

Semplice introduzione all'affascinante tecnica dei computer

Volume 1

JUNIOR COMPUTER

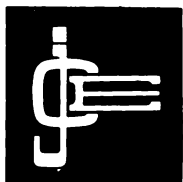


elektor

Junior Computer 1

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 1981
Stampato da S.p.A. Alberto Matarelli - Milano

Un libro per lo Junior Computer?!

Vi siete quindi decisi a compiere i primi passi con lo "Junior Computer" nel mondo dei computer - oppure forse per ora solo a leggere questa prima pagina. Vogliamo perciò esporVi in breve cosa vi attende e che cosa sta "dietro" il concetto dello "Junior Computer".

Bene, cominciamo!

Lo Junior Computer è un micro-computer già piuttosto cresciutello, poiché con il suo moderno microprocessore 6502 esso offre sin dal principio tutte le possibilità di una comoda programmazione e della successiva espansione ad un sistema più grande "con tutte le curve".

Lo Junior Computer è un micro-computer per l'autocostruzione, un micro computer completo su un unico circuito stampato. Ciò costituisce non soltanto un risparmio di spazio e di molti lunghi ed ingombranti collegamenti, ma anche di danaro. Infatti, il sistema base, con questo libro, sono tutto ciò che occorre per l'apprendimento.

Il libro consta di quattro capitoli.

Il primo capitolo contiene una descrizione particolareggiata del montaggio, ma inizia con una introduzione allo Junior Computer e la descrizione dei singoli blocchi funzionali.

Il secondo capitolo si occupa già dei principi della programmazione - la matematica del computer, come maneggiare gli "uni" e gli "zeri".

Nel terzo capitolo si va già sul sodo. Appena terminato di costruire lo Junior, lo si porta in vita.

"Indirizzi", "dati", "codici OP", "operando", "diagramma di flusso" - dapprima termini arcani e misteriosi vengono svelati nel corso di questo capitolo.

Nel capitolo 4, infine, si riportano alcuni programmi per la sperimentazione pratica.

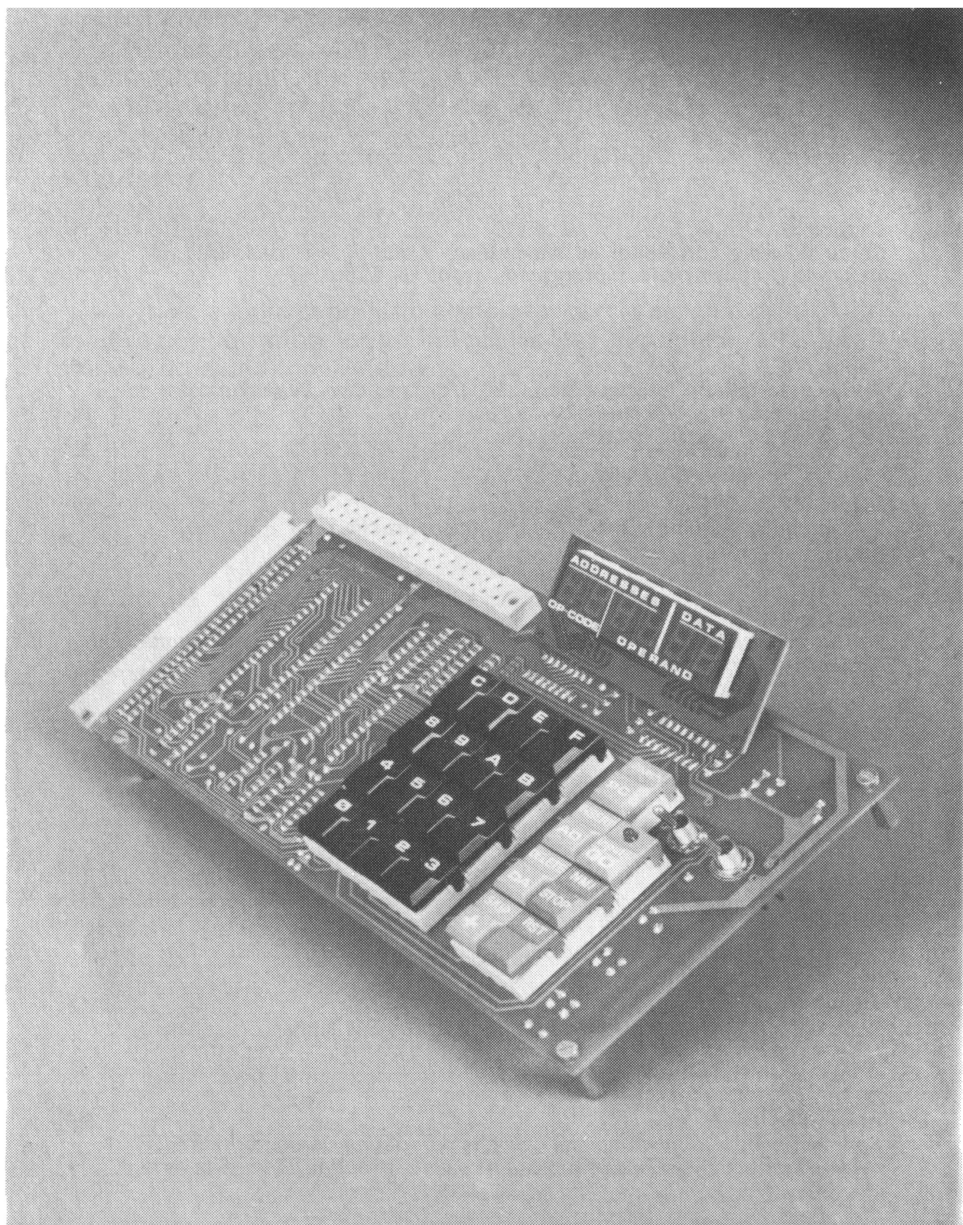
Duro è l'inizio, come ben si sa, e pertanto il sistema di insegnamento dello Junior Computer vuole già sin d'ora comunicare un pò della semplicità con la quale ci si può dedicare al proprio Junior non appena se ne è cominciato a capire qualcosa. In un secondo volume il programmatore dilettante imparerà poi a conoscere nuove possibilità di applicazioni del suo "vecchio" Junior Computer.

Speriamo di aver destato la curiosità di tutti quelli che hanno un certo interesse a questi argomenti: sta ora a loro riuscire anche a soddisfarla!

Gli Autori

Indice

Capitolo 1	7
Prima presa di contatto - <i>Della CPU, PIA, RAM, ROM ed altri elementi strutturali.</i>	
Capitolo 2	43
Pensare e calcolare digitale - <i>Come maneggiare correttamente gli "uni" e gli "zeri".</i>	
Capitolo 3	61
Programmare - <i>Il "galateo" dello Junior.</i>	
Capitolo 4	145
Suggerimenti e programmi dimostrativi - <i>Per programmare e sperimentare occorre studiare.</i>	
Appendice 1	167
Codice OP in ordinamento esadecimale.	
Appendice 2	169
Sommario delle istruzioni	
Appendice 3	175
Lo "Hex Dump" del programma Monitor	
Appendice 4	177
I collegamenti del connettore	



Prima presa di contatto

Questo libro è il primo di una serie di volumi che contengono la descrizione di un micro-computer per l'autocostruzione. Il sistema, sviluppato da Elektor sulla base del ben noto micro-processore 6502, ha ricevuto, a buona ragione, il nome "Junior Computer": gli Autori tenevano infatti a liberare gli amanti della elettronica, non ancora molto in confidenza con la tecnica del computer, dal "timor panico" che spesso impedisce una partecipazione attiva in questo campo affascinante. Il cammino che qui viene descritto è conseguentemente orientato alla praticità dell'autocostruzione. Esso non termina in un vicolo cieco, ma conduce il lettore ad una comprensione completa del montaggio e dell'applicazione di un micro-computer costruito con le proprie mani.

Facciamo conoscenza con un computer

In linea di massima un computer è una macchina semplice. E tuttavia, se si getta per la prima volta uno sguardo nell'interno di un computer, se ne riceve una impressione a dir poco "scioccante". I numerosi elementi strutturali, i moduli, i fasci di cavetti suscitano facilmente dubbi sulle proprie capacità di costruire da soli un computer e poi di lavorarci assieme. In questo libro, tuttavia, mostreremo che non è difficile mettere assieme da soli un micro-computer e che la sua programmazione ed il suo uso offrono un vero divertimento.

Un computer è destinato ad eseguire dei comandi che un programmatore ha introdotto precedentemente nel computer stesso. Prima che egli possa tuttavia introdurre tali comandi, questi devono essere trascritti in forma di un programma.

Il computer richiede al proprio utente non di progettare, ad esempio, un circuito, ma di scrivere un programma eseguibile dal computer. In questo caso, ad esempio, non è importante conoscere la "vita intima" di un micro-processore, ma è necessario per l'utente

sapere come il computer reagisce ai diversi comandi. Facciamo un paragone con un'automobile: per guidare un'auto non si deve conoscere come è costruito il motore, il cambio o la pompa di circolazione dell'olio.

Per tali motivi possiamo considerare lo Junior-Computer come una "scatola nera" (Black Box) di cui non ci interessa il contenuto quando la impieghiamo. Grande interesse rivestono invece gli elementi di collegamento del computer con il "mondo esterno", ai quali ad esempio appartengono il "display" e la tastiera. Mediante questi dispositivi possiamo servirci comodamente del computer senza conoscere l'interno dei singoli C1.

Lo Junior-Computer come "scatola nera"

Prima di avere di fronte a noi lo Junior-Computer come una "black box" dobbiamo naturalmente averlo costruito. Questo capitolo è dedicato quindi al suo montaggio. Qualche lettore si chiederà se la costruzione di un computer non vada al di là delle proprie forze. Vogliamo dire a questi lettori che lo Junior-Computer non presenta davvero grosse difficoltà. La Hardware, il suo "organismo", consta in sostanza di solo 11 C1. Forniremo ai nostri lettori una descrizione del montaggio dettagliata nei minimi particolari, affinché questa scarsa dozzina di circuiti integrati al termine svolga regolarmente i suoi compiti.

Un pò di teoria dei computer

Sin dall'infanzia siamo stati soliti giocare con i blocchi delle scatole di costruzione. Allora perché non ricondurre anche il computer ad uno schema a blocchi? La figura 1 illustra un tale schema a

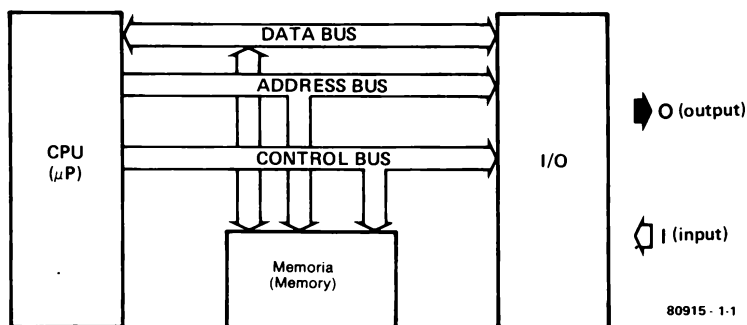


Figura 1. Non potrebbe essere più semplice: lo schema elementare a blocchi di un computer consta di 3 blocchi e di 3 BUS. I BUS sono "fasci di fili" che collegano fra loro gli elementi strutturali del computer, CPU, memoria ed I/O.

blocchi con 3 tipi di connessioni tra i singoli blocchi. Un computer lavora in base ad informazioni che sono rappresentate da segnali elettrici. Queste informazioni vengono anche chiamate dati. Lo scambio di informazioni o di dati tra i singoli blocchi avviene attraverso il bus dati.

Un computer senza "mondo esterno" è una macchina inutile. Per mondo esterno intendiamo, ad esempio, una tastiera (KEYBOARD), uno schermo visivo, un display a 7 segmenti, una stampante o una interfaccia per cassetta. Il blocco I/O (Input/Output) funge da stazione intermedia tra il micro-computer e il mondo esterno. Tramite questo blocco viene governato l'intero traffico di dati dell'interno verso l'esterno e viceversa.

I dati presenti sul bus dati vengono elaborati dalla CPU. La CPU o Central Processing Unit è il cervello del micro-computer. Tramite la CPU vengono svolti tutti i compiti entro il micro-computer. Appena un determinato compito è stato eseguito, il micro-processore verifica nella memoria quale sia il compito successivo. I compiti o istruzioni che il micro-processore esegue sono depositati sotto forma di codici digitali nella memoria del computer. Tutte le istruzioni riunite, che il computer esegue in una determinata applicazione, formano il programma. Nella memoria del computer sono quindi depositati i dati che il programmatore ha precedentemente inserito. Un computer di per sé non è intelligente. Ciò vuol dire che esso non può sviluppare propri pensieri; tuttavia esso può elaborare con incredibile velocità pensieri "premasticati" sotto forma di un programma. Per pensieri premasticati intendiamo questo: il computer esegue una dopo l'altra le istruzioni che precedentemente sono state introdotte in esso. Nella memoria del computer sono presenti solo dati introdotti in precedenza, ma anche altri dati che consentono il dialogo con il computer.

I dati sono la base per poter lavorare con un computer. Questi dati sono sempre di tipo digitale, ma possono assumere aspetti diversi. In primo luogo questi dati si possono presentare a venir registrati nella memoria sotto forma di un programma; in secondo luogo questi dati possono essere costituiti da impulsi per il comando di una stampante o di un qualche altro dispositivo collegato al computer. Quando in seguito parleremo di dati, si tratterà comunque sempre di segnali elettrici.

Tra la CPU e i due blocchi indicati nella figura 1 scorre un vivace traffico di dati. Per guidare questo traffico di dati tra i blocchi, ci occorre l'indirizzamento, che è di competenza del bus-indirizzi. Ciò significa che la CPU comunica tramite il bus-indirizzi con quei blocchi con i quali vuole scambiare dei dati. I dati possono provenire dalla CPU oppure scorrere da uno dei blocchi verso il CPU. Infine resta da considerare ancora il bus-comandi o bus di controllo. Questo bus è necessario per il governo interno del sistema del computer. Esso contiene fra gli altri un segnale di comando che stabilisce la direzione del trasporto dei dati sul bus-dati. I se-

gnali di comando sul bus di controllo vanno in parte fuori della CPU oppure affluiscono dall'esterno verso il micro-processore. Affinché la CPU possa eseguire comandi o istruzioni nella corretta successione, essa fornisce due segnali di clock al bus di controllo. Questi due segnali di clock alla frequenza di 1 MHz costituiscono il "battito cardiaco" del micro-computer.

Un pò più di tecnica

Dallo schema generale a blocchi della figura 1 passiamo allo schema a blocchi dello Junior-Computer della figura 2. Consideriamo prima i tre bus, le vere e proprie arterie dello Junior-Computer! Il bus-indirizzi consiste di 16 collegamenti elettrici. Ogni collegamento elettrico può assumere, indipendentemente dagli altri, due stati elettrici: sul collegamento c'è tensione oppure no.

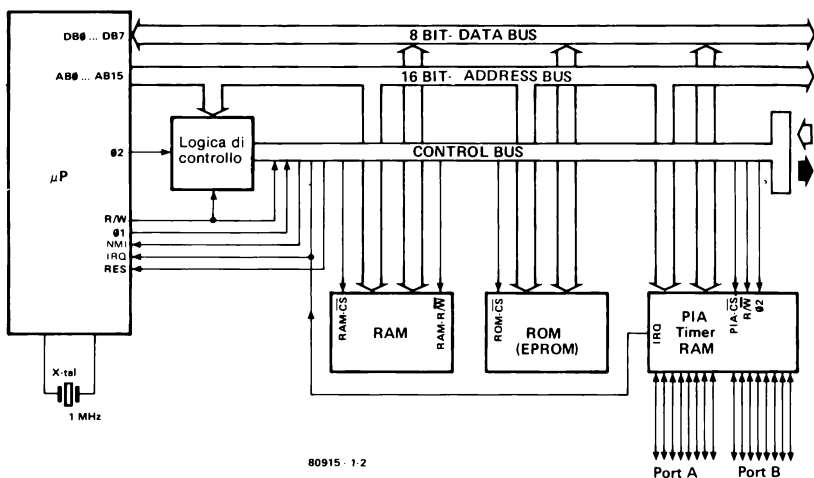


Figura 2. Schema dettagliato a blocchi dello Junior-Computer. Rispetto alla figura 1 la memoria è suddivisa in due blocchi: la RAM e la ROM. La RAM è la memoria di scrittura e lettura e la ROM la memoria di sola lettura dello Junior-Computer.

Con 16 collegamenti sono quindi possibili 2^{16} ovvero 65.536 diversi stati. Si tratta quindi di più di 65.000 diverse locazioni che la CPU può indirizzare nella memoria. Ad ogni collegamento indirizzato corrisponde un bit, perciò si parla di un bus-indirizzi a 16 bit. Il bit è l'unità più piccola di informazione digitale. Cosa intendiamo con ciò lo chiariremo meglio nel secondo capitolo.

Sul bus-dati, che consta di 8 collegamenti, corre l'intero traffico di dati. Diversamente che sul bus-indirizzi, il trasporto dati sul bus-

dati può avvenire in due direzioni. Perciò si usa anche dire che il bus-dati è un bus bidirezionale. Naturalmente il movimento dei dati non scorre contemporaneamente in entrambe le direzioni, ma solo fuori della CPU o verso il micro-processore.

La direzione di flusso di dati sul bus-dati è stabilita dal segnale di lettura/scrittura (Read/Write = segnale R/W); questo segnale si trova sul bus di controllo o bus di comando. Il segnale R/W si può paragonare a un segnale del traffico stradale, il segnale "strada a senso unico". Se la CPU legge dei dati dalla memoria o dall'I/O, la punta della freccia "strada a senso unico" è rivolta verso la CPU. Il segnale R/W assume allora lo stato logico 1. Nell'altro caso, quando la CPU scrive dati nella memoria o in I/O, questo segnale indica con la sua freccia che ci si muove dalla CPU nella direzione della memoria o dell'I/O. Il segnale R/W assume in questo caso lo stato logico 0. Per consentire il trasporto di dati in due direzioni sul bus-dati, sono necessari circuiti digitali con cosiddette uscite a tri-stato. Gli elementi circuitali tri-stato si caratterizzano per il fattore che le loro uscite possono assumere non due ma tre stati diversi:

- nessuna tensione = stato logico 0
- presenza di tensione = stato logico 1
- stato ad alta resistenza ohmica.

Proprio grazie allo stato ad alta resistenza ohmica che le uscite di questi componenti possono assumere è possibile svolgere su un bus-dati un traffico di dati in due direzioni.

Gli elementi di memoria, le "piccole cellule grigie" del computer

Un computer possiede due tipi di elementi di memoria: una memoria nella quale esso può solo leggere i dati e una memoria nella quale esso può registrare, come pure tornare a leggere i dati. Entrambi i tipi di memoria sono indispensabili per lavorare con un computer. Le due memorie sono illustrate nella figura 2 dai blocchi ROM e RAM. Rivediamo un momento la figura 1. Qui le ROM e RAM sono riunite in un unico blocco, la memoria. Perché quindi nel nostro computer sono necessarie due diverse memorie? Per poter lavorare con un computer è necessario disporre di un programma che deve essere sempre presente, il programma Monitor o programma residente. Questo programma Monitor è sempre registrato in una ROM. Inoltre il computer lavora con dati che sono variabili. Tra questi lo stesso programma, che il programmatore inserisce nel computer (con l'ausilio del programma Monitor). In altri casi è spesso necessario correggere il programma prima che esso lavori correttamente. Anche in questo caso occorre correggere dati presenti nella memoria. Dati che in frazioni di secondo possono essere modificati entrano nel computer attraverso la PIA oppure la I/O (fig. 2) o inversamente.

Dati permanenti, ossia dati che durante l'esecuzione di un pro-

gramma non mutano, vengono solo letti da una memoria. Dati non permanenti, ossia dati che si modificano continuamente, si trovano a essere registrati nella memoria oppure a essere prelevati dalla memoria. Si parla usualmente in questi casi di "scrittura" nella memoria o di "lettura" dalla memoria.

In un computer vi sono elementi di memoria dai quali è possibile esclusivamente leggere i dati. Un tal tipo di elemento di memoria si chiama ROM (abbreviazione di Read Only Memory). La memoria nella quale è registrato il programma Monitor dello Junior-Computer ha il carattere di una ROM, poiché da questa memoria si possono soltanto leggere i dati.

L'altro elemento di memoria presente è la RAM (abbreviazione di Random Access Memory). Da questa memoria si possono leggere dati, ma in una RAM è possibile anche scrivere i dati.

La memoria di lavoro dello Junior-Computer ha quindi il carattere di una RAM. Il traffico di dati dalla RAM o verso la RAM avviene tramite il bus-dati in due direzioni. Il segnale R/W sul bus di controllo determina se la CPU desidera leggere dati dalla RAM o scrivere dati nella RAM. A questo punto i due concetti di lettura e scrittura dovrebbero essere chiari. Lettura significa che la CPU legge o preleva dati dalla memoria. Questi dati, che erano registrati in determinate locazioni di memoria, vengono copiati nella CPU. Dopo la lettura dalle corrispondenti celle di memoria, i dati presenti in queste celle risultano inalterati e non sono perduti. Ciò può essere paragonato alla lettura di un libro: dopo la lettura le lettere sulle pagine non sono certo sparite! Nella scrittura di dati in una determinata cella di memoria della RAM invece i dati precedenti vengono "sovrascritti" dai nuovi (e quindi sono perduti).

Le RAM perdono il loro contenuto quando cade la loro tensione di alimentazione. Se i dati di una RAM debbono essere conservati anche dopo tolta la tensione di alimentazione, essi possono essere memorizzati su una cassetta da registratore a nastro o su un Floppy-Disk. Da questi dispositivi periferici di memoria i dati, quando necessario, possono poi venir riportati entro il computer. Quindi tratteremo l'espansione dello Junior-Computer, ritorneremo in maggiore dettaglio su questi due dispositivi di memoria esterni.

A proposito:

I dati che si trovano in una ROM vengono introdotti in questa memoria durante il suo processo di fabbricazione. Si dice quindi che una ROM è programmata a maschera. Ciò significa che l'informazione o il programma viene introdotto nella ROM mediante una maschera in fase di produzione. Una maschera di questo tipo ha una certa somiglianza con il master di un circuito stampato. Il programma Monitor dello Junior-Computer, tuttavia, non è registrato in una ROM programmata a maschera, ma in una EPROM

(EPROM è l'abbreviazione di "Erasable, user Programmable ROM"). Questo elemento di memoria può essere programmato indipendentemente con un programmatore EPROM, una "macchina per l'impressione dei dati". Le EPROM hanno la comoda proprietà che in esse i dati si possono cancellare usando luce ultravioletta. Dopo la cancellatura questi elementi di memoria sono nuovamente riprogrammabili.

Altri tipi di elementi di memoria che spesso si ritrovano nei computer si chiamano PROM e EAROM. Anche la PROM può essere programmata, ma le informazioni dopo la programmazione non sono, come nel caso della ROM, cancellabili.

