire una corrente d'uscita sino a 3 mA sotto una tensione di 1,5 V (Current-Source).

Attenzione!!! Le linee di porta del CI 6532 *non* sono protette contro sovraccarichi in tensione o in corrente. Collegando un dispositivo esterno, ad es. una stampante, occorre badare che la tensione sulle linee di porta non superi mai +7 V, o divenga addirittura negativa, altrimenti il CI 6532 può venire danneggiato.

I primi passi

A questo punto conosciamo tutte le cose più importanti riguardo le linee di porta. È tempo ormai di scrivere un piccolo programma esemplificativo, in cui trovino utilizzo i quattro registri di I/O del CI 6532. Il programma deve operare come segue:

1. Sulla porta A si deve leggere lo stato di otto interruttori, memorizzandolo nella CPU.
2. La configurazione di stato di questi interruttori deve essere convertita dallo Junior-Computer in una frequenza udibile.

3. Questa frequenza deve essere disponibile sulla linea PB0. Sarà previsto un piccolo amplificatore in grado di pilotare direttamente un altoparlante da 8 ohm.
4. Verrà steso lo schema di un circuito che realizzi le funzioni sopra citate.
5. Verrà steso un diagramma di flusso che sarà introdotto via Editor nello Junior-Computer e quindi assemblato.

Il programma dovrà in particolare soddisfare i punti 1...3. Il punto 4 si riferisce alla soluzione di tale problema con la progettazione di un adatto schema elettrico. Questo schema è illustrato in fig. 3a. Gli otto interruttori S0...S7 sono collegati alle linee di porta PA0...PA7. Dato che lo Junior-Computer deve "leggere" la posizione di questi diversi interruttori, le linee della Porta A vanno programmate quali input. Sulla linea PB0 deve trovarsi il segnale corrispondente ad una frequenza udibile. Perciò PB0 va programmato quale output. Per rendere udibile tale frequenza attraverso un altoparlante è previsto l'uso del transistor T1. Abbiamo così definita l'Hardware necessaria per la soluzione del nostro problema.

Passiamo ora allo sviluppo della Software: il programma (punto 5). In fig. 3b è illustrato un programma che converte una data configurazione di stato degli interruttori in una frequenza, ossia in un'onda quadra. Facciamo osservare che il programma descritto non è ottimale. Ma per ora non c'importa che esso sia curato, basta che serva a dimostrare la gestione dei quattro registri PAD, PADD, PBD e PBDD in modo facilmente comprensibile.

All'inizio del programma DEMO, per prima cosa, le linee di porta

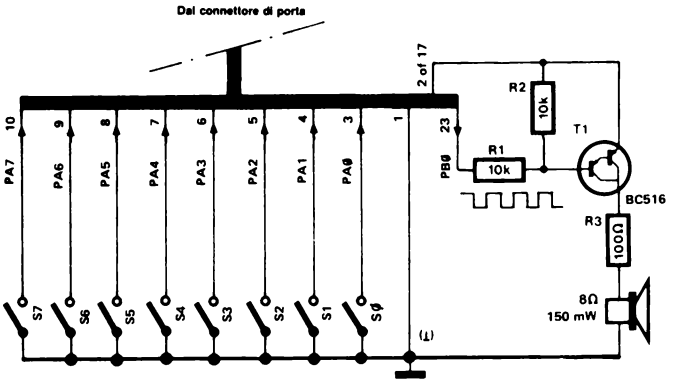


Figura 3a. La posizione degli interruttori S0 ... S7 rappresenta una configurazione di bit che viene letta entro lo Junior-Computer tramite la Porta A. Tutte le linee di porta della Porta A sono quindi definite input. Il computer converte la posizione degli interruttori in una frequenza udibile. A PB0 è collegato un amplificatore che pilota un altoparlante. PB0 pertanto è definito quale output. Su questa linea di porta compare la tensione ad onda quadra a frequenza udibile.

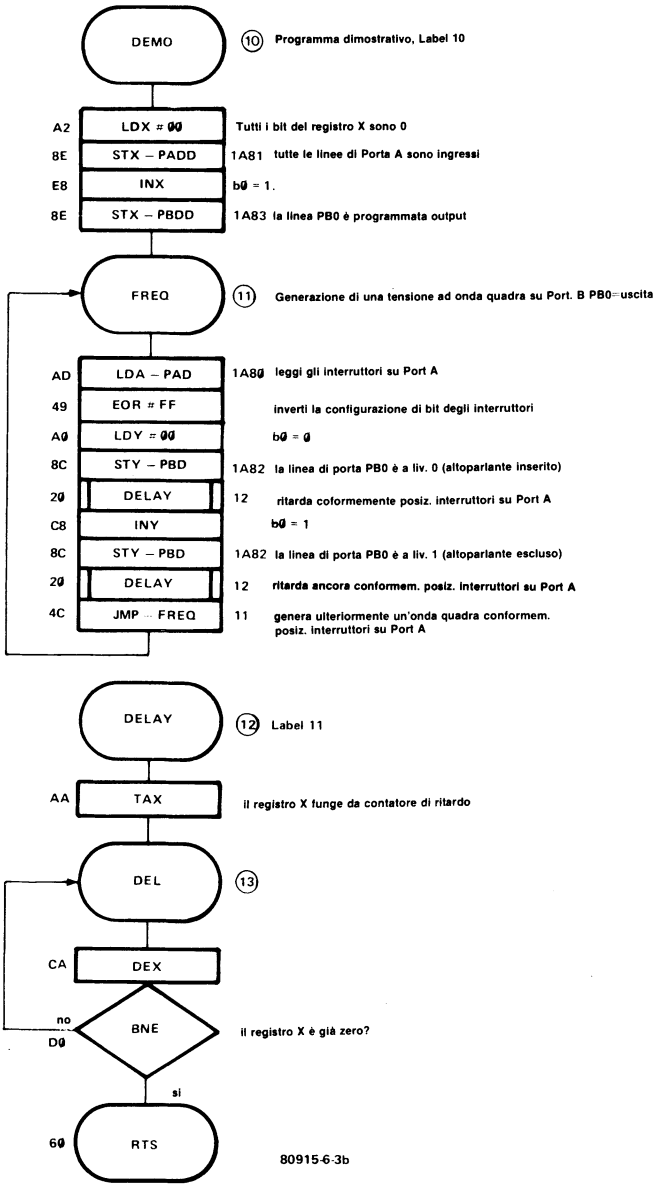


Figura 3b. Il programma DEMO legge, tramite la Porta A, la configurazione degli interruttori S0 ... S7. Un interruttore chiuso corrisponde al livello logico 1, un interruttore aperto al livello logico 0. Una volta che lo Junior-Computer ha letto la posizione degli interruttori, il programma DEMO la converte in una frequenza udibile. Il computer trasmette tale frequenza al mondo esterno tramite PB0 ed un amplificatore.

PA0...PA7 vengono definite input, e la linea PB0 output. Tutti i bit del registro di direzionamento dati PADD sono zero, e il b0 nel registro direzionamento dati PBDD vale 1. Tali valori non verranno modificati nel corso del programma.
Segue il loop relativo alla generazione della frequenza (Label FREQ). In esso il processore identifica le posizioni degli interruttori S0...S7 e le trasferisce nell'Accu. Resta da definire se far corrispondere alla posizione "interruttore chiuso" un livello logico 0 od 1. Fissiamo:

- interruttore chiuso = livello logico 1
- interruttore aperto = livello logico 0

Dopo che il relativo formato di bit è stato letto sulle linee di porta e depositato nel microprocessore (LDA-PAD), deve venire invertito nell'Accu: con EOR # FF. La definizione è così conclusa. In funzione della posizione degli interruttori deve ora venir generato un suono udibile. Il modo più semplice per realizzarlo è di portare la linea di porta PB0, dichiarata output, per un tempo definito al livello logico 0 e successivamente al livello logico 1. Su PB0 si stabilisce allora una tensione ad onda quadra, che viene resa udibile nell'altoparlante collegato.
I livelli alto e basso sulla linea PB0 sono pilotati dal registro Y: il relativo bit b0 d'ordine più basso viene alternativamente messo ad 1 od a 0, e poi trascritto nel registro dati PBD. Per quanto tempo PB0 debba restare a 0 od ad 1, è determinato dalla subroutine DELAY. In essa, la CPU copia prima il formato di bit corrispondente alle posizioni degli interruttori nel registro X (TAX). Quindi il registro X viene successivamente decrementato di 1 sino a che risulta 0: la durata di questa operazione è più o meno lunga o breve in funzione della configurazione degli interruttori. Così si ottiene su PB0 una frequenza più grave o più acuta.
Osservazione: Quando si scrive un dato formato di bit su di una o più linee di porta, esso viene conservato anche dopo l'operazione di scrittura (funzione "latch" di una linea di porta programmata quale uscita). Solo una nuova scrittura d'un nuovo formato di bit nei registri-dati PAD o PBD modifica la configurazione di bit sulle linee di porta.

Editing ed Assembling del programma DEMO

Dopo aver tracciato il diagramma di flusso per il programma DEMO, possiamo passare ad introdurlo nello Junior-Computer. Intendiamo assemblarlo in Pagina 0: ivi sono liberamente accessibili le locazioni di memoria con indirizzi da 0000 a 00E0. Perciò disponiamo il Pointer BEGAD su 0000 ed il Pointer ENDAD su 00E0. Lanciamo l'Assembler col tasto ST: si esce cioé dall'Editor mediante un NMI e si fa partire l'Assembler. Perciò bisogna prima

dell'Editing deporre il vettore NMI all'indirizzo 1F51 dell'Assembler. La fig. 3c illustra l'impostazione da tastiera del programma DEMO.

Premere i tasti:		Display:	Significato
RST			
AD	0 0 E 2	00E2 XX	
DA	0 0	00E2 00	BEGAD = 0000
+	0 0	00E3 00	
+	E 0	00E4 E0	ENDAD = 00E0
+	0 0	00E5 00	
AD	1 A 7 A	1A7A XX	
DA	5 1	1A7A51	Vettore NMI = 1F51
+	1 F	1A7B 1F	Indirizzo d'inizio dell'Assembler
AD	1 C B 5	1CB5 20	Indirizzo d'inizio dell'Editor
GO		77	
INSERT	F F 1 0 0 0	FF 10 00	Label 10: DEMO
INPUT	A 2 0 0	A2 00	LDX # 00
INPUT	8 E 8 1 1 A	8E 81 1A	STX-PADD
INPUT	E 8	E8	INX
INPUT	8 E 8 3 1 A	8E 83 1A	STX-PBDD
INPUT	F F 1 1 0 0	FF 11 00	Label 11: FREQ
INPUT	A D 8 1 A	AD 80 1A	LDA-PAD
INPUT	4 9 F F	49 FF	EOR # FF
INPUT	A 0 0 0	A0 00	LDY # 00
INPUT	8 C 8 2 1 A	8C 82 1A	STY-PBD
INPUT	2 0 1 2 0 0	20 12 00	JSR-DELAY (numero di Label 12)
INPUT	C 8	C8	INY
INPUT	8 C 8 2 1 A	8C 82 1A	STY-PBD
INPUT	2 0 1 2 0 0	20 12 00	JSR-DELAY (numero di Label 12)
INPUT	4 C 1 1 0 0	4C 11 00	JMP-FREQ (numero di Label 11)
INPUT	F F 1 2 0 0	FF 12 00	Label 12: DELAY
INPUT	A A	AA	TAX
INPUT	F F 1 3 0 0	FF 13 00	Label 13: DEL
INPUT	C A	CA	DEX
INPUT	D 0 1 3	D0 13	BNE al Label 13
INPUT	6 0	60	RTS
ST		XX XX XX	Lancio dell'Assembler via NMI
AD	0 0 0 0	0000 A2	Indirizzo iniziale di DEMO
GO			Inizio di DEMO

Figura 3c. Ecco come si può introdurre nello Junior-Computer mediante l'Editor il programma DEMO di fig. 3a, e poi assemblarlo. Il programma DEMO viene posto in pagina 0. Il lancio dell'Assembler avviene, tramite il vettore NMI, premendo il tasto ST. Se gli interruttori S0....S7 e l'amplificatore sono collegati al connettore di porta, il programma DEMO si può far partire dall'indirizzo 0000. Quando tutti gli interruttori sono chiusi, si udrà una nota grave.

Musica con lo Junior-Computer

Con il nostro Junior-Computer è possibile pure suonare delle melodie! A questo scopo dobbiamo collegare una piccola tastiera, composta di interruttori, alle linee di porta. La melodia suonata dovrà essere udibile tramite un altoparlante. Vogliamo dunque che siano soddisfatti i seguenti requisiti:

1. La tastiera deve essere munita, come una normale tastiera di pianoforte, di tasti bianchi e neri. La tastiera collegata al computer deve corrispondere a quella originale.

2. I tasti debbono risultare ordinati elettricamente in una matrice. La tastiera completa va collegata alla Porta A. Per il pilotaggio dell'altoparlante è previsto un amplificatore. Integrando fra loro questi requisiti, si ricava un circuito come quello in fig. 4a. Possiamo riconoscere in esso:
   - una tastiera tipo pianoforte,
   - una matrice di tasti collegati alla Porta A,
   - un circuito di amplificazione, collegato alla linea PB0.
3. Poiché i tasti sono ordinati in una matrice (fig. 4a), è possibile assegnare univocamente ad ogni tasto un determinato valore. Il formato della matrice è 4 x 4. Le righe sono denominate ROW0...ROW3, e le colonne COL0...COL3. Nella matrice sono pure indicati i valori dei singoli tasti. La fig. 4b mostra la realizzazione pratica della tastiera mediante Digitaster.
4. Dato che tutti i tasti sono collegati ad un'unica porta (Porta A), alcune linee di porta vanno definite quali ingressi, altre quali uscite. Vediamo come:
   - Le colonne COL0...COL3 sono collegate a PA0...PA3. Dato che il computer per calcolare il valore dei tasti legge la configurazione di stato delle colonne, le linee di porta PA0...PA3 vanno programmate quali input.
   - Le righe ROW0...ROW3 sono collegate a PA4...PA7. Queste righe devono assumere in successione lo stato logico 0, affinché il computer possa riconoscere un tasto premuto e calcolarne il valore. Dato che il computer scrive sulle righe ROW0...ROW3 della matrice un determinato formato di bit, le linee di porta PA7...PA4 vanno programmate quali output.
5. Il pilotaggio di un altoparlante tramite un amplificatore è già stato descritto. La linea di porta PB0 va programmata quale output.

Le strutture elettrica e meccanica della tastiera da piano ci sono ora chiare. Ci resta solo da sviluppare un piccolo programma per poter suonare una melodia sulla tastiera. Procediamo anche qui per passi successivi, e stabiliamo quali sono i programmi singoli di cui avremo bisogno:

- Un programma che calcola il valore di uno dei 16 tasti della tastiera.
- Un programma che verifica se un tasto risulta premuto.
- Un programma che provvede ad eliminare i rimbalzi dei tasti.
- Un programma che associa ad ogni tasto premuto una determinata frequenza.
- Una Lookup Table, in cui sono poste le 16 frequenze corrispondenti ai tasti della tastiera.

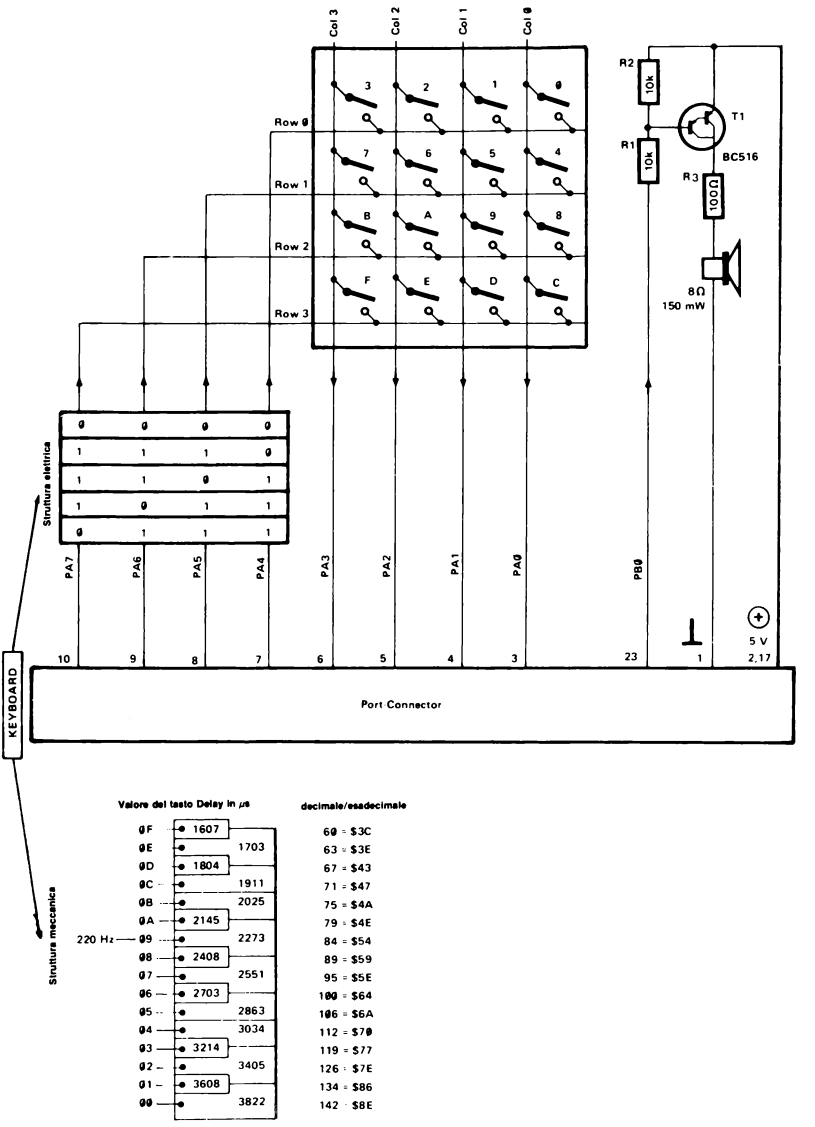


Figura 4a. Per il programma PLAY (Fig. 4g) viene collegata una vera e propria tastiera da pianoforte allo Junior-Computer. Ad ogni tasto premuto nella tastiera il computer deve generare una frequenza. La parte inferiore della figura mostra la disposizione dei tasti bianchi e neri. I numeri riportati sul tasti sono le durate dei semiperiodi delle note, in μs. Ad ogni tasto è assegnato un determinato valore di tasto (00 ... 0F). La parte destra mostra la disposizione della matrice dei tasti. Questa matrice di tasti è collegata elettricamente alla Porta A. Le linee di porta PA0 ... PA3 sono dichiarate input e le linee PA4 ... PA7 quali output. Alla linea PB0 è ancora collegato l'amplificatore con relativo altoparlante.

Un programma che calcola i valori dei 16 tasti della tastiera è mostrato in fig. 4c. Poiché questo programma provvede ad esaminare la matrice di tasti di fig. 4a, occorre che vengano definite le linee della Porta A prima che il processore salti nella subroutine KEYVAL. Come sappiamo, le linee di porta PA7...PA4 sono state dichiarate output e le PA3...PA0 input. Quindi il registro direzionamento dati della Porta A deve contenere il formato di bit 11110000 = F0. A ciò provvedono le istruzioni:

```
LDA # F0
STA-PADD
```

Così tutte le linee di porta risultano definite, e possiamo richiamare la subroutine KEYVAL. Per descriverla teniamo presenti la matrice di tasti di fig. 4a ed il diagramma di flusso di fig. 4c. All'inizio del programma la locazione di memoria ROW contiene F7 = 11110111. Il registro X funziona da contatore di riga, cioé il suo contenuto indica quale riga della matrice tasti viene letta, tramite la Porta A, e portata nel computer:

X = 03 corrispondente alla riga ROW0
X = 02 corrispondente alla riga ROW1
X = 01 corrispondente alla riga ROW2
X = 00 corrispondente alla riga ROW3

Ad ogni decremento del registro X il processore sposta il contenuto della cella di memoria ROW di un bit a sinistra e lo memorizza nel registro-dati. Ne consegue che le righe della matrice in successione assumono lo stato logico 0:

Registro X	PA7...PA4
X = 04	1111
X = 03	1110
X = 02	1101
X = 01	1011
X = 00	0111

Poiché le singole righe della matrice si azzerano in successione, è facile per il processore stabilire se in una data riga risulta premuto un tasto. L'informazione viene ricavata dalla lettura delle linee di porta PA3...PA0 (equivalente alla lettura delle colonne):

- se nessun tasto risulta premuto, allora:
  PA3...PA0 vale 1111. Il Nibble basso del byte dati contiene solo uno;
- se nessun tasto risulta premuto, allora:
  PA3...PA0 vale 1110, oppure
  1101, oppure
  1011, oppure
  0111 ossia il Nibble basso contiene uno, o
  più zeri, nel caso eventuale della pressione contemporanea di più tasti.

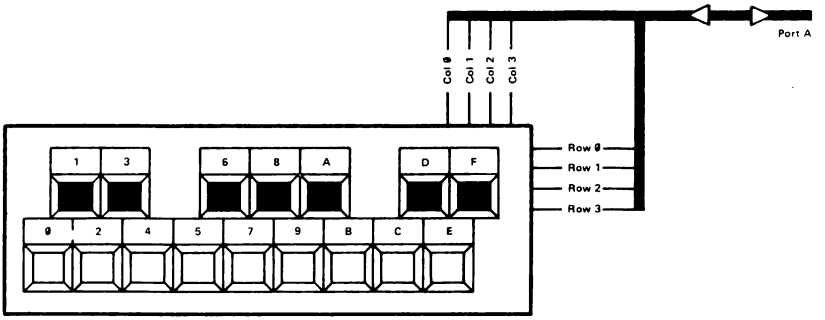


Figura 4b. Ecco il modo in cui si è costruita la tastiera da pianoforte con Digi-tasti bianchi e neri. Questi 16 tasti sono disposti elettricamente in una matrice 4 x 4; meccanicamente su due file. Sono indicati pure i valori dei singoli tasti.

Dato che nella lettura delle colonne della matrice viene letto PAD in tutta la sua ampiezza (con LDA-PAD), per cui viene posto nell'Accu anche il Nibble alto del dato, privo d'interesse, bisogna mascherarlo.

LDA-PAD	Il Nibble alto di PAD sta nell'Accu ed è uguale al Nibble alto della locazione di memoria ROW!
AND # 0F	mascheramento in 0000XXXX

Mediante il successivo confronto (CMP # 0F) ed il salto condizionato (BEQ) il processore determina se al momento nella riga risulta premuto un tasto.
Quando infine viene identificata una riga in cui risulta premuto un tasto, il computer provvede a salvare il numero di riga in TEMPX ed il formato di bit della colonna in KEY.
Ora il processore giunge al Label KEYB, ed il registro X viene impiegato come contatore di colonna (prima fungeva da contatore di riga). Il processore provvede a spostare il contenuto di ROW (= informazione di colonna) successivamente verso destra, sin quando il Flag-C diventa 0. Dopo quattro operazioni di spostamento il Flag C deve essersi azzerato, altrimenti vi è stato un errore di programma ed il computer ricomincia da capo il calcolo del valore del tasto (salto al Label KEYVAL).
Se invece il Flag C è 0, il programma verifica qual'é la riga in cui risulta premuto un tasto. Questa informazione sta nella locazione TEMPX. Con un semplice confronto (CMP) è possibile stabilire in quale riga è premuto un tasto. Prendiamo ad esempio la ROW2 (ROWC). Il valore dei tasti in una colonna aumenta di 4 da una riga a quella successiva. Quindi i tasti in ROW2 differiscono di 8 dai corrispondenti tasti di ROW0. Sommando semplicemente questo valore al numero di colonna si ottiene il valore del tasto premuto.

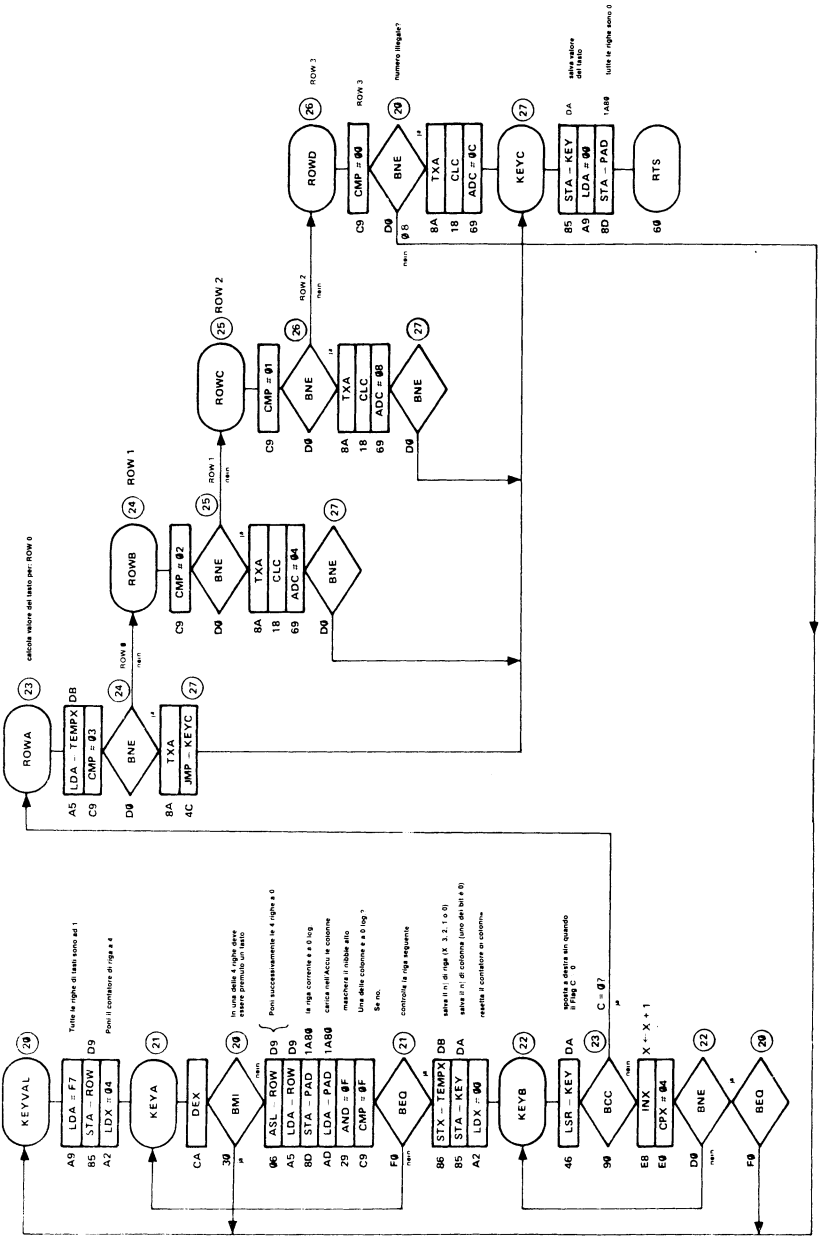


Figura 4c. La subroutine KEYVAL: serve a calcolare il valore di un tasto premuto nella matrice di tasti di fig. 4a. Il valore del tasto così determinato viene posto al rientro dalla subroutine nella cella di memoria KEY.

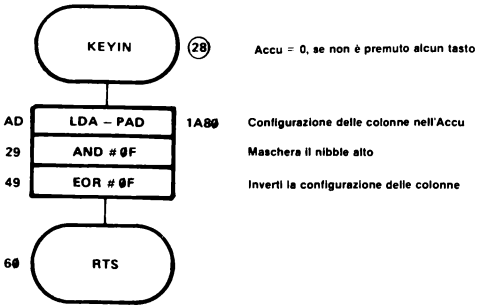


Figura 4d. La subroutine KEYIN: con essa si può controllare se è stato premto qualche tasto della matrice di tasti. Se il processore rientra da questa subroutine con contenuto dell'Accu 00, nessun tasto risulta premuto. Un valore dell'Accu diverso da zero significa che è stato premuto un tasto, ed il processore può ora mediante la subroutine KEYVAL calcolarne il valore.

Una volta calcolato il valore del tasto, il computer lo memorizza, al Label KEYC, nella cella di memoria KEY. Così questo valore resta conservato.
Successivamente, tutte le righe della matrice vengono azzerate. Ciò è necessario per il funzionamento della subroutine KEYIN (Fig. 4d). Questo sottoprogramma serve a stabilire in modo rapido se un qualsiasi tasto della tastiera è premuto: se in una qualche riga è premuto un tasto, una delle linee corrispondenti alle colonne (PA3...PA0) deve stare a livello logico 0. Dato che KEYIN

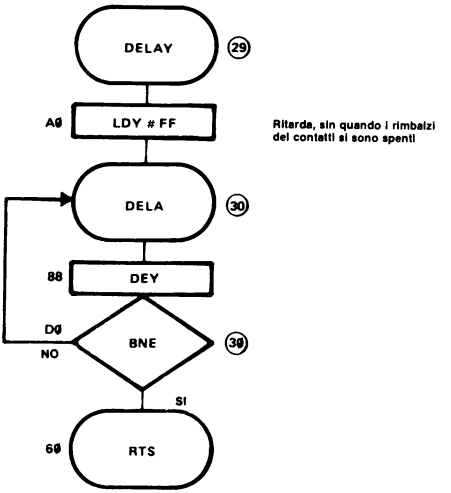


Figura 4e. La subroutine DELAY: premendo un tasto si manifestano sempre rimbalzi dei contatti. Questa subroutine obbliga il processore a permanre in un loop di attesa sin quando i rimbalzi si sono spenti.

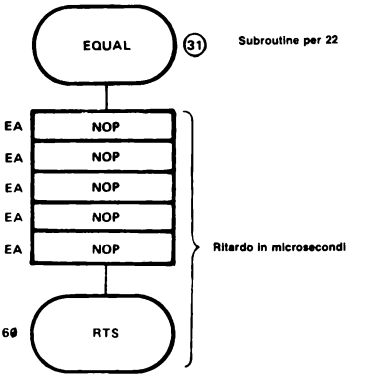


Figura 4f. La subroutine EQUAL: questa subroutine prova un ritardo di 22 μs (inclusa l'istruzione JSR), e viene impiegata per il programma PLAY.

provvede ad invertire il formato di bit letto (OER # 0F), risulterà:

Accu = 00, se non risulta premuto alcun tasto
Accu ≠ 00, se risulta premuto un tasto

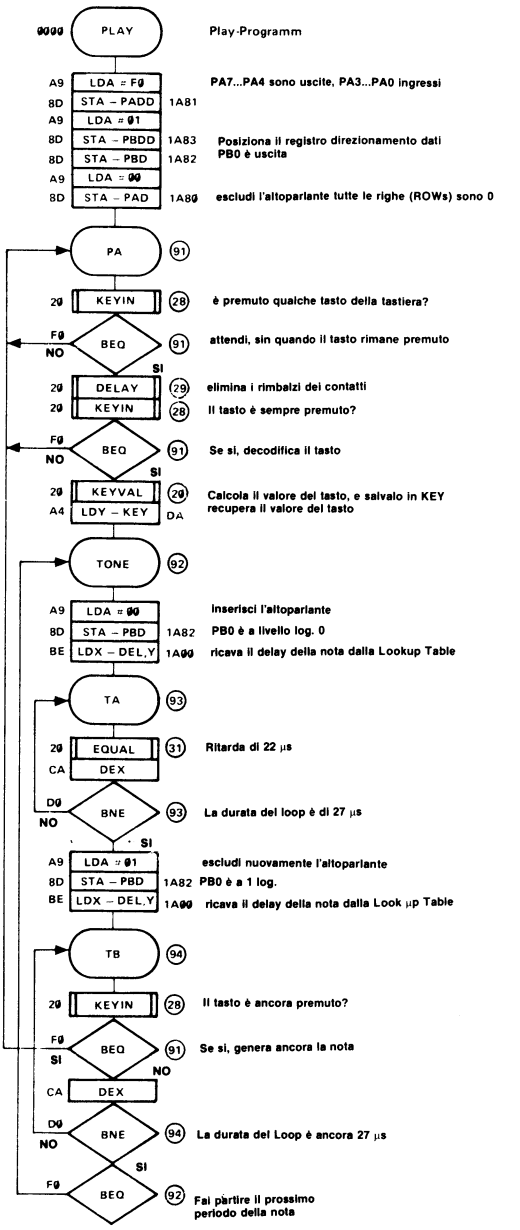
La subroutine successiva, illustrata in fig. 4e, si chiama DELAY. Essa viene impiegata per provvedere ad eliminare i rimbalzi dei tasti. Il ritardo (Delay) necessario viene generato dal ciclo di ritardo DELA.
Osservazione: nel 7° capitolo di questo libro tratteremo la tastiera ed il Display della piastra base dello Junior-Computer. La routine relativa alla tastiera ha diverse somiglianze con la subroutine KEYVAL. Dal punto di vista della programmazione delle porte, tuttavia, i due programmi hanno questa importante differenza: nel caso di KEYVAL le linee di porta risultano fra loro "intrecciate": ossia, una parte delle linee della Porta A funge da input ed un'altra da output (struttura a matrice).
Nelle routine SCANDS, SCAND e GETKEY del Monitor, descritte nel prossimo capitolo, sono le due porte, la Porta A e la Porta B, a risultare "intrecciate". Durante la scansione della tastiera la Porta A è programmata come input e la Porta B come output.
In fig. 4g è mostrato il diagramma di flusso di PLAY. Questo programma associa ad ogni tasto premuto una determinata frequenza e rende udibile la melodia suonata nell'altoparlante.
All'inzio del programma viene definito il PIA:

- PA7 ...PA4 sono uscite
- PA3 ... PA0 sono ingressi
- PB0 è uscita.

Poi giungiamo al Label PA. Lo Junior-Computer attende che un tasto venga premuto. Il riconoscimento da parte del computer se un tasto sia premuto o no avviene tramite la subroutine KEYIN (fig. 4d).

```
ROW   = $00D9
KEY   = $00DA
TEMPX = $00DB
DEL   = $1A00

PAD   = $1A80
PADD  = $1A81
PBD   = $1A82
PBDD  = $1A83
```




```
DELAYS

DEL: 1A00  8E
     1A01  86
     1A02  7E
     1A03  77
     1A04  70
     1A05  6A
     1A06  64
     1A07  5E
     1A08  59
     1A09  54
     1A0A  4E
     1A0B  4A
     1A0C  47
     1A0D  43
     1A0E  3E
     1A0F  3C
```

Figura 4g. Il diagramma di flusso del programma PLAY. Questo programma converte il valore di un tasto premuto nella tastiera in una frequenza udibile. Per la conversione valore di tasto/frequenza il processore si serve della Lookup Table DEL. In questa tabella sono dati i valori delle durate dei semiperiodi delle varie note da generare.

Se il computer identifica un tasto premuto, viene indirizzato alla subroutine DELAY (fig. 4e). Questo programma provvede ad eliminare i rimbalzi dei tasti della tastiera. Quando i rimbalzi sono stati completamente smorzati, il computer ripercorre la subroutine KEYIN e verifica se il tasto risulta sempre premuto. Solo al termine calcola, nella subroutine KEYVAL (fig. 4c), il valore del tasto premuto. Ora questo valore deve essere convertito in una frequenza. Ciò viene realizzato a partire dal Label TONE.
Prima di considerare in dettaglio la generazione delle note sonore, vediamo in generale com'è configurata la routine TONE:

- Label TONE: viene inserito l'altoparlante per un tempo fissato. La durata viene stabilita dal loop di programma TA ... BNE ... TA. La permanenza in questo loop dipende dal contenuto del registro X. Prima di entrare nel loop, il computer carica nel registro X un valore (delay) che corrisponde alla frequenza relativa al tasto premuto. Questa frequenza è depositata nella Lookup Table DEL, che considereremo più avanti.
- Label TB: l'altoparlante viene escluso per un tempo fissato. L'alternarsi di inserzione e disinserzione provoca la formazione di una tensione ad onda quadra su PB0. La durata del disinserimento viene stabilita dal loop di programma TB ... BNE ... TB. Anche i questo caso il tempo viene determinato dal contenuto del registro X. Come si vede dal diagramma di flusso, il microprocessore effettua un nuovo salto alla subroutine KEYIN, per verificare se il tasto risulta ancora premuto . Sin quando il tasto rimane premuto, su PB0 compare una tensione ad onda quadra. Solo quando il tasto torna a riposo il programma salta al Label PA ed attende la pressione di un nuovo tasto.

LookUp Table e Delays

Prima di poter calcolare i delay dalla Lookup Table, dobbiamo considerare i loop di programma TA ... BNE ... TA e TB .. BNE ... TB. Al principio di TB la CPU effettua un salto alla subroutine KEYIN; se un tasto premuto esegue la successiva istruzione BEQ (senza saltare), decrementa il registro X e salta per un tempo determinato al Label TB. Tutte le istruzioni, per venire eseguite, richiedono un dato tempo. Diamo alcuni valori:

JSR-KEYIN	=	6 μs
LDA-PAD	=	4 μs
AND # ØF	=	2 μs
EOR # ØF	=	2 μs
RTS	=	6 μs
BEQ	=	2 μs nessun Branch sinché il tasto resta premuto
DEX	=	2 μs
BNE-TB	=	3 μs viene eseguito il Branch (salto condizionato)
in totale	=	27 μs

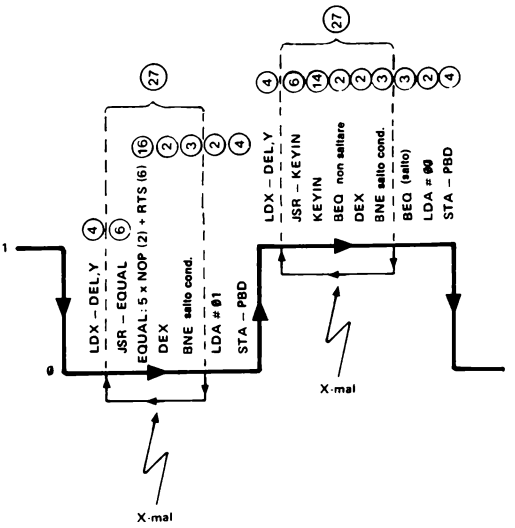


Figura 5. Con questo programma lo Junior-Computer genera sulla linea di porta PB0 una tensione ad onda quadra, che viene poi resa udibile tramite l'altoparlante.

Il tempo di permanenza nel loop è quindi di 27 μs moltiplicato per il contenuto del registro X. La fig. 5 illustra questo andamento in forma grafica. Anche il loop TA ... BNE ... TA deve avere corrispondentemente una durata di 27 μs. Dato che in questo loop non si effettua il salto alla subroutine KEYIN, occorre apportare una correzione alla durata nel loop: allo scopo è prevista la subroutine EQUAL, che compensa per i 22 μs mancanti. Ora possiamo calcolare la Lookup Table. In fig. 4a i tasti sono corredati di alcuni valori numerici che dobbiamo chiarire. Prendiamo ad esempio la nota musicale LA, la cui frequenza è 440 Hz. Tra periodo e frequenza corre la relazione:

$$T = 1/f$$

Una frequenza di 440 Hz ha dunque un periodo di $1/440 = 2273$ μs. In una tastiera da pianoforte la frequenza da un tasto a quello successivo sale o scende nel rapporto $\sqrt[12]{2} = 1{,}059463$. Possiamo quindi associare ad ogni tasto una frequenza ovvero una durata di periodo, come mostrato in fig. 4a. Conosciamo inoltre la durata dei due loop TA ... BNE ... TA e TB ... BNE .. TB: 27 μs. Per ogni tasto è ora facile calcolare gli esatti delay.
Torniamo alla nostra nota LA: la durata del suo periodo è di 2273 μs. Il delay nella Lookup Table per questa nota deve quindi essere: Delay = Periodo/27 = 2273 : 27 = 84_{10} = $54. Si possono allo stesso modo calcolare i delay per ogni nota, ricavando la Lookup Table LEN. Dato che il microprocessore per ogni nota suonata estrae due volte il delay della Lookup Table, le note effettivamente emesse sono di un'ottava inferiori a quelle calcolate!

Ora non ci resta che introdurre il programma PLAY (fig. 4g) insieme ai suoi sottoprogrammi KEYVAL, KEYIN, DELAY ed EQUAL (fig. 4c, d, e, f) nello Junior-Computer. Naturalmente impiegheremo a tale scopo ancora una volta l'Editor e l'Assembler. La lista dettagliata dell'impostazione da tastiera delle istruzioni del programma è riportata in fig. 6. Dopo l'assemblaggio del programma, e dopo aver introdotto la Lookup Table in Pagina 1A, dello Junior-Computer possiamo collegare la tastiera e l'amplificatore per la riproduzione sonora con l'altoparlante al PIA.
Ed ora: buon divertimento suonando il vostro pianino!

Figura 6. L'impostazione da tastiera del programma PLAY. Dopo aver impostati tutti i Label e le istruzioni, vien fatto partire l'Assembler col tasto ST. Ad assemblaggio effettuato, si può suonare una melodia sulla tastiera da piano. L'indirizzo di partenza di PLAY è 0000.

Tasti:		**Display:**	**Significato:**
RST		XXXX XX	
AD	0 0 E 2	00E2 XX	
DA	0 0	00E2 00	BEGAD = 0000
+	0 0	00E3 00	
+	E 0	00E4 E0	ENDAD = 00E0
+	0 0	00E5 00	
AD	1 A 7 A	1A7A XX	Vettore NMI = 1F51
DA	5 1	1A7A 51	(Indirizzo di partenza dell'Assembler)
+	1 F	1A7B 1F	
AD	1 C B 5	1CB5 20	
GO		77	Indirizzo iniziale dell'Editor
INSERT	F F 9 0 0 0	FF 90 00	Label 90: PLAY
INPUT	A 9 F 0	A9 F0	LDA # F0
INPUT	8 D 8 1 1 A	8D 81 1A	STA-PADD
INPUT	A 9 0 1	A9 01	LDA # 01
INPUT	8 D 8 3 1 A	8D 83 1A	STA-PBDD
INPUT	8 D 8 2 1 A	8D 82 1A	STA-PBD
INPUT	A 9 0 0	A9 00	LDA # 00
INPUT	8 D 8 0 1 A	8D 80 1A	STA-PAD
INPUT	F F 9 1 0 0	FF 91 00	Label 91: PA
INPUT	2 0 2 8 0 0	20 28 00	JSR-KEYIN (Label 28)
INPUT	F 0 9 1	F0 91	BEQ a PA (Label 91)
INPUT	2 0 2 9 0 0	20 29 00	JSR-DELAY (Label 29)
INPUT	2 0 2 8 0 0	20 28 00	JSR-KEYIN (Label 28)
INPUT	F 0 9 1	F0 91	BEQ a PA (Label 91)
INPUT	2 0 2 0 0 0	20 20 00	JSR-KEYVAL (Label 20)
INPUT	A 4 D A	A4 DA	LDYZ-KEY (00DA)
INPUT	F F 9 2 0 0	FF 92 00	Label 92: TONE
INPUT	A 9 0 0	A9 00	LDA # 00
INPUT	8 D 8 2 1 A	8D 82 1A	STA-PBD
INPUT	B E 0 0 1 A	BE 00 1A	LDX-DEL, Y (DEL = 1A00)
INPUT	F F 9 3 0 0	FF 93 00	Label 93: TA
INPUT	2 0 3 1 0 0	20 31 00	JSR-EQUAL (Label 31)
INPUT	C A	CA	DEX

Tasti:		Display:	Significato:
INPUT	D 0 9 3	D0 93	BNE a TA (Label 93)
INPUT	A 9 0 1	A9 01	LDA # 01
INPUT	8 D 8 2 1 A	8D 82 1A	STA-PBD
INPUT	B E 0 0 1 A	BE 00 1A	LDX-DEL, Y
INPUT	F F 9 4 0 0	FF 94 00	Label 94: TB
INPUT	2 0 2 8 0 0	20 28 00	JSR-KEYIN (Label 28)
INPUT	F 0 9 1	F0 91	BEQ a PA (Label 91)
INPUT	C A	CA	DEX
INPUT	D 0 9 4	D0 94	BNE a TB (Label 94)
INPUT	F 0 9 2	F0 92	BEQ a TONE (Label 92)
INPUT	F F 2 0 0 0	FF 20 00	Label 20: KEYVAL
INPUT	A 9 F 7	A9 F7	LDA # F7
INPUT	8 5 D 9	85 D9	STAZ-ROW (00D9)
INPUT	A 2 0 4	A2 04	LDX # 04
INPUT	F F 2 1 0 0	FF 21 00	Label 21: KEYA
INPUT	C A	CA	DEX
INPUT	3 0 2 0	30 20	BMI a KEYVAL (Label 20)
INPUT	0 6 D 9	06 D9	ASLZ-ROW (00D9)
INPUT	A 5 D 9	A5 D9	LDAZ-ROW (00D9)
INPUT	8 D 8 0 1 A	8D 80 1A	STA-PAD
INPUT	A D 8 0 1 A	AD 80 1A	LDA-PAD
INPUT	2 9 0 F	29 0F	AND # 0F
INPUT	C 9 0 F	C9 0F	CMP # 0F
INPUT	F 0 2 1	F0 21	BEQ a KEYA (Label 21)
INPUT	8 6 D B	86 DB	STXZ-TEMPX (00DB)
INPUT	8 5 D A	85 DA	STAZ-KEY (00DA)
INPUT	A 2 0 0	A2 00	LDX # 00
INPUT	F F 2 2 0 0	FF 22 00	Label 22: KEYB
INPUT	4 6 D A	46 DA	LSRZ-KEY (00DA)
INPUT	9 0 2 3	90 23	BCC a ROWA (Label 23)
INPUT	E 8	E8	INX
INPUT	E 0 0 4	E0 04	CPX # 04
INPUT	D 0 2 2	D0 22	BNE a KEYB (Label 22)
INPUT	F 0 2 0	F0 20	BEQ a KEYVAL (Label 20)
INPUT	F F 2 3 0 0	FF 23 00	Label 23: ROWA
INPUT	A 5 D B	A5 DB	LDAZ-TEMPX (00DB)
INPUT	C 9 0 3	C9 03	CMP # 03
INPUT	D 0 2 4	D0 24	BNE a ROWB (Label 24)
INPUT	8 A	8A	TXA
INPUT	4 C 2 7 0 0	4C 27 00	JMP-KEYC (label 27)
INPUT	F F 2 4 0 0	FF 24 00	Label 24: ROWB
INPUT	C 9 0 2	C9 02	CMP x 02
INPUT	D 0 2 5	D0 25	BNE a ROWE (Label 25)
INPUT	8 A	8A	TXA
INPUT	1 8	18	CLC
INPUT	6 9 0 4	69 04	ADC # 04
INPUT	D 0 2 7	D0 27	BNE a KEYC (Label 27)
INPUT	F F 2 5 0 0	FF 25 00	Label 25: ROWC
INPUT	C 9 0 1	C9 01	CMP # 01
INPUT	D 0 2 6	D0 26	BNE a ROWD (Label 26)
INPUT	8 A	8A	TXA
INPUT	1 8	18	CLC
INPUT	8 9 0 8	69 08	ADC x 08
INPUT	D 0 2 7	D0 27	BNE a KEYC (Label 27)
INPUT	F F 2 6 0 0	FF 26 00	Label 26: ROWD
INPUT	C 9 0 0	C9 00	CMP # 00
INPUT	D 0 2 0	D0 29	BNE a KEYVAL (Label 20)
INPUT	8 A	8A	TXA
INPUT	1 8	18	CLC
INPUT	6 9 0 C	69 0C	ADC # 0C
INPUT	F F 2 7 0 0	FF 27 00	Label 27: KEYC
INPUT	8 5 D A	85 DA	STAZ-KEY (00DA)
INPUT	A 9 0 0	A9 00	LDA # 00
INPUT	8 D 8 0 1 A	8D 80 1A	STA-PAD
INPUT	6 0	60	RTS
INPUT	F F 2 8 0 0	FF 28 00	Label 28: KEYIN
INPUT	A D 8 0 1 A	AD 80 1A	LDA-PAD

Tasti:		Display:	Significato
INPUT	2 9 0 F	29 0F	AND # 0F
INPUT	4 9 0 F	49 0F	EOR # 0F
INPUT	6 0	60	RTS
INPUT	F F 2 9 0 0	FF 29 00	Label 29: DELAY
INPUT	A 0 F F	A0 FF	LDY # FF
INPUT	F F 3 0 0 0	FF 30 00	Label 30: DELA
INPUT	8 8	88	DEY
INPUT	D 0 3 0	D0 30	BNE a DELA (Label 30)
INPUT	6 0	60	RTS

Tasti:		Display:	Significato
INPUT	F F 3 1 0 0	FF 31 00	Label 31: EQUAL
INPUT	E A	EA	NOP
INPUT	E A	EA	NOP
INPUT	E A	EA	NOP
INPUT	E A	EA	NOP
INPUT	E A	EA	NOP
INPUT	6 0	60	RTS

Tasti		Display:
SEARCH	F F 9 0	FF 90 00
SKIP		A9 F0
SKIP		8D 81 1A
SKIP		A9 01
SKIP	8D 83 1A	
SKIP		8D 82 1A
SKIP		A9 00
SKIP		8D 80 1A
SKIP		FF 91 00
SKIP		20 28 00
SKIP		F0 91
SKIP		20 29 00
usw		

Tasti:	Display:
ST	XXXXXX

Tasti:		Display:	Significato
AD	1 A 0 0	1A00 XX	LOOKUP TABLE LEN
DA	8 E	1A00 8E	LEN * $1A00
+	8 6	1A01 86	
+	7 E	1A02 7E	
+	7 7	1A03 77	
+	7 0	1A04 70	
+	6 A	1A05 6A	
+	6 4	1A06 64	
+	5 E	1A07 5E	
+	5 9	1A08 59	
+	5 4	1A09 54	
+	4 E	1A0A 4E	
+	4 A	1A0B 4A	
+	4 7	1A0C 47	
+	4 3	1A0D 43	
+	3 E	1A0E 3E	
+	3 C	1A0F 3C	
AD	0 2 0 0	0200	
GO			Inizio del programma.

Lancio del programma:

Tasti		Display
AD	0 0 0 0	0000A9
GO		

Adesso è possibile suonare una melodia sulla tastiera!

L'Interval-Timer

Lo Junior-Computer è anche provvisto di un Timer degli intervalli, anch'esso situato nel CI 6532. Questo timer è programmabile e marcia indipendentemente dal microprocessore: ossia, mentre questo timer è in funzione, la CPU è liberamente accessibile all'utente. Questo è un requisito necessario quando si vuole lavorare in modo efficiente, cioé rapido, con un microprocessore.
Adesso possiamo svelare un segreto: come mai lo Junior-Computer richiede così pochi circuiti integrati e costa quindi così poco. L'Interval-Timer ed il Peripheral Interface Adapter sono riuniti sul medesimo chip. Sono due componenti molto importanti dello Junior-Computer, e ne accrescono le capacità di impiego di un ordine di grandezza. Tuttavia, prima di poter lavorare con il timer degli intervalli occorre chiarirne bene il funzionamento. Per prima cosa, quindi, descriveremo il modello di programmazione del timer e considereremo alcuni programmi-modello in cui trova impiego l'Interval-Timer.

Lo schema a blocchi dell'Interval-Timer

La fig. 7 presenta lo schema a blocchi dell'Interval-Timer. Come si vede, il Timer di 3 blocchi:

- *Cella del timer:* la cella del timer ha la capacità di 8 bit è risulta collegata al bus dati. La cella del timer si può paragonare ad una normale cella di RAM: in essa si possono scrivere o leggere dati.
- *Registro-divisore:* il registro divisore è collegato alle linee indirizzo A0 ed A1 nonché col segnale di clock Φ2.

Insieme, la cella del timer ed il registro-divisore compongono l'Interval Timer.
Il timer degli intervalli si può paragonare ad un contatore all'indietro TTL o CMOS. Questi circuiti ben noti hanno la proprietà che il registro-contatore può essere fissato ad un dato valore binario. Ogni impulso di clock decrementa ogni volta di 1 il contatore, sin quando il contatore stesso non è azzerato. L'Interval-Timer dello Junior-Computer lavora esattamente allo stesso modo. Il segnale di clock Φ2 costituisce l'impulso di clock che fa contare all'indietro l'Interval-Timer. Tramite il registro divisore, il segnale Φ2 può

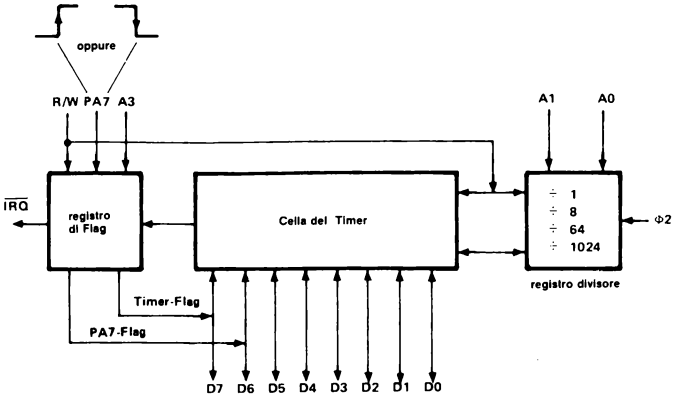


Figura 7. Lo schema a blocchi dell'Interval-Timer con il Flag-Register. Il registro divisore e la cella del timer, insieme, compongono il timer degli intervalli. Con il registro divisore è possibile fissare un dato rapporto di divisione. Sono disponibili quattro rapporti di divisione: :1, :8, :64 e :1024. La selezione del fattore di divisione è determinata dalla configurazione dei bit sulle due linee indirizzi A0 ed A1. La frequenza di clock Φ 2 pilota, tramite il registro divisore, la cella del timer. La frequenza di clock Φ 2 viene divisa per uno dei fattori citati. Il timer degli intervalli si può paragonare ad un contatore all'indietro programmabile. A seconda del rapporto di divisione selezionato, il contenuto della cella del timer viene decrementato di 1 ogni 1, ogni 8, ogni 64 oppure ogni 1024 µs. La partenza dell'Interval-Timer è provocata da un'operazione di scrittura nella cella del timer. Il timer conta allora all'indietro, sin quando il contenuto della cella del timer è divenuto inferiore a 0 (valore = FF). Quando ciò avviene, nel Flag-Register vien posto automaticamente il Timer-Flag ad 1. Il programmatore può abilitare oppure no il Timer a generare un IRQ. Se il timer viene abilitato ad emettere un Timer-IRQ, quando il Timer-Flag viene posto ad 1 la linea di IRQ diviene contemporaneamente = 0 log.
Nel Flag-Register è presente anche un secondo Flag: il PA7-Flag. Esso opera in connessione col rivelatore dei fronti. La linea di porta PA7 è l'ingresso del rivelatore dei fronti. Il rivelatore dei fronti è programmabile in modo che il PA7-Flag venga posto ad 1 in corrispondenza di un fronte positivo o negativo. Il programmatore può, abilitare oppure no il rivelatore dei fronti a generare un IRQ. Se il rivelatore dei fronti è abilitato ad emettere un IRQ, quando il PA7-Flag vien posto ad 1 la linea di IRQ diviene contemporaneamente 0 log.

venire diviso per determinati fattori: il fattore di divisione è fissato dalla configurazione dei bit sulle linee indirizzo A0 ed A1. Con questi due bit indirizzo si possono stabilire 4 valori del fattore di divisione:

A1	A0	Fattore di divisione	
0	0	÷ 1 T	
0	1	÷ 8 T	
1	0	÷ 64 T	T = 1 µs (per lo Junior-Computer)
1	1	÷ 1024 T	

La frequenza di clock Φ2 può quindi venire divisa per 1, 8, 64 o

1024 prima di decrementare il contenuto della cella del timer di un'unità. Chiariamo con un esempio. Scriviamo con la CPU il numero $1E nella cella del timer e scegliamo con le due linee indirizzo A1 ed A0 il fattore di divisione ÷ 64. Abbiamo ora:

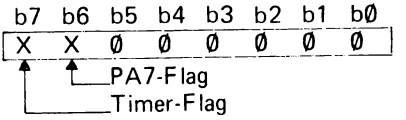
$1E = 30_{10} e 30_{10} per $64_{10} = 1920_{10}$

o in altri termini: con un fattore di divisione di 64 ed un offset del timer di $1E il contenuto della cella del timer diventa 0 dopo 1920 µsecondi. Prima di poter descrivere meglio il timer degli intervalli, dobbiamo ancora considerare un altro importante registro del CI 6532: il registro Flag.

- *Registro Flag*: il registro-Flag del 6532 ha due Flag:
  1. il Timer-Flag,
  2. il PA7-Flag (sarà descritto in altra sezione!)

Anche il registro Flag ha la capacità di 8 bit, ma solo due di tali bit sono impiegati come Flag: b7 e b6. Gli altri bit, ossia b5 ... b0, sono sempre nulli quando il processore legge in questo registro. Il Timer-Flag è attivato dall'Interval-Timer, il PA7-Flag da un fronte d'impulso positivo o negativo sulla linea di porta PA7.

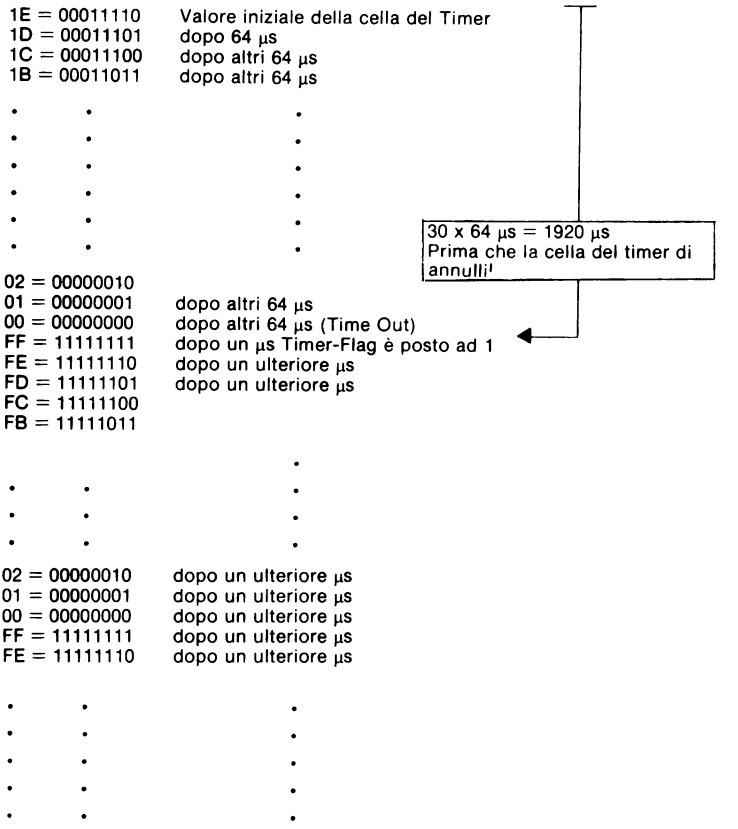
Struttura del registro-Flag:



bits
Registro-Flag

Quando viene attivato il Timer-Flag dell'Interval-Timer?

Il bit b7 del registro-Flag è il Timer-Flag. Questo Flag viene settato (= posto ad 1; b7=1) quando il timer ha terminato il suo ciclo. Con riferimento all'esempio citato in precedenza, vediamo a pagina seguente quando ciò accade:



Adesso è chiaro il modo in cui il registro Flag opera in connessione coll'Interval-Timer. Dopo che la CPU ha memorizzato un numero - l'offset del timer - nella cella del timer, ed allo stesso tempo con le due linee indirizzo A0 ed A1 è stato fissato il fattore di divisione, il timer degli intervalli inizia il suo conto all'indietro. Avendo scelto un fattore di divisione di 64 ed un offset del timer di $1E = 30_{10}$, la cella del timer si annulla dopo 1920 µs. L'istante in cui si azzera la cella di memoria è detto *Time Out*.

Altre nozioni importanti da sapere:

1. Il Timer-Flag nel registro Flag viene rimesso a 0 dopo ogni operazione di lettura nella cella del timer (lettura nelle celle RDTEN o RDTDIS).
2. Analogamente, il Timer-Flag nel registro Flag viene rimesso a zero dopo ogni operazione di scrittura in una delle celle del timer (scrittura in una fra CNTA e CNTH). Il Timer-Flag è quindi sempre 0 dopo ogni avvio del timer (= scrittura del valore dell'offset del timer nella cella del timer).

3. Il Timer-Flag nel registro Flag passa ad 1 1 µs **dopo il Time Out**. Nel caso del nostro esempio abbiamo:
   Timer-Offset = 1E = 30_{10}
   Fattore di divisione = 64
   Time Out dopo 30 x 64 = 1920 µs
   Il Timer-Flag viene posto ad 1 dopo 1920 + 1 = 1921 µs
4. Trascorso il Time Out, il fattore di divisione si fissa automaticamente ad 1, indipendentemente dal valore che esso aveva in precedenza. Se ad esempio il fattore di divisione preliminare era di 64, si ha il seguente andamento:
   - prima del Time Out il contenuto della cella del timer viene decrementato di una unità ogni 64 µs,
   - dopo il Time Out il contenuto della cella del timer viene decrementato di uno ogni secondo (1 µs corrisponde alla frequenza di 1 MHz del clock dello Junior-Computer).

Il registro Flag gestisce pure la linea IRQ collegata all'input IRQ della CPU. Se il Flag I nel registro di stato della CPU è nello stato 0, il CI 6532 è abilitato ad emettere un IRQ. La linea IRQ passa allora allo stato logico 0.
La gestione della linea IRQ è affidata ad entrambi i Flag del CI 6532. Se *l'uno o l'altro* fra il Timer-Flag del timer degli intervalli e il PA7-Flag sono ad 1, la linea IRQ *può* assumere lo stato logico 0 e generare un Interrupt Request. Il programmatore può tuttavia disabilitare il comando IRQ con il registro Flag del 6532. A tale scopo la linea indirizzo A3 è collegata al registro Flag.
- se A3 = 0 : Il CI 6532 *non può* generare un IRQ. La linea IRQ *non può* assumere lo stato logico 0.
- se A3 = 1 : Il CI 6532 *può* generare un IRQ. La linea IRQ *può* assumere lo stato logico 0.

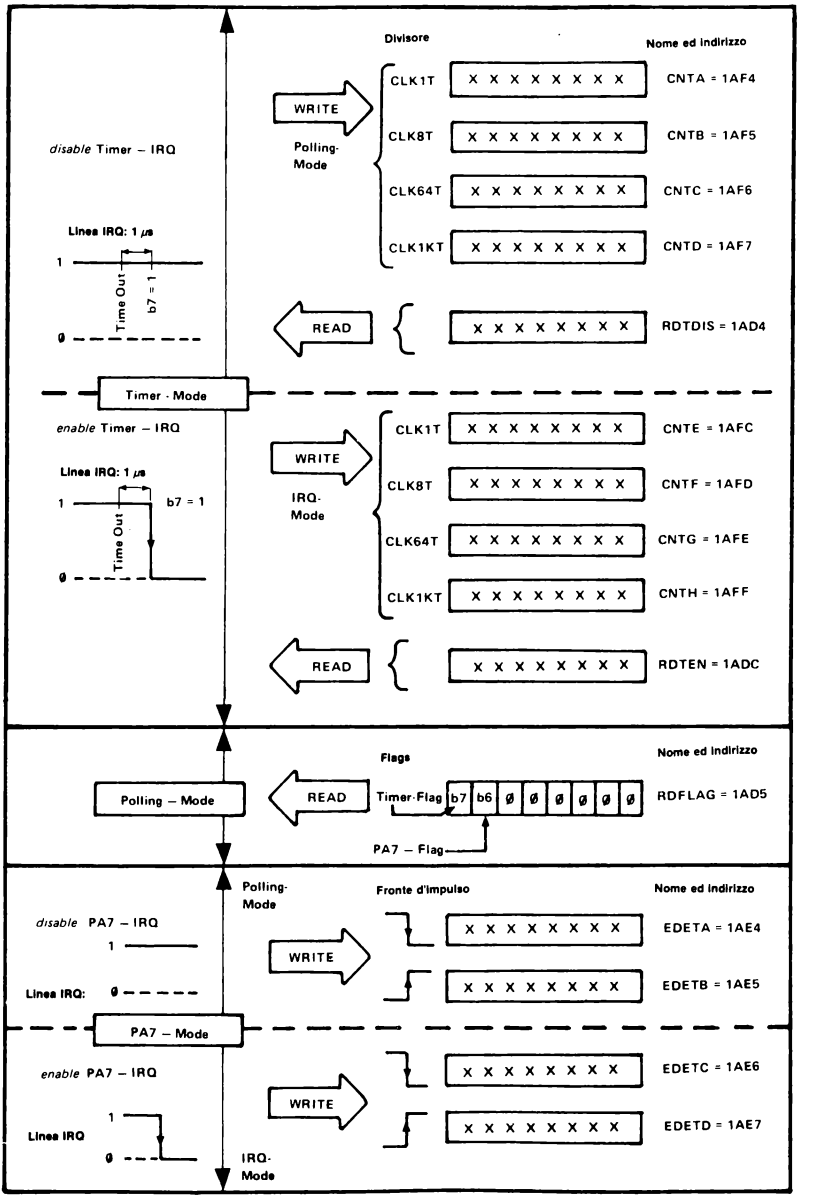
Riepilogo

Abbiamo illustrato lo schema a blocchi dell'Interval-Timer. Il timer degli intervalli consta di un registro divisore e di una cella del timer. La cella del timer è collegata al registro Flag. Nel registro Flag sono posti due Flag:
1. Il Timer-Flag (b7),
2. Il PA7-Flag (b6).

Il registro Flag gestisce la linea IRQ in funzione dello stato dei due Flag citati (Timer-Flag, PA7-Flag). Normalmente questa linea è allo stato logico 1. Quando tale linea è posta a 0, il CI 6532 è in grado di generare un Interrupt Request.
Il programmatore può impedire al 6532 l'emissione di un IRQ. Dopo un'operazione di lettura o di scrittura nella cella del timer il Timer-Flag nel registro Flag è **sempre** posto a 0. Se il programmatore lo richiede, il CI 6532 può generare un Timer-IRQ. Quando il timer ha finito il suo ciclo, il Timer-Flag viene posto ad 1. Ciò porta di conseguenza a 0 la linea IRQ, e la CPU effettua un salto ad una routine d'Interrupt.

Le dieci regole d'oro del Timer, ovvero il modello di programmazione dell'Interval Timer

Dopo aver ampiamente descritto lo schema a blocchi del timer degli intervalli, siamo in grado di comprendere facilmente il modello di programmazione. La fig. 8 mostra come si presentano



all'utente l'Interval-Timer, le celle del timer ed il registro Flag.

1. Il timer degli intervalli consta in tutto di 8 celle. I nomi delle celle del timer sono:
   CNTA, CNTB, CNTC, CNTD, CNTE, CNTF, CNTG e CNTH.
   Ad esse (vedi fig. 8) son associati gli indirizzi 1AF4 ... 1AF7 e 1AFC ... 1AFF.
2. Il processore può effettuare sia operazioni di lettura che di scrittura in una delle celle del timer.
3. Ad ogni cella del timer è assegnato un dato fattore di divisione. I fattori di divisione selezionabili sono quattro:
   CLK1T = 1
   CLK8T = 8
   CLK64T = 64
   CLK1KT = 1024
   Alla cella CNTA è assegnato un fattore di divisione di 1. Alla cella successiva CNTB un fattore di divisione di 8, e così via. Prima che il segnale di clock 2 provveda quindi a decrementare il contenuto della cella del timer di 1, esso viene diviso per un ben determinato fattore.
4. Il timer degli intervalli è suddiviso in due parti di 4 celle ciascuna. La prima parte è costituita dalle celle CNTA ... CNTD, l'altra dalle CNTE ... CNTH.

◄ **Figura 8. Il modello di programmazione del timer degli intervalli e del rivelatore dei fronti. L'Interval-Timer comprende le 8 celle CNTA...CNTH. Per effetto della scrittura del Timer-Offset (00...FF) in una delle 8 celle del timer si ha l'avvio del timer. Una volta partito, il timer conta all'indietro in funzione del rapporto di divisione, sino al Time-Out. Dopo il Time Out il Timer-Flag nel registro flag RDFLAG viene posto ad 1, ed il timer genera, se lo desidera l'utente, un IRQ (Timer IRQ) sulla linea IRQ. Se l'avvio del timer avviene mediante scrittura in una delle celle CNTA...CNTD del timer esso non può però generare un IRQ. La linea di IRQ rimane al livello logico 1, se il Timer Flag è alto (=1). Se la scrittura viene effettuata in una delle celle CNTE...CNTH del timer, questo parte analogamente e dopo il Timer Out genera un IRQ. La linea di IRQ passa a 0 quando il Timer Flag viene posto ad 1. Un'operazione di scrittura in una delle celle CNTA...CNTH determina se il timer degli intervalli opera in Polling- od IRQ-Mode. Il fattore di divisione è fissato dalla scrittura di un dato che fa partire il timer. Una lettura nelle celle RDTEN o RDTIS del timer prima del Time Out non modifica il fattore di divisione, ma provoca il cambiamento da Polling- ad IRQ-Mode o viceversa. La lettura nelle celle RDTEN o RDTIS del timer comunica informazioni sul contenuto corrente della cella del timer. Se l'Interval Timer opera in IRQ Mode, tramite la linea IRQ esso segnala al microprocessore che ha avuto lungo un Time Out. Se invece il timer opera in Polling Mode, è la lettura del registro Flag che segnala alla CPU se ha già avuto luogo un Time Out. Anche il rivelatore dei fronti, così come l'Interval Timer, può operare in Polling- od in IRQ-Mode. Un'operazione di scrittura in uno dei quattro registri di governo determina se il rivelatore dei fronti deve reagire ad un fronte positivo o negativo degli impulsi applicati alla linea PA7. Corrispondentemente sarà un fronte positivo o negativo a portare alto (1 log.) il PA7-Flag. Se il rivelatore dei fronti è abilitato a generare un IRQ (per effetto d'una scrittura nel registro di governo EDETC per ↓ o in EDETD per ↑), la linea IRQ passa a 0. quando il PA7-Flag passa ad 1. Se il rivelatore dei fronti opera in IRQ-Mode, viene utilizzata la linea IRQ per segnalare al microprocessore che su PA7 è stato applicato il dato fronte d'impulso.**

5. Quando il processore esegue un'operazione di scrittura in una delle celle CNTA ... CNTD, si ha la partenza del timer degli intervalli. Contemporaneamente ed in modo automatico il Timer-Flag (b7 nel registro Flag) viene messo a 0, e la linea IRQ passa ad 1.
   La scrittura nelle celle del timer CNTA ... CNTD fa sì che:
   - Il 6532 *non* è abilitato a generare Request sulla linea IRQ.
   - Il Timer-Flag (b7) del registro Flag viene *sempre* messo a 0.

   A seconda del fattore di divisione, può risultare più o meno lungo il tempo in cui il timer completa il suo ciclo. Dopo il Time Out (il contenuto della cella del timer è FF) si ha che:
   - La linea IRQ *non può* assumere lo stato logico 0 e quindi *non può* venir generato in Interrupt.
   - Il Timer Flag (b7 nel registro Flag) è posto ad 1.

6. Quando il processore esegue un'operazione di scrittura in una delle celle CNTE ... CNTH, si ha la partenza del timer. Contemporaneamente ed in modo automatico il Timer-Flag (b7 nel registro Flag) viene rimesso a 0, e la linea IRQ passa ad 1 (confronta punto 5).
   La scrittura nelle celle CNTE ... CNTH fa sì che:
   - Il 6532 è abilitato a generare un Interrupt Request sulla linea IRQ.
   - Il Timer-Flag (b7 nel registro Flag) viene **sempre** messo a 0.

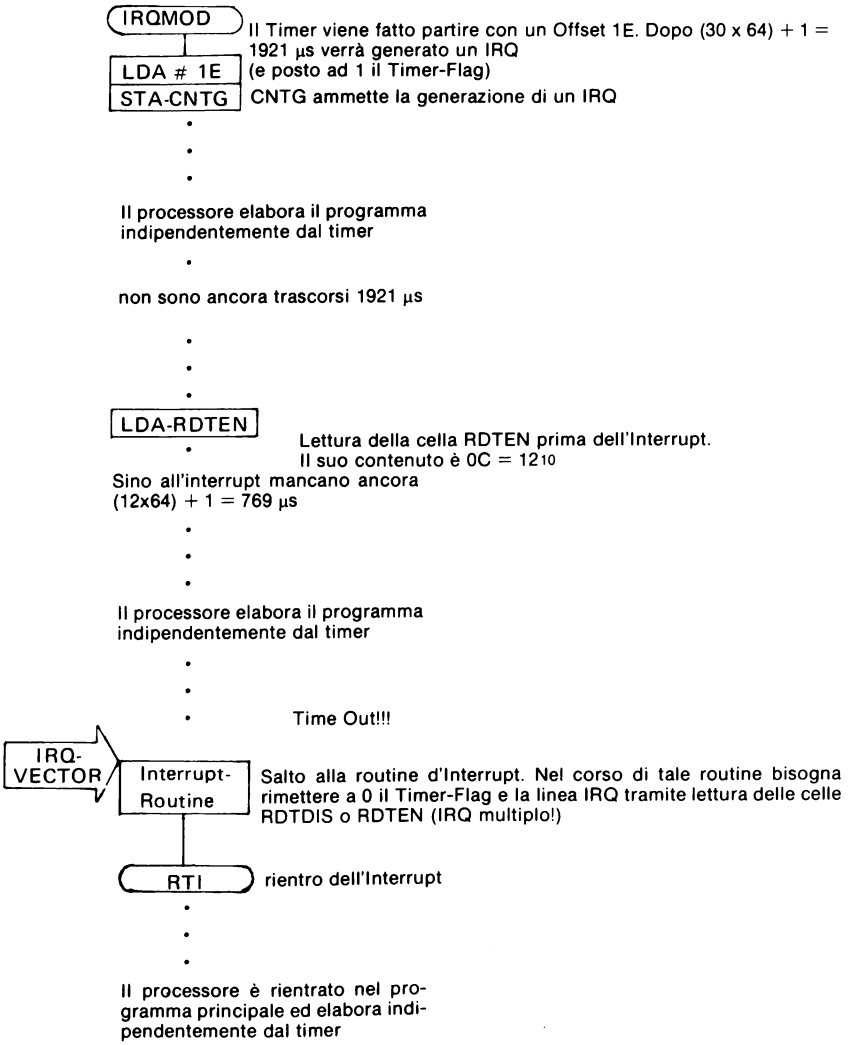
   A seconda del fattore di divisione il tempo in cui il ciclo del timer si completa può risultare diverso. Dopo il Time Out (il contenuto della cella del timer è FF) si ha che:
   - La linea IRQ passa allo stato logico 0, generando così un Interrupt Request (se il Flag I nel registro di stato vale 0: CLI).
   - Il Timer-Flag (b7 nel registro Flag) è posto ad 1.

7. Il processore può anche leggere il contenuto istantaneo della cella del timer. La lettura della cella RDTDIS all'indirizzo 1AD4 verifica quanto tempo manca sino al Time Out. La lettura della cella RDTDIS impedisce al timer di emettere un IRQ. La lettura della cella RDTEN all'indirizzo 1ADC informa pure sul tempo mancante sino al Time Out. La lettura della cella RDTEN, tuttavia, consente al timer di emettere un IRQ. Con l'operazione di lettura delle celle del timer RDTEN e RDTDIS il timer degli intervalli può venir commutato dal Polling - all'Interrupt-Mode o viceversa. L'operazione di lettura non modifica il contenuto del timer, ma fornisce solo informazioni sul suo stato momentaneo. Se l'operazione di lettura avviene prima del Time Out viene mantenuto il fattore di divisione in vigore. Un'operazione di lettura delle celle RDTDIS e RDTEN dopo il Time-Out informa sul tempo passato dopo il Time Out, in µs. Le considerazioni fatte saranno

meglio illustrate dagli esempi che seguono:
Cella del timer CNTG:

- Offset del timer = 1E = 30_{10}
- Fattore di divisione = 64
- CNTG può generare un IRQ
- Il tempo dalla partenza del timer degli intervalli sino alla generazione dell'IRQ (ed alla messa ad 1 del Timer-Flag) vale: (30 x 64) + 1 = 1921 µs.

Il diagramma di flusso seguente illustra la partenza del timer degli intervalli e la lettura nelle celle del timer in **Interrupt-Mode:**



•
•
•
•
•
•

LDA-RDTEN

Lettura della cella RDTEN dopo l'Interrupt.
Il suo contenuto sia A4.

•
•
•
•

A4 = 1Ø1ØØ1ØØ contenuto istantaneo della cella
$\overline{A4}$ = Ø1Ø11Ø11 suo complemento = 5B
$\overline{A4}$ + 1 = 5B + 1 = 5C = $92_{1Ø}$ ((complemento a 2)

Dall'Interrupt sono trascorsi 92 µs.
In totale dalla partenza del timer sono trascorsi 1921 + 92 = 2013 µs

•
•
•

Il timer decrementa di 1 ogni µs sinché il processore non lo fa partire

•
•
•

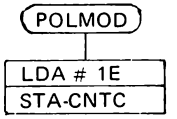
8. *Polling Mode:* Il Timer può anche funzionare in *Polling-Mode*. Il Timer cioé viene periodicamente o saltuariamente interrogato se si è già verificato il Time Out. In questo caso il programmatore vieta all'Interval-Timer di generare un Interrupt Request dopo un Time Out. La scrittura nelle celle CNTA ... CNTD o la lettura nella cella RDTDIS del timer impediscono all'Interval Timer di generare un IRQ. Quando il timer ha terminato il suo ciclo il Timer-Flag nel registro Flag viene posto ad 1 ma la linea IRQ rimane allo stato logico 1. Pertanto non può venir generato un Interrupt Request. *Nel Polling Mode il criterio per il Time Out è fissato dal Timer-Flag nel registro Flag e non dalla linea IRQ.* L'esempio che segue illustrerà il decorso dei programmi in Polling Mode. Utilizzeremo anche una nuova istruzione, che non ci è nota dal 1° volume: BIT. Con l'istruzione BIT si possono controllare i valori dei bit nella memoria del sistema ed anche di bit nel registro Flag del 6532. Il funzionamento generale dell'istruzione BIT è il seguente:

* A ∧ M il contenuto dell'accumulatore non è modificato, il risultato dell'operazione logica AND agisce sul Flag Z

* $M_7 \rightarrow N$ il b7 della cella di memoria viene trasferito al Flag N nel registro Processor Status (registro P nella CPU)
* $M_6 \rightarrow V$ il b6 della cella di memoria viene trasferito nel Flag V del registro Processor Status
* L'istruzione BIT dispone dei modi d'indirizzamento in Pagina Zero ed Assoluto.
* Nello Junior-Computer l'istruzione BIT-RDFLAG ha i seguenti effetti:
  - b7 nel registro Flag = Timer-Flag = Flag N (registro P nella CPU)
  - b6 nel registro Flag = PA7-Flag = Flag V (registro P nella CPU)

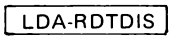
  Quando il processore scrive in una delle celle CNTA...CNTD del timer, viene fatto partire il timer, ed automaticamente rimesso a 0 il Timer-Flag (b7 nel registro Flag).
* La scrittura in una delle celle CNTA ... CNTD provoca i seguenti effetti:
  - il 6532 non può emettere Interrupt Request sulla linea IRQ
  - il Timer-Flag (b7 nel registro Flag) viene sempre rimesso a 0.
* A seconda del fattore di divisione il ciclo completo del timer dura più o meno a lungo. Dopo il Time Out (contenuto della cella del timer: FF) si verifica quanto segue:
  - la linea IRQ rimane allo stato logico 1 e non genera quindi IRQ
  - il Timer-Flag (b7 nel registro Flag) viene posto ad 1.
* Se il processore legge nelle celle RDTDIS del timer prima di un Time Out, ossia prima che il Timer-Flag sia passato ad 1, il valore letto fornisce il tempo che manca sino al momento in cui il Timer-Flag passerà ad 1. Riprendiamo il nostro esempio riferendoci alla cella CNTC del timer:
  - Timer-Offset = 1E = 30_{10}
  - Fattore di divisione = 64
  - CNTC non può generare un IRQ
  - Il tempo dalla partenza del timer sino a che il Timer-Flag passi ad 1 vale: (30 x 64) + 1 = 1921 µs.

Il diagramma di flusso che segue illustra la partenza del timer degli intervalli e la lettura nel registro Flag RDFLAG. Il timer verrà interrogato in Polling Mode, una volta periodicamente ed un'altra in modo saltuario. Il registro Flag RDFLAG fornisce indicazioni sullo stato momentaneo del timer:



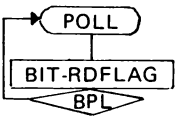
Si fa partire il Timer con l'Offset 1E. Dopo (30x64) + 1 = 1921 μs si porta ad 1 il Timer-Flag e non si possono quindi generare IRQ. CNTC non abilita alcun Interrupt Request.

Il processore elabora il programma indipendentemente dal timer. Non sono ancora trascorsi 1921 μs



La cella del timer viene letta in modo *non periodico*. Il Timer-Flag è a 0, dato che il timer non ha completato il suo ciclo. Il valore della cella sia 0C = 12_{10}

Mancano ancora (12x64) + 1 = 796 μs alla rimessa ad 1 del Timer-Flag. Il processore elabora il programma indipendentemente dal timer.



Il registro Flag viene letto in modo *periodico*. Il processore sta elaborando in un loop di attesa *in dipendenza* dal timer. Se il b7 in RDFLAG è 0, il ciclo del timer non è ancora completo. Quanto il b7 vale 1, invece, il timer ha completato il ciclo e il processore esce dal loop POLL..BPL..POLL. Il programma riprende indipendentemente dal timer.

Time Out!!!

Lettura della cella del timer dopo il Time Out. Automaticamente viene rimesso a 0 il Timer-Flag. Il contenuto della cella sia p.es. ancora A4.

A4 = 10100100

$\overline{A4}$ = 01011011 = 5B (complemento)

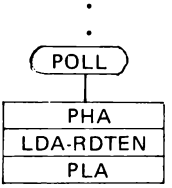
$\overline{A4}$ + 1 = 5B +1 = 5C = 92_{10} (complemento a 2)

Sono trascorsi 92 μs da quando il Timer-Flag è stato messo ad 1. Dalla partenza del timer sono trascorsi 1921 + 92 = 2013 μs.

Il timer decrementa di 1 ad ogni µs, sino a che viene riavviato dal processore.

•
•
•

9. Come già accennato, il Timer può funzionare sia in Interrupt che in Polling-Mode. A volte il programmatore desidera passare da un modo all'altro. Ossia, se ad es. il timer marcia in Polling-Mode ed il Timer-Flag è ancora a 0, l'utente può portare il timer a funzionare in Interrupt-Mode, senza che sia necessario farlo ripartire di nuovo:



Il timer degli intervalli funziona in Polling-Mode

salva il contenuto dell'Accu sullo Stack
commuta in Interrupt-Mode
ritorna allo stato originario

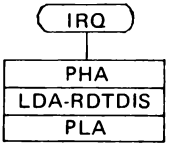
•
•
•

Il Timer degli intervalli opera in Interrupt-Mode. *Il fattore di divisione rimane inalterato anche in caso di lettura d'una cella del timer.*

•
•
•

La commutazione dall'Interrupt Mode al Polling-Mode avviene così:

•
•



Il timer degli intervalli funziona in Interrupt Mode

salva il contenuto dell'Accu sullo Stack
commuta in Polling-Mode
ritorna allo stato originario

•
•

Il Timer degli intervalli opera ora in Polling-Mode. *Il fattore di divisione non viene modificato dalla lettura in una cella del timer.*

•
•
•

10. *Lettura e scrittura nelle celle del Timer.*
Per le operazioni di lettura e scrittura nelle celle del timer è necessario tener conto delle particolarità seguenti:

* *Lettura delle celle RDTDIS e RDTEN prima del Time Out:*

- Tramite un'operazione di lettura il microprocessore apprende il contenuto istantaneo della cella del timer interessata. Durante l'elaborazione del programma il computer è in grado di calcolare quanto tempo gli rimane prima del completamento del ciclo del timer.
- Una lettura della cella RDTDIS del timer fa sì che il timer commuti in Polling-Mode, ed al Time Out non sia più in grado di generare un IRQ. In corrispondenza al Time Out viene solo posto ad 1 il Timer-Flag.
- Una lettura della cella RDTEN del timer fa sì che il timer commuti in IRQ-Mode; ed al Time Out avviene questo:
  a) il Timer-Flag viene posto ad 1,
  b) la linea IRQ passa dallo stato logico 1 a 0: corrispondentemente il timer emette un IRQ.
- I fattori di divisione per 1, 8, 64, 1024 sono fissati da una operazione di scrittura nelle celle CNTA ... CNTH del timer. La lettura delle celle RDTDIS e RDTEN del timer non modifica il fattore di divisione!
- Se dopo il Time Out viene letta una cella del timer, tale operazione di lettura rimette a 0 il Timer-Flag e la linea IRQ; il Timer-Flag rimane invece ad 1 quando la lettura d'una cella del timer coincide nel tempo con un Timer-Interrupt.
- Se il timer ha generato un Interrupt, il microprocessore salta ad una routine d'Interrupt. In essa deve essere prevista la lettura della cella RDTDIS, per rimettere a 0 la linea IRQ ed il Timer-Flag: in tal modo ci si assicura che dopo il rientro dalla routine d'Interrupt non venga generato ancora una volta lo stesso Interrupt.

* *Scrittura nelle celle CNTA ... CNTH del Timer:*

- Una scrittura nelle celle CNTA ... CNTD del timer fa sì che il timer parta in Polling-Mode. Alla partenza del timer resta anche fissato il fattore di divisione, che non si modifica più sino al Time Out.
- Una scrittura nelle celle CNTE ... CNTH del timer fa sì che il timer parta in IRQ-Mode. Alla partenza del timer resta pure fissato il fattore di divisione, che non si modifica più sino al Time Out.
- Un'operazione di scrittura in una delle celle CNTA ... CNTG del timer rimette in ogni caso a 0 il Timer-Flag e la linea IRQ.

Abbiamo così concluso la descrizione del Timer. Abbiamo imparato come si lavora con le 8 celle del timer ed il suo Flag. Abbiamo

ora anche un'idea più chiara sui concetti di Polling ed Interrupt-Mode. Prima di procedere ad un impiego pratico del timer degli intervalli sulla base di due programmi didattici, dobbiamo descrivere un altro elemento del CI 6532: il rivelatore dei fronti.

Il rivelatore dei fronti nel CI 6532

Lo Junior-Computer dispone anche di un rivelatore dei fronti. Esso risulta programmabile e consiste di quattro registri driver. Le denominazioni di questi registri sono:
EDETA, EDETB, EDETC ed EDETD. In fig. 8 sono mostrati gli indirizzi di tali registri.

* *L'Hardware del rivelatore dei fronti*
L'Hardware del rivelatore dei fronti si può descrivere in brevi tratti. Il rivelatore dei fronti possiede un input ed un output. L'input è costituito dalla linea PA7 del PIA, e l'output dalla linea IRQ del CI 6532. L'ingresso del rivelatore dei fronti è programmabile in modo che la linea IRQ venga attivata in corrispondenza al fronte positivo o negativo d'un impulso applicato alla linea PA7. Il programmatore ha tuttavia la facoltà di inibire al rivelatore dei fronti la generazione di un IRQ.
Quando il microprocessore opera in collegamento al rivelatore dei fronti, la linea PA7 deve venir programmata quale input.
La fig. 10 ci presenta due impieghi comuni del rivelatore dei fronti. Nel primo caso su PA7 viene inviato un segnale dati seriale; nel secondo vengono utilizzate tutte le linee della Porta A per leggere nello Junior-Computer un segnale parallelo largo 7 bit da una tastiera ASCII.

- *Ingresso seriale su PA7*
  Come sappiamo dal 1° volume, il computer elabora tutti i dati in modo parallelo. I dati sono della "larghezza" (capacità) di 8 bit.

Se ad esempio lo Junior-Computer deve lavorare con una stampante, un videodisplay o l'Elekterminal, il relativo scambio dati avviene in modo seriale in codice ASCII. In che consista questo codice verrà descritto ampiamente nel 3° volume; per ora le procedure pratiche con il codice ASCII non sono qui importanti. La fig. 10a illustra l'andamento nel tempo di un segnale ASCII seriale. Esso comprende:
1. un bit iniziale
2. 8 bit dati B0 ... B7
3. un bit di parità (Parity-Bit)
4. due bit terminali (Stop-Bit).

Ad ogni bit in un segnale ASCII seriale è assegnato un determinato tempo-bit t_2: l'intervallo fra bit e bit è quindi costante. La trasmissione d'un informazione seriale comincia sempre con un bit iniziale (Start-Bit). Se la linea sulla quale vengono trasmesse le informazioni seriali, è allo stato logico 1, significa per il computer che non seguono dati. Un salto della tensione sulla linea da 1

logico a 0 logico corrisponde quindi ad uno Start-bit. Questa variazione di tensione è facilmente avvertibile mediante il rivelatore dei fronti del CI 6532. Nell'esempio che stiamo considerando il rivelatore dei fronti va programmato dunque in modo da reagire al fronte di discesa (= negativo) (inizio dello Start-bit).
Dopo che lo Junior-Computer ha riconosciuto tramite il rivelatore dei fronti uno start-bit, la lettura del successivo segnale ASCII seriale diventa per lui cosa assai semplice:

1. Viene riconosciuto lo Start-bit (fronte negativo su PA7)
2. Il microcomputer attende 1,5 tempi-bit e preleva quindi un "campione" da PA7; ossia, verifica se la tensione su PA7 ha lo stato logico 0 od 1. Corrispondentemente può assegnare uno "0" od "1" al bit dati B0. Un tempo bit dopo è il bit dati b1 a presentarsi sulla linea di porta PA7. Lo Junior-Computer verifica nuovamente se si tratti di "0" od "1". Questo processo si ripete sino a quando alla fine è stato letto il bit dati b7. In tutti i casi i singoli bit vengono esaminati in corrispondenza del tempo di mezzo per accertarne lo stato logico di 0 od 1.
3. Un tempo-bit dopo i bit dati b7 sulla linea PA7 si presenta il bit di parità (Parity-bit). Esso viene trasmesso a fini di controllo: esso infatti segnala se il formato di bit dati appena trasmesso contiene un numero pari o dispari di 0 o di 1. In tal modo il computer può facilmente controllare se nella trasmissione dei dati è stato commesso un errore.
4. La trasmissione dei dati è conclusa con due bit terminali (Stop-bit). Si noti che i due Stop-bit hanno polarità invertita rispetto allo Start-bit (Start-bit=0 log., Stop-bit=1 log.).
5. Il microcomputer attende sin quando la linea dati assume lo stato logico 0. Il rivelatore dei fronti nel CI 6532 avverte questo salto di tensione, ed il processo ora descritto ricomincia.

- *Ingresso parallelo sulla Porta A*
  L'Junior-Computer è in grado di leggere segnali d'ingresso a disposizione parallela sulla Porta A (lo stesso vale naturalmente anche per la Porta B).

In fig. 10b è illustrato il collegamento di una tastiera ASCII, la quale fornisce il codice ASCII di un tasto premuto in modo parallelo (b0 ... b6). Si ricordi che il codice ASCII ha una "larghezza" di 7 bit (sarà descritto in dettaglio nel 3° volume). Di norma, una tastiera fornisce il codice in modo parallelo. Non appena il codice di tastiera b0...b6 è presente stabilmente sulle linee dati, la tastiera emette un impulso: questo impulso di tastiera nei computer viene chiamato "Data Strobe". Il rivelatore dei fronti è in grado di riconoscere facilmente anche il Data-Strobe. Nel nostro esempio si tratta di impulso positivo. Per lo Junior-Computer un fronte positivo d'impulso su PA7 significa: sulle linee dati della tastiera è

ADH = 1 A								ADL								
1				A												
A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	128 bytes PIA-RAM
X(0)	X(0)	X(0)	X(1)	X(1)	X(0)	1	X(0)	0	X	X	X	X	X	X	X	(1A00 . . . 1A7F)
							1A80	1	X(0)	X(0)	X(0)	X(0)	0	0	0	PAD
							1A81	1	X(0)	X(0)	X(0)	X(0)	0	0	1	PADD
							1A82	1	X(0)	X(0)	X(0)	X(0)	0	1	0	PBD
							1A83	1	X(0)	X(0)	X(0)	X(0)	0	1	1	PBDD
							1AF4	1	X(1)	X(1)	1	0	1	0	0	CLK1T
							1AF5	1	X(1)	X(1)	1	0	1	0	1	CLK8T
						inibisci	1AF6	1	X(1)	X(1)	1	0	1	1	0	CLK64T
						Timer-IRQ	1AF7	1	X(1)	X(1)	1	0	1	1	1	CLK1KT
							1AFC	1	X(1)	X(1)	1	1	1	0	0	CLK1T
							1AFD	1	X(1)	X(1)	1	1	1	0	1	CLK8T
						abilita	1AFE	1	X(1)	X(1)	1	1	1	1	0	CLK64T
						Timer IRQ	1AFF	1	X(1)	X(1)	1	1	1	1	1	CLK1KT
						inibisci Timer-IRQ	1AD4	1	X(1)	X(0)	X(1)	0	1	X(0)	0	read Timer
						abilita Timer-IRQ	1ADC	1	X(1)	X(0)	X(1)	1	1	X(0)	0	leggi Timer
							1AD5	1	X(1)	X(0)	X(1)	X(0)	1	X(0)	1	leggi Flag Register
						inibisci PA7-IRQ	1AE4	1	X(1)	X(1)	0	X(0)	1	0	0	scrivi neg EDET
						inibisci PA7-IRQ	1AE5	1	X(1)	X(1)	0	X(0)	1	0	1	scrivi pos EDET
						abilita PA7-IRQ	1AE6	1	X(1)	X(1)	0	X(0)	1	1	0	scrivi neg EDET
						abilita PA7-IRQ	1AE7	1	X(1)	X(1)	0	X(0)	1	1	1	scrivi pos EDET

Figura 9. Questa tabella indica quale formato di bit deve essere presente sul bus indirizzi perché vengano indirizzati i diversi registri del CI 6532. Tramite la linea RS = A7 il computer commuta dalla RAM al Timer e rivelatore dei fronti.

presente un formato di bit che deve essere letto ed introdotto nel computer.
Il significato e gli scopi del rivelatore dei fronti dovrebbero risultare chiari da questi due tipi d'impiego. Il rivelatore dei fronti si dimostra dispositivo indispensabile per la trasmissione dei dati fra il computer e le unità periferiche collegate. Nel 3° volume vedremo qualche altro esempio di applicazione del rivelatore dei fronti.

Il modello di programmazione del rivelatore dei fronti

Dal punto di vista della software, il rivelatore di fronti consta di quattro registri driver. Le loro denominazioni sono EDETA, EDETB, EDETC ed EDETD. I relativi indirizzi sono indicati in fig. 9. Scrivendo un determinato byte in uno di questi registri si comunica al rivelatore dei fronti se deve reagire ad un fronte positivo o negativo d'un impulso sulla linea di porta PA7.
Il rivelatore dei fronti lavora assieme al registro Flag del CI 6532. Il flag per il rivelatore di fronti nel registro Flag è b6. In seguito lo chiameremo *PA7-Flag*. Se ad esempio il rivelatore dei fronti è programmato in modo da reagire ai fronti negativi su PA7, quando interviene un tal tipo d'impulso il PA7-Flag nel registro Flag viene posto ad 1. Lo stesso vale naturalmente se programmato per reagire ad un fronte positivo su PA7.
Analogamente al modo in cui impulsi positivi o negativi possono

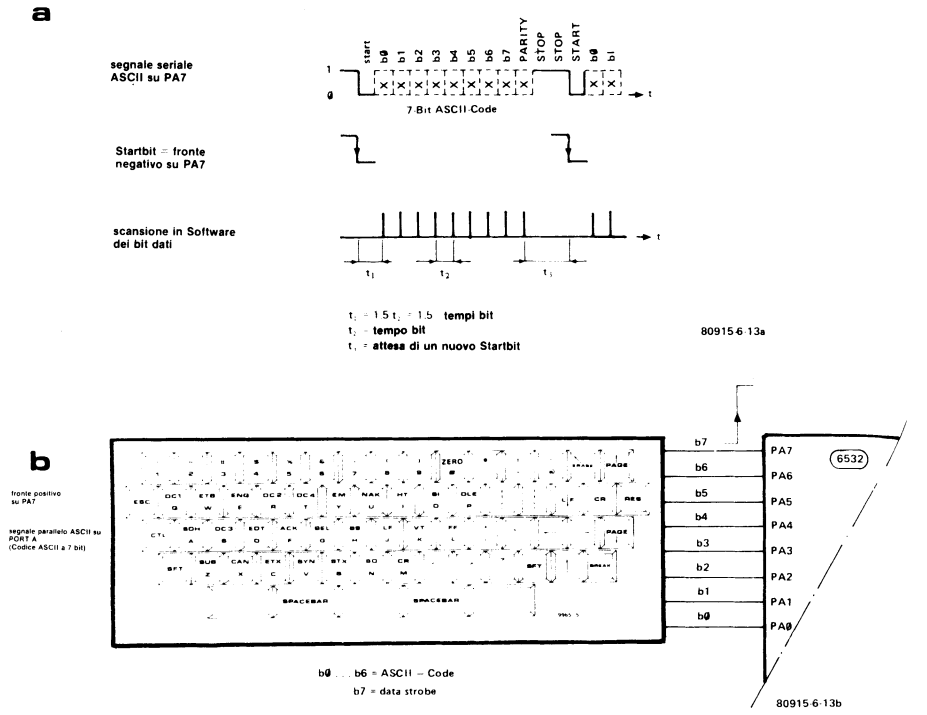


Figura 10. Un esempio di applicazione pratica del rivelatore dei fronti. In fig. 10a la linea PA7 è programmata per ricevere un segnale seriale. Il rivelatore dei fronti attende il fronte negativo del bit iniziale (Startbit). Appena giunge un impulso di questo tipo, il computer inizia a leggere il formato seriale dei bit. In fig. 10b, il computer legge un segnale parallelo sulla Porta A. Un fronte positivo d'impulso su PA7 segnala al computer di accogliere un carattere ASCII dalla tastiera.

influire sul PA7-Flag, essi possono venire impiegati pure per gestire la linea d'Interrupt. Se il programmatore abilita il rivelatore dei fronti a generare un Interrupt Request (a seguito della scrittura di un qualsiasi formato di bit in uno dei registri driver), la linea d'Interrupt del CI 6532 viene messa a 0 e contemporaneamente il PA7-Flag ad 1. Tra il Timer-Flag ed il PA7-Flag si possono indicare quindi le seguenti analogie:

- *Timer Flag*: il Timer-Flag è portato automaticamente ad 1 quando si verifica un Time Out. Il programmatore può impedire al timer di generare un IRQ. Se invece il timer è abilitato a generare un IRQ, contemporaneamente il Timer Flag passa ad 1 e la linea IRQ a 0.
- *PA7-Flag:* si porta automaticamente ad 1 quando su PA7 è applicato un fronte d'impulso. Il rivelatore dei fronti reagirà,

ponendo ad 1 il PA7-Flag, ai fronti positivi o negativi in funzione delle operazioni di scrittura nei registri driver. Il programmatore può impedire al rivelatore di fronti di generare un IRQ. Se invece il rivelatore di fronti è abilitato a generare un IRQ, contemporaneamente il PA7-Flag passa ad 1 e la linea IRQ a 0.

- *Ricapitolando:* Il Timer-Flag ed il PA7-Flag sono situati nel registro Flag del CI 6532. I due Flag sono indipendenti l'uno dall'altro. Entrambi i Flag possono essere verificati in Polling-Mode.

Per il Timer-Flag si ha:

BIT-RDFLAG	verifica lo stato del Timer-Flag
BPL	il Timer-Flag è alto (log. 1)? se sì, effettua i salto; se no, attendi

Per il PA7-Flag si ha:

BIT-RGFLAG	il PA7-Flag è alto (log. 1)? se sì, effettua il salto; se no, attendi
BVC	verifica lo stato del PA7-Flag

Definizione del tipo di fronte d'impulso su PA7

Tramite un'operazione di **scrittura** in uno dei 4 registri driver EDETA ... EDETD si comunica al rivelatore dei fronti se deve reagire ad un fronte positivo o negativo su PA7. La scrittura d'un qualsiasi formato di bit nei registri di governo EDETA ... EDETB fa sì che quando il PA7-Flag viene portato ad 1 non possa venire generato un Interrupt Request. Se il rivelatore di fronti deve invece generare un Interrupt Request quando si presenta su PA7 un dato fronte d'impulso, si richiede una operazione di scrittura in uno dei registri driver EDETC ... EDETD. Il tutto viene chiarito dai seguenti esempi:

STA-EDETA: Il rivelatore dei fronti reagisce ad un fronte d'impulso negativo su PA7. Quando ciò accade, il PA7-Flag nel registro Flag viene posto ad 1 e *non viene generato alcun IRQ.*

STA-EDETB: Il rivelatore dei fronti reagisce ad un fronte d'impulso positivo su PA7. Quando ciò accade, il PA7-Flag nel registro Flag viene posto ad 1 e *non viene generato alcun IRQ.*

STA-EDETC: Il rivelatore dei fronti reagisce ad un fronte d'impulso negativo su PA7. Quando ciò accade, il PA7-Flag nel registro Flag viene posto ad 1 e *viene generato un IRQ.*

STA-EDETD: Il rivelatore dei fronti reagisce ad un fronte d'impulso positivo su PA7. Quando ciò accade, il PA7-Flag nel registro Flag viene posto ad 1 e *viene generato un Interrupt.*

Altre particolarità del CI 6532

Come già sappiamo, il Timer ed il Rivelatore dei fronti lavorano assieme al registro Flag RDFLAG. b7 è il relativo Timer-Flag e b6 il PA7-Flag. Se l'utente abilita il timer od il rivelatore di fronti a generare un IRQ, quando il Timer-Flag od il PA7-Flag sono portati ad 1 la linea IRQ passa a 0. Per non generare più volte di seguito lo stesso IRQ, la linea di IRQ deve sempre venir resettata (= portata a 0) non appena il processore giunge nella routine d'Interrupt. Quando si resetta la linea IRQ occorre sempre badare ai seguenti particolari:

1) *Il Timer genera un IRQ:* se il programmatore ha abilitato il timer degli intervalli a generare un IRQ, dopo il Time Out il Timer-Flag viene messo ad 1: la linea d'IRQ genera così un Interrupt Request. La CPU effettua un salto in una routine di Interrupt. All'inizio di questa routine occorre procedere a resettare a 0 la linea IRQ, al fine di evitare che, dopo il rientro (RTI) dalla routine d'Interrupt, il medesimo IRQ del timer venga attivato ad un impulso iniziale.
   Il resettaggio della linea IRQ e del Timer-Flag durante la routine d'Interrupt sono provocati da una operazione di scrittura in una delle celle CNTA ... CNTH del timer, o da un'operazione di lettura nelle celle RDTDIS o RDTEN del timer. Una operazione di scrittura in una delle celle del timer fa ripartire di bel nuovo il timer e definisce al tempo stesso se il timer potrà o meno emettere un IRQ. Un'operazione di lettura in una delle celle del timer non fa ripartire il timer, ma solo definisce lo stato dell'IRQ (ammesso o no).

2) *Il rivelatore di fronti genera un IRQ*: se il programmatore ha abilitato il rivelatore dei fronti ad emettere un Interrupt Request, quando si presenta su PA7 un determinato fronte d'impulso viene messo ad 1 il PA7-Flag e la linea IRQ genera un Interrupt Request. La CPU effettua un salto in una routine d'Interrupt. Nel corso di questa routine bisogna resettare a 0 la linea IRQ, per evitare che, dopo il rientro (RTI) dalla routine d'Interrupt, venga generato nuovamente il medesimo IRQ di PA7. *Il resettaggio della linea IRQ e del PA7-Flag durante la*

routine d'Interrupt sono provocati da un'operazione di lettura del registro Flag RDFLAG.

3) Il vettore IRQ nello Junior-Computer si trova nelle locazioni 1A7E e 1A7F della RAM. Il computer può quindi nel corso di un programma definire o modificare il vettore IRQ. In tal modo è possibile attribuire sia al Timer che al Rivelatore di fronti proprie routine Interrupt.
Il processore può riconoscere, leggendo nel registro Flag RDFLAG, quale dei due Flag ha generato un Interrupt. Se è il Timer-Flag a risultare allo stato logico 1, il computer dispone il vettore IRQ verso la Timer-Interrupt-Routine; nell'altro caso lo disporrà indirizzato alla PA7-Interrupt-Routine. È anche possibile "annidare" le due routine Interrupt l'una entro l'altra. Se si vuole assicurare una velocità di elaborazione elevata, è pratica comune l'annidamento di più routine di Interrupt. Ci torneremo sopra nel corso d'un prossimo volume.

4) *La linea RES:* tramite la linea RES viene fatto partire lo Junior-Computer. Questa linea agisce pure su diversi registri del CI 6532. Se questa linea assume lo stato logico 0 si ha:
   - Il contenuto di tutti i registri I/O viene azzerato. Pertanto tutte le linee delle Porte A e B vengono definite input. Non può così avvenire per errore che unità periferiche distruggano il contenuto dei registri output PAD e PBD.
   - Non può venir generato alcun Timer-IRQ o PA7-IRQ; escludendo in tal modo che il processore salti in una routine d'Interrupt per cui non è stato fissato il vettore IRQ.
   - Il segnale RES programma il rivelatore di fronti per reagire ad un impulso negativo su PA7. Durante il reset tuttavia può accadere che per errore il PA7-Flag venga posto ad 1 ad opera di un'unità esterna.

Prima di impiegare il rivelatore di fronti, perciò, bisogna provvedere a resettare a 0 il PA7-Flag con un'operazione di lettura del registro Flag RDFLAG.

Così ora sappiamo tutto sui registri del CI 6532. Nei programmi didattici che seguono verrà mostrato dettagliatamente come programmare il Timer ed il PIA. Abbiamo volutamente tenuto fuori da questo discorso il rivelatore di fronti, dato che esso sarà oggetto di approfondimento in un successivo volume. La fig. 9 mostra pure il tipo di formato di bit che deve presentarsi sul bus indirizzi quando il microprocessore vuole interrogare i registri del CI 6532. A tal fine viene impiegata la linea indirizzo A7 quale linea $\overline{RS}$ ($\overline{RS}$ = RAM Select). Se $\overline{RS}$ vale 0 logico, sono interessate le 128 celle di RAM del CI 6532. Se invece $\overline{RS}$ è 1, la CPU opera coll'Interval-Timer, il PIA od il rivelatore dei fronti. Per quanto riguarda gli indirizzi: 1A00 ... 1AF7 costituiscono la RAM, e 1A80 ... 1AFF sono gli indirizzi dei restanti registri. Va precisato che non tutti gli

indirizzi da 1A84 a 1AFF trovano impiego: ciò dipende dall'incompletezza del codice indirizzi sul chip del 6532. La perdita di un paio di celle di memoria su un totale di 65 kbyte risulta comunque trascurabile.
La fig. 11 presenta lo schema a blocchi completo dei circuiti del CI 6532, ossia la configurazione Hardware del PIA, del timer degli intervalli, del rivelatore di fronti e della RAM. La corrispondente configurazione Software, ossia il modello di programmazione, è riportato in fig. 8.

Impiego pratico dell'Interval-Timer

1) *Il timer opera in Interrupt-Mode:* è tempo che impariamo ad adoperare l'Interval-Timer. Lo faremo scrivendo due programmi, che chiameremo INPUT e REPEAT. Con essi ci sarà possibile introdurre nello Junior-Computer, tramite la tastiera di fig. 4a, una melodia, e poi farcela risuonare a volontà. Il computer memorizza in questo caso le melodia in una determinata zona di memoria. Il programma INPUT è destinato all'introduzione della melodia nel computer.
Dopo aver inserito la melodia nel computer, vogliamo che esso la riproduca spontaneamente. A ciò provvede il programma REPEAT. Questa sezione descrive in primo luogo l'impiego pratico dell'Interval-Timer. Perciò faremo ricorso ad alcuni programmi precedentemente visti a proposito del programma PLAY (fig. 4). Si tratta delle subroutine KEYIN, KEYVAL ed EQUAL. Anche l'interfaccia, ossia l'amplificatore con altoparlante e la tastiera, è la stessa.
Il primo problema sta nell'introduzione nello Junior-

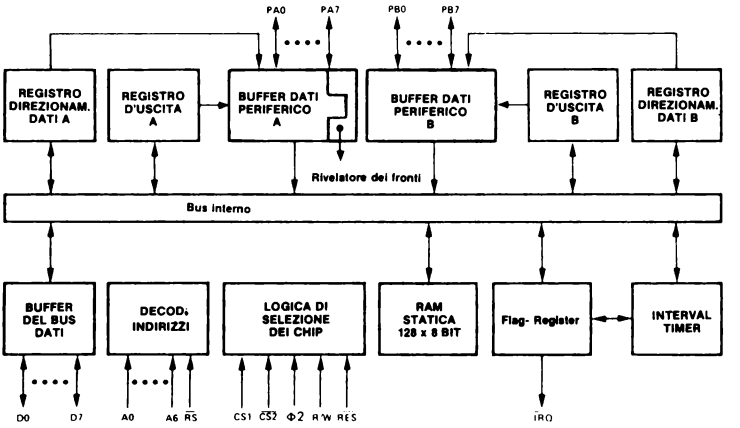


Figura 11. Lo schema a blocchi del CI 6532. Il PIA, il timer degli intervalli, il rivelatore dei fronti e la RAM sono collegati fra loro tramite il bus dati interno. Tutti i blocchi descritti separatamente sinora sono qui riuniti in un unico schema.

Computer di una melodia che successivamente il computer deve riprodurre, ossia suonare. Che informazioni dobbiamo inserire nel computer? Per poter riprodurre una nota, occorrono due informazioni:

1) L'altezza, ossia la frequenza della nota. Con il programma PLAY ci è stato possibile assegnare una data frequenza ad ogni tasto premuto nella tastiera. Lo stesso principio verrà impiegato pure nel programma INPUT.
2) Se il computer deve riprodurre la melodia introdotta, deve pure risultare definito il tempo per cui il tasto è rimasto premuto.

È quindi necessario poter calcolare la durata del periodo in cui il tasto è rimasto premuto. Questa durata la possiamo misurare nel modo più semplice mediante l'Interval-Timer. È chiaro dunque che il computer, per ogni nota suonata, deve memorizzare interamente sia la durata del periodo della nota, che la durata del tempo di pressione del tasto. Per ogni nota occorre quindi riservare due locazioni di memoria. Una misura della durata del periodo della nota è fornita dal valore del tasto premuto della tastiera. Dato che sono presenti 16 tasti, il valore del tasto va da 0 a F. La durata per cui il tasto è stato premuto viene misurata dal timer degli intervalli. Questa durata potrà assumere i valori 00 ... FF.
Ora possiamo riassumere i requisiti a cui deve soddisfare la routine d'inserzione INPUT:

1) Quando viene premuto un tasto nella tastiera, nell'altoparlante deve risuonare la corrispondente nota.
2) Contemporaneamente, il computer deve misurare la durata per cui il tasto resta premuto. Apparentemente i due fatti procedono parallelamente: la generazione del suono e la misura della durata per cui il tasto rimane premuto; le normali tecniche di programmazione non risultano idonee. Dobbiamo fare ricorso alla programmazione con gli Interrupt.
3) Ad ogni tasto premuto nella tastiera rimane assegnato un valore. Il computer calcola questo valore mediante la subroutine KEYVAL.
4) Si deve poter interrogare velocemente la tastiera: non vi sono problemi, impieghiamo la nota subroutine KEYIN. Riconosciuto il tasto, occore un programma che elimini ogni errore di rimbalzi dei tasti: a ciò provvede la nota subroutine DELAY.
5) Dopo che un tasto è stato rilasciato, l'altoparlante deve ammutolirsi, e debbono venir memorizzati sia il valore del tasto premuto che la durata di pressione del tasto. La memoria assegnata alla melodia avrà gli indirizzi 0100 ... 01D8 in Pagina 1. Quando questo spazio di memoria è completamente occupato, lo Junior-Computer si deve rifar vivo tramite il Monito ed ignorare ulteriormente la tastiera.

6) Prima di memorizzare la melodia nello spazio a ciò destinato, lo Junior-Computer deve predisporre tale zona di memoria all'inserimento della melodia. Ciò avviene riempiendo l'intera zona di memoria di 77, ed attivando il Pointer relativo.
7) Se ad esempio vengono premuti in successione i tasti dai valori 9, A, B e C, le informazioni memorizzate saranno:

Indirizzo	Dati
0100	09 = valore del tasto
0101	WW = 00 ... FF = durata della pressione del tasto
0102	0A = valore del tasto
0103	XX = 00 ... FF = durata della pressione del tasto
0104	0B = valore del tasto
0105	YY = 00 ... FF = durata della pressione del tasto
0106	0C = valore del tasto
0107	ZZ = 00 ... FF = durata della pressione del tasto
0108	77 = carattere di riempimento preliminare
.	.
.	.
.	.
01D7	77
01D8	77 = termine della memoria per la melodia

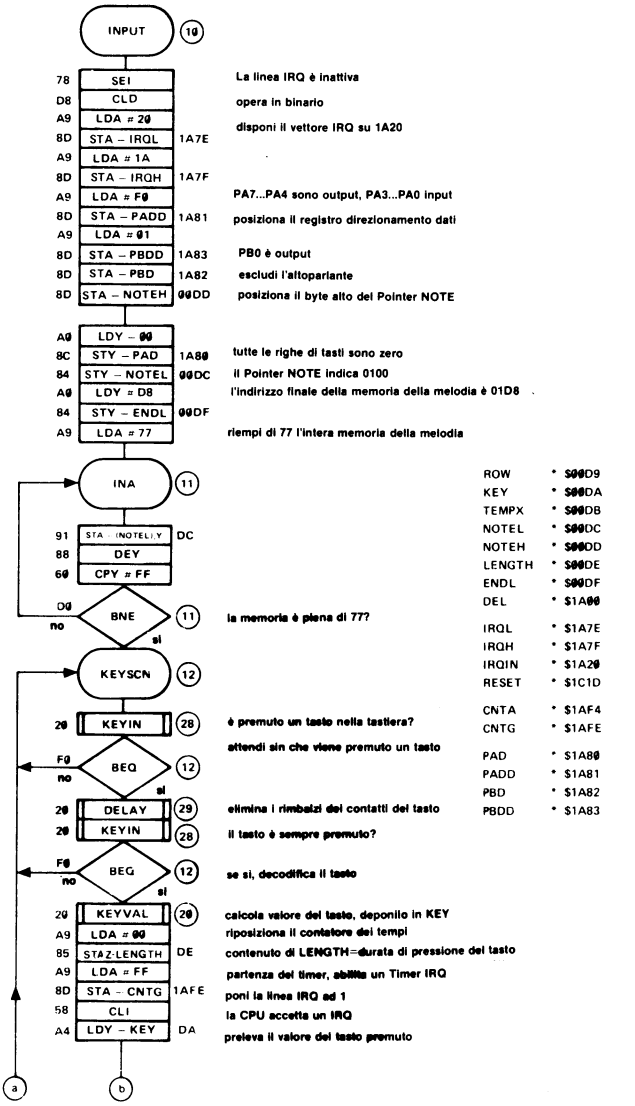
Il programma INPUT

I requisi posti, 1) ... 7), sono soddisfatti dal programma INPUT. Il relativo diagramma di flusso è dato in fig. 12. Al principio, il programma disabilità la linea IRQ: la relativa istruzione è SEI. Ciò impedisce la generazione per errore di un Interrupt.

Quindi, il computer depone il vettore IRQ all'indirizzo 1A20. A questo punto inizia la routine Interrupt (fig. 12c), mediante la quale si può misurare la durata della pressione d'un tasto nella tastiera. Una descrizione più precisa seguirà più avanti. Segue la programmazione del PIA. Conformemente alla fig. 4A, alla Porta A è collegata la matrice dei tasti della tastiera.

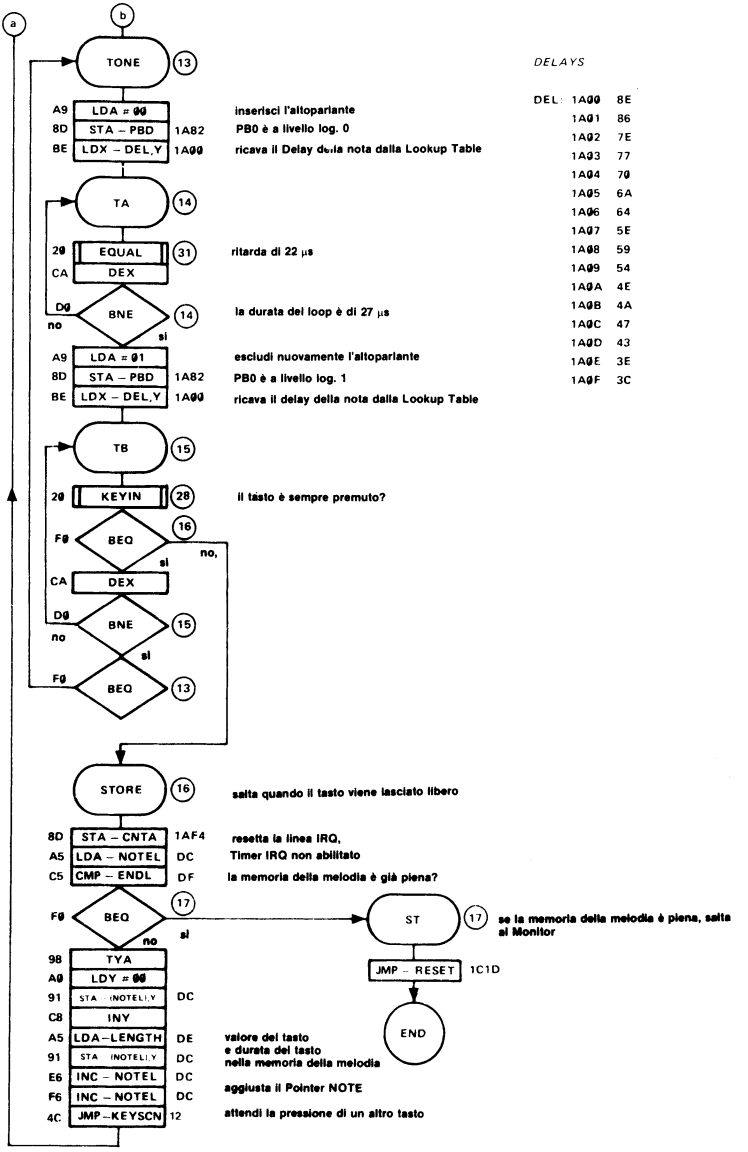
Pertanto, le linee PA7 ... PA4 vanno definite quali output e le linee PA3 ... PA0 input. L'amplificatore con relativo altoparlante è ancora collegato a PB0: perciò tale linea va definita quale uscita.
Inoltre resta ancora da disporre il Pointer NOTE (fig. 12d) per la zona di memoria della melodia. Questo pointer "punta" un indirizzo in Pagina 1, al quale va deposto il valore del successivo tasto premuto nella tastiera. All'inizio quindi tale pointer indica l'indirizzo 0100. L'indirizzo terminale della memoria per la melodia è 01D8. Quando il Pointer supererà tale valore, il computer dovrà fare ritorno nel Monitor. Nella locazione di memoria ENDL viene quindi depositato il valore $D8. Il confronto del byte basso indirizzo del Pointer NOTE con il contenuto di ENDL verificherà quindi se la memoria è ormai piena. Ora il computer perviene al Label INA. Si provvede alla predisposizione della memoria della melodia: il computer riempie di 77 le locazioni d'indirizzo 0100 ... 01D8.

12a



Poi il computer passa ad interrogare la tastiera. Se nessun tasto risulta premuto, esso si trattiene nel loop d'attesa KEYSCN-BEQ-KEYSCN. Se il contenuto dell'Accumulatore al rientro di KEYSCN risulta diverso da 0, significa che un qualche tasto è premuto. Si procede ad eliminare i rimbalzi (DELAY, fig. 4e). Se il tasto permane premuto, il computer salta alla subroutine KEYVAL (Fig. 4c). Ivi calcola il valore del tasto premuto: dato che i tasti sono 16, il campo di valori va da 00 a 0F. Al rientro da KEYVAL, il valore

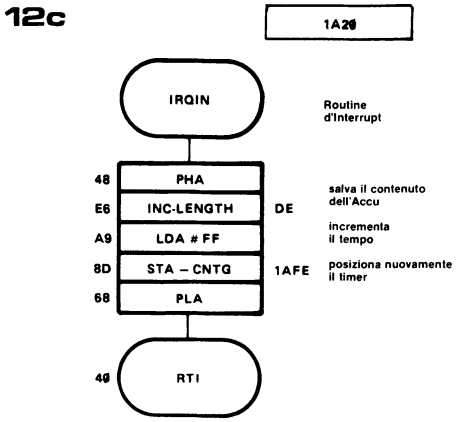
12b



DELAYS

DEL:	1A00	8E
	1A01	86
	1A02	7E
	1A03	77
	1A04	70
	1A05	6A
	1A06	64
	1A07	5E
	1A08	59
	1A09	54
	1A0A	4E
	1A0B	4A
	1A0C	47
	1A0D	43
	1A0E	3E
	1A0F	3C

calcolato del tasto viene deposto nella cella di memoria KEY. I diagrammi di flusso delle subroutine impiegate sono mostrati nelle fig. 4c, d, e, f.; la relativa descrizione è già stata fornita a proposito del programma PLAY.
Ed ora le cose si fanno interessanti! Dopo che il computer ha appreso il valore di un tasto premuto della tastiera, può emettere



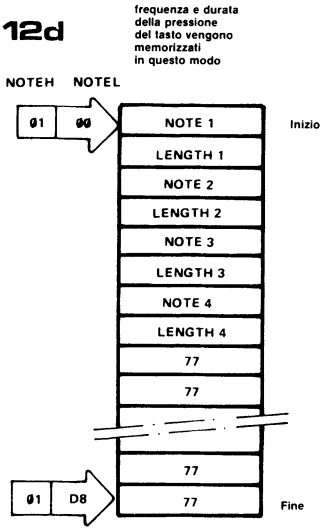


Figura 12a, b, c e d. Mediante il programma INPUT e la tastiera da piano è possibile memorizzare una melodia nello Junior-Computer. Quando si preme un tasto, nell'altoparlante risuona la nota corrispondente. Mentre il computer genera la nota esso misura anche la durata della pressione del tasto (non superiore a 4 secondi). Per poter svolgere entrambi questi compiti apparentemente contemporanei, il timer opera in IRQ Mode. In questa routine, come nel programma PLAY, il valore di un tasto deve venir convertito in una frequenza. A tal fine lo Junior Computer si avvale ancora della Lookup Table DEL (fig. 4g), nella quale legge la durata del semiperiodo relativo alla data frequenza. Quando il tasto viene rilasciato, il computer memorizza il valore del tasto e la durata di pressione del tasto nella memoria della melodia, in Pagina 1. Il programma modifica pure il Pointer indirizzi NOTE (fig. 12d) cosicché indichi sempre l'indirizzo a cui deve essere posta la prossima nota (=valore del tasto).

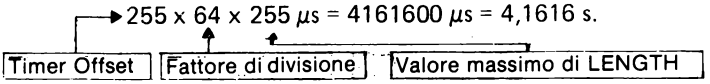
un suono tramite l'altoparlante. Per generare la nota esso si serve ancora della stessa Lookup Table DEL impiegata nel programma PLAY. (fig. 4g) La generazione della nota principia al Label TONE. Il programma ricalca fedelmente quello corrispondente di PLAY. Mentre il computer emette attraverso l'altoparlante la nota corrispondente al tasto premuto, deve anche misurare la durata di pressione del tasto. Perciò facciamo ricorso alla programmazione d'Interrupt; a tal fine fra la subroutine KEYVAL ed il Label TONE sono inserite appositamente alcune altre istruzioni, che hanno il seguente significato:

1) Il contenuto della locazione di memoria LENGTH è la misura della durata di pressione del tasto. Prima che lo Junior-Computer generi la nota relativa ad un tasto premuto, il contenuto di LENGHTH deve essere riportato a 0 (LDA 00, STAZ-LENGTH). Poi la CPU fa partire l'Interval Timer, scrivendo FF, offset del timer, nella cella CNTG del timer. Rivediamo a tal proposito le caratteristiche della cella CNTG del timer alla luce di fig. 8!

CNTG:
- Il fattore di divisione vale 64.
- Il timer degli intervalli è abilitato ad emettere un IRQ dopo un Time Out.
- Dopo che il timer è partito, esso genererà un Interrupt Request dopo (FF x 64) + 1 = (255 x 64) + 1 = 16321 µs.
- Nel momento che parte il timer la linea IRQ è messa a 0.

Dopo la partenza del timer e la corrispondente messa a 0 della linea IRQ, la CPU gestisce un Interrupt Request del timer. L'istruzione successiva, CLI, significa appunto: gestisci un interrupt Request, se il timer genera un IRQ.
Solo adesso il computer inizia la fase di generazione della nota. A tale scopo esso preleva dalla cella di memoria KEY il valore del tasto premuto, e quindi dalla Lookup Table DEL (fig. 4g) la durata del periodo della nota da suonare. Per la durata di un semiperiodo l'altoparlante è inserito, e per altro semiperiodo escluso (Label TA: altoparlante inserito; Label TB: altoparlante escluso). Durante l'emissione della nota, il computer, nella subroutine KEYIN (fig. 4d), controlla lo stato della tastiera. Viene così a sapere se il tasto è ancora premuto oppure no. Sin quando il tasto è premuto il computer emette la nota tramite l'altoparlante. Da periodo a periodo trascorre un tempo determinato; intanto l'Interval Timer marcia indipendentemente dalla CPU. Dopo 16321 µs il timer genera un IRQ: la linea IRQ ritorna a 0 ed il computer salta alla routine d'Interrupt IRQIN (fig. 12c). In essa il computer salva inizialmente sullo Stack il contenuto dell'Accu (PHA), e poi incrementa il contenuto della cella di memoria LENGTH. La CPU fa ripartire quindi il timer degli intervalli (LDA # FF, STA-CNTG), e rimette a zero la

linea IRQ. Da qui in avanti il timer marcia ancora indipendentemente dalla CPU. Prima del rientro dalla routine d'Interrupt viene ristabilito il contenuto originario dell'Accumulatore (PLA).
Dopo il rientro dalla Interrupt-Routine, il computer torna ad occuparsi della generazione della nota. Dopo 16320 µs il timer ha nuovamente terminato il suo ciclo. Il processore, dopo la generazione del Timer IRQ salta nuovamente alla routine d'Interrupt ed incrementa di uno il contenuto della cella LENGTH. La massima durata di pressione d'un tasto che il computer è in grado di misurare è quindi:



(Si sono trascurati i tempi necessari al computer per gestire un Interrupt Request e quello di permanenza nella Interrupt Routine). Il computer dunque è in grado di misurare la durata della pressione d'un tasto, mentre genera una nota. La durata massima della pressione d'un tasto ammonta a circa 4 secondi. Se viene superato tale tempo, lo Junior-Computer calcola un tempo errato: non è prevista tuttavia l'emissione d'un messaggio di errore (per mantenere il programma breve e comprensibile). Dopo ciascun Interrupt Request il computer abbandona la routine di generazione della nota, che inizia al Label TONE, per ritornarvi dopo l'esecuzione della routine d'Interrupt. Successivamente la CPU percorre periodicamente la subroutine KEY (fig. 4d). Quando finalmente il tasto viene rilasciato, il processore rientra da KEYIN con il valore 0 nell'Accu. Viene così introdotta la generazione della nota; nelle locazioni di memoria KEY e LENGTH stanno tutte le informazioni necessarie sul tasto appena rilasciato:

- in KEY sta il valore del tasto
- in LENGTH sta un numero che dà la durata di pressione del tasto.

Però, nel frattempo, il timer potrebbe emettere nuovamente un Interrupt Request, il che porterebbe ad un valore errato per la misura in LENGTH. Per evitare che il timer degli intervalli vada fuori controllo dopo l'abbandono del loop di generazione della nota, gli viene impedito di emettere un nuovo IRQ. Ciò si realizza con un'operazione di scrittura nella cella CNTA del timer (STA-CNTA). Si sarebbe naturalmente potuto scegliere pure un'altra fra le celle CNTA ... CNTD. La scrittura di una di queste celle disabilita il timer dal generare un Interrupt Request. Successivamente, il computer confronta il byte indirizzo più basso del Pointer NOTE con il contenuto della locazione ENDL. Riconosce così se la memoria della melodia è completamente riempita oppure no.
Se la memoria della melodia non è ancora piena, il computer trasferisce il contenuto delle celle KEY e LENGTH in tale memo-

ria. I numeri 77 presenti inizialmente nella memoria vengono così sovrascritti dai dati relativi al tasto testé lasciato libero. La CPU fornisce al Pointer NOTE un nuovo indirizzo (due volte INCZ-NOTEL): ad esso sarà deposto il valore del prossimo tasto premuto. Lo Junior-Computer attende ora la pressione di un nuovo tasto della tastiera.
In fig. 13 è riportata l'impostazione da tastiera del programma INPUT. Per tale programma si è scelta la zona di memoria 0200 ... 03FF. Dopo introdotto il programma INPUT con la tastiera, si fa partire l'Assembler col tasto ST: perciò, prima di aver fatto partire l'Editor, bisogna depositare nelle celle di memoria 1A7A e LA7B il vettore NMI corretto. Il vettore NMI indica l'indirizzo iniziale dell'Assembler. Dopo l'assemblaggio del programma INPUT, occorre ancora inserire nello Junior-Computer la Lookup Table LEN e la routine d'Interrupt: anche questo è riportato in fig. 13. Dopo aver caricato tutto, si può lanciare il programma INPUT; naturalmente bisogna che sia collegata la tastiera tipo piano. Come per il programma PLAY, possiamo ora suonare una melodia: lo Junior-Computer memorizza però questa volta la melodia nell'apposita memoria. Il programma INPUT può essere abbandonato in tre modi:
- premendo il tasto RST;
- per riempimento completo della memoria della melodia con dati: il computer rientra allora automaticamente nel Monitor;
- premendo il tasto ST. In tal caso il vettore NMI deve indicare l'indirizzo iniziale della successiva routine REPEAT (tale indirizzo è 0000).

Così risulta possibile inserire una melodia nel computer e riascoltare quante volte vogliamo premendo ogni volta il tasto ST.

Attenzione: non spegnere lo Junior-Computer dopo il caricamento della routine INPUT: bisogna ancora caricare la routine REPEAT! INPUT e REPEAT sono due programmi indipendenti fra loro, ma lavorano in stretto collegamento: il programma INPUT inserisce una melodia nel computer, ed il programma REPEAT ripete la melodia introdotta.

Figura 13. L'impostazione da tastiera della routine INPUT. Grazie all'Editor non risulta difficile introdurre nello Junior-Computer questo programma relativamente lungo. La routine Interrupt e la Lookup Table DEL vengono introdotte separatamente, perché sono poste in un'altra zona di memoria. Una volta assemblata la routine INPUT, non spegnere il computer, perché occorre ancora caricare la routine REPEAT! Questa costituisce l'inversa di INPUT: la melodia precedentemente introdotta viene riprodotta direttamente dal computer.

Tasti:		Display:	Significato
RST		XXXX XX	
AD	0 0 E 2	00E2 XX	
DA	0 0	00E2 00	BEGAD indica 0200
+	0 2	00E3 02	
+	F F	00E4 FF	ENDAD indica 03FF
+	0 3	00E5 03	
AD	1 A 7 A	1A7A XX	
DA	5 1	1A7A 51	Vettore NMI = 1F51
+	1 F	1A7B 1F	(Avvio Assembler con ST)
AD	1 C B 5	1CB5 20	Lancio dell'Editor
GO		77	Editor pronto a partire
INSERT	F F 1 0 0 0	FF 10 00	Label 10: INPUT
INPUT	7 8	78	SEI
INPUT	D 8	D8	CLD
INPUT	A 9 2 0	A9 20	LDA # 20
INPUT	8 D 7 E 1 A	8D 7E 1A	STA-IRQL
INPUT	A 9 1 A	A9 1A	LDA # 1A
INPUT	8 D 7 F 1 A	8D 7F 1A	STA-IRQH
INPUT	A 9 F 0	A9 F0	LDA # F0
INPUT	8 D 8 1 1 A	8D 81 1A	STA-PADD
INPUT	A 9 0 1	A9 01	LDA # 01
INPUT	8 D 8 3 1 A	8D 83 1A	STA-PBDD
INPUT	8 D 8 2 1 A	8D 82 1A	STA-PBD
INPUT	8 5 D D	85 DD	STAZ-NOTEH
INPUT	A 0 0 0	A0 00	LDY # 00
INPUT	8 C 8 0 1 A	8C 80 1A	STY-PAD
INPUT	8 4 D C	84 DC	STYZ-NOTEL
INPUT	A 0 D 8	A0 D8	LDY # D8
INPUT	8 4 D F	84 DF	STYZ-ENDL
INPUT	A 9 7 7	A9 77	LDA # 77
INPUT	F F 1 1 0 0	FF 11 00	Label 11: INA
INPUT	9 1 D C	91 DC	STA-(NOTEL), Y
INPUT	8 8	88	DEY
INPUT	C 0 F F	C0 FF	CPY # FF
INPUT	D 0 1 1	D0 11	BNE a INA (Label 11)
INPUT	F F 1 2 0 0	FF 12 00	Label 12: KEYSCN
INPUT	2 0 2 8 0 0	20 28 00	JSR-KEYIN (Label 28)
INPUT	F 0 1 2	F0 12	BEQ a KEYSCN (Label 12)
INPUT	2 0 2 9 0 0	20 29 00	JSR-DELAY (Label 29)
INPUT	2 0 2 8 0 0	20 28 00	JSR-KEYIN (Label 28)
INPUT	F 0 1 2	F0 12	BEQ a KEYSCN (Label 12)
INPUT	2 0 2 0 0 0	20 20 00	JSR-KEYVAL (Label 20)
INPUT	A 9 0 0	A9 00	LDA # 00
INPUT	8 5 D E	85 DE	STAZ-LENGTH
INPUT	A 9 F F	A9 FF	LDA # FF
INPUT	8 D F E 1 A	8D FE 1A	STA-CNTG
INPUT	5 8	58	CLI
INPUT	A 4 D A	A4 DA	LDYZ-KEY
INPUT	F F 1 3 0 0	FF 13 00	Label 13: TONE
INPUT	A 9 0 0	A9 00	LDA # 00
INPUT	8 D 8 2 1 A	8D 82 1A	STA-PBD
INPUT	B E 0 0 1 A	BE 00 1A	LDX-DEL, Y
INPUT	F F 1 4 0 0	FF 14 00	Label 14: TA
INPUT	2 0 3 1 0 0	20 31 00	JSR-EQUAL (Label 31)
INPUT	C A	CA	DEX
INPUT	D 0 1 4	D0 14	BNE a TA (Label 14)
INPUT	A 9 0 1	A9 01	LDA # 01
INPUT	8 D 8 2 1 A	8D 82 1A	STA-PBD
INPUT	B E 0 0 1 A	BE 00 1A	LDX-DEL, Y
INPUT	F F 1 5 0 0	FF 15 00	Label: TB
INPUT	2 0 2 8 0 0	20 28 00	JSR-KEYIN (Label 28)
INPUT	F 0 1 6	F0 16	BEQ a STORE (Label 16)
INPUT	C A	CA	DEX
INPUT	D 0 1 5	D0 15	BNE a TB (label 15)
INPUT	F 0 1 3	F0 13	BEQ a TONE (Label 13)
INPUT	F F 1 6 0 0	FF 16 00	Label 16: STORE
INPUT	8 D F 4 1 A	8D F4 1A	STA-CNTA

Tasti:		Display:	Significato
INPUT	A 5 D C	AC D5	LDAZ-NOTEL
INPUT	C 5 D F	C5 DF	CMPZ-ENDL
INPUT	F 0 1 7	F0 17	BEQ a ST (Label 17)
INPUT	9 8	98	TYA
INPUT	A 0 0 0	A0 00	LDY # 00
INPUT	9 1 D C	91 DC	STA-(NOTEL), Y
INPUT	C 8	C8	INY
INPUT	A 5 D E	A5 DE	LDA-LENGTH
INPUT	9 1 D C	91 DC	STA-(NOTEL), Y
INPUT	E 6 D C	E6 DC	INC-NOTEL
INPUT	E 6 D C	E6 DC	INC-NOTEL
INPUT	4 C 1 2 0 0	4C 12 00	JMP-KEYSCN (Label 12)
INPUT	F F 1 7 0 0	FF 17 00	Label 17 (ST)
INPUT	4 C 1 D 1 C	4C 1D 1C	JMP-RESET; ultima istruzione di INPUT
INPUT	F F 2 0 0 0	FF 20 00	Label 20: KEYVAL
INPUT	A 9 F 7	A9 F7	LDA # F7
INPUT	8 5 D 9	85 D9	STAZ-ROW (00D9)
INPUT	A 2 0 4	A2 04	LDX # 04
INPUT	F F 2 1 0 0	FF 21 00	Label 21: KEYA
INPUT	C A	CA	DEX
INPUT	3 0 2 0	30 20	BMI a KEYVAL (Label 20)
INPUT	0 6 D 9	06 D9	ASLZ-ROW (00 D9)
INPUT	A 5 D 9	A5 D9	LDAZ-ROW (00D9)
INPUT	8 D 8 0 1 A	8D 80 1A	STA-PAD
INPUT	A D 8 0 1 A	AD 80 1A	LDA-PAD
INPUT	2 9 0 F	29 0F	AND # 0F
INPUT	C 9 0 F	C9 0F	CMP # 0F
INPUT	F 0 2 1	F0 21	BEQ a KEYA (Label 21)
INPUT	8 6 D B	86 DB	STXZ-TEMPX (00DB)
INPUT	8 5 D A	85 DA	STAZ-KEY (00DA)
INPUT	A 2 0 0	A2 00	LDX # 00
INPUT	F F 2 2 0 0	FF 22 00	Label 22: KEYB
INPUT	4 6 D A	46 DA	LSRZ-KEY (00DA)
INPUT	9 0 2 3	90 23	BCC a ROWA (Label 23)
INPUT	E 8	E8	INX
INPUT	E 0 0 4	E0 04	CPX # 04
INPUT	D 0 2 2	D0 22	BNE a KEYB (Label 22)
INPUT	F 0 2 0	F0 20	BEQ a KEYVAL (Label 20)
INPUT	F F 2 3 0 0	FF 23 00	Label 23: ROWA
INPUT	A 5 D 8	A5 D8	LDAZ-TEMPX (00DB)
INPUT	C 9 0 3	C9 03	CMP # 03
INPUT	D 0 2 4	D0 24	BNE a ROWB (Label 24)
INPUT	8 A	8A	TXA
INPUT	4 C 2 7 0 0	4C 27 00	JMP-KEYC (Label 27)
INPUT	F F 2 4 0 0	FF 24 00	Label 24: ROWB
INPUT	C 9 0 2	C9 02	CMP # 02
INPUT	D 0 2 5	D0 25	BNE a ROWC (Label 25)
INPUT	8 A	8A	TXA
INPUT	1 8	18	CLC
INPUT	6 9 0 4	69 04	ADC # 04
INPUT	D 0 2 7	D0 27	BNE a KEYC (Label 27)
INPUT	F F 2 5 0 0	FF 25 00	Label 25: ROWC
INPUT	C 9 0 1	C9 01	CMP # 01
INPUT	D 0 2 6	D0 26	BNE a ROWD (Label 26)
INPUT	8 A	8A	TXA
INPUT	1 8	18	CLC
INPUT	6 9 0 8	69 08	ADC # 08
INPUT	D 0 2 7	D0 27	BNE a KEYC (Label 27)
INPUT	F F 2 6 0 0	FF 26 00	Label 26: ROWD
INPUT	C 9 0 0	C9 00	CMP # 00
INPUT	D 0 2 0	D0 20	BNE a KEYVAL (Label 20)
INPUT	8 A	8A	TXA
INPUT	1 8	18	CLC
INPUT	6 9 0 C	69 0C	ADC # 0C
INPUT	F F 2 7 0 0	FF 27 00	Label 27: KEYC
INPUT	8 5 D A	85 DA	STAZ-KEY (00DA)

Tasti:		Display:	Significato
INPUT	A 9 0 0	A9 00	LDA # 00
INPUT	8 D 8 0 1 A	8D D0 1A	STA-PAD
INPUT	6 0	60	RTS
INPUT	F F 2 8 0 0	FF 28 00	Label 28: KEYIN
INPUT	A D 8 0 1 A	AD 80 1A	LDA-PAD
INPUT	2 9 0 F	29 0F	AND # 0F
INPUT	4 9 0 F	49 0F	EOR # 0F
INPUT	6 0	60	RTS
INPUT	F F 2 9 0 0	FF 29 00	Label 29: DELAY
INPUT	A 0 F F	A0 FF	LDY # FF
INPUT	F F 3 0 0 0	FF 30 00	Label 30: DELA
INPUT	8 8	88	DEY
INPUT	D 0 3 0	D0 30	BNE a DELA (Label 30)
INPUT	6 0	60	RTS
INPUT	F F 3 1 0 0	FF 31 00	Label 31: EQUAL
INPUT	E A	EA	NOP
INPUT	E A	EA	NOP
INPUT	E A	EA	NOP
INPUT	E A	EA	NOP
INPUT	E A	EA	NOP
INPUT	6 0	60	RTS

Abbiamo così introdotto nello Junior-Computer il programma INPUT e tutte le sue subroutine; verifichiamo la presenza di eventuali errori di battitura:

Tasti:		Display:	Significato
SEARCH	F F 1 0	FF 10 00	INPUT inizia al Label 10
SKIP		78	
SKIP		D8	
SKIP		A9 20	
SKIP		8D 7E 1A	
.		.	
.		.	
.		.	
.		.	

Tutto è stato correttamente impostato? Se sì, passiamo all'assemblaggio di INPUT, e premiamo perciò il tasto ST.
Dobbiamo ancora inserire la routine d'Interrupt IRQUIN (fig. 12c) e la Lookup Table (fig. 6) nello Junior-Computer. Per l'introduzione della routine d'Interrupt impieghiamo ancora l'Editor: questa routine non richiede assemblaggio!

Tasti:		Display:	Significato		
AD	0 0 E 2	00E2 XX			
DA	2 0	00E2 20	BEGAD	1A20	128 bytes PIA-RAM
+	1 A	00E3 1A			
+	7 9	00E4 79	ENDAD	1A79	
+	1 A	00E5 1A			
AD	1 C B 5	1CB5 20	Lancio dell'Editor		
GO		77	Editor pronto a partire		

Tasti:		Display:	Significato
INSERT	4 8	48	PHA
INPUT	E 6 D E	E6 DE	INCZ-LENGTH
INPUT	A 9 F F	A9 FF	LDA # FF
INPUT	8 D F E 1 A	8D FE 1A	STA-CNTG
INPUT	6 8	68	PLA
INPUT	4 0	40	RTI
RST			Rientro nel Monitor
AD	1 A 0 0	1A00 XX	LOOKUP TABLE LEN
DA	8 E	1A00 8E	LEN * $1A00
+	8 6	1A01 86	
+	7 E	1A02 7E	
+	7 7	1A03 77	
+	7 0	1A04 70	
+	6 A	1A05 6A	
+	6 4	1A06 64	
+	5 E	1A07 5E	
+	5 9	1A08 59	
+	5 4	1A09 54	
+	4 E	1A0A 4E	
+	4 A	1A0B 4A	
+	4 7	1A0C 47	
+	4 3	1A0D 43	
+	3 E	1A0E 3E	
+	3 C	1A0F 3C	
AD	0 2 0 0	0200	
GO			Inizio del programma.