I dati nella PROM sono determinati dai "fusible links", ossia giunzioni sul chip di silicio che vengono bruciate mediante impulsi di corrente. In generale una giunzione "bruciata" ha lo stato logico 1 e una giunzione non "bruciata" lo stato logico zero. In tal modo è possibile imprimere in una PROM dati arbitrari.

La EAROM come la EPROM può essere cancellata dopo la programmazione. In questo caso tuttavia ciò non avviene con l'uso di luce ultravioletta, ma con impulsi di corrente. Dopo la cancellazione anche la EAROM può essere riprogrammata.

Il PIA, il blocco I/O dello Junior-Computer

Come già sappiamo, con l'unità I/O viene stabilito il contatto tra il computer e il mondo esterno. Nella figura 2 il blocco PIA corrisponde all'I/O. PIA è l'abbreviazione di "Peripheral Interface Adapter". Dalla PIA partono verso l'esterno due bus: la porta A e la porta B; esse sono collegate ad un proprio connettore.

Oltre alla porta A ed alla porta B conducono verso l'esterno tre ulteriori bus: il bus-indirizzi, il bus-dati e il bus di controllo. Questi bus sono previsti per l'espansione dello Junior-Computer. L'espansione significa tuttavia un ingrandimento del "mondo interno" del computer e per ora non ci interessa. Nel nostro caso il mondo esterno dello Junior-Computer è costituito dalla tastiera esadecimale e dal display a 7 segmenti e 6 cifre. Questo mondo esterno viene comandato dalla CPU tramite la PIA. Le due porte che escono dalla PIA sono un "prolungamento" del bus-dati. Queste porte, come il bus-dati, hanno una capacità di 8 bit. La selezione delle porte, che determina a quale porta il bus-dati viene collegato, avviene tramite il bus-indirizzi. Ogni porta può venir programmata indipendentemente dall'altra come ingresso oppure uscita. Se una porta viene programmata quale ingresso, la CPU vi può leggere tramite il bus-dati. Se ad esempio, su una porta programmata come ingresso è presente un segnale digitale seriale, la CPU può trasformare questa informazione seriale in formazione parallela. A tal scopo ovviamente deve essere previsto un apposito programma che metta in grado la CPU di effettuare questa trasformazione. Nel volume 2, che per ora non è ancora all'orizzonte,

impareremo a scrivere questo tipo di programmi.

Così come le porte, indipendenti, una dall'altra, possono essere programmate quali ingressi, esse possono anche essere, indipendentemente l'una dall'altra, programmate quali uscite. Così possiamo descrivere nel "mondo esterno" i dati tramite il bus-dati. Affinché la lettura o la scrittura dalle o entro le porte possono avvenire secondo il ritmo di lavoro della CPU, il bus di controllo indirizza alla PIA alcuni segnali di comando.

La CPU, il centro operativo del computer

La CPU governa il completo sistema del computer tramite il bus-indirizzi, il bus-dati bidirezionale e il bus di controllo e un generatore interno di clock col suo cristallo di quarzo esterno da 1 MHz. Il generatore di clock fornisce due segnali $\Phi 1$ e $\Phi 2$ che sono particolarmente importanti per il funzionamento del computer. $\Phi 1$ costituisce il clock-indirizzi e $\Phi 2$ il clock-dati. Prima che possa avvenire un trasporto di dati, deve venir indirizzata una cella di memoria nel sistema del computer. Solo dopo questo indirizzamento si possono trasferire i dati. Il clock $\Phi 1$ è responsabile della stabilità dell'informazione sul bus-indirizzi, mentre il clock $\Phi 2$ della stabilità dei dati sul bus-dati. Poiché $\Phi 1$ e $\Phi 2$ sono strettamente collegati, si parla di un clock a due fasi. Un registro interno importante della CPU è il Program Counter "PC" (non indicato nella figura 2!). Questo contatore di programma guida il micro-processore tramite il bus-indirizzi lungo il programma. Ciò vuol dire che, grazie al contatore di programma, le istruzioni o i comandi vengono eseguiti in modo sequenziale. Gli indirizzi, che sono la base dei dati nel micro-computer, vengono derivati dal programma in modo diretto oppure indiretto. È proprio grazie a questi indirizzi che il micro-processore si trova a suo agio nel programma.

Scusi, disturbo?

Dobbiamo ancora considerare tre segnali indicati nella figura 2. Sono i segnali RES, IRQ e NMI. Del segnale RES ci occuperemo qui solo poco. Con questo segnale si fa partire lo Junior-Computer. Appena inserita la tensione di rete, la CPU non sa in quale posizione del programma deve iniziare l'elaborazione. Con il segnale RES le viene comunicato che essa si deve prima di tutto interessare del programma Monitor dello Junior-Computer. I segnali IRQ e NMI servono a "disturbare" in modo costruttivo la CPU durante l'esecuzione di un programma.

IRQ è l'abbreviazione di "Interrupt ReQuest" e NMI l'abbreviazione di "Non Maskable Interrupt". Per "disturbo" intendiamo una interruzione del programma in corso, per la quale il computer deve interessarsi di altri argomenti importanti. L'iniziativa di un tale disturbo viene dall'esterno. Con un tale Interrupt si comunica al

computer che esso si deve interessare rapidamente di un tasto che è stato premuto, o che il programmatore deve fargli una importante comunicazione, oppure che esso deve velocemente servire un dispositivo periferico, ad esempio una stampante. Dopo che la CPU ha recepito un tale "impulso di disturbo" ad uno degli ingressi Interrupt, esso si occupa immediatamente del "disturbatore". Dopo aver servito questo disturbatore e averlo così soddisfatto, il micro-processore torna ad occuparsi del programma. Per lo più sono parecchi questi guastafeste che "bussano" contemporaneamente agli ingressi di Interrupt della CPU. In questo caso ogni disturbatore si vede assegnare una determinata priorità. Tanto maggiore è questa priorità, tanto prima verrà servito. Una differenza importante tra IRQ e NMI è che la CPU può ignorare un segnale IRQ quando il programmatore glielo consente. Per un segnale NMI invece esso deve immediatamente occuparsi, che lo voglia o no, del disturbatore.

Così abbiamo trattato anche la figura 2. Non del tutto poiché nella PIA è contenuta ancora un'altra interessante funzione: un timer. Questo timer può essere comandato dal programma; possiede la interessante caratteristica che esso marcia indipendentemente dalla CPU. Cosa vuol dire? Spesso, durante lo scambio di dati tra il computer e i dispositivi collegati, devono venire introdotti dei ritardi di tempo. Pensiamo ad esempio solo al tempo necessario per trasferire un bit. Tali ritardi di tempo si possono facilmente realizzare tramite il timer, mentre la CPU indipendentemente dal timer continua ad occuparsi del programma. Il blocco "Control Logic" è un cosiddetto codificatore di indirizzi. La Control Logic genera segnali Chip-Select. Questi segnali stabiliscono se la CPU vuole comunicare con la RAM, la ROM o la PIA. Tanto basti per la vita intima dello Junior-Computer. Ora è tempo di conoscere lo Junior-Computer dall'esterno.

La tastiera e il display, le unità ingresso/uscita dello Junior-Computer

La tastiera ed il display sono le unità ingresso/uscita dello Junior-Computer. Entrambe sono necessarie per prendere contatto con il computer. La figura 3 indica come queste unità sono collegate allo Junior-Computer. Come si vede facilmente, sia la tastiera che il display sono collegate alle porte A e B. Una tastiera esadecimale costituisce il mezzo più semplice per alimentare un computer. Il display esadecimale è necessario per mostrare al programmatore ciò che egli ha introdotto nel computer tramite i tasti. Questi due organi esadecimali di ingresso e di uscita sono stati scelti per rendere il nostro computer il più possibile pregevole. Certo, avremmo potuto collegare senza difficoltà allo Junior-Computer una stampante, un terminale video o un altro dispositivo periferico, ma allora il progetto sarebbe risultato molto più caro, ed abbiamo cercato

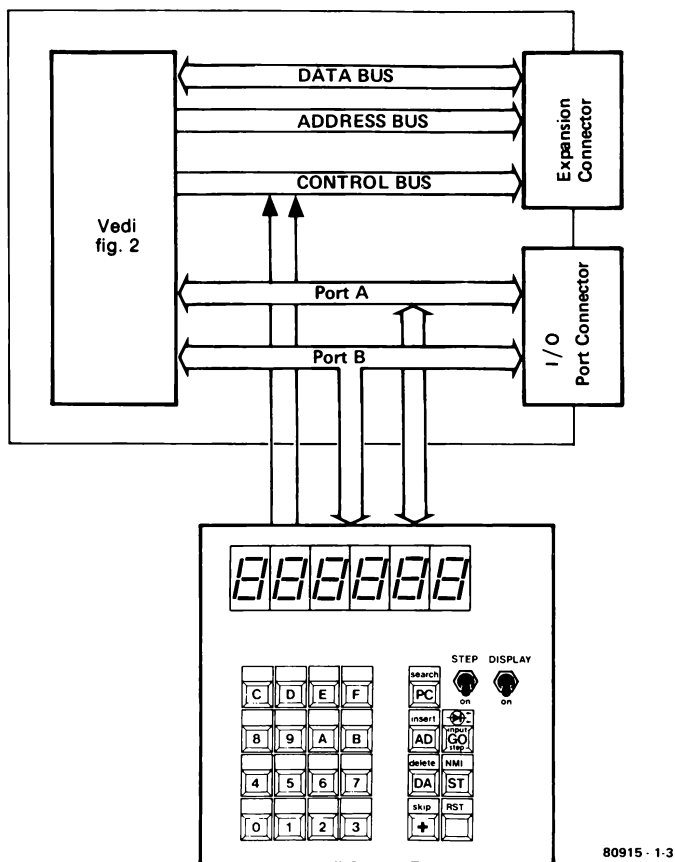


Figura 3. Collegamento del display e della tastiera alle linee I/O, le porte A e B. La porta A funziona alternativamente da ingresso e uscita, mentre la porta B solo da uscita. I tasti ST ed RST sono collegati direttamente al bus di controllo e non vengono interrogati dal programma Monitor.

di evitarlo almeno inizialmente. Nei libri successivi di questa serie vi mostreremo in modo particolareggiato come lo Junior-Computer possa venire espanso in modo semplice e possa svilupparsi sino al "Senior-Computer". Oltre al connettore I/O è presente anche un connettore per l'espansione (expansion connector) per cui non vi è alcuna difficoltà ad una futura espansione del sistema. Per essere esaurienti, precisiamo che lo expansion connector risulta a 64 poli ed è compatibile con un bus SC/MP. Le 16 piste delle porte A e B sono collegate ad un connettore a 31 poli per il collegamento esterno.

Come abbiamo già detto, la tastiera ed il display sono collegati alle porte A e B dell'Interface Adapter PIA periferico. Per la porta A è ammesso un trasferimento di dati bidirezionale, per la porta B invece un traffico unidirezionale. Ciò vuol dire che la porta A può lavorare una volta come ingresso e successivamente come uscita, mentre la porta B funge solo da uscita. Il motivo di ciò è il seguente: la porta B con l'ausilio del programma Monitor fornisce l'impulso di scansione per la tastiera e l'impulso strobe per il display multiplex. Per ciò è sufficiente che esso sia soltanto programmato come uscita. Nel caso della porta A le cose stanno diversamente: innanzitutto tramite la porta A viene letto un tasto eventualmente premuto ed introdotto il relativo segnale nel computer. Inoltre, tramite la stessa porta, viene servito il display a 7 segmenti. La porta A è programmata come ingresso per la lettura di un tasto attivato; e come uscita per il comando dei segmenti del display. Per adesso non è possibile comprendere pienamente tutti questi dettagli, ma sarà già chiaro quanto l'unità PIA sia impiegabile in modo universale.

Dalla tastiera partono ancora due collegamenti al bus di controllo. Questi collegamenti provengono dal tasto RST e dal tasto ST. Questi due tasti sono tasti di Hardware, ossia essi non vengono periodicamente interrogati dalla CPU come gli altri tasti della tastiera. Lavorando con lo Junior-Computer impareremo a conoscere meglio sia la tastiera che il display, ma questo lo vedremo nei capitoli successivi. Per intanto è importante l'esatta descrizione della parte elettronica dello Junior-Computer, perché prima di programmarlo dobbiamo naturalmente costruirlo. Passiamo quindi a descrivere "la carne e il sangue" dello Junior-Computer Computer: l'Hardware.

Il suo circuito

La figura 4 mostra come è composto lo Junior-Computer. Descriviamo ora il suo circuito. Cominciamo da IC1, il micro-processore. Esperte "volpi" dei calcolatori saranno pronte a giurare che questo tipo, il 6502, non è il miglior processore possibile; eppure questa è una CPU da leccarsi le dita. Perché? Lasciamo parlare il micro-processore da se stesso:

- poche istruzioni ma molto efficaci e multivalenti (il che rende il computer facilmente programmabile).;
- la 6502 è la CPU con il maggior numero di tipi di indirizzi. Questa CPU prevede già nella prima generazione i tipi di indirizzi che negli altri micro-processori sono stati introdotti solo nella terza generazione;
- istruzioni mnemoniche in forma stenografica semplice e di facile apprendimento;
- è la CPU più diffusa tra gli amatori interessati ai computer.

Potremmo ancora proseguire questa lista, ma risulta ormai chiaro

perché abbiamo scelto per lo Junior Computer proprio questo e nessun altro micro-processore.

Solo pochi componenti esterni sono necessari attorno al micro-processore, perché il suo cuore, il generatore di clock, possa battere, N1, R1, D1, C1 ed il quarzo da 1 MHz regolano il passo del generatore di clock interno. Come già detto, il cuore della CPU 6502 batte con una frequenza di 1 MHz. Dallo schema è facilmente riconoscibile come i due segnali di clock $\Phi 1$, e $\Phi 2$, che sono responsabili della preparazione dei bus-indirizzi e dei bus-dati, vengono inviati sul bus di controllo.

Dal micro-processore (IC1) parte anche il bus-indirizzi. Esso comprende le linee A 0 ad A 15. IC1 costituisce pure la stazione di trasmissione e ricezione per il bus-dati con le linee da D 0 a D 7 (ricordiamo che su questo bus avviene uno scambio di dati in due direzioni). Su entrambi questi bus le informazioni sono presenti sotto forma di segnali elettrici.

In che modo? Molto semplice! le linee indirizzi e quelle dati sono ordinate in una ben determinata successione.

Questa successione è da A 15 ad A 0 per il bus-indirizzi e da D 7 a D 0 per il bus dati. Su ognuno di questi collegamenti vi può essere tensione oppure no. Se una delle linee indirizzi o dati è sotto tensione, essa è allo stato logico 1, se non vi è tensione è allo stato logico 0. Scriviamo ora gli zeri e gli uno sul bus indirizzi (A 15...A 0) in ordine di successione da sinistra a destra su un foglio di carta: otteniamo un numero binario consistente di 16 elementi. Lo stesso vale per il bus-dati: ma in questo caso sono possibili solo combinazioni di 8 elementi. I numeri o più esattamente le disposizioni di zeri ed uno su questi due bus costituiscono un codice. Un tal codice sul bus-indirizzi corrisponde ad un indirizzo e un codice sul bus-dati corrisponde a dati che devono essere elaborati nel computer. Usualmente le disposizioni di zeri e di uno sul bus-indirizzi e sul bus-dati vengono definiti "parole". Poiché il bus indirizzi ha 16 linee, su di esso si possono rappresentare parole della capacità di 16 bit. Sul bus-dati si possono invece rappresentare solo parole della capacità di 8 bit, poiché su esso sono presenti solo 8 linee. Dato che un computer è una macchina binaria digitale, esso "capisce" in sostanza solo due numeri: 0 e 1 o, in termini tecnici: presenza di tensione, assenza di tensione.

L'organizzazione di memoria dello Junior-Computer

Consideriamo ancora lo schema circuitale dello Junior-Computer nella fig. 4.

Si vede che tramite i bus-indirizzi e i bus-dati gli elementi di memoria dello Junior-Computer sono collegati col micro-processore IC1. Una di tali memorie è la EPROM (IC2) nella quale è registrato il programma Monitor. Dalla EPROM si possono solo leggere dati. La memoria di scrittura e di lettura è costituita dalle due

RAM IC4 ed IC5. Come dice già il loro nome, da queste memorie si possono sia leggere che scrivere dati. I dati che vengono trasportati sul bus-dati sono costituiti da 8 bit. Perciò anche le posizioni di memoria nella EPROM e nella RAM devono avere la capacità di 8 bit. Nel seguito una parola lunga 8 bit sarà definita come "byte". La EPROM contiene 1024 posizioni di memoria ciascuna della capacità di 8 bit, in cui può memorizzare un byte. Arrotondando il numero 1024 si parla di un K, intendendo $K = 1000 = \text{kilo}$. Tutti i bytes presenti in una EPROM devono essere richiamabili in modo univoco: ciò risulta facilmente possibile con 10 collegamenti di indirizzo, A 0...A 9, con i quali sono indirizzabili 2^{10} , ovvero 1024 celle di memoria.

Nelle RAM invece si possono indirizzare solo 1024 mezzi bytes con i collegamenti indirizzo A 0...A 9. Per memorizzare un intero byte nella memoria di scrittura e lettura sono quindi necessari due CI: uno per memorizzare i bit dati D7...D4, e un altro per i bit dati D3...D0. Ciò viene tecnicamente realizzato mediante IC4 e IC5: i due CI vengono indirizzati contemporaneamente tramite il bus-indirizzi.

La EPROM e la RAM ricevono anche segnali dal bus di controllo, cosiddetti segnali di selezione.

Dato che entrambe le memorie sono collegate in parallelo al bus-indirizzi (A0...A9) occorre realizzare una separazione fra di loro per evitare un doppio indirizzamento. Questa separazione avviene tramite i segnali CS, K0 e K7. Entrambi i segnali di separazione provengono dal codificatore di indirizzi IC6. K0 si riferisce alla RAM e K7 alla EPROM. Se su un conduttore CS esiste tensione, ossia lo stato logico 1, allora la corrispondente memoria risulta scollegata dal bus-dati. In altre parole i conduttori dati della memoria assumono uno stato ad alta resistenza ohmica. Se invece un conduttore CS assume lo stato logico 0, la memoria ad esso collegata può partecipare al traffico dei dati. Ciò si può anche esprimere dicendo che la CPU desidererebbe intraprendere uno scambio dati con la relativa memoria.

Si rendono necessari 10 conduttori di indirizzo per indirizzare 1.024 celle di memoria distinte sia nella RAM che nella EPROM. Dato che il bus-indirizzi è formato da 16 conduttori, rimangono ancora liberi 6 conduttori. Con questi 6 collegamenti si possono stabilire $2^6 = 64$ combinazioni. Così è possibile indirizzare 64 blocchi di memoria ciascuno da 1 K byte. Torna così il conto che con 16 linee di indirizzo si possono indirizzare 64 K byte. Come mai? Il K byte d'ordine 0 è anch'esso un blocco di memoria con 1024 locazioni di memoria. Tramite 6 conduttori si possono indirizzare $2^6 = 64$ blocchi da 1 K byte, ossia 64 K byte! In effetti si tratta di 65536 celle di memoria.

Con 64 collegamenti CS dunque si può suddividere il campo indirizzi in 64 blocchi da 1 K byte; ad ogni blocco da 1 K byte corrispondono i collegamenti indirizzo A 0...A 9 ed il relativo segnale

CS. Nel modello standard dello Junior-Computer si fa uso di solo 8 conduttori CS.

Si tratta dei segnali CS, K0...K7. Questi segnali CS sono generati dal codificatore di indirizzi IC6, al quale si collegano i conduttori indirizzo A10...A12. Tre di questi 8 segnali CS sono necessari per il governo interno dello Junior-Computer; gli altri 5 sono portati sul connettore di espansione in previsione dell'allargamento del sistema. Mentre K0 e K7 trovano impiego per la selezione tra la RAM e la EPROM, K6 è collegato alla PIA (IC3). Un sommario dei segnali CS disponibili è dato dalla seguente tabella:

A15...A13	A12	A11	A10	A9...A0	attivo	blocco di memoria
X	0	0	0	X	K0	1 K RAM (IC4, IC5)
X	0	0	1	X	K1	1 K esterno
X	0	1	0	X	K2	1 K esterno
X	0	1	1	X	K3	1 K esterno
X	1	0	0	X	K4	1 K esterno
X	1	0	1	X	K5	1 K esterno
X	1	1	0	X	K6	RAM in PIA, Ports, Timer (IC3)
X	1	1	1	X	K7	1 K EPROM (IC2)

Questa tabella contiene diverse colonne alle quali sono riferiti determinati bit del bus-indirizzi. La prima colonna che contiene i bit indirizzi A13...A15 non è importante per il campo indirizzi dello Junior-Computer poiché i relativi conduttori di indirizzo non sono collegati al codificatore indirizzi IC6. La quinta colonna contiene i bit di indirizzo A0...A9. Questi bit indirizzo servono a selezionare da un campo di memoria di 1 K byte le celle di memoria specificatamente indirizzate. Anche questa colonna, tuttavia, non ha niente a che vedere con l'effettiva codificazione degli indirizzi. Le "X" nella prima e quinta colonna stanno per "don't care" che significa stato-indifferente.

Le colonne A12, A11 e A10 sono invece quelle importanti per la codificazione degli indirizzi dello Junior-Computer. Con questi 3 bit indirizzi si possono separare gli 8 blocchi di memoria che sono collegati in parallelo al bus-indirizzi A0...A9. I segnali di separazione di questi blocchi di memoria si chiamano K0...K7. Non tutti e 8 questi segnali trovano impiego nella codificazione interna degli indirizzi. K0 nel circuito stampato di base dello Junior-Computer è riferito alla RAM (IC4, IC5). Quando questo conduttore ha lo stato logico 0, significa che la CPU intende mettersi in contatto con la memoria RAM. Ciò avviene quando A12...A10 valgono 0. Il conduttore CS K7 è riferito alla EPROM IC2. Ogni volta che la CPU legge dati da questa ROM, K7 ha lo stato logico 0. Ciò avviene quando su A12...A10 vi è tensione, ossia lo schema di bit relativo è 111. Il segnale CS K6 risulta attivo quando la CPU intende scambiare dati con la PIA (IC3). Ciò accade quando sui conduttori indi-

rizzo A12...A10 la configurazione dei bit vale 110. Gli altri segnali CS K1...K5 vengono impiegati solo nella successiva espansione dello Junior-Computer. Nel sistema espanso essi servono al governo di alcune EPROM che contengono i programmi per il comando di una stampante (TTY), dello Elekterminal e per poter lavorare con due registratori a cassetta. Questo sarà comunque descritto in uno dei volumi successivi della serie. Per il comando della RAM (IC4, IC5), si richiede ancora un altro segnale: RAM-R/W. Questo segnale di comando decide tramite il bus-dati se i dati devono essere trasportati alla RAM od in essa venire letti. Quando la RAM-R/W è dello stato logico 1 e la RAM risulta attivata tramite il collegamento CS KO, la CPU legge dati da una determinata cella della RAM. L'inverso avviene quando la RAM-R/W ha lo stato logico 0: in tal caso la CPU scrive dati in una cella della RAM che viene selezionata tramite A0...A9. Più esattamente: la CPU scrive in questa cella RAM "sopra" i vecchi dati, cancellandoli. Questo segnale RAM-R/W è una combinazione del segnale R/W della CPU e del segnale di clock Φ 2. Questo serve ad escludere con sicurezza la scrittura di dati in una cella della RAM sinché i dati non risultano risiedere stabilmente sul bus-dati.

Abbiamo così trattato alcuni dei segnali di comando che possono essere presenti sul bus di controllo dello Junior-Computer. Passiamo ora a conoscere alcuni altri importanti segnali di tale bus. Con il segnale di Reset si portano il micro-processore IC1 e la PIA IC3 in posizione di partenza. Questo avviene quando il conduttore RES si porta per alcuni microsecondi allo stato logico 0. Questo conduttore normalmente si trova allo stato 1, poiché esso è portato tramite la resistenza di pull-up R2 al livello della tensione di alimentazione. Mediante il tasto RST si attiva il Reset. Per eliminare i contatti di rimbalzo di questo tasto è previsto IC8. Questo CI contiene sul proprio chip due timer. Uno di essi serve ad eliminare i rimbalzi del tasto RST, l'altro quelli del tasto ST sul quale torneremo più avanti. Anche il conduttore NMI appartiene al bus di controllo. Questo collegamento, a differenza del collegamento RES, risulta sensibile ai fronti dei segnali ad onda quadra. Ciò vuol dire che un impulso negativo sul piedino NMI del micro-processore genera un Interrupt. Vedremo alla fine del 3° capitolo nei particolari di che cosa si tratta. Come si ricava dal circuito (fig. 4), in condizioni normali il conduttore NMI tramite la resistenza R3 si trova allo stato logico 1. Col tasto ST e con l'uscita della porta nand (nand-gate) N5, si può applicare su questo conduttore un fronte di discesa (wired OR). Col tasto ST si può, con l'ausilio del programma Monitor, interrompere un programma in corso, e mediante la porta N5 si può portare lo Junior-Computer in "step by step mode". Con l'interruttore S24 chiuso (ON) si possono verificare passo passo i programmi. Il computer procede elaborando una istruzione alla volta e poi fermandosi sin quando non gli viene richiesto di eseguire il comando successivo. In tal modo è molto

facile controllare eventuale errori nel programma. Poiché lo "step by step mode" viene gestito dal programma Monitor nella EPROM, si deve escludere l'impiego del programma Monitor finché durano tali condizioni: perciò il segnale CS K7 è anche applicato ad un ingresso della porta N5.

Il singolo passo di programma nello "step by step mode" viene comandato dall'uscita SYNC della CPU. Ogni volta che il micro-processore estrae un comando dalla memoria, questa uscita passa allo stato logico 1. Ne segue un NMI per cui lo Junior-Computer viene costretto ad inserirsi in un ciclo di attesa.

Anche l'ingresso IRQ del micro-processore è collegato al bus di controllo. Questo conduttore viene montato tramite la resistenza R4 allo stato logico 1. L'uscita IRQ della PIA IC3 è collegata al conduttore IRQ. IRQ può essere attivato solo quando questo è consentito espressamente dal programma. Il piedino IRQ della CPU, come il piedino RES, risulta sensibile al livello del segnale e deve quindi per generare un IRQ mantenersi per alcuni micro secondi allo stato logico 0.

Il conduttore $\Phi 2$ del bus di controllo è anche collegato ad IC3, la PIA. Ciò è richiesto perché in questo circuito integrato si trova un timer che può essere influenzato dal programma o, più propriamente, può essere programmato. Questo timer, che marcia indipendentemente dal micro-processore ha come base dei tempi il segnale di clock $\Phi 2$. Anche il conduttore R/W è collegato ad IC3. Ciò perché in questo componente universale è contenuta una RAM di 128 byte, nonché diversi registri. Altri due piedini della CPU non impiegati sono analogamente collegati al bus di controllo: RDY ed SO. Se si vogliono collegare RAM dinamiche allo Junior-Computer è necessario disporre del segnale RDY. Lo SO serve per le successive espansioni della Hardware. Riassumendo, sul bus di controllo sono presenti i seguenti segnali:

- R/W Read/Write - signal
- RAM-R/W Read/Write - signal per la RAM
- K0...K7 segnali chip-select
- $\Phi 1$ e $\Phi 2$ segnali di clock per il bus-indirizzi e il bus-dati
- RES conduttore di Reset
- IRQ e NMI collegamenti per l'Interrupt
- RDY comando di RAM dinamiche
- SO ed EX espansioni della Hardware del computer (per EX vedi IC6)

La PIA, la "linea calda" verso il mondo esterno

Lo schema circuitale (fig. 4) comprende un altro importante elemento del computer, la PIA (IC3). Essa ha due porte: porta A e porta B. Come già detto in precedenza, attraverso queste porte si può commutare il bus-dati dello Junior-Computer. È il programma a determinare con quale porta la CPU desidera collegarsi. La

selezione delle porte avviene tramite le linee indirizzo a più basso indice A0...A3.

Ad ogni porta corrisponde un registro I/O e un registro di direccionamento dei dati. Entrambi questi registri hanno la capacità di 8 bit. In entrambi si possono sia scrivere che leggere dati. Quando la CPU scrive nel registro di direccionamento dei dati una determinata configurazione di bit, risultano corrispondentemente determinati gli ingressi e le uscite della porta relativa: uno zero nel registro di direccionamento dei dati definisce la corrispondente linea come ingresso, un 1 la definisce come uscita. Anche il segnale R/W è collegato con la PIA. Quando questo segnale fornito dal bus di controllo ha lo stato logico 1, la CPU può leggere i dati che si trovano fuori della relativa porta. Per dati dobbiamo come sempre intendere tensioni con conformazione TTL. Se invece la linea R/W ha lo stato logico zero, i dati possono essere scritti dalla CPU in una delle due porte.

La IC3 contiene anche una RAM da 1/8 K. Così, assieme ai 1024 bytes di IC4 e IC5, vi sono 1.152 bytes di RAM disponibili sulla piastra di base dello Junior-Computer. Per l'indirizzamento di queste 128 celle di memoria nell'IC3, servono i collegamenti di indirizzo A0...A6. La linea indirizzo A7 è collegata al piedino \overline{RS} . Questa linea di indirizzo decide se la CPU desidera scambiare dati con la RAM o con le porte ed il timer. Se A7 ha lo stato logico 0 è interessata la RAM. Quando A7 ha lo stato logico 1, sono invece richiamati il timer e le porte. Per ottenere con il minimo dispendio di mezzi una codificazione di indirizzi pressoché ininterrotta, altri due segnali CS vengono inviati alla PIA. CS2 corrisponde a K6 del decoder degli indirizzi IC6, mentre CS1 è collegata con la linea di indirizzi A9. CS2 è responsabile per l'indirizzamento generale di IC3, mentre tramite A9 (= CS1) viene fissata l'ultima parte di un blocco di memoria da 1 K. Molto sarebbe ancora da dire riguardo la PIA e il timer: a questi due interessanti elementi abbiamo perciò dedicato un intero capitolo nel volume 2 di questa serie.

Tastiera e display - il mondo esterno dello Junior-Computer

La tastiera ed il display costituiscono il mondo esterno incorporato nello Junior-Computer. Li definiamo incorporati perché entrambi risultano rigidamente collegati alla piastra di base. Quattro conduttori della porta B e sette conduttori della porta A assieme ad un pò di hardware (IC7 e IC11) necessitano per servire l'unità di ingresso/uscita. Come si vede dallo schema, la tastiera comprende 23 tasti disposti in forma di matrice. Questa matrice ha tre righe (ROW0...ROW2) e sette colonne (COL 0...COL 6). Sedici tasti della tastiera servono all'introduzione dei dati ed altri cinque sono tasti di comando che hanno una funzione di governo. Chiariremo comunque nel 3° capitolo di questo libro come si lavora con questi tasti.

I dati diretti al display ed i dati provenienti dalla tastiera corrono tramite i 7 conduttori della porta A. Lo Junior-Computer trae qui vantaggio dalla proprietà di poter lavorare con una porta in due direzioni. La gestione della tastiera e del display è affidata al programma Monitor nella EPROM. Esso serve periodicamente l'unità di entrata ed uscita. Il programma stesso interroga per verificare se un qualche tasto della tastiera risulta premuto. Quando è così il programma provvede a decodificare il tasto attivato. Anche i contatti di rimbalzo dei tasti vengono eliminati via Software.

Dato che i conduttori delle porte possono pilotare direttamente solo un carico di tipo TTL, per il pilotaggio del display è necessaria una stazione intermedia che possa fornire le relativamente forti intensità di corrente richieste dai segmenti. A ciò sono destinati la decodifica BCD-decimale IC7 e il pilota dei segmenti IC11. La decodifica BCD-decimale ha 10 linee in uscita; assumono lo stato logico 0 in funzione dello schema di bit applicato ai quattro ingressi A-B-C-D. Gli ingressi della decodifica sono comandati tramite i quattro collegamenti di porta PB1...PB4.

Le tre righe della matrice della tastiera sono collegate con le prime tre uscite di IC7. Un'uscita rimane inutilizzata e le uscite 4...9 comandano il catodo comune dei 6 display. Se uno di tali catodi viene portato a potenziale di massa dalla relativa uscita di IC7, si può far accendere a richiesta ciascun segmento del corrispondente display. Un segmento si illumina quando l'ingresso del buffer di IC1 ad esso corrispondente assume lo stato logico 0. Il comando dei display è affidato anch'esso al programma Monitor dello Junior-Computer. I simboli sul display si formano in corrispondenza alla conformazione dei bit sui conduttori PA 0...PA 6.

Lo IC7 comanda anche la tastiera. Se si preme un tasto qualsiasi della matrice il programma riconosce quale tasto è stato attivato. Anzi è il programma Monitor a riconoscere per primo se un tasto è stato attivato dal fatto che un qualunque conduttore della porta A ha assunto lo stato logico 0. Due tasti non risultano collegati con la nostra matrice: il tasto RST e il tasto ST. Questi due tasti tramite un dispositivo anti-rimbalzo (IC8) sono collegati direttamente con la CPU. La tastiera comprende quindi i tasti di Software S3...S23 ed i tasti di Hardware S1 ed S2.

Sulla piastra di base dello Junior-Computer sono ancora presenti due interruttori: S24 ed S25. Il primo interruttore serve a porre il computer nello "step by step mode"; col secondo si può escludere il display. Ciò risulta importante quando le connessioni della porta A, PA 0...PA 6, debbono comandare un dispositivo esterno: si evita così il fastidioso sfarfallio dei display a 7 segmenti.

Per poter lavorare lo Junior-Computer ha naturalmente bisogno di una alimentazione, ricavata dalla rete. Nella figura 5 è illustrato a tale scopo un circuito standard che comprende solo tre CI ed un paio di condensatori oltre al raddrizzatore. L'alimentatore fornisce le seguenti tensioni:

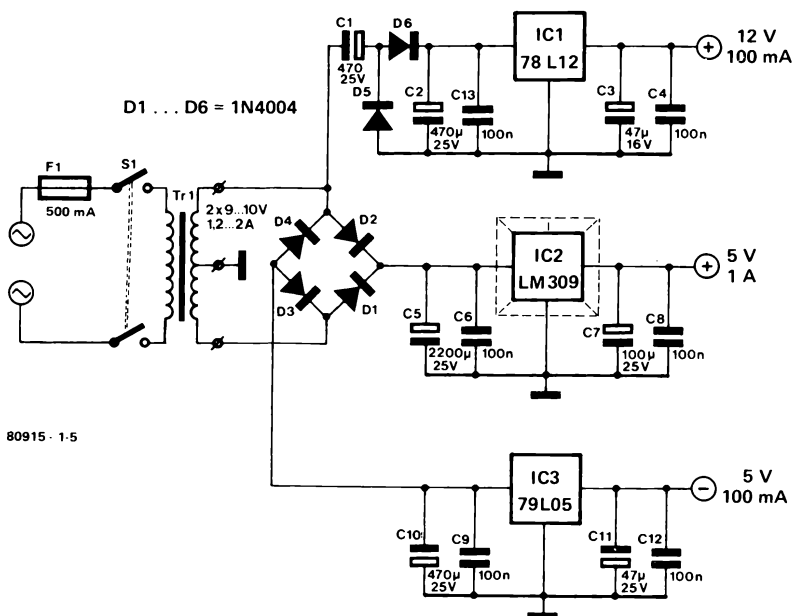


Figura 5. L'alimentatore da rete dello Junior-Computer fornisce tre tensioni di alimentazione stabilizzate.

- + 5V, carico continuo 1A
- 5V, carico continuo 100mA
- + 12V, carico continuo 100mA

Per tenere bassa la dissipazione dello stabilizzatore da +5V, è necessaria una bassa tensione sul secondario del trasformatore. Per poter utilizzare un trasformatore standard con due avvolgimenti secondari uguali si è previsto per lo stabilizzatore a +12V un duplicatore di tensione. Esso consta dei componenti C1, C2 nonché D5, D6.

Questo basti per lo schema dello Junior-Computer. Abbiamo cercato di non entrare in troppi dettagli senza tuttavia risultare superficiali. Quando si vuole assimilare la tecnica dei (micro) computer, l'Hardware e il Software dovrebbero procedere a braccetto: e questo ci sembra un requisito evidente.

Il montaggio dello Junior-Computer

Ed ora siamo finalmente arrivati al montaggio dello Junior-Computer. Montare da soli questo micro-computer risulta un'impresa piuttosto facile. Abbiamo già discusso i compiti dei CI che in esso sono impiegati. Ora dobbiamo, con questi CI e pochi altri componenti, assemblare i circuiti stampati base, dei display e del-

l'alimentatore. Sconsigliamo vivamente di fabbricare da sé il circuito stampato di base a doppia faccia. Si tratta di forare e provvedere ai contatti di oltre 600 fori ed è assai facile commettere degli errori.

Lo Junior-Computer è un "single board computer", ossia tutti i componenti sono applicati su un'unica piastra. Però, come sappiamo, lo Junior-Computer si compone di tre circuiti stampati. Perché allora definirlo "single board computer"? Uno di questi circuiti stampati comprende l'alimentatore che non appartiene al computer vero e proprio. Rimangono dunque il circuito stampato base e quello del display.

Quest'ultimo è un piccolo circuito stampato addizionale che è collegato saldamente al circuito stampato base. Certo, avremmo potuto montare i 6 display sul circuito stampato base, ma questo sarebbe risultato più grande e più caro. Per aggirare questo problema abbiamo montato il circuito dei display sulla piastra base inclinato ad un angolo di 45°. Ciò ha anche un fine pratico: un display obliquo si può leggere in modo più comodo! Perciò possiamo ben affermare che lo Junior-Computer è un "single board computer" benché consti di due circuiti stampati.

Il circuito stampato base

Date le dimensioni di questo volume, abbiamo dovuto riprodurre il circuito stampato base dello Junior-Computer rimpicciolito. Chi nonostante i nostri avvertimenti voglia realizzare da solo questo circuito stampato necessita delle dimensioni reali del circuito stesso. Le potrà trovare nel numero di aprile 1981 di *Elektor*. Se si realizza da soli il proprio circuito stampato, è altamente raccomandabile verificare tutti i collegamenti passanti che uniscono i punti opposti sulle due parti della piastra. La verifica può essere effettuata con un ohmmetro o ancor meglio con un cicalino ed un trasformatore da campanelli o addirittura il trasformatore del nostro alimentatore. La fig. 14a mostra il circuito di un simile dispositivo di controllo acustico. La fig. 14b mostra come si possa supplire ad un ponticello passante mediante un pezzettino di filo di rame o con il filo di collegamento di un componente.

Sulla parte superiore della piastra base a due facce si trovano la tastiera ed il display di sei cifre a sette segmenti. Nella parte inferiore sono collegati, capovolti, la CPU, la PIA, la memoria e gli altri componenti. La serigrafia dei componenti della parte superiore è illustrata in fig. 6, quella della parte inferiore in fig. 7. L'andamento delle piste sulla faccia superiore e inferiore è analogamente riportata nelle figg. 8 e 9. Per effettuare le saldature si raccomanda l'impiego di un saldatore adatto, del tipo a punta sottile, della potenza di 20-25 watt. Lo stagno impiegato per le saldature non dovrebbe avere un diametro superiore ad un millimetro, altrimenti è impossibile un esatto dosaggio dello stagno.

Per il montaggio dei pezzi sulla piastra procediamo preferibilmente come segue: le resistenze R1...R20 vengono inserite nei fori della piastra e quindi saldate. I fili sporgenti vengono quindi tagliati più corti possibile con un tronchesino affilato. Per quei lettori che non hanno molta confidenza con il codice a colori delle resistenze, forniamo in ausilio la seguente tabella:

100 K :	marrone-nero-giallo-(oro)
3 K 3 :	arancio-arancio-rosso-(oro)
4 K 7 :	giallo-violetto-rosso-(oro)
330 Ω :	arancio-arancio-marrone-(oro)
68 Ω :	blu-grigio-nero-(oro)
2 K 2 :	rosso-rosso-rosso-(oro)
68 K :	blu-grigio-arancio-(oro)

Quindi si salda il diodo D1. Questo diodo non deve essere un tipo qualsiasi ma esclusivamente un 1N4148. Nel saldare il diodo si presti attenzione alla giusta polarità, altrimenti l'oscillatore di clock del micro-processore non funzionerà. Quindi si passa ai condensatori. Nel montare i condensatori elettrolitici al tantalio, si deve analogamente prestare attenzione alla corretta polarità. Il polo positivo dei condensatori elettrolitici nella serigrafia sul circuito stampato è indicato da una barra vuota, il polo negativo da una barra piena. Alcuni condensatori elettrolitici al tantalio non riportano l'indicazione del reoforo positivo con un "più". Al suo posto vi è un punto colorato: se si osserva il punto colorato dal davanti con i due reofori puntati verso il basso, il reoforo di destra costituisce il polo positivo del condensatore elettrolitico al tantalio.

Dopo aver saldato i condensatori, le resistenze ed il diodo sulla parte superiore della piastra di base, si montano i 23 tasti ed i due interruttori sulla faccia anteriore. Gli attacchi a saldare degli interruttori vanno collegati con filo isolato con il circuito stampato. Nel montaggio dei tasti si badi che questi siano fissati verticalmente fra di essi.

Quindi sulla piastra vengono montati i circuiti integrati. Per IC1 ed IC2 sono previsti zoccoli da circuiti integrati con 40 piedini, per IC2 uno zoccolo con 24 piedini. Gli zoccoli per questi circuiti integrati dovrebbero avere contatti dorati. Gli altri CI vengono direttamente saldati sul circuito stampato. Qui infatti non dovrebbero intervenire fenomeni di invecchiamento dei contatti. La EPROM (IC2) deve in ogni caso essere provvista di zoccolo poiché solo in tal caso il programma Monitor potrà venir sostituito da un altro programma.

Ora è il turno del quarzo da 1 MHz e del LED nel tasto GO (S22) che vengono inseriti sul circuito stampato. Nel montaggio del LED si presti attenzione alla corretta polarità. Il reoforo più lungo di questo diodo è il polo positivo (rappresentato dal triangolo nella seri-

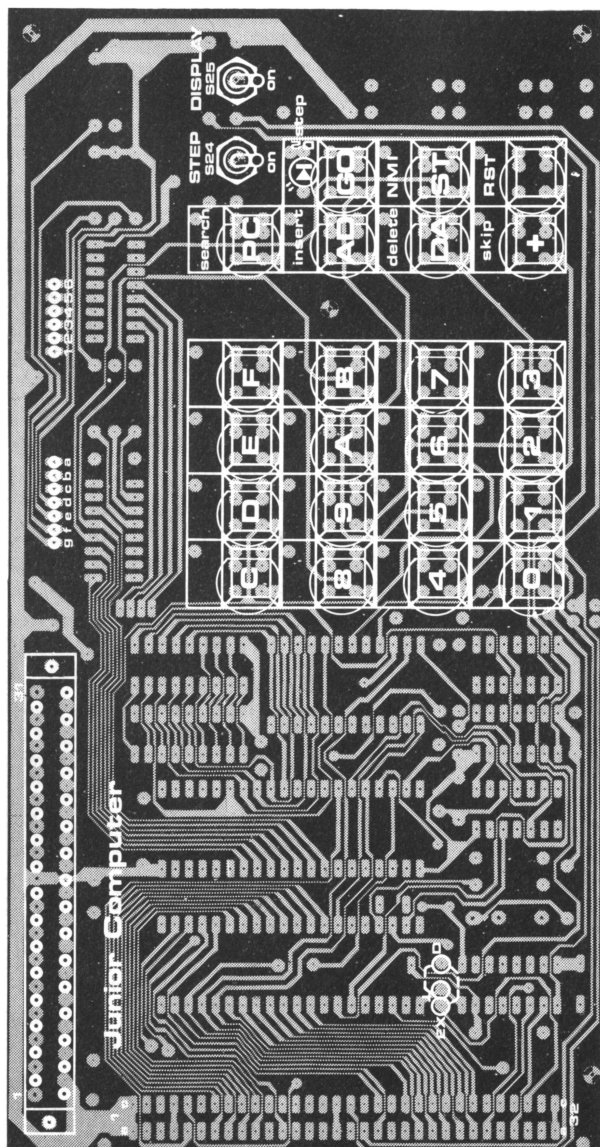


Figura 6. Lato superiore e serigrafia del circuito stampato di base a doppia faccia.

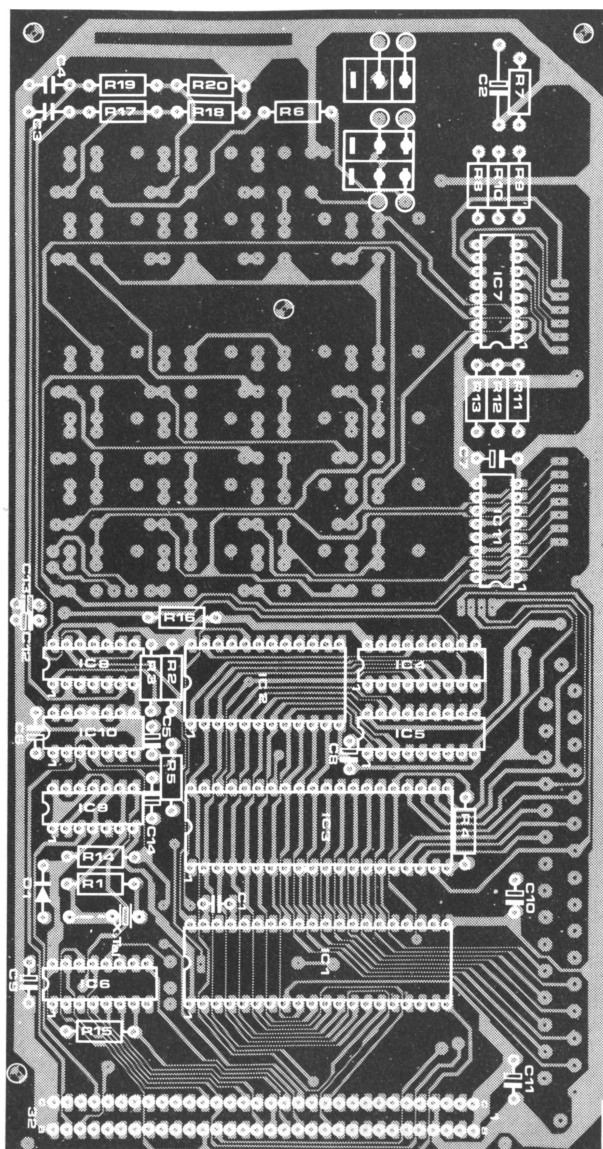


Figura 7. Lato inferiore del circuito stampato di base. Su questo lato si trovano montati le resistenze, i condensatori, i diodi, i CI e il quarzo da 1 MHz.