2) *L'interval-Timer opera alternativamente in Interrupt e in Polling-Mode*

Il programma REPEAT è capace di riprodurre la melodia previamente inserita con la routine INPUT. Il computer ripete cioé autonomamente la melodia inserita nella corrispondente memoria. Per la riproduzione della melodia memorizzata si debbono soddisfare i seguenti requisiti:

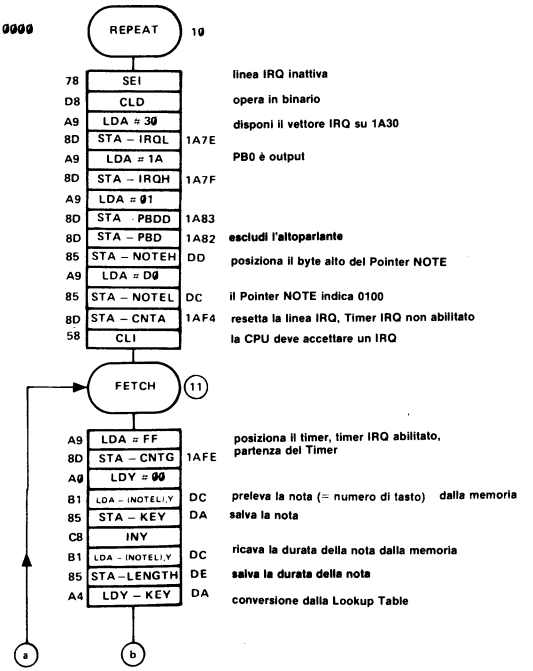
1) Il computer converte i valori dei tasti memorizzati in una nota. Viene ancora utilizzata a tale scopo la Lookup Table LEN.
2) Per ogni singola nota è pure memorizzata la durata di pressione del relativo tasto. Nella riproduzione della melodia inserita, la routine d'Interrupt converte la durata del tasto in una durata di nota (Interrupt-Mode).
3) Per evitare che nella riproduzione della melodia le note non si confondano fra loro, è necessaria una piccola pausa fra le singole note. Lo Junior-Computer genera autonomamente questa pausa tramite il timer. L'interval-Timer opera in tal caso in Polling-Mode.
4) Dopo aver risuonato l'intera melodia presente nella memoria, lo Junior-Computer si rifà vivo tramite il Monitor.

Questi requisiti sono tutti soddisfatti dalla routine REPEAT. Il relativo diagramma di flusso è illustrato nelle fig. 14a e 14b. Per descriverla ci potremo tener brevi, perché fra le due routine REPEAT e INPUT esistono molte analogie.

Al principio di REPEAT il processore viene disabilitato alla generazione di un Interrupt Request. La CPU deposita poi il vettore

IRQ della routine REPEAT. L'indirizzo relativo a 1A30 (il corrispondente indirizzo per INPUT era 1A20: così risulta possibile far funzionare INPUT e REPEAT indipendentemente l'uno dall'altro). Dato che a PB0 risulta collegato l'amplificatore con relativo altoparlante, occorre programmare tale linea di porta quale uscita. Occorre anche rimettere il Pointer NOTE: inizialmente esso indica l'indirizzo 0100, dove è depositato il primo valore di tasto della melodia da riprodurre. Dato che il Timer-Flag e la linea IRQ non sono ancora definiti, è necessaria un'operazione di scrittura in una delle celle del timer. L'istruzione STA-CNTA inibisce per prima cosa la generazione di un IRQ ed inoltre pone a 0 la linea IRQ. L'istruzione successiva, CLI, rende libera la linea IRQ; ossia: d'ora in poi la CPU dovrà accettare un Timer-Interrupt.
Adesso il microprocessore giunge al Label FETCH. Il timer vien fatto partire. L'Offset del timer è FF, ed il fattore di divisione 64 come per la routine INPUT. Un'operazione di scrittura in una delle celle (CNTG) del timer comunica al timer che esso è autorizzato a generare un Interrupt Request. Le istruzioni LDA FF, STA-CNTG fanno partire il timer, che marcia indipendentemente dalla CPU sino al Time Out. Il computer ha così tempo sufficiente (FF x 64 + 1 μs) per estrarre il valore del tasto e la durata di pressione dello stesso dalla memoria della melodia. Poi, il valore del tasto viene

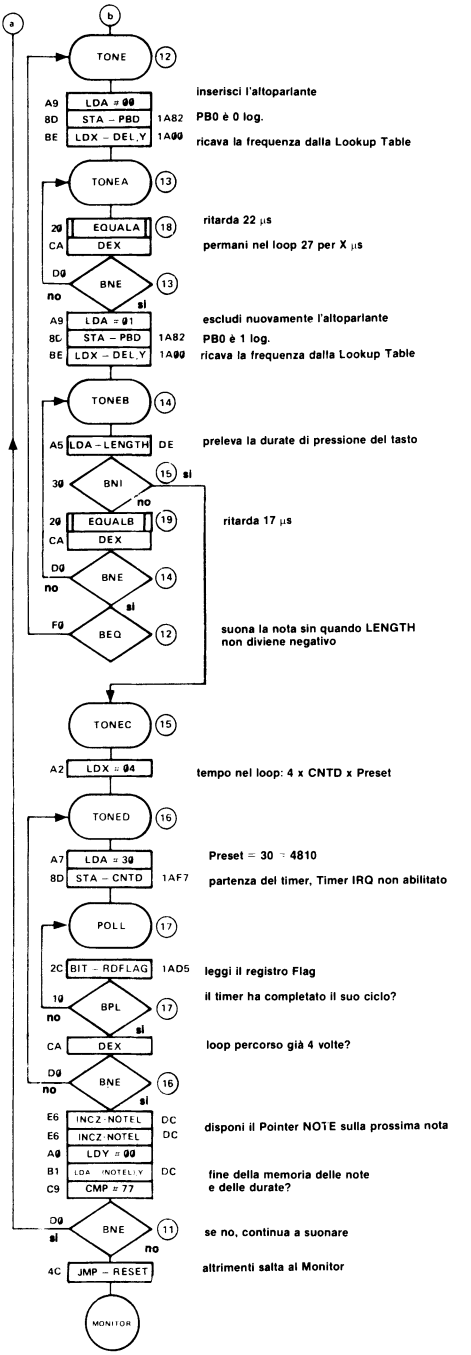
14a




```
KEY     *  $00DA
NOTEL   *  $00DC
NOTEH   *  $00DD
LENGTH  *  $00DE
DEL     *  $1A00
IRQL    *  $1A7E
IRQH    *  $1A7F
IRQRE   *  $1A30
RESET   *  $1C1D
RDFLAG  *  $1AD5
CNTA    *  $1AF4
CNTG    *  $1AFE
CNTD    *  $1AF7
PBD     *  $1A82
PBDD    *  $1A83
```

DELAYS

```
DEL:  1A00  8E
      1A01  86
      1A02  7E
      1A03  77
      1A04  70
      1A05  6A
      1A06  64
      1A07  5E
      1A08  59
      1A09  54
      1A0A  4E
      1A0B  4A
      1A0C  47
      1A0D  43
      1A0E  3E
      1A0F  3C
```


a
b
TONE
12
A9 LDA # 00
8D STA – PBD 1A82
BE LDX – DEL,Y 1A00
inserisci l'altoparlante
PB0 è 0 log.
ricava la frequenza dalla Lookup Table
TONEA
13
20 EQUALA
18
CA DEX
ritarda 22 µs
permani nel loop 27 per X µs
D0 BNE
13
no
si
A9 LDA # 01
8D STA – PBD 1A82
BE LDX – DEL,Y 1A00
escludi nuovamente l'altoparlante
PB0 è 1 log.
ricava la frequenza dalla Lookup Table
TONEB
14
A5 LDA – LENGTH DE
preleva la durate di pressione del tasto
15 si
30 BNI
no
20 EQUALB
19
CA DEX
ritarda 17 µs
D0 BNE
14
no
si
F0 BEQ
12
suona la nota sin quando LENGTH non diviene negativo
TONEC
15
A2 LDX # 04
tempo nel loop: 4 x CNTD x Preset
TONED
16
A7 LDA # 30
8D STA – CNTD 1AF7
Preset = 30 ÷ 4810
partenza del timer, Timer IRQ non abilitato
POLL
17
2C BIT – RDFLAG 1AD5
leggi il registro Flag
10 BPL
17
no
si
il timer ha completato il suo ciclo?
CA DEX
loop percorso già 4 volte?
D0 BNE
16
no
si
E6 INCZ-NOTEL DC
E6 INCZ-NOTEL DC
A0 LDY # 00
B1 LDA (NOTEL),Y DC
C9 CMP # 77
disponi il Pointer NOTE sulla prossima nota
fine della memoria delle note e delle durate?
D0 BNE
11
si
no
se no, continua a suonare
4C JMP – RESET
altrimenti salta al Monitor
MONITOR

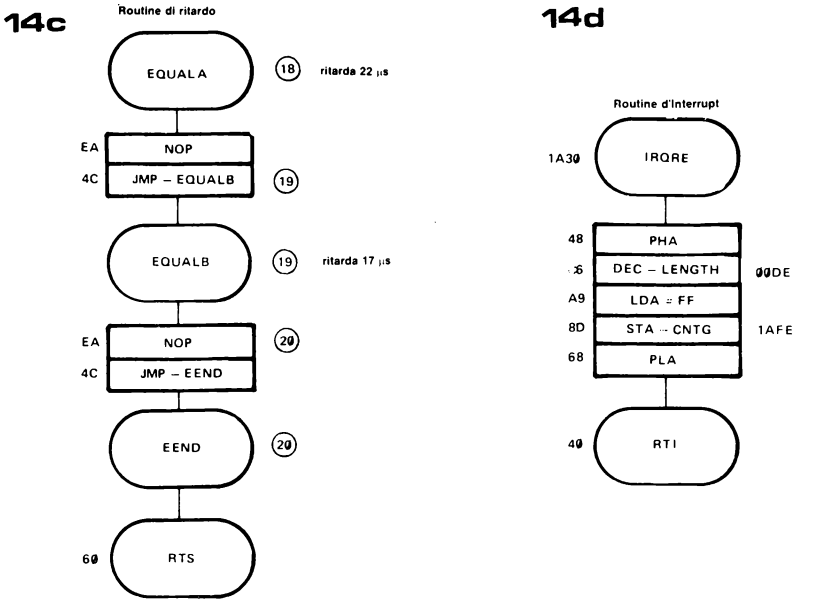


Figura 14. Col programma REPEAT è possibile far riprodurre dallo Junior Computer una melodia precedentemente introdotta. Affinché nel suonare la melodia non si sovrappongano le varie note, il computer genera automaticamente una piccola pausa fra nota e nota, utilizzando a tale scopo il timer degli intervalli. L'Interval-Timer durante il programma REPEAT opera alternativamente in Polling- ed in IRQ-Mode. REPEAT comincia all'indirizzo 0000.

convertito in una frequenza ricavandola dalla Lookup Table LEN (dopo il Label TONE). Al Label TONEA lo Junior-Computer inserisce l'altoparlante per la durata d'un semiperiodo ed a TONEB lo esclude per un altro semiperiodo. Si noti che, così come avveniva per INPUT, i tempi di permanenza nei loop citati durante la generazione della nota sono di 27 µs.

Lo Junior-Computer genera ora per diversi periodi una nota. Ad un certo momento si è concluso il ciclo del timer, che ha così emesso un Timer-IRQ. Il computer salta quindi alla Interrupt-Routine IRQRE, che inizia all'indirizzo 1A30 (il relativo vettore IRQ è stato posto ad 1A30 all'inizio di REPEAT). Nel programma d'Interrupt IRQRE (fig. 14d) avviene esattamente il contrario che in IRQIN:

IRQRE: il processore decrementa il contenuto della cella LENGTH

IRQIN: il processore incrementa il contenuto della cella LENGTH

Nel programma REPEAT quindi il processore emette una nota (a partire dal Label TONE) per tutto il tempo sinché il contenuto di

LENGTH, per successivi Timer-Interrupt, diventa negativo. Quando ciò avviene, il programma salta al Label TONEC. Prima di ricavare la prossima nota dalla memoria della melodia e riprodurla, il computer deve effettuare una piccola pausa, per evitare che le singole note si confondano l'un l'altra. Come detto prima, il timer nell'esecuzione di questa pausa deve operare in Polling-Mode. A tal fine in corrispondenza a TONED si inibisce il timer dal generare un Interrupt Request. Le istruzioni LDA # 30 e STA-CNTD commutano appunto il timer in Polling-Mode. L'operazione di scrittura STA-CNTD opera come segue:

- viene fatto partire l'Interval-Timer,
- l'offset del timer vale $30 = 48_{10}$,
- il Timer-Flag nel registro Flag viene posto a 0,
- il fattore di divisione viene fissato a 1024,
- il timer non può emettere un IRQ dopo il Time Out.

Dopo avviato il timer, il computer permane nel loop d'attesa POLL-BPL-POLL sin quando dopo un Time Out (48×1024) + 1 µs il Timer-Flag non viene rimesso ad 1. Questo processo si ripete in tutto quattro volte, dato che dopo ogni Time Out il registro X viene decrementato di uno. Ossia, il tempo totale di permanenza nel loop d'attesa viene moltiplicato per 4. Quando questo tempo di pausa fra due note è trascorso, il computer dispone il Pointer NOTE sulla prossima nota da suonare in memoria (due volte INCZ-NOTEL). Prima di iniziare a riprodurre la nota, si verifica se il Pointer Note non sia già pervenuto alla fine della melodia. La fine della melodia è indicata dal carattere 77 con valore di EOF (End of File) (vedi capitolo 5°). Quando infine sono state riprodotte tutte le note della melodia, lo Junior-Computer si rifà vivo tramite il Monitor.

L'impostazione da tastiera della routine REPEAT è mostrata in fig. 15. Grazie agli esaurienti commenti non dovrebbe risultare difficile procedere all'assemblaggio del programma in Pagina 0. In fondo, per i nostri lettori che hanno poca pratica con la musica, è pure riportato come introdurre una nota arietta. Ed ora, buon divertimento nell'inserimento e nel riascolto delle vostre melodie!

Con ciò concludiamo il capitolo 6. In esso si è mostrato come si possono programmare l'Interval-Timer, il rivelatore dei fronti ed il Peripheral Interface Adapter. La materia, un po' arida e molto teorica, è stata resa più leggera da un paio di programmi divertenti. Nel seguito faremo spesso riferimento a questo capitolo, quando verranno collegati allo Junior-Computer unità periferiche, come ad esempio una stampante o l'ELEKTERMINAL. Nel 3° volume sarà descritta un'interfaccia per registratore a cassette con relativa Software. Sarà ancora il CI 6532 a gestire due registratori ed a generare il segnale FSK. Lo Junior-Computer assumerà così vesti di vero e proprio Minicomputer. Basterà un ulteriore piccolo salto per passare al Personal Computer, in grado di comprendere

linguaggi superiori, per esempio il Basic. Ma prima di essere arrivati così lontano, spiegheremo la struttura del Monitor, del programma Editor e dell'Assembler, tutti presenti nella EPROM dello Junior-Computer. *Nota:* per tutti i programmi studiati in questo capitolo nell'appendice sono riportati i relativi Source-Listing con commento!

Figura 15. L'impostazione da tastiera del programma REPEAT.

Tasti:		**Display:**	**Significato**
AD	0 0 E 2	00E2 XX	
DA	0 0	00E2 00	BEGAD 0000
+	0 0	00E3 00	
+	E 0	00E4 E0	ENDAD 00E0
+	0 0	00E5 00	
AD	1 A 7 A	1A7A XX	V = 1F51
DA	5 1	1A7A 51	(Avvio dell'Assembler col tasto ST)
+	1 F	1A7B 1F	
AD	1 C B 5	1CB5 20	Avvio dell'Editor
GO		77	Editor pronto a partire
INSERT	F F 1 0 0 0	FF 10 00	Label 10: REPEAT
INPUT	7 8	78	SEI
INPUT	D 8	D8	CLD
INPUT	A 9 3 0	A9 30	LDA # 30
INPUT	8 D 7 E 1 A	8D 7E 1A	STA-IRQL
INPUT	A 9 1 A	A9 1A	LDA # 1A
INPUT	8 D 7 F 1 A	8D 7F 1A	STA-IRQH
INPUT	A 9 0 1	A9 01	LDA # 01
INPUT	8 D 8 3 1 A	8D 83 1A	STA-PBDD
INPUT	8 D 8 2 1 A	8D 82 1A	STA-PBD
INPUT	8 5 D D	85 DD	STAZ-NOTEH
INPUT	A 9 0 0	A9 00	LDA # 00
INPUT	8 5 D C	85 DC	STAZ-NOTEL
INPUT	8 D F 4 1 A	8D F4 1A	STA-CNTA
INPUT	5 8	58	CLI
INPUT	F F 1 1 0 0	FF 11 00	Label 11: FETCH
INPUT	A 9 F F	A9 FF	LDA # FF
INPUT	8 D F E 1 A	8D FE 1A	STA-CNTG
INPUT	A 0 0 0	A0 00	LDY # 00
INPUT	B 1 D C	B1 DC	LDA-(NOTEL), Y
INPUT	8 5 D A	85 DA	STAZ-KEY
INPUT	C 8	C8	INY
INPUT	B 1 D C	B1 DC	LDA-(NOTEL), Y
INPUT	8 5 D E	85 DE	STA-LENGTH
INPUT	A 4 D A	A4 DA	LDYZ-KEY
INPUT	F F 1 2 0 0	FF 12 00	Label 12: TONE
INPUT	A 9 0 0	A9 00	LDA # 00
INPUT	8 D 8 2 1 A	8D 82 1A	STA-PBD
INPUT	B E 0 0 1 A	BE 00 1A	LDX-DEL, Y
INPUT	F F 1 3 0 0	FF 13 00	Label 13: TONEA
INPUT	2 0 1 8 0 0	20 18 00	JSR-EQUALA (Label 18)
INPUT	C A	CA	DEX
INPUT	D 0 1 3	D0 13	BNE a TONEA (Label 13)
INPUT	A 9 0 1	A9 01	LDA # 01
INPUT	8 D 8 2 1 A	8D 82 1A	STA-PBD
INPUT	B E 0 0 1 A	BE 00 1A	LDX-DEL, Y
INPUT	F F 1 4 0 0	FF 14 00	Label 14: TONEB

Tasti:		Display:	Significato
INPUT	A 5 D E	A5 DE	LDAZ-LENGTH
INPUT	3 0 1 5	30 15	BMI a Label TONEC (Label 15)
INPUT	2 0 1 9 0 0	20 19 00	JSR-EQUALB (Label 19)
INPUT	C A	CA	DEX
INPUT	D 0 1 4	D0 14	BNE a TONEB (LABEL 14)
INPUT	F 0 1 2	F0 12	BEQ a TONE (Label 12)
INPUT	F F 1 5 0 0	FF 15 00	Label 15: TONEC
INPUT	A 2 0 4	A2 04	LDX # 04
INPUT	F F 1 6 0 0	FF 16 00	Label 16: TONED
INPUT	A 9 3 0	A9 30	LDA # 30
INPUT	8 D F 7 1 A	8D F7 1A	STA-CNTD
INPUT	F F 1 7 0 0	FF 17 00	Label 17: POLL
INPUT	2 C D 5 1 A	2C D5 1A	BIT-RDFLAG
INPUT	1 0 1 7	10 17	BPL a POLL (Label 17)
INPUT	C A	CA	DEX
INPUT	D 0 1 6	D0 16	BNE a TONED (Label 16)
INPUT	E 6 D C	E6 DC	INCZ-NOTEL
INPUT	E 6 D C	E6 DC	INCZ-NOTEL
INPUT	A 0 0 0	A0 00	LDY # 00
INPUT	B 1 D C	B1 DC	LDA-(NOTEL), Y
INPUT	C 9 7 7	C9 77	CMP # 77
INPUT	D 0 1 1	D0 11	BNE a FETCH (Label 11)
INPUT	4 C 1 D 1 C	4C 1D 1C	JMP-RESET (Monitor; RESET = 1C1D)
INPUT	F F 1 8 0 0	FF 18 00	Label 18: Subroutine EQUALA
INPUT	E A	EA	NOP
INPUT	4 C 1 9 0 0	4C 19 00	JMP-EQUALB (Label 19)
INPUT	F F 1 9 0 0	FF 19 00	Label 19: EQUALB
INPUT	E A	EA	NOP
INPUT	4 C 2 0 0 0	4C 20 00	JMP-EEND (Label 20)
INPUT	F F 2 0 0 0	FF 20 00	Label 20: EEND
INPUT	6 0	60	RTS

Verifica del programma REPEAT:

Tasti:		Display:
SEARCH	F F 1 0	FF 10
SKIP		78
SKIP		D8
SKIP		A9 30
SKIP		8D 7E 1A
.		.
.		.
.		.

Tutto correttamente impostato? Se sì, si può passare all'assemblaggio di REPEAT.
Quindi introduciamo la Interrupt-Routine IRQRE. Non occorre assemblare questa parte di programma!

Tasti:		Display:	Significato	
AD	0 0 E 2	00E2 XX		
DA	3 0	00E2 30		
+	1 A	00E3 1A	BEGAD	1A30
+	7 9	00E4 79		
+	1 A	00E5 1A	ENDAD	1A79

AD	1 C B 5	1CB5 20	Lancio dell'Editor
GO		77	Editor pronto a partire
INSERT	4 8	48	PHA
INPUT	C 6 D E	C6 DE	DEC-LENGTH
INPUT	A 9 F F	A9 FF	LDA # FF
INPUT	8 D F E 1 A	8D FE 1A	STA-CNTG
INPUT	6 8	68	PLA
INPUT	4 0	40	RTI
RST			Rientro al monitor

Tasti:		**Display:**	**Significato**
			Non occorre caricare nuovamente la Lookup Table LEN, già presente dalla routine INPUT. Se tramite INPUT è stata introdotta in precedenza una melodia, lo Junior-C. la riproduce lanciando la routine REPEAT:
AD	0 2 0 0	0200 A9	Indirizzo di partenza di INPUT
GO		spento	Tramite la tastiera suoniamo l'arietta "Alle kleinen Entchen...". Ogni tasto corrisponde ad un numero, per cui non sarà difficile anche ai lettori non musicali suonare l'arietta:

Premere i tasti da piano nell'ordine	**Display:**	**Significato**
0 2 4 5 7 7 9 9 9 7 5 5 5 5 4 4 2 2 2	9spento	La melodia è stata inserita nello Junior. Il computer ha memorizzato in Pagina 1 tutti i valori e le durate dei tasti, occupando due celle per ogni nota
2 0		

Tasti:		**Display:**	**Significato**
AD	0 0 0 0	0000 78	Indirizzo di partenza di REPEAT mentre il
GO		spento	computer suona l'arietta. Al termine della melodia il display visualizza:
		0000 78	Il programma può venire ancora replicato premendo il tasto GO.

7

Il Programma Monitor

Software elementare per la tastiera ed il Display

Se l'utente vuole utilizzare un computer, questo deve essere in grado di accogliere i dati introdotti e di comunicare i dati elaborati. Tra il computer e l'utente si stabilisce quindi un accoppiamento reattivo. Perché fra uomo e macchina si possa stabilire un dialogo è necessario che nella memoria del computer sia presente un programma. Il programma che consente di realizzare tale dialogo si chiama programma Monitor o brevemente Monitor. Il Monitor dello Junior-Computer svolge i seguenti compiti:

- **visualizzazione degli indirizzi e dei dati su display a 6 cifre;**
- **sondaggio dei tasti dati e tasti comando;**
- **gestione del display in multiplex;**
- **esecuzione dei comandi attivati dall'utente premendo uno dei tasti AD, DA, +, PC e GO.**

Diverse subroutine supportano il lavoro del Monitor dello Junior-Computer. Il grande vantaggio che se ne ricava è che il programmatore può inserire tali subroutine in propri programmi senza bisogno di svilupparli separatamente. Inoltre, i vettori Interrupt IRQ e NMI sono deposti in celle di una RAM: perciò è sempre possibile modificare i valori di tali vettori nel corso di un programma. Ciò è importante, perché consente di richiamare con uno stesso vettore distinti programmi Interrupt. Questo capitolo, che descrive il Monitor, ci chiarirà tutte queste relazioni.

Quel che già sappiamo del Monitor

Il programma Monitor è depositato in una parte della EPROM dell'Junior-Computer. In gergo si usa dire che è un programma "residente". Ciò vuol dire che esso è sempre disponibile, anche dopo disinserita la tensione di rete. Quando il computer si trova

nel Monitor, ha bisogno di disporre anche di alcune locazioni di memoria in Pagina Zero. In queste celle vengono deposti dati che il computer deve richiamare più volte nel corso del programma Monitor, ad esempio per svolgere determinati compiti. Riportiamoci al 1° volume! In esso molti programmi erano conclusi con un'istruzione BRK. In questi casi il Monitor conserva in Pagina Zero il contenuto di tutti i registri della CPU. Il programmatore è così in grado di controllare il contenuto di tali registri, prima che lo Junior-Computer risalti indietro nel programma Monitor per gestire il display e la tastiera. Il programma Monitor dello Junior-Computer è in realtà un loop di attesa. Quando si trova in tale loop, il computer sonda periodicamente la tastiera e gestisce il display a 6 cifre in sistema multiplex. A tal fine trasferisce il contenuto dei tre buffer di display POINTH, POINTL ed INH ai display. Se l'utente preme uno dei cinque tasti comandi AD, DA, +, PC o GO il computer provvede alle operazioni relative a tali comandi. Se ad esempio viene premuto il tasto AD, il computer interpreta i dati successivi introdotti con i tasti 0 ... F quali indirizzi. Se era premuto il tasto Dati, i dati introdotti vengono deposti agli indirizzi indicati. Anche gli altri tasti comandi ci sono noti dal 1° volume.
Esistono diversi modi per entrare o per uscire dal Monitor dello Junior-Computer:

1) *Salto al Monitor:*

* Pressione del tasto RST: lo Junior-Computer esegue un salto nel Monitor, e definisce le due Porte del Peripheral Interface Adapter. Quindi il computer permane in un loop d'attesa, durante il quale gestisce il display e la tastiera. Lo Junior-Computer risulta allora accessibile tramite i tasti dati 0...F nonché i tasti comandi AD, DA, +, PC e GO.
* Generazione di un NMI: mediante il vettore NMI può analogamente verificarsi un salto al programma Monitor. Se il vettore NMI indica l'indirizzo 1C00, il computer provvede a salvare il contenuto di tutti i registri interni della CPU. Nuovamente lo Junior-Computer si mantiene in un loop d'attesa; in esso gestisce il display e la tastiera, ed è accessibile tramite i tasti dati 0 .... F nonché i tasti comando AD, DA, +, PC e GO. Il PIA non viene riprogrammato, se il Vettore NMI indica 1C00.
* Generazione di un IRQ: anche in questo caso viene effettuato un salto al Monitor, tramite il vettore IRQ. Ciò è tuttavia permesso solo se il Flag I del registro P della CPU è posto a 0 (CLI). Se il vettore IRQ indica 1C00, vale quanto si è detto prima per il vettore NMI.
* Salto al Monitor tramite il comando BRK: nel 1° volume diversi programmi sono stati conclusi con l'istruzione BRK. Come sappiamo, l'istruzione BRK opera in collegamento col vettore IRQ. Se con l'istruzione BRK si vuole conseguire un salto al

Monitor, occorre che il vettore IRQ indichi l'indirizzo 1C00. Lo Junior-Computer provvede allora a salvare il contenuto di tutti i registri interni della CPU, e perviene in un loop di attesa, come per RST, NMI ed IRQ.

* Salto al monitor tramite un'istruzione JSR: il programmatore può inserire alcune subroutine del Monitor nel proprio programma. Il rientro della subroutine del Monitor al programma principale avviene tramite l'istruzione RTS al termine della subroutine.
* Salto al Monitor tramite un'istruzione JMP: il programmatore può rientrare al Monitor al termine del proprio programma. Quando lo Junior-Computer ha terminato di elaborare un programma esterno, il display si illumina, e l'utente ha così modo di accorgersi che il programma è terminato. Se si esegue un salto al Monitor con un'istruzione JMP, il salto dovrebbe sempre avvenire all'indirizzo 1C1D (JMP-1C1D ovvero 4C 1D 1C). Da questo indirizzo parte una sequenza di reset ed il PIA viene riprogrammato per il sondaggio della tastiera e la gestione del display. Anche in questo caso lo Junior-Computer si mantiene in un ciclo d'attesa come nei casi precedenti (RST, NMI, IRQ e BRK).

2) *Uscite con salto dal Monitor:*

* Pressione del tasto GO: il modo più comune per abbandonare il programma Monitor è premere il tasto GO. Lasciato il Monitor, la CPU inizia l'elaborazione partendo dall'indirizzo visualizzato sul display. Il display resta poi spento sino a quando non viene effettuato un nuovo salto al Monitor.
* Generazione di un NMI: la CPU lascia il programma Monitor affidandosi al vettore NMI. Se al termine della routine d'Interrupt è posta un'istruzione RTI, si rientra nel Monitor. Se invece dopo la emissione del NMI non viene eseguito il rientro al Monitor, occorre che prima di lasciare il Monitor venga ridefinito l'I/O. Così si evita l'accensione a caso del display! Se si desidera che il display rimanga completamente escluso, si deve far seguire, dopo usciti dal Monitor tramite un NMI o IRQ, ..., la sequenza di programma.

```
(PHA)      salva il contenuto dell'Accu
LDA # 06
STA-PBD    tutti i catodi comuni dei display sono scollegati
LDA # 00
STA-PAD    nessun segmento di display assorbe corrente
(PLA)      ristabilisci il contenuto originario dell'Accu
```

* Si noti che il contenuto dell'Accu va salvato solo nel caso che esso sia richiesto dopo usciti dal Monitor. Per questo le istruzioni PHA e PLA figurano fra parentesi.
* Generazione di un IRQ: la CPU può analogamente uscire dal Monitor tramite il vettore IRQ. È necessario tuttavia che il Flag I

nel registro P della CPU sia stato posto a 0, affinché il processore possa eseguire l'IRQ. In ogni caso per il resto vale quanto detto prima per l'NMI.

Ricordarsi: i vettori NMI ed IRQ sono posti in Pagina 1A della RAM. A questi due vettori sono associati i seguenti indirizzi:

NMIL * $1A7A
NMIH * $1A7B
IRQL * $1A7E
IRQH * $1A7F

Il contenuto di queste locazioni di memoria stabilisce dove deve saltare la CPU dopo un NMI, un IRQ od un BRK. Due locazioni di memoria servono a definire il vettore Interrupt, ossia:

NMIH, NMIL = vettore di Non-Maskable Interrupt
IRQH, IRQL = vettore di Interrupt ReQuest.

Questi due vettori possono essere fissati a mano dal programmatore, od anche modificati dal programma.

Quanto sopra costituisce un piccolo riassunto del contenuto del capitolo 3 del 1° volume, ritenuto necessario per comprendere il complesso diagramma di flusso del programma Monitor.

Il diagramma di flusso generale del Monitor

La figura 1 mostra il diagramma di flusso globale del Monitor. Sono riportate le più comuni vie di ingresso ed uscita. Il programma Monitor dispone di tre ingressi e due uscite. **Non** è compreso l'abbandono del Monitor via NMI od IRQ!

Gli ingressi e le uscite del Monitor

L'Input ① conduce al Label RESET. La CPU perviene al Monitor attraverso questo ingresso quando si preme il tasto RST. Al Label RESET è attribuito l'indirizzo 1C1D. A partire da tale Label il processore programma il PIA ed inizializza lo Stack-Pointer. Inoltre il registro Processor Status viene posto in uno stato definito. Diremo più oltre nei particolari come ciò avviene. Per ora basti sapere che tra i Label RESET e START lo Junior-Computer viene predisposto a gestire il display ed a sondare la tastiera. Il computer inoltre opera di qui in avanti in sistema binario (CLD); il Flag I nel registro P della CPU è posto ad 1. Lo Junior Computer non può quindi rispondere ad un IRQ.

In sommario: vi sono molti modi diversi per giungere al Monitor dello Junior-Computer attraverso l'Input ①:

- pressione del tasto RST
- istruzione di salto al Label RESET: JMP-1C1D
- tramite un NMI: il relativo vettore NMI punta all'indirizzo 1C1D (NMIH, NMIL = 1C1D). Alle locazioni 1A7A e 1A7B debbono trovarsi i seguenti byte:
  NMIL = $1A7A contiene 1D
  NMIH = $1A7B contiene 1C
  } Label RESET

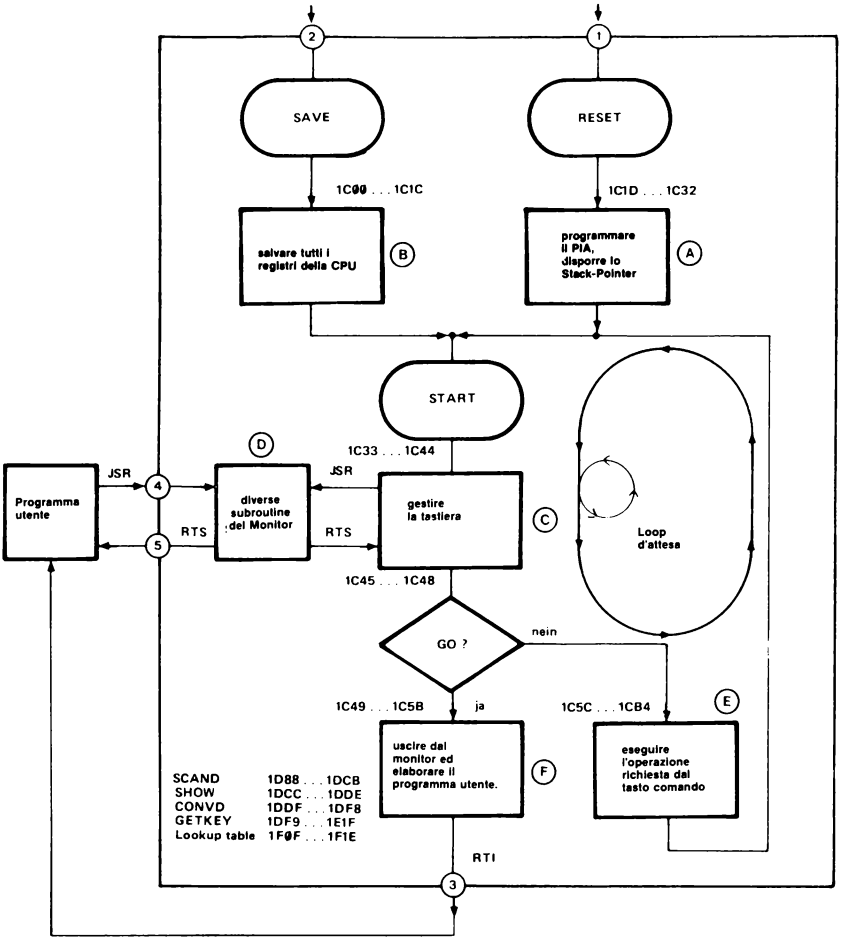


Figura 1. Il diagramma di flusso globale del Monitor. Il programma Monitor ha tre ingressi e due uscite. Vari indirizzi descritti in questo diagramma di flusso possono essere anche consultati nel Source Listing in appendice al volume.

- tramite un IRQ (se il Flag I vale 0): il vettore IRQ punta all'indirizzo 1C1D (IRQH, IRQL = 1C1D). Alle locazioni 1A7E e 1A7F debbono trovarsi i seguenti byte:
  IRQL = $1A7E contiene 1D
  IRQH = $1A7F contiene 1C } Label RESET
- con un comando BRK: anche questo opera tramite il vettore IRQ. Se si vuole saltare all'Input ① del Monitor con questo comando, (fig. 1), il vettore IRQ deve indicare l'indirizzo 1C1D.

L'Input ② porta al Label SAVE. A questo Label è assegnato l'indirizzo 1C00. Tra i due Label SAVE e START tutti i registri interni della

CPU vengono deposti in Pagina Zero ai seguenti indirizzi:
PCL: all'indirizzo 00EF
PCH: all'indirizzo 00F0
registro P: all'indirizzo 00F1
User Stack Pointer: all'indirizzo 00F2
Accumulatore: all'indirizzo 00F3
registro Y: all'indirizzo 00F4
registro X: all'indirizzo 00F5

Attenzione! L'Input ② dovrebbe essere riservato al salto nel Monitor tramite vettori Interrupt. Solo in tal caso infatti vengono conservati i registri PCH, PCL e Status della CPU, ed inoltre lo Stack-Pointer risulta correttamente posizionato dopo l'uscita dal Monitor con il tasto GO. Chiariamo più sotto questi particolari. Si ricordi: *se il programmatore sceglie un altro modo operativo per il PIA, al rientro nel Monitor tramite l'Input ② l'I/O per la gestione del display e l'interrogazione della tastiera non risultano definiti!*

In sommario: vi sono molti diversi modi di accedere al Monitor dello Junior-Computer attraverso l'Input ②:

- pressione del tasto ST; contemporaneamente viene generato un NMI. Il relativo vettore NMI deve indicare l'indirizzo 1C00 (NMIH, NMIL = 1C00). Nelle locazioni 1A7A ed 1A7B si debbono trovare i seguenti byte:
  NMIL = 1A7A contiene 00 e
  NMIH = 1A7B contiene 1C.
- tramite un NMI, generato in Step-by-Step-Mode: già nel 1° volume si è visto come seguire programmi in Step-by-step-Mode; ossia come percorrerli passo passo. Più precisamente: dopo la pressione del tasto GO, la CPU esegue l'istruzione visualizzata sul display. Quindi, il display visualizza l'istruzione immediatamente successiva e provvede poi a conservare in Pagina Zero tutti i registri della CPU. Allo scopo di far lavorare lo Junior-Computer in Step-by-step-Mode, occorre preliminarmente disporre il vettore NMI sull'indirizzo 1C00 (Label SAVE). Chiariremo più oltre questo modo di elaborare passo passo un programma;
- tramite un NMI, generato da una qualche unità periferica, ad es. una stampante, ecc. Anche in questo caso il vettore NMI può indicare l'indirizzo 1C00;
- tramite un IRQ (se il Flag I vale 0): il vettore IRQ indica l'indirizzo 1C00 (IRQH, IRQL = 1C00). Nelle locazioni 1A7E ed 1A7F debbono trovarsi i seguenti byte:
  IRQL = $1A7E contiene 00
  IRQH = $1A7F contiene 1C
- non si dovrebbe entrare più in successione nel Monitor attraverso l'Input ② mediante il comando BRK. Nel 1° volume, per semplicità, si sono effettuati salti nel Monitor tramite BRK e

l'Input ②. Ciò ha comportato questi errori:
Overflow dello Stack.
Sullo Stack viene posto un valore errato del Program Counter (PC + 2).

* L'Input ④ conduce a varie subroutine del Monitor. L'utente può inserire queste subroutine in propri programmi e richiamarle con l'istruzione JSR. L'istruzione RTS al termine di tali subroutine provvede poi al rientro nel programma utente (Output ⑤).

In tal modo sono stati descritti tutti i compiti che la CPU svolge fra i Label SAVE e START. A partire dal Label START lo Junior-Computer si mantiene in un loop di attesa, che può essere abbandonato premendo il tasto GO. In questo loop il computer svolge le seguenti funzioni:

1. Gestione del display e sondaggio della tastiera, col richiamo di varie subroutine del Monitor.
2. Se viene premuto un tasto, il computer verifica se si tratta del tasto GO: se non è così, rimane nel Monitor. Se è stato premuto un tasto comandi, provvede ad eseguire il relativo comando. Se è stato premuto un tasto dati, questi vengono assegnati al display indirizzi o dati. Quindi il processo si rinnova al Label START.
3. Se nel corso del loop viene premuto ad un certo momento il tasto GO, il computer esce dal programma Monitor, e tramite l'istruzione RTI salta al programma utente. Torneremo in dettaglio su questo punto illustrando il diagramma di flusso particolareggiato del Monitor. Normalmente, dal Monitor si esce attraverso l'Output ③.

Breve intermezzo: cos'é lo Step-by-Step?

Parlando del diagramma di flusso generale del Monitor, non dobbiamo trascurare lo Step-by-Step-Mode. In questa modalità operativa, Hardware e Software dello Junior-Computer lavorano in stretta connessione fra loro. Vediamo come, considerando ancora una volta il circuito elettrico dello Junior-Computer (Volume 1°, capitolo 1, Fig. 4). Il piedino 7 della CPU (IC1, uscita SYNC) è collegato alla porta NAND N5. L'altro ingresso della porta riceve il segnale $\overline{CS}$ via K7. L'output del NAND è collegato alla linea NMI del processore. Quando l'interruttore S24 è chiuso, il computer lavora in Step-by-step Mode. Questo modo operativo può essere descritto come segue:

a. Il display visualizza un indirizzo cui corrisponde un'istruzione. L'interruttore S24 è chiuso.
b. L'utente desidera eseguire l'istruzione macchina al momento indicata: a tal fine preme il tasto GO. Il processore abbandona il Monitor attraverso l'Output ③ ed inizia l'elaborazione dall'indirizzo indicato, dal quale ha prelevato la relativa istruzione macchina.

c. Mentre il processore preleva dalla memoria l'istruzione (codice OP), l'uscita SYNC (piedino 7) della CPU passa allo stato logico 1. In questo frattempo l'EPROM IC2 non risulta indirizzata e quindi la linea K7 del segnale $\overline{CS}$ è allo stato 1. Pertanto l'uscita SYNC della CPU produce un impulso a fronte negativo all'output della porta NAND N5.
d. Questo impulso negativo genera un NMI. Il processo esegue l'istruzione visualizzata sul display e poi passa a servire il NMI. Tramite il vettore NMI la CPU è indirizzata al Label SAVE d'indirizzo 1C00 (Input ②).
e. Il processore si ritrova ora nel programma Monitor. La EPROM (IC2) risulta indirizzata e la linea K7 si porta a 0. Non può essere generato alcun NMI, dato che l'uscita del NAND N5 rimane sempre 1. A partire dal Label SAVE la CPU provvede a conservare i contenuti di tutti i registri interni in Pagina Zero, ed il Program Counter indica il codice OP della successiva istruzione. Dal Label SAVE il nuovo contenuto del Program Counter viene portato nei buffer di display POINTH e POINTL. A partire dal Label START il computer visualizza sul display la successiva istruzione con il relativo indirizzo. Una nuova pressione del tasto GO rinnova tutto il processo.

Svolgimento dettagliato degli Interrupt e del comando BRK

Finora risulta soltanto chiaro dove debbono venire posti i vettori d'Interrupt per NMI e IRQ, affinché la CPU possa svolgere un Interrupt. Non è stato però ancora spiegato come la CPU preleva i vettori NMI ed IRQ dalle locazioni di memoria 1A7A, 1A7B e 1A7E, 1A7F. Queste operazioni ci vengono chiarite in fig. 2 in modo esauriente. Dalla parte sinistra troviamo una zona di memoria, che consta di varie parti: la parte superiore comprende celle RAM, nelle quali è posto il programma utente. Si tratta come già sappiamo delle Pagine 0, 1, 2, 3 e della RAM da 1/8 K in Pagina 1A. La parte inferiore è occupata da una parte del programma Monitor nell'EPROM (identica a quella di fig. 1).
Per illustare il concetto d'Interrupt, si sono rappresentate solo le locazioni di memoria più importanti. Nella parte destra di fig. 2 vediamo le varie possibilità di ingresso od uscita dal Monitor. Per Monitor intendiamo qui la parte di EPROM già descritta e con cui si è lavorato nel 1° volume; ed inoltre le 6 locazioni di memoria per i vettori NMI, Reset ed IRQ con indirizzi 1FFA ... 1FFFF. I punti che seguono danno tutte le più importanti informazioni su questi vettori:

1. Pressione del tasto RST
La CPU preleva dalle celle 1FFC e 1FFD della EPROM il vettore di Reset. Tramite ① poi salta al Label RESET di fig. 1. A questo Label

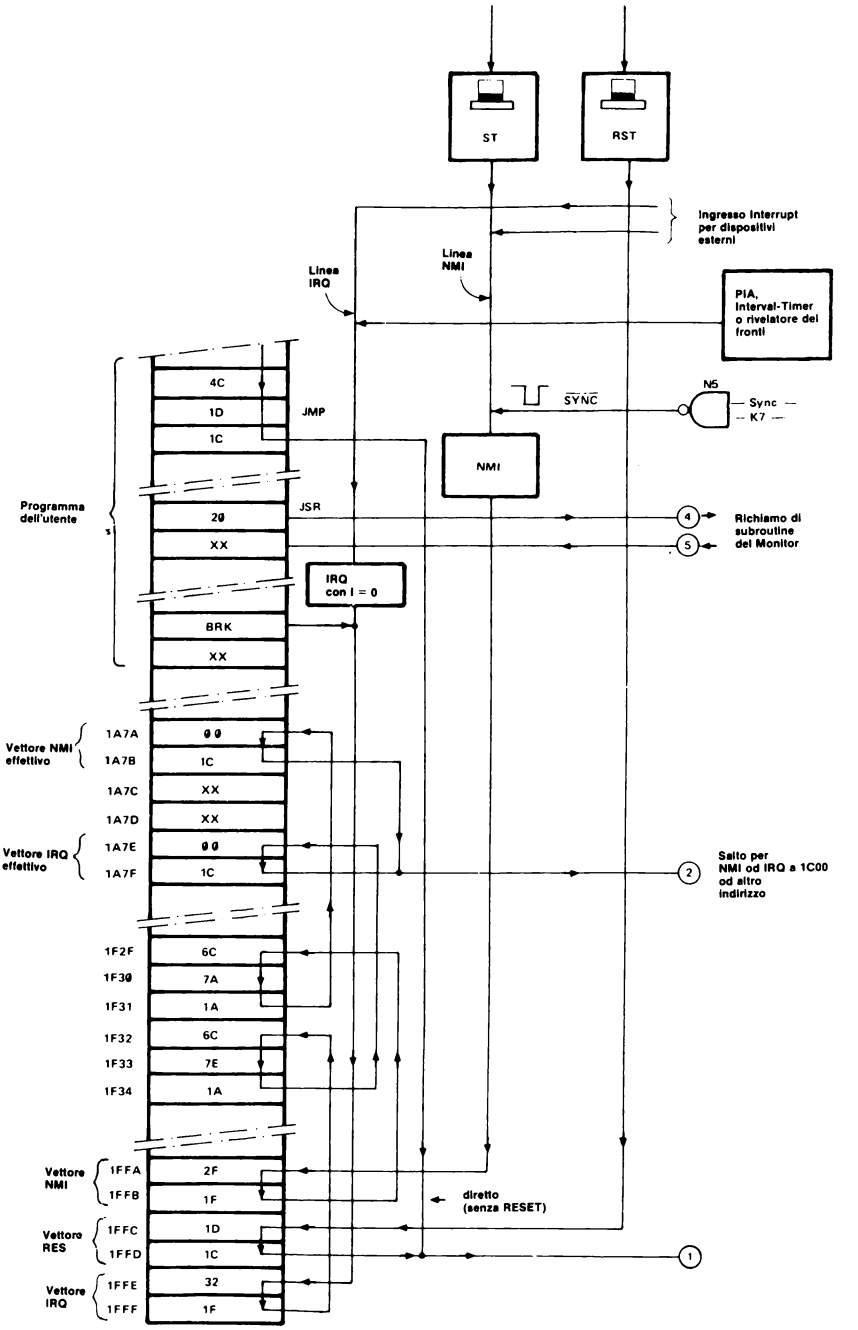


Figura 2. Come si entra e si esce dal Monitor? Questa figura ci illustra tutte le varie possibilità. Poiché i vettori di Interrupt segnalano salti indiretti a programma, essi possono venire modificati dal programmatore manualmente o via programma.

si può pervenire anche direttamente dal programma utente tramite un'istruzione JMP.

2. Generazione di un NMI

Un NMI può essere generato nello Junior-Computer in modi diversi:

* Premendo il tasto ST.
* Da un'unità esterna collegata, tramite la linea NMI sul bus di controllo. Questa linea è pure collegata al connettore di espansione.
* Da un fronte d'impulso negativo all'uscita di N5, quando lo Junior-Computer opera in Step-by-step Mode.

Non appena un fronte negativo viene applicato al piedino NMI della CPU, il computer svolge un NMI. La fig. 2 illustra i singoli passi di una tale sequenza:

- Dopo l'arrivo del fronte d'impulso negativo sul piedino NMI, la CPU esegue l'istruzione in corso, e poi passa ad occuparsi dell'Interrupt. Prima di prelevare il vettore NMI, la CPU provvede a salvare nell'ordine sullo Stack i registri PCH, PCL e P. Il Flag I è posto ad 1.
- Il processore preleva il vettore NMI dalle locazioni 1FFA e 1FFB al termine della EPROM. A tale indirizzo nel programma Monitor sta un'istruzione di salto indiretto: JMP - (IND) = JMP-(1A7A) = 6C 7A 1A. Come ci è noto dal 1° volume, un salto indiretto ci porta a due celle di memoria nelle quali è posto l'effettivo indirizzo di salto. Queste due celle nello Junior-Computer sono situate in Pagina 1A ed hanno indirizzi 1A7A e 1A7B. Il contenuto di queste due celle di memoria è quindi l'indirizzo, al quale deve rivolgersi il computer per iniziare l'elaborazione dopo la generazione di un NMI.

In fig. 2 l'effettivo vettore NMI indica l'indirizzo 1C00. Quando lo Junior-Computer giunge a 2, passa al Label SAVE di Fig. 1. Dato che le locazioni 1A7A e 1A7B sono celle d'una RAM, il vettore d'Interrupt può essere posto ad un indirizzo qualsiasi.

3. Generazione di un IRQ

Affinché la CPU possa svolgere un IRQ, il Flag I nel registro Processor Status deve essere a 0 (CLI). Nello Junior-Computer un IRQ può venir generato in diversi modi:

* Da un'unità esterna collegata, tramite la linea IRQ sul bus di controllo. Questa linea è pure collegata al connettore di espansione.
* Da un Timer-IRQ (vedi capitolo 6)
* Da un PA7-IRQ (vedi capitolo 6)

Se la linea IRQ è mantenuta per alcuni microsecondi allo stato logico 0, il computer passa a svolgere una sequenza IRQ. Il programmatore deve badare che, prima del rientro dalla routine d'Interrupt, la linea IRQ venga rimessa ad 1, affinché il processore non replichi ancora una volta il medesimo I.R. (vedi capitolo 6). La fig.

2 illustra i singoli passi di programma nello svolgimento di un IRQ:

- Dopo che il piedino IRQ della CPU è passato allo stato logico 0, il processore esegue l'istruzione in corso, e poi passa ad occuparsi dell'IRQ. Prima di prelevare il vettore IRQ, la CPU provvede a salvare sullo Stack, nell'ordine, i registri PCH, PCL e P. Il Flag I è posto ad 1.
- Il processore preleva il vettore IRQ dalle locazioni 1FFE e 1FFF. Nello Junior-Computer questo vettore indica l'indirizzo 1F32. A questo indirizzo nel programma Monitor è situata un'istruzione di salto indiretto: JMP-(IND) = JMP-(1A7E) = 6C 7E 1A. Come ci è noto dal 1° volume, un salto indiretto ci porta a due celle di memoria nelle quali è posto l'effettivo indirizzo di salto. Queste due celle relative all'effettivo vettore IRQ nello Junior-Computer sono situate in Pagina 1A ed hanno gli indirizzi 1A7E ed 1A7F. Il contenuto di queste due celle di memoria è dunque l'indirizzo a cui deve rivolgersi il computer per iniziare a lavorare dopo la generazione di un IRQ.

In fig. 2 il vettore IRQ indica l'indirizzo 1C00, come prima. Quando lo Junior-Computer giunge a 2, passa al Label SAVE in fig 1. Dato che le locazioni d'indirizzo 1A7E e 1A7F sono celle d'una RAM, il vettore IRQ può essere posto ad un indirizzo qualsiasi.

4. Il comando BRK

Sebbene in diversi programmi del 1° volume si sia fatto uso del comando BRK, esso non è ancora stato descritto in dettaglio. È quanto perciò facciamo ora:

- Il computer sta elaborando un programma introdotto dall'utente. Al termine di questo programma la CPU incontra un comando BRK (codice operativo 00). Il comando BRK opera in connessione al vettore IRQ; ossia il programma effettua un salto come in una sequenza IRQ (vedi punto 3). Prima di prelevare il vettore IRQ, il processore provvede a salvare sullo Stack, nell'ordine, i registri PCH, PCL e P. Il Flag B nel registro P è posto ad 1. Se ad esempio il comando BRK è posto all'indirizzo 022A, sullo Stack viene salvato il valore del Program Counter come 022C - valore non corretto.
- Il processore preleva il vettore IRQ dalle celle 1FFE e 1FFF. Nello Junior-Computer questo vettore indica l'indirizzo 1F32. A questo indirizzo nel programma Monitor sta un'istruzione di salto indiretto. Questo salto indiretto porta la CPU a due locazioni in Pagina 1A, in cui è posto l'indirizzo effettivo. Le due celle che contengono l'indirizzo di salto effettivo, hanno indirizzi 1A7E e 1A7F. In fig. 2 si vede che la CPU, dopo il comando BRK, inizia l'elaborazione dall'indirizzo 1C00. Dato che il vettore IRQ è posto in celle di una RAM, col comando BRK si possono effettuare salti ad indirizzi qualsiasi.

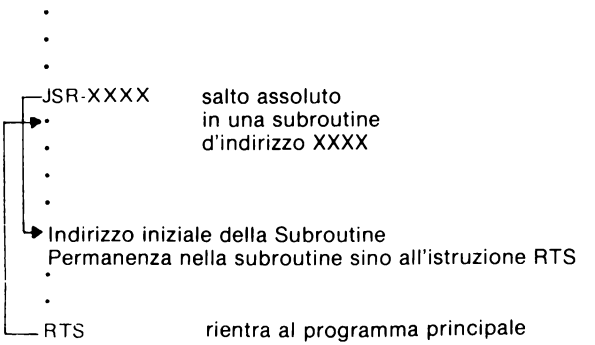
5. Flag BRK contro Flag I, ed altro ancora

Se il processore svolge un IRQ, automaticamente pone ad 1 il Flag

I nel registro P. Se svolge un BRK, automaticamente pone ad 1 il Flag B nel registro P. A che servono questi Flag? La risposta è molto semplice. Sia il comando BRK che l'Interrupt Request lavorano in connessione col vettore IRQ. Se ad esempio più stampanti operano collegate ad un computer centrale, e richiedono tramite le corrispondenti linee d'Interrupt di venire servite dal computer, ci troviamo di fronte a più Interrupt annidati uno entro l'altro. Se l'Interval Timer risulta abilitato a generare un IRQ, è facile trovare 5 o 6 IRQ annidati fra loro. Lo Junior-Computer possiede 16 linee I/O: teoricamente vi potrebbero essere collegate 16 stampanti od altre unità.
Attivando il Flag I, il computer apprende che è stato generato un Interrupt (NMI o IRQ) tramite uno degli ingressi Interrupt della CPU. Il Flag B attivato (= 1) segnala invece al computer che la sequenza d'Interrupt non è stata generata da un segnale applicato agli input Interrupt, ma dal comando BRK. Questo comando, tramite il vettore IRQ, provoca un salto nel programma. In tal modo risulta definito in modo univoco se il vettore IRQ è stato utilizzato da un Interrupt Request o dal comando BRK. Ora dovrebbe risultar chiaro il motivo della presenza dei Flag B e Flag I nel registro P: dopo l'esecuzione del comando BRK, il Flag B rimane alto (log. 1). Se più oltre la CPU incontra un segnale d'Interrupt, contemporaneamente vengono messi il Flag I ad 1 e il Flag B a 0.

Salto mediante il comando BRK in una routine d'Interrupt, che si conclude con RTI

Durante la ricerca di eventuali errori di programma, il programmatore spesso ha bisogno di saltare in una routine d'Interrupt al cui termine è posta l'istruzione RTI. Questo può essere paragonato al salto ad una subroutine; solo che in questo caso l'istruzione di salto eseguita con il comando BRK è lunga non 3, ma solo 1 byte. Tra il comando e l'istruzione JSR si possono osservare le seguenti analogie:



Un salto ad una routine d'Interrupt col comando BRK avviene per il tramite del vettore IRQ. Prima del salto, vengono salvati sullo Stack il contenuto del registro P ed il valore **non corretto** del Program Counter. Prima dell'istruzione RTI si richiede perciò la correzione del Program Counter:

```
.
.
.
BRK  Salta dal programma principale ad una routine Interrupt, che inizia allo
.    indirizzo indicato dal vettore IRQ. Procedi prima a salvare nell'ordine
.    PCH, PCL+2 ed il registro P ponendoli sullo Stack (automatico)
.
```

Indirizzo iniziale della routine d'Interrupt. Permanenza nella routine sino all'istruzione RTI

```
.
.
.
```

La routine d'Interrupt è stata elaborata, va effettuato un salto di rientro al programma principale, inoltre bisogna correggere il PC

```
.
.
.
PLA          recupera il valore originario del registro P dallo Stack
STA-MEM      salva il registro P
PLA          recupera il valore non corretto del PCL dallo Stack
STA-MEM+1    salva il valore non corretto del PC
PLA          recupera il valore non corretto di PCH dallo Stack
STA-MEM+2    salva il valore non corretto del PC
SEC
LDA-MEM+1    preleva il valore non corretto di PCL
SBC # Ø1     correggi il PCL
STA-MEM+1    salva il valore corretto di PCL
LDA-MEM+2    preleva il valore non corretto di PCH
SBC # ØØ     correggi il PCH
PHA          deposita il PCH corretto sullo Stack
LDA-MEM+1    recupera il valore non corretto di PCL
PHA          deposita il PCL corretto sullo Stack
LDA-MEM      recupera il valore originario del registro P
PHA          deposita il valore originario di P sullo Stack
             (Attenzione! Questa sequenza di programma non può venire
             eseguita in Step by step Mode)

RTI          rientra nel programma principale, con il valore corretto del
             Program Counter
```

Osservazione interessante

Durante la verifica di un Basic-Compiler si dovevano spesso effettuare, tramite il comando BRK, salti in una routine di Interrupt conclusa da un'istruzione RTI. Questa routine, a seguito di varie operazioni, aveva influito sui Flag nel registro P. Al rientro della subroutine d'Interrupt nel programma principale, perciò non interessava il contenuto originario del registro P, che il processore aveva salvato sullo Stack prima di eseguire il salto. Se il programmatore richiede invece di conservare dopo il rientro dalla Subroutine d'Interrupt lo stato originale del registro P prima dell'istruzione RTI, si deve impiegare la sequenza di programma:

```
.
.
.
BRK   Salta dal programma d'inizio in una routine di Interrupt, che inizia
.     all'indirizzo indicato dal vettore IRQ. Salva preventivamente nell'ordine
.     PCH, PCL+2 ed il registro P sullo stack (automatico)
.
.
.
Indirizzo d'inizio della subroutine.
.Permanenza nella routine d'Interrupt
.sino all'istruzione RTI
.
.

La routine d'Interrupt è terminata e si
deve rientrare nel programma principa-
le. L'utente richiede nel programma
principale il contenuto del registro P,
modificato dalla routine Interrupt; si de-
ve inoltre correggere pure il PC:

.
.
.
PHP
PLA          salva il contenuto del registro P modificato
STA-MEM
PLA          recupera dallo Stack il contenuto originario di P
PLA          recupera dallo Stack il valore non corretto di PCL
STA-MEM+1    salva il valore non corretto del Program Counter
PLA          recupera il valore non corretto di PCH
STA-MEM+2    salva il valore non corretto del Program Counter
SEC
LDA-MEM+1    recupera il valore non corretto di PCL
SBC # Ø1     correggi PCL
STA-MEM+1    salva il valore corretto di PCL
LDA-MEM+2    preleva il valore non corretto di PCH
SBC # ØØ     correggi PCH
```

PHA	deposita sullo Stack il valore corretto di PHA
LDA-MEM+1	recupera il valore corretto di PCL
PHA	deposita sullo Stack il valore corretto di PCL
LDA-MEM	recupera il valore originario del registro P
PHA	deposita sullo Stack il valore originario di P
	(Attenzione! Questa sequenza di programma non può essere elaborata in Step by step Mode)
RTI	rientra al programma principale con il valore corretto del Program Counter

Nel programma principale è ora disponibile all'utente il valore del registro P, modificato nel corso della routine d'Interrupt. Oltre agli Input NMI ed IRQ, in fig. 2 ne sono illustrati altri. Il Monitor principia all'indirizzo 1C00. Normalmente si esce da questo programma premendo il tasto GO. Il relativo Output è il n° 3 (si confronti con fig. 1). Attraverso l'Input 4 il programmatore può richiamare determinate subroutine del Monitor. Al termine di queste subroutine (RTS) il processore riabbandona il Monitor e rientra nel programma utente.
Abbiamo così descritto tutti gli Input e gli Output del programma Monitor, ed abbiamo afferrato il diagramma di flusso generalizzato di fig. 1. Il motivo per cui abbiamo indugiato nella descrizione del comando BRK e dell'Interrupt è questo: i programmi assumono dimensioni sempre più grandi e richiedono sempre più tempo. L'impegno di tempo per la verifica dei programmi impostati ed i tempi di elaborazione di tali programmi possono ora venire ridotti al minimo, avendo imparato il comando BRK ed il concetto d'Interrupt. Ed ora passiamo a descrivere il programma del Monitor, istruzione per istruzione!

Il Monitor

In fig. 1 sono riportati tre importanti Label del Monitor: RESET, SAVE e START. Vedremo ora di studiare istruzione dopo istruzione l'andamento del programma dopo tali Label. Seguendo il diagramma di flusso del Monitor il principiante ha modo di imparare a conoscere a fondo il "cervello" del suo Junior-Computer. C'é inoltre un altro motivo per cui trattiamo in modo così esauriente il Monitor: i volumi sullo Junior-Computer sono scritti in modo che si possa apprendere autodidatticamente la programmazione in linguaggio macchina. Il lettore volonteroso, seguendo il diagramma di flusso, può apprendere come vengono stesi i programmi da programmatori esperti.

RESET

Affinché lo Junior-Computer si possa rifar vivo tramite il display e sondare la tastiera, dopo la pressione del tasto RST, occorre che esso si porti in uno stato definito. Come sappiamo dal capitolo 6, a tal fine è necessario che il PIA sia opportunamente programmato. Un piccolo programma deve inoltre provvedere a porre in uno stato definito altri registri interni della CPU, come lo Stack-pointer ed il registro Processor Status. Dopo prelevato il vettore di Reset dalle locazioni di indirizzi 1FFC e 1FFE, lo Junior-Computer perviene al Label RESET di fig. 3. Per prima cosa provvede a definire gli I/O della Porta B. Le linee di porta PB1...PB4 vengono programmate quali uscite. Come risulta dal 1° volume, alcune locazioni di memoria in Pagina Zero sono riservate per conservare i contenuti dei registri interni della CPU. Una di queste locazioni è denominata PREG: in essa viene salvato il contenuto del Processor Status. Quando, dopo una sequenza di Reset, si lascia per la prima volta il Monitor tramite la pressione del tasto GO (vedi fig. 1), il contenuto della locazione PREG deve avere un valore definito. Ne chiariremo il motivo più avanti. In PREG viene dunque caricato il valore 04. In fig. 3 è illustrata anche la struttura del registro Processor Status (registro P); si riconoscono facilmente i Flag ed il loro stato: il Flag I è posto ad 1, tutti gli altri Flag a 0. Essendo quindi basso anche il Flag D, il computer opera di qui in avanti in modo binario. Poiché il Flag I è alto, la CPU non può svolgere un IRQ. Un'unità collegata esternamente al bus di controllo, per esempio una stampante, non è quindi in grado di generare per errore un IRQ. Successivamente, il processore scrive il numero nella locazione di memoria che risulta diversa da 0. Se il valore di MODE è diverso da 0, lo Junior-Computer opera in AD-MODE; se il valore di MODE è 0, in DA-MODE. Inoltre il computer scrive il valore 03 nella cella di memoria BYTES. Ossia, sui 6 display dovranno venire visualizzati 3 byte: due byte indirizzo e 1 byte dati. Ricordiamo (capitolo 5) l'impostazione dei programmi mediante l'Editor: il display ha una lunghezza che varia in dipendenza dell'istruzione visualizzata.
Se il contenuto di BYTES vale 01, si illuminano solo i due display di sinistra. Se il valore è 02, si accendono i due display di sinistra ed i due centrali. Tutti e sei i display sono accesi quando il valore di BYTES è 03.

La CPU punta inoltre lo Stack Pointer all'indirizzo 01FF. Dato che lo Stack Pointer della CPU 6502 è sempre in Pagina 1, basta indicare il byte basso del Pointer, il valore del byte alto è sempre 01. Le relative istruzioni sono: LDX FF, TXS. Dato che lo Stack Pointer viene anche conservato in Pagina Zero, occorre depositare il byte basso dello Stack Pointer anche nella locazione SPUSER.

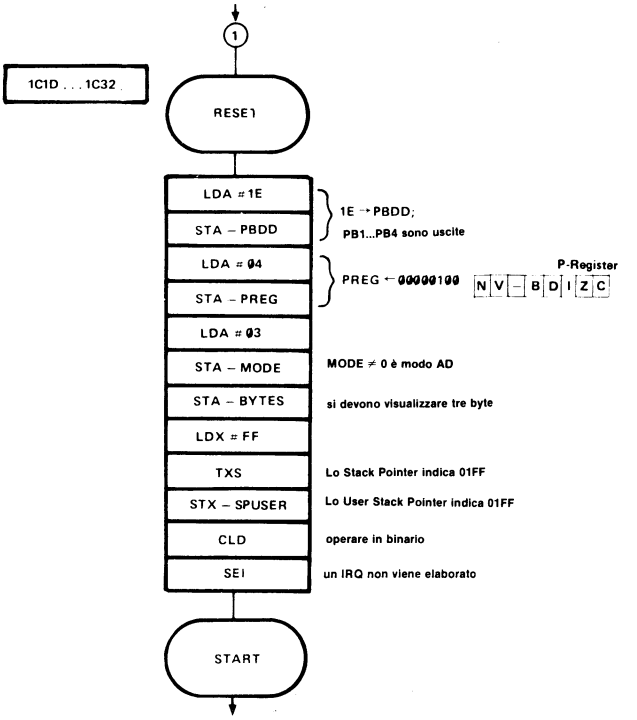


Figura 3. Se il Monitor viene avviato tramite il tasto RST, la CPU percorre la routine RESET. In questa routine il computer definisce l'I/O per la gestione del display e della tastiera. Inoltre vengono fissati pure gli stati di alcuni registri interni della CPU: lo Stack Pointer ed il registro Processor Status.

Le due successive istruzioni fanno sì che il computer nel programma Monitor operi in modo binario e non accetti alcun IRQ. Alla prima uscita dal Monitor con il tasto GO, il Flag I viene posto ad 1 ed il Flag D a 0, dato che in questo caso il contenuto della locazione PREG rappresenta l'effettivo stato del registro P. I motivi saranno chiariti più avanti. Quando si giunge al Label START, si conclude la sequenza di Reset.

SAVE

Riconsideriamo ancora la fig. 1. Dopo l'input ② si trova il Label SAVE. La sequenza di programma che parte da tale Label provvede a conservare i valori di tutti i registri interni della CPU in Pagina Zero. Siamo saltati nel Monitor attraverso l'Input ② in Step by Step Mode, ossia per effetto di un NMI generato dal fronte negativo di un impulso presente all'uscita della porta NAND N5 (fig. 2).
La fig. 4a ci mostra come vengono salvati in Pagina Zero i singoli

registri interni della CPU. Al Label SAVE si perviene in Step by step Mode mediante un NMI. Come già sappiamo, un NMI provvede a depositare sullo Stack PCH, PCL ed il registro P, nell'ordine. La fig. 4b presenta la situazione dello Stack, quando la CPU giunge al Label SAVE: prima del NMI, lo Stack Pointer indicava l'indirizzo 01XX. A questo indirizzo la CPU depone il PCH, e decrementa di 1 la parte inferiore dello Stack-Pointer, che indica così ora 01XX-1. A questa locazione la CPU deposita il PCL, e decrementa ancora una volta lo Stack Pointer, che ora indica 01XX-2. A tale indirizzo viene deposto il contenuto del registro P, e lo Stack Pointer viene portato a segnare 01XX-3.
Adesso il processore è giunto al Label SAVE. Per prima cosa, il contenuto dell'accumulatore viene salvato memorizzandolo in ACC (00F3). I punti ① ... ④ in fig. 4a, b illustrano l'andamento del programma.
Con la successiva istruzione, PLA, il contenuto del registro P viene recuperato dallo Stack e memorizzato in PREG (punto ②). Al Label SAVE il processore incontra un'altra istruzione PLA. Con questa si recupera dallo Stack il byte basso del Program Counter, che la CPU salva memorizzando in Pagina Zero nella cella PCL nonché in POINTL (punto ③). Con la ulteriore istruzione PLA il processore recupera il byte del Program Counter dallo Stack, e lo salva analogamente nelle locazioni di memoria PCH e POINTH in Pagina Zero (punto ④). Ricapitoliamo: prima del Label SAVE lo Stack Pointer indicava l'indirizzo 01XX. Dopo l'Interrupt la CPU ha provveduto a salvare PCH, PCL ed il registro P sullo Stack. Giunto al Label SAVE, lo Stack Pointer indica l'indirizzo 01XX-3. Tra i Label SAVE e SAVEB il processore ha eseguito per tre volte successive un'istruzione PLA. A questo punto (punto ④) lo Stack Pointer ripunta 01XX, ritornando al suo stato originario.
Sin qui il processore, con l'ausilio dello Stack quale memoria temporanea, ha conservato i valori di PCL, PCH e del registro P, ed ha rinfrescato il buffer di display, per effetto dell'operazione di scrittura del nuovo valore del Program Counter nelle celle POINTL, POINTH. Ora si devono salvare ancora gli altri registri della CPU. Il processore esegue tale compito a partire dal Label SAVEB. I contenuti dei registri Y ed X sono salvati rispettivamente nelle locazioni YREG ed XREG. Neppure il valore dello Stack-Pointer è ancora stato salvato. Le due successive istruzioni, TSX e STX-SPUSER salvano il contenuto dello Stack Pointer nella cella SPUSER. Lo stato dello Stack Pointer non si era più modificato a partire dal punto ④. Nella cella di memoria SPUSER possiamo verificare il valore del byte basso d'indirizzo dello Stack-Pointer, XX. Così si sono salvati i registri interni della CPU. Susseguentemente il processore fa ancora qualcosa di utile: scrive un numero, diverso da zero, nella locazione MODE: ciò fa sì che lo Junior-Computer, in Step by step Mode, dopo l'esecuzione di un'istruzione commuti automaticamene in AD-MODE. In tal modo si evita

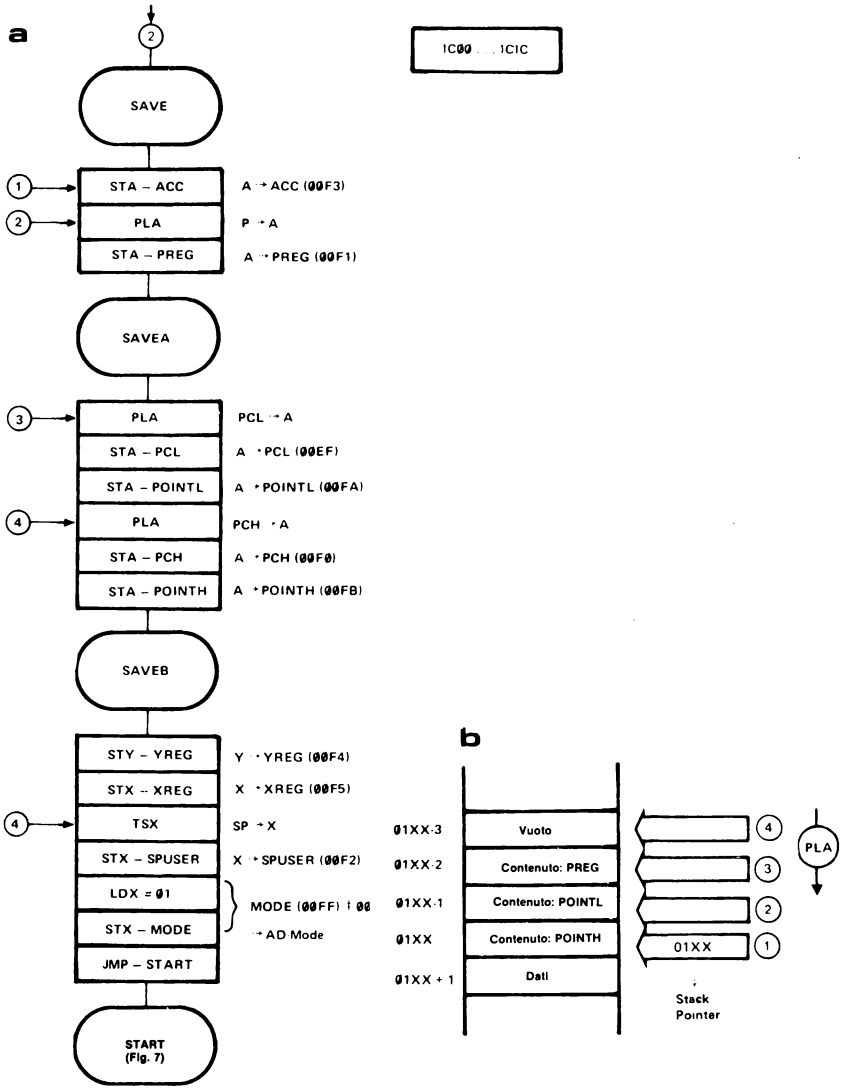


Figura 4a. Questo è il diagramma di flusso dettagliato della Routine SAVE. Mediante questa routine lo Junior-Computer può percorrere un programma in Step by Step Mode. Tutti i registri interni della CPU vengono copiati in determinate locazioni in Pagia Zero. Normalmente SAVE è richiamato tramite un NMI. Dopo l'esecuzione di ciascuna istruzione in Step by Step Mode, di seguito al Label SAVEA il display viene rinfrescato automaticamente.