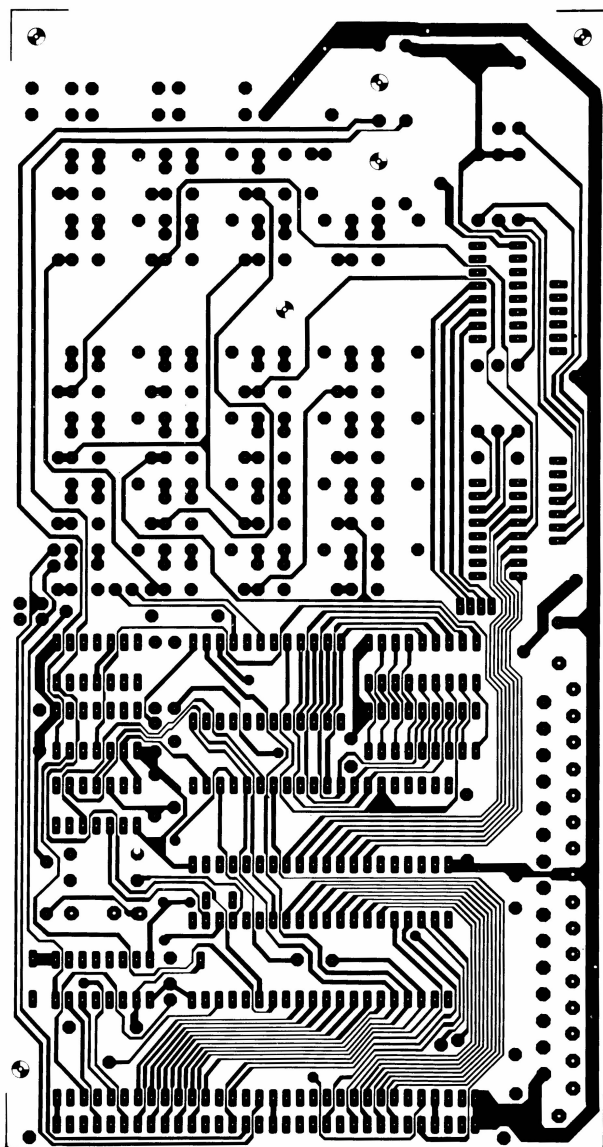


Figura 8. Disegno delle piste di rame sulla parte superiore del circuito stampato di base (scala rimpicciolita).

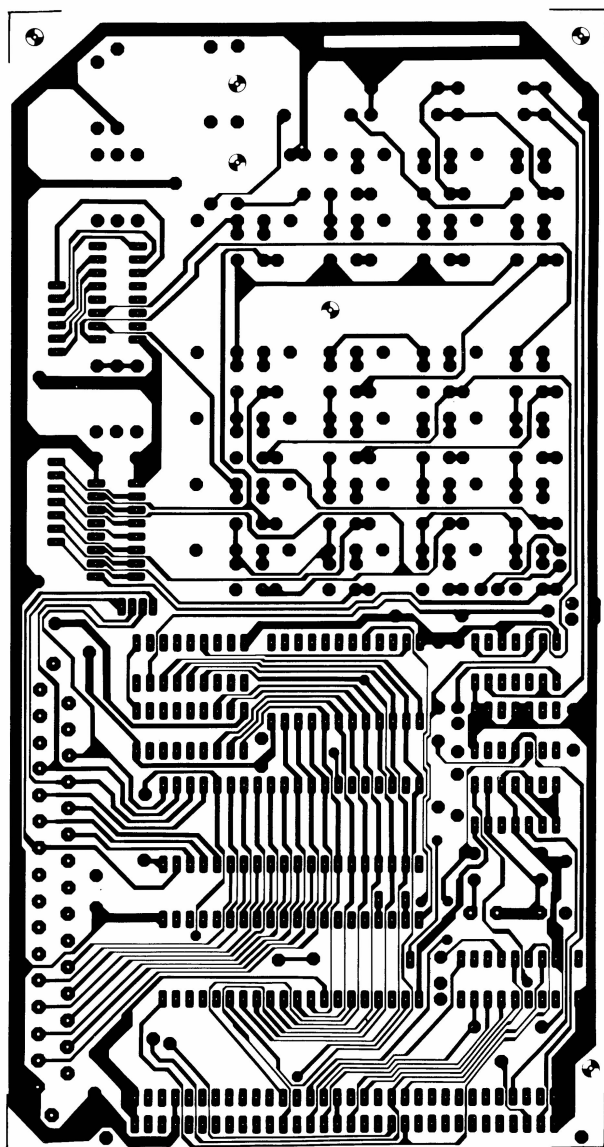


Figura 9. Disegno delle piste di rame sul lato inferiore del circuito stampa di base.

Lista dei componenti per la piastra di base

Il circuito elettrico dello Junior-Computer è illustrato in fig. 4. Il disegno del circuito stampato e le relative serigrafie sono riportati nelle figg. 6 ... 8.

Resistenze

R1 = 100 k
R2,R3,R4,R14,R15,R16 = 3k:
R5 = 4k7
R6 = 330 Ω
R7 ... R13 = 68 Ω
R17,R19 = 2k2
R18,R20 = 68 k

Semiconduttori:

IC1 = 6502 (Rockwell)
IC2 = 2708
IC3 = 6532 (Rockwell)
IC4,IC5 = 2114
IC6,IC7 = 74145
IC8 = 556
IC9 = 74LS132
IC10 = 74LS01, 7401
IC11 = ULN2003 (Sprague)
D1 = 1N4148

Condensatori:

C1 = 4p7 ceramico
C2 = 47 μ /6 V al tantalio
C3,C4 = 100 n MKH
C5 ... C14 = 1 μ /35 V Tantalio


Varie:

XTAL = Quarzo da 1 MHz
S1...S21, S23 = Digitast (shadow)
S22 = Digitast munito di spia LED (shadow)
S24 = Deviatore a 2 vie, 2 posizioni
S25 = Deviatore a 1 via, 2 posizioni
* Connettore a pettine a 31 punti
C42334-A56-A63 (Siemens) DIN 41617
* Connettore a pettine a 64 punti
C42334-A191-A502 (Siemens)*
DIN 41612
* (non richiesto a questo stadio del montaggio)
Zoccolo per CI a 24 piedini
Zoccolo per CI a 40 piedini

grafia del circuito stampato). Se si vuole risparmiare denaro, non è necessario ancora montare il connettore di espansione a 64 poli sulla piastra. Esso è necessario infatti solo nell'espansione dello Junior-Computer. In tal caso le linee di alimentazione vengono saldate direttamente sulle piazzole di questo connettore. I collegamenti delle tre tensioni di alimentazione sul connettore di espansione sono:

+ 5V : piedino 1 A oppure 1 C
massa = zero V : piedino 4 A o 4C o 32 A o 32 C
- 5V : piedino 18 A
+ 12V : piedino 17 C

Quando invece si impiega il connettore questi collegamenti si possono facilmente ritrovare perché sulla faccia inferiore della piastra sono indicati i numeri dei collegamenti. La lunghezza dei cavi di collegamento tra la parte alimentazione e lo Junior-Computer può arrivare sino ad un metro.

Si saldano sulla piastra il connettore delle porte a 31 poli e un ponticello. Questo ponticello giace tra i punti  e D.

Dopo aver saldato tutti i componenti sulla piastra di base, si fissano sulla parte inferiore della piastra con viti M3 sei distanziatori in alluminio con filettatura interna. Questi distanziatori dovrebbero avere una lunghezza di 10 mm. Due distanziatori vanno montati sotto il connettore delle porte, gli altri quattro negli angoli della piastra di base e nella zona di suddivisione tra i tasti. In tal modo la piastra acquista una grande rigidità meccanica ed anche una forte pressione sui tasti non riesce più a piegare la piastra.

Abbiamo così completato il montaggio meccanico ed elettrico della piastra di base. Prima di passare al montaggio degli altri circuiti stampati, dovremmo sottoporre la piastra di base ad un controllo accurato. Si deve in particolare badare a verificare che tutti i componenti siano situati in posizione corretta e che sia rispettata, e corrisponda alla serigrafia del circuito stampato, la polarità dei condensatori e dei diodi.

Il circuito stampato dei display

La serigrafia e l'andamento delle piste di rame sono riportate nelle figg. 10 e 11. Dato che le piste sul circuito stampato dei display sono molto sottili, le saldature devono essere effettuate con partico-

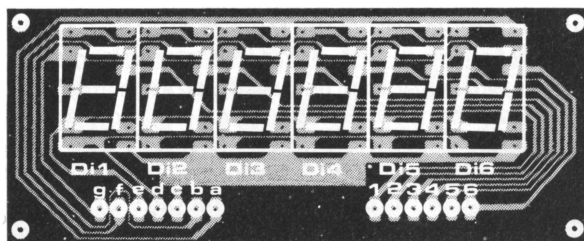


Figura 10. Serigrafia del circuito stampato del display.

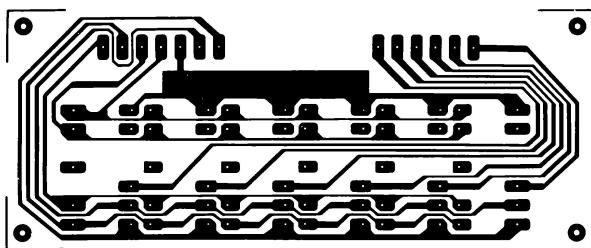


Figura 11. Disegno delle piste di rame sul circuito stampato dei display.

Lista dei componenti per la piastra dei display

Semiconduttori: Di1, Di2, Di3, Di4, Di5, Di6 = MAN4640A, a catodo comune

lare cura. Su questa piastra vengono montati solamente 6 display e 13 fili di collegamento. È meglio cominciare con i fili che costituiscono i collegamenti con la piastra base. Si può impiegare del filo di rame non isolato da 0.8 mm., che viene tagliato in pezzi lunghi 20 mm. ciascuno. Questi 13 pezzi di filo vengono introdotti nei fori all'uopo previsti "a...g" e "1...6" e quindi saldati. Dopo aver fissato questi fili, li si raddrizza mediante una pinza a becchi. I display vengono fissati direttamente sul circuito stampato e saldati senza l'impiego di zoccoli. Dopo questo lavoro preliminare, colleghiamo il circuito stampato del display con il circuito stampato base dello Junior-Computer ed effettuiamo le relative saldature. La piastra porta-display viene quindi inclinata sino ad assumere un angolo di 45° rispetto alla piastra di base. Naturalmente non è indispensabile montare la piastra porta-display nel modo indicato sulla piastra base dello Junior-Computer. Se ad esempio il computer verrà collocato entro un contenitore, la piastra porta-display potrà essere sistemata ad una distanza qualsiasi dalla piastra di base. In tal caso i 13 collegamenti tra le due piastre verranno effettuati con una piattina multipolare (flat cable).

Il circuito stampato dell'alimentatore da rete

La serigrafia e l'andamento delle piste di rame sono riportate nelle figg. 12 e 13. Non dovrebbe esservi difficoltà nel montaggio di questa piastra. Si deve solo prestare attenzione alla giusta polarità dei diodi e dei condensatori elettrolitici. Nel montaggio dello stabilizzatore dei +5V si farà ricorso ad un dissipatore termico del tipo a ragno. Prima di saldare IC2 esso verrà fissato con due viti M 3 sulla piastra. La cosa migliore è di impiegare un dissipatore che porti forature adatte ad un contenitore TO3.

Terminato di montare i componenti sul circuito stampato dell'alimentatore, lo colleghiamo al trasformatore di rete. Il primario del trasformatore va collegato con l'interruttore di rete S1 e il fusibile F1. E con ciò abbiamo terminato il montaggio di tutti i componenti dello Junior-Computer.

Ed ora siamo al momento culminante!

Ora metteremo in funzione lo Junior-Computer. Prima di collegare l'alimentatore da rete con le piastre del computer, eseguiremo per misura di sicurezza un paio di controlli. Inseriamo con S1 l'alimentatore e con un tester verifichiamo i valori delle tre tensioni di alimentazione di +5V, -5V e +12V. Queste tensioni devono corrispondere ai valori nominali con una tolleranza del 5%. Se non risulta così occorre procedere ad un nuovo controllo dell'alimentatore. Normalmente questo tipo di alimentatore non dovrebbe dare guai di sorta. Se l'alimentatore risulta in ordine, lo colleghiamo con lo Junior-Computer. A questo punto è necessario verificare

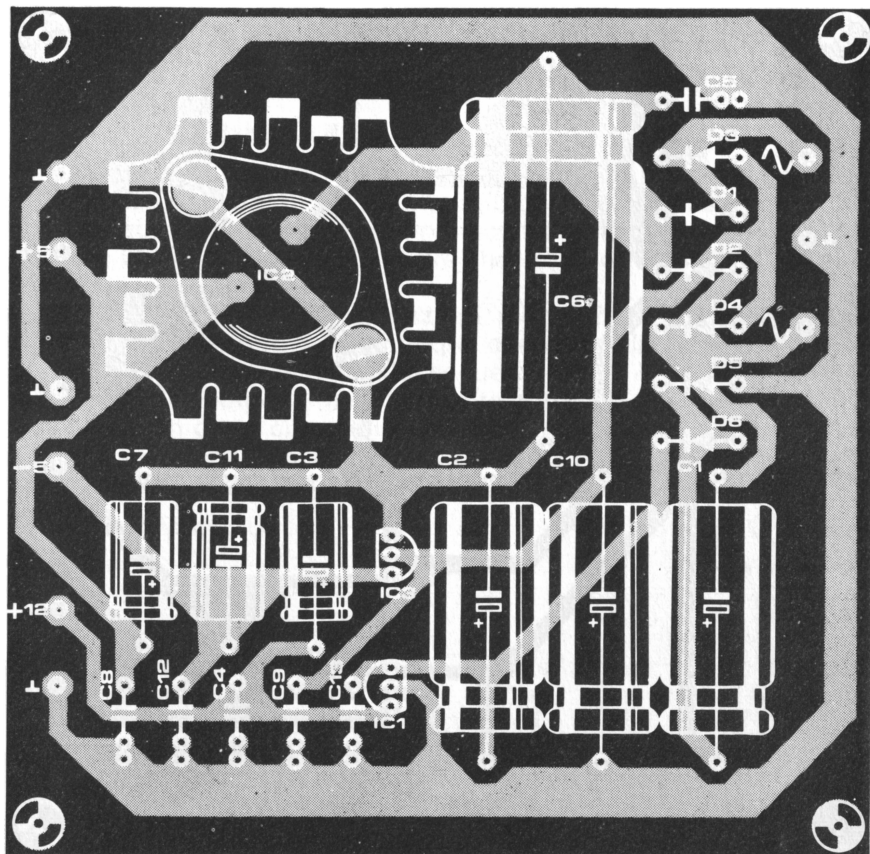


Figura 12. Serigrafia dei componenti del circuito stampato dell'alimentatore.

Lista dei componenti per la piastra dell'alimentatore

Il circuito dell'alimentatore è mostrato in fig. 5. Le figg. 12 e 13 illustrano la serigrafia dei componenti ed il disegno delle piste del circuito stampato.

Condensatori:

C1, C2, C10 = 470 μ /25 V
 C3, C11 = 47 μ /25 V
 C4, C5, C8, C9, C12, C13 = 100 n
 MKH
 C6 = 2200 μ /25 V
 C7 = 100 μ /25 V

Semiconduttori:

IC1 = 78L12ACP (al 5%)
 IC2 = LM309K + dissipatore termico
 per contenitore TO3
 IC3 = 79L05ACP (al 5%)
 D1...D6 = 1N4004

Varie:

Tr1 = Trasformatore di rete, secondario
 2x9...10V/1,2...2A
 S1 = interruttore di rete, a 2 vie
 F1 = fusibile da 500 mA, ritardato con
 portafusibile (esterno)

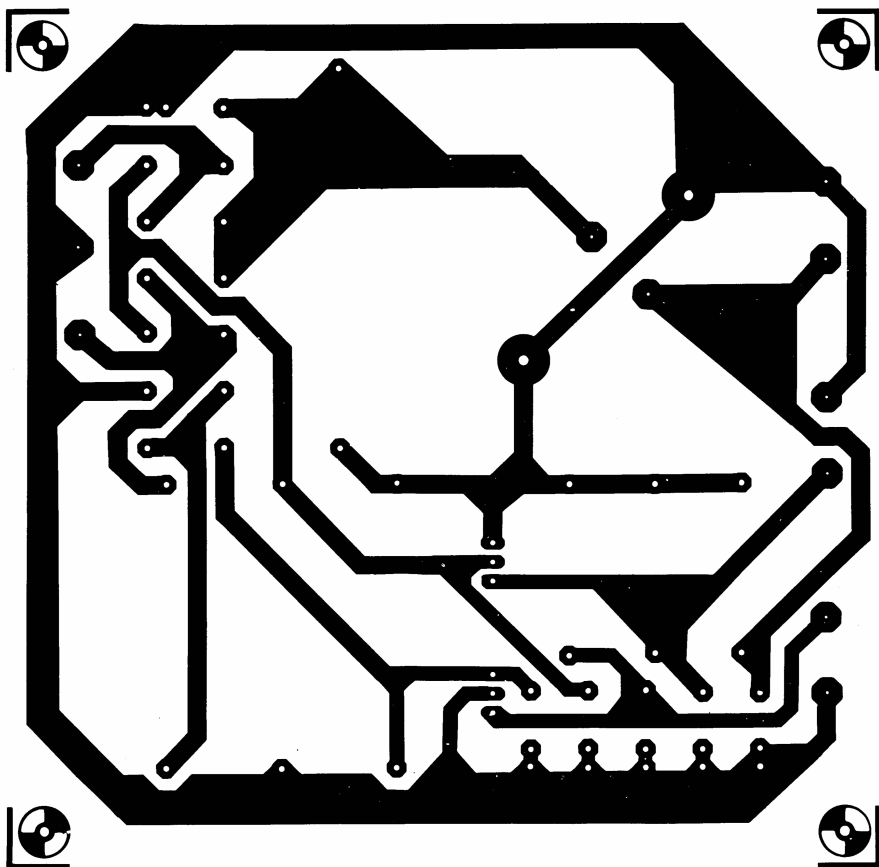


Figura 13. Disegno delle piste di rame sul circuito stampato dell'alimentatore.

l'esattezza dei quattro collegamenti. Eventuali errori nel collegamento delle tensioni di alimentazione possono senz'altro nuocere alla "salute" dello Junior-Computer.

Dopo aver collegato la piastra di base con la piastra di alimentazione non collegata alla rete, inseriamo lo "step by step mode" ossia poniamo l'interruttore S24 in posizione "OFF".

Quindi poniamo l'interruttore dei display S25 in posizione "ON" e reinseriamo l'alimentatore. Il display rimane buio. Niente paura! Ciò è del tutto normale quando si applica allo Junior-Computer la tensione di alimentazione. Affinché il display si illumini occorre prima premere il tasto RST. Quando si illumina, esso mostra dei numeri esadecimali a caso. Cosa si intenda per numeri esadecimali sarà descritto nel capitolo seguente di questo libro. La fig. 15 mostra in quali modi i segmenti dei display si possono illuminare. Se

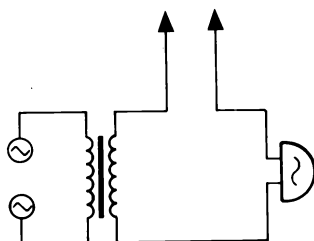
tutti i 6 display si illuminano dopo il rilascio del tasto RST, significa che la Hardware dello Junior-Computer funziona regolarmente.

Che fare se il display non si illumina?

Riteniamo che siano rari i casi in cui il display rimanga buio. I circuiti stampati sono stati progettati e sviluppati con la massima cura e il relativo Hardware ha subito un collaudo di quasi due anni. Tuttavia in singoli casi può avvenire che lo Junior-Computer non funzioni.

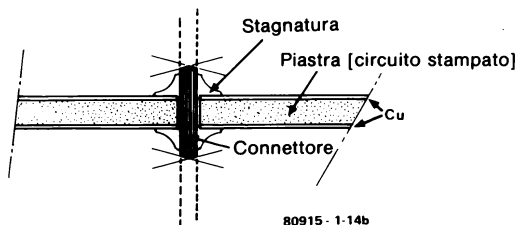
Assunto che l'alimentatore funzioni regolarmente, benché lo Junior-Computer non dia comunque altri cenni di vita, il motivo può risiedere in uno o l'altro dei seguenti casi:

- Saldando i componenti sulla piastra di base o dei display si è formato un corto circuito imprevisto. Chi ha poca confidenza con il saldatore dovrebbe perciò verificare tutti i punti di saldatura dello Junior-Computer con cura.



80915 - 1-14a

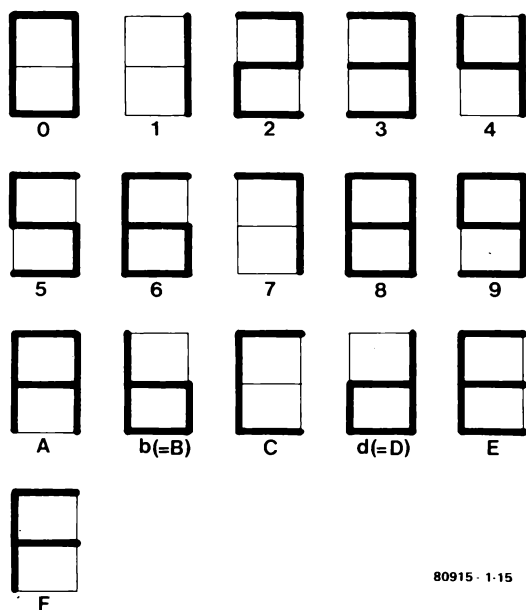
Figura 14a. Il controllo dei ponticelli passanti del circuito stampato di base a doppia faccia può essere effettuato con un ohmmetro. Un metodo più elegante è il controllo "acustico": si richiedono un trasformatore ed un cicalino. Tale metodo va però impiegato prima del montaggio degli altri componenti sul circuito stampato.