Figura 4b. In Step by Step Mode lo Stack Pointer funge da importante memoria intermedia temporanea. Per estrarre i dati posti sullo Stack il programma Monitor si giova dello Stack Pointer.

che l'utente cancelli erroneamente delle istruzioni sovrascrivendole con dei dati.
Ora il processore è pervenuto al Label START, e si occupa di gestire il display e dell'interrogazione periodica della tastiera. Premendo il tasto GO, come ben si sa, si può uscire dal programma Monitor (vedi fig. 1) e saltare al programma utente. Nella sezione che segue descriviamo passo passo come avviene questo salto.

L'uscita dal Monitor

Come già sappiamo, lo Junior-Computer quando si trova nel programma Monitor si mantiene in un loop di attesa. Durante questo ciclo esso gestisce il display e sonda periodicamente la

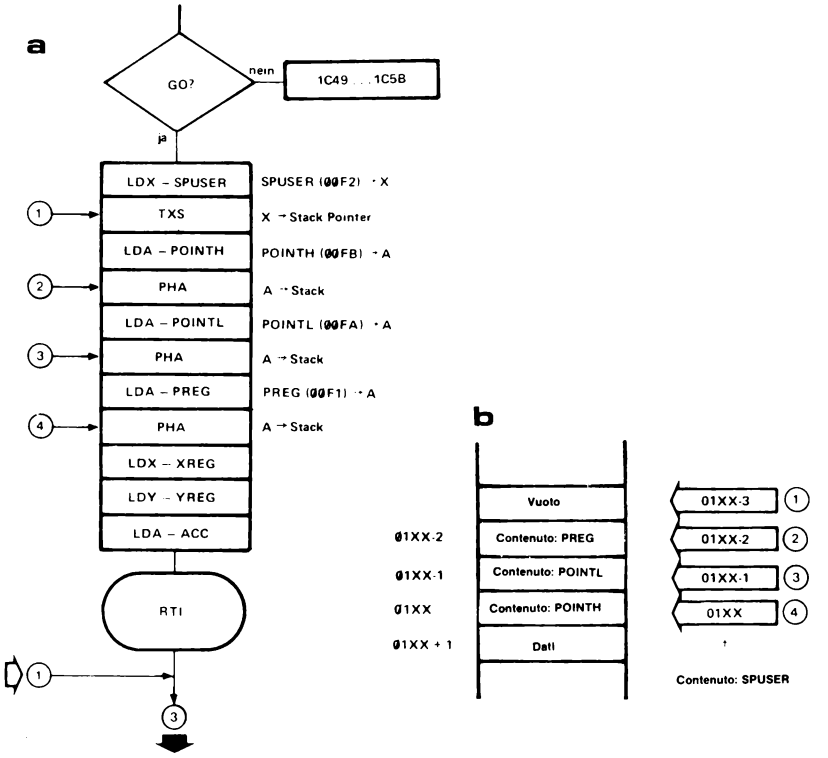


Figura 5a. Ecco il diagramma di flusso dettagliato del tratto di programma svolto dal processore quando l'utente preme il tasto G0. Uno dei modi per uscire dal programma Monitor è appunto la pressione di questo tasto, col quale si salta al programma utente. Le situazioni istantanee dello Stack Pointer qui illustrate coincidono con quelle di fig. 4b.

Figura 5b. Stato dello Stack e dello Stack Pointer in vari momenti di esecuzione del programma di fig. 5a.

tastiera. Se il programmatore preme il tasto GO, il computer esce dal loop e salta al programma utente. Come sappiamo dal 1° volume: dopo la pressione del tasto GO, lo Junior-Computer inizia l'elaborazione dall'indirizzo che al momento è visualizzato sul display. Appena uscito dal programma Monitor, lo Junior-Computer non si occupa più del display e della tastiera, ma esclusivamente del programma utente.
In fig. 5a e 5b è mostrato come lo Junior-Computer esce dal programma Monitor premendo il tasto GO ed inizia l'elaborazione a partire dall'indirizzo visualizzato. Anche qui lo Stack viene utilizzato quale memoria temporanea intermedia (punti ① ... ④). Per prima cosa, la CPU ristabilisce il valore originario dello Stack Pointer: le relative istruzioni sono: LDX-SPUSER, TXS. Lo Stack Pointer indica ora nuovamente l'indirizzo 01XX (punto ①). Poi si provvede a trasferire sullo Stack il contenuto dei buffer di display POINTH e POINTL. Dopo aver deposto POINTH sullo Stack, lo Stack Pointer indica l'indirizzo 01XX-1 (punto ②); e dopo il trasferimento di POINTL sullo Stack, lo Stack Pointer indica 01XX-2 (punto ③). Ora il contenuto di POINTH, POINTL viene salvato, come nuovo Program Counter, sullo Stack. Quindi, il contenuto originario del registro P va sullo Stack: le relative istruzioni sono: LDA-PREG, PHA. Lo Stack Pointer segna ora l'indirizzo 01XX-3 (punto ④). Adesso che il Program Counter ed il registro P sono sullo Stack, la CPU ristabilisce i contenuti originari dell'Accumulatore nonché dei due registri-indice X ed Y. In tal modo tutti i registri interni della CPU hanno riassunto uno stato definito e lo Junior Computer può rientrare con un salto nel programma principale. Questo salto viene effettuato a seguito dell'istruzione RTI. Come rammentiamo, ad una istruzione RTI vengono prelevati nell'ordine dallo Stack il registro P, PCL e PCH. PCH e PCL costituiscono il contenuto dei due buffer di display POINTH e POINTL. Il processore abbandona il loop d'attesa nel programma Monitor ed inizia l'elaborazione a partire dall'indirizzo visualizzato sul display.

Osservazione: se lo Junior-Computer si trova in Step by step Mode, alla pressione del tasto GO esso effettua la sequenza illustrata in Fig. 5. Viene eseguito il salto all'indirizzo visualizzato sul display, al quale si trova un'istruzione, che viene eseguita. Per effetto del caricamento del codice OP, viene generato un NMI tramite l'output SYNC della CPU. Il processore completa l'esecuzione dell'operazione in corso, e svolge quindi il NMI. Il vettore NMI, nello Step by step Mode, indica l'indirizzo 1C00. A questo indirizzo si trova il Label SAVE. La CPU esegue quindi la sequenza di programma illustrata in fig. 4. Tutti i registri interni vengono salvati, e lo Junior-Computer commuta in AD-Mode. Giunto così al Label START, il processore si occupa, in un loop di attesa, di display e tastiera. Sul display compare l'istruzione immediatamente successiva con relativo indirizzo. Lo Junior-Computer,

successivamente al Label START, attende una nuova pressione del tasto GO, per eseguire l'istruzione visualizzata. Ad ogni pressione del tasto GO si replica l'intero processo ora descritto.

Così lo Junior-Computer "dà retta" all'utente

In questa sezione descriveremo come reagisce lo Junior-Computer quando viene premuto un tasto. Occorre a tal fine che il computer, entrato in un loop di attesa, riconosca il tasto premuto. Il riconoscimento del tasto, ossia il calcolo del valore del tasto, avviene in una subroutine. Al termine di questa subroutine, il valore del tasto si trova nell'Accumulatore della CPU. Già dal 1° volume sappiamo che ai tasti sono assegnati i seguenti valori:

0:00	5:05	A:0A	F:0F	PC:14
1:01	6:06	B:0B	AD:10	tasto errato
2:02	7:07	C:0C	DA:11	od illegale: 15
3:03	8:08	D:0D	+:12	
4:04	9:09	E:0E	GO:13	

In fig. 6 si mostra come il computer identifica un tasto premuto. Una istruzione CMP seguita da un'istruzione BNE "filtra" il tasto premuto. La CPU effettua un salto, se il tasto effettivamente premuto non corrisponde al valore del tasto di paragone. Se invece i due valori si corrispondono, il processore si occupa del tasto premuto.
La tastiera dello Junior-Computer si può suddividere in due zone: una zona per i tasti dati ed una zona per i tasti comando. Cosa avviene premendo il tasto GO, già lo abbiamo imparato. Se si preme il tasto AD, tasto comando, il contenuto della locazione MODE assume un valore diverso da zero. Lo Junior-Computer interpreta ogni tasto dati premuto quale indirizzo. Tramite la seguente istruzione BNE il programma salta, via STEPA, al Label START.
Se invece è stato premuto per esempio il tasto comando DA, il computer rende eguale a 0 il contenuto di MODE. Lo Junior-Computer interpreta allora ogni tasto dati premuto quali dati, che vengono depositati all'indirizzo visualizzato e compaiono successivamente sul display.
Premendo il tasto +, il computer incrementa di 1 l'indirizzo visualizzato, e presenta successivamente sul display i dati giacenti a tale indirizzo. Supponiamo, per es., che il contenuto di POINTL prima della pressione del tasto + fosse FF; dopo premuto + questo vale 00. Si deve di conseguenza correggere il valore del buffer di display POINTH, ossia incrementarlo di 1. Questa correzione viene svolta mediante un'istruzione BNE. Successivamente il programma salta nuovamente, via STEPA, al Label START.

Premendo il tasto PC, il computer copia l'ultimo valore del Program Counter sul display indirizzi. Bastano un paio di istruzioni. Poi vengono copiati PCL, PCH (posti in Pagina Zero) nei buffer di display POINTL, POINTH, ed infine il programma Monitor risalta al Label START. Il funzionamento del tasto PC ci è già noto dal 1° volume: operando in Step by step Mode, è facile che si debba spesso controllare lo stato dei registri interni della CPU dopo singoli tratti di programma. Dato che questi registri sono posti in Pagina Zero, bisogna periodicamente lasciare il programma utente per qualche tempo. La pressione del tasto PC ristabilisce allora il contenuto originario del display. Un'ulteriore pressione del tasto GO fa passare alla successiva istruzione..
Quanto sopra descritto ha valore alla condizione, che il programma Monitor riconosca esattamente tutti i tasti comando. Poiché ad ogni tasto è assegnato un valore fisso, possono verificarsi solo valori fra 0 e 14. Può accadere comunque, per un disturbo sull'alimentazione o per rimbalzi nei tasti, che lo Junior-Computer calcoli un valore non corretto di un tasto. Se questo (nonostante la Software anti-rimbalzo) accade, il computer ignora semplicemente il tasto, al Label ILLKEY, e salta nuovamente, via STEPA, al Label START.

Modifica dei dati

Se l'utente non preme un tasto comandi bensì un tasto dati, il programma esegue un salto al Label DATA. A partire da questo Label il computer reagisce ai tasti dati 0...F. Per prima cosa, il valore del tasto va nella cella di memoria KEY, lasciando l'accumulatore libero per altri compiti. Poi, viene trasferito il contenuto della cella di memoria MODE nel registro Y. Il contenuto del registro Y a questo punto può essere o meno uguale a 0. Se ad es. il registro Y è 0, il tasto dati premuto, il cui valore di tasto sta in KEY, viene interpretato quale introduzione di un dato nello Junior-Computer. Se invece il contenuto del registro Y è diverso da 0, si segnala al computer che l'utente desidera introdurre un nuovo indirizzo, ed il dato nella cella di memoria KEY viene associato al display indirizzi. Un'istruzione BNE distingue fra introduzione di indirizzi e di dati: LDY-MODE, BNE-MODE, BNE-ADDRES.
Se il registro Y era 0, si trattava di una introduzione di dati e non viene eseguito il salto al Label ADDRES. Lo Junior-Computer provvede a spostare il tasto premuto da destra a sinistra nel display dati, che sono i due display più a destra, a cui è assegnato il buffer di display INH. Le istruzioni per lo spostamento a sinistra nel display dati funzionano come segue:
Tramite il Pointer-indirizzi POINTH, POINTL la CPU carica i dati visualizzati sul display nell'Accu, servendosi dell'indirizzamento indiretto indicizzato, per cui il registro Y vale 0. Il valore del tasto di

KEY a questo momento non subisce ancora lo spostamento nel display. Supponiamo a titolo di esempio che i dati appena caricati constino dei bit p ... w disposti nell'Accumulatore come segue:

p	q	r	s	t	u	v	w . . .	Contenuto dell'Accu dopo il caricamento
q	r	s	t	u	v	w	Ø . . .	Contenuto dell'Accu dopo il 1° ASL
r	s	t	u	v	w	Ø	Ø . . .	Contenuto dell'Accu dopo il 2° ASL
s	t	u	v	w	Ø	Ø	Ø . . .	Contenuto dell'Accu dopo il 3° ASL
t	u	v	w	Ø	Ø	Ø	Ø . . .	Contenuto dell'Accu dopo il 4° ASL

Dopo 4 spostamenti il Nibble dati basso è passato in quello alto, mentre il contenuto di quello alto è andato perduto. Il Nibble a seguito degli spostamenti è ora uguale a zero.
Il contenuto della cella del buffer KEY vale ØØØØXXXX, dove gli X sono i valori dei tasti premuti. Mediante una operazione OR tra l'Accu e la cella di memoria KEY si ottiene ora:

t	u	v	w	Ø	Ø	Ø	Ø	Contenuto dell'Accu prima di ORA
Ø	Ø	Ø	Ø	X	X	X	X	Contenuto di KEY
t	u	v	w	X	X	X	X	Risultato nell'Accu dopo ORA

Il Nibble alto non è stato modificato, mentre il Nibble basso è divenuto eguale al valore dei tasti premuti. In tal modo si è realizzato lo spostamento dei tasti premuti da destra verso sinistra nell'Accu. Il byte dati così ottenuto viene memorizzato dalla CPU, mediante il Pointer indirizzi POINTH, POINTL, all'indirizzo indicato. Infine si ha ancora il salto al Label START via STEPA.
Il Label STEPA può a prima vista apparire singolare. Esso è comunque necessario, perché dopo di esso stanno delle istruzioni Branch che portano al Label START. Queste istruzioni di Branch passano attraverso un campo di memoria di più di - 128 passi: perciò è necessario saltare direttamente a START tramite STEPA.

Modifica di indirizzi

Così come il programmatore può modificare dati presenti nella memoria dello Junior-Computer, egli può introdurre nuovi indirizzi. Il contenuto delle celle POINTH e POINTL, che costituiscono il Pointer indirizzi, può venir modificato dopo aver premuto il tasto AD. Queste modifiche vengono effettuate dal tratto di programma del Monitor situato dopo il Label ADDRES. Poiché nel registro X inzialmente si è caricato 04, il processore replica 4 volte il loop di programma ADLOOP-BNE-ADLOOP. Il contenuto di POINTL viene corrispondentemente spostato di quattro posizioni a sinistra, ed il contenuto di POINTH ruotato 4 volte a sinistra.
Ricordare: in corrispondenza ad ASL in b0 viene inserito uno 0 ed il Flag C viene posto eguale a b7. Invece per ROL è il Flag C a

essere spostato in b0 e b7 posto eguale al valore del Flag C. Con le assunzioni che seguono si potranno facilmente comprendere le operazioni di spostamento e di rotazione:
Il nibble alto di POINTH sia h i j k
il nibble basso di POINTH sia l m n o
il nibble alto di POINTL sia p q r s
il nibble basso di POINTL sia t u v w
Inizialmente il Flag C può risultare 0 od 1. La situazione iniziale corrisponde all'indirizzo al momento visualizzato sul display. L'indirizzo modificato è l'indirizzo che compare sul display se il programmatore preme, in modo AD, uno dei tasti dati 0... F. L'indirizzo modificato è formato dal computer nel modo seguente:

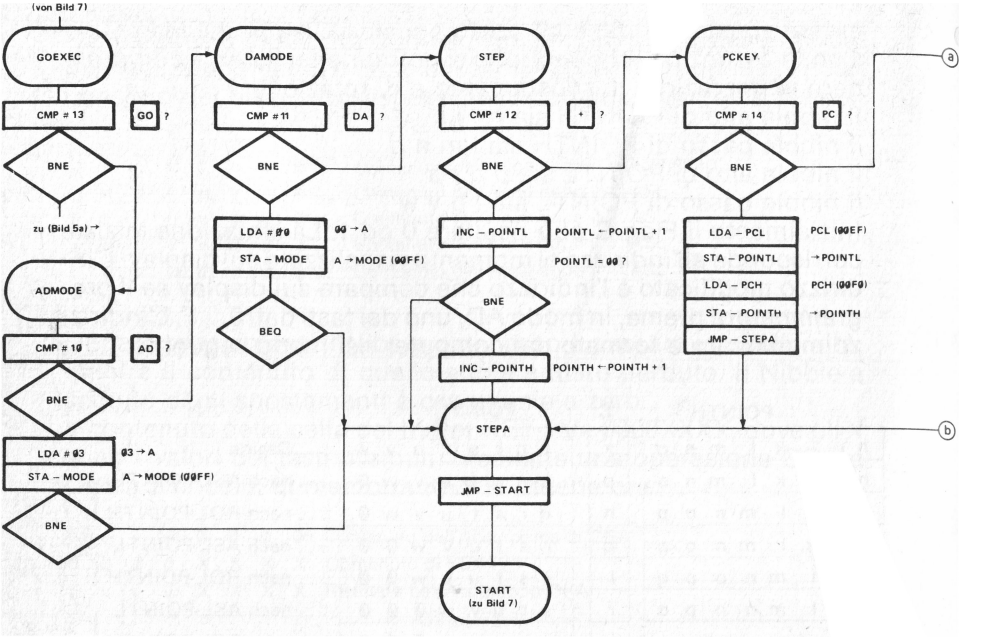
POINTH	C	POINTL		
h i j k l m n o	x	p q r s t u v w	... Beginn	
h i j k l m n o	p	q r s t u v w Ø	... nach ASL-POINTL	X = Ø4
i j k l m n o p	h	q r s t u v w Ø	... nach ROL-POINTH	
i j k l m n o p	q	r s t u v w Ø Ø	... nach ASL-POINTL	X = Ø3
j k l m n o p q	i	r s t u v w Ø Ø	... nach ROL-POINTH	
j k l m n o p q	r	s t u v w Ø Ø Ø	... nach ASL-POINTL	X = Ø2
k l m n o p q r	j	s t u v w Ø Ø Ø	... nach ROL-POINTH	
k l m n o p q r	s	t u v w Ø Ø Ø Ø	... nach ASL-POINTL	X = Ø1
l m n o p q r s	k	t u v w Ø Ø Ø Ø	... nach ROL-POINTH	

Adesso il contenuto dei due buffer di display POINTH e POINTL risulta spostato di 4 posizioni verso sinistra. Il Nibble alto di POINTL figura ora come nibble basso di POINTH, mentre il nibble alto di POINTH è andato perso. Nelle operazioni di spostamento/rotazione è il Flag C a fungere da stazione intermedia.

Il contenuto di POINTL viene ora trasferito nell'Accu e viene eseguita una relazione OR col contenuto della cella KEY:

```
t u v w Ø Ø Ø Ø  Contenuto dell'Accu prima di ORAZ-KEY
Ø Ø Ø Ø X X X X  Contenuto di KEY
t u v w X X X X  Risultato dop ORAZ-KEY
```

Il risultato così ottenuto viene memorizzato dal processore nella locazione POINTL, rinfrescando al tempo stesso il buffer di display per la visualizzazione dell'indirizzo. Il Nibble basso di POINTL corrisponde al valore dei tasti indirizzo premuti. Tramite STEPA il programma salta al Label START, e lo Junior-Computer presenta l'indirizzo modificato con i relativi dati sul display. Dunque: un tasto numerico corrisponde sempre ad un nibble dati, che viene trasferito al display indirizzo o al display dati. A tal fine lo



Junior-Computer controlla il valore della cella MODE e stabilisce così se il tasto dati premuto deve essere interpretato come indirizzo o come normale dato da introdurre nel computer.
Così abbiamo descritto la parte principale del programma Monitor. In Fig. 1 al principio del capitolo è riportato il diagramma a blocchi, mentre la fig. 6 mostra il diagramma di flusso dettagliato. Da quest'ultimo si ricava il modo con cui lo Junior-Computer distingue la pressione di un tasto dati da quella di un tasto comando, e come esso converte in comandi i singoli tasti comandi. Così pure risulta chiaro il modo in cui lo Junior-Computer elabora internamente dati e comandi introdotti dal programmatore. La prossima sezione mostrerà come il computer si fa vivo tramite il display: solo allora può stabilirsi un dialogo fra programmatore e computer.

Così lo Junior-Computer "risponde" all'utente

Consideriamo ancora una volta la fig. 1! Immediatamente dopo il Label START il computer è al servizio della tastiera e del display. Ossia, calcola il valore di un tasto premuto e lo visualizza sul display, se si tratta di uno dei tasti dati 0...F. Visualizzando sul display un tasto dati premuto il computer risponde al programmatore.

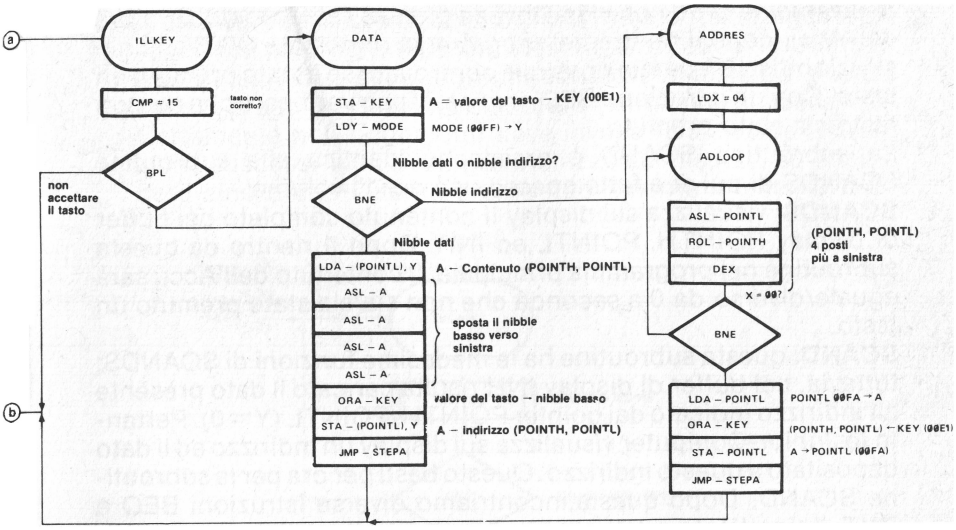


Figura 6. La figura mostra come vengono "filtrati" i tasti comando AD, DA, + e PC. Per ogni diverso tasto vi è una distinta routine, che il processore svolge dopo aver identificato il tasto comando. Dopo il Label DATA i tasti dati vengono spostati nel display dati, mentre dopo il Label ADDRESS i tasti dati sono portati nel display indirizzi.

Sappiamo già che al Label START si può pervenire in diversi modi:

* dopo il trattamento riservato ad un tasto premuto nella tastiera, il programma Monitor esegue un salto al Label START;
* dopo il rientro nel programma Monitor attraverso il Label SAVE il computer salta al Label START. Abbiamo tratto vantaggio di questa possibilità quando lo Junior-Computer opera in Step by step Mode oppure al termine di un programma era situato un comando BRK;
* anche quando viene premuto il tasto RST il Programma Monitor, dopo aver provveduto a programmare il PIA, salta al Label START.

Dopo il Label START il computer permane in un loop di attesa, da cui in generale può uscire premendo il tasto GO. Ora descriveremo istruzione per istruzione cosa avviene nel programma Monitor dopo il Label START.

In fig. 7 sono illustrati i primi passi di programma dopo START. La subroutine SCAND (che descriveremo più oltre) provvede alla visualizzazione dei tre buffer di display POINTH, POINTL e INH ed a verificare se un tasto risulta premuto oppure no. Non appena nel corso di SCAND viene premuto un tasto della tastiera, si ha il rientro nel programma principale, con un valore nell'Accumulato-

re diverso da 0. Se non viene invece premuto alcun tasto, il valore dell'Accu dopo il rientro nel programma principale è 0. Con un'istruzione BNE è perciò possibile controllare se è stato premuto un tasto. Con un'istruzione BEQ, invece, si può verificare che nessun tasto sia stato premuto.
La subroutine SCAND è pressoché identica alla subroutine SCANDS di cui si è fatto spesso uso nel 1° volume:
SCANDS: visualizza sul display il contenuto completo dei buffer di display POINTH, POINTL ed INH. Dopo il rientro da questa subroutine nel programma principale, il contenuto dell'Accu sarà eguale/diverso da 0 a seconda che non sia/sia stato premuto un tasto.
SCAND: questa subroutine ha le medesime funzioni di SCANDS; tuttavia, nel buffer di display INH risulta caricato il dato presente all'indirizzo indicato dai pointer POINTH, POINTL (Y = 0). Pertanto lo Junior-Computer visualizza sul display un indirizzo ed il dato depositato a questo indirizzo. Questo basti per ora per la subroutine SCAND. Dopo questa,incontriamo diverse istruzioni BEQ e BNE. Il significato di tali istruzioni sarà chiarito dall'esempio pratico che segue:

1) Supponiamo che lo Junior-Computer stia operando in Step by step Mode. Sul display è visualizzata un'istruzione, che il computer deve elaborare, ed il relativo indirizzo.
2) L'utente preme il tasto GO. Lo Junior-Computer elabora l'istruzione visualizzata. Mentre sta provvedendo a caricare l'istruzione macchina nella CPU, tramite l'output SYNC e la porta N5 viene emesso un NMI (fig. 2). Il computer termina l'elaborazione dell'istruzione in corso e tramite il vettore NMI salta al Label SAVE (vedi il diagramma a blocchi in fig. 1 di questo capitolo). Ivi giunto, salva tutti i registri interni della CPU in prestabilite locazioni in Pagina Zero, e rinfresca i buffer di display POINTH, POINTL per visualizzare l'indirizzo sul display. Il Pointer indirizzi POINTH, POINTL indica quindi l'indirizzo a cui è posta la successiva istruzione. Lo Junior-Computer è ora nuovamente giunto al Label START.
3) Che cosa deve ora fare il computer? Chiaro: deve visualizzare sul display la successiva istruzione con relativo indirizzo. Il buffer di display per gli indirizzi, cioé POINTH, POINTL è stato precedentemente corretto nella subroutine SAVE. L'indirizzo mostrato da queste due locazioni di memoria è quello dell'istruzione macchina successiva. L'istruzione, che sarà successivamente elaborata, viene caricata dal computer, nel corso della subroutine SCAND (fig. 7), nel buffer di display INH. Così è stato rinfrescato l'intero display. Tutte queste operazioni, descritte ai punti 1...3, vengono eseguite dal computer in frazioni di secondo! La subroutine SCAND, come è noto, identifica un eventuale tasto premuto della tastiera. La

pressione del tasto GO ha fatto sì che il computer svolgesse la sequenza di programma ora descritta. Quando lo Junior-Computer rientra dalla subroutine SCAND, il tasto GO risulta ancora premuto. Deve ora elaborare l'istruzione successiva? Certamente no! Il computer deve invece visualizzare l'istruzione successiva ed il relativo indirizzo sul display, ed attendere sino a che l'utente preme nuovamente il tasto GO. E

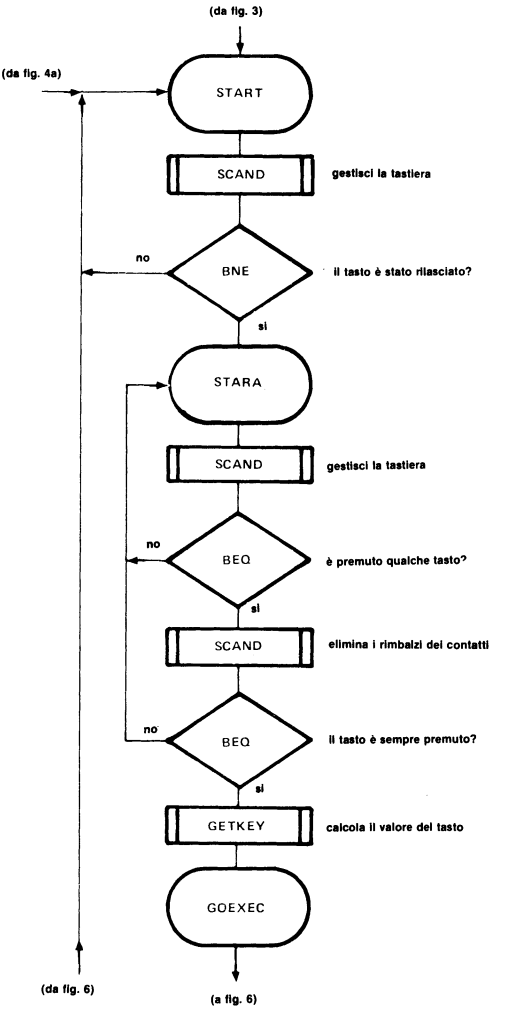


Figura 7. Questa parte del programma Monitor esegue il trasferimento del contenuto dei buffer di display POINTH, POINTL ed INH al display dello Junior Computer. I dati da visualizzare vengono opportunamente convertiti nel codice a 7 segmenti. Successivamente il processore passa a sondare la tastiera. Se viene accertato un tasto premuto, il valore del tasto viene calcolato nella subroutine GETKEY.

prima di poter ripremere il tasto GO, esso deve venire naturalmente rilasciato! Dopo eseguita l'istruzione, perciò, lo Junior-Computer si trattiene nel loop di attesa START-BNE-START sino a che il tasto GO viene rilasciato.

4) L'utente dunque lascia libero il tasto GO. Il computer al rientro dalla subroutine SCAND ha un contenuto dell'Accu di 0, (indice che nessun tasto è premuto), e perviene al Label STARA. Segue un nuovo salto alla subroutine SCAND: istruzioni e relativo indirizzo compaiono sul display. Il tasto GO, che nello Step by step Mode provoca l'elaborazione dell'istruzione al momento visualizzata, non è ancora stato premuto. Quindi il contenuto dell'Accu al rientro dalla subroutine SCAND è 0. Lo Junior-Computer permane nel loop di attesa STARA-BEQ-STARA.
5) Il programmatore desidera ora che il computer svolga l'istruzione successiva. Preme perciò il tasto GO. Lo Junior-Computer, al rientro dalla subroutine SCAND durante il ciclo STARA-BEQ-STARA; ha il valore 0 nell'Accu (indice che un tasto risulta premuto). Lo svantaggio dei tasti meccanici è nei loro rimbalzi. Perciò viene richiamata ancora una volta la subroutine SCAND. Quando il computer rientra da questa subroutine, è trascorso abbastanza tempo ed i rimbalzi del tasto si sono completamente smorzati. La seconda istruzione BEQ controlla se il tasto risulta ancora premuto. Se non è così, il programma salta al Label STARA, il che significa che è stato un impulso di disturbo a provocare l'uscita dal loop di attesa. Il computer attende la nuova pressione di un tasto.
6) Se invece il tasto, dopo l'ultimo richiamo di SCAND, risulta ancora premuto, il computer esce dal loop di attesa e salta alla subroutine GETKEY. In questa subroutine esso calcola il valore del tasto premuto. Dopo il rientro dalla subroutine GETKEY, il valore corrispondente al tasto sta nell'Accu della CPU, e siamo pervenuti al Label GOEXEC. Dopo questo Label vengono nuovamente trattati i tasti dati ed i tasti comandi (fig. 6).

Ricapitolazione

Dopo il Label START, il computer provvede al servizio della tastiera e del display. Vengono utilizzate a tale scopo le subroutine SCAND e GETKEY, nonché istruzioni BNE e BEQ... nel loop il computer permane STARA-BEQ-STARA sin quando viene premuto un tasto. Nel corso di questo ciclo di attesa viene anche gestito il display. Tra il Label START ed il Label GOEXEC il computer passa attraverso diversi loop, per svolgere compiti definiti:

* Loop START-BNE-START: in questo loop il computer gestisce la tastiera ed il display. Permane nel loop sin quando un tasto premuto viene lasciato libero.

* Loop STARA-BEQ-STARA: in questo loop il computer gestisce la tastiera ed il display. Permane nel loop sino a quando viene premuto un tasto. Durante l'introduzione di dati, lo Junior-Computer passa buona parte del suo tempo in questo loop.
* Loop STARA-BEQ-BEQ-STARA: in questo loop si provvede all'eliminazione dei rimbalzi dei tasti. Il salto dal secondo BEQ al Label STARA ha luogo solo quando dopo la subroutine SCAND i rimbalzi del tasto non si sono ancora spenti, oppure nel caso in cui un impulso di disturbo abbia provocato una pressione illegale di un tasto.

I rimbalzi dei tasti

Supponiamo che la pressione d'un tasto porti una delle linee di porta PA0 ... PA6 allo stato logico 0: idealmente l'andamento della tensione su questa linea di porta sarebbe un impulso negativo di onda quadra (fig. 8a). Se si abbandona il tasto, tutte le linee di porta tornano allo stato 1. In pratica, la pressione di un tasto provoca una serie di rimbalzi del contatto, come mostra la fig. 8b. Ai tempi t3 e t5 la tensione supera la soglia di trigger della porta logica presente nel CI 6532. Il computer non è in grado da solo, in corrispondenza a tali istanti, di stabilire se il tasto è premuto oppure no. Perciò il programma Monitor deve disporre di un tempo di ritardo, per sondare lo stato della linea di porta solo dopo che il rimbalzo dei tasti si è spento, al punto t6. La subroutine SCAND impegna il computer per alcuni millesecondi: questo

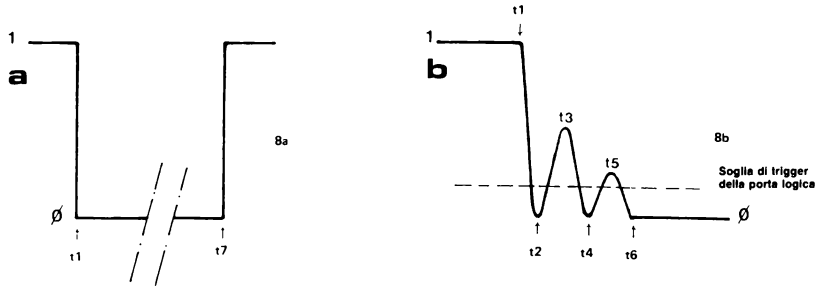


Figura 8. La fig. 8a mostra come dovrebbe essere l'andamento ideale della tensione su di una linea di Porta A, quando viene attivato uno dei tasti che il computer sta sondando. Se l'utente preme un tasto, la corrispondente linea di porta vien portata a livello logico 0 (tempo t1). A tempo t7 il tasto viene lasciato libero. La linea di porta si riporta a livello logico 1.
La fig. 8b mostra invece come si presenta in pratica l'andamento della tensione su una linea di Porta A, quando l'utente preme il relativo tasto. Si manifestano rimbalzi dei contatti del tasto. Questi rimbalzi vengono eliminati richiamando per due volte la subroutine SCAND di fig. 7. Così nel calcolo del valore del tasto nella subroutine GETKEY non si avranno errori, dato che la linea di porta si è già stabilizzata.

tempo è sufficiente al completo smorzamento dei rimbalzi del tasto. Il ritardo di tempo richiesto è provocato dalla sequenza di programma BEQ-SCAND-BEQ. Se in corrispondenza alla seconda istruzione BEQ è ancora presente il rimbalzo dei contatti, il processore ripercorre per una seconda volta la subroutine SCAND. In altri termini: la subroutine SCAND viene percorsa per tutto il tempo necessario al completo esaurimento dei rimbalzi del tasto sulla linea di porta.

Il programmatore "conversa" con lo Junior-Computer

Come abbiamo ripetutamente detto, il discorso fra utente e computer avviene attraverso la tastiera ed il display. L'I/O dello Junior-Computer svolge a tal fine un duro lavoro: l'I/O gestisce una Hardware (= Display) governata da Software, oppure il computer legge tramite l'I/O dati generati dalla Hardware (= Tastiera).
In fig. 9 si illustra nuovamente (confronta volume 1°, capitolo 1, fig. 4) l'I/O con i suoi collegamenti. La Porta B (PB1 ... PB4) lavora esclusivamente come uscita, e risulta collegata ai punti A ... D di IC7. In funzione della configurazione dei bit applicati agli ingressi A ... D, una delle uscite di IC7 assume lo stato logico 0. Sei delle dieci uscite sono collegate ai dati comuni del display. Se i segmenti di un dato display si devono accendere, il corrispondente catodo del display deve essere messo a potenziale di massa. Questo compito è svolto dalla Porta B tramite il commutatore del display IC7 (decodifica BCD-decimale). I particolari sono forniti dalla seguente "tabella della verità":

Tabella 1

PB7	PB6	PB5	PB4	PB3	PB2	PB1	PBØ	display acceso	Display
			D	C	B	A			
X	X	X	Ø	1	Ø	Ø	X	Di1	ADH (buffer POINTH)
X	X	X	Ø	1	Ø	1	X	Di2	
X	X	X	Ø	1	1	Ø	X	Di3	ADL (buffer POINTL)
X	X	X	Ø	1	1	1	X	Di4	
X	X	X	1	Ø	Ø	Ø	X	Di5	dati (buffer INH)
X	X	X	1	Ø	Ø	1	X	Di6	

Quando lo stato di un bit di porta è indicato con "X" significa che esso può essere indifferentemente 0 od 1 logico, dato che alla relativa linea di porta non è collegato alcunché.
Se un determinato display risulta inserito, tramite il commutatore di display IC7, e tutte le uscite di IC11 sono allo stato logico 1, si accendono tutti i segmenti del display, che segna quindi "8". Un display deve poter riprodurre i valori 0 ... F, ossia 16 diversi valori. Si deve quindi poter includere od escludere secondo necessità determinati segmenti. Questa funzione è svolta dalla Porta A,

quando sia stata definita quale output. La Porta A opera quindi come commutatore dei segmenti.
Se ad es. il segmento denominato "c" deve resettare spento, dobbiamo fare in modo che l'uscita dell'invertitore collegato al piedino 12 di IC11 sia a potenziale di massa. La corrente da questo output passa attraverso la resistenza R11, ed il segmento non riceve tensione. Per ottenerlo, è necessario che il corrispondente input dell'invertitore (piedino 5 di IC11) sia alto (log. 1). In altri termini: la linea output PA2 deve trovarsi allo stato logico 1. In questo modo è possibile includere od escludere, in funzioni della configurazione di bit presenti sulle linee d'uscita della Porta A, i segmenti di un display. Se una delle linee di Porta A è ad 1 logico, è spento il corrispondente segmento; se è a 0, il segmento si accende. La Tabella 2 mostra quale formato di bit deve essere applicato alle linee della porta A per fare accendere i corrispondenti valori di numeri esadecimali sul display. Le denominazioni precise dei segmenti di un display a 7 segmenti sono indicate in fig. 10.

Tabella 2

PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0	esadecimale
	$\overline{g}$	$\overline{f}$	$\overline{e}$	$\overline{d}$	$\overline{c}$	$\overline{b}$	$\overline{a}$	
X	1	0	0	0	0	0	0	0
X	1	1	1	1	0	0	1	1
X	0	1	0	0	1	0	0	2
X	0	1	1	0	0	0	0	3
X	0	0	1	1	0	0	1	4
X	0	0	1	0	0	1	0	5
X	0	0	0	0	0	1	0	6
X	1	1	1	1	0	0	0	7
X	0	0	0	0	0	0	0	8
X	0	0	1	0	0	0	0	9
X	0	0	0	1	0	0	0	A
X	0	0	0	0	0	1	1	B
X	1	0	0	0	1	1	0	C
X	0	1	0	0	0	0	1	D
X	0	0	0	0	1	1	0	E
X	0	0	0	1	1	1	0	F

Quando vengono pilotati i segmenti, PA7 risulta definito quale ingresso. A tale linea non risulta collegato nulla. Le configurazioni di bit dei singoli segmenti di Tabella 2 sono poste in una Lookup Table nella EPROM dello Junior-Computer. I 16 numeri esadecimali di questa tabella si trovano agli indirizzi 1F0F ... 1F1E (vedi Source Listing in appendice al volume).

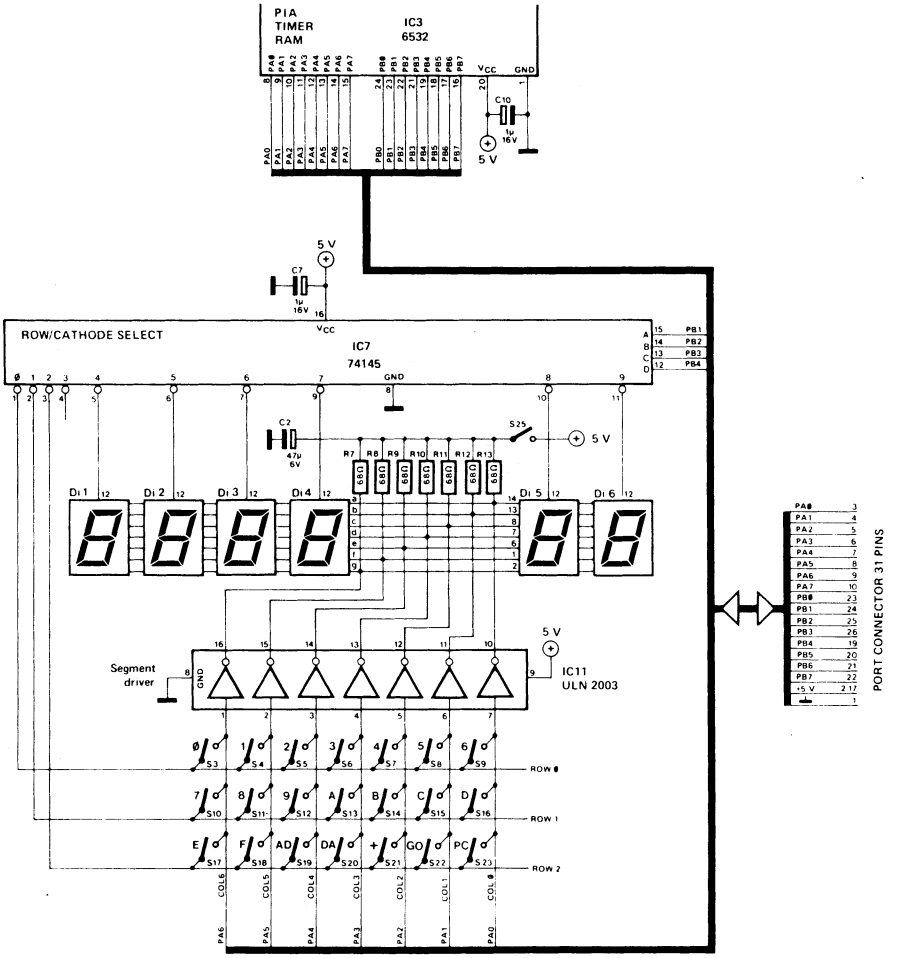


Figura 9. Estratto del circuito elettrico dello Junior Computer. Questi sono i collegamenti fra display, tastiera e connettore di porta ed il PIA (IC3).

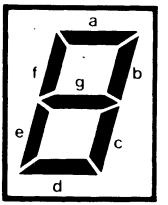


Figura 10. I segmenti di un display sono identificati con le lettere da a a g. Ad ogni segmento corrisponde una linea della Porta A. Per la presenza del pilota dei segmenti, IC11, le linee di porta vengono collegate invertite ai segmenti. Se una delle linee di Porta A è a livello logico 0 si accende il relativo segmento di display, mentre rimane spento se la linea è a livello 1.

Tasto premuto o non premuto?

In fig. 9 si vede che ogni tasto ha due collegamenti. Uno dei contatti è collegato ad uno degli output di IC7, l'altro con una delle linee di porta PA0 ... PA6. I tasti sono ordinati in una matrice, che consta di tre righe e sette colonne. I tasti di una stessa riga sono collegati in comune con un dato output di IC7. I tasti di una stessa colonna sono collegati in comune con una data linea di porta.
Per poter stabilire se è stato premuto un qualsiasi tasto, la Porta A è dichiarata input. Il tipo di configurazione di bit applicato alle linee PB1 ... PB4 fa sì che una delle uscite "0", "1" o "2" di IC7 passi allo stato logico 0. Corrispondentemente, la pressione di un tasto porta a 0 log. una delle linee di porta PA0...PA6.
In tal modo è possibile identificare in modo univoco un tasto:

1. La configurazione dei bit presenti sulle linee di porta PB1...PB4 determina la riga, in cui si trova il tasto premuto.
2. Altrettanto facile è determinare la colonna in cui è stato premuto il tasto: basta che il processore legga la Porta A, ad esempio nell'Accu. Sarà 0 il bit dell'Accu corrispondente alla colonna in cui è premuto il tasto.

La Tabella 3 e la Tabella 4 riassumono il modo in cui il computer può identificare il tasto premuto nella matrice di tastiera:

Tabella 3

PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0	Port B
			D	C	B	A		
X	X	X	0	0	0	0	X	Riga 0 con i tasti 0, 1, 2, 3, 4, 5 e 6
X	X	X	0	0	0	1	X	Riga 1 con i tasti 7, 8, 9, A, B, C e D
X	X	X	0	0	1	0	X	Riga 2 con i tasti E, F, AD, DA, +, GO e PC

Tabella 4

	PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0
Tasti 0, 7 e E	X	0	1	1	1	1	1	1
Tasti 1, 8 e F	X	1	0	1	1	1	1	1
Tasti 2, 9 e AD	X	1	1	0	1	1	1	1
Tasti 3, A e DA	X	1	1	1	0	1	1	1
Tasti 4, B e +	X	1	1	1	1	0	1	1
Tasti 5, C e GO	X	1	1	1	1	1	0	1
Tasti 6, D e PC	X	1	1	1	1	1	1	0

La Subroutine SCAND (meno AK)

Nel corso della Subroutine SCAND lo Junior-Computer pilota il display e sonda la tastiera. Prima di descrivere istruzione per istruzione questa subroutine, premettiamo una descrizione a bre-

vi tratti di quel che succede in questa subroutine. Da fig. 11 vediamo che la subroutine SCAND ha l'ausilio di due altre subroutine: esse si chiamano SHOW e CONVD.
Sul display devono venire riprodotti i contenuti di tre buffer di display: sono i due buffer indirizzi POINTH, POINTL ed il buffer dati INH. Questi tre buffer possono accogliere sei nibble dati e riempire così l'intero display di caratteri. Nel corso della subroutine CONVD uno dei sei display viene attivato per un tempo definito: in tal caso si accendono due o più segmenti. Nel corso della subroutine SHOW viene determinato quale nibble dati dei tre buffer POINTH, POINTL ed INH deve risultare illuminato sul display, ossia quale display debba essere attivato dopo il salto alla subroutine CONVD. Dato che ogni buffer di display contiene due nibble dati, si dovranno eseguire due salti da SHOW alla subroutine CONVD. I buffer da visualizzare sul display sono tre: perciò nel corso delle subroutine SCAND o SCANDS la subroutine SHOW dovrà essere richiamata tre volte. Nel corso della subroutine SCAND, quindi, tutti e sei i display si accendono uno dopo l'altro per un tempo fissato. I display vengono perciò pilotati in multiplex dal computer. Per lo Junior-Computer si parla dunque di un *display pilotato in multiplex dalla Software*. Sappiamo che lo Junior-Computer, sin quando nessun tasto viene premuto, si trova in un loop di attesa (vedi fig. 7, secondo richiamo della subroutine SCAND seguito da BEQ). Durante questo periodo il computer serve periodicamente i sei display.
La fig. 12 illustra le istruzioni della subroutine SCAND. Ci limitiamo dapprima al tratto di programma posto fra i Label SCAND e AK.
Si incomincia con rinfrescare il buffer dati INH. In questo buffer vengono deposti i dati delle locazioni di memoria indicate dal Pointer indirizzi POINTH, POINTL. (Durante l'Editing i tre buffer di display sono impiegati in modo diverso). Giungiamo così al Label SCANDS. Il computer definisce la Porta A quale output, e la utilizza quale commutatore dei segmenti. Caricando 08 nel registro X e trasferendo il contenuto di X alla Porta B (PB1...PB4 sono definite quali uscite) nel corso della subroutine CONVD, si pone il commutatore di display IC7 (volume 1°, capitolo 1, fig. 4) nello stato "Di1". Notiamo: il registro X tramite la Porta B governa il commutatore del display IC7. Il contenuto del registro X viene modificato nel corso delle subroutine SHOW e CONVD. Il registro X, perciò, indica sempre il display che dovrà successivamente essere illuminato.
Ora viene trasferito nel registro Y della CPU il contenuto di BYTES: esso determina se devono venire accesi due, quattro o tutti e sei i display. Dopo la pressione del tasto RTS in BYTES è posto il valore 3, il che vuol dire che tutti e tre i buffer di display POINTH, POINTL ed INH debbono venire visualizzati sul display. I due nibble dati di POINTH vengono ora trasferiti nell'Accu e si

effettua un salto alla subroutine SHOW. In successione, nel corso delle subroutine SHOW e CONVD i due nibble-dati di POINTH compaiono sui display Di1 e Di2 (vedi fig. 11). Dopo aver decrementato di 1 il registro Y, viene ancora richiamata la subroutine SHOW per visualizzare su Di3 e Di4 il contenuto di POINTL. Così è stato visualizzato sul display il Pointer indirizzi, POINTH, POINTL. Nuovamente si decrementa di 1 il registro Y e si visualiz-

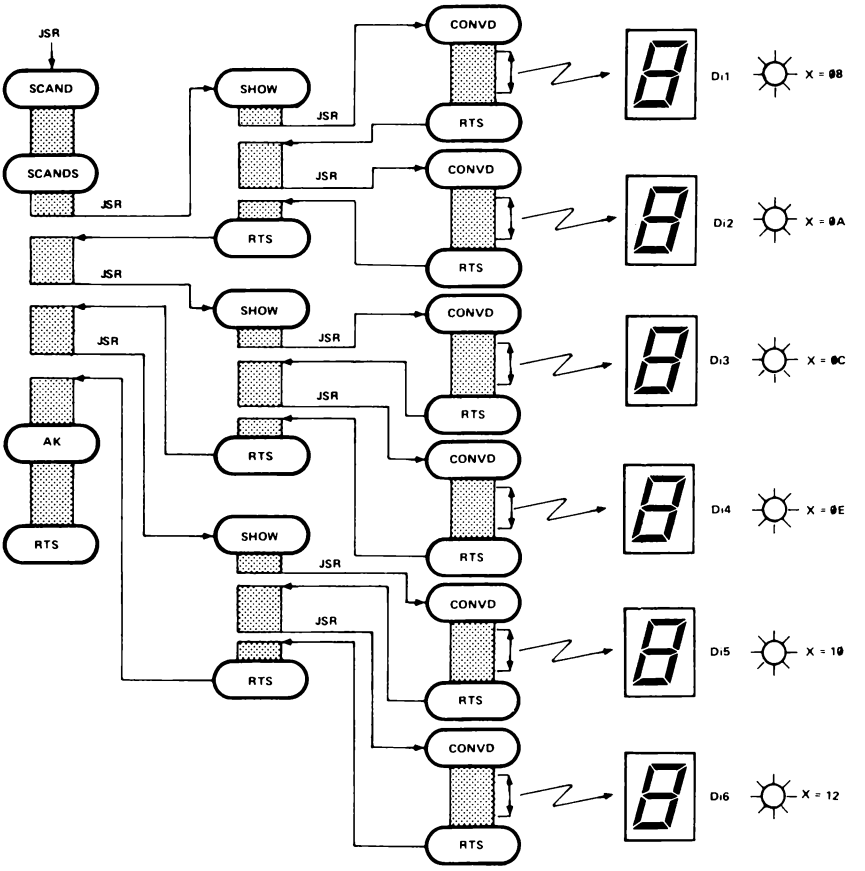


Figura 11. Nel corso delle subroutine SCAND o SCANDS lo Junior-Computer gestisce il display a 7 segmenti. Le due subroutine SHOW e CONVD sono "inscatolate" l'una entro l'altra (Subroutine Nesting). Dopo il Label SCANDS il computer determina se debba venire visualizzato il contenuto del buffer di display POINTH, POINTL od INH. Nella subroutine SHOW viene visualizzato sul display il contenuto del buffer di display selezionato. Nella subroutine CONVD un numero esadecimale viene convertito in una corrispondente configurazione dei 7 segmenti. Dato che devono venire visualizzati tre buffer, la subroutine CONVD viene richiamata 6 volte nel corso di SCANDS, dato che lo Junior-Computer ha 6 display. La subroutine AK è un'appendice delle subroutine SCAND/SCANDS. In questa sezione di programma il computer verifica se è stato effettivamente premuto qualche tasto.

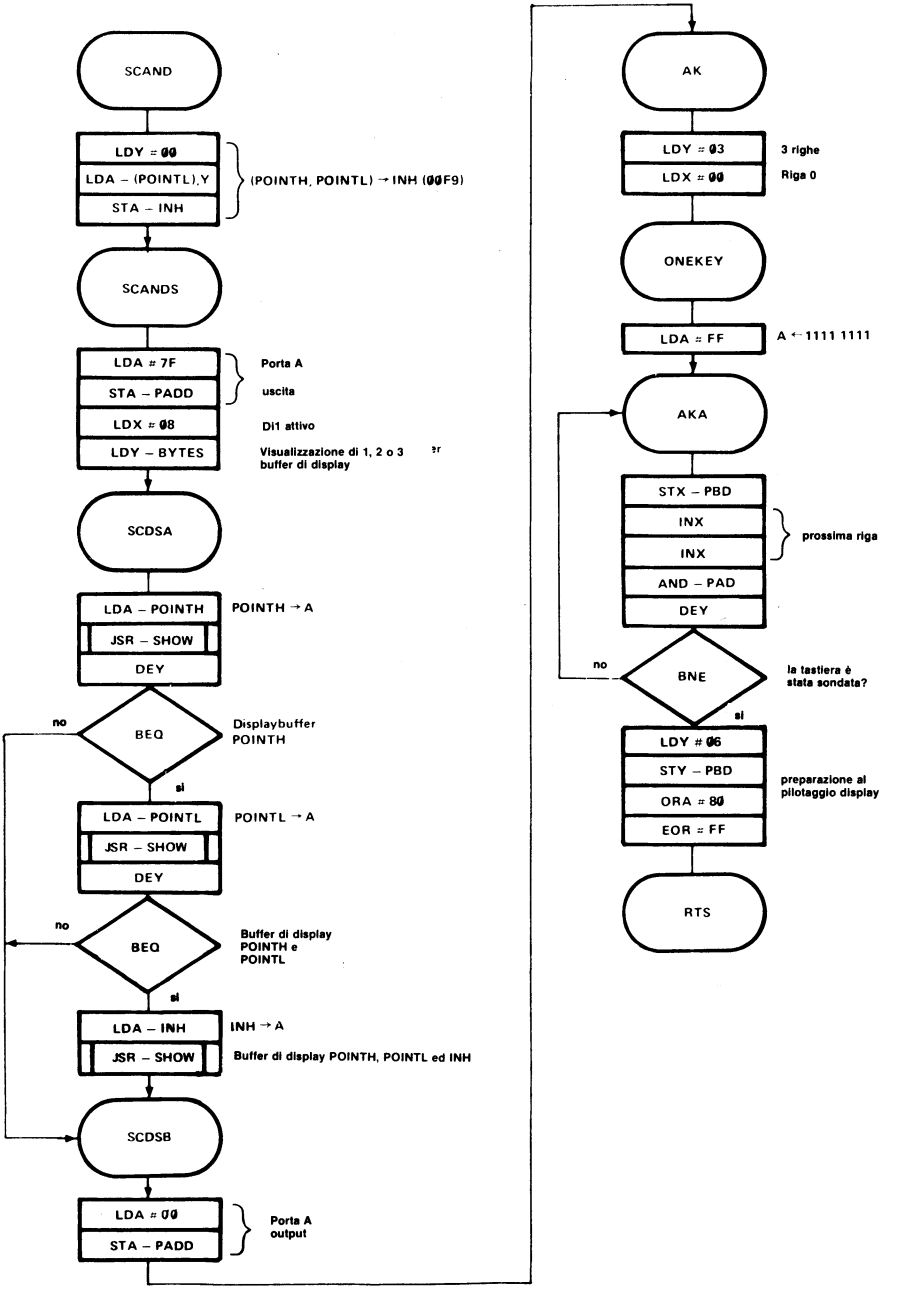


Figura 12. Diagramma di flusso dettagliato delle subroutine SCAND/SCANDS ed AK. Quest'ultima è una subroutine indipendente mediante la quale il programmatore può controllare se è stato premuto qualche tasto nella tastiera.

za il contenuto di INH sui display Di5 e Di6. Le due istruzioni seguenti definiscono la Porta A quale input, in preparazione alla subroutine AK, che tratteremo in un momento successivo.
A prima vista, l'impiego del registro Y quale contatore per i buffer può sembrare un lusso, dato che il numero dei buffer di display da visualizzare è ben noto. Ma la subroutine SCANDS viene utilizzata anche dall'Editor; e, come sappiamo, nell'Editing il display può avere una lunghezza variabile. Resta quindi giustificato l'uso in generale del registro Y come contatore dei buffer.

La Subroutine SHOW

La Subroutine SHOW è illustrata in fig. 13. Prima di richiamare questa subroutine, occorre che nell'Accu sia posto il contenuto del buffer di display da visualizzare sul display. Un byte consta, come sappiamo, di due Nibble. A ciascun nibble resta associato dunque un display a sette segmenti. Nel corso della subroutine SHOW per due volte viene effettuato un salto alla subroutine CONVD. La subroutine CONVD opera con il contenuto dell'Accu quale "input", il cui nibble dati alto deve risultare nullo. Il nibble basso dell'Accu è il valore del numero che deve essere visualizzato sul display a 7 segmenti. Dato che il funzionamento in multiplex pilota il display a 6 cifre da sinistra verso destra, viene sempre visualizzato per primo il nibble dati alto e dopo quello basso.

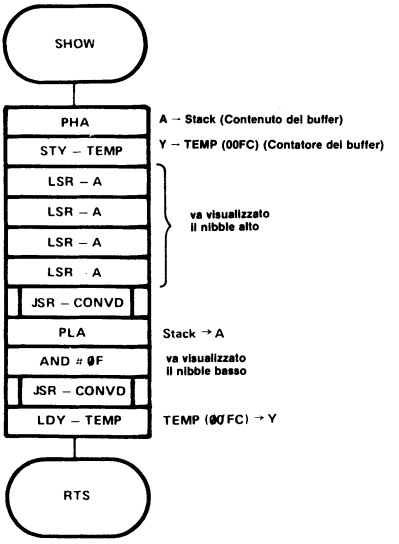


Figura 13. Diagramma di flusso dettagliato della subroutine SHOW. Questa subroutine deposita il nibble dato da visualizzare nell'Accu e successivamente, mediante la subroutine CONVD, lo presenta sul display. Il computer visualizza per primo il nibble alto e poi quello basso del dato presente nell'Accu.

Perciò all'inizio della subroutine SHOW è prevista una operazione di spostamento a destra ripetuto quattro volte, in cui il nibble basso è il numero da visualizzare ed il nibble alto viene automaticamente annullato (dopo 4 LSR-A ripetuti). Per far si che poi possa venire visualizzato anche il nibble dati basso del buffer di display, è necessario, prima dell'operazione di spostamento, conservare il contenuto del buffer di display: ciò è assicurato da un'istruzione PHA. Lo Stack viene qui usato come memoria intermedia. Bisogna inoltre salvare anche il contenuto del contatore dei buffer, perché il registro Y, durante la subroutine CONVD, deve restare libero per un altro impiego. Ora si richiama la subroutine CONVD, e si visualizza sul display il nibble dati alto. Nel corso della subroutine CONVD il computer converte il numero esadecimale da visualizzare in una corrispondente configurazione dei 7 segmenti. Diremo più avanti come ciò si realizzi.
Dopo il rientro dalla subroutine CONVD alla subroutine SHOW, viene ristabilito il contenuto originario dell'Accumulatore. Esso è a questo punto ancora costituito dall'intero contenuto dei buffer di display POINTH, POINTL ed INH da visualizzare. Ora si deve visualizzare il nibble dati basso: ecco quindi l'istruzione AND # OF. Un altro salto alla subroutine CONVD, ed il nibble dati basso viene visualizzato sul relativo display a 7 segmenti. Dopo il rientro dalla subroutine CONVD, l'istruzione LDY-TEMP ristabilisce il valore originario del contatore di buffer. Il computer può così occuparsi del seguente buffer di display da visualizzare.

La subroutine CONVD

Come già detto, questa subroutine conforma il nibble dati da visualizzare sul display a 7 segmenti. Un numero esadecimale deve venire trasformato in una corrispondente disposizione di segmenti illuminati. Prima di richiamare la subroutine CONVD, il nibble dati basso deve contenere la cifra esadecimale da visualizzare. In fig. 14 mostriamo la struttura della subroutine CONVD. Per prima cosa, il processore trasferisce il numero da rappresentare nel registro Y, che viene utilizzato quale registro indice. (TAY). Ora si carica indicizzato l'accumulatore, ossia ricaviamo da una Lookup Table la configurazione di segmenti del numero esadecimale da rappresentare. Se ad esempio si deve visualizzare la cifra esadecimale "D", l'istruzione LDA-LOOK, Y ricava il formato n° 21. Per convertire una cifra esadecimale in una configurazione di 7 segmenti sono quindi in sostanza necessarie solo:
TAY
LDA-LOOK, Y ed una Lookup-Table.
Il formato di segmenti passa ora dall'Accu alla Porta A, definita quale output. Il registro X, tramite il PIA, pilota il commutatore dei segmenti (decodifica): il contenuto del registro X viene perciò portato alla Porta B. Solo ora si illumina il corrispondente display

a 7 segmenti, riproducendo la cifra richiesta. Prossima domanda: quanto a lungo resta acceso il display? Questo compito è affidato al loop DELAY-BPE-DELAY. Il processore rimane in questa sequenza sinché resta ad 1 il Flag N. In tal modo, il display attivo al momento resta acceso per qualcosa più di 630 μs. Quando si esce dal loop DELAY-BPE-DELAY il contenuto del registro Y è FF: il computer deposita questi bit sulle linee della Porta A, disinserendo così tutti i segmenti dei display. L'istruzione relativa è STY-PAD. Poi, con le istruzioni LDY # 06 e STY-PBD, si pone il commutatore di display nello stato neutro. A questo punto risulta attivo l'output di IC7 non collegato (volume 1°, capitolo 1, fig. 4), e nessuno dei catodi dei display è posto a massa. Al termine della subroutine CONVD il contenuto del registro X, incrementato due volte, è divenuto 08. Ad un nuovo richiamo della subroutine CONVD il commutatore di display viene resettato con una istruzione STX-PBD (Di1 si illumina), ed il processo descritto si ripete.

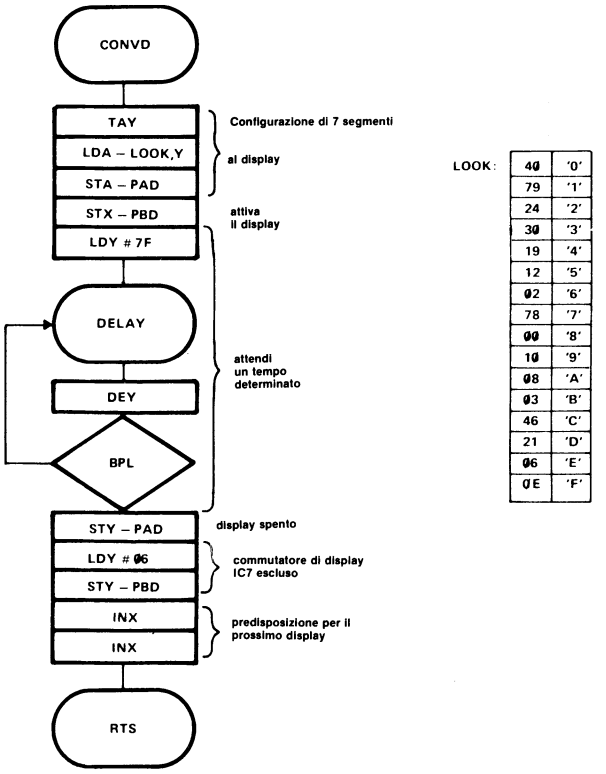


Figura 14. Diagramma di flusso dettagliato della subroutine CONVD. In questa subroutine il computer converte il numero esadecimale da visualizzare nel codice a 7 segmenti. Successivamente il corrispondente display viene inserito per circa 600 μs, dopo di che il computer passa ad occuparsi del prossimo display.

Prima del richiamo della subroutine CONVD bisogna curare che il nibble dati alto presente nell'Accu sia zero. Altrimenti il contenuto del registro Y può risultare maggiore di 0F e potrebbe venir superato l'indirizzo massimo della Lookup Table. Dato che dopo l'indirizzo 1F1E (fine della Lookup Table) esistono degli altri byte, sul display potrebbero venire accesi segmenti arbitrari visualizzanti una cifra errata.

AK, l'appendice delle subroutine SCAND e SCANDS

La subroutine AK si collega al termine delle subroutine SCAND e SCANDS, ossia, le 3 subroutine SCAND, SCANDS ed AK sono richiamate con lo stesso comando RTS. La subroutine AK ha il compito di rivelare la presenta di un qualsiasi tasto premuto nella tastiera. Per chiarire il funzionamento di AK, consideriamo le fig. 9 e 16. La configurazione di bit applicata alle linee della Porta B stabilisce da un lato quale display risulta inserito e dall'altro quale riga della tastiera deve essere sondata. Il registro X provvede al governo della Porta B.
Come sappiamo già dal 1° volume, al termine delle subroutine SCAND, SCANDS nonché AK il contenuto dell'Accu risulta diverso da 0 se il computer ha rivelato un tasto premuto in una riga qualsiasi della tastiera. Come avviene ciò?
All'inizio della subroutine AK, tutti i bit sulle linee della Porta A sono1, dato che tali linee sono portate "alte" tramite resistenze di pull-up interne. Poi si carica 03 nel registro Y, che in AK funziona da contatore di riga. Nello Junior-Computer le righe da verificare sono tre: ROW0, ROW1 e ROW2. È il formato di bit del registro X a determinare quale di queste tre righe viene selezionata tramite la Porta B. Dopo il Label ONEKEY si carica FF nell'Accu. Successivamente, il contenuto del registro X (X=00) viene portato alla Porta B, e la riga ROW0 viene posta a massa (vedi Tabella 3). Il doppio incremento del registro X che segue costituisce una preparazione per la riga successiva. Viene poi eseguita un'operazione AND fra l'Accu ed il formato di bit sulle linee di Porta A. Dopo aver percorso per tre volte il loop AKA-BNE-AKA, il numero di zeri nell'Accu corrisponde ai tasti (contemporaneamente) premuti in una qualsiasi colonna (COL0 ... COL6). Se ad es. in COL3 è stato premuto uno dei tasti 3, A o DA, dopo la relazione AND il bit "b3" dell'Accu vale 0.
Usciti dal loop, il commutatore di display e riga (IC7 sulla piastra base dello Junior-Computer) viene nuovamente posto allo stato neutro. Le Tabelle 1 e 3 chiariscono ulteriormente la cosa. L'istruzione ORA # 80 rende b7 nell'Accu eguale ad 1. La cosa è importante, perché alla Porta A nel modello standard dello Junior-Computer possono risultare collegate delle unità periferiche, ad esempio una stampante; ed il valore di b7 in corrispondenza all'istruzione ORA potrebbe modificarsi nel tempo. Così è tutto

pronto per la seguente istruzione EOR: con essa vengono invertiti tutti i bit dell'Accu (EOR # FF). Come sappiamo, un bit dell'Accu è zero se durante il ciclo di programma AKA-BNE-AKA è stato premuto un qualche tasto. Pertanto, dopo l'istruzione EOR il contenuto dell'Accu è diverso da 0, se è stato premuto un tasto; eguale a 0, se non è stato premuto alcun tasto. Così abbiamo descritto anche la subroutine AK. Essa serve solo a riconoscere se un qualche tasto della tastiera è stato premuto. Il modo con cui i computer calcola il valore del tasto premuto è descritto nella successiva sezione.

La Subroutine GETKEY

Dopo che il computer ha localizzato un tasto premuto nella tastiera, deve ancora calcolarne il valore. La subroutine GETKEY calcola questo valore del tasto. In fig. 16 è ancora mostrato come i singoli tasti sono ordinati nella matrice di tastiera. Sono anche riportati i valori dei tasti e relativi simboli. Le singole istruzioni della subroutine GETKEY sono elencate nelle fig. 15a e 15b. In fig. 15b, per maggiore chiarezza, è stato inserito pure il tratto di programma ONEKEY da fig. 12.
Dapprima si carica il numero esadecimale 21 nel registro X: ciò fa sì che al susseguente salto alla subroutine ONEKEY sia la riga ROW0 di tasti a venire verificata. Il caricamento di 01 nel registro Y significa che nel corso di ONEKEY deve venire trattata una sola riga. Dopo il rientro da ONEKEY, il contenuto dell'Accu risulta diverso da 0 se è stato premuto un tasto nellà riga ROW0. Se invece in questa riga non è stato premuto alcun tasto, il valore nell'Accu è zero. Con la seguente istruzione BNE il programma salta al Label KEYIN, se nel corso di ONEKEY è risultato premuto un tasto. In caso contrario, col contenuto del registro X diverso da 27, il programma salta al Label GETKEA e viene ancora richiamata la subroutine ONEKEY.
Che significa il confronto del contenuto del registro X con il valore 27? All'inizio di GETKEY X vale 21, e risulta quindi attivata la riga di tasti ROW0. Nel corso di ONEKEY il registro X viene incrementato per due volte di 1, per cui il suo valore è ora di 23. 23 risulta diverso dà 27, e si ripassa ancora una volta per la subroutine ONEKEY. In questo caso però non è attiva ROW0, ma la riga di tasti ROW1. Al rientro da ONEKEY il valore del registro X è 25. Se anche nella riga ROW1 non risulta premuto alcun tasto, il processore ripercorre ancora una volta la subroutine ONEKEY: dopo il terzo passaggio il valore di X è salito a 27.
Ora sono state sondate tutte le righe di tasti. Se dopo aver percorso tre volte la subroutine ONEKEY ancora non risulta identificato un tasto premuto, c'é qualche errore. Il computer non deve effettuare il salto al Label KEYIN ed iniziare il calcolo del valore del tasto. Perciò, dopo l'istruzione CPX # 27, si carica 15 nell'Accu,

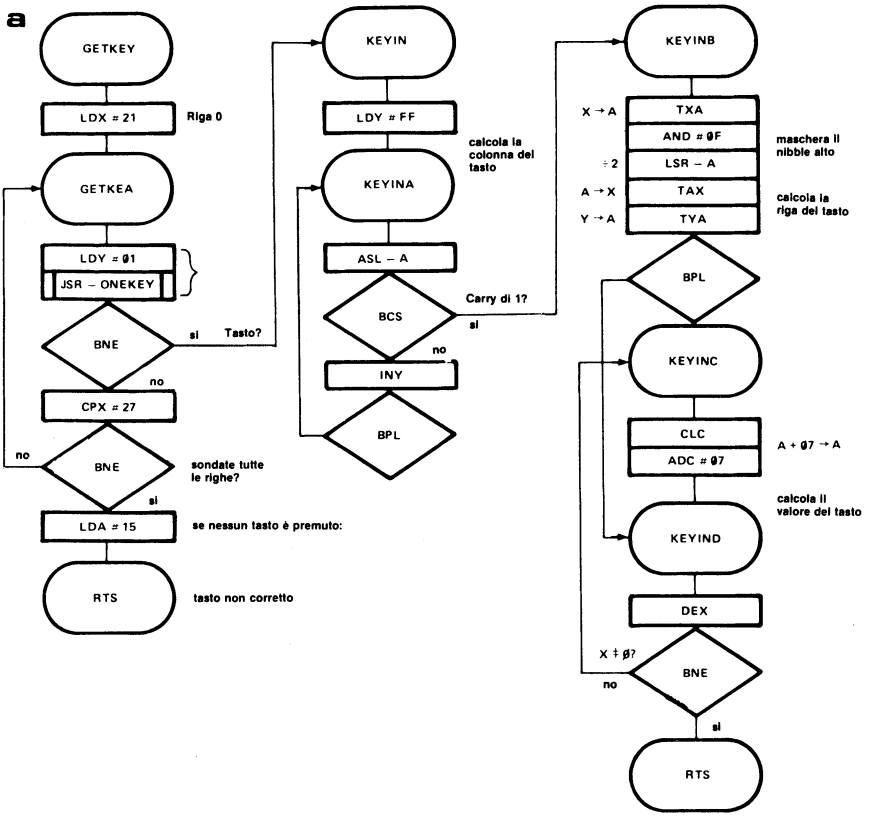
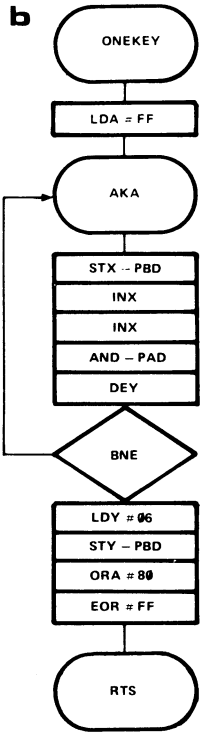


Figura 15. Nella subroutine GETKEY il computer calcola il valore d'un tasto premuto. Per il sondaggio delle ROW (righe) viene utilizzata la subroutine ONE-KEY, che è una parte della subroutine AK.

comunicando così al computer che deve ignorare il tasto premuto e ricominciare il sondaggio della tastiera. Se il computer rientra dalla subroutine GETKEY con un valore dell'Accu di 15, si tratta di un "falso tasto".
Giungiamo così ancora al Label KEYIN. A partire da questo Label, il computer calcola il valore del tasto premuto. Rivediamo la fig. 16, che riporta le coordinate X ed Y dei singoli tasti di tastiera. Al principio di KEYIN si carica FF nel registro Y. Il computer provvede poi a spostare il contenuto dell'Accu almeno due volte a sinistra. Prima di questo spostamento l'Accu conteneva sette 0 ed un 1. Il computer esegue una serie di spostamenti verso sinistra sin quando nel Flag C compare 1. Il contenuto del registro Y, dopo le operazioni di spostamento (BCS), indica il posto del tasto che era premuto. Il valore Y, cioé, informa in quale colonna COL0 ... COL6) si trova il tasto premuto. Si veda al proposito fig. 16. Per i



tasti situati nella riga ROW0 il valore Y dopo le operazioni di spostamento dà direttamente il valore del tasto. Per un tasto in ROW1 occorre aggiungere 07, per un tasto in riga ROW2 occorre sommare 0E (il valore esadecimale 0E corrisponde a due volte 07). Di queste addizioni, quando sono eventualmente richieste, si occupa la parte di programma denominato KEYINB in fig. 15a. Prima ancora, comunque, che il programma sia progredito a questo punto, il contenuto del registro X, cioé l'informazione sul numero di riga, viene portato nell'Accu. Il valore di X è 23 per un tasto della riga ROW0, 25 per un tasto di ROW1 e 27 per un tasto di ROW2. È facile verificare che prima del Label KEYINC il valore del registro X è 01, se si tratta di un tasto premuto nella riga ROW0; 02, se il tasto premuto è nella riga ROW1, e 03, se nella riga ROW2. A partire dal Label KEYINC viene pertanto sommato 1 o 2 volte il valore 07 al numero di colonna del tasto, a seconda della riga in cui era il tasto premuto. Il ciclo di programma KEYINC-BNE-KEYINC non viene invece percorso affatto nel caso di tasti della riga ROW0, dato che dopo il Label KEYIND a causa dell'istruzione DEX il registro X ha già valore zero.
Al rientro dalla subroutine GETKEY il contenuto dell'Accu è il valore del tasto premuto.

Per chiarire ulteriormente la subroutine GETKEY, vediamo cosa avviene quando sulla tastiera dello Junior-Computer è stato premuto il tasto "C". Inizialmente, il valore del registro X è 21, ed il valore del registro Y 01. Segue un salto alla subroutine ONEKEY. Dopo aver caricato X=21 sulle linee di Porta B, la serie di tasti che si trova in ROW0 ha uno dei collegamenti a massa. Il tasto C non appartiene tuttavia alla riga ROW0, ma è fra i tasti di ROW1. Alla lettura dei tasti in ROW0 si ha la seguente situazione:

```
X1111111   Porta A
11111111   Accu = FF
X1111111   Relazione AND con l'Accu= FF
11111111   Relazione OR con 80
00000000   Relazione OR con FF
```

Dato che il contenuto dell'Accu è zero, il programma non effettua il salto al Label KEYIN. Nel corso della subroutine ONEKEY anche il contento del registro X si è modificato:

```
00100001 = 21 (per ROW0)
00100010 = dopo INX
00100011 = dopo INX (per ROW1)
```

Al rientro dalla subroutine ONEKEY, l'istruzione BNE reagisce al contenuto del registro modificato per ultimo o all'ultima modifica del Flag Z. L'istruzione che modifica per ultima il Flag Z è EOR # FF. Dato che risulta premuto il tasto "C", ed il Flag Z vale 1, non ha luogo il salto al Label KEYIN. Il computer verifica poi se il valore

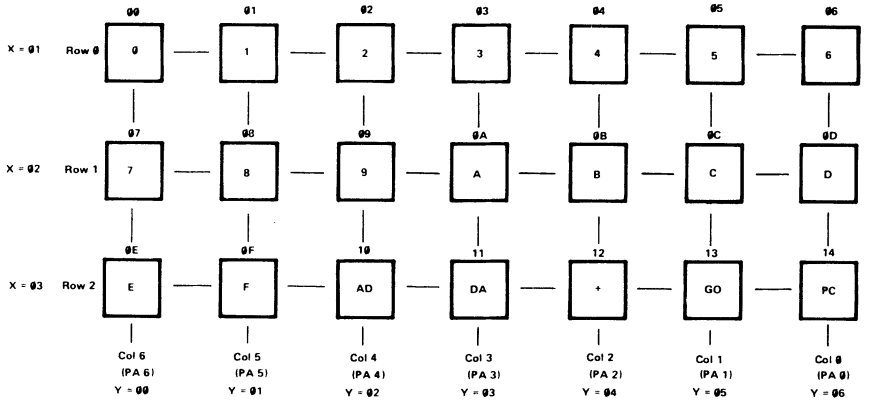


Figura 16. Questo è l'aspetto della matrice di tasti dello Junior-Computer. Ad ogni tasto è assegnata una coordinata X ed una Y. Tuttavia il registro X contiene la coordinata Y, ed il registro Y la coordinata X del tasto. I tasti sono disposti in matrice di 7 x 3.

del registro X è già di 27. Ciò non è ancora verificato, perché, dopo aver percorso ONEKEY soltanto una volta, il valore di X è attualmente solo di 23. Pertanto il programma salta al Label GETKEA; il registro Y viene ancora posto eguale ad 1, e si salta alla subroutine ONEKEY. Nuovamente si carica FF nell'Accu e si deposita sulla Porta B il nuovo contenuto del registro X. Un contatto dei tasti ordinati nella riga ROW1 è ora a potenziale di massa. Il tasto "C" si trova in ROW1, ed il formato di bit alla Porta A è il seguente:

X1111101 PA1 = colonna 1 (v. fig. 16) = tasto "C"

mentre, dopo la relazione AND con PAD, il contenuto dell'Accu è

X1111101

Dopo la relazione OR con FF il contenuto dell'Accu diviene

11111101 La successiva istruzione EOR # FF dà in Accu
00000010 mentre il registro X nel frattempo è passato a

25. Al rientro dalla subroutine ONEYKEY incontriamo una nuova istruzione BNE. Dato che a questo punto il contenuto dell'Accu (00000010) è diverso da zero, si ha il salto al Label KEYIN. Il contenuto dell'Accu viene gradualmente spostato verso sinistra sino a che il Flag C assume il valore 1:

Ø	ØØØØØØ1Ø	FF	Inizio
Ø	ØØØØ1ØØ	FF	dopo il 1° ASL-A
Ø	ØØØ1ØØØ	ØØ	dopo il 2° ASL-A
Ø	ØØ1ØØØØ	Ø1	dopo il 3° ASL-A
Ø	Ø1ØØØØØ	Ø2	dopo il 4° ASL-A
Ø	1ØØØØØØ	Ø3	dopo il 5° ASL-A
Ø	ØØØØØØØ	Ø4	dopo il 6° ASL-A
1	ØØØØØØØ	Ø5	dopo il 7° ASL-A
↑		↑	
Flag C		Registro Y dopo istr. ASL-A e prima di BCS	

Dopo 7 spostamenti successivi il Flag C è diventato 1, ed il programma salta al Label KEYINB. Dato che è attivo il tasto "C", il contenuto del registro X è eguale a 23 e quello di Y a 5. Quindi, viene copiato nell'Accu il contenuto del registro X:

ØØ1ØØ1Ø1	Accu prima della relazione AND
ØØØØØ1Ø1	Accu dopo l'AND con 0F
ØØØØØØ1Ø	Accu dopo LSR-A (= 02)

Il contenuto dell'Accu viene ritrasferito nel registro X (TAX). A questo punto il valore di X è 02. Ora si copia il registro Y nell'Accu (TYA): il valore dell'Accu è 05, che non è negativo. Perciò il programma salta al Label KEYIND, decrementando di 1 il registro

X. Il nuovo contenuto del registro X è 1: valore diverso da zero, per cui il programma salta al Label KEYINC, e quindi, in funzione del contenuto del registro X, si somma 07 oppure 0E al numero di colonna del tasto. Rammentiamo: il contenuto dell'Accu è stato spostato 5 volte verso sinistra, prima che il Flag C passasse ad 1. Il tasto "C" ha quindi il valore provvisorio di 5. Il tasto "C" si trova nella riga ROW1 della matrice di tastiera (fig. 16).
Al valore provvisorio 05, dopo il Label KEYINC, si somma 07, ottenendo: 05 + 07 = 0C. Questo è già il valore finale del valore del tasto "C". Dopo la somma si decrementata di 1 il registro X, che nel nostro esempio diventa quindi 0. Nell'Accu della CPU si trava quindi, al rientro nel programma principale, il valore corretto del tasto premuto.
Così si conclude il capitolo dedicato al Progrmma Monitor.
Dalle 1024 locazioni di EPROM, che da esso sono occupate quasi per intero, abbiamo tratto importanti sezioni di programma. La routine principale del Monitor occupa 181 byte, e 152 sono occupati dalla Subroutine del Monitor. Le routine per il sondaggio della tastiera e per la gestione del display, inoltre, saranno da noi impiegate in successivo capitolo, quando descriveremo la Software dell'Editor.

8

Il Programma Editor

Questo capitolo descrive il Programma Editor. Cosa succede nello Junior-Computer, quando si introducono dati in Editor-Mode? Cosa avviene nella EPROM quando viene premuto uno dei tasti-comandi INSERT, INPUT, DELETE ecc.? In che modo provvede l'Editor a depositare nella memoria di lavoro dello Junior-Computer un programma impostato, per la successiva elaborazione da parte dell'Assembler? Questo capitolo risponde a tutte queste domande, sulla base di un diagramma di flusso dettagliato nei singoli byte.

Come ben sappiamo, la EPROM contiene tre grossi programmi: il Monitor per la visualizzazione degli indirizzi e dei dati; l'Editor e l'Assembler. Tra questi programmi è l'Editor ad impegnare il maggior spazio di memoria. Già conosciamo l'Editor dal capitolo 5 come efficace ausilio per l'introduzione di grossi programmi. Qui faremo seguire la descrizione e spiegazione del programma Editor, che consta di diverse subroutine. Queste subroutine sono pure disponibili al programmatore per l'inserimento in propri programmi, risparmiando così molto tempo nella stesura dei programmi stessi. Una di tali subroutine ci è già nota dal 1° volume: la subroutine GETBYT, ovvero: lettura di due valori dei tasti nell'Accu della CPU.
L'Editor opera insieme ai cinque tasti comando SEARCH, INSERT, INPUT, SKIP e DELETE, nonché i tasti dati 0 ... F. La pressione dei comandi SEARCH, INSERT od INPUT fa sì che il computer si attenda l'attivazione di due, quattro o sei tasti dati. Ciò dipende dalla lunghezza delle istruzioni della CPU 6502: questo processore prevede istruzioni della lunghezza di uno, due o tre byte. Nel caso del comando SEARCH lo Junior-Computer si attende sempre quattro tasti dati, perché dovrà effettuare la ricerca di un dato formato di due byte nella memoria.
Prima di descrivere il programma principale dell'Editor e le sue subroutine, presentiamo alcune particolarità dell'Editor sulle

quali non si era entrati nei dettagli nel capitolo 5. Il capitolo 5, in sostanza, insegnava come usare l'Editor. Il capitolo che segue descrive il modo in cui il processore procede nella EPROM durante l'Editor.

Caratteristiche dell'Editor

Prima di poter lavorare tramite l'Editor, bisogna procedere a depositare nelle celle di memoria BEGADH, BEGADL e ENDADH, ENDADL gli indirizzi di inizio e fine del campo di memoria in cui l'Editor dovrà collocare il programma. È anche possibile introdurre istruzioni nel computer mediante l'Editor, senza che esse vengano successivamente assemblate. In tal caso tuttavia il programmatore non deve introdurre dei Label, dato che la CPU 6502 non potrà far niente coll'indicatore di Label FF. Occorre quindi sempre impiegare L'Assembler dopo l'Editor, se nella zona di memoria previamente definita si sono introdotti dei Label. L'Assembler provvede, com'é noto, ad eliminare da un programma tutti i Label che sono stati in precedenza impostati tramite l'Editor. Un altro pseudocomando della CPU 6502 viene utilizzato nel corso dell'Editor: il carattere 77 con valore di EOF (= End Of File = fine pagina). Il carattere EOF viene posto alla fine di tutte le istruzioni impostate. Come sappiamo dal capitolo 5,al termine di un programma impostato fra il carattere EOF e l'indirizzo di fine ENDADH, ENDADL devono trovarsi ancora almeno 6 locazioni libere. In caso diverso sorgono grossi problemi al successivo lancio del programma Assembler: la Symbol-Table che viene allestita dall'Assembler finisce per sovrascrivere e cancellare la parte terminale del programma impostato (vedi capitolo 9).
Una volta lanciato l'Editor, mediante i tasti comando INSERT ed INPUT possiamo inserire altre istruzioni nel computer. Dopo l'introduzione di una istruzione, lo spazio di memoria occupato si incrementa di una, due o tre locazioni. Quanto sia questo incremento di spazio di memoria viene determinato dal codice OP dell'istruzione nuova introdotta. L'incremento di spazio di memoria per l'introduzione di una nuova istruzione provoca pure uno spostamento del carattere EOF 77. Se ad esempio viene introdotta una istruzione lunga due byte, il carattere EOF risulta spostato di due posizioni di memoria verso il basso.
Quando si preme il tasto DELETE, succede il contrario. La pressione di questo elimina dalla memoria dello Junior-Computer l'istruzione al momento visualizzata sul display. La conseguente lacuna viene colmata dal computer spostando verso l'alto di uno, due o tre byte secondo necessità l'intero blocco di istruzioni successive. Alla nuova situazione viene pure adattata la posizione dei caratteri EOF: il computer sposta analogamente questo carattere di uno, due o tre byte verso l'alto.
Col comando SEARCH si possono ricercare nella memoria dello

Junior-Computer determinati formati di due byte. Questi possono essere un codice OP + primo byte dell'operando, oppure un Label col pseudocodice OP FF + numero di Label. La ricerca parte dall'indirizzo iniziale BEGADH, BEGADL e termina all'indirizzo al quale è situato il codice OP del formato di due byte oggetto di ricerca.
Un altro tipo di comando di ricerca è SKIP. Esso identifica nella memoria l'istruzione che segue immediatamente a quella visualizzata al momento, e la presenta sul display. Sia il comando SEARCH che SKIP possono emettere un messaggio di errore. Se ad esempio mediante il comando SEARCH viene ricercato un formato di byte che non si trova in memoria, il display indica EEEEEE sin quando non viene lasciato libero il tasto SEARCH. Rilasciato questo tasto, sul display compare una data istruzione: è quella alla quale il computer ha interrotto l'operazione di ricerca. Anche per SKIP è possibile la segnalazione di errore: infatti al termine di un blocco dati consistente di istruzioni e Label è presente il carattere EOF. Se l'utente, con l'applicazione ripetuta del comando SKIP, giunge ad una istruzione successiva al carattere EOF, sul display compare analogamente EEEEEE sino a quando viene mantenuto premuto il tasto SKIP.

Il Pointer indirizzi dell'Editor

L'Editor depone i dati in un campo di memoria che principia a BEGAD e termina a ENDAD. La zona di memoria fra BEGAD ed ENDAD si chiama "File" (= pagina o blocco dati). L'Editor richiede quattro Pointer indirizzi per la gestione del File; essi sono:
1. BEGAD: questo Pointer indirizzi è posto nelle celle di memoria BEGADL, indirizzo 00E2, e BEGADH, indirizzo 00E3. BEGAD fissa l'indirizzo iniziale del file in cui devono venire deposti dati.
2. ENDAD: questo Pointer indirizzi è posto nelle locazioni ENDADL, indirizzo 00E4, e ENDADH, indirizzo 00E5. ENDAD fissa l'indirizzo finale del file in cui devono venire deposti dati dall'Editor.
3. CURAD: questo Pointer indirizzi è posto nelle locazioni CURADL, indirizzo 00E6, e CURADH, indirizzo 00E7. CURAD è un'abbreviazione dell'inglese "current address", cioé indirizzo momentaneo. L'Editor necessita del Pointer indirizzi CURAD per gestire il display a 6 cifre sulla piastra base dello Junior-Computer. Dato che il display indica tutte le istruzioni del file, nel corso dell'editing il Pointer indirizzi CURAD cambia continuamente valore. Il programma Editor aggiorna il Pointer CURAD in modo che esso indichi sempre il codice OP dell'istruzione al momento visualizzata sul display.
4. CEND: questo Pointer indirizzi è posto nelle locazioni CENDL, l'indirizzo 00E8, e CENDH, indirizzo 00E9. CEND è un'abbreviazione dell'inglese "current end address", ossia indirizzo terminale

momentaneo. Il Pointer CEND si modifica, così come CURAD, nel corso dell'editing, in modo continuo. CEND viene modificato quando l'utente utilizza i tasti INSERT, INPUT o DELETE per inserire o eliminare istruzioni nel computer. Il Pointer indirizzi CEND viene aggiornato dall'Editor in modo che esso indichi sempre la locazione di memoria immediatamente successiva al carattere EOF 77. Se ad es. il carattere EOF è all'indirizzo 0259, CEND indica 025A.
Il Pointer di display CURAD ed il Pointer di indirizzo finale variabile CEND vengono fissati alla partenza dell'Editor (Cold Start Entry, vedi capitolo 5) in un determinato stato iniziale. Nel caso di Cold Start Entry CURAD indica l'indirizzo di principio del file. Il computer pone quindi a tale indirizzo iniziale del file il carattere 77 di EOF: è questo il motivo per cui dopo un Cold Start Entry vediamo il carattere EOF sul display. Poi, il Pointer CEND si dispone in modo da indicare la locazione di memoria immediatamente dopo il carattere EOF. Dato che non si sono ancora introdotte istruzioni, CEND indica attualmente l'indirizzo BEGAD + 1. La fig. 2a presenta questa situazione iniziale.
Oltre al Cold Start Entry, nell'Editor esiste anche la possibilità di un Warm Start Entry. In questo secondo caso l'Editor inizia il suo lavoro come al solito, però il Pointer di display CURAD, il Pointer indirizzi CEND ed il carattere EOF *non* subiscono modifiche. Come detto nel capitolo 5, l'Editor può essere avviato sia premendo il tasto GO che premendo il tasto ST. L'indirizzo di partenza per il Cold Start Entry è 1CD5. Questo è pure l'indirizzo che deve essere indicato dal vettore NMI, quando si fa partire l'Editor con il tasto ST. Lo stesso vale naturalmente per il Warm Start Entry, con indirizzo di partenza 1CCA. L'uscita dall'Editor può avvenire tramite il tasto RST oppure il tasto ST. Se si vuole uscire dall'Editor con il tasto ST, occorre previamente disporre il vettore NMI (vedi capitolo 5). Durante l'editing i buffer di display non contengono più indirizzi e dati, bensì istruzioni visualizzate dal computer sul display a sei cifre. I buffer di display hanno anche nell'Editor le denominazioni POINTH, POINTL ed INH. POINTH contiene i dati relativi ai display Di1 e Di2, POINTL i dati di Di3 e Di4, ed INH i dati di Di5 e Di6. I compiti dei buffer di display differiscono comunque da quelli descritti nel capitolo 7:

* Introducendo i dati il display si riempie da sinistra a destra.
* Il computer legge i dati byte per byte. Occorre quindi che siano premuti due tasti prima che il relativo byte sia visualizzato sul display.
* Nell'introduzione di un'istruzione, il computer legge per primo il codice OP, e calcola la lunghezza della relativa istruzione. Di conseguenza, risulta noto se l'istruzione da visualizzare sul display è lunga uno, due o tre byte.

Il computer riconosce quindi, all'introduzione del codice OP, se

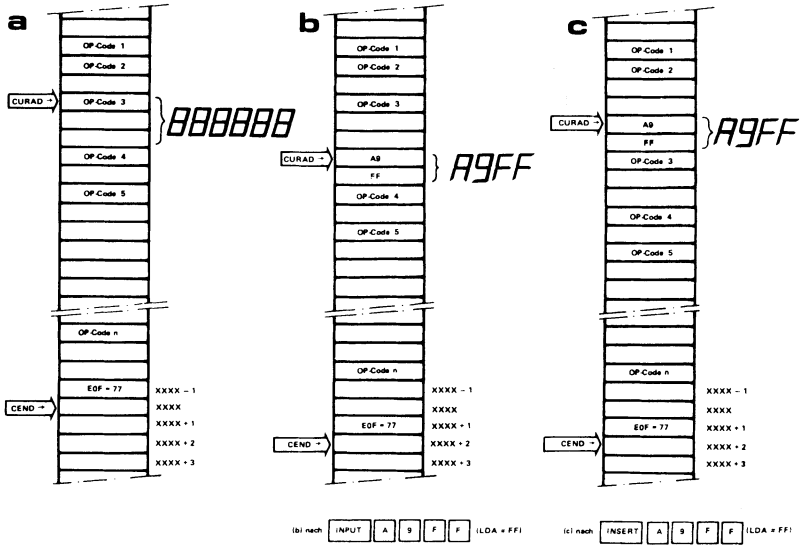


Figura 1. La fig. 1a ci mostra una sezione del file introdotto dal programmatore nello Junior-Computer con l'ausilio dell'Editor. In ogni posizione del file possono, se richiesto, venire inserite altre istruzioni.
Le fig. 1b e 1c mostrano la differenza fra il comando INPUT e quello INSERT. In entrambi, i casi si suppone di inserire l'istruzione LDA # FF ovvero A9 FF nel file già costituito. Il comando INPUT pone l'istruzione LDA # FF dietro l'istruzione col codice OP③. Invece il comando INSERT dispone l'istruzione LDA # FF prima di quella col codice OP③. Dopo un comando INPUT od INSERT lo Junior-Computer visualizza l'istruzione inserita sul display.

dovranno accendersi i display Di1...Di2, Di3...Di4 o Di5...Di6. Nel corso dell'Editor i buffer di display hanno le seguenti funzioni:
POINTH: contiene sempre il codice OP di una istruzione, il pseudo-codice OP di un Label od il carattere EOF.
POINTL: contiene sempre il primo byte di un operando (se presente) oppure un numero di Label.
INH: contiene sempre il secondo byte di un operando (se presente) o il limitatore di un Label.
Prima di addentrarci ulteriormente nella Software dell'Editor, ancora un paio di osservazioni sui tasti comando INSERT, INPUT e DELETE. Consideriamo la fig. 1, che è suddivisa in tre parti.
* La Fig. 1a ci mostra la situazione iniziale. Il codice OP③ed i due byte dell'operando seguente sono visualizzati sul display. Il Pointer indirizzi CURAD indica il codice OP③. Il carattere EOF si trova all'indirizzo XXXX-1 e il contenuto del Pointer CEND è XXXX.
* Fig. 1b: il programmatore intende inserire una nuova istruzione dopo quella visualizzata al momento: LDA#FF. A tale scopo utilizza il tasto comando INPUT. La nuova istruzione, LDA#FF, viene

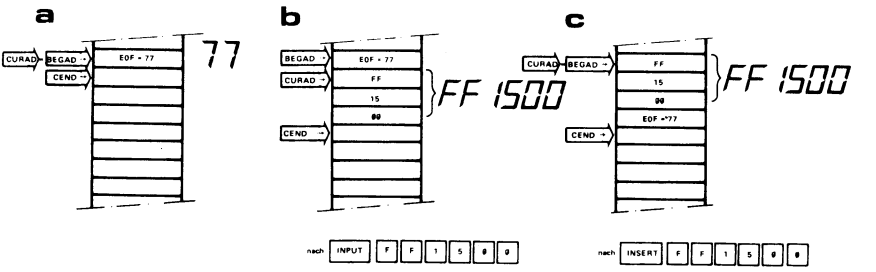


Figura 2. La fig. 2a presenta la situazione iniziale che si verifica quando l'Editor viene fatto partire con Cold Start Entry. Dato che CURAD = BEGAD, il primo a essere visualizzato sul display è il carattere EOF. Se inizialmente si è inserita l'istruzione LDA # FF col comando INPUT, questa istruzione sarebbe posta dopo il carattere EOF. Ciò non è ammesso, perché l'Editor ad ogni successiva introduzione di dati darebbe una segnalazione di errore. La fig. 2b mostra questa situazione non corretta.
In fig. 2c l'inserzione del dato avviene mediante il tasto INSERT. Il Label FF 15 00 viene così posto prima del carattere EOF. Solo ora potrà venire impiegato il comando INPUT.

posta dopo l'istruzione precedentemente visualizzata (il codice OP ③ con i suoi due byte operando). Il computer provvede a spostare di due posizioni di memoria tutti i byte situati dopo i due

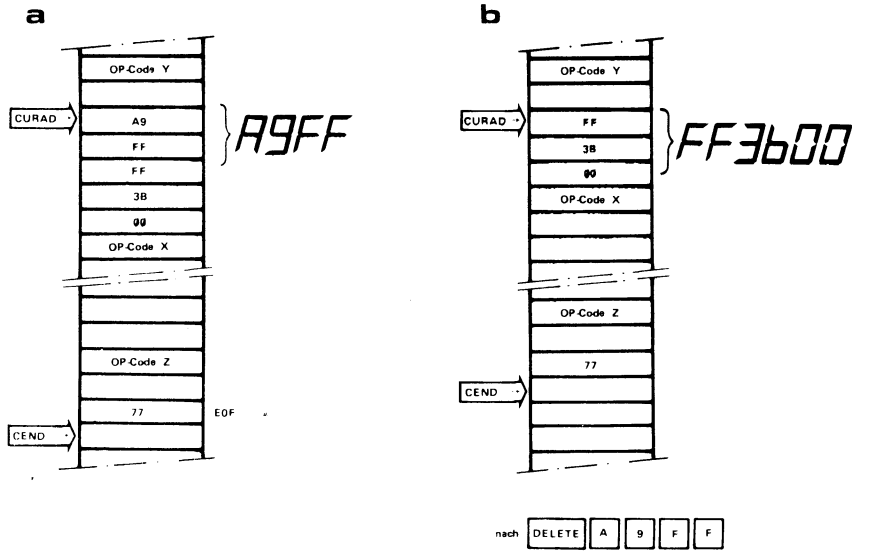


Figura 3. In questo esempio si mostra come viene eliminata l'istruzione LDA # FF ovvero A9 FF dal file col comando DELETE. Il blocco di dati posto fra CURAD + 2 e CEND viene a tal fine spostato di due bytes verso l'alto. L'istruzione LDA # FF viene così sovrascritta dal Label FF 3B 00.

byte operando del codice OP ③ . Anche il carattere EOF viene spostato di due posizioni più in basso, dato che l'istruzione LDA#FF è lunga due byte.

* La Fig. 1c ci mostra l'analoga procedura per il comando INSERT. Col comando INSERT la nuova istruzione viene posta prima del codice OP③. L'istruzione LDA#FF, dunque, viene collocata al posto immediatamente precedente il codice OP nella memoria dello Junior-Computer. Dopo l'esecuzione del comando INSERT, il carattere EOF viene spostato, come nel caso precedente, di due posizioni più in basso.

Sappiamo dal capitolo 5 che dopo il Cold Start Entry la prima istruzione deve *sempre* venire introdotta con il comando INSERT. La fig. 2 chiarisce il perché di tale procedura. Vediamo, partendo dalla situazione iniziale di fig. 2a, cosa succede quando si introduce nello Junior-Computer il Label FF 15 00 con il tasto INPUT: il carattere EOF resta dove si trovava, e quindi il Label FF 15 00 viene posto dopo il carattere EOF (fig. 2b): ciò può avere disastrose conseguenze nell'editing.

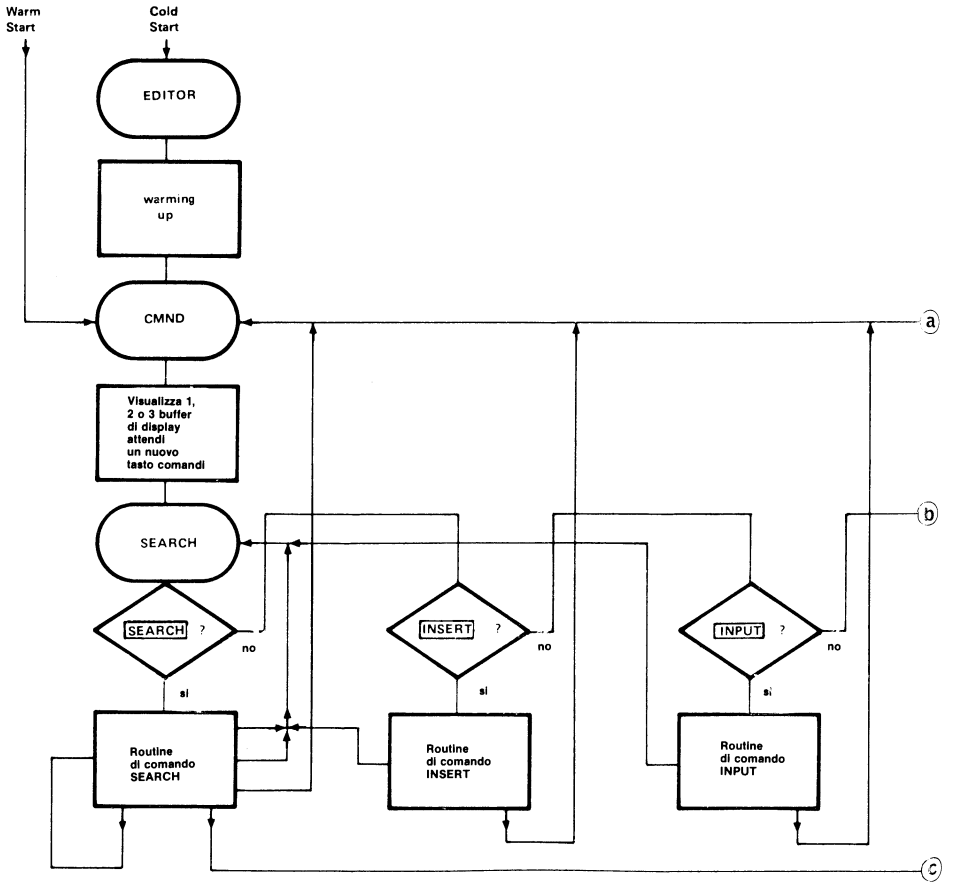
La fig. 2c ci mostra la procedura corretta dopo l'avvio dell'Editor via Cold Start Entry. Il programmatore introduce la prima istruzione (in questo caso un Label) con il comando INSERT. Il Label FF 15 00 si trova ora prima del carattere EOF (77). Dato che un Label è lungo tre byte, il carattere EOF è stato spostato di tre posti più in basso. A questo punto il File contiene un'istruzione FF 15 00 ed il carattere EOF 77. Il file inizia all'indirizzo BEGAD e termina all'indirizzo CEND. Quante più istruzioni vengono ora introdotte nel file, di altrettante posizioni migra verso il basso il carattere EOF. Anche il Pointer CEND si allontana sempre più dal Pointer BEGAD, segno che il file sta assumendo dimensioni sempre maggiori.

La fig. 3 mostra gli effetti del comando DELETE. Supponiamo che ad una certa posizione nel file sia presente l'istruzione LDA#FF seguita dal Label FF 3B 00. Vogliamo eliminare l'istruzione LDA#FF dal file. Prima di tutto ricerchiamo tale istruzione con il comando SEARCH (altra possibilità: vedi capitolo 5). Quando il display visualizza l'istruzione LDA#FF, cioé compare A9 FF, possiamo premere il tasto DELETE. Abbiamo allora la seguente situazione:

* L'istruzione LDA#FF è lunga 2 byte. Dopo A9 FF è posto il Label FF 3B 00.
* A9 viene eliminato dal file, in quanto il computer sposta l'intero blocco di dati che inizia all'indirizzo CURAD + 2 e termina all'indirizzo CEND di 2 byte verso l'alto. L'istruzione A9 00 viene così sovrascritta dai dati successivi e scompare dal file.
* Dopo l'operazione di spostamento, il Pointer CEND viene pure spostato verso l'alto di due posti. Dopo il comando DELETE, quindi il file è divenuto più corto di 2 byte.

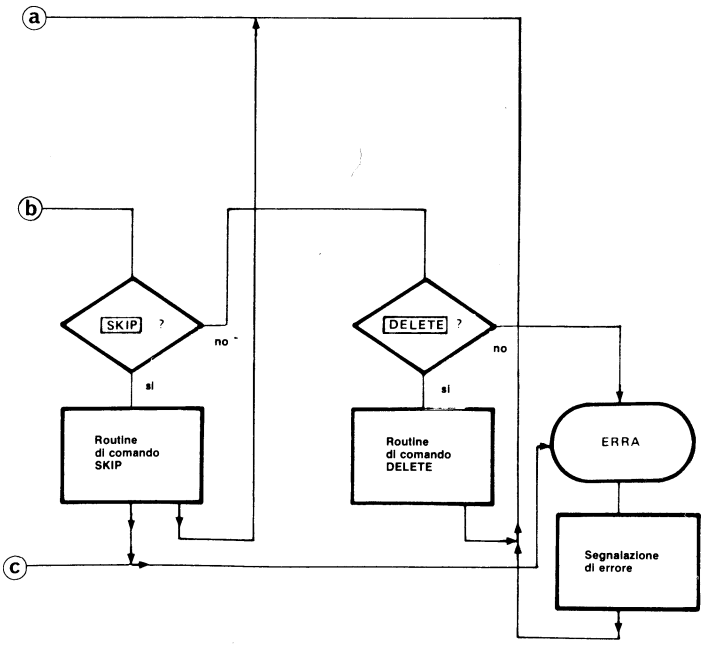
Il diagramma di flusso globake dell'Editor

La fig. 4 illustra il diagramma di flusso globale dell'Editor. Questo diagramma di flusso può essere paragonato al diagramma di flusso globale del programma Monitor (Fig. 1, capitolo 7). Tra i due diagrammi esistono differenze e somiglianze. Vediamo prima le somiglianze. La predisposizione del computer ad operare in Editor-Mode ("warming up") è paragonabile al programma RESET del Monitor. Inoltre, il Label centrale CMND dell'Editor è comparabile al Label START del Monitor. Pure le singole funzioni che l'Junior-Computer svolge gra i Label CMND e SEARCH nell'Editor sono confrontabili con il blocco "C" del programma Monitor: in entrambi, il computer gestisce il display e verifica la tastiera. Dopo il Label SEARCH in Fig. 4 il computer sonda tutti i tasti



comando e riconosce così se l'utente vuole eseguire un comando INSERT, un INPUT, un SEARCH, uno SKIP od un DELETE.
Abbiamo citato le somiglianze fra i due diagrammi di flusso dell'Editor e del Monitor. Vediamo ora le differenze fra i diagrammi di flusso globali dei due programmi. Il programma Editor, a differenza del Monitor non ha un'uscita riconoscibile. Una volta entrati con salto nell'Editor via Cold - o Warm Start Entry, non è possibile uscirne con la pressione di un tasto comando (il tasto GO) come nel Monitor. Per il computer, l'Editor è un loop senza fine, che può venire abbandonato solo mediante i tasti di Hardware RST e ST.
Un'altra differenza è costituita dal Label ERRA. Appartiene ad una sezione dell'Editor che visualizza EEEEEE sul display, se l'utente ha commesso qualche errore di servizio. L'"intelligenza" del Monitor è invece tanto bassa ed elementare che si può tranquillamente rinunciare ai messaggi di errore. Invece nel corso dell'editing è facile commettere errori di impostazione nell'introduzione di dati: è bene dunque che il computer segnali all'utente lo sbaglio. Messaggi di errore vengono emessi nelle seguenti circostanze:

Figura 4. Il diagramma di flusso generale dell'Editor. Da esso risulta chiaro come vengono "filtrati" i 5 tasti comando dell'Editor. Per ogni tasto comando il computer percorre una diversa routine, per convergere poi in ogni caso al punto centrale CMND del programma. Se l'utente compie un errore di impostazione, l'Editor salta al Label ERRA ed emette un messaggio di errore.



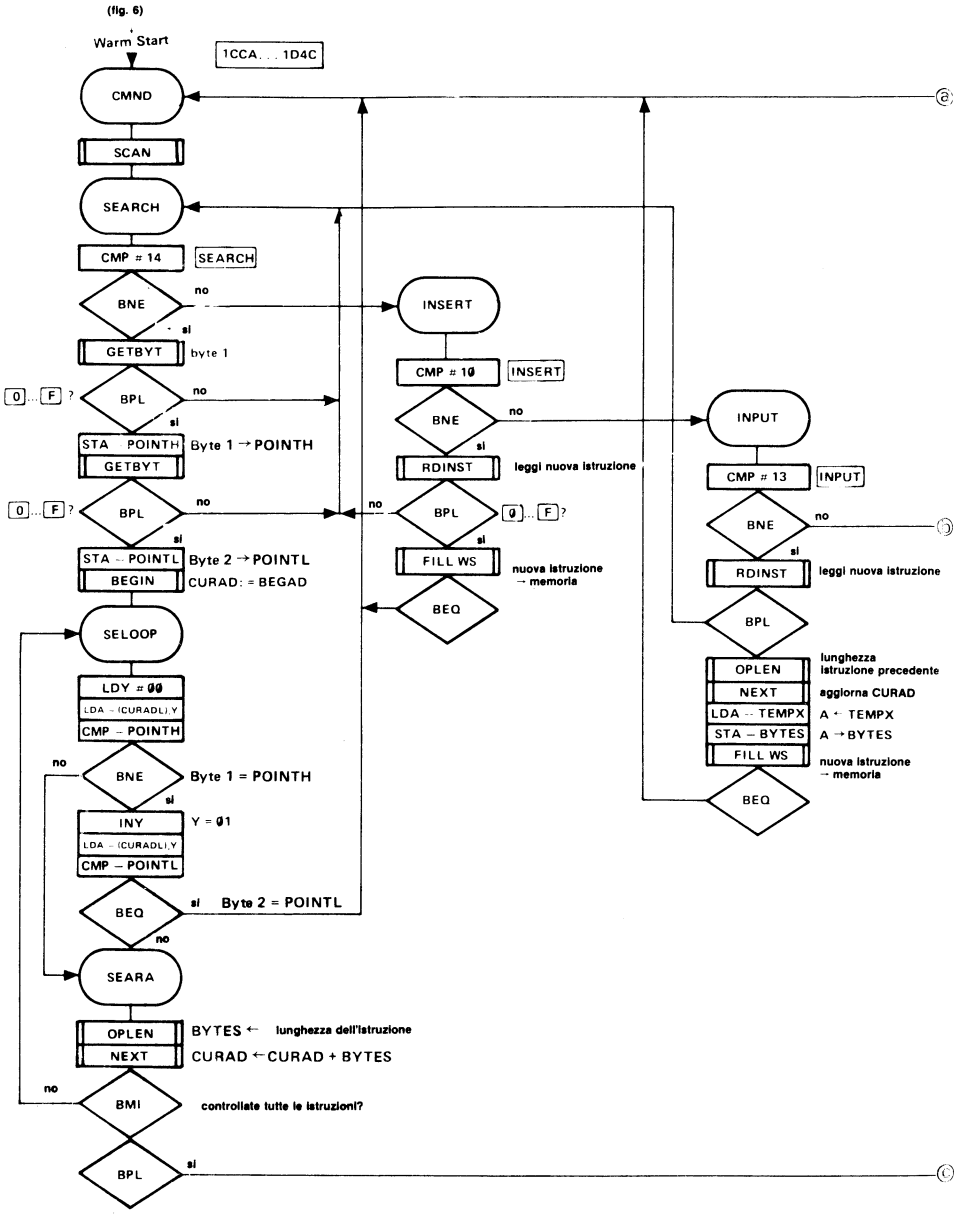
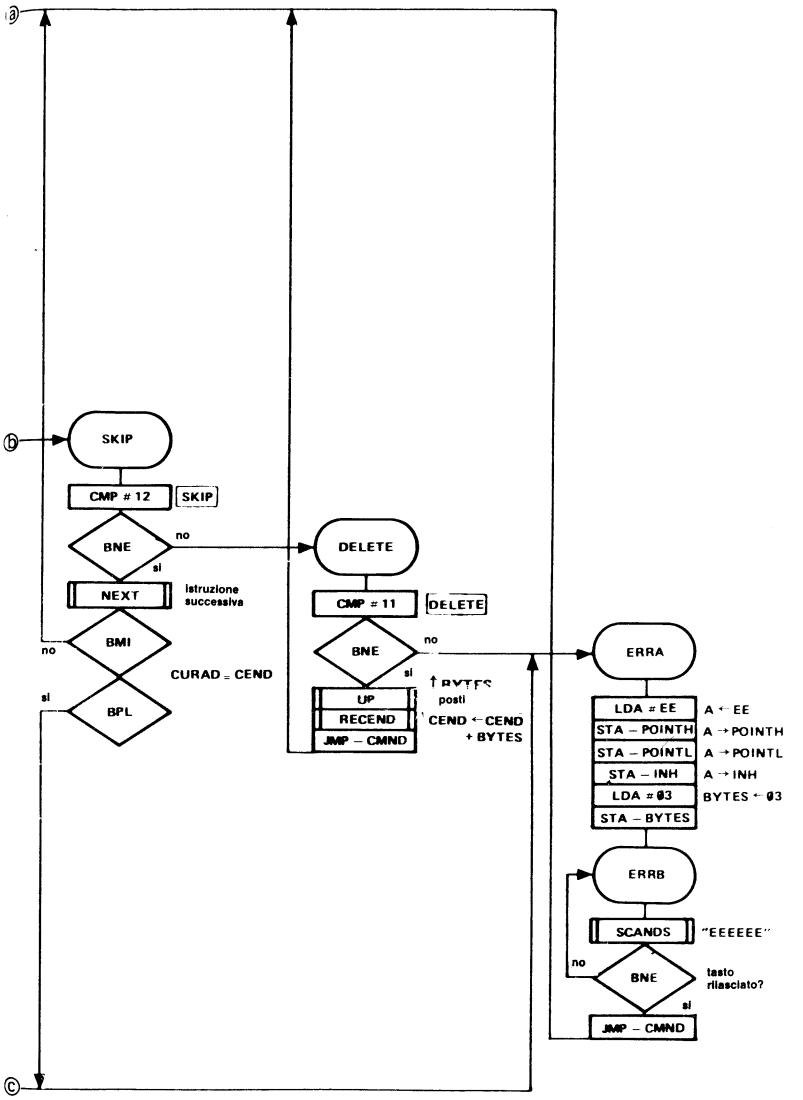


Figura 5. Il diagramma di flusso dettagliato del programma principale Editor. CMND è il Label corrispondente al Warm Start Entry.


a
b
c
SKIP
CMP # 12
SKIP
BNE
no
si
NEXT
istruzione successiva
BMI
no
CURAD = CEND
si
BPL
DELETE
CMP # 11
DELETE
BNE
no
si
UP
RECEND
JMP – CMND
posti
CEND ← CEND + BYTES
ERRA
LDA # EE
A ← EE
STA – POINTH
A → POINTH
STA – POINTL
A → POINTL
STA – INH
A → INH
LDA # Ø3
BYTES ← Ø3
STA – BYTES
ERRB
SCANDS
"EEEEEE"
no
BNE
tasto rilasciato?
si
JMP – CMND

* L'utente ricerca, tramite il comando SEARCH, un determinato formato di due byte nel file. Se tale configurazione di byte non risulta presente nel file, il computer visualizza EEEEEE sul display per il tempo che rimane premuto il tasto SEARCH.
* L'utente percorre, con il comando SKIP, un file. Da ultimo giunge al carattere EOF e preme ancora il tasto SKIP. Dopo il carattere EOF, come sappiamo, il file termina. Perciò il computer visualizza ancora EEEEEE sul display: in tal modo l'utente viene informato che si trova fuori dei limiti del file, fissati da BEGAD e CEND.
* Il computer emette messaggio di errore anche nel caso che l'utenza azioni un tasto sbagliato. È il caso, ad es., in cui lo Jounior-Computer attende la pressione di un tasto comando, mentre l'utente preme un tasto dati.

Sussiste pure un'altra differenza col diagramma di flusso globale del Monitor. In Fig. 4 notiamo che vengono controllati i tasti comando, ma dove vengono trattati dal computer i tasti dati 0...F? I tasti dati, è noto, servono per l'introduzione di istruzioni, ossia per introdurre codici OP e byte operandi.

Come si può vedere dal diagramma di flusso globale (fig. 4), lo Junior-Computer provvede al trattamento dei dati nelle routine SEARCH, INSERT ed INPUT. Nello svolgimento di queste routine la subroutine GETBYT (che ci è nota dal 1° volume) accudisce i tasti dati e forma un byte con i valori di due tasti premuti uno dopo l'altro. La subroutine GETBYT d'altro canto accetta solo i tasti dati, ed ignora perciò i tasti comando! Ne risulta che l'utente può, *durante* l'introduzione di dati (= impostazione di una istruzione) passare all'esecuzione di comandi (= pressione di uno dei tasti comando INSERT, INPUT, SEARCH, SKIP e DELETE), senza introdurre dati sbagliati nel file. È facile comprenderne il motivo, perché sappiamo dal capitolo 5 che i buffer di display ed il File sono separati fra loro. Il contenuto dei tre buffer di display viene copiato dal display dentro il file solo dopo che l'utente ha terminato l'introduzione dell'istruzione completa. Se quindi l'utente durante l'impostazione di una istruzione preme un tasto comandi, l'Editor effettua un salto al Label centrale SEARCH. Ivi l'Editor attende l'impostazione di un nuovo tasto comando e salta successivamente ad una delle routine SEARCH, INSERT, INPUT, SKIP o DELETE.

Il diagramma di flusso dettagliato dell'Editor

Ora che conosciamo il diagramma di flusso globale dell'Editor, dedichiamoci al diagramma di flusso in dettaglio. In fig. 5 è rappresentato il diagramma di flusso dettagliato dell'Editor, privo

della routine di inizializzazione per Cold Start Entry. Questa routine di inizializzazione, che predispone la memoria di lavoro dello Junior-Computer all'introduzione di dati, è riportata in fig. 6. Come si vede da queste due figure, il programma Editor si compone di diverse subroutine, che impareremo a conoscere descrivendo il programma. Inoltre, l'utente può inserire nei propri programmi diverse di tali subroutine, risparmiando così un sacco di lavoro nella stesura dei programmi.

Cold Start Entry

La fig. 6 presenta il diagramma di flusso dettagliato per il lancio dell'Editor via Cold Start Entry. Il programma Editor inizia con la

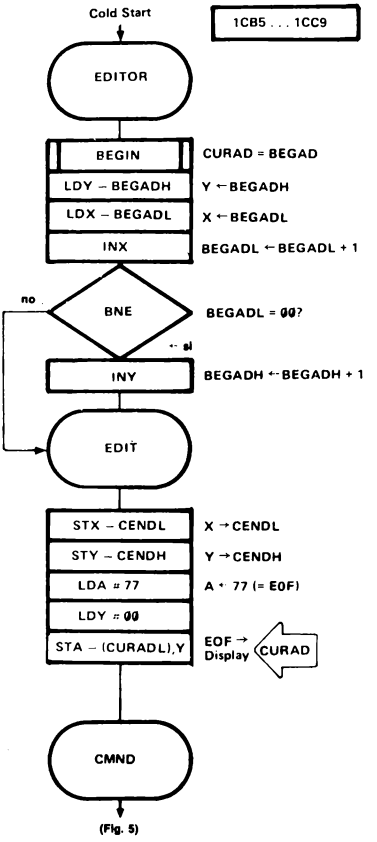


Figura 6. Nel caso di Cold Start Entry occorre predisporre la memoria operativa alla ricezione di dati. I pointer CURAD e CEND sono disposti automaticamente dal computer sugli indirizzi corretti. Viene anche scritto il carattere EOF nel file.

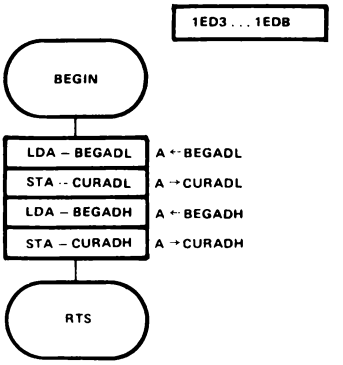


Figura 7. La subroutine BEGIN posiziona il Pointer di display CURAD all'indirizzo indicato da BEGAD; ossia: CURAD = BEGAD.

Subroutine BEGIN, mostrata in fig. 7. La subroutine BEGIN comprende solo 4 istruzioni, tuttavia è di enorme importanza per l'Editor e l'Assembler. Essa provvede a far coincidere i valori del Pointer di display CURAD con quello del Pointer indirizzi BEGAD. Il pointer indirizzi BEGAD indica, come è noto, l'indirizzo di inizio del File in cui il programmatore intende deporre istruzioni e Label. Al rientro dalla subroutine BEGIN il computer, con l'ausilio dei registri X ed Y, pone il pointer indirizzi variabile CEND = BEGAD + 1. Tutto logico, se consideriamo che la locazione cui punta il Pointer di display CURAD è attualmente occupata da 77, e CEND indica la locazione immediatamente successiva a quella contenente il carattere EOF.
Come si fa sì che CEND divenga = BEGAD + 1? Per prima cosa, BEGAD viene incrementato di 1. Se prima di tale incremento BEGADL valeva FF, dopo l'incremento passa a 00. Contemporaneamente occorre quindi incrementare di 1 pure BEGADH. Il computer controlla tale situazione con un'istruzione BNE, che porta eventualmente ad incrementare il registro Y. Dato che BEGAD non deve più modificarsi per tutto il corso del programma Editor, i registri X ed Y vengono adibiti a memorie intermedie. Le due istruzioni seguenti STX e STY fanno sì che CEND = BEGAD + 1. Le tre ultime istruzioni di questa parte del programma servono a deporre il carattere EOF, 77, nella locazione indicata da CURAD. Così viene stabilita la situazione illustrata in fig. 2a.

La gestione del display e dei tasti dell'Editor

La sezione di programma fra i Label CMND e SEARCH nelle fig. 4 e 5 si limita al pilotaggio del display ed al sondaggio della tastiera, In questa sezione il computer determina se devono venire visualizzati uno, due o tutti e tre i buffer di display e se risulta premuto

un qualunque tasto della tastiera dello Junior-Computer. Dopo aver identificato un tasto premuto, il computer ne calcola nel modo ormai noto il valore. Tutti questi compiti vengono svolti dal computer nella subroutine SCAND, il cui diagramma di flusso dettagliato è riportato in fig. 8.
All'inizio di questa subroutine si carica 02 nel registro X e 00 nel registro Y. Ne segue che a partire dal Label FILBUF viene copiato nei tre buffer il display POINTH, POINTL ed INH il contenuto di tre successive locazioni di memoria:

1. X = 02, Y = 00: copia nel buffer di display POINTH il contenuto della cella indicata dal Pointer di display CURAD
2. X = 01, Y = 01: copia nel buffer di display POINTL il contenuto della locazione di memoria CURAD + 1
3. X = 00, Y = 02: copia nel buffer di display INH il contenuto della cella di memoria CURAD + 2.
4. X = FF: la successiva istruzione BPL non porta a salti ed il computer giunge alla subroutine OPLEN. Questa subroutine risulta pressoché identica alla subroutine LENACC del 1° volume. La subroutine OPLEN calcola la lunghezza dell'istruzione indicata come codice OP dal Pointer di display CURAD. Dopo il rientro dalla subroutine OPLEN il valore della lunghezza della istruzione da visualizzare sul display si trova nella locazione di RAM BYTES (indirizzo: 00F6). Nota la lunghezza dell'istruzione da visualizzare, il computer sa se, mediante la subroutine SCANDS, deve visualizzare sul display solo il contenuto di POINTH, od il contenuto di POINTH e POINTL, od infine il contenuto di POINTH, POINTL ed INH. I tre buffer di display citati contengono i seguenti dati:

POINTH: contiene il codice OP dell'istruzione;
POINTL: contiene il primo byte dell'operando;
INH: contiene il secondo byte dell'operando.

Come già citato, è la subroutine SCANDS a svolgere il pilotaggio dei display e la scansione della tastiera. SCANDS la conosciamo già dal capitolo 7: corrisponde alla subroutine SCANDS privata della prima parte di programma. In questa prima parte, nel buffer INH venivano caricati i dati posti nella locazione indicata dal Pointer indirizzi POINTH, POINTL. Questa parte di SCAND non è necessaria nel programma Editor. SCANDS, con tutte le sue subroutine, è nota in linea di principio dal capitolo 7: chi avesse scordato l'esatto modo di funzionamento di questa importante subroutine è pregato di rileggere quel capitolo. Qui di seguito riassumiamo solo le cose più importanti riguardo SCANDS:
SCANDS provvede, assieme alla locazione di memoria BYTES, alla gestione completa del display. Se il contenuto di BYTES vale 01, il computer visualizza sul display solo il contenuto di POINTH. L'istruzione da visualizzare ha una lunghezza di 1 byte, ed i quattro display di destra restano spenti. Se il contenuto di BYTES vale 02, il computer visualizza il contenuto dei buffer POINTH e

POINTL sul display. L'istruzione da visualizzare ha una lunghezza di 2 byte, e i due ultimi display di destra restano spenti. Se infine il contenuto di BYTES è 03, il computer visualizza il contenuto di POINTH, POINTL ed INH sul display. L'istruzione da visualizzare ha una lunghezza di 3 byte, e tutti e 6 i display sono accesi.
Al termine della subroutine SCANDS il computer verifica se è premuto un qualsiasi tasto nella tastiera. Se sì, il contenuto dell'Accu al rientro dalla subroutine SCANDS è diverso da zero. Se invece non risulta premuto alcun tasto, il contenuto dell'accumulatore vale 0. È un fatto che occorre tener sempre ben presente nella successiva descrizione del programma Editor. Dopo la subroutine OPLEN (fig. 8) si giunge al Label SCANA. La parte di programma compresa fra il Label SCANA e l'istruzione conclusiva RTS ci è già nota dal capitolo 7. La prima sezione del programma, SCANA-BNE-SCANA, viene percorsa sino a quando è stato rilasciato l'ultimo tasto premuto. In questo loop il computer fa comparire già una parte o l'istruzione completa sul display.
Quando il tasto è lasciato libero, giungiamo al Label SCANB. Il computer permane nel loop SCANB-BEQ-SCANB la maggior parte del tempo. In questo loop esso attende fin che viene premuto un nuovo tasto. Quando viene premuto un tasto, viene richiamata ancora una volta la subroutine SCANDS. Come detto nel capitolo 7, questo secondo richiamo di SCANDS provvede ad eliminare i rimbalzi del tasto. Se al rientro da SCANDS il tasto risulta ancora premuto, nella successiva subroutine GETKEY ne viene calcolato il valore. Il computer esce dalla subroutine SCAN, dopo il rientro dalla subroutine GETKEY, con il valore del tasto nell'accumulatore.
Ricordiamo che i tasti di comando nell'Editor hanno i seguenti valori:
SEARCH: valore di tasto 14 (come il tasto PC)
INSERT: valore di tasto 10 (come il tasto AD)
INPUT: valore di tasto 13 (come il tasto GO)
SKIP: valore di tasto 12 (come il tasto +)
DELETE: valore di tasto 11 (come il tasto DA)
La verifica di un dato tasto comando viene effettuata con un'istruzione CMP, seguita da un'istruzione BNE. Se si trova attivo un tasto comando, alla successiva istruzione BNE il salto di programma non ha luogo, ed invece il computer si occupa del tasto che è stato premuto (vedi fig. 4 e 5).

Il messaggio di errore EEEEEE

La parte di programma che segue il Label ERRA in fig. 5 viene svolta quando è necessaria la segnalazione di un errore commesso. I casi in cui questo si rende necessario li abbiamo già illustrati in modo esauriente. Le prime istruzioni dopo il Label ERRA caricano EE nei tre buffer di display POINTH, POINTL ed INH. Dato

che per un messaggio di errore tutti e 6 i display devono illuminarsi, si carica 03 nella cella BYTES. La parte di programma che viene dopo ERRB è un loop di attesa, nel corso del quale viene svolta la routine display/tastiera SCANDS. Il messaggio di errore EEEEEE rimane sul display per tutto il tempo che resta premuto il tasto che ha provocato l'errore. Quando questo tasto viene lasciato libero, il programma risalta al Label centrale CMND.

La Routine SEARCH

Di seguito al Label SEARCH inizia una routine che viene percorsa dal computer ogni volta che il programmatore preme il tasto SEARCH. Sappiamo che il computer rientra dalla subroutine SCAN con il valore del tasto comando nell'Accu. Con le due istruzioni CMP # 14 e BNE si provvede a "filtrare" il tasto SEARCH. La parte di programma relativa al comando SEARCH deve svolgere le seguenti funzioni:

1. Col comando SEARCH viene effettuata la ricerca d'una data configurazione di due byte nel file.
2. La ricerca deve iniziare da BEGAD e terminare non appena è stato rintracciato il dato formato di byte.
3. Se il formato di byte ricercato non è presente nel file compreso fra BEGAD e CEND, il computer deve emettere un messaggio di errore.

All'inizio della routine SEARCH incontriamo due volte la subroutine GETBYT, che conosciamo dal 1° volume. In realtà sappiamo che cosa provoca la subroutine GETBYT, ma non come funziona. Vediamo dunque di rimediare a questa lacuna!

La Subroutine GETBYT

La fig. 9 ci presenta il diagramma di flusso dettagliato della subroutine GETBYT. Il compito di questa subroutine è di comporre in un unico byte due tasti dati premuti in successione. GETBYT accetta solo tasti dati, e non tasti comando.
I due tasti premuti vengono composti assieme nell'Accu nel seguente modo: il valore di tasto del tasto dati premuto per primo viene trasferto al Nibble alto dell'Accu, ed il valore del secondo tasto premuto è trasferito al nibble basso dell'Accu.Al principio della subroutine GETBYT viene richiamata la subroutine SCANA. Questa costituisce la seconda parte della subroutine SCAND descritta in precedenza (fig. 8). Di seguito al Label SCANA in fig. 8 il computer svolge i seguenti compiti:

* Determina se l'utente ha già premuto un nuovo tasto.
* Se è premuto un nuovo tasto, il computer calcola il valore di tasto relativo richiamando la subroutine GETKEY.
* Mentre si svolgono le azioni citate, il computer gestisce il display e sonda la tastiera. Quale dei tre buffer di display

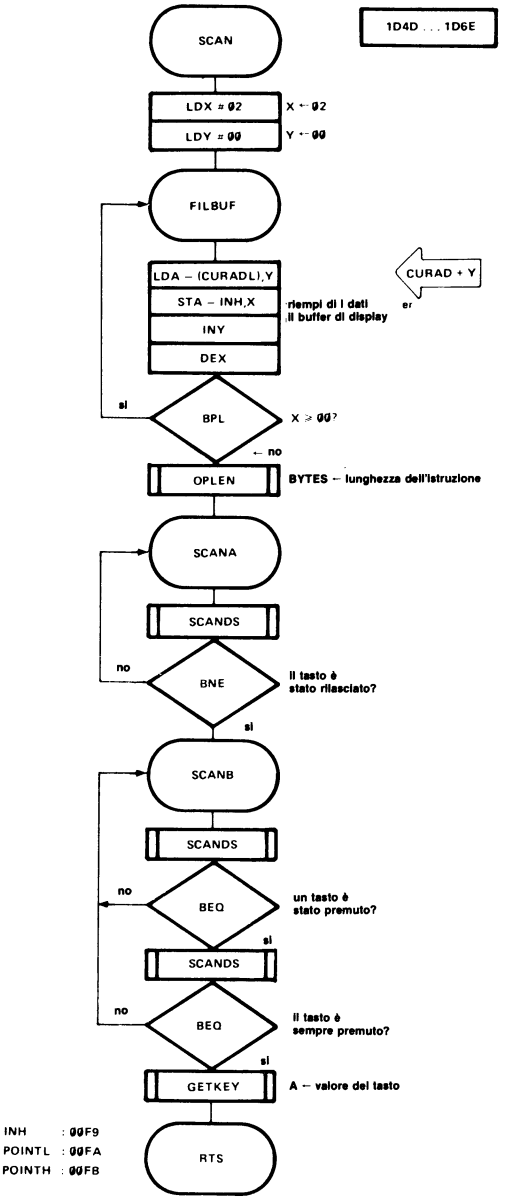


Figura 8. La subroutine SCAN introduce nei buffer di display POINTH, POINTL ed INH il codice OP e gli operandi d'una istruzione. Di seguito, visualizza questa istruzione sul display. Il display può assumere una lunghezza variabile. SCAN calcola allo stesso tempo il valore d'un tasto, se risulta premuto un tasto della tastiera. Per il pilotaggio del display ed il sondaggio della tastiera vengono adibite le subroutine SCANDS e GETKEY.

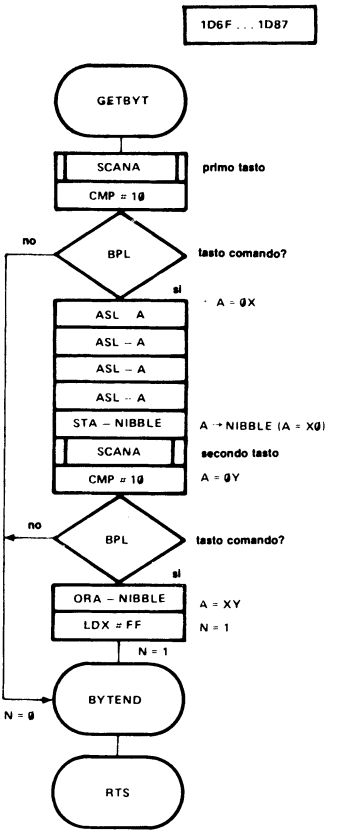


Figura 9. La subroutine GETBYT introduce nell'Accu i valori di due tasti premuti in successione. Il Flag N viene posto ad 1, quando vengono premuti solo tasti dati. Eventuali tasti comando vengono ignorati dalla subroutine GETBYT, col resettare a zero il Flag N dopo il rientro al programma principale.

POINTH, POINTL ed INH debba comparire sul display è ancora il contenuto di BYTES a stabilirlo.

I tasti comando hanno un valore di tasto eguale o maggiore di 10. I tasti dati, invece, hanno un valore eguale o minore di 0F. Con le due istruzioni CMP # 10 e BPL è quindi possibile distinguere i tasti comando dai tasti dati. Se durante la subroutine GETBYT viene premuto un dato tasto comando, dopo l'istruzione CMP # 10 il Flag N è = 0, ed il programma salta, mediante il comando RTS, alla routine SEARCH.

Se invece durante la subroutine GETBYT viene premuto un tasto dati, il computer rientra dalla subroutine SCANA con il valore del tasto dati nell'Accu. Se si tratta del tasto premuto per primo, dopo un'operazione di spostamento ripetuta quattro volte (4 volte ASL-

A) il contenuto dell'Accu cambia da 0X a X0. X corrisponde al valore del tasto premuto per primo.
Per la lettura del tasto premuto per secondo viene ancora richiamata la subroutine SCANA. Nel corso di questa subroutine l'accumulatore funge da registro operativo. Perciò, prima del secondo richiamo di SCANA occorre salvare il contenuto dell'Accu nella locazione di memoria NIBBLE. Solo allora viene richiamata SCANA e successivamente, mediante le istruzioni CMP # 10 e BPL, si determina se è stato premuto un tasto dati od un tasto comandi. Se si tratta di un eventuale tasto comando, esso viene anche in questo caso ignorato, ed il programma salta, con Flag N=0, alla routine SEARCH.

Se invece era un tasto dati dal valore Y, dopo l'istruzione ORA-NIBBLE abbiamo un byte dati XY. X rappresenta il valore del primo tasto. Y il valore del secondo tasto dati. Al termine della subroutine GETBYT, con l'istruzione LDX # FF, si pone ad 1 il Flag N. Così, dopo l'istruzione RTS, si può controllare se durante la subroutine GETBYT è stato premuto un tasto comando.
Facciamo rilevare, fra parentesi, che GETBYT è una subroutine veramente universale. Il programmatore si accorgerà ben presto quando risulti utile questa subroutine nello sviluppo di una "Do It Yourself Software" per propri programmi.
Torniamo alla routine SEARCH in fig. 5! Dopo aver richiamata la prima volta la subroutine GETBYT il Flag N è = 0, se nel corso di questa subroutine è stato premuto un tasto comando. Se invece sono stati premuti due tasti dati, il Flag N = 1. È dunque facile stabilire al rientro da GETBYT, se si è ritornati al Label SEARCH saltando, via un'istruzione BPL, oppure no.

Se nel corso della subroutine SEARCH sono stati premuti due tasti dati, questi due tasti compongono la prima parte del formato di due byte da ricercare. Questa prima parte viene trasferita nel buffer di display POINTH, e compare sul display nel corso del secondo richiamo della subroutine GETBYT. Mentre si trova nuovamente in GETBYT, il computer attende l'introduzione della seconda parte del formato di due byte da ricercare. Se viene premuto un tasto comando, il computer salta al Label SEARCH ed attende ancora l'introduzione corretta dei dati. Se si sono premuti due tasti dati, questi compongono la seconda parte del formato di due byte da ricercare. Il valore di questi tasti viene trasferito al buffer di display POINTL. I due buffer POINTH e POINTL sono i buffer dati del formato di due byte.
Conosciamo così integralmente la configurazione di byte che deve venire ricercata nel file fra BEGAD e CEND, e può iniziare l'operazione di ricerca. Dapprima viene richiamata la subroutine BEGIN e si rende CURAD = BEGAD. Quindi viene immediatamente percorso il successivo tratto di programma della routine SEARCH che parte dal Label SELOOP, fin quando viene rinvenuto il

richiesto formato di byte. Trovata la configurazione di due byte desiderata, il programma salta indietro al Label centrale CMND, e visualizza sul display il citato formato di byte, nel corso della successiva subroutine SCAN. Può succedere che il formato di due byte richiesto non venga rinvenuto nel file fra BEGAD e CEND: in tal caso la routine di ricerca che segue SELOOP salta al Label ERRA, fornendo così la segnalazione di errore sul display. Come avviene in dettaglio l'operazione di ricerca? Il computer a tale scopo confronta successivamente tutte le istruzioni presenti nel file con il formato di due byte posto nei due buffer di display POINTH, POINTL. In termini espliciti si avrà:

1. Carica nell'Accu il codice OP indicato da CURAD. Confronta questo codice OP col contenuto del buffer di display POINTH.
2. Se il contenuto di POINTH *non è eguale* al codice OP indicato da CURAD, calcola la lunghezza dell'istruzione nella subroutine OPLEN. Sposta il valore di CURAD corrispondentemente di uno, due o tre posti di memoria più in basso. Ciò equivale a dire: disponi il Pointer CURAD per indicare il codice OP dell'istruzione successiva. Ciò avviene con l'ausilio della subroutine OPLEN. Nella subroutine NEXT il computer controlla pure che il valore di CURAD non abbia già superato CEND. Se ciò fosse, il programma salta al Label ERRA per l'emissione di un messaggio di errore. Il formato di due byte richiesto non risulta in tal caso presente nel file.
3. Se il contenuto di POINTH *è eguale* al codice OP indicato da CURAD, preleva dall'Accu il byte successivo al codice OP. Confronta ora questo byte con il contenuto del buffer di display POINTL. Se anche questo corrisponde, ossia se il byte successivo al codice OP indicato da CURAD coincide con quello presente in POINTL, il formato richiesto è stato trovato. La routine SEARCH salta al punto centrale CMND nel programma, e visualizza il formato di byte rinvenuto. Col comando SEARCH è dunque possibile ricercare istruzioni lunghe due o tre byte.
4. Se il contenuto di POINTH *è eguale* al codice OP indicato da CURAD, mentre il contenuto di POINTL *non è eguale* al contenuto della locazione CURAD + 1, continua l'operazione di ricerca. Determina di nuovo la lunghezza dell'istruzione indicata da CURAD (con OPLEN), e sposta il Pointer CURAD di due o tre posti più in basso (con NEXT). CURAD indica ora il codice OP dell'istruzione successiva. Ripeti il confronto fra l'istruzione, il cui codice OP è indicato da CURAD, ed il contenuto dei buffer POINTH e POINTL, sin quando si rintraccia il formato di due byte richiesto, ovvero si supera in CURAD il limite indicato da CEND.

Il comando SEARCH confronta quindi fra loro quattro byte. Si

tratta dei due byte posti nei due buffer di display POINTH e POINTL, confrontati con i due byte indicati dal Pointer CURAD e CURAD + 1. Nel corso della subroutine NEXT, CURAD viene ogni volta disposto ad indicare il codice OP della successiva istruzione. Dato che nel corso dell'operazione di ricerca l'Accu viene caricato indirettamente indicizzato, il contenuto del registro Y è 0 nel primo confronto, e poi 1 nel secondo. Per Y = 00 il confronto viene effettuato fra il codice OP corrente ed il contenuto del buffer di display POINTH; per Y = 01 fra il primo byte operando dell'istruzione da ricercare (formato di due byte dato) ed il contenuto del buffer di display POINTL. Il corso della ricerca nel file di una istruzione dovrebbe ora essere sufficientemente chiaro. Durante tale ricerca, il computer passa nelle due subroutine OPLEN e NEXT. Dopo aver descritto come esse funzionano, passeremo ora ad illustrarne la struttura istruzione per istruzione.

La Subroutine NEXT

La fig. 10 ci mostra il diagramma di flusso dettagliato della subroutine NEXT. Dopo aver resettato a 0 il Carry Flag, il Pointer CURAD viene incrementato di una quantità pari al valore della locazione di memoria BYTES. In BYTES è sempre contenuto il valore della lunghezza dell'istruzione visualizzata sul display. Questo incremento viene eseguito dall'addizione a 16 bit CURAD + BYTES. Il Pointer di display CURAD indica ora la successiva istruzione. La seconda parte della subroutine NEXT verifica se il Pointer CURAD abbia già superato il valore del Pointer indirizzi corrente CEND. Ciò si realizza con la sottrazione a 16 bit CURAD - CEND. Il risultato di questa sottrazione agisce sul Flag N nel modo seguente:

1. N = 1: CURAD è maggiore di CEND; deve quindi essere emesso un messaggio di errore.
2. N = 0: CURAD non ha ancora superato CEND, ma si sta muovendo fra BEGAD e CEND. Dopo l'istruzione RTS tutto risulta normale: il programma può continuare regolarmente.

La subroutine OPLEN calcola di nuovo la lungheza dell'istruzione il cui codice OP è indicato da CURAD.
Parte di questa subroutine ci era già nota dal 1° volume. Al termine di questo capitolo torneremo ancora in particolare su questa interessante subroutine.