80915 - 1-14b

Figura 14b. Illustrazione del modo in cui effettuare un ponticello passante su di un circuito stampato autocostruito con uno spezzone di filo o mediante il reoforo di un componente.

- Cattive saldature possono anch'esse essere uno dei motivi di mal funzionamento del computer. In casi dubbi si provveda a ri-passare con lo stagno eventuali saldature "fredde".
- Cattivi contatti negli spinotti e nelle molle dei connettori. Impiegando connettori già usati ciò può facilmente verificarsi. Cattivi contatti sono possibili anche tra i CI e i relativi zoccoli. Spesso nell'introdurre i CI negli zoccoli si può avere la piegatura di un piedino. Ciò avviene in realtà più facilmente di quanto non si creda!
- Diodi e condensatori elettrolitici sistemati con polarità errata possono essere un'altra delle cause per cui il computer non funziona. Inoltre i collegamenti delle tre tensioni di alimentazione devono risultare saldati ai corretti piedini del connettore di espansione. Questi sono i più probabili casi di errore. Forniamo comunque un certo numero ulteriore di suggerimenti per rintracciare eventuali errori.
- Misuriamo con un tester la tensione tra il piedino 13 e il piedino 7 di IC8. Questa tensione deve aggirarsi su 5V. Se si preme ora il tasto RST questa tensione deve scendere a circa 0,5V. Se ciò non avviene, il motivo può essere in uno dei seguenti compo-



80915 - 1-15

Figura 15. Dopo la pressione del tasto RST, sul display dello Junior-Computer compaiono 6 cifre esadecimali a caso. Qui sono illustrati tutti i tipi di cifre che possono comparire illuminate sul display.

nenti difettosi: il doppio timer IC8, la resistenza di pull up R2 o il tasto RST stesso.

- Se tutto quanto sopra risulta in ordine, controlliamo con un ohmmetro se il piedino 12 di IC6 risulta a massa. Se non è così, vi è un errore nel ponticello sulla piastra di base o questo non risulta saldato.
- Anche il generatore di clock può essere controllato facilmente. Con un oscilloscopio possiamo misurare le tensioni sui piedini 30A e 27A del connettore di espansione. Ivi risultano presenti i due segnali ad onda quadra $\Phi 1$ e $\Phi 2$. La frequenza deve risultare 1 MHz ($= 1 \mu s$) e l'ampiezza relativa dovrebbe essere di 3...5V picco-picco. Questa verifica è facile in particolare con uno oscilloscopio a due tracce, per controllare che questi due segnali di clock non si sovrappongono. Se sullo schermo non si osserva alcuna tensione ad onda quadra, significa che il generatore di clock non oscilla. I componenti da sospettare risultano allora C1, IC9, D1 o il quarzo.

Se comunque dovessero verificarsi delle difficoltà con la Hardware dello Junior-Computer, mettetevi in contatto scritto o telefonico con noi. Siamo pronti a darvi cordiale consulenza ed assistenza!

Un contenitore per lo Junior ?

Non c'è molto da dire sull'impiego di un contenitore per lo Junior-Computer. Se lo si acquista o lo si fabbrica da soli è da preferirsi un contenitore abbastanza spazioso. Il contenitore deve infatti poter alloggiare i seguenti elementi di un computer completo:

- Lo Junior-Computer
- L'Elekterminal con tastiera ASCII
- 5 Euro-schede innestate sul bus a circuito stampato
- Diverse prese per spine ad innesto per il collegamento di due registratori a cassette, due Floppy Disk, una stampante, eccetera.
- Spazio per un alimentatore "robusto" con relativo trasformatore (passando al sistema espanso si impiegherà il trasformatore modello SC/MP, con secondari da 10V/6.5A e 15V/1A).

Anche nel caso di successive espansioni la tastiera ed il display dello Junior-Computer devono risultare accessibili dall'esterno. Ossia si deve poter lavorare alternativamente con la tastiera esadecimale sulla piastra di base, il display a 6 cifre a 7 segmenti, nonché una tastiera ASCII ed una stampante. Chi volesse collegare allo Junior-Computer una stampante a foglia metallica, lo può eventualmente inserire nel contenitore. Si può impiegare un qualsiasi tipo di stampante a foglia metallica (ad esempio stampante DATAMEGA). La cosa importante è che la testina di stampa abbia 7 aghi.

Chi non desidera ricorrere ad un contenitore può ricorrere ad una

soluzione che risulta economica e che abbiamo sperimentato con successo nel laboratorio di Elektor.

Lo Junior-Computer ed il suo alimentatore con relativo trasformatore vengono montati su una piastra di plexiglas spessa 5 mm. Questa lastra ha le misure di 25x30 cm. Sopra l'alimentatore viene poi disposto un coperchio di plexiglas per evitare qualsiasi contatto con la tensione di rete.

Così abbiamo descritto la Hardware dello Junior-Computer. Nei capitoli che seguono ci occuperemo principalmente della programmazione di questa nostra macchina. Diamo quindi inizio a questa nuova avventura!

Le EPROM vengono poste in commercio non programmate. Però una EPROM non programmata risulta inutilizzabile per lo Junior-Computer. Solo dopo l'ausilio di una macchina di programmazione (EPROM-Programmer) viene "impresso" il programma Monitor dello Junior-Computer nella EPROM esso può venir inserito nel suo zoccolo. Se possedete direttamente un tale EPROM-Programmer allora sarete in grado con l'ausilio del "Monitor Dump" esadecimale posto al termine di questo libro di programmare da soli la EPROM.

Pensare e calcolare in digitale

“Lo puoi contare sulle tue dieci dita”. Certamente ciascuno avrà udito o detto qualche volta queste parole. Noi siamo in grado di far conto sulle dieci dita perché utilizziamo il sistema decimale per i nostri calcoli.

Lo (Junior-)Computer si occupa anch'esso di calcoli e dei relativi dati: ma gli mancano le dieci dita per contare. Deve accontentarsi di due: ogni sistema digitale infatti lavora con il sistema binario di numerazione. Di questo tratta il capitolo che segue.

A che pensa la maggior parte delle persone quando essi sentono pronunciare il numero 1984? Forse al romanzo futuribile “1984” di George Orwell? Qualcuno - ad esempio i matematici - decompone questo numero in potenze di dieci: $(1 \cdot 10^3 + 9 \cdot 10^2 + 8 + 10 \cdot 4 \cdot 10^0 = 1984)$.

I calcolatori lavorano con numeri che per noi umani abituati a pensare in modo decimale non sono molto comuni. Perciò impareremo in questo capitolo come si devono trattare i numeri binari ed esadecimali con i quali lavora lo Junior-Computer. È indispensabile leggere questo capitolo se si vuol poi imparare come programmare.

Il sistema decimale o delle potenze di dieci è certamente il sistema numerico più conosciuto e più usato. Questo sistema si fonda sul numero 10. Si dice pure che 10 è la base del sistema decimale. Il sistema decimale adopera i simboli in cifre 1; 2; 3; 4; 5; 6; 7; 8; 9; 0. Si impiegano in tutto dieci simboli per la rappresentazione di numeri qualsiasi semplicemente scrivendoli uno dietro l'altro. Un calcolatore digitale, come già detto, può trattare le informazioni soltanto sotto forma di numeri. Se ad esempio un computer deve controllare una tensione, una corrente od una frequenza, ciò gli risulta possibile solo se noi associamo ad esempio ad una tensione di 5 volt il numero 5, ad una tensione di 6 volt il numero 6, e così via. Una associazione di questo tipo è tecnicamente realizzabile,

ma complicata e cara. Risulta molto più semplice limitarsi a soli due stati e attribuire ad essi le cifre 0 ed 1. Tecnicamente ciò si realizza con degli interruttori in modo assai facile procedendo come segue:

"0" = stato logico 0 = nessuna corrente, nessuna tensione, interruttore aperto, stato logico "basso" ovvero "L", lampada spenta (quando si adotta la logica positiva).

"1" = stato logico 1, = corrente o tensione inserita, interruttore chiuso, stato logico "alto" ovvero "H", lampada accesa. I moderni calcolatori digitali, così come lo Junior-Computer, lavorano esclusivamente su questo principio. Un calcolatore si serve quindi di un sistema numerico che si limita a due cifre: 0 ed 1, il sistema di numerazione binari. Per un numero binario valgono naturalmente le stesse regole di calcolo che per un sistema decimale:

$$1010_2 = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0$$

L'unica differenza consiste nel fatto che il sistema di numerazione binario ha la base 2, mentre il decimale ha la base 10. Quando nel calcolo col sistema numerico decimale in una posizione si supera la riserva di cifre disponibili (ossia il risultato superiore a 9) si effettua un riporto sulla posizione successiva. Nel sistema binario si ha il riporto già quando si supera il numero 1. Prima di trattare ulteriormente la teoria numerica utilizzata da un micro-computer, dobbiamo chiarire i concetti di bit, byte e dei codici digitali.

Bit, byte ed i codici digitali

Bit è l'abbreviazione di "Binary digit" ossia di cifra binaria. L'unità fondamentale di informazione nella tecnica digitale è costituita dal bit. Un ben noto elemento di memoria, il Flip-flop D, può ad esempio memorizzare un bit. Una successione di bit come 0101 può venir convertita in numero decimale: $0101 = 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 5_{10}$. Per distinguere i vari numeri nei diversi sistemi di numerazione, la base del sistema di numerazione - come nell'esempio precedente - viene scritto come indice. Quindi $0101_2 = 5_{10}$ (che si pronuncia: zero uno zero uno nel sistema binario corrispondente a cinque nel sistema decimale). La successione di bit 01000101 corrisponde alla lettera "E" nel codice ASCII. Di questo codice ci occuperemo più oltre. Se si uniscono assieme più bit, si ottiene una "parola" della lunghezza di più bit. La parola in binario 0101_2 è lunga 4 bit, così come la parola ELEKTOR è lunga 7 lettere. Una parola binaria composta di 8 bit ovvero lunga 8 bit viene chiamata byte. Nel capitolo 1 abbiamo trattato l'organizzazione di memoria. La RAM dello Junior-Computer può memorizzare parole lunghe 4 bit. Per memorizzare 8 bit ossia 1 byte necessitano quindi 2 di queste RAM sotto forma di CI. Nella EPROM invece si memorizzano parole della lunghezza di 8 bit. Ogni parola letta da un tale elemento di memoria corrisponde ad 1 byte. Un codice

digitale si compone analogamente ad una parola di singoli bit, ossia di una successione di 0 ed 1. I codici digitali rappresentano i dati in modo tale che il computer li possa assimilare. Quando il programmatore vuole introdurre il numero 9_{10} in un computer, deve prima codificare questo numero in modo che il computer lo possa comprendere.

Esistono diversi tipi di codici digitali:

- codici con i quali vengono codificati le cifre decimali 0...9, ad esempio codice BCD, codice GRAY.
- Codici che trovano impiego nei circuiti digitali, ad esempio per contare in avanti od all'indietro quanto capita un determinato avvenimento.
- Codici con i quali si possono rappresentare cifre decimali, lettere od operazioni in forma binaria: codice ASCII, codice Baudot.
- Codici che rappresentano il repertorio di istruzioni del micro-processore 6502.

Le abbreviazioni di alcuni dei codici citati sopra hanno il seguente significato:

BCD = Binary Codec Decimal = codice decimale codificato binario

ASCII = American Standard Code for Information Interchange = codice standard americano per lo scambio di informazioni.

Numeri binari, decimali ed esadecimali

Quando impieghiamo i numeri siamo abituati a calcolare con essi nel sistema decimale. Le singole cifre di un numero hanno valori che dipendono dalla loro posizione. Perciò la stessa cifra nei diversi sistemi di numerazione può avere valori differenti. Prendiamo ad esempio l'abituale sistema decimale: in esso il numero $1984_{10} = 1 \cdot 10^3 + 9 \cdot 10^2 + 8 \cdot 10^1 + 4 \cdot 10^0$; l'avevamo già ricordato all'inizio del capitolo. Se consideriamo invece il numero 1984 nel sistema numerico a base 16 si ha: $1984_{16} = 1 \cdot 16 \cdot 16 \cdot 16 \cdot 16 \cdot 16 \cdot 16 \cdot 16 \cdot 6532_{10}$. Il sistema numerico a base 16 viene chiamato sistema numerico esadecimale (in certi casi abbreviato sistema Hexal). Dobbiamo adesso imparare come si trasforma un numero da un sistema numerico all'altro. Mettiamo prima a raffronto ancora una volta i vari sistemi numerici citati con i loro simboli per le relative cifre:

- il sistema binario: 0; 1.
 - il sistema decimale: 0; 1; 2; 3; 4; 5; 6; 7; 8; 9.
 - il sistema esadecimale: 0; 1; 2; 3; 4; 5; 6; 7; 8; 9; A; B; C; D; E; F.
- Come si vede, il sistema esadecimale impiega i simboli in cifre 0...9 esattamente come il sistema decimale. Per poter impiegare una sola "cifra" per i valori superiori a 9 vengono introdotti i

simboli A...F. La fig. 1 mostra come sono costituite le successioni numeriche nei sistemi binario, decimale ed esadecimale.

Le regole di calcolo a cui ci ha abituato il sistema decimale sono trasferibili pari pari a tutti gli altri sistemi numerici. Nei paragrafi seguenti descriviamo le quattro operazioni fondamentali - addizione, sottrazione, moltiplicazione e divisione - nel sistema numerico binario.

L'addizione binaria

Per l'addizione nel sistema decimale, come è noto, si presenta la necessità di un riporto sulla posizione successiva quando si supera il numero 10. L'esempio di addizione che segue serve a ricordarsi come questo avviene.

$$\begin{array}{r}
 129_{10} \quad 1^{\circ} \text{ numero} \\
 + 243_{10} \quad 2^{\circ} \text{ numero} \\
 \hline
 1 \quad \text{riporto} \\
 372
 \end{array}$$

Quando si addizionano due numeri binari si ha un riporto da una posizione all'altra quando la somma in una colonna risulta uguale o superiore a 2. Vediamolo col seguente esempio:

$$\begin{array}{r}
 101101 \quad 1^{\circ} \text{ numero binario} \\
 + \quad 10101 \quad 2^{\circ} \text{ numero binario} \\
 \hline
 1111 \quad 1 \quad \text{riporti} \\
 1000010
 \end{array}$$

Nella tecnica dei computer questo riporto viene chiamato Carry.

La sottrazione binaria

Nella sottrazione si toglie dal minuendo (numero dal quale si deve sottrarre un altro) il valore del sottraendo (numero che viene sottratto). Il risultato si chiama anche differenza. Nel sistema binario la sottrazione viene effettuata esattamente come nel sistema decimale ordinario. Cominciamo col sottrarre quindi due cifre decimali:

$$\begin{array}{r}
 1984 \text{ minuendo} \\
 - \quad 199 \text{ sottraendo} \\
 \hline
 11 \text{ "prestiti"} \\
 1785 \text{ differenza}
 \end{array}$$

Per poter sottrarre dall'ultima posizione a destra il numero 9 da

binario	decimale	esa- decimale	nibble
0	0	0	0000
1	1	1	0001
10	2	2	0010
11	3	3	0011
100	4	4	0100
101	5	5	0101
110	6	6	0110
111	7	7	0111
1000	8	8	1000
1001	9	9	1001
1010	10	A	1010
1011	11	B	1011
1100	12	C	1100
1101	13	D	1101
1110	14	E	1110
1111	15	F	1111
10000	16	10	
10001	17	11	
.	.	.	
.	.	.	
.	.	.	
.	.	.	

Figura 1. In questa tabella sono posti a confronto i valori dei numeri binari, decimali ed esadecimali. La scrittura esadecimale è una scrittura abbreviata dei numeri binari.

numero 4, si è provveduto al “prestito” della posizione successiva dalla sinistra. Nel caso dei numeri binari la sottrazione avviene nello stesso modo:

$$\begin{array}{r}
 11001001 \text{ minuendo} \\
 - 1001000 \text{ sottraendo} \\
 \quad \text{“prestiti”} \\
 \hline
 10000001 \text{ differenza}
 \end{array}$$

In questo esempio nel minuendo ogni cifra è maggiore o eguale a quella di posizione corrispondente del sottraendo: non si deve quindi ricorrere a “prestiti” dalla posizione successiva. Nel secondo esempio il caso è diverso e si devono effettuare “prestiti”.

$$\begin{array}{r}
 11000001 \text{ minuendo} \\
 - 1111110 \text{ sottraendo} \\
 111111 \text{ “prestiti”} \\
 \hline
 01000011 \text{ differenza}
 \end{array}$$

La moltiplicazione binaria

Anche la moltiplicazione nel sistema binario segue le medesime regole della moltiplicazione nel sistema decimale. Si formano cioè separatamente i prodotti con le singole posizioni del moltiplicatore (il numero col quale l'altro deve essere moltiplicato) e i singoli prodotti opportunamente spostati per assumere la posizione corretta, vengono addizionati. Rispetto alla moltiplicazione decimale il sistema binario presenta alcune semplificazioni. Dato che nelle varie posizioni del moltiplicatore si possono avere solo i valori numerici 0 o 1, il moltiplicando (numero col quale viene moltiplicato il moltiplicatore) deve essere semplicemente moltiplicato per 0 o 1. Nella moltiplicazione binaria le operazioni parziali consistono quindi solo nella addizione e nello spostamento di una posizione. È importante ricordarlo quando in seguito svilupperemo i programmi. Prima di riportare un esempio di moltiplicazione binaria, ricordiamoci ancora con un esempio come viene effettuata la moltiplicazione decimale.

$$233 \times 147$$

$$\begin{array}{r} 233 \\ 932 \\ + 1631 \\ 11 \quad \text{riporto} \\ \hline 34251 \end{array}$$

Ed ecco l'esempio di moltiplicazione binaria:

$$1011 \times 1010$$

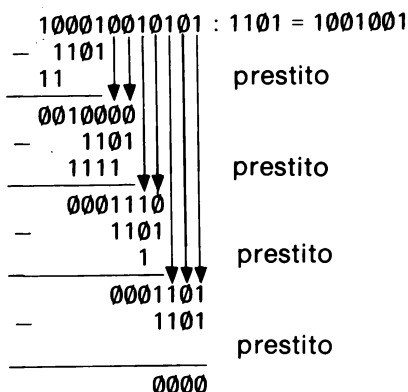
$$\begin{array}{r} 1011 \\ 0000 \\ 1011 \\ + 0000 \\ 1 \quad \text{riporto} \\ \hline 1101110 \end{array}$$

La divisione binaria

La divisione binaria si serve anch'essa del medesimo algoritmo della divisione decimale. Le uniche operazioni necessarie nella divisione binaria sono lo spostamento di una posizione e la sottrazione. Vediamo prima l'esempio di una divisione decimale:

$$\begin{array}{r} 2091_{10} : 17_{10} = 123_{10} \\ - 17 \\ \hline 39 \\ - 34 \\ \hline 51 \\ - 51 \\ \hline 00 \end{array}$$

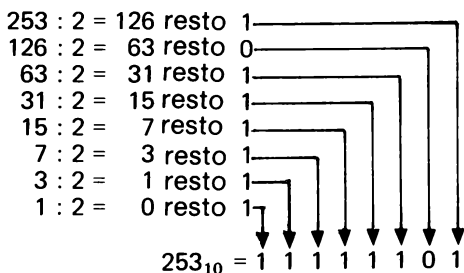
Dato che nella divisione le cifre del dividendo (numero che viene diviso) e del divisore (numero col quale viene diviso il precedente) possono assumere solo i valori 0 e 1, si ha ancora una volta una notevole semplificazione del calcolo.



Conversione di un numero decimale in numero binario

Spesso è necessario convertire un numero decimale in numero binario. Il numero decimale viene allora trasformato in potenze di 2, il che avviene mediante divisione successiva per 2 fin quando è possibile:

$$253_{10} = ?_2$$



Conversione di un numero esadecimale in numero binario

La trasformazione di un numero esadecimale in numero binario procede nell'ordine inverso. Si cerca nella fig. 1 a quale numero binario corrispondono le singole cifre esadecimali:

$$12 \text{ BF} = ?_2$$

Conversione:

1 2 B F

Sec. fig. 1 : 0001 0010 1011 1111

Condensando i gruppi di 4 cifre binarie in un unico numero per semplice giustapposizione si ricava

$$\begin{aligned} 12 \text{ BF} &= \underline{0001} \underline{0010} \underline{1011} \underline{1111}_2 \\ &= 100101011111_2 \end{aligned}$$

Conversione di un numero esadecimale in numero decimale

Per questa trasformazione il numero esadecimale viene considerato come somma di potenze di 16 nel sistema decimale. Consideriamo ad esempio il numero esadecimale ABBA:

$$ABBA = A \times (16_{10})^3 + B \times (16_{10})^2 + B \times (16_{10})^1 + A \times (16_{10})^0$$

Usando la fig. 1 i simboli A e B esadecimali vengono convertiti in numeri decimali. $A = 10_{10}$ e $B = 11_{10}$. Tralasciando per chiarezza gli indici possiamo scrivere:

$$\begin{aligned} ABBA &= A \times 16^3 + B \times 16^2 + B \times 16^1 + A \times 16^0 \\ &= 10 \times 16^3 + 11 \times 16^2 + 11 \times 16^1 + 10 \times 16^0 \\ &= 43962_{10} \end{aligned}$$

Conversione di un numero binario in numero esadecimale

Si voglia convertire il numero 10010010_2 in numero esadecimale. Abbiamo cioè di fronte il seguente problema:

$$10010010_2 = ?_{16}$$

I numeri esadecimali vengono indicati usando un "16" come indice oppure premettendo un segno di dollaro (\$). Come primo passo per risolvere il nostro problema dividiamo il numero binario in gruppi di 4 cifre cominciando da destra:

$$\begin{array}{ccc} 10010010 &= & \underline{1001} \quad \underline{0010} = 92_{16} = \$ 92 \\ && 9 \quad \quad 2 \end{array}$$

Ricaviamo quindi dalla fig. 1 quale valore esadecimale corrisponde a ciascun gruppo di 4 cifre nel sistema binario. 1001 ha il valore esadecimale \$9 e 0010 il valore \$2. Queste cifre - riunite in un unico numero - forniscono il numero esadecimale \$92 ovvero 92_{16} . Proviamo ora a trasformare il numero binario 111101010_2 in un numero esadecimale. Cominciando da destra suddividiamolo in gruppi di 4 cifre:

$$\begin{array}{ccc} \underline{0001} & 1110 & 1010 = \$ 1EA \\ 1 & E & A \end{array}$$

Nel gruppo di 4 cifre più a sinistra abbiamo complementato le cifre mancanti con degli 0 (sono quelli sottolineati).