La Routine INSERT

Per descrivere la Routine INSERT rivediamo ancora la fig. 5. Il tasto comando INSERT viene "filtrato" tramite le due istruzioni CMP # 10 e BNE. Se il computer rientra dalla subroutine SCAN col valore 10 nell'Accu nel programma principale dell'Editor, vie-

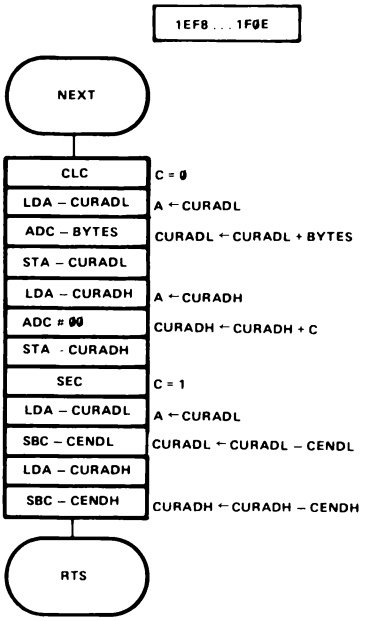


Figura 10. La subroutine NEXT sposta il Pointer di display CURAD di uno, due o tre indirizzi verso il basso. Al tempo stesso verifica se CURAD non abbia già oltrepassato il limite CEND.

ne eseguito il comando INSERT. Esso ha i seguenti compiti:

1. Leggi un'istruzione impostata dalla tastiera e presente nei buffer di display dello Junior-Computer. Determina la lunghezza dell'istruzione.
2. Letta l'istruzione completa nei buffer di display, copiala in memoria in posizione immediatamente precedente l'istruzione al momento visualizzata sul display.

Per svolgere queste funzioni, nel programma Editor sono previste due subroutine, la subroutine RDINST (= ReaD INSTruction = leggi un'istruzione) e la subroutine FILLWS (= FILL Work Space = riempi la memoria operativa, ossia copia il contenuto del buffer di display nel file). Cosa sono dunque queste due subroutine?

La Subroutine RDINST

Il diagramma di flusso dettagliato della subroutine RDINST è dato in fig. 11. La funzione di questa subroutine è di copiare un'istruzione lunga uno, due o tre byte presente nei buffer di display dello Junior-Computer. La subroutine RDINST inizia con la subroutine GETBYT. Con essa vengono letti nel computer due tasti dati: essi corrispondono al codice OP dell'istruzione da leggere. Al rientro

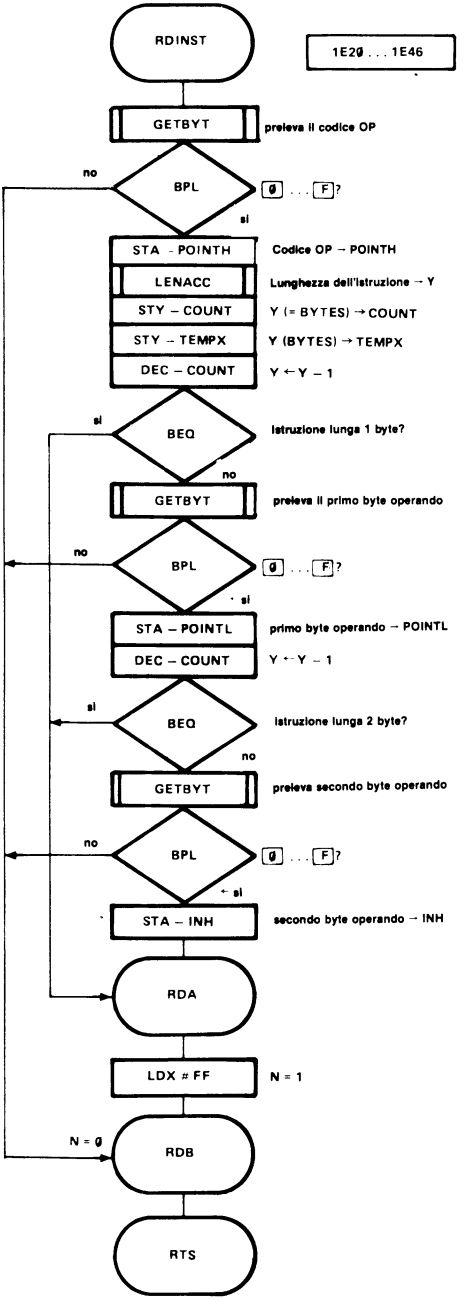


Figura 11. La subroutine RDINST legge nel computer un'istruzione impostata dalla tastiera. Una volta letta tutta l'istruzione il processore esce dalla subroutine con il Flag N posto ad 1.

da GETBYT il codice OP è nell'Accu, e viene trasferito nel buffer di display POINTH. Se l'utente nel corso della subroutine GETBYT ha premuto un tasto comando, il computer torna come al solito dalla subroutine RDINST al programma Editor, col Flag N posto ad 1.
Posto il codice OP dell'istruzione nel buffer POINTH, il computer determina la lunghezza dell'istruzione corrispondente a tale codice OP. Si richiama perciò la subroutine LENACC, che costituisce una parte della già citata subroutine OPLEN. Tratteremo entrambe queste subroutine alla fine del capitolo. La lughezza dell'istruzione così stabilita viene distribuita nelle tre celle di memoria BYTES, COUNT e TEMPX.
Dopo aver decrementato di uno la locazione COUNT, il computer controlla se oltre al codice OP debbano venir letti altri byte operandi nel buffer di display. Se l'istruzione risulta lunga solo un

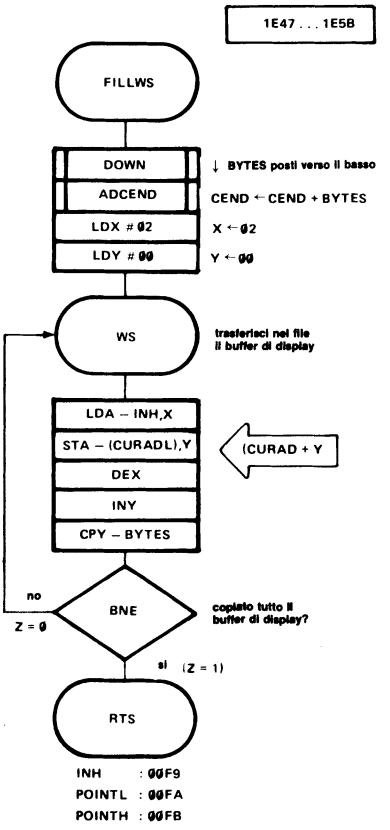


Figura 12. La subroutine FILLWS libera nel file lo spazio necessario per inserire nella memoria operativa una nuova istruzione, copiandola dal display.

byte, il programma salta al Label RDA e si carica FF nel registro X. Il Flag N, per effetto di questa operazione, passa ad 1. E questo per i due seguenti motivi:

* Se il computer al rientro dalla subroutine RDINST trova il Flag N ad 1, vuol dire che è stata introdotta l'intera istruzione. L'istruzione impostata è ora nel buffer di display dello Junior-Computer.
* Se il valore del Flag N al rientro dalla subroutine RDINST è invece 0, vuol dire che l'utente nel corso dell'impostazione dell'istruzione ha premuto un tasto comando. Nel buffer di display è quindi presente una parte dell'istruzione, che però non deve venire trasferita nella memoria operativa (= file) del computer.

Dallo stato (1 o 0) del Flag N al rientro dalla subroutine RDINST nel programma principale si può quindi controllare se la data istruzione debba venire copiata dal buffer di display nella memoria operativa, o se il computer si deve occupare di un nuovo tasto comando.
Ora però facciamo un passo indietro, per considerare la lettura di istruzioni lunghe più di un byte. Se dopo il primo decremento il contenuto della cella di memoria COUNT = 00, significa che l'istruzione è lunga due o tre byte. Viene perciò nuovamente richiamata la subroutine GETBYT. Il computer legge in tal modo il primo byte operando dell'istruzione nel buffer di display. Questo primo byte viene posto in POINTL. Quindi il contenuto di COUNT viene nuovamente decrementato di 1. Se a questo punto COUNT vale zero, l'istruzione da leggere è lunga due byte, ed il computer rientra nel programma principale Editor col Flag N = 1. Se invece COUNT è ancora diverso da 0, l'istruzione da leggere è lunga tre byte. Perciò si richiama GETBYT per la terza volta. Il computer legge questa volta il secondo byte operando, e lo porta nel buffer di display INH, rientrando infine nel programma principale dell'Editor con il Flag N = 1.

Le Subroutine FILLWS e ADCEND

Riprendiamo ancora una volta la fig. 5. Di seguito al Label INSERT si è provveduto a "filtrare" il comando INSERT, saltando poi alla subroutine RDINST. Nel corso di tale subroutine si è letta un'istruzione nel buffer di display, rientrando con il Flag N = appena terminato di deporre l'intera istruzione nel buffer di display. Con queste premesse, il processore non esegue il successivo BPL e giunge alla subroutine FILLWS. Questa subroutine trasporta l'istruzione introdotta dal buffer di display al file dello Junior-Computer, e corregge successivamente il Pointer indirizzi CEND. Le fig. 12 e 13 illustrano il diagramma di flusso dettagliato delle subroutine FILLWS e ADCEND.
Al principio di FILLWS è posta la subroutine DOWN. Discuteremo

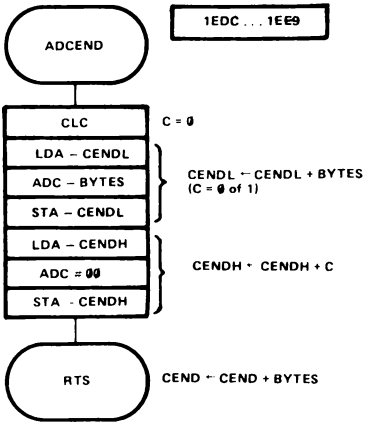


Figura 13. La subroutine ADCEND costituisce un sotto programma di FILLWS, che serve a spostare il Pointer CEND di uno, due o tre posizioni verso il basso.

questa subroutine, relativamente complicata, più avanti. Per ora di essa ci interessa conoscere essenzialmente quanto segue:

1. Col comando INSERT viene richiesto di inserire una nuova istruzione prima di quella visualizzata sul display. La subroutine DOWN provvede a predisporre il relativo spazio in memoria.
2. Il Pointer di display CURAD indica il codice OP dell'istruzione al momento visualizzata sul display. Dato che avanti a questa istruzione deve essere inserita un'istruzione lunga uno, due o tre byte, la subroutine DOWN deve corrispondentemente spostare verso il basso di uno, due o tre posizioni l'intero blocco dati fra CURAD e CEND.
3. Dopo che in tal modo si è fatto il posto necessario in memoria, lo Junior-Computer può copiare nel file dal buffer di display la nuova istruzione impostata.

Tanto ci basti per ora sapere in merito alla subroutine DOWN, per meglio comprendere la subroutine FILLWS. Prima che, nella routine INSERT, venga richiamata la subroutine FILLWS, il computer, durante la subroutine RDINST, ha determinato la lunghezza dell'istruzione appena impostata. La lunghezza dell'istruzione, prima del richiamo di FILLWS, si trova in BYTES. Quando il computer è giunto alla subroutine FILLWS e poi salta alla subroutine DOWN, il blocco di dati fra CURAD e CEND viene spostato verso il basso di tanti posti quanti corrispondono al valore posto in BYTES. Se ad esempio l'istruzione impostata è lunga 2 byte, nella cella BYTES si trova 02 (esadecimale). Pertanto la subroutine DOWN provvede a spostare verso il basso di due posizioni il blocco dati fra CURAD e CEND. Il modo in cui la subroutine DOWN fa posto ad una nuova istruzione, copiata dal buffer di

display nel file, è dunque chiaro. Al rientro da DOWN alla subroutine FILLWS (fig. 12) bisogna ancora correggere il valore del Pointer indirizzi CEND. Questo compito è svolto dalla subroutine ADCEND (= ADvance Current END Address = posiziona il Pointer CEND di uno, due o tre byte più in basso). Nella subroutine ADCEND (fig. 13) il computer esegue un'addizione di 16 bit, sommando al Pointer indirizzi CEND il valore della locazione BYTES: CEND = CEND + BYTES. Di seguito al Label WS (fig. 12) l'istruzione appena impostata viene copiata dal buffer di display nel file. Inizialmente il registro X = 02 ed il registro Y = 00. Il successivo decorso del programma può essere ancora riassunto in tre fasi:

1. Il contenuto di POINT (= codice OP dell'istruzione impostata) viene trasferito alla locazione indicata dal Pointer di display CURAD (X = 02, Y = 00).
2. In funzione della lunghezza (= valore di BYTES) dell'istruzione, il valore di POINTL (= primo byte operando) viene trasferito alla locazione indicata da CURAD + 1 (X = 01, Y = 01).
3. In funzione della lunghezza dell'istruzione (valore di BYTES), il valore di INH (= secondo byte operando) viene trasferito alla cella indicata da CURAD + 2 (X = 00, Y = 02).

Il confronto fra il valore del registro Y ed il contenuto di BYTES segnala se, dopo copiato il codice OP dal buffer di display POINTH, devono venire trasferiti ulteriori byte operandi dai buffer di display ed INH nel file.
Quando tutti i byte sono stati trasferiti al file dai buffer di display, il computer rientra dalla subroutine FILLWS alla routine INSERT (fig. 5). Uscendo dalla subroutine FILLWS il Falg Z risulta sempre posto = 1. Pertanto il programma, al termine di INSERT, mediante un'istruzione BEQ salta verso il Label centrale CMND, ed il programma Editor attende nuovamente la pressione di un tasto comando. Il comando INSERT è così esaurito.

La routine INPUT

La routine INPUT presenta molte somiglianze con la routine INSERT. È dunque sufficiente limitarsi alle differenze fra i due programmi. Che compiti deve svolgere la routine INSERT? Ricordiamo dal capitolo 5:

1. Lettura di un'istruzione da tastiera nel buffer di display.
2. Disporre l'istruzione ora letta subito dopo l'istruzione visualizzata sul display.

Per illustrare la routine INPUT riconsideriamo ancora la fig. 5. Il comando INPUT viene "filtrato" dalle due istruzioni CMP # 13 e BNE, e si salta poi alla nota subroutine RDINS. Questa subroutine legge una istruzione dalla tastiera e la copia nel buffer di display. Se durante la subroutine RDINS l'utente preme un tasto comando, si esce da RDINS e la seguente istruzione BPL riconduce il computer nella parte centrale del programma, al Label CMND. Se

si è invece introdotta l'istruzione completa nei buffer di display, si ha un salto nella subroutine OPLEN. Il pointer di display CURAD attualmente indica ancora la precedente istruzione del file. Il computer calcola dunque nella subroutine OPLEN la lunghezza dell'istruzione precedentemente visualizzata. Il richiamo della subroutine NEXT fa spostare il Pointer di display CURAD verso il basso di un numero di posizioni pari alla lunghezza dell'istruzione precedente. CURAD indica ora l'indirizzo a cui deve essere posta nel file la nuova istruzione. Al rientro dalla subroutine NEXT, il contenuto della cella di memoria TEMPX viene trasferito nella cella BYTES. In BYTES, come sappiamo, si trova solitamente la lunghezza di un'istruzione. Cosa c'era in TEMPX? Riconsideriamo un momento la subroutine RDINST (fig. 11). Durante questa subroutine, in occasione della lettura del codice OP della nuova istruzione, la relativa lunghezza dell'istruzione viene deposta in TEMPX. Il computer utilizza l'informazione sulla lunghezza dell'istruzione nella routine INPUT, prima di saltare alla subroutine FILLWS. Così, nella subroutine FILLWS, dopo l'istruzione visualizzata in precedenza sul display viene inserita la nuova istruzione introdotta. Dato che il Pointer di display CURAD non ha subito modifiche da NEXT in avanti, al rientro verso il Label centrale CMND esso indica già il codice OP dell'istruzione appena introdotta. Tale istruzione viene poi visualizzata sul display mediante la subroutine SCAN, ed il computer attende la pressione di un nuovo tasto comando. Abbiamo così esaurito pure il comando INPUT.

La routine SKIP

Per descrivere la routine SKIP ci riferiamo ancora alla fig. 5. Sappiamo dal 1° volume che il comando SKIP viene impiegato durante la verifica di un programma impostato tramite l'Editor. Quali sono i compiti della routine SKIP? Sono chiariti nei due punti seguenti:

1. Saltare alla successiva istruzione ad ogni pressione del tasto SKIP.
2. Emettere un messaggio di errore quando, con il comando SKIP, si sia superato il limite fissato dal carattere EOF.

Il comando SKIP viene "filtrato" mediante le due istruzioni CMP # 12 e BNE. L'unico elemento che deve essere variato per effetto del comando SKIP è il valore del Pointer di display CURAD. Conosciamo già la subroutine che sposta tale Pointer di uno, due o tre posti verso il basso: è la subroutine NEXT (fig. 10). In questa subroutine il computer verifica se il Pointer CURAD ha già superato il limite indicato dal pointer indirizzo finale corrente CEND. Se CURAD risulta maggiore di CEND, deve seguire la segnalazione di errore. Mediante un'istruzione BPL il programma salta al Label ERRA: il display visualizza EEEEEE sin quando non viene rilasciato il tasto SKIP. In casi normali, invece, il programma esce

dalla routine SKIP tramite un'istruzione NMI, verso il Label centrale CMND; lo Junior-Computer attende che venga premuto un nuovo tasto comando.

La routine DELETE

Quali sono i compiti che deve svolgere la routine DELETE? Sappiamo, dal volume 1, che premendo il tasto DELETE viene eliminata dalla memoria operativa dello Junior-Computer l'istruzione al momento visualizzata sul display. Le operazioni da compiere a tale scopo si possono riassumere in poche parole:

1. Sposta di uno, due o tre posizioni verso l'alto il blocco dati tra l'istruzione, che segue immediatamente l'istruzione visualizzata, ed il Pointer d'indirizzo finale corrente CEND.
2. L'entità esatta di tale spostamento è determinata dalla lunghezza dell'istruzione visualizzata al momento sul display.
3. L'istruzione attualmente visualizzata viene eliminata in quanto viene sovrascritta dal blocco; dati che viene spostato verso l'alto (vedi punto 1).

La subroutine UP esegue questo spostamento del blocco dati verso l'alto, cancellando per sovrascrittura l'istruzione visualizzata sul display. Sul display compare allora l'istruzione immediatamente seguente l'istruzione eliminata. Dato che per l'eliminazione di questa istruzione il file si accorcia di uno, due o tre byte, occorre spostare corrispondentemente verso l'alto il Pointer indirizzo finale corrente CEND. Questo compito è svolto dalla subroutine RECEND (= REduce Current END Address = diminuisci il Pointer CEND di un valore corrispondente alla lunghezza dell'i-

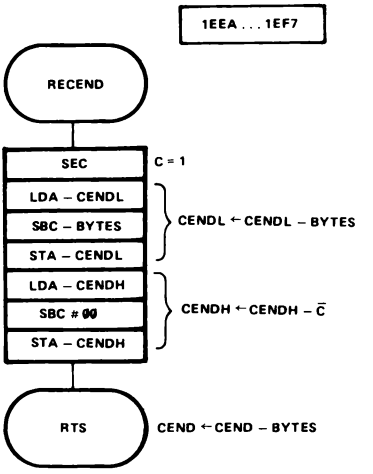


Figura 14. RECEND rappresenta l'inversione della subroutine ADCEND. Con essa il Pointer CEND viene spostato di uno, due o tre indirizzi verso l'alto.

struzione eliminata dal file). La fig. 14 mostra il diagramma di flusso dettagliato di RECEND. In questa subroutine il computer esegue una sottrazione di 16 bit, che costituiscono quindi l'inverso della subroutine ADCEND di fig. 13 (addizione di 16 bit). La subroutine RECEND sposta il Pointer CEND di una, due o tre locazioni verso l'alto, mentre la subroutine ADCEND sposta il Pointer CEND di altrettante locazioni verso il basso.
Abbiamo così imparato tutte e cinque le routine comandi dell'Editor. Manca solo l'illustrazione delle subroutine DOWN, UP ed OPLEN/LENACC. Queste subroutine assommano in sé l'intera intelligenza dell'Editor e dell'Assembler.

La Subroutine DOWN

La Sobroutine DOWN fa parte della già citata subroutine FILLWS. Quest'ultima viene richiesta per i comandi INSERT ed INPUT. Per questi si richiede, all'atto dell'introduzione di una nuova istruzione in memoria dello Junior-Computer, di creare dello spazio libero. Il computer deve cioé spostare un blocco dati di uno, due o tre bytes verso il basso entro il file. Tale funzione è assicurata dalla Subroutine DOWN.
In fig. 15 sono le singole istruzioni della subroutine DOWN; in fig. 16 è mostrato il funzionamento di tale subroutine. Per la subroutine DOWN il computer si avvale del Pointer indirizzi MOVAD (= MOve ADdress). Questo Pointer indica sempre un byte nel file che deve correntemente venire spostato verso il basso di uno, due o tre locazioni. Perciò le prime quattro istruzioni di DOWN eguagliano il Pointer MOVAD al Pointer CEND. Ossia, inizialmente MOVAD indica lo stesso indirizzo di CEND, cioé la locazione immediatamente seguente il carattere EOF 77. Quindi il computer giunge al Label DNLOOP. Con LDA - (MOVADL), Y si carica nell'Accu il contenuto della cella indicata da MOVAD (Y = 00). Di quante locazioni debba venire spostato verso il basso il byte appena introdotto dipende, come al solito, dal valore di BYTES. In BYTES è la lunghezza dell'istruzione che deve venir trasferita dal buffer di display nel file. Perciò si carica nel registro Y il valore di BYTES, e si memorizza il contenuto dell'Accu di un numero BYTES di locazioni più basso: STA - (MOVADL), Y. Segue quindi un confronto fra i valori dei due Pointer CURAD e MOVAD. Se MOVAD differisce ancora da CURAD significa che occorre spostare altri byte verso il basso nel file. Il computer, a partire dal Label DNA, diminuisce di 1 il valore di MOVAD, il che equivale a dire che MOVAD si muove di un posto verso l'alto rispetto al termine del file. (vedi anche fig. 16). Rientrato al Label DNLOOP, il computer sposta il byte indicato ora da MOVAD di uno, due o tre posti corrispondentemente verso il basso. Il processo descritto si ripete poi, partendo dal confronto fra MOVAD di uno, due o tre posti corrispondentemente verso il basso. Il processo descritto si

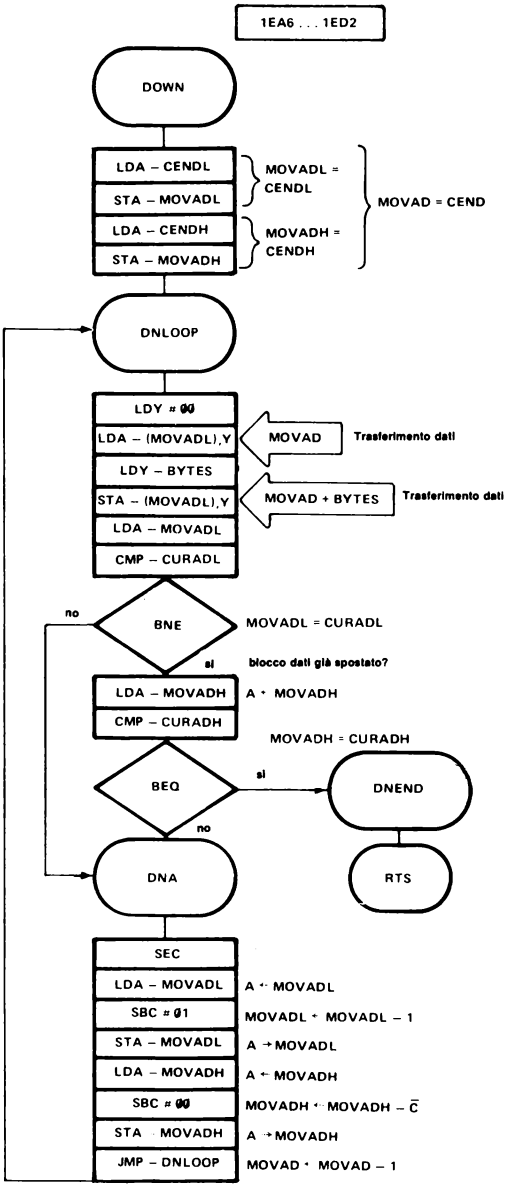


Figura 15. La subroutine DOWN sposta un intero blocco di dati di una, due o tre posizioni verso il basso. In tal modo viene fatto spazio per l'inserzione di una nuova istruzione nel file.

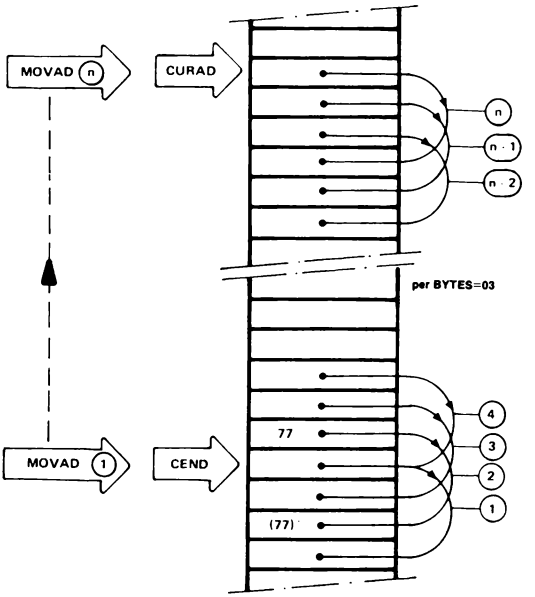


Figura 16. Come si svolge il meccanismo di spostamento della subroutine DOWN. Nell'esempio, un blocco dati completo del file viene spostato verso il basso di tre indirizzi, dato che il contenuto della locazione BYTES vale 03.

ripete poi, partendo dal confronto fra MOVAD e CURAD. Quando infine, dopo vari decrementi, MOVAD diventa eguale a CURAD, tutti i byte a partire dal Pointer di display CURAD sono stati spostati verso il basso. In tal modo dietro CURAD si è creato uno spazio di uno, due o tre byte nel quale è ora possibile copiare il contenuto dei buffer di display. La fig. 16 ci mostra graficamente il modo in cui la subroutine DOWN sposta verso il basso i singoli byte fra i Pointer CURAD e CEND. Il Pointer MOVAD indica in ogni caso il byte che deve essere correntemente trasferito dall'alto verso il basso. Dopo ogni spostamento verso il basso il computer decrementa di 1 il Pointer MOVAD. Il valore iniziale di MOVAD quindi è uguale a CEND, quello finale uguale a CURAD. In fig. 16 il contenuto di BYTES è posto uguale a 03. Dopo CURAD vengono quindi liberate tre locazioni di memoria, nelle quali è possibile scrivere un'istruzione lunga tre byte.
Nello Junior-Computer modello standard la massima lunghezza di un file consecutivo ammonta a 1/2KByte. Con file così corti la permanenza del computer nella subroutine DOWN è assai breve, dato che si tratta di spostare un numero limitato di byte.
Nel caso di espansione del computer, i file possono facilmente raggiungere lunghezze di due-quattro KByte. Quando in un tale file più lungo si vogliono inserire istruzioni con i comandi INSERT

od INPUT, specie all'inizio del file, lo Junior-Computer spende maggior tempo nella subroutine DOWN, che ammonta comunque a frazioni di secondo. Lo si riconosce dal fatto che il display resta spento per breve tempo.

La Subroutine UP

La subroutine UP non è altro che l'inversa della subroutine DOWN. Essa viene utilizzata dallo Junior-Computer quando:

* l'utente preme il tasto DELETE,
* nell'assemblaggio, quando vengono eliminati tutti i Label dal file (come descriveremo meglio nel capitolo seguente).

La fig. 17 presenta le singole istruzioni della subroutine UP. Questa subroutine si serve nuovamente del Pointer MOVAD, che indica sempre il byte nel file che deve venire spostato verso l'alto di uno, due o tre posizioni. La fig. 18 mostra qual'é il blocco dati che viene spostato dal computer dal basso verso l'alto.
Il computer sposta di uno, due o tre locazioni verso l'alto il blocco dati che principia a CURAD + BYTES e termina a CEND + BYTES. In tal modo l'istruzione il cui codice OP è indicato da CURAD viene cancellata, ossia sovrascritta.
Dopo il Label UP, il Pointer MOVAD viene posto eguale al Pointer di display CURAD. Dopo UPLOOP si carica nell'Accu il contenuto della cella indicata dal Pointer MOVAD + BYTES: LDA - (MOVADL), Y. Quindi il processore scrive il contenuto dell'Accu nella locazione indicata da MOVAD: STA - (MOVADL), Y (Y è 00). Il computer trasporta singoli byte del file di uno, due o tre posizioni verso l'alto, avendo caricato un byte dal file nell'Accu dall'indirizzo MOVAD + BYTES, e riscrivendolo poi nel file all'indirizzo MOVAD + BYTES - BYTES = MOVAD. Dopo questo spostamento, il Pointer MOVAD viene incrementato di 1. MOVAD si muove dunque entro il file dall'alto verso il basso. A partire dal Label UPA il computer confronta MOVAD con CEND. Se MOVAD non è eguale a CEND, vi sono ancora byte da spostare nel file dal basso verso l'alto: il programma salta al Label UPLOOP e svolge una nuova operazione di spostamento. Quando invece, dopo i veri incrementi, il Pointer MOVAD sia eguale al Pointer CURAD, non vi sono più spostamenti di byte da fare ed il computer esce dalla subroutine UP per tornare nel programma principale.

La Subroutine OPEN/LENACC

Un tratto della subroutine OPLEN lo abbiamo già conosciuto nel 1° volume, capitolo 4: è la subroutine LENACC. Nel corso di questa subroutine il computer calcola la lunghezza di una istruzione. Prima di richiamare la subroutine OPLEN bisogna che nell'Accu sia posto il codice OP dell'istruzione, perché solo in tal

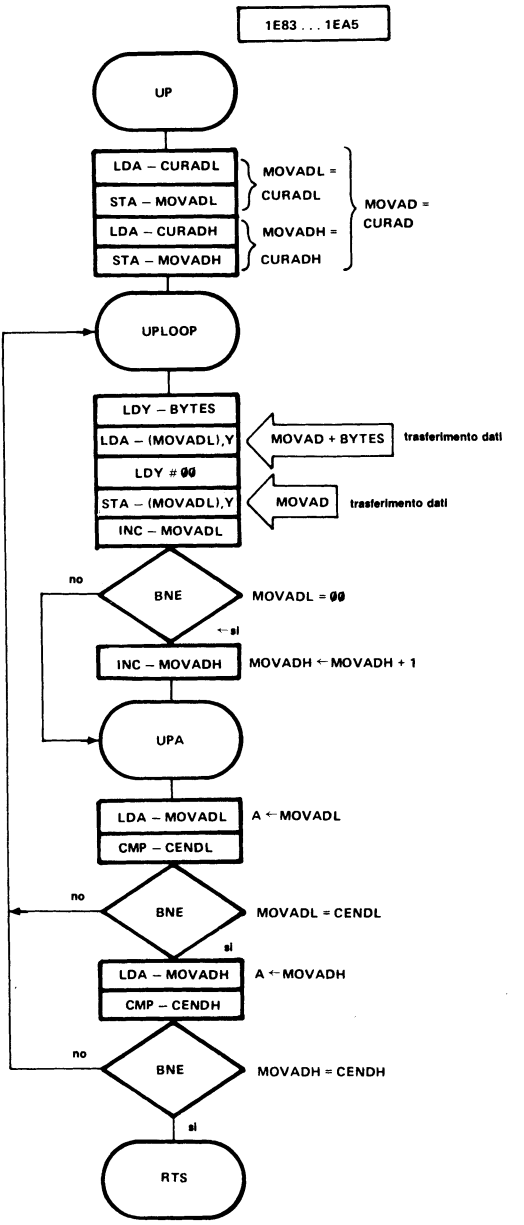


Figura 17. La subroutine UP è l'inversione della subroutine DOWN. UP esegue lo spostamento di un blocco di dati del file di uno, due o tre posizioni più in alto. Così si eliminano dal file Label od istruzioni. La subroutine UP è impiegata sia dall'Editor che dall'Assembler.

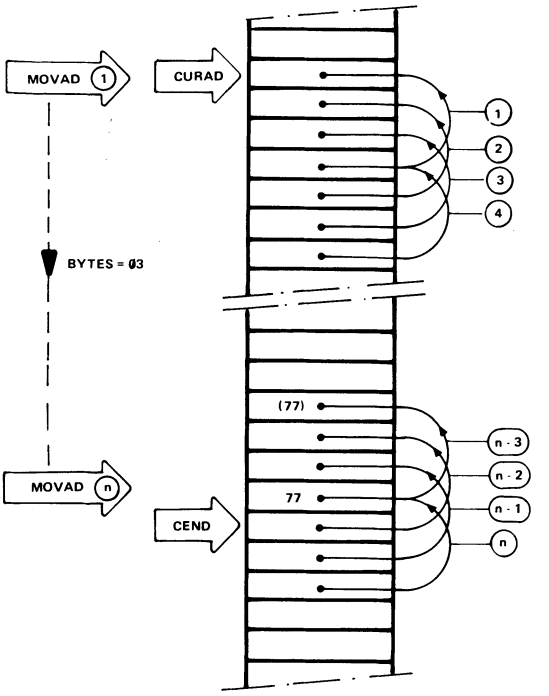


Figura 18. Come si svolge il meccanismo di spostamento della subroutine UP. Nell'esempio, un blocco dati del file viene spostato di tre indirizzi verso l'alto, dato che il contenuto della locazione BYTES vale 03.

caso può funzionare LENACC. OPLEN è in pratica un affinamento della subroutine LENACC: infatti al richiamo di OPLEN il codice OP dell'istruzione indicata dal Pointer di display CURAD viene caricato automaticamente nell'Accu.
Vi sono tuttavia alcune piccole differenze fra la subroutine LENACC dell'EPROM dello Junior-Computer e la subroutine LENACC descritta nel 1° volume, pag. 158. Per prima cosa, i due registri X ed Y sono scambiati fra loro: ciò non ha tuttavia conseguenze sul funzionamento di LENACC. Come sappiamo dal 1° volume, le lunghezze delle istruzioni della CPU sono memorizzate in una Lookup-Table, denominata LEN. Alcune posizioni di questa Lookup Table sono riempite da altri tipi di byte, dato che all'identificatore dei Label, FF, deve essere assegnata una lunghezza di 3 byte, ed al carattere EOF una lunghezza di 1 byte. La fig. 20 ci mostra in quali colonne dei codici OP si trovano l'identificatore dei Label ed il carattere EOF. Nelle colonne 7 ed F sono pure indicate le lunghezze assegnate a questi pseudo-codici. Non intendiamo ripetere qui in dettaglio la descrizione della subrouti-

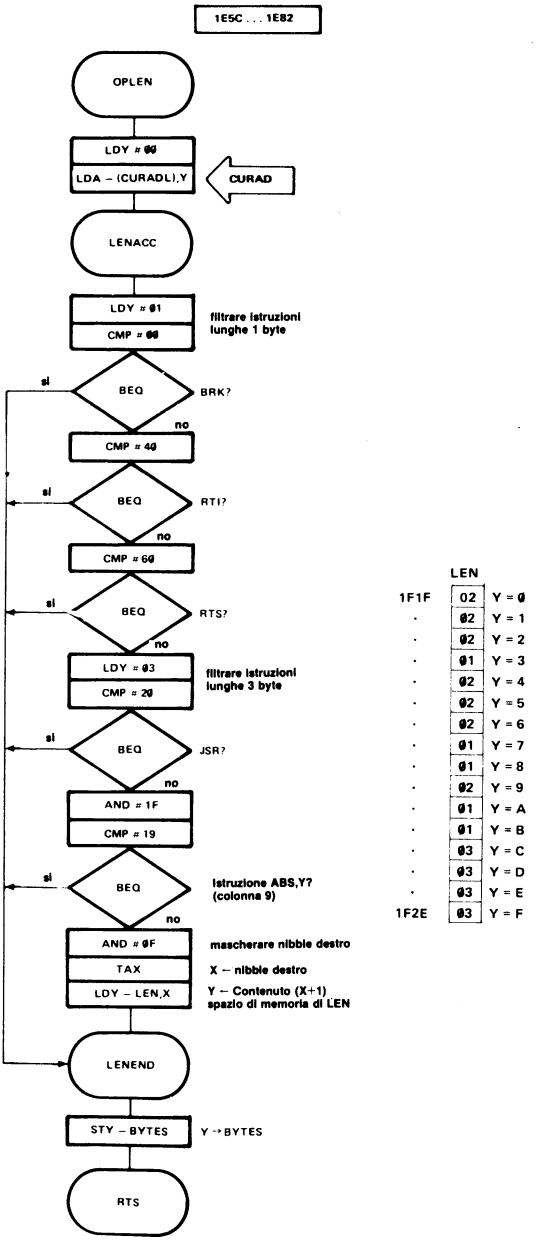


Figura 19. La subroutine OPLEN calcola la lunghezza dell'istruzione il cui codice OP è indicato dal Pointer di display CURAD. Il calcolo della lunghezza dell'istruzione principia dopo il Label LENACC.

nibble dati basso

nibble dati alto	0	1	2	3	4	5	6	7
0	BRK (1)	ORA (IND,X) (2)				ORA Z (2)	ASL Z (2)	
1	BPL (2)	ORA (IND),Y (2)				ORA Z,X (2)	ASL Z,X (2)	
2	JSR (3)	AND (IND,X) (2)			BIT Z (2)	AND Z (2)	ROL Z (2)	
3	BMI (2)	AND (IND),Y (2)				AND Z,X (2)	ROL Z,X (2)	
4	RTI (1)	EOR (IND,X) (2)				EOR Z (2)	LSR Z (2)	
5	BVC (2)	EOR (IND),Y (2)				EOR Z,X (2)	LSR Z,X (2)	
6	RTS (1)	ADC (IND,X) (2)				ADC Z (2)	ROR Z (2)	
7	BVS (2)	ADC (IND),Y (2)				ADC Z,X (2)	ROR Z,X (2)	EOF 77 (1)
8		STA (IND,X) (2)			STY Z (2)	STA Z (2)	STX Z (2)	
9	BCC (2)	STA (IND),Y (2)			STY Z,X (2)	STA Z,X (2)	STX Z,Y (2)	
A	LDY # (2)	LDA (IND,X) (2)	LDX # (2)		LDY Z (2)	LDA Z (2)	LDX Z (2)	
B	BCS (2)	LDA (IND),Y (2)			LDY Z,X (2)	LDA Z,X (2)	LDX Z,Y (2)	
C	CPY # (2)	CMP (IND,X) (2)			CPY Z (2)	CMP Z (2)	DEC Z (2)	
D	BNE (2)	CMP (IND),Y (2)				CMP Z,X (2)	DEC Z,X (2)	
E	CPX # (2)	SBC (IND,X) (2)			CPX Z (2)	SBC Z (2)	INC Z (2)	
F	BEQ (2)	SBC (IND),Y (2)				SBC Z,X (2)	INC Z,X (2)	

nibble dati basso

nibble dati alto	8	9	A	B	C	D	E	F	
0	PHP (1)	ORA # (2)	ASL A (1)			ORA ABS (3)	ASL ABS (3)		0
1	CLC (1)	ORA ABS,Y (3)				ORA ABS,X (3)	ASL ABS,X (3)		1
2	PLP (1)	AND # (2)	ROL A (1)		BIT ABS (3)	AND ABS (3)	ROL ABS (3)		2
3	SEC (1)	AND ABS,Y (3)				AND ABS,X (3)	ROL ABS,X (3)		3
4	PHA (1)	EOR # (2)	LSR A (1)		JMP ABS (3)	EOR ABS (3)	LSR ABS (3)		4
5	CLI (1)	EOR ABS,Y (3)				EOR ABS,X (3)	LSR ABS,X (3)		5
6	PLA (1)	ADC # (2)	ROR A (1)		JMP IND (3)	ADC ABS (3)	ROR ABS (3)		6
7	SEI (1)	ADC ABS,Y (3)				ADC ABS,X (3)	ROR ABS,X (3)		7
8	DEY (1)		TXA (1)		STY ABS (3)	STA ABS (3)	STX ABS (3)		8
9	TYA (1)	STA ABS,Y (3)	TXS (1)			STA ABS,X (3)			9
A	TAY (1)	LDA # (2)	TAX (1)		LDY ABS (3)	LDA ABS (3)	LDX ABS (3)		A
B	CLV (1)	LDA ABS,Y (3)	TSX (1)		LDY ABS,X (3)	LDA ABS,X (3)	LDX ABS,Y (3)		B
C	INY (1)	CMP # (2)	DEX (1)		CPY ABS (3)	CMP ABS (3)	DEC ABS (3)		C
D	CLD (1)	CMP ABS,Y (3)				CMP ABS,X (3)	DEC ABS,X (3)		D
E	INX (1)	SBC # (2)	NOP (1)		CPX ABS (3)	SBC ABS (3)	INC ABS (3)		E
F	SED (1)	SBC ABS,Y (3)				SBC ABS,X (3)	INC ABS,X (3)	Label FF (3)	F

Figura 20. La tabella dei codici OP della CPU 6502. Vi sono riportati pure lo pseudo-codice OP FF ed il carattere EOF.

ne OPLEN/LENACC: nel 1° volume si è già infatti descritto esaurientemente il modo di funzionamento di questa subroutine. Tuttavia, dato che essa troverà frequentemente impiego nel seguito, ne ricordiamo in breve il corso:

1. Quando viene richiamata la subroutine OPLEN, per prima cosa si carica nell'Accu il codice OP dell'istruzione indicata dal Pointer di display CURAD. Questo Pointer indica sempre un normale codice OP della CPU 6502, oppure un pseudo-codice come l'identificatore di Label FF od il carattere EOF.
2. Di seguito, dopo il Label LENACC, si carica 01 nel registro Y e quindi si provvede a "filtrare" eventuali istruzioni irregolari (vedi 1° vol, pag. 156), quali BRK, RTI ed RTS. Se nell'Accu è presente una di queste tre istruzioni, trasferisce il valore di Y nella cella di memoria BYTES, attribuendo cioé alle citate 3 istruzioni una lunghezza di 1 byte. Altra istruzione irregolare possibile è JSR: è lunga tre byte, e va pure "filtrata" separatamente. In corrispondenza si caricherà 03 nel registro Y trasferendo poi tale valore in BYTES, nel caso in cui nell'Accu sia presente il codice OP d'una istruzione JSR (= 20).

3. Se a questo punto nell'Accu non si trova un'istruzione irregolare tipo BRK, RTI, RTS e JSR, interessano solo i 5 bit inferiori del contenuto dell'Accu. L'istruzione successiva nella subroutine OPLEN è perciò: AND # 1F. Il registro Y vale ancora 03. Con l'istruzione seguente, CMP # 19, vengono "filtrate" tutte le istruzioni lunghe tre byte della colonna 9 di tabella (fig.) 20. Tali sono le istruzioni dotate di indirizzamento ABS, Y (ABSolute Indexed, Y Addressing; vedi volume 1, pag. 122).
4. Se nell'Accu non è presente alcuna istruzione dotata di ABS, Y Addressing, interessano solo gli ultimi quattro bit dell'Accu. L'istruzione successiva perciò è: AND # 0F. Il numero risultante viene impiegato quale indice per la Lookup Table LEN (TAX). Poi il computer ricava dalla Lookup Table la lunghezza dell'istruzione e la deposita ancora nella locazione BYTES. Le relative istruzioni sono: LDY-LEN, X e STY-BYTES.

E così abbiamo completato la conoscenza dell'Editor presente nella EPROM dello Junior-Computer. Le numerose subroutine dell'Editor dovrebbero costituire una grande semplificazione per lo sviluppo dei programmi: sono infatti facilmente inseribili in programmi "Do It Yourself". Il programmatore deve solo ricordare quali registri della CPU o quali locazioni di memoria debbono venire preventivamente caricati prima di richiamare le subroutine dell'Editor. Il "Source Listing" completo dell'Editor potrà essere consultato alla fine di questo volume in Appendice.

9

Il Programma Assembler

Anche il programma Assembler è contenuto nella EPROM dello Junior-Computer. Compito dell'Assembler è di elaborare ulteriormente un programma impostato mediante l'Editor affinché possa essere compreso dalla CPU 6502 (assemblaggio = adattamento). Quando si lavora con l'Editor, l'operando di una istruzione di salto incondizionato o condizionato è spesso un numero di Label, ossia un indirizzo simbolico. Anche i Label sono pseudo-istruzioni che il processore non è in grado di comprendere. Ecco quindi che il programma Assembler per "adattare" al processore un programma impostato deve svolgere questi compiti:

* **Tutti i Label devono essere eliminati dal programma.**
* **Per le istruzioni JMP e JSR gli indirizzi simbolici (numeri di Label) devono venire sostituiti dagli indirizzi effettivi.**
* **Per le istruzioni di salto condizionato gli offset simbolici (numeri di Label) devono venire sostituiti con gli offset reali.**

In questo capitolo descriveremo come l'Assembler opera su di un file introdotto mediante l'Editor.

Nel capitolo 5 abbiamo già citato i grandi vantaggi offerti dall'Editor e dall'Assembler. Impostando un programma non è più necessario conoscere gli indirizzi di partenza delle subroutine o gli offset delle istruzioni di salto condizionato. Al loro posto si impiegano i Label, indirizzi simbolici. L'introduzione dei Label in un file direttamente ad opera del computer consente un gran risparmio di tempo e di calcoli. Grazie all'Editor, è possibile inserire senza fatica nella memoria del computer programmi anche di grandi dimensioni. E dopo l'impiego dell'Assembler si può star certi che tutti gli offset e gli indirizzi di inizio delle subroutine sono corretti. Ci viene dunque risparmiato un laborioso e lungo ripercorrimento dei programmi in "Step by step Mode". Altro vantaggio offerto dall'Editor ed Assembler consiste nel compattamento massimo possibile dei programmi nella memoria dello Junior-Computer.

Il diagramma di flusso generale dell'Assembler

L'assemblaggio di un file introdotto mediante l'Editor avviene in due fasi. Nel capitolo 5 abbiamo perciò parlato di un "Two Pass Assembler" (= Assembler in due fasi). Corrispondentemente, e per renderlo più comprensibile, abbiamo diviso il diagramma di flusso globale in due figure: Fig. 1 per la prima fase dell'Assembler, Fig. 2 per la seconda fase.
Cominciamo con la Fig. 1! Dopo il Label ASSEMB, indirizzo iniziale 1F51, troviamo un tratto di programma che esegue tutto il lavoro di preparazione per l'assemblaggio. Tali preparativi consistono nell'assegnazione di determinati stati a diversi Pointer indirizzi.
Segue il Label PASSA, che costituisce il vero e proprio primo stadio (fase) dell'Assembler. In questa prima fase di assemblaggio lo Junior-Computer si occupa esclusivamente dei Label presenti nel file. Come effettua il computer il ritrovamento dei singoli Label nel file? È un compito relativamente facile, dato che per ogni istruzione presente nel file il computer è in grado di calcolare la relativa lunghezza. La subroutine necessaria la conosciamo: OPLEN. Per la ricerca dei Label il computer parte dall'inizio del file, e si chiede: il codice OP della prima istruzione è FF? In caso affermativo, si tratta di un Label; se no, è un normale codice OP. Se si tratta di un Label, ossia d'un istruzione col codice OP FF, lo Junior-Computer si "annota" l'indirizzo a cui è situato il pseudo codice OP FF, nonché il numero di Label, che segue immediatamente allo pseudo-codice OP FF. Ci si chiede ora: dove annota il computer l'indirizzo a cui è situato il pseudocodice OP FF ed il seguente numero di Label? L'indirizzo del pseudocodice OP e numero di Label vengono depositati dal computer sul cosiddetto Symbol-Stack. Questo non è altro che una sorta di raccoglitore di appunti, confrontabile al normale Stack in Pagina 1 dello Junior-Computer. L'unica differenza fra i due tipi di Stack è che lo Stackpointer dello Stack in Pagina 1 è governato dalla CPU (via Hardware), mentre lo Stackpointer del Symbol-Stack viene regolato da un programma (via Software). I due Stack hanno in comune il fatto che crescono dal basso verso l'alto. Ed ora torniamo alla prima fase del nostro Assembler!

Dopo che il computer ha rintracciato un Label nel file, e ne ha corrispondentemente deposto sul Symbol-Stack indirizzo del pseudocodice OP e numero di Label, il Label stesso deve venire eliminato dal file (tra BEGAD e CEND). Anche la subroutine che è capace di eliminare un Label ci è nota, dal comando DELETE: si tratta della subroutine UP. Dato che un Label è sempre lungo tre byte, il blocco dati che inizia dopo il Label e termina all'indirizzo indicato da CEND viene spostato di tre byte verso l'alto. In tal modo il Label risulta eliminato dal file e questo si è accorciato di tre byte. Si capisce subito che corrispondentemente si deve ag-

giornare anche il Pointer indirizzi CEND, che deve analogamente muoversi di tre posizioni verso l'alto (vedi capitolo 8, fig. 5 e fig. 18). Adesso, il codice OP dell'istruzione successiva al Label cancellato, si trova all'indirizzo a cui prima si trovava il pseudo-codice

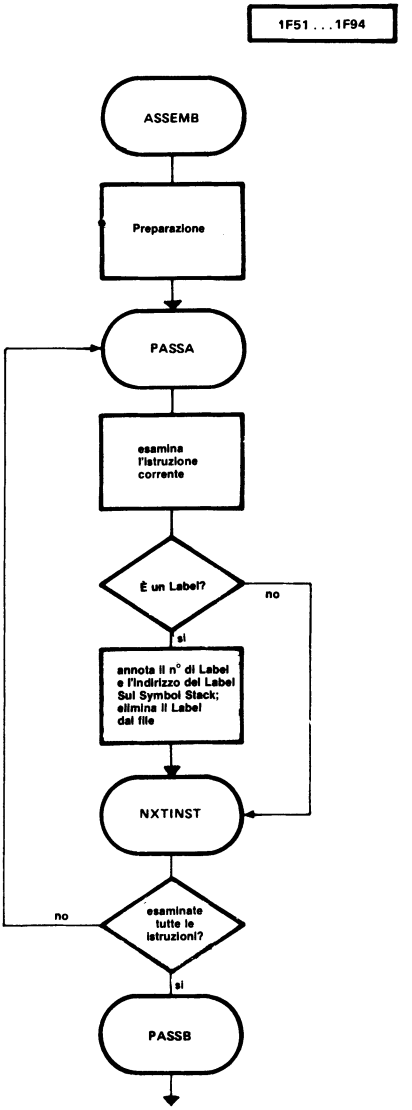


Figura 1. L'Assembler elabora il file introdotto mediante l'Editor in due fasi. Questa figura illustra il diagramma di flusso globale della prima fase dell'assemblaggio. In essa il computer provvede ad eliminare dal file tutti i Label, deponendo sul Symbol Stack tutti i relativi numeri di Label e gli indirizzi a cui si trovano i Label.

OP FF del Label. Il processo descritto risulta chiaramente dalle fig. 3a e 3b.
Eliminato il Label dal file, ed avendo memorizzato tutto quanto interessa il Label citato sul Symbol-Stack, lo Junior-Computer passa a ricercare un altro Label nel file. Salta perciò di istruzione in istruzione, verificando se essa abbia lo pseudocodice OP FF. Non appena rintraccia lo pseudocodice OP FF, si ripete l'intero procedimento:

1. Poni l'indirizzo a cui si trova lo pseudocodice OP FF sul Symbol-Stack (prima il byte indirizzo alto e poi quello basso).
2. Estrai dal file il numero di Label che segue immediatamente lo pseudocodice OP, e deponilo analogamente sul Symbol-Stack. Così gli elementi caratteristici del Label (il byte indirizzo alto e quello basso dell'indirizzo a cui si trovava lo pseudo codice OP del Label nonché il numero di Label) sono stati memorizzati sul Symbol-Stack, il blocco per appunti dello Junior-Computer. Più oltre ci occuperemo della struttura di questo Symbol-Stack.
3. Elimina il Label dal file. L'istruzione che seguiva il Label si trova ora all'indirizzo a cui era prima posto il Label stesso.
4. Ricerca eventuali altri Label nel file, e termina l'operazione di ricerca quando non trovi più Label nel file.

Consideriamo quindi nuovamente la fig. 3. In essa scorgiamo quattro tappe dell'operazione di ricerca di un Label nella prima fase di assemblaggio dei programmi:
Fig. 3a: All'indirizzo 0200 il computer trova lo pseudocodice di un Label. Dopo aver deposto sul Symbol-Stack tutto quello che occorre ricordare del Label citato, il Label viene eliminato dal file.
Fig. 3b: Il computer ha eliminato il Label dal file. Dato che un Label ha sempre una lunghezza di 3 byte, il blocco dati che segue il Label viene spostato di tre posizioni verso l'alto. All'indirizzo originale 0200 del Label viene ora a trovarsi il codice OP dell'istruzione che seguiva direttamente il Label. Il file è divenuto più corto di 3 byte, dopo l'eliminazione del Label.
Fig. 3c: Lo Junior-Computer prosegue la ricerca dei Label. Tutte le istruzioni, il cui codice OP è *diverso* da FF, non vengono considerate in tale ricerca. All'indirizzo 0217 il computer rintraccia nuovamente un Label, caratterizzato dallo pseudocodice FF. Tutte le caratteristiche di questo nuovo Label vengono anch'esse deposte sul Symbol Stack, e successivamente il Label viene eliminato dal file.
Fig. 3d: Il Label col n° 12 è stato cancellato dal file. Il blocco di dati che segue il Label viene nuovamente spostato verso l'alto di tre byte. Così all'indirizzo 0217, a cui era prima lo pseudocodice del Label, si trova ora il codice OP dell'istruzione che seguiva il Label.
In tal modo abbiamo appreso tutto quanto serve della prima fase dell'Assembling. Passiamo alla seconda fase (PASSB). In fig. 2

troviamo il diagramma di flusso globale di questa seconda fase. A questo punto sono già stati cancellati dal file tutti i Label. Subito dopo il Label PASSB lo Junior-Computer passa a controllare l'una dopo l'altra tutte le istruzioni, partendo dalla prima del

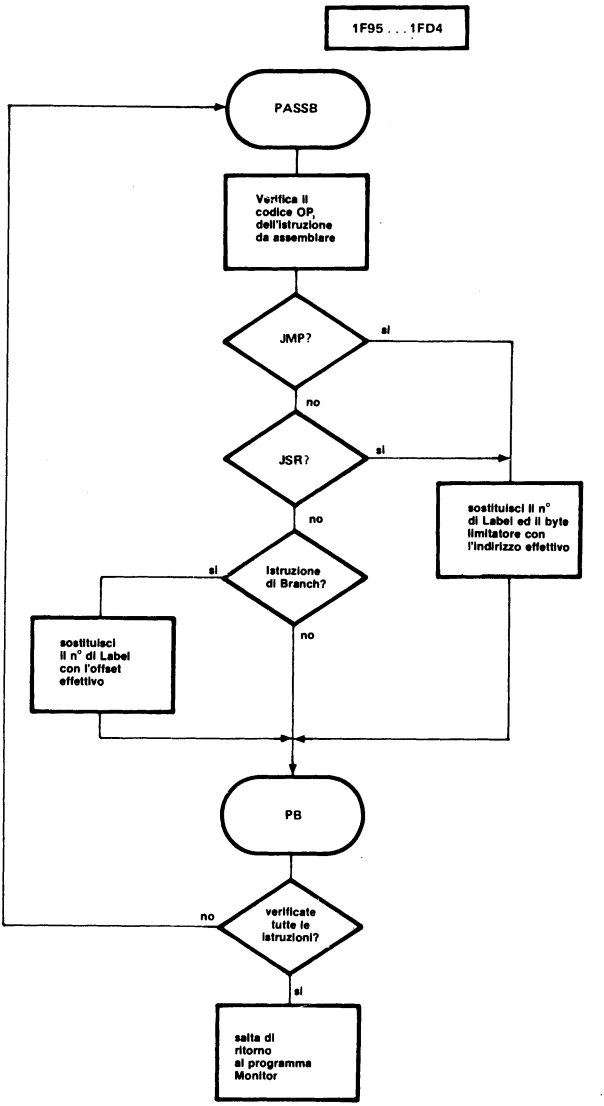


Figura 2. Il diagramma di flusso globale della seconda fase di assemblaggio. Dopo le istruzioni di salto incondizionato il n] di Label ed il byte limitatore vengono sostituiti dall'effettivo indirizzo di salto. Inoltre, dopo le istruzioni di salto condizionato (Branch), i numeri di Label vengono sostituiti in questa fase dai valori degli offset effettivi. In entrambi i casi il computer si appoggia al Symbol Stack.

file. Quali sono ora le istruzioni che interessano al computer? Si tratta delle istruzioni dopo il cui codice OP è posto un numero di Label. Simili istruzioni le conosciamo già dal capitolo 5:

* istruzioni JMP col codice OP 4C,
* istruzioni JSR col codice OP 20,
* istruzioni di Branch: BPL, BMI, BEQ, BNE, BCC, BCS, BVC e BVS.

Nella seconda fase di assemblaggio il computer deve assegnare a queste istruzioni gli indirizzi effettivi e gli offest. Perciò si deve provvedere a "filtrare" una dopo l'altra questi tipi di istruzioni dal file. Il computer le può riconoscere facilmente in base al loro codice OP:

1. *Il computer trova un'istruzione JMP:* dopo il codice OP 4C si trova un n° di Label. A questo n° di Label corrisponde un indirizzo che durante la prima fase di assemblaggio (PASSA), assieme al n° di Label del Label eliminato in tale fase dal file, è stato deposto sul Symbol-Stack. Pertanto, ora il computer cerca sul Symbol Stack un n° di Label corrispondente a quello posto dopo il codice OP 4C. Trovato questo numero di Label sullo Stack, il computer recupera anche l'indirizzo relativo a questo numero di Label, sempre dallo Stack, e pone questo indirizzo, lungo due byte, dopo il codice OP 4C. Il n° di Label ed il byte limitatore che originariamente erano posti dopo il codice OP 4C vengono così sovrascritti dall'effettivo indirizzo di salto. L'istruzione JMP col codice OP 4C è stata così assemblata.
2. *Il computer trova un'istruzione JSR:* dopo il codice OP 20, come nel caso dell'istruzione JMP, si trova un numero di Label. Per assemblare questa istruzione, il computer recupera dal Symbol Stack l'indirizzo assoluto relativo al n° di Label, e la dispone dopo il codice OP dell'istruzione JSR. Questa risulta così assemblata. Il procedimento è del tutto analogo a quello visto per l'istruzione JMP.
3. *Il computer trova un'istruzione di Branch (salto condizionato):* la CPU 6502 prevede come è noto pure istruzioni di Branch. Se il computer nella seconda fase dell'assemblaggio incontra il codice OP di un'istruzione di Branch, il n° di Label che segue tale codice OP deve venire sostituito dall'offset effettivo. L'assemblaggio di un'istruzione di Branch ha luogo nel modo seguente:

   * Il computer verifica quale numero di Label è posto dietro il codice OP dell'istruzione Branch. Per tale n° di Label è stato deposto sul Symbol Stack un indirizzo assoluto corrispondente.

   * Il computer ricerca sul Symbol Stack il n° di Label posto dietro il codice OP dell'istruzione Branch. Trovatala sullo Stack, risulta pure noto il relativo indirizzo assoluto.

   * Il computer sa inoltre l'indirizzo a cui si trova l'istruzione

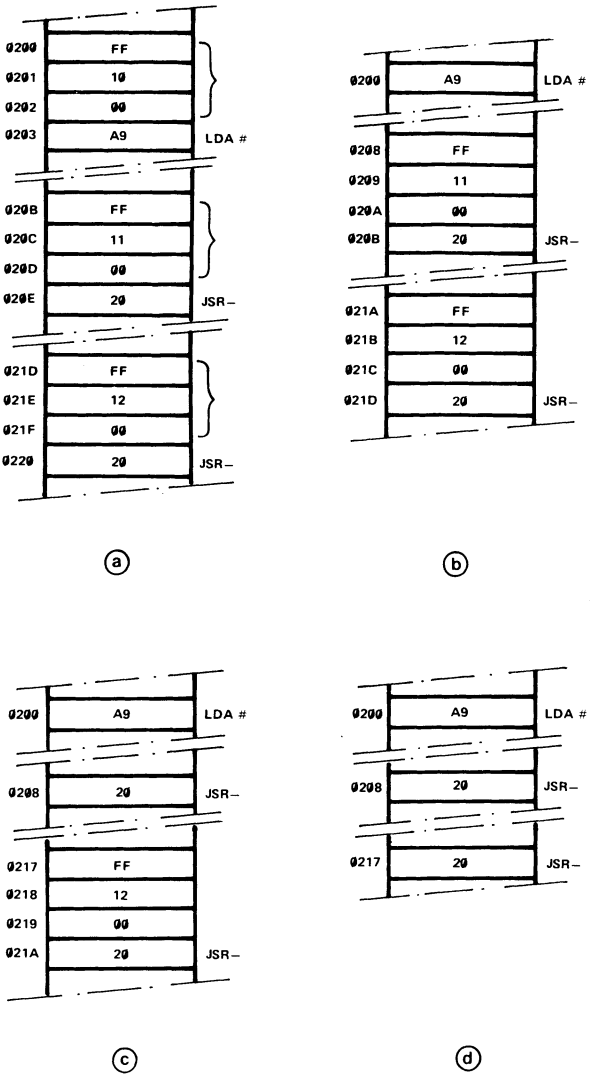


Figura 3. Prima della prima fase di assemblaggio, tutti i Label sono ancora presenti nel file. Tale situazione è illustrata in fig. 3a. Successivamente, il computer percorre l'intero file dall'alto fino in basso, eliminando tutti i Label. Tutte le informazioni sui Label vengono conservate dal computer sul Symbol Stack. La fig. 3b illustra la situazione creata dopo l'eliminazione del Label n° 10 dal file. Analogamente, la fig. 3c mostra il file dopo eliminato il Label n° 11 e la fig. 3d dopo eliminato il Label n° 12. I codici OP che seguivano i Label sono ora situati agli indirizzi a cui prima era posto lo pseudocodice FF.

Branch in corso di assemblaggio. Dai due indirizzi, quello relativo al n° di Label e quello a cui si trova il codice OP dell'istruzione di Branch, il computer è in grado di calcolare l'offset reale.
* Il computer inserisce l'offset così calcolato dopo l'istruzione di Branch da assemblare. Così pure questa è stata assemblata.

4. *Il computer trova un'istruzione JMP, JSR o di Branch, il cui numero di Label non si trova sul Symbol Stack:* dal capitolo 5 sappiamo che in determinate circostanze non si provvede ad assemblare un'istruzione JMP, JSR o Branch. È il caso in cui *non* si adopera il n° di Label dopo il codice OP quale n° di Label dopo l'identificazione di Label FF. In altri termini, quando il Label non è presente nel file. In tal caso naturalmente il computer, nella prima fase di assemblaggio (PASSA, vedi fig. 1), non è stato in grado di deporre sullo Stack il n° di Label ed il corrispondente indirizzo assoluto. Se il computer dunque incontra, nella seconda fase di assemblaggio, un'istruzione JMP, JSR o Branch il cui numero di Label non figura sul Symbol Stack, tale istruzione non viene assemblata: ossia, dopo il codice OP non vengono inseriti dati ricavati dal Symbol Stack. Per semplicità, in fig. 2 questo caso di mancato assemblaggio non è stato riportato.

Dovrebbe a questo punto risultare abbastanza chiaro ciò che avviene nel computer dopo il lancio dell'Assembler. L'assembler opera sul file impostato tramite Editor in due fasi. Nella prima fase di assemblaggio il computer elimina tutti i label dal file, e salva sul Symbol Stack i numeri di Label ed i relativi indirizzi assoluti. Nel corso di PASSA per il computer sono interessanti solo le istruzioni che cominciano col pseudocodice OP FF. Deposti tutti i numeri di Label e relativi indirizzi assoluti sul Symbol Stack, il computer passa alla seconda fase di assemblaggio (PASSB).
Nel corso della seconda fase di assemblaggio il computer si interessa solo dei codici OP 4C, 20 e degli 8 codici OP delle istruzioni di Branch. Dopo questi codici OP sono situati i numeri di Label (e, nel caso delle istruzioni JMP, JSR, i byte limitatori). Il computer ricerca ora nel Symbol Stack quel numero di Label che è posto dopo il codice OP dell'istruzione in corso di assemblaggio. Ad ogni n° di Label sul Symbol Stack corrisponde un indirizzo assoluto; tale indirizzo assoluto, nel caso delle istruzioni di salto incondizionato, viene inserito immediatamente dietro il codice OP. Nel caso dell'istruzione di Branch, invece, il computer calcola l'offset a partire da tale indirizzo. Sinora non abbiamo ancora considerato il Simbol-Stack e la sua organizzazione in dettaglio. Con le conoscenze ora acquisite, illustrando di seguito il diagramma di flusso dettagliato dell'Assembler, chiariremo pure il funzionamento del Symbol Stack.

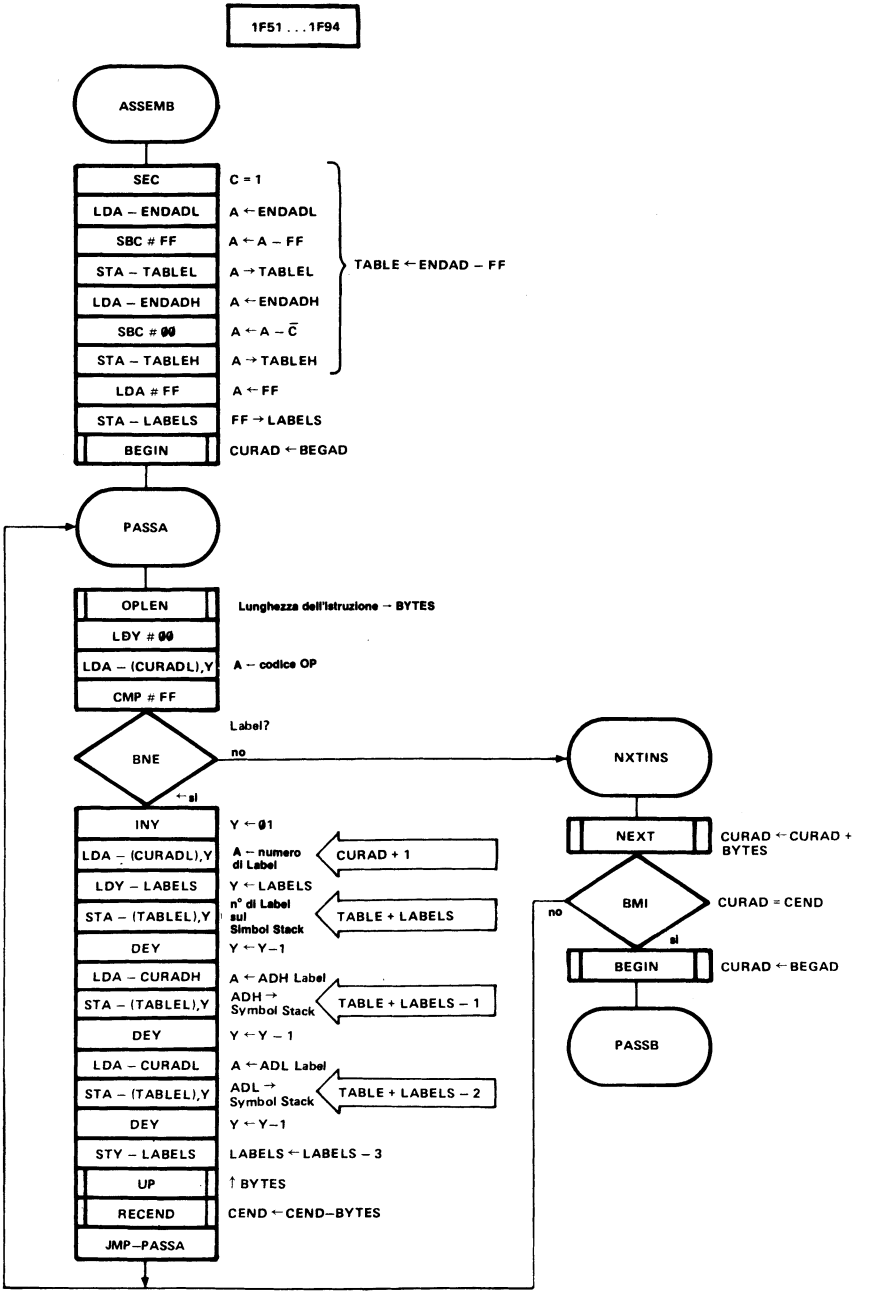


Figura 4. Il diagramma di flusso dettagliato della prima fase di assemblaggio.

Il diagramma di flusso dettagliato dell'Assembler

Possiamo dividere anche il diagramma di flusso dettagliato dell'Assembler in due parti (fasi). La fig. 4 illustra quella parte dell'Assembler che elimina i Label dal file e salva sul Symbol-Stack i numeri di Label ed i relativi indirizzi assoluti.
L'assembler parte dal Label ASSEMB d'indirizzo 1F51. Nel tratto successivo di programma sino al Label PASSA il computer dispone vari Pointer e predispone il funzionamento del Symbol Stack. Alcune osservazioni prima di entrare nei dettagli: il programma Assembler si collega strettamente col programma Editor. In particolare, l'Assembler impiega diverse subroutine dell'Editor. È perciò opportuno rileggere e ristudiare il contenuto del capitolo 8, se non si sono ancora completamente assimilate le operazioni delle varie subroutine nell'Editor.
L'Assembler utilizza un nuovo Pointer indirizzi: TABLE. Questo Pointer è posto nelle due celle di RAM di Pagina Zero TABLEH, TABLEL. Gli indirizzi del Pointer indirizzi TABLE sono:
TABLEL: indirizzo 00ED, e
TABLEH: indirizzo 00ED.
Il contenuto di queste celle di memoria è l'indirizzo indicato dal Pointer TABLE. Questo Pointer indirizzi fissa il limite superiore del campo di memoria in cui, dopo l'eliminazione dei Label dal file, sono poste tutte le informazioni necessarie su questi Label. TABLE, cioé, è l'indirizzo limite del Symbol Stack. L'altro indirizzo limite del Symbol Stack è TABLE + FF: pertanto la lunghezza

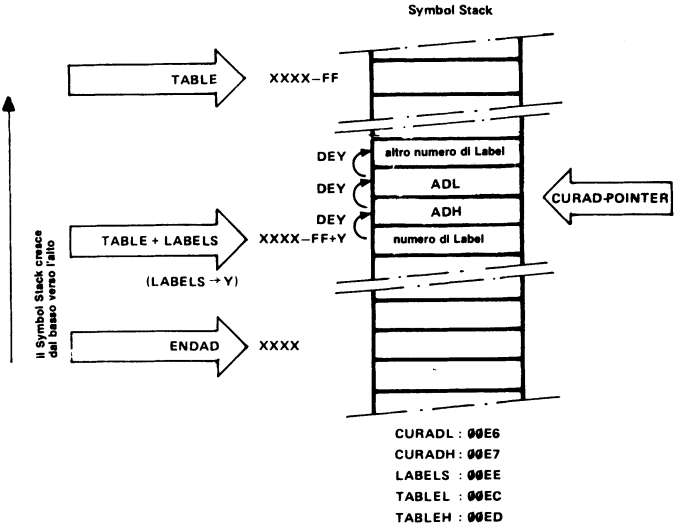
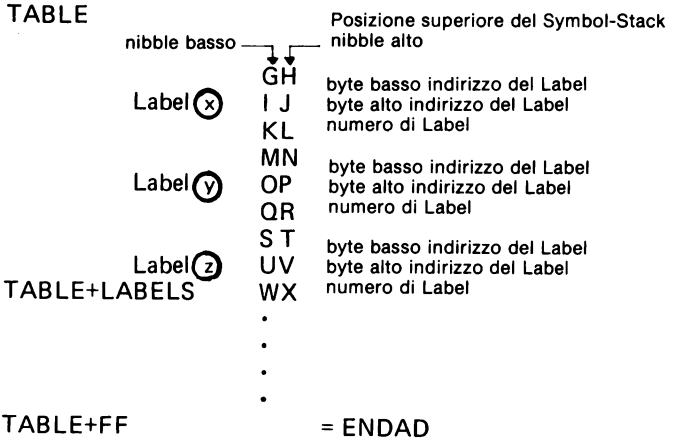


Figura 5. tramite il Pointer TABLE + LABELS lo Junior Computer dirige il traffico di dati da e verso i Symbol Stack. Il Symbol Stack cresce dal basso verso l'alto. Prima viene deposto il numero di Label, e poi il contenuto del Pointer di display CURAD.

massima del Symbol Stack ammonta a 256 byte.
Le prime 7 istruzioni successive al Label ASSEMB dispongono il Pointer indirizzi TABLE in modo che indichi un indirizzo distante 256 (FF) locazioni prima dell'indirizzo ENDAD. In fig. 5 è mostrato graficamente il modo in cui il computer fissa il Pointer indirizzi TABLE in funzione dell'indirizzo finale assoluto del file ENDAD. Oltre alla precedente, l'Assembler utilizza ancora un'altra cella di RAM, denominata LABELS, con indirizzo 00EE. Quando si somma il contenuto della locazione LABELS al contenuto dello Stack Pointer TABLE, il valore dell'indirizzo a cui il Symbol Stack Pointer correntemente puntava viene aumentato. Il nuovo indirizzo indicato dal Symbol Stack Pointer è quindi TABLE + LABELS. Durante la prima fase di assemblaggio il Symbol Stack Pointer indica sempre la locazione superiore non occupata del Symbol Stack. Quando nel corso della prima fase viene ritrovato un Label, nella locazione superiore non occupata del Symbol Stack viene posto il n° di Label; sopra di questo il byte alto indirizzo del Label correntemente assemblato, e sopra questo il byte basso indirizzo del Label.
L'organizzazione del Symbol Stack quindi è la seguente:



Così abbiamo detto tutto sulla struttura del Symbol Stack. Dopo aver disposto tutti i Pointer importanti per l'Assembler, il Symbol Stack indica il medesimo indirizzo del Pointer indirizzi ENDAD. (vedi pure fig. 5). Immediatamente prima del Label PASSA viene richiamata la subroutine BEGIN, che è nota dal capitolo 8 (fig. 7). Dopo percorsa questa subroutine, il Pointer di display CURAD indica il medesimo indirizzo del Pointer BEGAD, che, come sappiamo, è l'indirizzo d'inizio del file.