Aritmetica dei complementi a 2 (Two's complement arithmetic)

Sinora abbiamo considerato solo numeri positivi. Inoltre non abbiamo tenuto conto che ogni calcolatore impiega una determinata lunghezza di parole. Lo Junior-Computer, come sappiamo, impiega una lunghezza di parola di 8 bit ovvero 1 byte. Esso può quindi trattare numeri da 00000000_2 a 11111111_2 , ovvero $\$00...\FF o ancora $0_{10}...255_{10}$. Quando non è sufficiente questa riserva di numeri si possono formare numeri di lunghezze di parola arbitrarie giustapponendo più bytes uno accanto all'altro. Nel programma naturalmente occorre tenere debito conto di ciò.

I *numeri binari positivi* si possono rappresentare su una scala numerica. La fig. 2 ne mostra un esempio nel quale sono riportati in parte i numeri binari da 00000000 a 11111111 . Per una maggior comprensione si sono anche riportati i valori dei numeri decimali equivalenti. Ad ogni numero binario positivo corrisponde un punto su tale scala.

I *numeri binari negativi* analogamente ai numeri decimali negativi si possono caratterizzare con un segno meno. Dal punto di vista del calcolatore questo tipo di rappresentazione presenta tuttavia difficoltà e risulta non economico in termini di impiego dei circuiti. Nei calcolatori digitali come lo Junior-Computer i numeri negativi vengono rappresentati dal complemento a 2 (Two's complement).

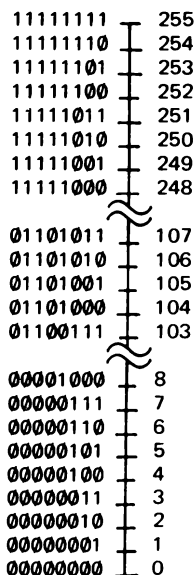


Figura 2. La CPU 6502 è un microprocessore da 8 bit. Con 8 bit si possono rappresentare 256 numeri su una scala numerica



01111111	+	127	\$7F
01111110	+	126	\$7E
01111101	+	125	\$7D
			
00000111	+	7	\$07
00000110	+	6	\$06
00000101	+	5	\$05
00000100	+	4	\$04
00000011	+	3	\$03
00000010	+	2	\$02
00000001	+	1	\$01
00000000	+	0	\$00
11111111	+	-1	\$FF
11111110	+	-2	\$FE
11111101	+	-3	\$FD
11111100	+	-4	\$FC
11111011	+	-5	\$FB
11111010	+	-6	\$FA
11111001	+	-7	\$F9
11111000	+	-8	\$F8
			
10000010	+	-126	\$82
10000001	+	-127	\$81
10000000	+	-128	\$80

Figura 3. Con il complemento a due è possibile rappresentare numeri lunghi sino a 8 bit (di lunghezza arbitraria). Il bit di posizione più alta denota il segno. Se il bit più elevato è 0, si tratta di un numero positivo, se invece è 1, il numero binario è negativo. I numeri binari sono qui rappresentati su una scala numerica accanto ai corrispondenti numeri decimali ed esadecimali. Il segno \$ caratterizza i numeri esadecimali.

Consideriamo ora un pò meglio questo modo di rappresentazione di un numero binario negativo (fig. 3). Cos'è un complemento? Per esso intendiamo la differenza alla cifra massima possibile nella medesima posizione. Il complemento del numero decimale 1984 si può ad esempio calcolare come segue:

$$\begin{aligned}
 A &= 1984_{10} \\
 \bar{A} &= \text{complemento} \\
 \bar{A} &= 9999 - 1984 = 8015
 \end{aligned}$$

Il complemento di 1984_{10} è quindi 8015_{10} . Applichiamo lo stesso principio ad un numero binario:

$$\begin{array}{rcl}
 B &= & 01110101 \\
 \bar{B} &= ? & \text{massimo numero binario di} \\
 & & \text{11111111 equal lunghezza} \\
 & - & 01110101 \text{ numero binario "B"} \\
 & \text{-----} & \text{riporti} \\
 \hline
 & & 10001010 \text{ Complemento ad uno "\bar{B}"}
 \end{array}$$

Esaminando il risultato della sottrazione indicata osserviamo che il complemento consiste in una semplice inversione bit per bit della cifra binaria. Nella sottrazione del numero binario dalla cifra massima in eguale posizione si ha sempre un aggiustamento ad 1. Perciò si suole chiamare "B" come complemento ad 1 del numero binario "B". Se si somma un numero binario al proprio complemento ad 1, il risultato ha sempre la forma 11...11. Per dimostrarlo prendiamo ancora l'esempio precedente:

$$\begin{array}{r}
 01110101 \text{ "B"} \\
 + 10001010 \text{ "}\bar{B}\text{"} \\
 \hline
 11111111 \text{ "B + }\bar{B}\text{"} \\
 + \quad \quad 1 \\
 \hline
 100000000 \text{ "B + }\bar{B} + 1\text{"}
 \end{array}$$

Il risultato della prima addizione " $B + \bar{B}$ " ha dunque effettivamente la struttura 11...11. Se a questo risultato sommiamo ancora il numero 1 otteniamo il numero binario 100000000. Trascurando il riporto nella nona posizione (prima a sinistra) il risultato vale quindi 0.

Per un numero binario lungo 8 bit - come in questo esempio - possiamo allora scrivere:

$$\begin{aligned}
 B + \bar{B} + 1 &= 0 \text{ oppure} \\
 \bar{B} + 1 &= -B
 \end{aligned}$$

$B + 1$ rappresenta quindi il numero binario negativo $-B$ o in altri termini: se si somma ad un qualsiasi numero binario B il proprio complemento a 2 $\bar{B} + 1$ il risultato vale sempre 0. Il seguente esempio dimostra quanto abbiamo detto:

$$\begin{array}{ll}
 01110101 & \text{numero binario "B"} \\
 10001010 & \text{complemento ad uno "}\bar{B}\text{"} \\
 10001011 & \text{complemento a due "}\bar{B} + 1 = -B
 \end{array}$$

Se ora sommiamo $-B$ a $+B$ otteniamo:

$$\begin{array}{r}
 01110101 \text{ +B} \\
 + 10001011 \text{ -B} \\
 \hline
 100000000 \text{ +B + (-B)}
 \end{array}$$

in cui non si deve tener conto del riporto nella nona posizione. Se dunque il complemento a 2 è una rappresentazione dei numeri binari negativi, una sottrazione di numeri binari può anche in questo caso ricondursi ad una addizione. Come è ben noto, la differenza $121_{10} - 68_{10} = 53_{10}$ si può scrivere anche sotto forma di somma: $121_{10} + (-68_{10}) = 53_{10}$. Trasferiamo questo concetto ai numeri binari equivalenti:

$$\begin{array}{r}
 121_{10} = 01111001_2 \\
 68_{10} = 01000100_2 \\
 -68_{10} = 10111011_2 \quad \text{complemento ad 1} \\
 + \quad \quad \quad 1_2 \\
 \hline
 10111100_2 \quad \text{complemento a 2}
 \end{array}$$

$$\begin{array}{r}
 121_{10} \\
 - 68_{10} \\
 \hline
 53_{10}
 \end{array}
 \qquad
 \begin{array}{r}
 01111001_2 \\
 + 10111100_2 \\
 \hline
 100110101_2
 \end{array}$$

Trascurando il riporto della nona posizione dell'addizione binaria il risultato corretto è:

$$53_{10} = 00110101_2.$$

Un altro esempio contribuirà a rendere più comprensibile questa "arida" teoria dei numeri. Due numeri binari di lunghezza differente debbono essere sottratti l'uno dall'altro:

$$01110111 - 0111 = ?$$

Per prima cosa portiamo i due numeri alla stessa lunghezza premettendo degli zeri nelle posizioni più a sinistra del numero più corto. Poi formiamo il complemento a 2 del numero binario che deve essere sottratto.

$$\begin{array}{r}
 00000111_2 \quad \text{numero binario} \\
 11111000_2 \quad \text{complemento ad 1} \\
 + \quad \quad \quad 1_2 \\
 \hline
 11111001_2 \quad \text{complemento a 2}
 \end{array}$$

Ora sommiamo il complemento a 2 del numero binario provvisto del segno meno col primo numero binario positivo:

$$\begin{array}{r}
 01110111_2 \\
 + 11111001_2 \\
 \hline
 101110000_2
 \end{array}$$

Anche qui trascuriamo il riporto nella nona posizione ed otteniamo il risultato corretto 01110000_2 .

I numeri binari negativi si possono raffigurare con i loro complementi a 2 nella scala numerica di fig. 3. La tabella di fig. 4 mostra la corrispondenza fra numeri binari negativi e numeri negativi decimali od esadecimali. E in tal modo abbiamo finalmente concluso il capitolo della "arida" teoria dei numeri e possiamo finalmente dedicarci alla programmazione.

LSD	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
MSD	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127

+

LSD	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
MSD	8	128	127	126	125	124	123	122	121	120	119	118	117	116	115	114	113
9	112	111	110	109	108	107	106	105	104	103	102	101	100	99	98	97	
A	96	95	94	93	92	91	90	89	88	87	86	85	84	83	82	81	
B	80	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	
C	64	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	
D	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	
E	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	
F	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	

-

Figura 4. La scala numerica di Fig. 3 è qui riportata in forma tabellare. Con questa tabella è possibile convertire facilmente un numero decimale in esadecimale. La tabella è divisa in 2 parti: quella superiore serve per i numeri positivi, quella inferiore per i numeri negativi rappresentati dal loro complemento a 2.

Esempio: $59_{10} = \$3B$
 $-92_{10} = \$A4$
 $\$27 = 39_{10}$
 $\$F8 = -8_{10}$

Poiché nella programmazione è molto importante il calcolo con numeri binari ed esadecimali, per chiudere suggeriamo di eseguire ancora i seguenti esercizi.

1. Qual'è la scrittura esadecimale di $00101111?$
2. Qual'è la scrittura esadecimale di $11111?$
3. Qual'è la scrittura esadecimale di $10100111?$
4. Qual'è la scrittura esadecimale di $110101010?$
5. Qual'è la scrittura esadecimale di $010?$
6. Qual'è la scrittura esadecimale di $001011?$
7. Quale numero binario corrisponde al numero esadecimale $132?$
8. Quale numero binario corrisponde al numero esadecimale $A014?$
9. Quale numero binario corrisponde al numero esadecimale $0356?$
10. Quale numero binario corrisponde al numero esadecimale $A561?$
11. Quale numero binario corrisponde al numero esadecimale $ABBA?$
12. Quanto vale $01001111 + 11000111?$
13. Quanto vale $1110011 + 1111111?$
14. Quanto vale $11111111 + 1?$
15. Quanto vale $11110000 + 1111?$
16. Quanto vale $10101010 + 1010101?$
17. Quanto vale $01110100 - 1101?$
18. Quanto vale $11110000 - 1111?$
19. Quanto vale $10111000 - 10000001?$
20. Quanto vale $10101111 - 10101111?$
21. Quanto vale $100 - 11111011?$
22. Quanto vale $11110001 \times 01111?$
23. Quanto vale $101 \times 11111111?$

24. Quanto vale $1010 \times 1010?$
25. Quanto vale $11 \times 11111111?$
26. Quanto vale $4 \times ABBA?$
27. Quanto vale $10000000101 : 111?$
28. Quanto vale $110100000000 : 1101?$
29. Quanto vale $10011011110110010 : 1001110?$
30. Quanto vale $\$B9A0 : \$0B?$

Soluzioni

1. 2F
2. 1F
3. A7
4. 1AA
5. 2
6. 0B
7. 000100110010
8. 1010000000010100
9. 0000001101010110
10. 1010010101100001
11. 1010101110111010
12. 100010110
13. 101110010
14. 100000000
15. 11111111
16. 11111111
17. 01100111
18. 11100001
19. 00110111
20. 0
21. 100001001 (-247 nel complemento a 2 di 9 bit)
22. 111000011111
23. 100111111011
24. 1100100
25. 1011111101
26. $101010111011101000 = \$2AEE8$
27. 10010011
28. 100000000
29. 1111111111
30. $1000011100000 = \$10E0$

Preparazione al capitolo successivo

Nel progettare un programma il programmatore usualmente ricorre ad una Flow Chart (diagramma di flusso). Esso si può considerare lo "schema circuitale" di un programma. La fig. 5 mostra i simboli principali di un diagramma di flusso. I punti di "inizio" e "fine" di un programma si indicano anche come Label. Essi possono tuttavia comparire anche in altri punti arbitrari del programma come sovrascritte.

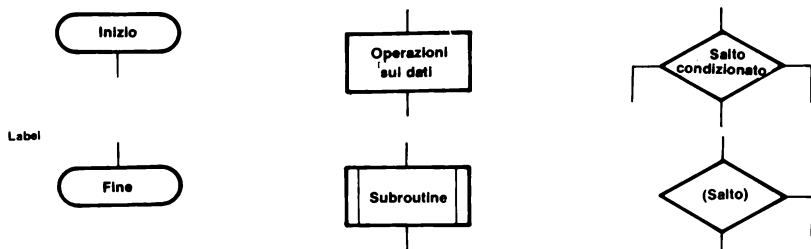
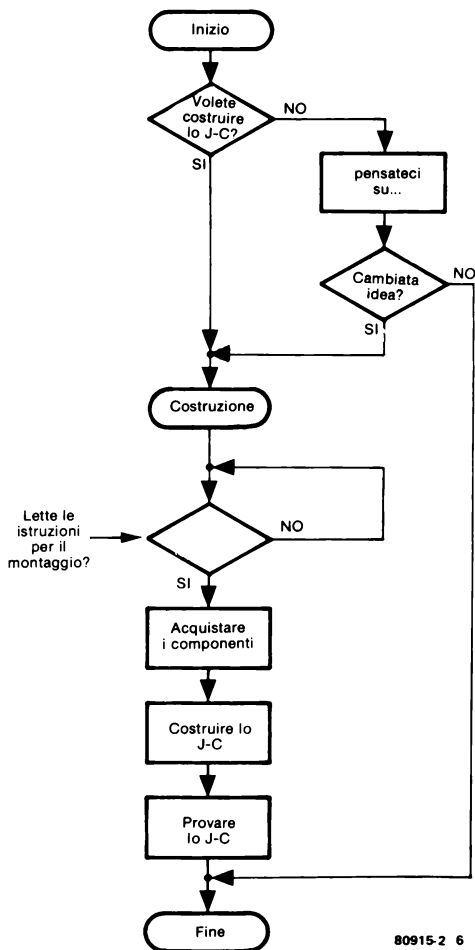
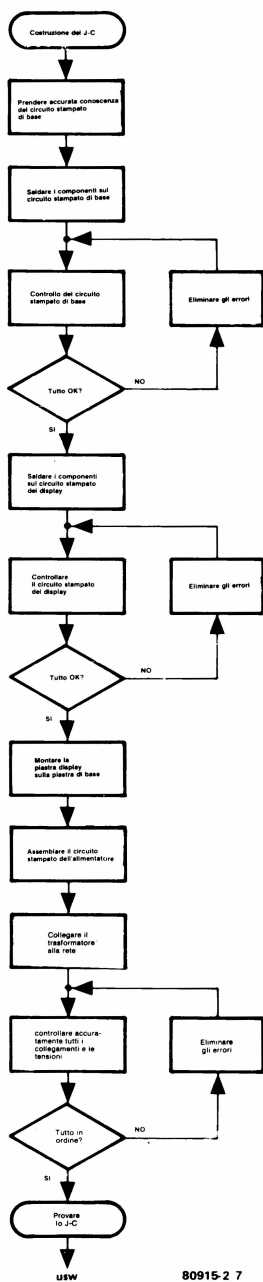


Figura 5. Simboli impiegati nei diagrammi di flusso. Costituiscono le strutture di base di una Flow Chart.



80915-2 6

Figura 6. Il diagramma di flusso globale o grezzo della costruzione dello Junior-Computer.



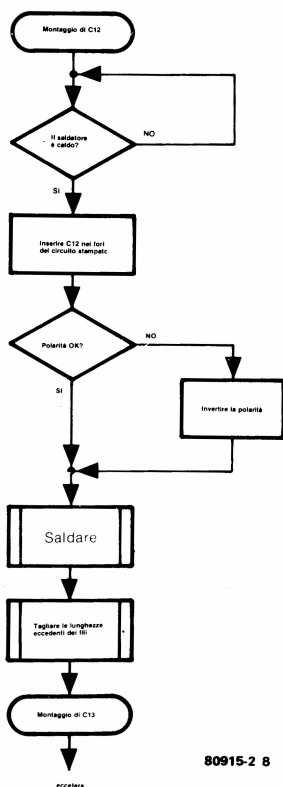
80915-2 7

Figura 7. Il diagramma di flusso particolareggiato della costruzione dello Junior-Computer. Esso è stato ricavato dalla Flow Chart globale di fig. 6.

Vi sono inoltre i rombi che sono i simboli di una diramazione del programma. In funzione di una o più condizioni il programma subisce delle deviazioni. Le operazioni sui dati che devono venire effettuate dal programma sono riportate in rettangoli. Le Subroutine sono anch'esse riportate in rettangoli muniti di due linee verticali parallele sui lati esterni.

Con questi pochi simboli è già possibile tracciare un diagramma di flusso. La fig. 6 ad esempio illustra il diagramma di flusso globale o grezzo della costruzione dello Junior-Computer.

Tale diagramma di flusso di base costituisce il punto di partenza di qualsiasi programma. La Flow Chart dettagliata per la costruzione dello Junior-Computer è riportata in fig. 7. Anche questo programma può essere steso in ulteriore dettaglio: ad esempio la fig. 8 mostra come un singolo componente (C12) viene montato sul circuito stampato e ivi saldato. Il precedente con cui si passa da una Flow Chart grezza ad una dettagliata verrà da noi costantemente applicato nel capitolo 3.



80915-2 8

Figura 8. Flow Chart dettagliata del montaggio e saldatura del condensatore C12 sul circuito stampato di base dello Junior-Computer.

Programmare

Dopo aver costruito lo Junior-Computer ed esserci impadroniti nel 2° capitolo di sufficienti informazioni di base, verrà ora illustrato come si può programmare lo Junior-Computer. Come si scrive un programma, come reagisce il computer a questo programma e quali sono gli impieghi dello Junior-Computer? In questo capitolo risponderemo esaurientemente a queste domande.

Ed eccolo ora davanti a noi lo Junior-Computer con il suo display, i suoi 23 tasti e due interruttori sulla piastra di base. La fig. 1a ci mostra come esso ci si presenta. Dopo aver inserito la tensione di rete e premuto il tasto RST i 6 display si illuminano. Essi ci mostrano inizialmente cifre esadecimali casuali. La pressione del tasto RST fa sì che il computer si colleghi con il programma Monitor nella EPROM. Grazie al Monitor si può colloquiare col computer tramite la tastiera. Perciò lo Junior-Computer verifica periodicamente se qualche tasto della tastiera è stato premuto. Se si tratta dei tasti 0...F, nel display compare il simbolo corrispondente al tasto attivato.

I 4 display sulla sinistra mostrano con cifre esadecimali gli indirizzi. Sono possibili tutti gli indirizzi da 0000 a FFFF. I due display sulla destra indicano il contenuto della posizione di memoria che è stata richiamata dall'indirizzo che compare sul display.

Supponiamo di voler caricare in determinate celle di memoria della RAM dei dati. Ad esempio vogliamo registrare nella cella di memoria con l'indirizzo 0200 il dato 18, nella successiva A9 e così via. Procediamo allora come segue:

				indirizzo;	dati;	Display	istruzioni
RST				xxxx	xx		
AD				xxxx	xx		
0	2	0	0	0200	xx		
DA				0200	xx		
		1	8	0200	18	0200 1 8	CLC
+		A	9	0201	A9	0201 A 9	LDA #
+		0	3	0202	03	0202 0 3	
+		6	9	0203	69	0203 6 9	ADC #
+		0	7	0204	07	0204 0 7	
+		8	D	0205	8D	0205 8 D	STA—
+		0	0	0206	00	0206 0 0	
+		0	3	0207	03	0207 0 3	
+		4	C	0208	4C	0208 4 C	JMP—
+		0	0	0209	00	0209 0 0	
+		0	3	020A	03	020A 0 3	
AD						020B X X	
1	A	7	A	1A7A	xx	020C X X	
DA				1A7A	xx	1A79 X X	
		0	0	1A7A	00	1A7A 0 0	
+		1	C	1A7B	1C	1A7B 1 C	
						1A7C X X	

Come abbiamo operato nei particolari? Per prima cosa premendo il tasto RST abbiamo richiamato il programma Monitor. Il display indirizzi e dati ci mostra delle cifre esadecimali a caso. Perciò nella riga corrispondente abbiamo indicato con delle "X" (X = don't care = indifferente) il contenuto del display indirizzi e dati. Premendo il tasto AD comunichiamo allo Junior-Computer che intendiamo introdurre tramite i tasti 0...F un indirizzo. Questo indirizzo è 0200. Si azionano nell'ordine i tasti 0, 2 e 0 (due volte) e sui 4 display indirizzo compare l'indirizzo desiderato. Poi premiamo il tasto DA. Con esso comunichiamo al computer che desideriamo deporre dei dati tramite i tasti 0...F all'indirizzo di memoria indicato. Il primo dato da caricare è 18. Dopo aver premuto i tasti 1 e 8 nell'ordine, nella cella di memoria con indirizzo 0200 si trova il dato 18. Quando parliamo di caricare dati in una cella di memoria non usiamo una espressione corretta: in realtà abbiamo sovrascritto nella data cella di memoria i dati vecchi con dei nuovi. Quella cella poteva infatti benissimo non essere vuota! Premendo il tasto +/- l'indirizzo nel display viene incrementato di 1. I dati che introdurremo di seguito verranno quindi depositati al nuovo indirizzo. A questo punto non è necessario premere nuovamente

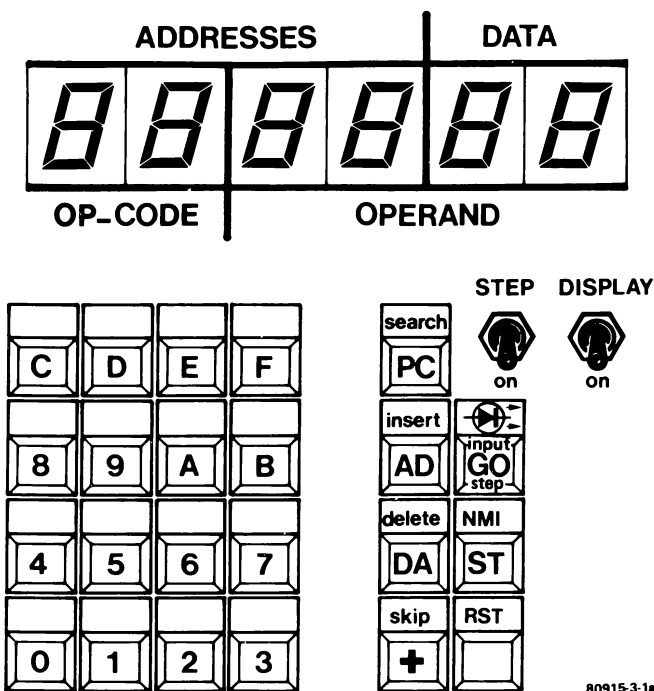


Figura 1a. Così si presentano la tastiera ed il display dello Junior-Computer all'utilizzatore. Oltre ai 16 tasti dati esadecimali, troviamo 7 tasti funzione e 2 interruttori. I tasti funzione svolgono una doppia funzione. Per il programmatore in questo volume, sono importanti le diciture PC, AD, DA, +, GO, ST ed RST. Nel volume Junior-Computer 2 vedremo che le diciture SEARCH, INSERT, DELETE, SKIP e IMPULS interessano per far partire l'Editor. L'Editor fa parte del programma Monitor.

Anche il display svolge una doppia funzione. Qui esso viene impiegato solo come display degli indirizzi e dei dati.

il tasto dati (DA). Lo Junior-Computer tiene infatti memoria se da ultimo è stato premuto il tasto DA oppure quello AD. Se si vuole impostare un nuovo indirizzo (ad esempio 1A7A) occorre prima premere nuovamente il tasto indirizzi per comunicare al computer che i dati che saranno introdotti debbono venire interpretati come indirizzo.

Cosa abbiamo quindi imparato dopo aver depositato a determinati indirizzi un paio di dati? Una quantità di cose! Adesso conosciamo la maggior parte dei tasti della tastiera:

- i tasti 0...F servono all'introduzione di indirizzi e per il caricamento di dati.
- Il tasto AD: con esso comunichiamo allo Junior-Computer che vorremmo introdurre con dei tasti 0...F degli indirizzi.

- Il tasto DA: premendo questo tasto comunichiamo allo Junior-Computer che vorremmo depositare tramite i tasti 0...F dei dati in determinate celle di memoria della RAM.
- Il tasto +/— : la pressione di questo tasto provoca l'incremento di un'unità dell'indirizzo indicato sul display. Non viene modificata la modalità indirizzi oppure dati.
- Il tasto RST: con questo tasto viene richiamato il programma Monitor dello Junior-Computer. Solo dopo la pressione di questo tasto si può colloquiare con lo Junior-Computer tramite la tastiera. Questo tasto pone automaticamente il computer in modalità indirizzi.

Introducendo indirizzi e dati il contenuto dei tasti premuti si sposta da destra verso sinistra nei display. Introducendo ad esempio l'indirizzo 1A7A si ha la successione: dopo 1:XXX1; dopo A: XX1A; dopo 7: X1A7; e alla fine dopo A: 1A7A. Caricando i dati citati nei diversi indirizzi non abbiamo introdotto dati qualsiasi ma abbiamo caricato un vero e proprio piccolo programma: il programma che addiziona due cifre esadecimali. Senza entrare in dettagli mostriamo come questa dozzina di numeri esadecimali formi un programma che lo Junior-Computer è in grado di comprendere.

All'indirizzo 0200 è posizionato il dato 18. Questo è il codice dell'istruzione CLC, Clear Carry. Dopo che la CPU ha eseguito questa istruzione, essa perviene al successivo indirizzo 0201. Qui è depositato A9, il codice per LDA, Load Accumulator immediate. Caricare (Load) che cosa? La risposta sta nel successivo indirizzo 0202. L'accumulatore viene quindi caricato con 03. All'indirizzo 0203 troviamo 69. Questo è il codice per l'istruzione ADC, Add memory to accumulator with Carry. Ciò significa: aggiungi (somma) al contenuto dell'accumulatore il numero che segue all'istruzione ADC. Il contenuto della cella di memoria con indirizzo 0204 è 07. Il contenuto dell'accumulatore è 03. Il contenuto dopo l'esecuzione dell'istruzione ADC è 0A: infatti $03 + 07$ dà il numero esadecimale 0A. Ed adesso siamo curiosi di sapere che cosa succede al contenuto di questo accumulatore.

All'indirizzo 0205 troviamo 8D. Questo è il codice per l'istruzione STA, STore Accumulator in memory. Detto in altre parole: deposita il contenuto dell'accumulatore in una determinata cella di memoria. In quale cella di memoria? L'indirizzo di questa cella si trova nei due successivi indirizzi 0206 e 0207. Nel primo troviamo la parte bassa del byte-indirizzo della cella di memoria nel quale deve essere scritto il contenuto dell'accumulatore, nel secondo la parte alta del byte-indirizzo. La cella di memoria in cui deve essere registrato il contenuto dell'accumulatore ha dunque l'indirizzo 0300. Eseguita l'istruzione STA, la CPU arriva all'indirizzo 0208 dove trova 4C, ossia il codice per JMP (JuMP): salta ad un dato indirizzo. A quale indirizzo deve essere effettuato il salto? Esso si trova nelle due celle di memoria che seguono l'istruzione JMP: 0209 e 020A. In queste due celle si trovano la parte bassa e la parte

alta del byte-indirizzi dell'indirizzo di arrivo del salto: 0300. A questo indirizzo troviamo il risultato dell'addizione precedente: OA. OA può essere un numero esadecimale ma può anche rappresentare un'istruzione! La CPU interpreta il contenuto della cella di memoria 0300 come una istruzione: OA è il codice di ASL-A = Aritmetich Shift Left-Accumulator, ossia sposta il contenuto dell'accumulatore di un bit a sinistra.

Il perché ed il percome questo programma funziona non vogliamo per ora discuterlo. Abbiamo potuto comunque vedere che le istruzioni e i dati che corrispondono a queste istruzioni vengono registrate l'una dopo l'altra (sequenzialmente) nella memoria del computer.

Un'istruzione si trova depositata nella memoria in forma di un numero esadecimale. Un'istruzione dipende a sua volta da dati che descrivano più esattamente l'istruzione stessa. Ricordiamoci dell'istruzione STA precedente: 8D. In quale cella di memoria deve essere caricato il contenuto dell'accumulatore? I dati relativi si trovano nelle due celle di memoria che seguono l'istruzione STA: 0300.

Abbiamo così appreso che i dati e le istruzioni per la macchina sono registrati a pari diritto nella medesima memoria. Non abbiamo cioè memorie separate per le istruzioni e per i dati! L'idea relativa è venuta per la prima volta negli anni 40 ad un certo signor Neumann. Le istruzioni di macchina e i dati ad esse corrispondenti si trovano sequenzialmente e indifferentemente nella memoria del micro-computer. Il computer si muove attraverso il programma secondo il "ciclo di Neumann". Visto dalla CPU, questo ciclo si svolge come segue: - Estrarre il codice dell'istruzione macchina corrente (ad esempio 8D = STA).

- Decodificare l'istruzione macchina (devono venir memorizzati dati: non caricati, manipolati o fatti saltare ad un nuovo indirizzo. L'informazione relativa sta nel codice 8D).

- Estrarre l'operando. Per operando si intendono dati che definiscono meglio l'istruzione macchina. (Nell'esempio l'istruzione macchina è 8D = STA. Dove devono venire memorizzati i dati? L'operando, ossia i due bytes successivi alla citata istruzione macchina, danno l'informazione richiesta: i dati devono venire depositati nella cella di memoria 0300).

- Eseguire l'istruzione. Per istruzione si intende l'istruzione macchina corredata del relativo operando. Mentre viene eseguita l'istruzione, il micro-processore controlla a quale indirizzo si trova la successiva istruzione macchina. Questa istruzione viene a sua volta elaborata come sopra descritto. Così la CPU quindi si muove attraverso il programma secondo il "ciclo di Neumann".

Il modello di programmazione della CPU 6502

Prima di addentrarci nella tecnica di programmazione della CPU

6502 o dello Junior-Computer, diamo uno sguardo alla struttura dei registri interni del micro-processore. Il contenuto dei registri interni della CPU risulta infatti influenzabile dal programmatore tramite il programma. E prima di agire su questi registri occorre naturalmente sapere che cos'è che viene influenzato. Tutti i registri interni della CPU 6502 sono riportati nella fig. 1.

L'accumulatore "A" (nel seguito lo abbrevieremo con Accu) è un registro da 8 bit. Esso viene utilizzato quale stazione intermedia quando si estraggono dati della memoria per successivamente trasferirli in un'altra cella di memoria. La maggior parte delle operazioni che vengono effettuate dal programma passano attraverso l'accumulatore.

Un'altra unità della CPU è la ALU, abbreviazione per Arithmetic Logic Unit. Questa unità lavora in stretto collegamento con l'accumulatore. Dal punto di vista tecnico la ALU possiede due ingressi da 8 bit ed un'uscita da 8 bit. I dati che per effetto delle istruzioni vengono ai due ingressi della ALU vengono elaborati da questa unità e i risultati di questa operazione alla fine si trova nell'Accu. Come dice già il nome stesso ALU, in questa unità vengono eseguite operazioni sia aritmetiche che logiche. Un'operazione aritmetica è ad esempio una addizione o una sottrazione. Un'operazione logica è costituita ad esempio dalle relazioni AND, OR,

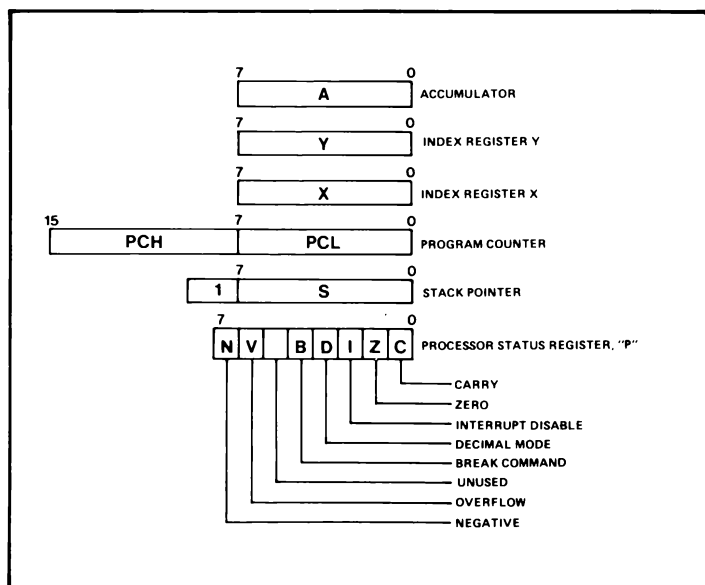


Figura 1b. I registri della CPU del micro-processore 6502. Il contenuto di questi registri è influenzabile dal programma.

EX - OR. Un altro registro interno della CPU è il *Processor Status "P"*. Questo registro indica sotto forma di "Flag" lo stato delle diverse operazioni. I Flag (Flag = bandiera) si possono paragonare ai Flip-flop D. I Flag vengono posti a 1 o a 0 non solo per effetto del risultato delle operazioni, ma anche mediante specifiche istruzioni.

Uno dei Flag del registro *Processor Status* è il Carry Flag "C". Esso viene posto allo stato logico 1 quando ad esempio in una addizione di due cifre si ha un riporto dall'ottavo al nono bit. Se in questa addizione non si ha riporto, il Carry Flag nel registro P viene messo a zero. Un altro Flag è il Flag-N. Questo Flag viene posto ad 1 quando il risultato di una operazione è negativo: in caso diverso esso rimane posto a zero. Lo stesso può dirsi per il Flag-Z nel registro P. Esso indica se il risultato di una operazione vale zero oppure no. Tratteremo più diffusamente il registro *Processor Status* in altra sezione, ma a questo punto la sua funzione dovrebbe già risultare chiara. La CPU 6502 ha delle istruzioni con le quali si possono controllare i Flag nel registro P. Il programmatore può in tal modo agire sul programma modificandolo o correggendolo.

Il contatore di programma, ovvero *Programm Counter "PC"* è un registro da 16 bit. Esso risulta diviso nel registro PCL (L = low) e PCH (H = high). Questo contatore guida il microprocessore attraverso il programma. Ricordiamoci del "ciclo di Neumann"! Il contenuto del contatore di programma è sempre costituito da un indirizzo. Il PC indica per prima cosa una istruzione macchina che si trova in qualche posizione nella memoria. Se a questa istruzione corrispondono uno o più operandi, anche questi ricevono il loro indirizzo dal contatore di programma e sono trasferiti nella CPU tramite il bus-dati.

Altri due registri sono compresi nella CPU 6502: i registri X e Y. Il contenuto di questi registri è influenzabile tramite istruzioni. I registri X e Y si possono considerare, come l'accumulatore, memorie intermedie, ma sono anche predisposti per diversi tipi di indirizzamento. Quando tratteremo l'indirizzamento indicizzato ed indiritto, torneremo in dettaglio su questi registri denominati registri indice.

Anche lo Stack Pointer "S" lo descriveremo meglio in altra parte del libro. Con questo Pointer è possibile registrare nello Stack (catasta), quando si effettuano salti in sotto-programmi, degli indirizzi che debbono essere utilizzati per il ritorno del programma principale. Questo Pointer dischiude inoltre al programmatore molte altre possibilità.

Il repertorio di indirizzamenti e di istruzioni

Il programmatore dello Junior-Computer ha a disposizione 13 diversi tipi di indirizzamento. Questa abbondanza di modalità di indirizzamento costituiscono il punto forte della CPU 6502. L'In-

struction Set, ovvero il gruppo (repertorio) di istruzioni non è invece particolarmente ricco: 56 istruzioni. La combinazione di un numero limitato di istruzioni "potenti" con un gran numero di tipi di indirizzamento provvedono una facilità di programmazione ottimale. La maggior parte di produttori di CPU con altri microprocessori si stanno accodando a questa tendenza del 6502. Perché? La risposta è facile poche istruzioni vuol dire che il programmatore non deve tenere a memoria molte istruzioni macchina. Tutte le istruzioni macchina sono caratterizzate da un codice esadecimale e tre lettere maiuscole. Queste tre lettere maiuscole descrivono in modo stenografico l'istruzione macchina e vengono chiamate "Mnemonics". "Mnemonic" (in italiano mnemonica) significa arte della memoria. Arte della memoria significa qui descrivere a parole in modo stenografico numeri esadecimali che rappresentano istruzioni macchina. Le poche ma potenti istruzioni macchina della CPU 6502, assieme ad una larga gamma di indirizzamenti, possono così essere impiegate in modo ottimale e semplice.

Immediate Addressing Indirizzamento immediato

L'Immediate Addressing si riferisce ai registri interni della CPU. Sono le abbreviazioni mnemoniche a specificare quale registro viene interessato. Le istruzioni di questo tipo di indirizzamento constano di 2 bytes. Il primo byte rappresenta l'istruzione macchina o codice OP, il secondo byte dati qualsiasi. Il carattere immediato di queste istruzioni è dato dal simbolo "Rail Road Crossing: #". Esempi di tali istruzioni sono:

LDA # 7A: Carica nell'Accu 7A

LDX # 3B: Carica nel registro X 3B

LDY # 2F: Carica nel registro Y 2F

Altro esempio di questo tipo di indirizzamento è l'istruzione ADC. La scrittura completa dell'ADC è: $A + M + C \rightarrow A$. Ossia, aggiungi al contenuto dell'accumulatore il contenuto della cella di memoria che segue immediatamente all'istruzione ADC. Al risultato aggiungi inoltre il valore del Flag C. Prima di ogni addizione questo Flag deve essere sempre messo a 0. Ciò può essere ottenuto con l'istruzione CLC (Clear Carry). Vediamo un programma di addizione di questo tipo un pò più da vicino:

CLC : Clear Carry (C = 0)

LDA # 13 : carica 13 nell'accumulatore

ADC # 08 : aggiungi 08

BRK : stop non appena è eseguita l'addizione.

L'istruzione BRK (BRK = BReaK) posta al termine del programma è necessaria per formare il computer dopo l'esecuzione dell'addizione. Come introdurre ora questo programma nello Junior-Computer per poterlo quindi eseguire? Cerchiamo prima nella tabella al termine di questo libro (Instruction Set) i codici istruzio-

ne delle singole istruzioni macchina. Troviamo così: CLC = 18; LDA # = A9; ADC # = 69; BRK = 00. Per indirizzo di inizio scegliamo 0100. Procediamo quindi come segue: (dopo aver attivato il computer, con l'interruttore dei display "ON" e l'interruttore Step "OFF"):

				indirizzo;	dati;	
RST	AD			xxxx	xx	
0	1	0	0	0100	xx	
DA		1	8	0100	18	CLC
+		A	9	0101	A9	LDA #
+		1	3	0102	13	
+		6	9	0103	69	ADC #
+		0	8	0104	08	
+		0	0	0105	00	BRK
AD				0105	00	
1	A	7	E	1A7E	xx	
DA		0	0	1A7E	00	
+		1	C	1A7F	1C	
AD						
0	1	0	0	0100	18	
GO				0107	xx	inizio programma;
AD				0107	xx	
0	0	F	3	00F3	1B	risultato

Ci restano un paio di cose ancora da chiarire. Perché il salto improvviso dall'indirizzo 0105 all'indirizzo 1A7A? E perché il risultato si trova all'indirizzo 00F3? Dopo l'istruzione BRK all'indirizzo 0105 il programma dovrebbe propriamente essere terminato. Quello che segue sono procedure che riguardano il programma Monitor dello Junior-Computer. Dopo eseguita l'istruzione BRK il micro-processore si interessa degli indirizzi 1A7E e 1A7F. In queste due posizioni di memoria è depositato l'indirizzo di partenza (1C00) di una Save Routine. Questa Save Routine è una parte del programma Monitor e memorizza tutti i registri interni della CPU in determinate celle della RAM, quando la CPU incontra un'istruzione BRK. Le specifiche celle di memoria hanno i seguenti indirizzi:

00EF : contenuto del PCL
 00F0 : contenuto del PCH
 00F1 : contenuto del registro P
 00F2 : contenuto dello Stack Pointer "S"
 00F3 : contenuto dell'accumulatore "A"
 00F4 : contenuto del registro Y
 00F5 : contenuto del registro X

Nella cella di memoria con l'indirizzo 00F3 viene quindi depositato il contenuto dell'accumulatore. Quindi in questa cella di memoria troviamo ancora il risultato dell'addizione: 1B. Ma prima di poter leggere il risultato di questa cella di memoria dobbiamo far partire il programma di addizione. Perciò diamo l'indirizzo di partenza 0100 ed inizializziamo il programma con il tasto GO.

Allo stesso modo con cui con lo Junior-Computer possiamo addizionare numeri, possiamo anche eseguire una sottrazione. La scrittura di questa istruzione è SBC che significa A-M-C→A, ovvero: togli dal contenuto dell'Accu il contenuto della cella di memoria che segue immediatamente l'istruzione SBC. Da tale risultato togli ancora il Flag C negato. Qui si impiega il Flag C negato poiché il micro-processore nella sottrazione si serve del complemento a 2 (vedi capitolo 2). Ne segue che prima di eseguire una sottrazione il Flag C deve essere posto a 1. Questo requisito è soddisfatto usando l'istruzione SEC (SEt Carry). Un programma per la sottrazione prende quindi il seguente aspetto:

SEC SEt Carry

LDA 13 carica l'accumulatore con 13

SBC 08 sottrai 08 dall'accumulatore

BRK stop non appena eseguita la sottrazione.

Dalla lista dell'Instruction Set nelle pagine finali del libro determiniamo i codici OP per le istruzioni impiegate: SEC = 38; LDA = A9; SBC = E9; BRK = 00. Per indirizzo di partenza prendiamo 0100 ed introduciamo il programma nello Junior-Computer come segue:

				Indirizzo	Dati	
RST	AD			xxxx	xx	
0	1	0	0	0100	xx	
DA		3	8	0100	38	SEC
+		A	9	0101	A9	LDA #
+		1	3	0102	13	
+		E	9	0103	E9	SBC #
+		0	8	0104	08	
+		0	0	0105	00	BRK
AD				0105	00	} Vedi osservazione
1	A	7	E	1A7E	xx	
DA		0	0	1A7E	00	
+		1	C	1A7F	1C	
AD						
0	1	0	0	0100	38	
GO				0107	xx	inizio programma
AD				0107	xx	
0	0	F	3	00F3	0B	risultato

Osservazioni: il caricamento di dati nelle celle di memoria 1A73 e 1A7F è importante solo nel caso che lo Junior-Computer, prima del caricamento del programma di sottrazione, fosse spento. Nella cella di memoria 00F3 troviamo ancora il risultato della sottrazione: 0B. Infatti 13 (19 decimale) meno 08 (08 decimale) fa 0B (11 decimale).

Sinora abbiamo impiegato solo le istruzioni aritmetiche ADC ed SBC. La CPU 6502 dispone tuttavia di tutta un'altra serie di istruzioni logiche. Con queste istruzioni si possono eseguire relazioni AND, OR ed EX-OR. Per eseguire una relazione OR si impiega l'istruzione ORA. La scrittura estesa di questa istruzione è:

A-V-M→A. Il codice OP di questa istruzione è 09. Con un programma dimostriamo l'uso di questa istruzione:

LDA # AA carica l'accumulatore con AA

ORA # 0F esegui bit per bit una relazione OR

BRK stop quando la relazione OR è stata eseguita.

AA in binario si scrive : 10101010

0F in binario : 00001111

Risultato : 10101111 (AF; posto nell'accumulatore)

La relazione OR viene quindi eseguita sui bit nella medesima posizione. La fig. 2a illustra il modo di procedere impiegando come Hardware delle porte (Gates). Un'altra operazione logica è la relazione AND. L'istruzione che esegue questa istruzione si chiama anch'essa AND. La sua scrittura funzionale è: A-V-M→A. Il codice OP della AND è 29. Mostriamo anche qui come opera questa istruzione con un esempio di programma:

LDA # AA carica l'accumulatore con AA

AND # 0F esegui bit per bit una relazione AND

BRK stop quando la relazione A è eseguita.

AA in binario è : 10101010

0F in binario : 00001111

Risultato : 00001010 (0A; posto nell'Accu)

Anche qui la relazione AND viene eseguita sui bit di uguale posizione. La fig. 2d illustra la soluzione in Hardware con porte AND. La terza operazione logica è la relazione EX-OR (OR esclusivo). La sua scrittura è: A-V-M→A. La sua abbreviazione è EOR col codice OP 49. Anche qui mostriamo l'uso di questa relazione con un programma:

LDA # AA carica l'accumulatore con AA

EOR # 0F esegui bit per bit una relazione EOR

BRK stop quando la relazione EOR è eseguita.