PASSA

Ora il computer è pervenuto al Label PASSA. A partire da questo Label inizia la prima fase di assemblaggio. Dapprima, il programma salta alla subroutine OPLEN (capitolo 8, fig. 19). Questa subroutine calcola la lunghezza di un'istruzione o pone il risultato nella locazione di memoria BYTES. In questo modo si sa di quante posizioni occorre spostare il Pointer di display CURAD quando si dovrà esaminare l'istruzione seguente.
L'istruzione LDA - (CURADL), Y, con Y = 00, carica nell'Accu il codice OP dell'istruzione correntemente indicata dal Pointer CURAD. Le due istruzioni seguenti, CMP # FF e BNE, verificano se si tratta dello pseudo-codice di un Label, oppure no. Se non è lo pseudo codice OP FF, il programma salta (fig. 4) al Label NXTINS (= NeXT INStruction). Nella subroutine NEXT (capitolo 8, fig. 10) il computer dispone il Pointer di display CURAD sull'istruzione successiva, sommando al precedente valore di CURAD il contenuto della cella BYTES. Nel corso della subroutine NEXT si verifica pure se il Pointer CURAD, che indica sempre il codice OP d'una istruzione, punta sempre nel file fra BEGAD e CEND, ovvero ha già superato il limite CEND. Se si trova ancora fra BEGAD e CEND, il computer deve nuovamente verificare la successiva istruzione per la presenza dello pseudo codice OP FF, e salta quindi ancora al Label PASSA. Viene calcolata la lunghezza dell'istruzione seguente con la subroutine OPLEN, ed il computer verifica se si tratta dello pseudocodice OP FF. L'intero procedimento viene insomma replicato.
Quando però il computer trova effettivamente lo pseudocodice OP FF, non viene effettuato, dopo l'istruzione CMP # FF, il salto BNE-NEXTINS, bensì ci si occupa del Label rinvenuto nel file. Per vedere come ciò avviene nei particolari, riprendiamo in esame le fig. 4 e 5. La fig. 4 illustra la prima fase dell'assemblaggio e la fig. 5 un estratto del *Symbol-Stack*:

1. Mediante l'istruzione INY si incrementa di 1, da 00 a 01, il contenuto del registro Y. L'istruzione seguene, LDA - (CURADL), Y carica nell'Accu il contenuto della locazione indicata da CURAD + 1. In tal modo il n° di Label che seguiva lo pseudocodice OP è ora nell'Accu.

2. Il contenuto della cella LABELS viene trasferito nel registro Y. La successiva istruzione STA - (TABLE), Y pone il n° del Label correntemente in esame sul Symbol Stack. L'indirizzo a cui viene deposto sul Symbol Stack il n° di Label è quello indicato dal corrente Symbol-Stack Pointer, TABLE + LABELS. Dato che alla partenza dell'Assembler il contenuto della cella di memoria LABELS è = FF, mentre a questo punto il Pointer TABLE segna l'indirizzo ENDAD-FF, il 1° numero di Label viene deposto sul Symbol Stack all'indirizzo indicato dal

Pointer indirizzo terminale ENDAD. Questa locazione, ed altre cinque, come ricordiamo dal capitolo 5, erano state appositamente riservate per l'Assembler, affinché nessuna parte del programma impostato tramite l'Editor venisse sovrascritta dal Symbol Stack.

3. DEY
   LDA-CURADH
   STA-(TABLEL), Y
   Lo Junior-Computer ha rintracciato nel file un Label caratterizzato dallo pseudocodice OP FF. Il Pointer di display CURAD indica ancora lo pseudo codice OP del Label. Dopo aver decrementato il registro Y, il byte alto d'indirizzo del Pointer di display CURAD viene caricato nell'Accu. Questo byte alto dell'indirizzo, a cui è posto lo pseudocodice OP del Label è qui costituito dal contenuto della cella CURADH. Il computer depone questo byte alto indirizzo del Label sul Symbol Stack. L'indirizzo di Symbol Stack a cui è stato collocato il byte alto d'indirizzo del Label è TABLE + LABELS - 1. Il byte alto dell'indirizzo del Label è dunque posto una posizione oltre il n° di Label precedentemente memorizzato.

4. DEY
   LDA-CURADL
   STA-(TABLE), Y
   Ora il computer carica nell'Accu il byte basso d'indirizzo del Pointer di display CURAD, che viene poi deposto sul Symbol Stack. L'indirizzo di Symbol Stack a cui viene collocato tale byte basso dell'indirizzo del Label è TABLE + LABELS - 2. Il byte basso d'indirizzo del Label è dunque posto una posizione oltre il corrispondente byte alto.

5. DEY
   STY-LABELS
   Il registro Y a questo punto è stato decrementato successivamente per tre volte. Ossia, il Pointer del Symbol Stack si è corrispondentemente spostato di tre posizioni verso l'alto. TABLE + LABELS indica ora l'indirizzo nel Symbol Stack a cui verrà deposto il prossimo n° di Label, se nel file esistono ancora altri Label con lo pseudocodice OP FF.

6. JSR-UP
   JSR-RECEND
   Tutte le informazioni relative al Label sono state salvate sul Symbol Stack. Il Label stesso è ora superfluo, e può venire eliminato dal file posto fra BEGAD e CURAD. L'eliminazione del Label è assicurata dalla nota subroutine UP (capitolo 8, fig. 17). Questa subroutine sposta verso l'alto di tre posti il blocco di dati del file che inizia dall'istruzione successiva al Label e termina a CEND. Il file si è accorciato di tre byte dopo

questa operazione di spostamento; il Pointer d'indirizzo finale corrente CEND (capitolo 8, fig. 14), nella successiva subroutine RECEND, viene analogamente aggiornato tre posizioni più in alto.

7. JMP-PASSA
   Lo Junior-Computer esamina ora le restanti istruzioni del file verificando la presenza dello pseudocodice OP FF. Continua cioé la ricerca dei Label. Se il computer trova nel file un altro Label, trasferisce al solito n° di Label ed indirizzo del Label sul Symbol Stack, e poi provvede ad eliminare il Label dal file. Se l'istruzione esaminata non corrisponde ad un Label, il programma Assembler (fig. 4) salta al Label NEXTINS, proseguendo di istruzione in istruzione sin quando rintraccia un Label. Nel corso della subroutine NEXT, che dispone ogni volta il Pointer di display CURAD sul codice OP dell'istruzione successiva, il computer controlla che CURAD non abbia oltrepassato CEND. Se il Pointer CURAD ha raggiunto la fine del file, è terminata la prima fase dell'assemblaggio e lo Junior-Computer, dopo la subroutine BEGIN, giunge al Label PASSB. Inizia la seconda fase dell'assemblaggio.

Dal capitolo 5 sappiamo che alla fine di un programma ENDAD e CEND devono restare liberi. Cioé, se non si è sicuri che il programma impostato sia risultato troppo lungo, occorre verificare se CEND sta almeno 6 posizioni di memoria sopra il Pointer ENDAD. È facile controllare, leggendo il contenuto del Pointer in Pagina 0, il progredire del Pointer CEND:
CENDL ha indirizzo 00E8
CENDH ha indirizzo 00E9; e
ENDADL ha indirizzo 00E4
ENDADH ha indirizzo 00E5.
Qual'é il motivo della distanza di 6 locazioni citata sopra? Nel corso della prima fase di assemblaggio, come ripetutamente detto, i numeri di Label e gli indirizzi assoluti dei Label vengono posti sul Symbol Stack. In totale si tratta di 3 locazioni che vengono occupate per ogni Label. Il Symbol Stack principia all'indirizzo indicato dal Pointer ENDAD. Una volta eliminato il Label dal file, il file stesso si accorcia di tre byte, ed il Pointer CEND si muove di tre byte verso l'alto. Se viene eliminato un ulteriore Label c'é apparentemente di nuovo posto sul Symbol Stack per deporvi il n° di Label e l'indirizzo assoluto del Label. Ci si chiede dunque: perché necessitano sei posizioni, quando sembra che ne bastino tre? Per rispondere, riconsideriamo la subroutine UP nel capitolo 8, fig. 17. Questa subroutine elimina i Label dal file. Un Label, lo sappiamo, è lungo tre byte. Eliminando un Label dal file, perciò, il blocco dati che segue il Label viene spostato di tre byte l'alto. Il procedimento in questione è mostrato in fig. 18 del capitolo 8: spostando di tre byte verso l'alto il blocco di dati, la subroutine UP trasporta verso l'alto anche altri tre byte posti dietro il Pointer

CEND. Questi tre byte sono superflui per l'Assembler. Se quindi si lasciassero liberi fra i due Pointer CEND ed ENDED solo tre posizioni di memoria, verso la fine del file gli ultimi dati che precedono CEND verrebbero sovrascritti da altri dati.
La fig. 6 mostra un estratto del Symbol Stack come si presenterebbe dopo la prima fase di assemblaggio del programma del capitolo 5, fig. 1...4. Tutti e sette i Label sono stati posti sul Symbol Stack con n° di Label e relativi indirizzi assoluti. È facile vedere come il Symbol Stack si sia riempito progressivamente di dati dal basso verso l'alto. Il Label che si trova al "fondo" dello Stack è quello deposto per primo e per primo eliminato dal file. Il suo numero è 10, ed era posto all'indirizzo 0200. Il Label assemblato per ultimo ha n° di Label 16 e si trovava all'indirizzo 024C. Nel file, esso era l'ultimo Label prima del carattere EOF, e si trova quindi sopra tutti gli altri sul Symbol Stack. A questo punto il Pointer del Symbol Stack TABLE + LABELS indica 02EA.

PASSB

Di seguito al Label PASSB inizia la seconda fase di assemblaggio. La fig. 7 ne mostra il diagramma di flusso di dettaglio. Inizialmen-

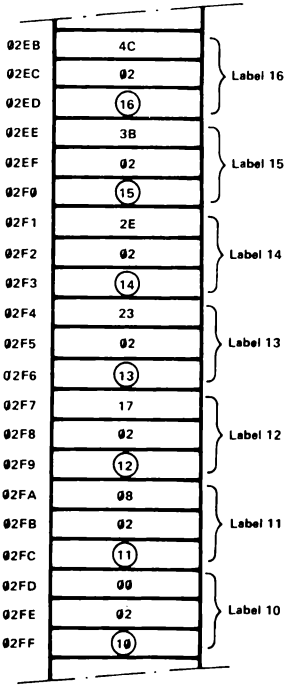


Figura 6. Dal capitolo 5 risulta già noto come si lavora con l'Editor e l'Assembler. La figura mostra come si presenta il Symbol Stack dopo la prima fase di assemblaggio, per l'esempio citato.

te, il Pointer di display CURAD è uguale al Pointer indirizzi BEGAD. L'eguagliare questi due Pointer è stata l'ultima operazione della prima fase di assemblaggio (fig. 5). Durante la seconda fase dell'assemblaggio, il computer verifica nuovamente tutte le istruzioni che sono ora presenti nel file. Nella prima fase il computer si interessava esclusivamente dei Label, caratterizzati dallo pseudocodice OP FF. Nella seconda fase i Label sono stati eliminati, ed il computer concerta il suo interesse esclusivamente a quelle istruzioni che richiedono di essere assemblate. Si tratta delle istruzioni dopo il cui codice OP sta un numero di Label:

* istruzioni JMP
* istruzioni JSR, e
* istruzioni di Branch (salto condizionato).

Dopo tali istruzioni si possono però trovare anche indirizzi assoluti od offset. Ad ogni modo, nella seconda fase di assemblaggio, i salti incondizionati e quelli condizionati vanno trattati distintamente. A tal fine valgono le seguenti considerazioni:
Come per PASSA (fig. 4), pure PASSB principia richiamando la subroutine OPLEN. In tal modo si viene a conoscere di quante posizioni si debba spostare verso il basso CURAD, per indicare il codice OP dell'istruzione seguente. L'istruzione LDA - (CURADL), Y carica - con Y = 00 - nell'Accu il codice OP dell'istruzione in esame. Quindi il computer provvede a "filtrare" le istruzioni JMP e JSR nonché quelle di Branch:

1. CMP # 4C
   BEQ separa l'istruzione di salto incondizionato JMP (codice OP = 4C)
2. CMP # 20
   BEQ separa l'istruzione di salto incondizionato JSR (codice OP = 20)
3. AND 1F
   CMP # 10
   BEQ separa tutte le istruzioni con i codici OP 10, 30, F0, D0, 90, B0, 50 e 70, a cui corrispondono nell'ordine le istruzioni BPL, BMI, BEQ, BNE, BCC, BCS, BVC e BVS. Per il mascheramento con 1F i 5 bit inferiori dell'Accu sono rimasti invariati, mentre i 3 bit più alti sono divenuti = 0. Se si considerano i codici OP delle istruzioni di salto condizionato si riconosce che il nibble basso nel codice OP è sempre zero. Perciò il mascheramento con 1F del codice OP di una qualsiasi istruzione di Branch porta sempre al risultato 10. Confrontando, con CMP # 10, il computer è così in grado di stabilire se il codice OP in esame è quello di un'istruzione di Branch oppure no.

Se non si tratta di un istruzione di salto o di Branch, il computer non deve assemblarla. Pertanto il computer, dopo il Label PB, torna ad occuparsi dell'istruzione successiva nel file. Il richiamo della subroutine NEXT dispone il Pointer di display CURAD sul

codice OP dell'istruzione successiva. Se il computer non è ancora giunto alla fine del file, il programma Assembler torna al Label PASSB e, dopo aver calcolato la lunghezza della nuova istruzione

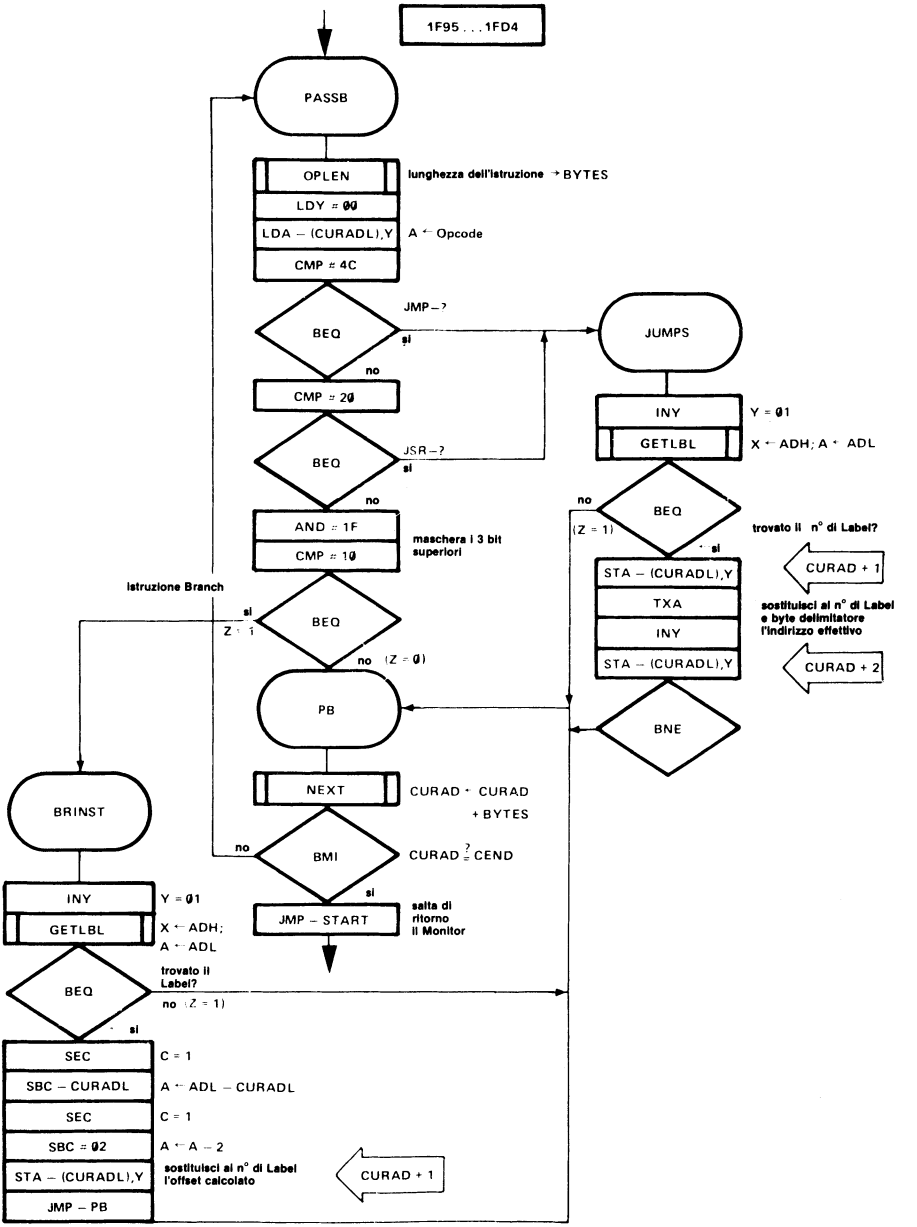


Figura 7. Il diagramma di flusso dettagliato della seconda fase di assemblaggio.

(OPLEN), si occupa dell'istruzione successiva. Si rinnova quindi interamente il procedimento già descritto.
Se invece si tratta di un'istruzione di salto o di Branch, esse devono venire assemblate. Ossia, il computer deve disporre dopo i codici OP di tali istruzioni l'effettivo indirizzo assoluto dei salti. Per le istruzioni di branch (salti condizionati) inoltre debbono venir calcolati gli offset relativi, ed inseriti dopo i codici OP. La sezione seguente spiega come il computer svolge questi compiti con l'ausilio del Symbol Stack.

Istruzioni JMP e JSR

A partire dal Label JUMPS vengono assemblate tutte le istruzioni di salto incondizionato. A tal fine sul Symbol Stack viene rintracciato l'indirizzo effettivo di salto, collocandolo poi come byte operando dietro il codice OP dell'istruzione di salto. Dopo aver incrementato il registro Y (Y è ora = 01), si passa alla subroutine GETLBL. Questa subroutine, illustrata in fig. 8, verrà descritta più oltre. Per ora basti sapere quanto segue:

1. La subroutine GETLBL ricerca sul Symbol Stack il numero di Label che si trova dopo il codice OP dell'istruzione in corso di assemblaggio. Se il n° di Label posto dopo il codice OP dell'istruzione di salto è effettivamente presente sul Symbol Stack, il computer è allora anche in grado di determinare l'indirizzo che corrisponde al dato n° di Label.
2. L'indirizzo corrispondente al n° di Label è anch'esso recuperato dal computer sul Symbol Stack. Questo indirizzo ha 16 bit, ossia è lungo due byte. GETLBL sistema questi due byte indirizzo prima del rientro dalla subroutine in due registri interni della CPU:
   * il byte alto indirizzo del Label, dopo il rientro dalla subroutine, è situato nel registro X,
   * il byte basso indirizzo del Label, dopo il rientro dalla subroutine, è situato nell'Accu della CPU.
3. Se il computer ha rintracciato sul Symbol Stack il n° di Label posto dopo il codice OP dell'istruzione in corso di assemblaggio, esso rientra dalla subroutine GETLBL con il Flag Z = 1.
4. Se il computer non rintraccia sul Symbol Stack il n° di Label posto dopo il codice OP dell'istruzione in corso di assemblaggio, esso rientra dalla subroutine GETLBL con il Falg N = 0.

Supponiamo per il momento che il computer abbia ritrovato, nella subroutine GETLBL, il desiderato n° di Label sul Symbol Stack. Esso rientra al programma principale con i byte alto e basso indirizzo del Label memorizzati rispettivamente nel registro Y e nell'Accu della CPU. Il Flag Z è alto (= 1), ed alla successiva istruzione BEQ il salto al Label PB non ha luogo. Il registro Y vale sempre 01. L'istruzione STA - (CURAD), Y pone il byte basso

indirizzo del Label dietro il codice OP dell'istruzione in fase di assemblaggio. Poi il byte alto indirizzo viene copiato dal registro Y nell'Accu, ed il registro Y viene incrementato di 1. La successiva istruzione STA - (CURADL), Y scrive l'indirizzo effettivo del salto nel file.
Prima dell'assemblaggio l'istruzione di salto era presente nel file nella forma: Cod. OP XX 00. Il codice OP è 20 o 4C; XX rapresenta un qualsiasi n° di Label. Il byte limitatore è costituito da 00.
Dopo l'assemblaggio l'istruzione di salto risulta posta nel file con i corretti valori degli operandi: Cod. OP ADL ADH. Il codice OP non ha subito variazioni. Ma i due byte operandi ADL (byte basso d'indirizzo del Label = byte basso dell'indirizzo assoluto di salto) ed ADH (byte alto d'indirizzo del Label = byte alto dell'indirizzo assoluto del salto) hanno sovrascritto i precedenti valori XX e 00. L'intera istruzione di salto risulta così assemblata. Dopo il codice OP dell'istruzione di salto assemblata sta il corretto operando: l'indirizzo assoluto di salto ADL, ADH, recuperato dal computer sul Simbol Stack.

Istruzioni di Branch (salti condizionati)

Quando il computer nel corso della seconda fase di assemblaggio rintraccia un'istruzione di Branch nel file, deve calcolarne il relativo offset. Inoltre deve eliminare il numero di Label, posto dopo il codice OP dell'istruzione Branch, dal file. Le istruzioni di salto condizionato vengono assemblate a partire dal Label BRINST. Dopo aver incrementato il registro Y (Y vale ora 01), si richiama la subroutine GETLBL. In questa subroutine il computer recupera dal Symbol Stack l'indirizzo assoluto del Label a cui deve aver luogo il salto. Se il computer rintraccia sul Symbol Stack il n° di Label posto dopo il codice OP dell'istruzione in corso di assemblaggio, esso rientra nel programma principale dell'Assembler dalla subroutine GETLBL con il Flag Z posto a 0. In questo caso, naturalmente, i byte alto e basso d'indirizo del Label sono stati posti nel registro X e nell'Accu. Come fa ora il computer per calcolare l'effettivo valore di offset dell'istruzione di Branch e disporlo poi dietro il codice OP? Per rispondere a questa domanda, vediamo quali sono le informazioni che a questo punto il computer conosce sull'istruzione di Branch in corso di assemblaggio:

1. Al rientro dalla subroutine GETLBL il computer sa qual'é l'indirizzo a cui deve portare il salto condizionato (Branch). Questo indirizzo di Label si trova nel registro X e nell'Accu della CPU. I due byte indirizzo posti nella CPU indicano l'*indirizzo di arrivo*, a cui deve portare il salto.
2. Tramite il Pointer di display CURAD il computer sa qual'é l'indirizzo a cui si trova il codice OP dell'istruzione in corso di assemblaggio. Questo è l'*indirizzo di partenza* da cui deve

aver luogo il salto. L'indirizzo di partenza è contenuto nel Pointer di display CURAD, situato nelle locazioni 00E6 e 00E7 in Pagina Zero.

3. In base ai valori degli indirizzi di arrivo e di partenza il computer può calcolare l'offset effettivo:
   Offset = indirizzo di arrivo - indirizzo di partenza - 02.
   Si deve ancora sottrarre 02 dalla differenza fra i due indirizzi perché il contatore di programma della CPU dopo aver decodificato un'istruzione di Branch indica il codice OP dell'istruzione seguente. Calcolato così l'offset, questo deve ancora essere inserito dopo il codice OP dell'istruzione in corso di assemblaggio. L'istruzione STA - (CURADL), Y che segue il Label BRINST fa sì che il valore dell'offset venga sovrascritto al n° di Label posto dopo il codice OP dell'istruzione Branch. Anche questa è stata così assemblata completamente.

A questo punto conosciamo perfettamente il programma principale dell'Assembler. Questo programma elabora ulteriormente un file impostato mediante l'Editor, in due fasi:
1.a fase: sul Symbol Stack vengono posti gli indirizzi assoluti dei Label con i relativi numeri di Label. Successivamente il computer elimina tutti i Label dal file, spostando per ciascuno di essi il blocco dati di tre posizioni verso l'alto. In tale operazione naturalmente anche i Label ancora presenti nel file vengono spostati di tre posizioni verso l'alto. Perciò, nel corso della prima fase di assemblaggio, i codici OP dei Label non ancora cancellati cambiano continuamente indirizzo. Le informazioni relative ai Label già eliminati dal file sono contenute nel Symbol Stack. Gli indirizzi dei Label che sono già stati posti sul Symbol Stack non vengono modificati nel corso ulteriore dell'assemblaggio.
2.a fase: dopo l'eliminazione di tutti i Label dal file e la conservazione di tutte le informazioni relative a questi Label sul Symbol Stack, inizia la seconda fase dell'assemblaggio. Nel corso di essa il computer verifica tutte le istruzioni dopo il cui codice OP è presente un n° di Label. A tal fine occorre identificare i codici OP delle istruzioni JMP, JSR e di Branch presenti nel file. Per queste istruzini di salto, il computer ricerca nel Symbol Stack i corrispondenti numeri di Label, che sono posti dopo il codice OP dell'istruzione in corso di assemblaggio momentaneo. In tal modo ne ricava l'indirizzo di Label. L'indirizzo di Label è eguale all'indirizzo assoluto di salto, che viene inserito nel file dopo il codice OP dell'istruzione in questione. Per le istruzioni di salto condizionato deve venire invece calcolato il valore dell'offset. Esso può essere calcolato dal computer in base all'indirizzo di partenza e l'indirizzo di arrivo. Calcolato l'offset, questo viene inserito nel file dopo il codice OP dell'istruzione.

La subroutine GETLBL

GETLBL costituisce il cuore dell'Assembler. In questa subroutine il computer ricava l'indirizzo di un Label dal Symbol Stack, quando deve assemblare una istruzione JMP, JSR o di Branch. I compiti svolti da questa subroutine si riassumono nei seguenti punti:

1. Cerca il numero di Label sul Symbol Stack.

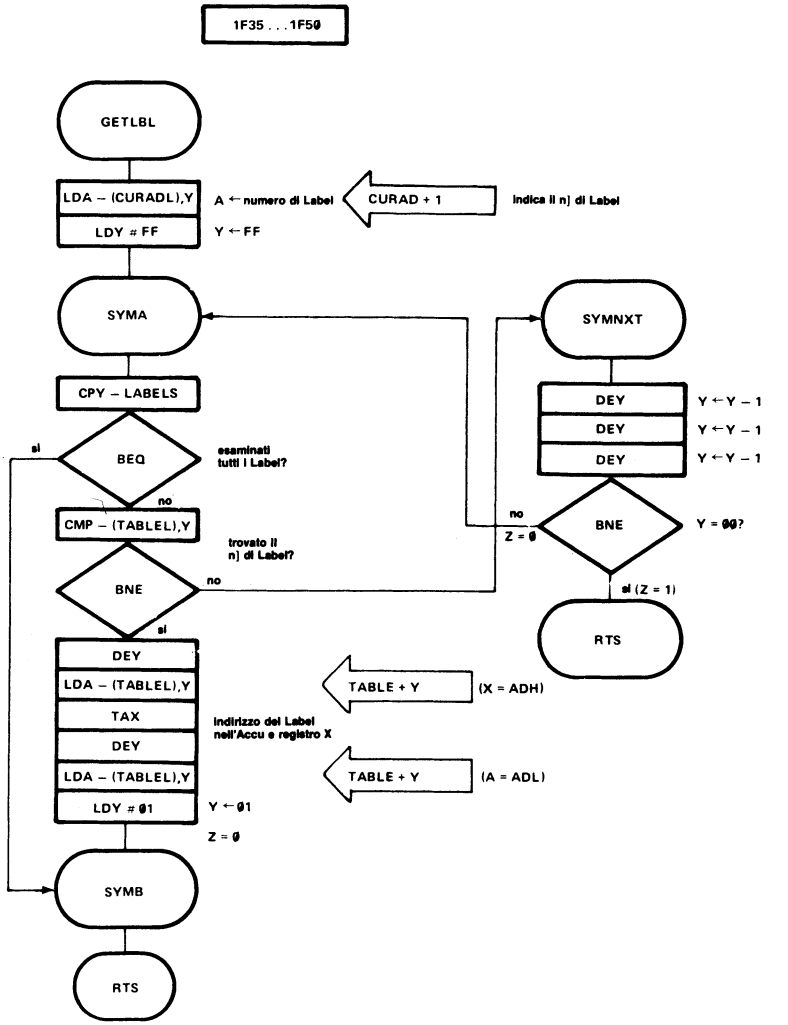


Figura 8. GETLBL è la subroutine focale dell'Assembler nel corso della seconda fase. In essa il computer accerta il numero di Label dell'istruzione da assemblare, e depone l'indirizzo relativo al n° di Label in questione in due registri interni della CPU: l'Accu contiene l'ADL e il registro X l'ADH.

2. Trovato il n° di Label, recupera l'indirizzo relativo dal Symbol Stack. Deponi il byte alto d'indirizzo del Label nel registro X della CPU, ed il byte basso nell'Accu.
3. Se non trovi il n° di Label cercato sul Symbol Stack, rientra al programma Assembler con il Flag Z posto ad 1.

La fig. 8 mostra come al principio venga caricato nell'Accu il n° di Label situato dopo il codice OP dell'istruzione in corso di assemblaggio:
LDA - (CURADL), Y con: Y = 01.
Successivamente si carica FF nel registro Y. Prima di proseguire la descrizione della subroutine GETLBL, sarebbe opportuno ricordare quanto segue:

* L'indirizzo indicato dal Pointer del Symbol Stack TABLE + LABELS è la cella di memoria libera superiore del Symbol Stack. Durante la seconda fase di assemblaggio il Symbol Stack è cresciuto dall'indirizzo TABLE + FF = ENDAD all'indirizzo TABLE + LABEL.
* TABLE + FF costituisce il "fondo" del Symbol Stack, così come TABLE + LABELS ne costituisce la "cima". Fra questi due indirizzi si trovano tutte le informazioni relative ai Label che sono state poste sul Symbol Stack durante PASSA.
* La ricerca di un numero di Label sul Symbol Stack principia dal "fondo" (indirizzo TABLE + FF) e finisce alla "cima" (indirizzo TABLE + LABELS).

Dopo il Label SYMA, il computer verifica se il Symbol Table Pointer indica che esso è già giunto alla "cima". Ciò avviene tramite l'istruzione CPY-LABELS. L'istruzione successiva, CMP-(TABLEL), Y, esegue il confronto fra un n° di Label sul Symbol Stack con uno del file. In che modo? Prima che venga eseguito questo confronto, nel computer si sono stabilite le seguenti condizioni:

1. Nel corso di PASSB il computer ha incontrato un'istruzione dopo il cui codice OP era un numero di Label. Tale istruzione deve essere quindi assemblata. Il relativo compito è svolto dalla subroutine GETLBL.
2. Per prima cosa, il n° di Label posto dopo il codice OP dell'istruzione da assemblare viene posto nell'Accu.
3. Il Symbol-Table Pointer indica ora l'indirizzo TABLE + FF. A questo indirizzo è stato deposto sul Symbol Stack il primo n° di Label. Il computer esegue il confronto fra il n° di Label nell'Accu ed il n° di Label sul Symbol Stack. La relativa istruzione è: CMP-(TABLEL), Y. Se i due valori sono diversi il programa salta al Label SYMNXT. Dopo questo Label il registro Y viene decrementato per tre volte.
4. Dato che dopo tale triplice decremento il registro Y differisce ancora da 0, il programma salta al Label SYMA. Qui si verifica nuovamente se si sono confrontati già tutti i numeri di Label

del Symbol Stack con il n° di Label presente nell'Accu. Poiché in precedenza il registro Y è stato decrementato per tre volte, il Symbol Table Pointer TABLE + Y indica un ulteriore n° di Label deposto sullo Stack. Anche questo viene confrontato con il n° di Label nell'Accu.

5. Se i due numeri di Label sono ancora diversi fra loro, il registro Y viene ulteriormente decrementato: il Symbol Stack Pointer indica un indirizzo dello Stack a cui è stato deposto un altro n° di Label. Se invece i due numeri di Label al confronto risultano eguali, il n° di Label ricercato è stato trovato.
6. Il n° di Label ricercato è dunque stato rintracciato. Il computer può ricavare dal Symbol Stack l'indirizzo corrispondente a tale n° di Label. Prima si recupera il byte alto d'indirizzo del Label dal Symbol Stack. Le istruzioni per ciò sono:
   DEY
   LDA-(TABLEL), Y
   TAX
   Il Symbol Table Pointer viene per primo spostato verso l'alto di una posizione ed indica così il byte alto indirizzo del Label. Questo byte viene trasferito nell'Accu e poi nel registro X della CPU. Un altro decremento di 1 del registro Y provoca un nuovo spostamento di 1 posizione verso l'alto del Symbol Stack Pointer. Esso indica ora la locazione in cui è posto il byte basso d'indirizzo del Label. Anche questo byte viene caricato nell'Accu. Le relative istruzioni sono:
   DEY
   LDA-(TABLEL), Y
   A questo punto abbiamo il byte alto d'indirizzo del Label nel registro X ed il corrispondente byte basso nell'Accu. Questa è una delle condizioni con cui il processore deve rientrare dalla subroutine GETLBL alla routine PASSB dell'Assembler.
7. Altra condizione che la subroutine GETLBL deve soddisfare dopo il rientro nel programma Assembler è la seguente:
   * Se il computer ha rintracciato sul Symbol Stack un n° di Label corrispondente al n° di Label dell'istruzione in corso di assemblaggio, il Flag Z deve essere rimesso a 0.
   * Se invece il n° di Label ricercato, dell'istruzione in corso di assemblaggio, non si trova sul Symbol Stack, il Flag Z deve essere posto ad 1.
     Il resettaggio a 0 del Flag Z è assicurato dall'istruzione LDY 01 precedente il Label SYMB. Questa istruzione viene eseguita solo se il computer ha trovato sul Symbol Stack il n° di Label cercato.

Se invece il computer ha controllato tutto il Symbol Stack senza ritrovare il n° di Label cercato, ossia quello dell'istruzione in corso, il Flag Z deve essere posto ad 1. Se il n° di Label non è presente sul Symbol Stack, la ricerca viene interrotta appena il Pointer del

Symbol Stack giunge alla "cima" dello Stack. Solo in questo caso si verifica che TABLE + Y e TABLE + LABELS indicano lo stesso indirizzo. Il confronto fra gli indici LABELS ed Y segnala quindi se il cercato n° di Label è presente o non sul Symbol Stack. Questo confronto viene effettuato, dopo il Label SYMA, con:
CPY-LABELS
Se il n° di Label non si trova sul Symbol Stack, il risultato di questo confronto è 0. Corrispondentemente il Flag Z viene, come necessario, posto ad 1, ed il computer rientra nel programma Assembler, immediatamente prima del Label SYMB.
Ed ora conosciamo a fondo anche l'Assembler. Editor ed Assembler sono fra loro strettamente collegati: entrambi i programmi infatti impiegano le stesse subroutine. Adesso che queste subroutine sono divenute chiare al programmatore, questi può comodamente inserirle in propri programmi. Al termine del libro sono elencati i Source Listing di tutti i programmi descritti. Da essi si possono ricavare molte informazioni utili quando si vogliono utilizzare separatamente le subroutine del Monitor, dell'Editor o dell'Assembler. Sono forniti pure i Source Listing dei programmi didattici sviluppati nei capitolo 5 e 6.

La routine BRANCH dell'EPROM

Nel 1° volume "Junior-Computer 1", in varie occasioni, si è accennato che nell'EPROM è situato un programma che calcola gli offset delle istruzioni di Branch. Questo programma si chiama BRANCH ed inizia all'indirizzo 1FD5. BRANCH (Fig. 9) è costituito da un loop di programma senza fine, da cui si può uscire solo mediante l'uso dei tasti RST, ST od un NMI od un IRQ generati esternamente.
Inizialmente, subito dopo partita la routine BRANCH, il display segna 00 00 00. L'utente introduce ora il byte basso dell'indirizzo a cui si trova il codice OP dell'istruzione di Branch.
Dopo l'introduzione, questo byte basso d'indirizzo compare visualizzato sui due display di sinistra a sette segmenti.
Segue l'introduzione del secondo dato: l'utente introduce il byte basso dell'indirizzo a cui deve portare l'istruzione di Branch. Esso compare, dopo impostato, visualizzato sui due display centrali. I due display di destra indicano invece l'offset calcolato. Impostando gli indirizzi è sufficiente introdurre i byte bassi dell'indirizzo di arrivo e di quello di partenza, in quanto la CPU 6502 può spaziare su un campo di +127 passi in avanti e -128 passi all'indietro.
Dopo il calcolo di un offset, premendo un qualsiasi tasto comando il display torna a segnare 00 00 00. Lo stesso vale anche durante l'introduzione dei due byte bassi indirizzo citati.
Trattando la routine BRANCH possiamo limitarci ad una breve descrizione, dato che per il calcolo dell'offset vale quanto era stato detto nel caso analogo per l'Assembler, capitolo 9, fig. 7.

Dapprima si carica 00 nei buffer di display INH, POINTL e POINTH, ed il computer viene commutato per calcolo in modo binario. Dopo il Label BR si richiama la subroutine GETBYT. Questa depone i valori di due tasti dati premuti nell'Accu, e rientra, in questo caso, nella routine BRANCH col Flag N = 1. Se però viene premuto un qualche tasto comandi, il Flag N viene posto = 0, ed il computer salta al Label BRANCH. Anche il display viene

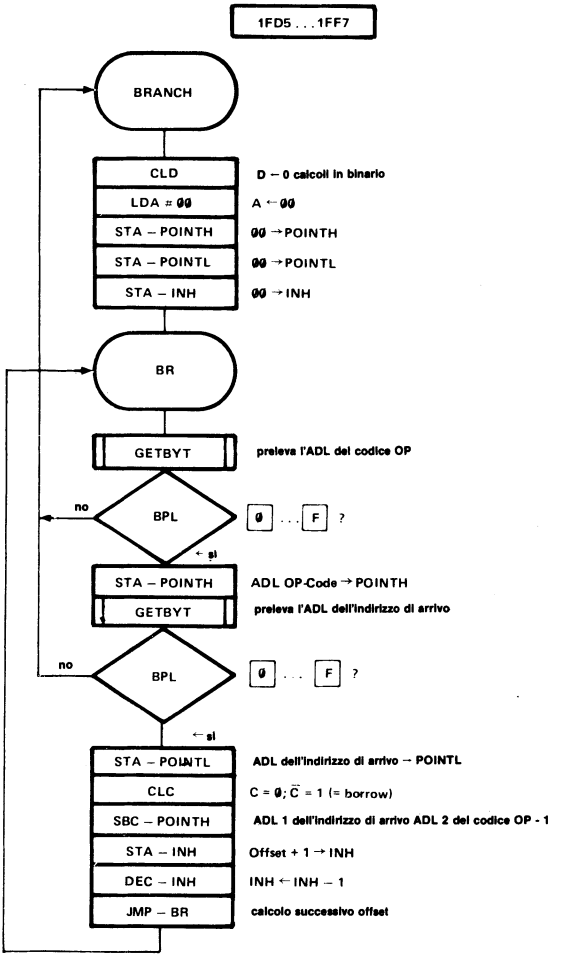


Figura 9. Nel 1° volume si è spiegato come viene calcolato l'offset. Il programma che viene utilizzato per questo scopo è la subroutine BRANCH, posta nell'EPROM dello Junior-Computer. Questa figura ne mostra il diagramma di flusso dettagliato.

rimesso a 0. Trasferiti i valori dei due tasti dati nell'Accu, essi passano poi nel buffer di display POINTH. POINTH contiene dunque il byte basso dell'indirizzo a cui si trova il codice OP dell'istruzione di Branch. Quando si richiama per la seconda volta la subroutine GETBYT, è il byte basso dell'indirizzo di arrivo del salto che viene analogamente letto e posto nell'Accu. Trasferito tale byte nel buffer di display POINTL, lo Junior Computer inizia il calcolo dell'offset.
Nell'accumulatore abbiamo dunque il byte basso dell'indirizzo di arrivo del salto condizionato. Da esso il computer sottrae il byte basso dell'indirizzo a cui è posto il codice OP dell'istruzione Branch. Dato che prima della sottrazione il Flag C è a 0 (il che non è ammesso!), il risultato è errato perché superiore di 1 al valore esatto. Questo offset scorretto è trasferito al buffer di display INH ed ivi decrementato: in tal modo ora nel buffer INH è presente il valore corretto dell'offset. Il programma salta indietro al Label BR e visualizza infine i byte bassi degli indirizzi di arrivo e di partenza nonché il relativo offset.

Per i lettori pignoli

Ecco come viene calcolato l'offset nella routine BRANCH:
Offset = Indirizzo di arrivo - indirizzo di partenza - 1 - 1.
Invece nella routine Assembler il calcolo avviene così:
Offset = indirizzo di arrivo - indirizzo di partenza - 2.
Così è finito pure il 2° volume.
I due volumi ci hanno mostrato come è possibile allestire con un piccolo computer in modo semplice programmi degni di questo nome. Abbiamo così concluso la descrizione dello Junior-Computer modello standard. Al termine del volume troverete un'Appendice, in cui sono riportati un sommario delle diverse subroutine ed i Source Listing commentati dei vari programmi. Questa Appendice dovrebbe costituire una vera "miniera d'oro" per chi ha l'hobby della programmazione (Software). Chi ci ha seguito sin qui attraverso i primi due volumi sullo Junior-Computer non avrà certo difficoltà a essere dei nostri anche per il 3° volume: in esso, grazie ad una non costosa espansione dell-'Hardware, il nostro Junior-Computer si svilupperà ad un vero e proprio Personal-Computer!

Appendici

1 Sommario delle Subroutine

ADCEND
Chiamata: JSR-ADCEND oppure JSR-1ED8 oppure 20 D8 1E.
La subroutine ADCEND sposta il Pointer CEND di uno, due o tre posizioni verso il basso. Il contenuto della cella di memoria BYTES 00F6 determina l'entità dello spostamento di CEND.

AK
Chiamata: JSR-AK oppure JSR-1DB1 oppure 20 B1 1D. La subroutine AK è usata come appendice delle subroutine SCAND e SCAN/SCANDS. Quando vengono chiamate le subroutine SCAN/SCANDS automaticamente si passa pure attraverso AK. La subroutine AK sonda la tastiera dello Junior-Computer. Se nella tastiera è premuto qualche tasto, il processore rientra da questa subroutine con l'Accu ≠ 00. Se invece non è stato premuto alcun tasto, il contenuto dell'Accu al rientro = 00

BEGIN
Chiamata: JSR-BEGIN oppure JSR-1ED3 oppure 20 D3 1E
La subroutine BEGIN rende il Pointer CURAD (pointer di display) eguale al Pointer BEGAD. La relativa scrittura simbolica è:
CURAD = BEGAD

CONVD
Chiamata: JSR-CONVD oppure JSR-1DDF oppure 20 DF 1D
La subroutine CONVD provvede al pilotaggio temporaneo del display. Il contenuto dei buffer di display viene convertito in codice a 7 segmenti con l'ausilio di una Lookup Table.

DOWN
Chiamata: JSR-DOWN oppure JSR-1EA6 oppure 20 A6 1E
La subroutine DOWN sposta un blocco di dati fra il Pointer CURAD e CEND di uno, due o tre posizioni verso il basso. Il contenuto della cella di memoria BYTES 00F6 determina di quante posizioni il blocco dati si debba spostare verso il basso.

FILLWS
Chiamata: JSR-FILLWS oppure JSR-1E47 oppure 20 47 1E
La subroutine FILLWS trasferisce un'istruzione dal buffer di di-

splay nella memoria operativa dello Junior-Computer. A tal fine viene reso libero nella memoria operativa uno spazio di uno, due o tre byte. Quando il processore rientra da questa subroutine, il Flag N è sempre posto a 1.

GETBYT
Chiamata: JSR-GETBYT oppure JSR-1D6F oppure 20 6F 1D
La subroutine GETBYT legge due tasti dati e pone il loro valore nell'Accu. Nel corso di questa subroutine il computer gestisce pure il display. Il tasto premuto per primo diviene il nibble alto nell'Accu. Altre caratteristiche di GETBYT:
- Posti i due valori dei tasti dati nell'Accu, il computer rientra da GETBYT col Flag N alto (N = 1).
- Se viene premuto un tasto comandi, il computer rientra da questa subroutine con il Flag N basso (N = 0).

GETKEY
Chiamata: JSR-GETKEY oppure JSR-1DF9 oppure 20 F9 1D
La subroutine GETKEY calcola il valore di tasto di un tasto premuto. Al rientro da questa subroutine il valore calcolato del tasto è nell'Accu.

LENACC
Chiamata: JSR-LENACC oppure JSR-1E 60 oppure 20 60 1E
La subroutine LENACC calcola la lunghezza di una qualsiasi istruzione della CPU 6502. Prima di chiamare questa routine, nell'Accu deve risultare presente il codice OP dell'istruzione. La lunghezza dell'istruzione viene posta nella cella di memoria BYTES 00F6.

NEXT
Chiamata: JSR-NEXT oppure JSR-1EF8 oppure 20 F8 1E
La subroutine NEXT sposta il Pointer di display CURAD di una, due o tre posizioni verso il basso. Il contenuto della cella di memoria BYTES 00F6 determina di quante posizioni si deve muovere CURAD.

OPLEN
Chiamata: JSR-OPLEN oppure JSR-1E5C oppure 20 5C 1E
La subroutine OPLEN calcola la lunghezza di una qualsiasi istruzione della CPU 6502, il cui codice OP sia indicato dal Pointer di display CURAD. La lunghezza d'istruzione così calcolata viene posta nella cella di memoria BYTES 00F6.

RDINST
Chiamata: JSR-RDINST oppure JSR-1E20 oppure 20 20 1E
La subroutine RDINST legge un'istruzione dalla tastiera e la trasferisce nei buffer di display:
- POINTH contiene il codice OP,
- POINTL contiene il primo operando (se presente),
- INH contiene il secondo operando (se presente).

Mentre l'utente imposta da tastiera un'istruzione, RDINST provvede anche a gestire, il display. Introdotta l'istruzione completa, il processore rientra da questa subroutine con il Flag N posto ad 1. Se tuttavia durante RDINST viene premuto un tasto comandi, il computer sospende la lettura dell'istruzione e rientra dalla subroutine con il Flag N basso (N = 0).

RECEND
Chiamata: JSR-RECEND oppure JSR-1EEA oppure 20 EA 1E
La subroutine RECEND sposta il Pointer CEND di una, due o tre posizioni verso l'alto. Il contenuto della cella di memoria BYTES determina di quante posizioni verso l'alto si debba muovere CEND.

SCAN
Chiamata: JSR-SCAN oppure JSR-1D4D oppure 20 4D 1D
La subroutine SCAN riempie i buffer di display POINTH, POINTL ed INH con una istruzione della CPU 6502 e li visualizza poi sul display. Provvede anche al sondaggio della tastiera. SCAN svolge i seguenti compiti:
- trasporta ai buffer di display l'istruzione il cui codice OP è indicato dal Pointer di display CURAD,
- calcola automaticamente la lunghezza dell'istruzione visualizzata sul display,
- opera con lunghezza del display variabile,
- se SCAN riconosce un tasto premuto nella tastiera, rientra con il valore del tasto posto nell'Accu.

SCAND
Chiamata: JSR-SCAND oppure JSR-1D88 oppure 20 88 1D
SCAND è una subroutine che trasferisce nel buffer di display INH i dati il cui indirizzo è indicato dal Pointer indirizzi POINT. Per il resto è identica a SCANDS.

SCANDS
Chiamata: JSR-SCANDS oppure JSR-1D8E oppure 20 8E 1D
La subroutine visualizza sul display il contenuto dei buffer di display POINTH, POINTL ed INH. La lunghezza del display può essere variabile. Caricando 01, 02 o 03 nella cella di memoria BYTES si possono verificare le situazioni seguenti:
- BYTES contiene 01: sul display viene visualizzato solo il contenuto di POINTH.
- BYTES contiene 02: sul display vengono visualizzati solo i contenuti di POINTH e POINTL.
- BYTES contiene 03: sul display vengono visualizzati POINTH, POINTL ed INH

SHOW
Chiamata: JSR-SHOW oppure JSR-1DCC oppure 20 CC 1D
La subroutine SHOW trasferisce il contenuto di un dato buffer di display (POINTH, POINTL od INH) al display. Il registro X viene utilizzato come contatore del display. Il contenuto del registro X determina cioé quale dei 6 display verrà pilotato. SHOW può rappresentare sul display solo numeri esadecimali.

UP
Chiamata: JSR-IP oppure JSR-1E83 oppure 20 83 1E
La subroutine UP sposta verso l'alto un blocco di dati fra i Pointer CURAD e CEND di una, due o tre posizioni. Il contenuto della cella di memoria BYTES 00F6 determina di quanti posti debba muoversi il blocco di dati.

2 Source-Listing

* **Editor**
* **Assembler**
* **Monitor**
* **Routine BRANCH**

```
0000: 1C00            LOYS   ORG   $1C00  VERSION D
0010:
0020:
0030:
0040:
0050:                 SOURCE LISTING OF ELEKTOR'S JUNIOR COMPUTER
0060:
0070:                 WRITTEN BY A. NACHTMANN
0080:
0090:                 DATE: 7 FEB. 1980
0100:
0110:                 THE FEATURES OF JUNIOR'S MONITOR ARE:
0120:
0130:                 HEX ADDRESS DATA DISPLAY (ENTRY VIA RST)
0140:                 HEX EDITOR (START ADDRESS $1CB5)
0150:                 HEX ASSEMBLER (START ADDRESS $1F51)
0160:
0170:                 EDITOR'S POINTERS AND TEMPS IN PAGE ZERO
0180:
0190: 1C00            KEY    *      $00E1
0200: 1C00            BEGADL *      $00E2  BEGIN ADDRESS POINTER
0210: 1C00            BEGADH *      $00E3
0220: 1C00            ENDADL *      $00E4  END ADDRESS POINTER
0230: 1C00            ENDADH *      $00E5
0240: 1C00            CURADL *      $00E6  CURRENT ADDRESS POINTER
0250: 1C00            CURADH *      $00E7
0260: 1C00            CENDL  *      $00E8  CURRENT ADDRESS POINTER
0270: 1C00            CENDH  *      $00E9
0280: 1C00            MOVADL *      $00EA
0290: 1C00            MOVADH *      $00EB
0300: 1C00            TABLEL *      $00EC
0310: 1C00            TABLEH *      $00ED
0320: 1C00            LABELS *      $00EE
0330: 1C00            BYTES  *      $00F6  NUMBER OF BYTES TO BE DISPLAYED
0340: 1C00            COUNT  *      $00F7
0350:
0360:                 MPU REGISTERS IN PAGE ZERO
0370:
0380: 1C00            PCL    *      $00EF
0390: 1C00            PCH    *      $00F0
0400: 1C00            PREG   *      $00F1
0410: 1C00            SPUSER *      $00F2
0420: 1C00            ACC    *      $00F3
0430: 1C00            YREG   *      $00F4
0440: 1C00            XREG   *      $00F5
0450
0460:                 HEX DISPLAY BUFFERS IN PAGE ZERO
0470:
0480: 1C00            INL    *      $00F8
0490: 1C00            INH    *      $00F9
0500: 1C00            POINTL *      $00FA
0510: 1C00            POINTH *      $00FB
0520:
0530:                 TEMPORARY DATA BUFFERS IN PAGE ZERO
0540:
0550: 1C00            TEMP   *      $00FC
0560: 1C00            TEMPX  *      $00FD
0570: 1C00            NIBBLE *      $00FE
0580: 1C00            MODE   *      $00FF  (0 = DA MODE, ≠0 = AD MODE)
0590:
0600:                 MEMORY LOCATIONS IN THE 6532-IC
0610:
0620: 1C00            PAD    *      $1A80  DATA REGISTER OF PORT A
0630: 1C00            PADD   *      $1A81  DATA DIRECTION REGISTER OF PORT A
0640: 1C00            PBD    *      $1A82  DATA REGISTER OF PORT B
0650: 1C00            PBDD   *      $1A83  DATA DIRECTION REGISTER OF PORT B
0660:
0670:                 WRITE EDGE DETECT CONTROL
0680:
0690: 1C00            EDETA  *      $1AE4  NEG EDET DISABLE PA7-IRQ
0700: 1C00            EDETB  *      $1AE5  POS EDET DISABLE PA7-IRQ
0710: 1C00            WDETC  *      $1AE6  NEG EDET ENABLE PA7-IRQ
0720: 1C00            EDETD  *      $1AE7  POS EDET ENABLE PA7-IRQ
0730:
0740:                 READ FLAG REGISTER AND CLEAR TIMER & IRQ FLAG
0750:
0760: 1C00            RDFLAG *      $1AD5  BIT6=PA7-FLAG; BIT7=TIMER-FLAG
0770:
0780:                 WRITE COUNT INTO TIMER, DISABLE TIMER-IRQ
0790:
0800: 1C00            CNTA   *      $1AF4  CLK1T
0810: 1C00            CNTB   *      $1AF5  CLK8T
0820: 1C00            CNTC   *      $1AF6  CLK64T
0830: 1C00            CNTD   *      $1AF7  CLK1KT
0840:
0850:                 WRITE COUNT INTO TIMER, ENABLE TIMER-IRQ
0860:
0870: 1C00            CNTE   *      $1AFC  CLK1T
0880: 1C00            CNTF   *      $1AFD  CLK8T
0890: 1C00            CNTG   *      $1AFE  CLK64T
0900: 1C00            CNTH   *      $1AFF  CLK1KT
0910:
0920:                 INTERRUPT VECTORS: IRQ & NMI VECTORS SHOULD BE
0930:                 LOADED IN THE FOLLOWING MEMORY LOCATIONS FOR
0940:                 PROPER SYSTEM OPERATION.
```

```
0950:
0960: 1C00           NMIL   *      $1A7A  NMI LOWER BYTE
0970: 1C00           NMIH   *      $1A7B  NMI HIGHER BYTE
0980: 1C00           IRQL   *      $1A7E  IRQ LOWER BYTE
0990: 1C00           IRQH   *      $1A7F  IRQ HIGHER BYTE
1000:
1010:                BEGINNERS MAY LOAD INTO THESE LOCATIONS
1020:                $1C00 FOR STEP BY STEP MODUS AND BRK COMMAND
1030:
1040:
1050:
1060:                JUNIOR'S MAINROUTINES
1070:
1080: 1C00 85 F3     SAVE   STAZ  ACC    SAVE ACCU
1090: 1C02 68               PLA           GET CURRENT P-REGISTER
1100: 1C03 85 F1            STAZ  PREG    SAVE P-REGISTER
1110: 1C05 68        SAVEA  PLA           GET CURRENT PCL
1120: 1C06 85 EF            STAZ  PCL     SAVE CURRENT PCL
1130: 1C08 85 FA            STAZ  POINTL PCL TO DISPLAY BUFFER
1140: 1C0A 68               PLA           GET CURRENT PCH
1150: 1C0B 85 F0            STAZ  PCH     SAVE CURRENT PCH
1160: 1C0D 85 FB            STAZ  POINTH PCH TO DISPLAY BUFFER
1170: 1C0F 84 F4     SAVEB  STYZ  YREG    SAVE CURRENT Y-REGISTER
1180: 1C11 86 F5            STXZ  XREG    SAVE CURRENT X-REGISTER
1190: 1C13 BA               TSX           GET CURRENT SP
1200: 1C14 86 F2            STX   SPUSER SAVE CURRENT SP
1210: 1C16 A2 01            LDXIM $01     SET AD-MODE
1220: 1C18 86 FF            STXZ  MODE
1230: 1C1A 4C 33 1C         JMP   START
1240:
1250: 1C1D A9 1E     RESET  LDAIM $1E     PB1---PB4
1260: 1C1F 8D 83 1A         STA   PBDD     IS OUTPUT
1270: 1C22 A9 04            LDAIM $04     RESET P-REGISTER
1280: 1C24 85 F1            STAZ  PREG
1290: 1C26 A9 03            LDAIM $03
1300: 1C28 85 FF            STAZ  MODE     SET AD-MODE
1310: 1C2A 85 F6            STAZ  BYTES   DISPLAY POINTH,POINTL,INH
1320: 1C2C A2 FF            LDXIM $FF     ADJUST THE STACKPOINTER
1330: 1C2E 9A               TXS
1340: 1C2F 86 F2            STXZ  SPUSER
1350: 1C31 D8               CLD
1360: 1C32 78               SEI
1370:
1380: 1C33 20 88 1D  START  JSR   SCAND   DISPLAY DATA SPECIFIED BY POINTH,POINTL
1390: 1C36 D0 FB            BNE   START   WAIT UNTIL KEY IS RELEASED
1400: 1C38 20 88 1D  STARA  JSR   SCAND   DISPLAY DATA SPECIFIED BY POINT
1410: 1C3B F0 FB            BEQ   STARA    ANY KEY DEPRESSED
1420: 1C3D 20 88 1D         JSR   SCAND    DEBOUNCE KEY
1430: 1C40 F0 F6            BEQ   STARA   ANY KEY STILL DEPRESSED
1440: 1C42 20 F9 1D         JSR   GETKEY IF YES   DECODE KEY,RETURN WITH KEY IN ACCU
1450:
1460: 1C45 C9 13     GOEXEC CMPIM $13     GO-KEY?
1470: 1C47 D0 13            BNE   ADMODE
1480: 1C49 A6 F2            LDXZ  SPUSER  GET CURRENT SP
1490: 1C4B 9A               TXS
1500: 1C4C A5 FB            LDAZ  POINTH   START EXECUTION AT POINTH,POINTL
1510: 1C4E 48               PHA
1520: 1C4F A5 FA            LDAZ  POINTL
1530: 1C51 48               PHA
1540: 1C52 A5 F1            LDAZ  PREG    RESTORE CURRENT P REGISTER
1550: 1C54 48               PHA
1560: 1C55 A6 F5            LDXZ  XREG
1570: 1C57 A4 F4            LDYZ  YREG
1580: 1C59 A5 F3            LDAZ  ACC
1590: 1C5B 40               RTI           EXECUTE PROGRAM
1600: 1C5C C9 10     ADMODE CMPIM $10     AD-KEY?
1610: 1C5E D0 06            BNE   DAMODE
1620: 1C60 A9 03            LDAIM $03     SET AD-MODE
1630: 1C62 85 FF            STAZ  MODE
1640: 1C64 D0 14            BNE   STEPA
1650:
1660: 1C66 C9 11     DAMODE CMPIM $11     DA-KEY?
1670: 1C68 D0 06            BNE   STEP
1680: 1C6A A9 00            LDAIM $00     SET DA-MODE
1690: 1C6C 85 FF            STAZ  MODE
1700: 1C6E F0 0A            BEQ   STEPA
1710:
1720: 1C70 C9 12     STEP   CMPIM $12     PLUS-KEY?
1730: 1C72 D0 09            BNE   PCKEY
1740: 1C74 E6 FA            INCZ  POINTL
1750: 1C76 D0 02            BNE   STEPA
1760: 1C78 E6 FB            INCZ  POINTH
1770: 1C7A 4C 33 1C  STEPA  JMP   START
1780:
1790: 1C7D C9 14     PCKEY  CMPIM $14     PC-KEY?
1800: 1C7F D0 0B            BNE   ILLKEY
1810: 1C81 A5 EF            LDAZ  PCL
1820: 1C83 85 FA            STAZ  POINTL  LAST PC TO DISPLAY BUFFER
1830: 1C85 A5 F0            LDAZ  PCH
1840: 1C87 85 FB            STAZ  POINTH
1850: 1C89 4C 7A 1C         JMP   STEPA
1860:
1870: 1C8C C9 15     ILLKEY CMPIM $15     ILLEGAL KEY?
1880: 1C8E 10 EA            BPL   STEPA   IF YES,IGNORE IT
1890:
```

```
1900: 1C90 85 E1     DATA   STAZ  KEY     SAVE KEY
1910: 1C92 A4 FF            LDYZ  MODE    Y=0 IS DATA MODE,ELSE ADDRESS MODE
1920: 1C94 D0 0D            BNE   ADDRES
1930: 1C96 B1 FA            LDAIY POINTL GET DATA SPECIFIED
1940: 1C98 0A               ASLA          BY POINT
1950: 1C99 0A               ASLA          SHIFT LOW ORDER
1960: 1C9A 0A               ASLA          NIBBLE INTO HIGH ORDER NIBBLE
1970: 1C9B 0A               ASLA
1980: 1C9C 05 E1            ORAZ  KEY     DATA WITH KEY
1990: 1C9E 91 FA            STAIY POINTL RESTORE DATA
2000: 1CA0 4C 7A 1C         JMP   STEPA
2010:
2020: 1CA3 A2 04     ADDRES LDXIM $04     4 SHIFTS
2030: 1CA5 06 FA     ADLOOP ASLZ  POINTL  POINTH,POINTL 4 POSITIONS TO LEFT
2040: 1CA7 26 FB            ROLZ  POINTH
2050: 1CA9 CA               DEX
2060: 1CAA D0 F9            BNE   ADLOOP
2070: 1CAC A5 FA            LDAZ  POINTL
2080: 1CAE 05 E1            ORAZ  KEY     RESTORE ADDRESS
2090: 1CB0 85 FA            STAZ  POINTL
2100: 1CB2 4C 7A 1C         JMP   STEPA
2110:
2120:
2130:
2140:
2150:
2160:                JUNIOR'S HEX EDITOR
2170:
2180:                FOLLOWING COMMANDS ARE VALID:
2190:
2200:                "INSERT": INSERT A NEW LINE JUST BEFORE DISPLAYED LINE
2210:
2220:                "INPUT": INSERT A NEW LINE JUST BEHIND THE
2230:                         DISPLAYED LINE
2240:
2250:                "SEARCH": SEARCH IN WORKSPACE FOR A GIVEN 2BYTE PATTERN
2260:
2270:                "SKIP": SKIP TO NEXT INSTRUCTION
2280:
2290:                "DELETE": DELETE CURRENT DISPLAYED INSTRUCTION
2300:
2310:                AN ERROR IS INDICATED, IF THE INSTRUCTION POINTER
2320:                CURAD IS OUT OF RANGE
2330:
2340: 1CB5 20 D3 1E  EDITOR JSR   BEGIN  CURAD:=BEGAD
2350: 1CB8 A4 E3            LDYZ  BEGADH
2360: 1CBA A6 E2            LDXZ  BEGADL
2370: 1CBC E8               INX
2380: 1CBD D0 01            BNE   EDIT
2390: 1CBF C8               INY
2400: 1CC0 86 E8     EDIT   STXZ  CENDL  CEND:=BEGAD+1
2410: 1CC2 84 E9            STYZ  CENDH
2420: 1CC4 A9 77            LDAIM $77    DISPLAY "77"
2430: 1CC6 A0 00            LDYIM $00
2440: 1CC8 91 E6            STAIY CURADL
2450:
2460: 1CCA 20 4D 1D  CMND   JSR   SCAN   DISPLAY CURRENT INSTRUCTION,WAIT FOR A KEY
2470:
2480: 1CCD C9 14     SEARCH CMPIM $14    SEARCH COMMAND?
2490: 1CCF D0 2A            BNE   INSERT
2500: 1CD1 20 6F 1D         JSR   GETBYT READ 1ST BYTE
2510: 1CD4 10 F7            BPL   SEARCH COM. KEY?
2520: 1CD6 85 FB            STAZ  POINTH DISCARD DATA
2530: 1CD8 20 6F 1D         JSR   GETBYT READ 2ND BYTE
2540: 1CDB 10 F0            BPL   SEARCH COM. KEY?
2550: 1CDD 85 FA            STAZ  POINTL DISCARD DATA
2560: 1CDF 20 D3 1E         JSR   BEGIN  CURAD:=BEGAD
2570: 1CE2 A0 00     SELOOP LDYIM $00
2580: 1CE4 B1 E6            LDAIY CURADL COMPARE INSTRUCTION
2590: 1CE6 C5 FB            CMPZ  POINTH AGAINST DATA TO BE SEARCHED
2600: 1CE8 D0 07            BNE   SEARA  SKIP TO NEXT INSTRUCTION, IF NOT EQUAL
2610: 1CEA C8               INY
2620: 1CEB B1 E6            LDAIY CURADL
2630: 1CED C5 FA            CMPZ  POINTL
2640: 1CEF F0 D9            BEQ   CMND   RETURN, IF 2BYTE PATTERN IS FOUND
2650: 1CF1 20 5C 1E  SEARA  JSR   OPLEN  GET LENGTH OF THE CURRENT INSTRUCTION
2660: 1CF4 20 F8 1E         JSR   NEXT   SKIP TO NEXT INSTRUCTION
2670: 1CF7 30 E9            BMI   SELOOP SEARCH AGAIN, IF CURAD IS LESS THAN CEND
2680: 1CF9 10 3E            BPL   ERRA
2690:
2700: 1CFB C9 10     INSERT CMPIM $10    INSERT COMMAND?
2710: 1CFD D0 0A            BNE   INPUT
2720: 1CFF 20 20 1E         JSR   RDINST READ INSTRUCTION AND COMPUTE LENGTH
2730: 1D02 10 C9            BPL   SEARCH COM. KEY?
2740: 1D04 20 47 1E         JSR   FILLWS MOVE DATA IN WS DOWNWARD BY THE AM. IN BYTES
2750: 1D07 F0 C1            BEQ   CMND   RETURN TO DISPLAY THE INSERTED INSTR.
2760:
2770: 1D09 C9 13     INPUT  CMPIM $13    INPUT COMMAND?
2780: 1D0B D0 14            BNE   SKIP
2790: 1D0D 20 20 1E         JSR   RDINST READ INSTRUCTION AND COMPUTE LENGTH
2800: 1D10 10 BB            BPL   SEARCH COM. KEY?
2810: 1D12 20 5C 1E         JSR   OPLEN  LENGTH OF THE CURRENT INSTR.
2820: 1D15 20 F8 1E         JSR   NEXT   RETURN WITH N=1, IF CURAD IS LESS THAN CEND
2830: 1D18 A5 FD            LDAZ  TEMPX  LENGTH OF INSTR. TO BE INSERTED
2840: 1D1A 85 F6            STAZ  BYTES
```

```
2850: 1D1C 20 47 1E         JSR   FILLWS MOVE DATA IN WS DOWNWARD BY THE AM. IN BYTI
2860: 1D1F F0 A9            BEQ   CMND   RETURN TO DISPLAY THE INSERTED DATA
2870:
2880: 1D21 C9 12      SKIP  CMPIM $12    SKIP COMMAND?
2890: 1D23 D0 07            BNE   DELETE
2900: 1D25 20 F8 1E         JSR   NEXT   SKIP TO NEXT INSTRUCTION. CURAD LESS THAN '
2910: 1D28 30 A0            BMI   CMND
2920: 1D2A 10 0D            BPL   ERRA
2930:
2940: 1D2C C9 11      DELETE CMPIM $11    DELETE COMMAND?
2950: 1D2E D0 09            BNE   ERRA
2960: 1D30 20 83 1E         JSR   UP     DELETE CURRENT INSTR. BY MOVING UP THE WS
2970: 1D33 20 EA 1E         JSR   RECEND ADJUST CURRENT END ADDRESS
2980: 1D36 4C CA 1C         JMP   CMND
2990:
3000: 1D39 A9 EE      ERRA  LDAIM $EE
3010: 1D3B 85 FB            STAZ  POINTH
3020: 1D3D 85 FA            STAZ  POINTL
3030: 1D3F 85 F9            STAZ  INH
3040: 1D41 A9 03            LDAIM $03
3050: 1D43 85 F6            STAZ  BYTES
3060: 1D45 20 8E 1D   ERRB  JSR   SCANDS DISPLAY EEEEEE UNTIL KEY IS RELEASED
3070: 1D48 D0 FB            BNE   ERRB
3080: 1D4A 4C CA 1C         JMP   CMND
3090:
3100:
3110:
3120:
3130:                 EDITOR'S SUBROUTINES
3140:
3150:                 SCAN IS A SUBROUTINE, FILLING UP
3160:                 THE DISPLAY BUFFER DETERMINED BY
3170:                 CURAD. THEN THE DISPLAY IS SCANNED
3180:                 DEPENDING OF THE LENGTH OF THE INSTRUCTION
3190:                 POINTED BY CURAD
3200:                 IF A DEPRESSED KEY IS DETECTED
3210:
3220:
3230:
3240:
3250:                 SCAN RETURNS WITH VALUE IN ACCU
3260:
3270: 1D4D A2 02      SCAN  LDXIM $02    FILL UP THE DISPLAY BUFFER
3280: 1D4F A0 00            LDYIM $00
3290: 1D51 B1 E6      FILBUF LDAIY CURADL START FILLING AT OP CODE
3300: 1D53 95 F9            STAX  INH
3310: 1D55 C8               INY
3320: 1D56 CA               DEX
3330: 1D57 10 F8            BPL   FILBUF
3340: 1D59 20 5C 1E         JSR   OPLEN  STORE INSTRUCTION LENGTH IN BYTES
3350: 1D5C 20 8E 1D   SCANA JSR   SCANDS DISPLAY CURRENT INSTRUCTION
3360: 1D5F D0 FB            BNE   SCANA  KEY RELEASED?
3370: 1D61 20 8E 1D   SCANB JSR   SCANDS DISPLAY CURRENT INSTRUCTION
3380: 1D64 F0 FB            BEQ   SCANB  ANY KEY DEPRESSED?
3390: 1D66 20 8E 1D         JSR   SCANDS DISPLAY CURRENT INSTRUCTION
3400: 1D69 F0 F6            BEQ   SCANB  ANY KEY STILL DEPRESSED?
3410: 1D6B 20 F9 1D         JSR   GETKEY IF YES, RETURN WITH KEY IN ACCU
3420: 1D6E 60               RTS
3430:
3440:                 GETBYT READS 2 HEXKEYS AND COMPOSES
3450:                 THEIR VALUES IN THE A REGISTER. IF ONLY
3460:                 HEXKEYS WERE DEPRESSED, IT RETURNS WITH
3470:                 N=1. IF A COMMAND KEY WAS DEPRESSED, IT
3480:                 RETURNES WITH N=0.
3490:
3500: 1D6F 20 5C 1D   GETBYT JSR   SCANA  READ HIGH ORDER NIBBLE
3510: 1D72 C9 10            CMPIM $10
3520: 1D74 10 11            BPL   BYTEND COMMAND KEY?
3530: 1D76 0A               ASLA
3540: 1D77 0A               ASLA         IF NOT, SAVE HIGH ORDER NIBBLE
3550: 1D78 0A               ASLA
3560: 1D79 0A               ASLA
3570: 1D7A 85 FE            STA   NIBBLE
3580: 1D7C 20 5C 1D         JSR   SCANA  READ LOW ORDER NIBBLE
3590: 1D7F C9 10            CMPIM $10
3600: 1D81 10 04            BPL   BYTEND COMMAND KEY?
3610: 1D83 05 FE            ORA   NIBBLE IF  NOT, COMPOSE BYTE
3620: 1D85 A2 FF            LDXIM $FF    SET N=1
3630: 1D87 60         BYTEND RTS
3640:
3650:                 SCAND IS A SUBROUTINE SHOWING DATA SPECIFIED BY
3660:                 POINT.
3670:                 SCANDS IS A SUBROUTINE SHOWING THE CONTENTS OF
3680:                 DISPLAY BUFFER AS A FUNCTION OF BYTES.
3690:                 THE FOLLOWING SUBROUTINE AK SCANS THE KEYBOARD.
3700:                 IT RETURNS WITH A=0, IF NO KEY IS DEPRESSED AND
3710:                 WITH A≠0 IF A KEY IS DEPRESSED.
3720:                 WHEN SCAND OR SCANDS ARE LEFT, PA0...PA7 IS INPUT.
3730:
3740:
3750:
3760:
3770: 1D88 A0 00      SCAND LDYIM $00
3780: 1D8A B1 FA            LDAIY POINTL GET DATA SPECIFIED BY POINT
3790: 1D8C 85 F9            STAZ  INH
```

```
3800: 1D8E A9 7F     SCANDS LDAIM $7F
3810: 1D90 8D 81 1A         STA   PADD    PA0...PA6 IS OUTPUT
3820: 1D93 A2 08            LDXIM $08     ENABLE DISPLAY
3830: 1D95 A4 F6            LDYZ  BYTES   FETCH LENGTH FROM BYTES
3840: 1D97 A5 FB     SCDSA  LDAZ  POINTH  OUTPUT 1ST BYTE
3850: 1D99 20 CC 1D         JSR   SHOW
3860: 1D9C 88               DEY
3870: 1D9D F0 0D            BEQ   SCDSB   MORE BYTES?
3880: 1D9F A5 FA            LDAZ  POINTL
3890: 1DA1 20 CC 1D         JSR   SHOW    IF YES, OUTPUT 2ND BYTE
3900: 1DA4 88               DEY
3910: 1DA5 F0 05            BEQ   SCDSB   MORE BYTES?
3920: 1DA7 A5 F9            LDAZ  INH
3930: 1DA9 20 CC 1D         JSR   SHOW    IF YES, OUTPUT 3RD BYTE
3940: 1DAC A9 00     SCDSB  LDAIM $00
3950: 1DAE 8D 81 1A         STA   PADD    PA0...PA7 IS INPUT
3960:
3970: 1DB1 A0 03     AK     LDYIM $03     SCAN 3 ROWS
3980: 1DB3 A2 00            LDXIM $00     RESET ROW COUNTER
3990:
4000: 1DB5 A9 FF     ONEKEY LDAIM $FF
4010: 1DB7 8E 82 1A  AKA    STX   PBD     OUTPUT ROW NUMBER
4020: 1DBA E8               INX           ENABLE FOLLOWING ROW
4030: 1DBB E8               INX
4040: 1DBC 2D 80 1A         AND   PAD     INPUT ROW PATTERN
4050: 1DBF 88               DEY           ALL ROWS SCANNED?
4060: 1DC0 D0 F5            BNE   AKA
4070: 1DC2 A0 06            LDYIM $06     TURN DISPLAY OFF
4080: 1DC4 8C 82 1A         STY   PBD
4090: 1DC7 09 80            ORAIM $80     SET BIT7=1
4100: 1DC9 49 FF            EORIM $FF     INVERT KEY PATTERN
4110: 1DCB 60               RTS
4120:
4130:                       THE SUBROUTINE SHOW TRANSPORTS THE
4140:                       CONTENTS OF ANY DISPLAY BUFFER TO THE
4150:                       DISPLAY. THE X REGISTER IS USED AS A
4160:                       SCAN COUNTER. IT DETERMINES, IF POINTH,
4170:                       POINTL OR INH IS TRANSPORTED TO THE
4180:                       DISPLAY.
4190: 1DCC 48        SHOW   PHA           SAVE DISPLAY
4200: 1DCD 84 FC            STYZ  TEMP    SAVE Y REGISTER
4210: 1DCF 4A               LSRA
4220: 1DD0 4A               LSRA          GET HIGH ORDER NIBBLE
4230: 1DD1 4A               LSRA
4240: 1DD2 4A               LSRA
4250: 1DD3 20 DF 1D         JSR   CONVD   OUTPUT HIGH ORDER NIBBLE
4260: 1DD6 68               PLA           GET DISPLAY AGAIN
4270: 1DD7 29 0F            ANDIM $0F     MASK OFF HIGH ORDER NIBBLE
4280: 1DD9 20 DF 1D         JSR   CONVD   OUTPUT LOW ORDER NIBBLE
4290: 1DDC A4 FC            LDYZ  TEMP    RESTORE Y REGISTER
4300: 1DDE 60               RTS
4310:
4320:
4330:
4340:
4350:                       THE SUBROUTINE CONVD CONTROLS THE DISPLAY SCAN.
4360:                       IT CONVERTS THE CONTENTS OF THE DISPLAY BUFFER
4370:                       TO BE DISPLAYED INTO A SEGMENT PATTERN.
4380:
4390: 1DDF A8        CONVD  TAY           USE NIBBLE AS INDEX
4400: 1DE0 B9 0F 1F         LDAY  LOOK    FETCH SEGMENT PATTERN
4410: 1DE3 8D 80 1A         STA   PAD     OUTPUT SEGMENT PATTERN
4420: 1DE6 8E 82 1A         STX   PBD     OUTPUT DIGIT ENABLE
4430: 1DE9 A0 7F            LDYIM $7F
4440: 1DEB 88        DELAY  DEY           DELAY 500 US APPROX.
4450: 1DEC 10 FD            BPL   DELAY
4460: 1DEE 8C 80 1A         STY   PAD     TURN SEGMENTS OFF
4470: 1DF1 A0 06            LDYIM $06
4480: 1DF3 8C 82 1A         STY   PBD     TURN DISPLAY OFF
4490: 1DF6 E8               INX           ENABLE NEXT DIGIT
4500: 1DF7 E8               INX
4510: 1DF8 60               RTS
4520:
4530:                       GETKEY CONVERTS A DEPRESSED KEY INTO A
4540:                       HEX NUMBER. IT RETURNS WITH THE KEY VALUE
4550:                       IN ACCU.
4560:
4570: 1DF9 A2 21     GETKEY LDXIM $21     START AT ROW 0
4580: 1DFB A0 01     GETKEA LDYIM $01     GET ONE ROW
4590: 1DFD 20 B5 1D         JSR   ONEKEY A=0, NO KEY DEPRESSED
4600: 1E00 D0 07            BNE   KEYIN
4610: 1E02 E0 27            CPXIM $27
4620: 1E04 D0 F5            BNE   GETKEA EACH ROW SCANNED?
4630: 1E06 A9 15            LDAIM $15     RETURN IF INVALID KEY
4640: 1E08 60               RTS
4650: 1E09 A0 FF     KEYIN  LDYIM $FF
4660: 1E0B 0A        KEYINA ASLA          SHIFT LEFT UNTIL Y=KEY NUMBER
4670: 1E0C B0 03            BCS   KEYINB
4680: 1E0E C8               INY
4690: 1E0F 10 FA            BPL   KEYINA
4700: 1E11 8A        KEYINB TXA
4710: 1E12 29 0F            ANDIM $0F     MASK MSD
4720: 1E14 4A               LSRA          DEVIDE BY 2
4730: 1E15 AA               TAX
4740: 1E16 98               TYA
```

```
4750: 1E17 10 03           BPL   KEYIND
4760: 1E19 18        KEYINC CLC
4770: 1E1A 69 07           ADCIM $07    ADD ROW OFFSET
4780: 1E1C CA        KEYIND DEX
4790: 1E1D D0 FA           BNE   KEYINC
4800: 1E1F 60              RTS
4810:
4820:                      RDINST TRANSFERS AN INSTRUCTION FROM KEYBOARD
4830:                      TO THE DISPLAY BUFFER. IT RETURNS WITH N=0 IF
4840:                      A COMMAND KEY WAS DEPRESSED. ONCE THE ENTIRE
4850:                      INSTRUCTION  IS READ, RDINST RETURNS WITH N=1.
4860:
4870:
4880: 1E20 20 6F 1D  RDINST JSR   GETBYT READ OP CODE
4890: 1E23 10 21           BPL   RDB    RETURN, IF COMMAND KEY
4900: 1E25 85 FB           STAZ  POINTH STORE OP CODE IN DISPLAY BUFFER
4910: 1E27 20 60 1E        JSR   LENACC COMPUTE INSTRUCTION LENGTH
4920: 1E2A 84 F7           STYZ  COUNT
4930: 1E2C 84 FD           STYZ  TEMPX
4940: 1E2E C6 F7           DECZ  COUNT
4950: 1E30 F0 12           BEQ   RDA    1 BYTE INSTRUCTION?
4960: 1E32 20 6F 1D        JSR   GETBYT IF NOT, READ FIRST OPERAND
4970: 1E35 10 0F           BPL   RDB    RETURN, IF COMMAND KEY
4980: 1E37 85 FA           STAZ  POINTL STORE 1ST OPERAND IN DISPLAY BUFFER
4990: 1E39 C6 F7           DECZ  COUNT
5000: 1E3B F0 07           BEQ   RDA    2 BYTE INSTRUCTION?
5010: 1E3D 20 6F 1D        JSR   GETBYT IF NOT, READ 2ND OPERAND
5020: 1E40 10 04           BPL   RDB    RETURN IF COMMAND KEY
5030: 1E42 85 F9           STAZ  INH    STORE 2ND OPERAND IN DISPLAY BUFFER
5040: 1E44 A2 FF     RDA   LDXIM $FF    N=1
5050: 1E46 60        RDB   RTS
5060:
5070:                      FILLWS TRANSFERS THE DATA FROM DISPLAY TO
5080:                      WORKSPACE. IT'S ALWAYS LEFT WITH Z=1
5090:
5100: 1E47 20 A6 1E  FILLWS JSR   DOWN   MOVE DATA DOWN BY THE AMOUNT IN BYTES
5110: 1E4A 20 DC 1E        JSR   ADCEND ADJUST CURRENT END ADDRESS
5120: 1E4D A2 02           LDXIM $02
5130: 1E4F A0 00           LDYIM $00
5140: 1E51 B5 F9     WS    LDAZX INH    FETCH DATA FROM DISPLAY BUFFER
5150: 1E53 91 E6           STAIY CURADL INSERT DATA INTO DATA FIELD
5160: 1E55 CA              DEX
5170: 1E56 C8              INY
5180: 1E57 C4 F6           CPYZ  BYTES  ALL INSERTED?
5190: 1E59 D0 F6           BNE   WS     IF NOT, CONTINUE
5200: 1E5B 60              RTS
5210:
5220:                      OPLEN COMPUTES THE LENGTH OF ANY 6502 INSTR.
5230:                      THE INSTR. LENGTH IS SAVED IN BYTES.
5240:
5250: 1E5C A0 00     OPLEN LDYIM $00
5260: 1E5E B1 E6           LDAIY CURADL FETCH OP CODE FROM WS
5270: 1E60 A0 01     LENACC LDYIM $01    LENGTH OF OP CODE IS 1 BYTE
5280: 1E62 C9 00           CMPIM $00
5290: 1E64 F0 1A           BEQ   LENEND BRK INSTRUCTION?
5300: 1E66 C9 40           CMPIM $40
5310: 1E68 F0 16           BEQ   LENEND RTI INSTRUCTION?
5320: 1E6A C9 60           CMPIM $60
5330: 1E6C F0 12           BEQ   LENEND RTS INSTRUCTION?
5340: 1E6E A0 03           LDYIM $03
5350: 1E70 C9 20           CMPIM $20
5360: 1E72 F0 0C           BEQ   LENEND JSR INSTRUCTION?
5370: 1E74 29 1F           ANDIM $1F    STRIP TO 5 BITS
5380: 1E76 C9 19           CMPIM $19
5390: 1E78 F0 06           BEQ   LENEND ANY ABS,Y INSTRUCTION?
5400: 1E7A 29 0F           ANDIM $0F    STRIP TO 4 BITS
5410: 1E7C AA              TAX          USE NIBBLE AS INDEX
5420: 1E7D BC 1F 1F        LDYX  LEN    FETCH LENGTH FROM LEN
5430: 1E80 84 F6     LENEND STYZ  BYTES  DISCARD LENGTH IN BYTES
5440: 1E82 60              RTS
5450:
5460:                      UP MOVES A DATA FIELD BETWEEN CURAD AND CEND
5470:                      UPWARD BY THE AMOUNT IN BYTES
5480:
5490: 1E83 A5 E6     UP    LDAZ  CURADL
5500: 1E85 85 EA           STAZ  MOVADL
5510: 1E87 A5 E7           LDAZ  CURADH MOVAD:=CURAD
5520: 1E89 85 EB           STAZ  MOVADH
5530: 1E8B A4 F6     UPLOOP LDYZ  BYTES
5540: 1E8D B1 EA           LDAIY MOVADL MOVE UPWARD BY THE AMOUNT IN BYTES
5550: 1E8F A0 00           LDYIM $00
5560: 1E91 91 EA           STAIY MOVADL
5570: 1E93 E6 EA           INCZ  MOVADL
5580: 1E95 D0 02           BNE   UPA
5590: 1E97 E6 EB           INCZ  MOVADH MOVADH:=MOVADH+1
5600: 1E99 A5 EA     UPA   LDAZ  MOVADL
5610: 1E9B C5 E8           CMPZ  CENDL
5620: 1E9D D0 EC           BNE   UPLOOP ALL DATA MOVED?
5630: 1E9F A5 EB           LDAZ  MOVADH IF NOT, CONTINUE
5640: 1EA1 C5 E9           CMPZ  CENDH
5650: 1EA3 D0 E6           BNE   UPLOOP
5660: 1EA5 60              RTS
5670:
5680:                      DOWN MOVES A DATA FIELD BETWEEN CURAD
5690:                      AND CEND DOWNWARD BY THE AMOUNT IN BYTES
```

```
5700:
5710: 1EA6 A5 E8     DOWN    LDAZ  CENDL
5720: 1EA8 85 EA             STAZ  MOVADL MOVAD:=CEND
5730: 1EAA A5 E9             LDAZ  CENDH
5740: 1EAC 85 EB             STAZ  MOVADH
5750: 1EAE A0 00     DNLOOP LDYIM $00
5760: 1EB0 B1 EA             LDAIY MOVADL MOVE DOWNWARD BY THE AMOUNT IN BYTES
5770: 1EB2 A4 F6             LDYZ  BYTES
5780: 1EB4 91 EA             STAIY MOVADL
5790: 1EB6 A5 EA             LDAZ  MOVADL
5800: 1EB8 C5 E6             CMPZ  CURADL
5810: 1EBA D0 06             BNE   DNA     ALL DATA MOVED?
5820: 1EBC A5 EB             LDAZ  MOVADH IF NOT, CONTINUE
5830: 1EBE C5 E7             CMPZ  CURADH
5840: 1EC0 F0 10             BEQ   DNEND
5850: 1EC2 38        DNA     SEC
5860: 1EC3 A5 EA             LDAZ  MOVADL
5870: 1EC5 E9 01             SBCIM $01
5880: 1EC7 85 EA             STAZ  MOVADL
5890: 1EC9 A5 EB             LDAZ  MOVADH MOVAD:=MOVAD-1
5900: 1ECB E9 00             SBCIM $00
5910: 1ECD 85 EB             STAZ  MOVADH
5920: 1ECF 4C AE 1E          JMP   DNLOOP
5930: 1ED2 60        DNEND   RTS
5940:
5950:
5960:                        BEGIN SETS CURAD EQUAL TO BEGAD
5970:
5980: 1ED3 A5 E2     BEGIN   LDAZ  BEGADL
5990: 1ED5 85 E6             STAZ  CURADL
6000: 1ED7 A5 E3             LDAZ  BEGADH CURAD:=BEGAD
6010: 1ED9 85 E7             STAZ  CURADH
6020: 1EDB 60                RTS
6030:
6040:                        ADCEND ADVANCES CURRENT END ADDRESS
6050:                        DOWNWARD BY THE AMOUNT IN BYTES
6060:
6070: 1EDC 18        ADCEND CLC
6080: 1EDD A5 E8             LDAZ  CENDL
6090: 1EDF 65 F6             ADCZ  BYTES   CEND:=CEND+BYTES
6100: 1EE1 85 E8             STAZ  CENDL
6110: 1EE3 A5 E9             LDAZ  CENDH
6120: 1EE5 69 00             ADCIM $00
6130: 1EE7 85 E9             STAZ  CENDH
6140: 1EE9 60                RTS
6150:
6160:                        RECEND REDUCES THE CURRENT END ADDRESS
6170:                        BY THE AMOUNT IN BYTES
6180:
6190: 1EEA 38        RECEND SEC
6200: 1EEB A5 E8             LDAZ  CENDL
6210: 1EED E5 F6             SBCZ  BYTES   CEND:=CEND-BYTES
6220: 1EEF 85 E8             STAZ  CENDL
6230: 1EF1 A5 E9             LDAZ  CENDH
6240: 1EF3 E9 00             SBCIM $00
6250: 1EF5 85 E9             STAZ  CENDH
6260: 1EF7 60                RTS
6270:
6280:                        NEXT ADVANCES THE CURRENT DISPLAYED ADDRES
6290:                        DOWNWARD BY THE AMOUNT IN BYTES
6300:
6310: 1EF8 18        NEXT    CLC
6320: 1EF9 A5 E6             LDAZ  CURADL
6330: 1EFB 65 F6             ADCZ  BYTES   CURAD:=CURAD+BYTES
6340: 1EFD 85 E6             STAZ  CURADL
6350: 1EFF A5 E7             LDAZ  CURADH
6360: 1F01 69 00             ADCIM $00
6370: 1F03 85 E7             STAZ  CURADH
6380: 1F05 38                SEC
6390: 1F06 A5 E6             LDAZ  CURADL
6400: 1F08 E5 E8             SBCZ  CENDL
6410: 1F0A A5 E7             LDAZ  CURADH
6420: 1F0C E5 E9             SBCZ  CENDH
6430: 1F0E 60                RTS
6440:
6450:                        THE LOOKUP TABLE "LOOK" IS USED, TO CONVERT
6460:                        A HEX NUMBER INTO A 7 SEGMENT PATTERN.
6470:                        THE LOOKUP TABLE "LEN" IS USED, TO CONVERT AN
6480:                        INSTRUCTION INTO AN INSTRUCTION LENGTH.
6490:
6500:
6510: 1F0F 40        LOOK    =     $40       "0"
6520: 1F10 79                =     $79       "1"
6530: 1F11 24                =     $24       "2"
6540: 1F12 30                =     $30       "3"
6550: 1F13 19                =     $19       "4"
6560: 1F14 12                =     $12       "5"
6570: 1F15 02                =     $02       "6"
6580: 1F16 78                =     $78       "7"
6590: 1F17 00                =     $00       "8"
6600: 1F18 10                =     $10       "9"
6610: 1F19 08                =     $08       "A"
6620: 1F1A 03                =     $03       "B"
6630: 1F1B 46                =     $46       "C"
6640: 1F1C 21                =     $21       "D"
```

```
6650: 1F1D 06               =     $06         "E"
6660: 1F1E 0E               =     $0E         "F"
6670:
6680: 1F1F 02         LEN   =     $02
6690: 1F20 02               =     $02
6700: 1F21 02               =     $02
6710: 1F22 01               =     $01
6720: 1F23 02               =     $02
6730: 1F24 02               =     $02
6740: 1F25 02               =     $02
6750: 1F26 01               =     $01
6760: 1F27 01               =     $01
6770: 1F28 02               =     $02
6780: 1F29 01               =     $01
6790: 1F2A 01               =     $01
6800: 1F2B 03               =     $03
6810: 1F2C 03               =     $03
6820: 1F2D 03               =     $03
6830: 1F2E 03               =     $03
6840:
6850: 1F2F 6C 7A 1A         JMI   NMIL   JUMP TO A  USER SELECTABLE NMI VECTOR
6860: 1F32 6C 7E 1A         JMI   IRQL   JUMP TO A  USER SELECTABLE IRQ VECTOR
6870:
6880:                 GETLBL IS AN ASSEMBLER SUBROUTINE. IT SEARCHES FOR
6890:                 LABELS ON THE SYMBOL PSEUDO STACK. IF THIS STACK
6900:                 CONTAINS A VALID LABEL, IT RETURNS WITH THE
6910:                 HIGH ORDER LABEL ADDRESS IN X AND THE LOW ORDER LABEL
6920:                 ADDRESS IN A. IF NO VALID LABEL IS FOUND, IT RE-
6930:                 TURNES WITH Z=1.
6940:
6950: 1F35 B1 E6      GETLBL LDAIY CURADL FETCH CURRENT LABEL NUMBER FROM WS
6960: 1F37 A0 FF             LDYIM $FF    RESET PSEUDO STACK
6970: 1F39 C4 EE      SYMA   CPYZ  LABELS UPPER MOST SYMBOL TABLE ADDRESS?
6980: 1F3B F0 0D             BEQ   SYMB   IF YES, RETURN, NO LABEL ON PSEUDO STACK
6990: 1F3D D1 EC             CMPIY TABLEL LABEL NR. IN WS = LABEL NR. ON PSEUDO STACK?
7000: 1F3F D0 0A             BNE   SYMNXT
7010: 1F41 88                DEY          IF YES, GET HIGH ORDER ADD
7020: 1F42 B1 EC             LDAIY TABLEL
7030: 1F44 AA                TAX          DISCARD HIGH ORDER ADD IN X
7040: 1F45 88                DEY
7050: 1F46 B1 EC             LDAIY TABLEL GET LOW ORDER ADD
7060: 1F48 A0 01             LDYIM $01     PREPARE Y REGISTER
7070: 1F4A 60         SYMB   RTS
7080:
7090: 1F4B 88         SYMNXT DEY                    *********   *********
7100: 1F4C 88                DEY                    * X=ADH *   * A=ADL *
7110: 1F4D 88                DEY                    *********   *********
7120: 1F4E D0 E9             BNE   SYMA
7130: 1F50 60                RTS
7140:
7150:
7160:
7170:
7180:
7190:
7200:
7210:                 ASSEMBLER MAIN ROUTINE
7220:
7230:                 FOLLOWING INSTRUCTIONS ARE ASSEMBLED:
7240:
7250:                 JSR INSTRUCTION
7260:                 JMP INSTRUCTION
7270:                 BRANCH INSTRUCTIONS
7280:
7290:
7300:
7310:
7320:
7330:
7340: 1F51 38         ASSEMB SEC
7350: 1F52 A5 E4             LDAZ  ENDADL
7360: 1F54 E9 FF             SBCIM $FF
7370: 1F56 85 EC             STAZ  TABLEL TABLE:=ENDAD-$FF
7380: 1F58 A5 E5             LDAZ  ENDADH
7390: 1F5A E9 00             SBCIM $00
7400: 1F5C 85 ED             STAZ  TABLEH
7410: 1F5E A9 FF             LDAIM $FF
7420: 1F60 85 EE             STAZ  LABELS
7430: 1F62 20 D3 1E          JSR   BEGIN  CURAD:=BEGAD
7440:
7450: 1F65 20 5C 1E   PASSA  JSR   OPLEN  START PASS ONE, GET CURR. INSTR.
7460: 1F68 A0 00             LDYIM $00
7470: 1F6A B1 E6             LDAIY CURADL FETCH CURRENT INSTRUCTION
7480: 1F6C C9 FF             CMPIM $FF    IS THE CURRENT INSTR. A LABEL?
7490: 1F6E D0 1D             BNE   NXTINS
7500: 1F70 C8                INY
7510: 1F71 B1 E6             LDAIY CURADL IF YES, FETCH LABEL NR.
7520: 1F73 A4 EE             LDYZ  LABELS
7530: 1F75 91 EC             STAIY TABLEL DEPOSIT LABEL NR. ON SYMBOL STACK
7540: 1F77 88                DEY
7550: 1F78 A5 E7             LDAZ  CURADH GET HIGH ORDER ADD
7560: 1F7A 91 EC             STAIY TABLEL DEPOSIT ON SYMBOL STACK
7570: 1F7C 88                DEY
7580: 1F7D A5 E6             LDAZ  CURADL GET LOW ORDER ADD
7590: 1F7F 91 EC             STAIY TABLEL DEPOSIT ON SYMBOL STACK
```

```
7600: 1F81 88                DEY
7610: 1F82 84 EE             STYZ  LABELS ADJUST PSEUDO STACK POINTER
7620: 1F84 20 83 1E          JSR   UP     DELETE CURRENT LABEL IN WS
7630: 1F87 20 EA 1E          JSR   RECEND ADJUST CURRENT END ADD
7640: 1F8A 4C 65 1F          JMP   PASSA  LOOK FOR MORE LABELS
7650:
7660: 1F8D 20 F8 1E  NXTINS  JSR   NEXT   IF NO LABEL, SKIP TO NEXT INSTR.
7670: 1F90 30 D3             BMI   PASSA  ALL LABELS IN WS COLLECTED?
7680: 1F92 20 D3 1E          JSR   BEGIN  START PASS 2
7690: 1F95 20 5C 1E  PASSB   JSR   OPLEN  GET LENGTH OF THE CURRENT INSTR.
7700: 1F98 A0 00             LDYIM $00
7710: 1F9A B1 E6             LDAIY CURADL FETCH CURRENT INSTR.
7720: 1F9C C9 4C             CMPIM $4C    JMP INSTR.?
7730: 1F9E F0 16             BEQ   JUMPS
7740: 1FA0 C9 20             CMPIM $20    JSR INSTR.?
7750: 1FA2 F0 12             BEQ   JUMPS
7760: 1FA4 29 1F             ANDIM $1F    STRIP TO 5 BITS
7770: 1FA6 C9 10             CMPIM $10    ANY BRANCH INSTRUCTION?
7780: 1FA8 F0 1A             BEQ   BRINST
7790: 1FAA 20 F8 1E  PB      JSR   NEXT   IF NOT, RETURN
7800: 1FAD 30 E6             BMI   PASSB  ALL LABELS BETWEEN CURAD AND ENDAD ASSEMBLED?
7810: 1FAF A9 03             LDAIM $03    ENABLE 3 DISPLAY BUFFERS
7820: 1FB1 85 F6             STAZ  BYTES
7830: 1FB3 4C 33 1C          JMP   START  EXIT HERE  ********
7840:
7850:
7860: 1FB6 C8        JUMPS   INY          SET POINTER TO LABEL NR.
7870: 1FB7 20 35 1F          JSR   GETLBL GET LABEL ADD
7880: 1FBA F0 EE             BEQ   PB     RETURN, IF NOT FOUND
7890: 1FBC 91 E6             STAIY CURADL STORE LOW ORDER ADD
7900: 1FBE 8A                TXA
7910: 1FBF C8                INY
7920: 1FC0 91 E6             STAIY CURADL STORE HIGH ORDER ADD
7930: 1FC2 D0 E6             BNE   PB
7940:
7950: 1FC4 C8        BRINST  INY          SET POINTER TO LABEL NR.
7960: 1FC5 20 35 1F          JSR   GETLBL GET LABEL ADD.
7970: 1FC8 F0 E0             BEQ   PB     RETURN, IF LABEL NOT FOUND
7980: 1FCA 38                SEC
7990: 1FCB E5 E6             SBCZ  CURADL COMPUTE BRANCH OFFSET
8000: 1FCD 38                SEC
8010: 1FCE E9 02             SBCIM $02    DESTINATION-SOURCE-2=OFFSET
8020: 1FD0 91 E6             STAIY CURADL INSERT BRANCH OFFSET IN WS
8030: 1FD2 4C AA 1F          JMP   PB
8040:
8050:
8060:
8070:
8080:
8090:                THE SUBROUTINE BRANCH COMPUTES THE OFFSET OF BRANCH
8100:                INSTRUCTIONS. THE 2 RIGHT HAND DISPLAYS SHOW THE
8110:                COMPUTED OFFSET DEFINED BY THE 4 LEFT HAND DISPLAYS.
8120:                THE PROGRAM MUST BE STOPPED WITH THE RESET KEY.
8130:
8140: 1FD5 D8        BRANCH  CLD
8150: 1FD6 A9 00             LDAIM $00    RESET DISPLAY BUFFER
8160: 1FD8 85 FB             STAZ  POINTH
8170: 1FDA 85 FA             STAZ  POINTL
8180: 1FDC 85 F9             STAZ  INH
8190: 1FDE 20 6F 1D  BR      JSR   GETBYT READ SOURCE
8200: 1FE1 10 F2             BPL   BRANCH COMMAND KEY?
8210: 1FE3 85 FB             STAZ  POINTH SAVE SOURCE IN BUFFER
8220: 1FE5 20 6F 1D          JSR   GETBYT READ DESTINATION
8230: 1FE8 10 EB             BPL   BRANCH COMMAND KEY
8240: 1FEA 85 FA             STAZ  POINTL SAVE DESTINATION IN BUFFER
8250: 1FEC 18                CLC
8260: 1FED A5 FA             LDAZ  POINTL FETCH DESTINATION
8270: 1FEF E5 FB             SBCZ  POINTH SUBTRACT SOURCE
8280: 1FF1 85 F9             STAZ  INH
8290: 1FF3 C6 F9             DECZ  INH    EQUALIZE AND SAVE OFFSET IN BUFFER
8300: 1FF5 4C DE 1F          JMP   BR
8310:
8320:                VECTORS AT THE END OF THE MEMORY:
8330:
8340:                1FFA $2F   NMI VECTOR
8350:                1FFB $1F
8360:                1FFC $1D   RESET VECTOR
8370:                1FFD $1C
8380:                1FFE $32   IRQ OR BRK VECTOR
8390:                1FFF $1F
8400:
8410:                END OF JUNIOR'S MONITOR
8420:
8430:
8440:
8450:
8460:
8470:
8480:
8490:
8500:
8510:
8520:
8530:
```

```
8540: ACC     00F3   ADCEND 1EDC   ADDRES 1CA3   ADLOOP 1CA5
8550: ADMODE  1C5C   AK     1DB1   AKA    1DB7   ASSEMB 1F51
8560: BEGADH  00E3   BEGADL 00E2   BEGIN  1ED3   BR     1FDE
8570: BRANCH  1FD5   BRINST 1FC4   BYTEND 1D87   BYTES  00F6
8580: CENDH   00E9   CENDL  00E8   CMND   1CCA   CNTA   1AF4
8590: CNTB    1AF5   CNTC   1AF6   CNTD   1AF7   CNTE   1AFC
8600: CNTF    1AFD   CNTG   1AFE   CNTH   1AFF   CONVD  1DDF
8610: COUNT   00F7   CURADH 00E7   CURADL 00E6   DAMODE 1C66
8620: DATA    1C90   DELAY  1DEB   DELETE 1D2C   DNA    1EC2
8630: DNEND   1ED2   DNLOOP 1EAE   DOWN   1EA6   EDETA  1AE4
8640: EDETB   1AE5   EDETD  1AE7   EDIT   1CC0   EDITOR 1CB5
8650: ENDADH  00E5   ENDADL 00E4   ERRA   1D39   ERRB   1D45
8660: FILBUF  1D51   FILLWS 1E47   GETBYT 1D6F   GETKEA 1DFB
8670: GETKEY  1DF9   GETLBL 1F35   GOEXEC 1C45   ILLKEY 1C8C
8680: INH     00F9   INL    00F8   INPUT  1D09   INSERT 1CFB
8690: IRQH    1A7F   IRQL   1A7E   JUMPS  1FB6   KEYIN  1E09
8700: KEYINA  1E0B   KEYINB 1E11   KEYINC 1E19   KEYIND 1E1C
8710: KEY     00E1   LABELS 00EE   LENACC 1E60   LENEND 1E80
8720: LEN     1F1F   LOOK   1F0F   LOYS   1C00   MODE   00FF
8730: MOVADH  00EB   MOVADL 00EA   NEXT   1EF8   NIBBLE 00FE
8740: NMIH    1A7B   NMIL   1A7A   NXTINS 1FBD   ONEKEY 1DB5
8750: OPLEN   1E5C   PADD   1A81   PAD    1A80   PASSA  1F65
8760: PASSB   1F95   PB     1FAA   PBDD   1A83   PBD    1A82
8770: PCH     00F0   PCKEY  1C7D   PCL    00EF   POINTH 00FB
8780: POINTL  00FA   PREG   00F1   RDA    1E44   RDB    1E46
8790: RDFLAG  1AD5   RDINST 1E20   RECEND 1EEA   RESET  1C1D
8800: SAVE    1C00   SAVEA  1C05   SAVEB  1C0F   SCAN   1D4D
8810: SCANA   1D5C   SCANB  1D61   SCAND  1D88   SCANDS 1D8E
8820: SCDSA   1D97   SCDSB  1DAC   SEARA  1CF1   SEARCH 1CCD
8830: SELOOP  1CE2   SHOW   1DCC   SKIP   1D21   SPUSER 00F2
8840: STARA   1C38   START  1C33   STEP   1C70   STEPA  1C7A
8850: SYMA    1F39   SYMB   1F4A   SYMNXT 1F4B   TABLEH 00ED
8860: TABLEL  00EC   TEMP   00FC   TEMPX  00FD   UP     1E83
8870: UPA     1E99   UPLOOP 1E8B   WDETC  1AE6   WS     1E51
8880: XREG    00F5   YREG   00F4
```

3 Source-Listing

* **Conversione binario/decimale**
* **Routine Demo**
* **Routine PLAY**
* **Routine INPUT**
* **Routine REPEAT**
* **Subroutine diverse**
* **Routine d'Interrupt diverse**

```
0010:                BINARY DECIMAL CONVERSION
0020:
0030:
0040: 0200                 ORG   $0200
0050:
0060:                MEMORY CELLS
0070:
0080: 0200           INH    *     $00F9  DISPLAY BUFFERS
0090: 0200           POINTL *     $00FA
0100: 0200           POINTH *     $00FB
0110: 0200           HEXL   *     $00D7  DATA BUFFERS
0120: 0200           HEXH   *     $00D8
0130:
0140:                MONITOR SUBROUTINE GETBYT
0150:
0160: 0200           GETBYT *     $1D6F  KEYBOARD & DISPLAY SCAN
0170:
0180:                START OF DISPL
0190:
0200: 0200 A9 00     DISPL  LDAIM $00    DISPLAY 000000
0210: 0202 85 F9            STAZ  INH
0220: 0204 85 FA            STAZ  POINTL
0230: 0206 85 FB            STAZ  POINTH
0240: 0208 20 6F 1D  DA     JSR   GETBYT READ KEYBOARD, SCAN DISPLAY
0250: 020B 10 F3            BPL   DISPL  RETURN IF COMMAND KEY
0260: 020D 85 F9            STAZ  INH    BYTE TO DISPLAY BUFFER
0270: 020F 85 D7            STAZ  HEXL   BYTE TO DATA BUFFER
0280: 0211 20 17 02         JSR   HEXDEC BINARY DECIMAL CONVERSION
0290: 0214 4C 08 02         JMP   DA     WAIT FOR A NEW BYTE
0300:
0310:
0320:                SUBROUTINES OF THE CONVERSION PROGRAM
0330:
0340: 0217 20 2E 02  HEXDEC JSR   COMNUM COMPUTE ONES
0350: 021A 85 FA            STAZ  POINTL DISCARD ONES
0360: 021C 84 D7            STYZ  HEXL   GET CONTENTS OF THE SUBTRACTION COUNTER
0370: 021E 20 2E 02         JSR   COMNUM COMPUTE TENS
0380: 0221 A2 04            LDXIM $04    SET SHIFT COUNTER
0390: 0223 0A        HD     ASLA         SHIFT LEFT
0400: 0224 CA               DEX          ALL SHIFTS DONE
0410: 0225 D0 FC            BNE   HD      IF NOT, CONTINUE
0420: 0227 05 FA            ORAZ  POINTL TENS & ONES INTO 1 BYTE
0430: 0229 85 FA            STAZ  POINTL
0440: 022B 84 FB            STYZ  POINTH HUNDREDS TO DISPLAY BUFFER
0450: 022D 60               RTS
0460:
0470: 022E A0 00     COMNUM LDYIM $00    RESET HIGH ORDER HEX BUFFER
0480: 0230 84 D8            STYZ  HEXH
0490: 0232 20 3B 02         JSR   SUBTRA SUBTRACT Y*$0A
0500: 0235 18               CLC
0510: 0236 A5 D7            LDAZ  HEXL   CORRECT SUBTRACTION ERROR
0520: 0238 69 0A            ADCIM $0A
0530: 023A 60               RTS
0540:
0550: 023B 38        SUBTRA SEC
0560: 023C A5 D7            LDAZ  HEXL   16 BIT SUBTRACTION
0570: 023E E9 0A            SBCIM $0A
0580: 0240 85 D7            STAZ  HEXL
0590: 0242 A5 D8            LDAZ  HEXH
0600: 0244 E9 00            SBCIM $00
0610: 0246 30 04            BMI   SUB    COMPLETE SUBTRACTION, IF RESULT NEGATIVE
0620: 0248 C8               INY          SUBTRACTION COUNTER = Y+1
0630: 0249 4C 3B 02         JMP   SUBTRA CONTINUE SUBTRACTION
0640: 024C 60        SUB    RTS
0650:
0660:                END OF DISPL

    SYMBOL TABLE
    COMNUM 022E    DA     0208    DISPL  0200    GETBYT 1D6F
    HD     0223    HEXDEC 0217    HEXH   00D8    HEXL   00D7
    INH    00F9    POINTH 00FB    POINTL 00FA    SUBTRA 023B
    SUB    024C


    SYMBOL TABLE
    HEXL   00D7    HEXH   00D8    INH    00F9    POINTL 00FA
    POINTH 00FB    DISPL  0200    DA     0208    HEXDEC 0217
    HD     0223    COMNUM 022E    SUBTRA 023B    SUB    024C
    GETBYT 1D6F

0010:                DEMO ROUTINE
0020:
0030: 0000                 ORG   $0000
0040:
0050:                I/O DEFINITION
0060:
0070: 0000           PAD    *     $1A80  DATA REGISTER
0080: 0000           PADD   *     $1A81  DATA DIRECTION REGISTER
0090: 0000           PBD    *     $1A82  DATA REGISTER
0100: 0000           PBDD   *     $1A83  DATA DIRECTION REGISTER
0110:
0120: 0000 A2 00     DEMO   LDXIM $00
0130: 0002 8E 81 1A         STX   PADD   PA0...PA7 IS INPUT
0140: 0005 E8               INX
0150: 0006 8E 83 1A         STX   PBDD   PB0 IS OUTPUT
```

```
0160:
0170: 0009 AD 80 1A  FREQ   LDA   PAD    READ SWITCH PATTERN
0180: 000C 49 FF            EORIM $FF    INVERT PATTERN
0190: 000E A8 00            LDYIM $00    B0 IS ZERO
0200: 0010 8C 82 1A         STY   PBD    TOGGLE SPEAKER ON
0210: 0013 20 20 00         JSR   DELAY  DELAY = SWITCHES * LOOP TIME
0220: 0016 C8               INY
0230: 0017 8C 82 1A         STY   PBD    TOGGLE SPEAKER OFF
0240: 001A 20 20 00         JSR   DELAY  DELAY = SWITCHES * LOOP TIME
0250: 001D 4C 09 00         JMP   FREQ   RETURN
0260:
0270:                SUBROUTINE DELAY
0280:
0290: 0020 AA        DELAY  TAX          X-REGISTER IS THE DELAY COUNTER
0300: 0021 CA        DEL    DEX          DELAY LOOP
0310: 0022 D0 FD            BNE   DEL
0320: 0024 60               RTS
0330:

    SYMBOL TABLE
    DELAY  0020    DEL    0021    DEMO   0000    FREQ   0009
    PADD   1A81    PAD    1A80    PBDD   1A83    PBD    1A82


    SYMBOL TABLE
    DEMO   0000    FREQ   0009    DELAY  0020    DEL    0021
    PAD    1A80    PADD   1A81    PBD    1A82    PBDD   1A83

0010:                PLAY ROUTINE
0020:
0030: 0000                 ORG   $0000
0040:
0050:                TEMPORARY DATA BUFFERS IN PAGE ZERO
0060:
0070: 0000           ROW    *     $00D9  ROW BUFFER
0080: 0000           KEY    *     $00DA  KEY VALUE BUFFER
0090: 0000           TEMPX  *     $00DB  ROW NUMBER BUFFER
0100:
0110:                I/O DEFINITION
0120:
0130: 0000           PAD    *     $1A80
0140: 0000           PADD   *     $1A81
0150: 0000           PBD    *     $1A82
0160: 0000           PBDD   *     $1A83
0170:
0180:
0190: 0000 A9 F0     PLAY   LDAIM $F0    PA7...PA4 IS OUTPUT
0200: 0002 8D 81 1A         STA   PADD   PA3...PA0 IS INPUT
0210: 0005 A9 01            LDAIM $01
0220: 0007 8D 83 1A         STA   PBDD   PB0 IS OUTPUT
0230: 000A 8D 82 1A         STA   PBD    TOGGLE SPEAKER OFF
0240: 000D A9 00            LDAIM $00
0250: 000F 8D 80 1A         STA   PAD    ALL MATRIX ROWS ARE ZERO
0260:
0270: 0012 20 9C 00  PA     JSR   KEYIN   ANY KEY DEPRESSED?
0280: 0015 F0 FB            BEQ   PA     BRANCH, IF NO
0290: 0017 20 A4 00         JSR   DELAY  DEBOUNCE THE KEY
0300: 001A 20 9C 00         JSR   KEYIN   KEY STILL DEPRESSED
0310: 001D F0 F3            BEQ   PA     IF YES, CONTINUE
0320: 001F 20 44 00         JSR   KEYVAL COMPUTE THE KEY VALUE
0330: 0022 A4 DA            LDYZ  KEY    GET KEY VALUE
0340:
0350: 0024 A9 00     TONE   LDAIM $00    B0 = 0
0360: 0026 8D 82 1A         STA   PBD    TOGGLE SPEAKER ON
0370: 0029 BE 00 1A         LDXY  DEL    FETCH THE FREQUENCY
0380: 002C 20 AA 00  TA     JSR   EQUAL  EQUALIZE 22 MICRO SEC.
0390: 002F CA               DEX          HALF PERIODE PASSED?
0400: 0030 D0 FA            BNE   TA     WAIT, IF NOT
0410: 0032 A9 01            LDAIM $01    B0 = 1
0420: 0034 8D 82 1A         STA   PBD    TOGGLE SPEAKER OFF
0430: 0037 BE 00 1A         LDXY  DEL    FETCH THE FREQUENCY AGAIN
0440: 003A 20 9C 00  TB     JSR   KEYIN   ANY KEY STILL DEPRESSED?
0450: 003D F0 D3            BEQ   PA     IF YES, GENERATE A TONE
0460: 003F CA               DEX          HALF PERIODE PASSED?
0470: 0040 D0 F8            BNE   TB     WAIT, IF NOT
0480: 0042 F0 E0            BEQ   TONE   CONTINUE, IF YES
0490:
0500:

    SYMBOL TABLE
    DELA   00A6    DELAY  00A4    DEL    1A00    EQUAL  00AA
    KEYA   004A    KEYB   0063    KEYC   0094    KEYIN  009C
    KEYVAL 0044    KEY    00DA    PA     0012    PADD   1A81
    PAD    1A80    PBDD   1A83    PBD    1A82    PLAY   0000
    ROWA   006E    ROWB   0078    ROWC   0082    ROWD   008C
    ROW    00D9    TA     002C    TB     003A    TEMPX  00DB
    TONE   0024


    SYMBOL TABLE
    PLAY   0000    PA     0012    TONE   0024    TA     002C
    TB     003A    KEYVAL 0044    KEYA   004A    KEYB   0063
    ROWA   006E    ROWB   0078    ROWC   0082    ROWD   008C
    KEYC   0094    KEYIN  009C    DELAY  00A4    DELA   00A6
    EQUAL  00AA    ROW    00D9    KEY    00DA    TEMPX  00DB
    DEL    1A00    PAD    1A80    PADD   1A81    PBD    1A82
    PBDD   1A83
```

```
0010:                   SUBROUTINES OF THE PLAY PROGRAM
0020:
0030: 0044 A9 F7        KEYVAL LDAIM $F7    ALL ROWS ARE ONE
0040: 0046 85 D9               STAZ  ROW
0050: 0048 A2 04               LDXIM $04    SET UP ROW COUNTER
0060: 004A CA           KEYA   DEX          RETURN IF INVALID ROW
0070: 004B 30 F7               BMI   KEYVAL IS FOUND
0080: 004D 06 D9               ASLZ  ROW    SPECIFIED ROW IS ZERO
0090: 004F A5 D9               LDAZ  ROW
0100: 0051 8D 80 1A            STA   PAD    OUTPUT ROW NUMBER
0110: 0054 AD 80 1A            LDA   PAD    IF NO KEY IS DEPRESSED IN THE
0120: 0057 29 0F               ANDIM $0F    SPECIFIED ROW, OUTPUT
0130: 0059 C9 0F               CMPIM $0F    NEXT ROW NUMBER
0140: 005B F0 ED               BEQ   KEYA
0150: 005D 86 DB               STXZ  TEMPX  SAVE ROW NUMBER
0160: 005F 85 DA               STAZ  KEY    SAVE COLUMN NUMBER
0170: 0061 A2 00               LDXIM $00
0180: 0063 46 DA        KEYB   LSRZ  KEY    SHIFT UNTIL CARRY CLEAR
0190: 0065 90 07               BCC   ROWA   BRANCH TO COMPUTE KEY VALUE
0200: 0067 E8                  INX
0210: 0068 E0 04               CPXIM $04    ALL ROWS SCANNED?
0220: 006A D0 F7               BNE   KEYB   IF NOT CONTINUE
0230: 006C F0 D6               BEQ   KEYVAL RETURN IF INVALID ROW NUMBER
0240: 006E A5 DB        ROWA   LDAZ  TEMPX  GET ROW NUMBER AGAIN
0250: 0070 C9 03               CMPIM $03    ROW 0?
0260: 0072 D0 04               BNE   ROWB
0270: 0074 8A                  TXA
0280: 0075 4C 94 00            JMP   KEYC
0290:
0300: 0078 C9 02        ROWB   CMPIM $02    ROW 1?
0310: 007A D0 06               BNE   ROWC
0320: 007C 8A                  TXA
0330: 007D 18                  CLC
0340: 007E 69 04               ADCIM $04
0350: 0080 D0 12               BNE   KEYC
0360:
0370: 0082 C9 01        ROWC   CMPIM $01    ROW 2?
0380: 0084 D0 06               BNE   ROWD
0390: 0086 8A                  TXA
0400: 0087 18                  CLC
0410: 0088 69 08               ADCIM $08
0420: 008A D0 08               BNE   KEYC
0430:
0440: 008C C9 00        ROWD   CMPIM $00    ROW 3?
0450: 008E D0 B4               BNE   KEYVAL RETURN, IF ROW IS INVALID
0460: 0090 8A                  TXA
0470: 0091 18                  CLC
0480: 0092 69 0C               ADCIM $0C
0490:
0500: 0094 85 DA        KEYC   STAZ  KEY    SAVE KEY VALUE
0510: 0096 A9 00               LDAIM $00    RESET PORT A
0520: 0098 8D 80 1A            STA   PAD
0530: 009B 60                  RTS
0540:
0550: 009C AD 80 1A     KEYIN  LDA   PAD
0560: 009F 29 0F               ANDIM $0F    MASK OFF HIGH ORDER NIBBLE
0570: 00A1 49 0F               EORIM $0F    IF NO KEY: ACCU = $00
0580: 00A3 60                  RTS
0590:
0600: 00A4 A0 FF        DELAY  LDYIM $FF    SET DELAY COUNTER
0610: 00A6 88           DELA   DEY
0620: 00A7 D0 FD               BNE   DELA   TIME OUT ?
0630: 00A9 60                  RTS
0640:
0650: 00AA EA           EQUAL  NOP          EQUALIZE 20 MICRO SEC
0660: 00AB EA                  NOP
0670: 00AC EA                  NOP
0680: 00AD EA                  NOP
0690: 00AE EA                  NOP
0700: 00AF 60                  RTS
0710:
0720: 1A00                     ORG   $1A00
0730:
0740:                   FREQUENCY LOOKUP TABLE
0750:
0760: 1A00 8E           DEL    =     $8E
0770: 1A01 86                  =     $86
0780: 1A02 7E                  =     $7E
0790: 1A03 77                  =     $77
0800: 1A04 70                  =     $70
0810: 1A05 6A                  =     $6A
0820: 1A06 64                  =     $64
0830: 1A07 5E                  =     $5E
0840: 1A08 59                  =     $59
0850: 1A09 54                  =     $54
0860: 1A0A 4E                  =     $4E
0870: 1A0B 4A                  =     $4A
0880: 1A0C 47                  =     $47
0890: 1A0D 43                  =     $43
0900: 1A0E 3E                  =     $3E
0910: 1A0F 3C                  =     $3C
0920:
0930:
0940:                   END OF PLAY
```

```
0010:                 INPUT ROUTINE
0020:
0030: 0200                    ORG   $0200
0040:
0050:                 TEMPORARY DATA BUFFERS IN PAGE ZERO
0060:
0070: 0200            ROW     *     $00D9
0080: 0200            KEY     *     $00DA
0090: 0200            TEMPX   *     $00DB
0100: 0200            NOTEL   *     $00DC   NOTE POINTER
0110: 0200            NOTEH   *     $00DD
0120: 0200            LENGTH  *     $00DE   TIME OF A DEPRESSED KEY
0130: 0200            ENDL    *     $00DF   END OF THE NOTE BUFFER
0140:
0150:                 INTERVAL TIMER
0160:
0170: 0200            CNTA    *     $1AF4   DISABLE TIMER IRQ
0180: 0200            CNTG    *     $1AFE   ENABLE TIMER IRQ, CLK64T
0190:
0200:                 GOTO MONITOR
0210:
0220: 0200            RESET   *     $1C1D   NEW I/O DEFINITION
0230:
0240:                 I/O DEFINITION
0250:
0260: 0200            PAD     *     $1A80
0270: 0200            PADD    *     $1A81
0280: 0200            PBD     *     $1A82
0290: 0200            PBDD    *     $1A83
0300:
0310:                 IRQ VECTOR
0320:
0330: 0200            IRQL    *     $1A7E
0340: 0200            IRQH    *     $1A7F
0350:
0360:
0370:                 START OF THE INPUT PROGRAM
0380:
0390: 0200 78         INPUT   SEI           DISABLE IRQ LINE
0400: 0201 D8                 CLD
0410: 0202 A9 20              LDAIM IRQIN   SET UP IRQ VECTOR
0420: 0204 8D 7E 1A           STA   IRQL
0430: 0207 A9 1A              LDAIM IRQIN   /256
0440: 0209 8D 7F 1A           STA   IRQH
0450: 020C A9 F0              LDAIM $F0     PA7...PA4 IS OUTPUT, PA3...PA0 IS INPUT
0460: 020E 8D 81 1A           STA   PADD
0470: 0211 A9 01              LDAIM $01
0480: 0213 8D 83 1A           STA   PBDD    PB0 IS OUTPUT FOR SPEAKER
0490: 0216 8D 82 1A           STA   PBD     TOGGLE SPEAKER OFF
0500: 0219 85 DD              STAZ  NOTEH   HIGH ORDER BYTE OF NOTE POINTER
0510: 021B A0 00              LDYIM $00
0520: 021D 8C 80 1A           STY   PAD     SET ALL ROWS ZERO
0530: 0220 84 DC              STYZ  NOTEL   LOW ORDER BYTE OF NOTE POINTER
0540: 0222 A0 D8              LDYIM $D8     DEFINE ENDADDRESS OF INPUT BUFFER
0550: 0224 84 DF              STYZ  ENDL
0560: 0226 A9 77              LDAIM $77     LOAD EOF CHARACTER
0570: 0228 91 DC      INA     STAIY NOTEL   FILLUP WORKSPACE WITH EOFS
0580: 022A 88                 DEY
0590: 022B C0 FF              CPYIM $FF     WS FILLED UP?
0600: 022D D0 F9              BNE   INA     IF NOT CONTINUE
0610: 022F 20 E0 02   KEYSCN  JSR   KEYIN   ANY KEY DEPRESSED?
0620: 0232 F0 FB              BEQ   KEYSCN  WAIT IF NO KEY IS DEPRESSED
0630: 0234 20 E8 02           JSR   DELAY   DEBOUNCE KEYBOARD
0640: 0237 20 E0 02           JSR   KEYIN   STILL ANY KEY DEPRESSED
0650: 023A F0 F3              BEQ   KEYSCN  IF YES, CONTINUE
0660: 023C 20 88 02           JSR   KEYVAL  COMPUTE KEY NUMBER
0670: 023F A9 00              LDAIM $00
0680: 0241 85 DE              STAZ  LENGTH  RESET TIME COUNTER
0690: 0243 A9 FF              LDAIM $FF     START TIMER, ENABLE TIMER IRQ,
0700: 0245 8D FE 1A           STA   CNTG    RESET IRQ LINE
0710: 0248 58                 CLI           ENABLE IRQ LINE
0720: 0249 A4 DA              LDYZ  KEY     LOOKUP CONVERSION BY KEY VALUE
0730: 024B A9 00      TONE    LDAIM $00     TOGGLE SPEAKER ON
0740: 024D 8D 82 1A           STA   PBD     PB0 IS LOG 0
0750: 0250 BE 00 1A           LDXY  DEL     FETCH DELAY
0760: 0253 20 EE 02   TA      JSR   EQUAL   EQUALIZE 20 MIKRO SEC
0770: 0256 CA                 DEX
0780: 0257 D0 FA              BNE   TA       DELAY
0790: 0259 A9 01              LDAIM $01     TOGGLE SPEAKER OFF
0800: 025B 8D 82 1A           STA   PBD
0810: 025E BE 00 1A           LDXY  DEL     FETCH DELAY AGAIN
0820: 0261 20 E0 02   TB      JSR   KEYIN   ANY KEY STILL DEPRESSED?
0830: 0264 F0 05              BEQ   STORE   BRANCH IF KEY IS RELEASED
0840: 0266 CA                 DEX
0850: 0267 D0 F8              BNE   TB
0860: 0269 F0 E0              BEQ   TONE    CONTINUE AS LONG AS A KEY IS DEPRESSED
0870: 026B 8D F4 1A   STORE   STA   CNTA    RESET IRQ LINE,DISABLE TIMER IRQ
0880: 026E A5 DC              LDAZ  NOTEL
0890: 0270 C5 DF              CMPZ  ENDL    IS WORKSPACE FULL?
0900: 0272 F0 11              BEQ   ST      IF YES, EXIT HERE
0910: 0274 98                 TYA
0920: 0275 A0 00              LDYIM $00
0930: 0277 91 DC              STAIY NOTEL   STORE KEY VALUE IN WS
0940: 0279 C8                 INY
0950: 027A A5 DE              LDAZ  LENGTH  GET TIME OF THE DEPRESSED KEY
```

```
0960: 027C 91 DC            STAIY NOTEL  STORE KEY TIME IN WS
0970: 027E E6 DC            INCZ  NOTEL
0980: 0280 E6 DC            INCZ  NOTEL  ADJUST NOTE POINTER
0990: 0282 4C 2F 02         JMP   KEYSCN
1000: 0285 4C 1D 1C  ST     JMP   RESET  BACK TO MONITOR
1010:
0010:                SUBROUTINES OF THE INPUT PROGRAM
0020:
0030: 0288 A9 F7     KEYVAL LDAIM $F7    ALL ROWS ARE ONE
0040: 028A 85 D9            STAZ  ROW
0050: 028C A2 04            LDXIM $04    SET UP ROW COUNTER
0060: 028E CA        KEYA   DEX          RETURN IF INVALID ROW
0070: 028F 30 F7            BMI   KEYVAL IS FOUND
0080: 0291 06 D9            ASLZ  ROW    SPECIFIED ROW IS ZERO
0090: 0293 A5 D9            LDAZ  ROW
0100: 0295 8D 80 1A         STA   PAD    OUTPUT ROW NUMBER
0110: 0298 AD 80 1A         LDA   PAD    IF NO KEY IS DEPRESSED IN THE
0120: 029B 29 0F            ANDIM $0F    SPECIFIED ROW, OUTPUT
0130: 029D C9 0F            CMPIM $0F    NEXT ROW NUMBER
0140: 029F F0 ED            BEQ   KEYA
0150: 02A1 86 DB            STXZ  TEMPX  SAVE ROW NUMBER
0160: 02A3 85 DA            STAZ  KEY    SAVE COLUMN NUMBER
0170: 02A5 A2 00            LDXIM $00
0180: 02A7 46 DA     KEYB   LSRZ  KEY    SHIFT UNTIL CARRY CLEAR
0190: 02A9 90 07            BCC   ROWA   BRANCH TO COMPUTE KEY VALUE
0200: 02AB E8               INX
0210: 02AC E0 04            CPXIM $04    ALL ROWS SCANNED?
0220: 02AE D0 F7            BNE   KEYB   IF NOT CONTINUE
0230: 02B0 F0 D6            BEQ   KEYVAL RETURN IF INVALID ROW NUMBER
0240: 02B2 A5 DB     ROWA   LDAZ  TEMPX  GET ROW NUMBER AGAIN
0250: 02B4 C9 03            CMPIM $03    ROW 0?
0260: 02B6 D0 04            BNE   ROWB
0270: 02B8 8A               TXA
0280: 02B9 4C D8 02         JMP   KEYC
0290:
0300: 02BC C9 02     ROWB   CMPIM $02    ROW 1?
0310: 02BE D0 06            BNE   ROWC
0320: 02C0 8A               TXA
0330: 02C1 18               CLC
0340: 02C2 69 04            ADCIM $04
0350: 02C4 D0 12            BNE   KEYC
0360:
0370: 02C6 C9 01     ROWC   CMPIM $01    ROW 2?
0380: 02C8 D0 06            BNE   ROWD
0390: 02CA 8A               TXA
0400: 02CB 18               CLC
0410: 02CC 69 08            ADCIM $08
0420: 02CE D0 08            BNE   KEYC
0430:
0440: 02D0 C9 00     ROWD   CMPIM $00    ROW 3?
0450: 02D2 D0 B4            BNE   KEYVAL RETURN, IF ROW IS INVALID
0460: 02D4 8A               TXA
0470: 02D5 18               CLC
0480: 02D6 69 0C            ADCIM $0C
0490:
0500: 02D8 85 DA     KEYC   STAZ  KEY    SAVE KEY VALUE
0510: 02DA A9 00            LDAIM $00    RESET PORT A
0520: 02DC 8D 80 1A         STA   PAD
0530: 02DF 60               RTS
0540:
0550: 02E0 AD 80 1A  KEYIN  LDA   PAD
0560: 02E3 29 0F            ANDIM $0F    MASK OFF HIGH ORDER NIBBLE
0570: 02E5 49 0F            EORIM $0F    IF NO KEY: ACCU = $00
0580: 02E7 60               RTS
0590:
0600: 02E8 A0 FF     DELAY  LDYIM $FF    SET DELAY COUNTER
0610: 02EA 88        DELA   DEY
0620: 02EB D0 FD            BNE   DELA   TIME OUT ?
0630: 02ED 60               RTS
0640:
0650: 02EE EA        EQUAL  NOP          EQUALIZE 20 MICRO SEC
0660: 02EF EA               NOP
0670: 02F0 EA               NOP
0680: 02F1 EA               NOP
0690: 02F2 EA               NOP
0700: 02F3 60               RTS
0710:
```

```
0720: 1A00                    ORG   $1A00
0730:
0740:                  FREQUENCY LOOKUP TABLE
0750:
0760: 1A00 8E         DEL     =     $8E
0770: 1A01 86                 =     $86
0780: 1A02 7E                 =     $7E
0790: 1A03 77                 =     $77
0800: 1A04 70                 =     $70
0810: 1A05 6A                 =     $6A
0820: 1A06 64                 =     $64
0830: 1A07 5E                 =     $5E
0840: 1A08 59                 =     $59
0850: 1A09 54                 =     $54
0860: 1A0A 4E                 =     $4E
0870: 1A0B 4A                 =     $4A
0880: 1A0C 47                 =     $47
0890: 1A0D 43                 =     $43
0900: 1A0E 3E                 =     $3E
0910: 1A0F 3C                 =     $3C
0920:
0930:
0940: 1A20                    ORG   $1A20
0950:
0960:                  TIMER INTERRUPT PROGRAM
0970:
0980: 1A20 48         IRQIN   PHA           SAVE ACCU
0990: 1A21 E6 DE              INCZ  LENGTH  INCREMENT TIME
1000: 1A23 A9 FF              LDA1M $FF     TIMER OFFSET IS $FF
1010: 1A25 8D FE 1A           STA   CNTG    START TIMER AGAIN
1020: 1A28 68                 PLA           RESTORE ACCU
1030: 1A29 40                 RTI
1040:
1050:                  END OF INPUT
```

```
SYMBOL TABLE
CNTA    1AF4    CNTG    1AFE    DELA    02EA    DELAY   02E8
DEL     1A00    ENDL    00DF    EQUAL   02EE    INA     0228
INPUT   020C    IRQH    1A7F    IRQIN   1A20    IRQL    1A7E
KEYA    028E    KEYB    02A7    KEYC    02D8    KEYIN   02E0
KEYSCN  022F    KEYVAL  0288    KEY     00DA    LENGTH  00DE
NOTEH   00DD    NOTEL   00DC    PADD    1A81    PAD     1A80
PBDD    1A83    PBD     1A82    RESET   1C1D    ROWA    02B2
ROWB    02BC    ROWC    02C6    ROWD    02D0    ROW     00D9
ST      0285    STORE   026B    TA      0253    TB      0261
TEMPX   00DB    TONE    024B
```

```
SYMBOL TABLE
ROW     00D9    KEY     00DA    TEMPX   00DB    NOTEL   00DC
NOTEH   00DD    LENGTH  00DE    ENDL    00DF    INPUT   0200
INA     0228    KEYSCN  022F    TONE    024B    TA      0253
TB      0261    STORE   026B    ST      0285    KEYVAL  0288
KEYA    028E    KEYB    02A7    ROWA    02B2    ROWB    02BC
ROWC    02C6    ROWD    02D0    KEYC    02D8    KEYIN   02E0
DELAY   02E8    DELA    02EA    EQUAL   02EE    DEL     1A00
IRQIN   1A20    IRQL    1A7E    IRQH    1A7F    PAD     1A80
PADD    1A81    PBD     1A82    PBDD    1A83    CNTA    1AF4
CNTG    1AFE    RESET   1C1D
```

```
0010:                   REPEAT ROUTINE
0020:
0030: 0000                      ORG   $0000
0040:
0050:                   TEMPORARY DATABUFFERS IN PAGE ZERO
0060:
0070: 0000              KEY     *     $00DA
0080: 0000              NOTEL   *     $00DC
0090: 0000              NOTEH   *     $00DD
0100: 0000              LENGTH  *     $00DE
0110:
0120:                   INTERVAL TIMER
0130:
0140: 0000              CNTA    *     $1AF4  DISABLE TIMER IRQ
0150: 0000              CNTD    *     $1AF7  DISABLE TIMER IRQ, CLK1KT
0160: 0000              CNTC    *     $1AFE  ENABLE TIMER IRQ, CLK64T
0170: 0000              RDFLAG  *     $1AD5  B7 IS TIMER FLAG
0180:
0190:                   GOTO MONITOR
0200:
0210: 0000              RESET   *     $1C1D  NEW I/O DEFINITION
0220:
0230:                   I/O DEFINITION
0240:
0250: 0000              PBD     *     $1A82
0260: 0000              PBDD    *     $1A83
0270:
0280:                   IRQ VECTOR
0290:
0300: 0000              IRQL    *     $1A7E
0310: 0000              IRQH    *     $1A7F
0320:
0330:
0340:                   START OF THE REPEAT PROGRAM
0350:
0360: 0000 78           REPEAT SEI            DISABLE IRQ LINE
0370: 0001 D8                  CLD
0380: 0002 A9 30               LDAIM IRQRE   SET UP IRQ VECTOR
0390: 0004 8D 7E 1A            STA   IRQL
0400: 0007 A9 1A               LDAIM IRQRE   /256
0410: 0009 8D 7F 1A            STA   IRQH
0420: 000C A9 01               LDAIM $01     PB0 IS OUTPUT
0430: 000E 8D 83 1A            STA   PBDD
0440: 0011 8D 82 1A            STA   PBD     TOGGLE SPEAKER OFF
0450: 0014 85 DD               STAZ  NOTEH   SET NOTE POINTER
0460: 0016 A9 00               LDAIM $00
0470: 0018 85 DC               STAZ  NOTEL   SET NOTE POINTER
0480: 001A 8D F4 1A            STA   CNTA    RESET IRQ LINE, DISABLE TIMER IRQ
0490: 001D 58                  CLI           ENABLE CPU IRQ
0500:
0510: 001E A9 FF        FETCH  LDAIM $FF      SET TIMER ENABLE TIMER IRQ
0520: 0020 8D FE 1A            STA   CNTC
0530: 0023 A0 00               LDYIM $00     FETCH NOTE
0540: 0025 B1 DC               LDAIY NOTEL
0550: 0027 85 DA               STAZ  KEY
0560: 0029 C8                  INY           FETCH LENGTH
0570: 002A B1 DC               LDAIY NOTEL
0580: 002C 85 DE               STAZ  LENGTH
0590: 002E A4 DA               LDYZ  KEY     LOOKUP CONVERSION
0600:
0610: 0030 A9 00        TONE   LDAIM $00     TOGGLE SPEAKER ON
0620: 0032 8D 82 1A            STA   PBD
0630: 0035 BE 00 1A            LDXY  DEL     GET FREQUENCY
0640: 0038 20 70 00     TONEA  JSR   EQUALA  DELAY 22 MICRO SEC
0650: 003B CA                  DEX
0660: 003C D0 FA               BNE   TONEA   LOOP TIME IS 27 MIKRO SEC*X
0670: 003E A9 01               LDAIM $01     TOGGLE SPEAKER OFF
0680: 0040 8D 82 1A            STA   PBD
0690: 0043 BE 00 1A            LDXY  DEL     GET FREQUENCY AGAIN
0700: 0046 A5 DE        TONEB  LDAZ  LENGTH  GET LENGTH
0710: 0048 30 08               BMI   TONEC   TIME OUT?
0720: 004A 20 74 00            JSR   EQUALB  EQUALIZE 17 MICRO SEC
0730: 004D CA                  DEX
0740: 004E D0 F6               BNE   TONEB   LOOP TIME IS 27 MICRO SEC*X AGAIN
0750: 0050 F0 DE               BEQ   TONE    RETURN AFTER ONE PERIODE
0760: 0052 A2 04        TONEC  LDXIM $04     LOOP TIME = 4*CNTD*PRESET
0770: 0054 A9 30        TONED  LDAIM $30     PRESET = $30
0780: 0056 8D F7 1A            STA   CNTD    DISABLE TIMER IRQ
0790:
0800: 0059 2C D5 1A     POLL   BIT   RDFLAG  READ FLAG REGISTER, TIME OUT?
0810: 005C 10 FB               BPL   POLL    IS TIMER FLAG STILL ZERO?
0820: 005E CA                  DEX
0830: 005F D0 F3               BNE   TONED   LOOP COUNTER ZERO?
0840: 0061 E6 DC               INCZ  NOTEL   ADJUST NOTE POINTER
0850: 0063 E6 DC               INCZ  NOTEL
0860: 0065 A0 00               LDYIM $00
0870: 0067 B1 DC               LDAIY NOTEL   END OF NOTE BUFFER?
0880: 0069 C9 77               CMPIM $77     EOF CHARACTER
0890: 006B D0 B1               BNE   FETCH   IF NOT EOF, CONTINUE
0900: 006D 4C 1D 1C            JMP   RESET   ELSE BACK TO MONITOR
0910:
```

```
0010:                   SUBROUTINES OF THE REPEAT PROGRAM
0020:
0030:                   17/22 MICRO SEC SUBROUTINE
0040:
0050: 0070 EA           EQUALA NOP
0060: 0071 4C 74 00             JMP   EQUALB
0070: 0074 EA           EQUALB NOP
0080: 0075 4C 78 00             JMP   EEND
0090: 0078 60           EEND   RTS
0100:
0110: 1A00                      ORG   $1A00
0120:
0130:                   FREQUENCY LOOKUP TABLE
0140:
0150: 1A00 8E           DEL    =      $8E
0160: 1A01 86                  =      $86
0170: 1A02 7E                  =      $7E
0180: 1A03 77                  =      $77
0190: 1A04 70                  =      $70
0200: 1A05 6A                  =      $6A
0210: 1A06 64                  =      $64
0220: 1A07 5E                  =      $5E
0230: 1A08 59                  =      $59
0240: 1A09 54                  =      $54
0250: 1A0A 4E                  =      $4E
0260: 1A0B 4A                  =      $4A
0270: 1A0C 47                  =      $47
0280: 1A0D 43                  =      $43
0290: 1A0E 3E                  =      $3E
0300: 1A0F 3C                  =      $3C
0310:
0320:
0330: 1A30                      ORG   $1A30
0340:
0350:                   TIMER INTERRUPT PROGRAM
0360:
0370: 1A30 48           IRQRE  PHA           SAVE ACCU
0380: 1A31 C6 DE               DEC2  LENGTH DECREMENT TIME
0390: 1A33 A9 FF               LDAIM $FF    TIMER OFFSET IS $FF
0400: 1A35 8D FE 1A            STA   CNTG   START TIMER AGAIN
0410: 1A38 68                  PLA          RESTORE ACCU
0420: 1A39 40                  RTI
0430:
0440:                   END OF REPEAT
```

```
SYMBOL TABLE
CNTA    1AF4    CNTD    1AF7    CNTG    1AFE    DEL     1A00
EEND    0078    EQUALA  0070    EQUALB  0074    FETCH   001E
IRQH    1A7F    IRQL    1A7E    IRQRE   1A30    KEY     00DA
LENGTH  00DE    NOTEH   00DD    NOTEL   00DC    PBDD    1A83
PBD     1A82    POLL    0059    RDFLAG  1AD5    REPEAT  0000
RESET   1C1D    TONE    0030    TONEA   0038    TONEB   0046
TONEC   0052    TONED   0054
```

```
SYMBOL TABLE
REPEAT  0000    FETCH   001E    TONE    0030    TONEA   0038
TONEB   0046    TONEC   0052    TONED   0054    POLL    0059
EQUALA  0070    EQUALB  0074    EEND    0078    KEY     00DA
NOTEL   00DC    NOTEH   00DD    LENGTH  00DE    DEL     1A00
IRQRE   1A30    IRQL    1A7E    IRQH    1A7F    PBD     1A82
PBDD    1A83    RDFLAG  1AD5    CNTA    1AF4    CNTD    1AF7
CNTG    1AFE    RESET   1C1D
```

Molte persone collegano al termine "Computer" il concetto: meno lavoro - più tempo libero. Da un punto di vista più attuale e più realistico la formula dovrebbe trasformarsi in: più lavoro a parità di tempo. Ma il crescente numero di appassionati di computer a microprocessore porta ad ancora un'altra conclusione: più lavoro nel tempo libero!
Prima di poter premere il primo tasto si deve tuttavia affrontare il difficile problema della scelta, perché l'offerta di computer belli e pronti o da autocostruire, di libri, di pubblicazioni è imponente. Si dice vi siano state persone che non hanno più scorto il bosco di fronte a tanti alberi: cioé, non hanno neppure principiato, oppure si sono accorti a casa propria che il nuovo Home-Computer era un bidone.
ELEKTOR intende perciò dare una mano a tutti i principianti in questo campo:

- Con lo Junior-Computer viene offerto un microcomputer a singola piastra, per l'autocostruzione a prezzo conveniente, e come sistema di autoapprendimento.

Dopo che il 1° volume ha spianato la via per Computerlandia, descrivendo come si maneggia lo Junior-Computer, il 2° volume costituisce l'introduzione completa alla programmazione: si parla di "Monitor", "Editor", "assemblaggio" e "disassemblaggio". E tutto viene descritto esaurientemente mediante numerose tabelle, diagrammi di flusso e figure. Nel 3° volume vedremo come lo Junior-Computer si svilupperà in un maturo Personal-Computer. Saranno necessari un'interfaccia per registratori a cassette e TTY, una memoria di 5 KByte, ecc.
Il nostro sistema si presta quindi ad essere espanso per soddisfare requisiti specifici o generali d'impiego.
E dunque: Buon lavoro!

L. 14.500 **Cod. 3002**