AA in binario è : 10101010

0F in binario : 00001111

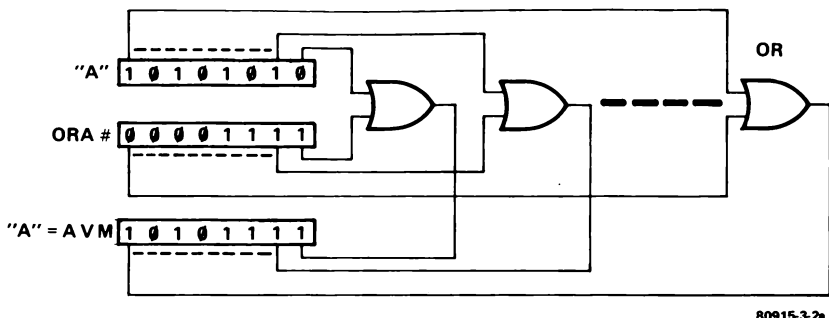
Risultato : 10100101 (A5; posto nell'Accu)

Una relazione OR viene anch'essa eseguita applicandola ai bit nella stessa posizione. Questi tre esempi vengono ora introdotti nello Junior-Computer. I tasti vanno premuti nel seguente ordine:

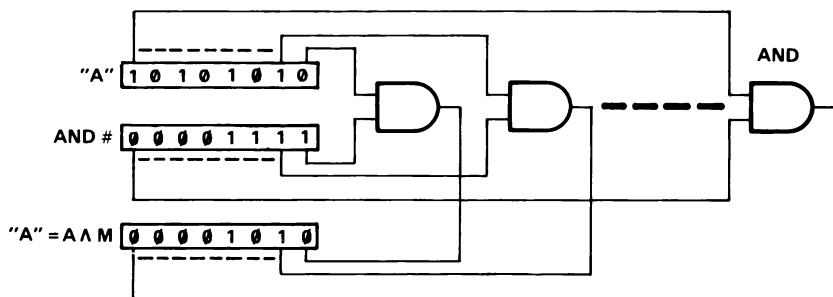
RST	AD			xxxx	xx		
0	1	0	0	0100	xx		
DA		A	9	0100	A9	LDA #	Inizio 1
+		A	A	0101	AA		
+		0	9	0102	09	ORA #	
+		0	F	0103	0F		
+		0	0	0104	00	BRK	Fine 1
+		A	9	0105	A9	LDA #	Inizio 2
+		A	A	0106	AA		
+		2	9	0107	29	AND #	
+		0	F	0108	0F		
+		0	0	0109	00	BRK	Fine 2
+		A	9	010A	A9	LDA #	Inizio 3
+		A	A	010B	AA		
+		4	9	010C	49	EOR #	
+		0	F	010D	0F		
+		0	0	010E	00	BRK	Fine 3
AD							
0	1	0	0	0100	A9	Indirizzo di start 1	
GO				0106	AA	Stop 1	
AD							
0	0	F	3	00F3	AF	Risultato 1	
AD							
0	1	0	5	0105	A9	Indirizzo di start 2	
GO				010B	AA	Stop 2	
AD							
0	0	F	3	00F3	0A	Risultato 2	
AD							
0	1	0	A	010A	A9	Indirizzo di start 3	
GO				0110	xx	Stop 3	
AD							
0	0	F	3	00F3	A5	Risultato 3	

Dato che ognuno dei tre programmi termina con l'istruzione BRK, li possiamo scrivere uno dietro l'altro. Se deve tuttavia prestare attenzione che per inizializzare ciascun tratto di programma occorre inserire con il tasto GO il corretto indirizzo di partenza (start), dato che l'istruzione BRK ha bisogno di uno "spazio di frenata" di due indirizzi. Ciò dipende dalla struttura interna di Hardware della CPU 6502.

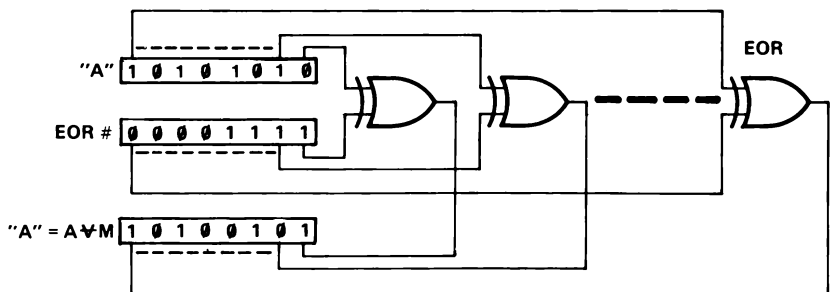
Le istruzioni logiche che abbiamo appena trattato hanno un importante significato nei programmi. Esse si rendono necessarie quando si devono influenzare determinati bit in un dato byte, ad



80915-3-2a



80915-3-2b



80915-3-2c

Figura 2. L'esecuzione logica delle istruzioni ORA, AND e EOR è realizzata simbolicamente con porte (gates) di Hardware.

esempio porre a 1 o porre a 0 oppure invertire determinati bit. In sommario si può dire:

istruzione AND : con questa istruzione si pongono a 0 i bit

istruzione OR : con essa si pongono a 1 i singoli bit

istruzione EOR : con questa si invertono i singoli bit.

Tanto basti per le istruzioni relative all'Immediate Addressing. Abbiamo imparato così 8 istruzioni: LDA #, LDX #, LDY #, ADC #, SBC #, ORA #, AND #, EOR #. Ed inoltre anche le istruzioni CLC,

SEC, BRK, che tratteremo meglio in relazione ad un altro tipo di indirizzamento, lo Implied Addressing.

Absolute Addressing **Indirizzamento assoluto o diretto**

Con l'Immediate Addressing i dati che seguono immediatamente il codice OP vengono caricati in un registro interno della CPU. Tale registro può essere l'accumulatore, il registro X od il registro Y. È tuttavia anche possibile caricare in questi registri dei dati che si trovano depositati ad un determinato indirizzo. Le istruzioni con l'Absolute Addressing sono lunghe 3 bytes e sono costituite come segue:

codice OP, ADL ed ADH. L'ADL e l'ADH sono due bytes indirizzi dai quali si possono estrarre dati o nei quali si possono scrivere i dati. Il programma che segue illustrerà quello che intendiamo dire:

LDA-0100	carica nell'accumulatore il contenuto della cella di memoria 0100
STA-015A	memorizza il contenuto dell'accumulatore nella cella di memoria 015A
LDA-0101	carica nell'accumulatore il contenuto della cella di memoria 0101
STA-015B	memorizza il contenuto dell'accumulatore nella cella di memoria 015B
LDA-0102	carica nell'accumulatore il contenuto della cella di memoria 0102
STA-015C	memorizza il contenuto dell'accumulatore nella cella di memoria 015C
LDA-0103	carica nell'accumulatore il contenuto della cella di memoria 0103
STA-015D	memorizza il contenuto dell'accumulatore nella cella di memoria 015D
LDA-0104	carica nell'accumulatore il contenuto della cella di memoria 0104
STA-015E	memorizza il contenuto dell'accumulatore nella cella di memoria 015E

Da notare: il trattino tra il codice OP e l'indirizzo denota il carattere di indirizzamento assoluto. Cosa esegue questo programma? Il contenuto delle celle di memoria di indirizzi 0100...0105 viene copiato nelle celle di memoria 015A...015E. La stazione intermedia per le operazioni di caricamento delle posizioni di memoria 0100...0105 in un'altra zona di memoria, nei citati indirizzi si trovano ancora gli stessi dati originali. La fig. 3 illustra questo procedimento di copiatura. L'istruzione LDA presente in questo esempio la conosciamo già da prima. L'unica differenza è che il simbolo “#” viene sostituito da un trattino “-”. L'istruzione STA è nuova. Le istruzioni di memorizzazione scrivono il contenuto dell'Accu, del

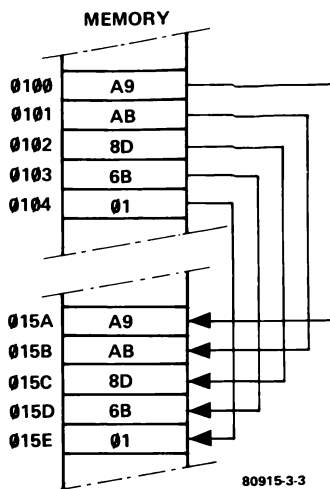


Figura 3. Il programma di copiatura nell'Absolute Addressing funziona così!

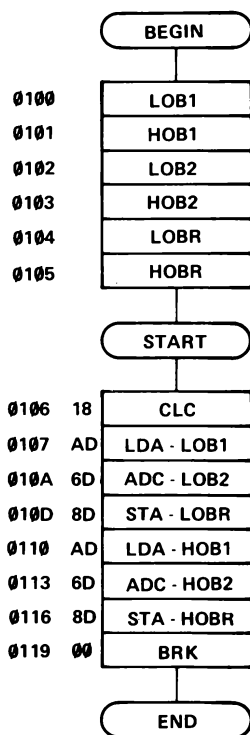


Figura 4. Il programma di flusso (Flow Chart) di un programma per l'addizione di due numeri di 16 bit.

registro X o Y in determinate celle di memoria. La scrittura simbolica relativa è A→M. Come già detto, le istruzioni con l'Absolute Addressing constano di 3 bytes. Il primo byte costituisce il codice operativo. Il secondo byte è la parte bassa del byte indirizzo ADL (L = Low) ed il terzo byte dell'istruzione è la parte alta del byte indirizzo ADH (H = High).

Per prendere meglio confidenza con l'Absolute Addressing, scriviamo un nuovo programma che serve ad addizionare due numeri da 16 bit. Per rendere il programma più chiaro associamo ai numeri dei nomi simbolici, come segue:

1° numero : HOB1, LOB1. Il significato è HOB1 = High Order Byte del 1° numero, LOB1 = Low Order Byte del 1° numero.

2° numero : HOB2, LOB2. HOB2 = High Order Byte del secondo numero, LOB2 = Low Order Byte del 2° numero.

Il risultato è ancora un numero lungo 16 bit, sul quale interviene eventualmente un riporto dalla 16ª alla 17ª posizione. L'eventuale riporto viene indicato mettendo ad 1 il Flag C.

Anche il risultato dell'addizione associamo un nome simbolico: HOBR, LOBR.

Prima di vedere in dettaglio il programma di addizione esaminiamo la struttura (Flow Chart) nella fig. 4. A partire dall'indirizzo 0100 vengono riservate 6 posizioni di memoria per i due numeri da sommare ed il risultato dell'addizione. Le posizioni di memoria relative hanno gli indirizzi 0100...0105. Prima di far partire il programma depositiamo nelle celle di memoria LOB1, HOB1, e LOB2, HOB2 i due numeri da sommare. Il programma scrive poi nelle celle di memoria LOBR ed HOBR il risultato dell'addizione. Il vero e proprio programma di addizione inizia dall'indirizzo 0106 e termina all'indirizzo 0119. Prima di poter eseguire il programma lo dobbiamo introdurre nello Junior-Computer. Supponiamo di voler addizionare mediante il programma i due numeri esadecimali 04EF (1° numero) e 23AB (2° numero). Ciò si realizza come segue:

(STEP: OFF; DISPLAY: ON)

RST	AD			xxxx	XX	
1	A	7	A	1A7A	XX	
DA		0	0	1A7A	00	} preparazione allo STEP (vedi sezione successiva);
+		1	C	1A7B	1C	
++				1A7D	XX	
+		0	0	1A7E	00	} preparazione al BRK;
+		1	C	1A7F	1C	
AD				1A7F	1C	
0	1	0	0	0100	XX	
DA		E	F	0100	EF	LOB1
+		0	4	0101	04	HOB1
+		A	B	0102	AB	LOB2

+		2	3		0103	23	HOB2
+					0104	XX	} posizione di memoria per LOBR o HOBR;
+					0105	XX	
+		1	8	CLC	0106	18	clear Carry-Flag
+		A	D	LDA-	0107	AD	} LOB1 in A
+		0	0		0108	00	
+		0	1		0109	01	
+		6	D	ADC-	010A	6D	} A+LOB2→A conta pure il Carry-Flag;
+		0	2		010B	02	
+		0	1		010C	01	
+		8	D	STA-	010D	8D	} risultato (= LOBR) all'indirizzo 0104;
+		0	4		010E	04	
+		0	1		010F	01	
+		A	D	LDA-	0110	AD	} HOB1 in A
+		0	1		0111	01	
+		0	1		0112	01	
+		6	D	ADC-	0113	6D	} A+HOB2→A (conta pure il Carry-Flag)
+		0	3		0114	03	
+		0	1		0115	01	
+		8	D	STA-	0116	8D	} Risultato (=HOBR) all'indirizzo 0105
+		0	5		0117	05	
+		0	1		0118	01	
+		0	0	BRK	0119	00	fine programma;
AD							
0	1	0	6		0106	18	indirizzo di partenza:
GO					011B	xx	inizio programma;
AD							
0	1	0	4		0104	9A	} risultato: LOBR
					0105	28	

Dopo esserci persuasi che questo programma addiziona i due numeri esadecimali 04EF e 23AB, verifichiamo in dettaglio come il programma si svolge entro il computer. Scriviamo i due numeri in forma binaria:

1° numero : 04EF = 0000 0100 1110 1111

2° numero : 23AB = 0010 0011 1010 1011

←HOB→ ←LOB→

Inizialmente all'indirizzo 0106 poniamo a 0 il Carry Flag e carichiamo il byte "LOB 1" basso nell'Accu. Quindi viene addizionato il byte basso del 2° "LOB2" numero e dopo l'addizione si pone a 1 il C-Flag:

		11101111	LOB1
		10101011	LOB2
	+	111 1111	carry
Carry	1	10011010	LOBR
		←9→←A→	

Il computer memorizza quindi nella cella di memoria 0104 il risultato "LOBR". Quindi si carica nell'Accu il byte alto del 1° numero "HOB1" al quale si sommano il byte alto del 2° numero "HOB2" più Carry della prima addizione. Nel computer si ha la seguente situazione:

		00000100	HOB1
		00100011	HOB2
		1	Carry del LOBR
	+	111	Carry
Carry	0	00101000	HOBR
		2 8	

Dopo la seconda addizione si pone a 0 il C-Flag. Il risultato dell'addizione viene quindi posto nella cella di memoria 0105. Negli indirizzi 0104 e 0105 possiamo leggere il risultato dell'addizione. Oltre a ciò, quando si accende lo Junior-Computer, dobbiamo registrare gli indirizzi 1A7A, 1A7B, nonché 1A7E, 1A7F l'indirizzo di partenza della Save Routine del Monitor. Come è noto questo indirizzo di partenza vale 1C00. Solo allora è possibile eseguire il programma di addizione nello "Step by Step Mode" (di cui trattiamo nella sezione che segue) ed eseguire il comando BRK.

Abbiamo così imparato a conoscere un nuovo tipo di indirizzamento: Absolute Addressing. Il nostro programma di addizione ha impiegato istruzioni che usavano pressoché solo questo tipo di indirizzamento. Le istruzioni con Absolute Addressing sono sempre lunghe 3 bytes: codice operativo - byte indirizzo basso - byte indirizzo alto. Le istruzioni che impiegano la Absolute Addressing sono ben più numerose. Le riassumiamo nella lista seguente:

Istruzioni di caricamento e di lettura:

LDA- Code AD (M → A) load accu with memory
 LDX- Code AE (M → X) load index X with memory
 LDY- Code AC (M → Y) load index Y with memory
 STA- Code 8D (A → M) store accumulator in memory
 STX- Code 8E (X → M) store index X in memory
 STY- Code 8C (Y → M) store index Y in memory

Istruzioni aritmetiche:

ADC- Code 6D (A+M+C → A) add memory to accumulator with carry
 SBC- Code ED (A-M- \bar{C} → A) subtract memory from accumulator with carry

nonché istruzioni di incremento e decremento:

INC- Code EE (M+1 → M) increment memory by one
 DEC- Code CE (M-1 → M) decrement memory by one

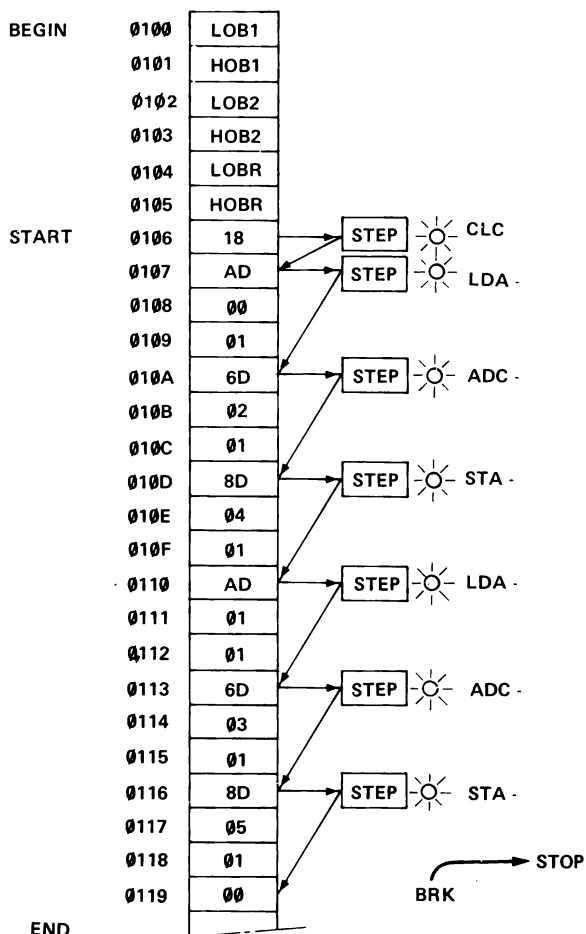


Figura 5. Esecuzione del programma di addizione di fig. 4 nello "Step by Step mode". Dopo la pressione del tasto GO il processore salta da un codice OP ad un altro. Quando il LED contenuto nel tasto GO è acceso, il computer opera in Step by Step mode.

Queste istruzioni aumentano (incrementano) o diminuiscono (decrementano) di una unità il contenuto di una determinata cella di memoria.

Istruzioni logiche:

ORA- Code 0D ($A \vee M \rightarrow A$) OR memory with accumulator

AND- Code 2D ($A \wedge M \rightarrow A$) AND memory with accumulator

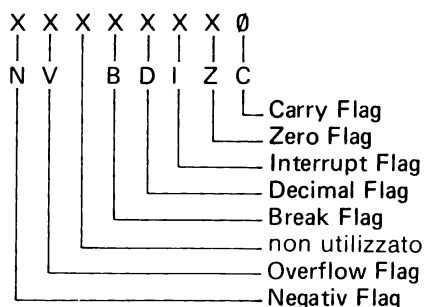
EOR- Code 4D ($A \nabla M \rightarrow A$) Exclusive OR memory with accumulator

Prima di descrivere altre possibilità di indirizzamento della CPU 6502, descriviamo una nuova ed interessante modalità di programmazione dello Junior-Computer, lo "Step by Step Mode".

Step by Step Mode: come percorrere un programma passo a passo

La pressione del tasto GO fa sì che un programma parta ad un determinato indirizzo. Con l'istruzione BRK a sua volta fermiamo il programma dopo la sua esecuzione. L'interruttore Step si trova in posizione OFF. Se portiamo invece questo interruttore in posizione ON, è possibile percorrere il programma passo a passo con il tasto GO. Affinché lo Junior-Computer possa eseguire un programma passo a passo in questo modo, nella cella di memoria 1A7A deve essere caricato il dato 00 e nella cella 1A7B il dato 1C. Al termine di questo capitolo ne chiariremo il perché. In fig. 5 è mostrato come si può eseguire il programma di addizione passo a passo (= step by step mode) con il tasto GO.

A tal scopo dobbiamo portare l'indirizzo 0106, l'indirizzo di partenza del programma di addizione, sul display. Il LED inserito nel tasto GO si illumina perché l'interruttore Step si trova in posizione ON. L'istruzione che troviamo all'indirizzo di partenza del programma di addizione è CLC con il codice OP18. Se ora si preme il tasto GO lo Junior-Computer esegue l'istruzione CLC e sul display compare il codice OP dell'istruzione successiva con il relativo indirizzo: 0107AD = LDA-. Ora vogliamo sapere se il Flag C è effettivamente a 0. Come sappiamo, tutti i registri interni della CPU vengono copiati in determinate celle di memoria. Il Flag C è una parte del registro P che viene conservata nella cella di memoria ad indirizzo 00F1. Premiamo quindi nell'ordine i tasti AD00F1 e sul display dati vediamo comparire il contenuto del registro Processor Status. La struttura di questo registro è stata già considerata quando abbiamo descritto il modello di programmazione della CPU 6502:



Ad ogni bit di questo registro è associato un Flag. I simboli "X" indicano che il valore assunto da questi Flag è indifferente. Qui interessa solo il contenuto del C Flag. Risulta che a questo Flag è associato il bit 0. Dato che il contenuto della cella di memoria 00F1 dopo l'istruzione CLC è un numero pari, il valore del Flag C deve

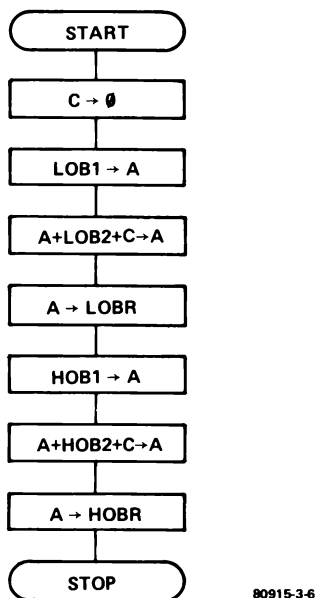


Figura 6. Il diagramma di flusso globale (grezzo) del programma di addizione di fig. 4.

essere 0. Se il Carry non fosse 0 indicherebbe che nella cella di memoria 00F1 è presente un numero dispari. Qui abbiamo pure controllato che l'istruzione CLC è stata effettivamente eseguita. Ora il computer deve eseguire l'istruzione successiva nel programma di addizione. Perciò dobbiamo rientrare nel programma di addizione. Ciò avviene facilmente premendo il tasto PC. PC sta per Program Counter ovvero contatore di programma. Dopo premuto questo tasto sul display dello Junior-Computer compare l'istruzione successiva, con relativo indirizzo, che verrà ora eseguita. Sarebbe anche stato possibile, premendo il tasto AD ed introducendo un indirizzo, portare lo Junior-Computer alla posizione del programma di addizione dalla quale si vuole eseguire la successiva istruzione.

Premendo il tasto PC dunque sul display compare 0107AD. AD è il codice OP dell'istruzione LDA-. Se premiamo ancora il tasto GO sul display compare 010A6D. 6D è codice OP dell'istruzione ADC a cui segue l'istruzione LDA. L'istruzione LDA-LOB1 è già stata eseguita. Ricordiamoci che LOB1 = EF. Nell'accumulatore della CPU deve quindi trovarsi EF dopo l'esecuzione dell'istruzione LDA. Come sappiamo il contenuto dell'Accu è depositato all'indirizzo 00F3. Controlliamo a questo indirizzo e vediamo comparire sul display: 00F3EF. Nell'accumulatore è stato quindi effettivamente caricato EF. Premendo il tasto PC possiamo rientrare nel

programma di addizione ed eseguire l'istruzione successiva. Lo "Step by Step Mode" è uno strumento efficace per controllare i programmi ed identificare eventuali errori. Particolarmente per i principianti, che compiono spesso errori di programmazione nella stesura dei loro programmi, questo tipo di programmazione dello Junior-Computer risulta indispensabile. Nella stesura di un programma si dovrebbe quindi procedere nel modo seguente:

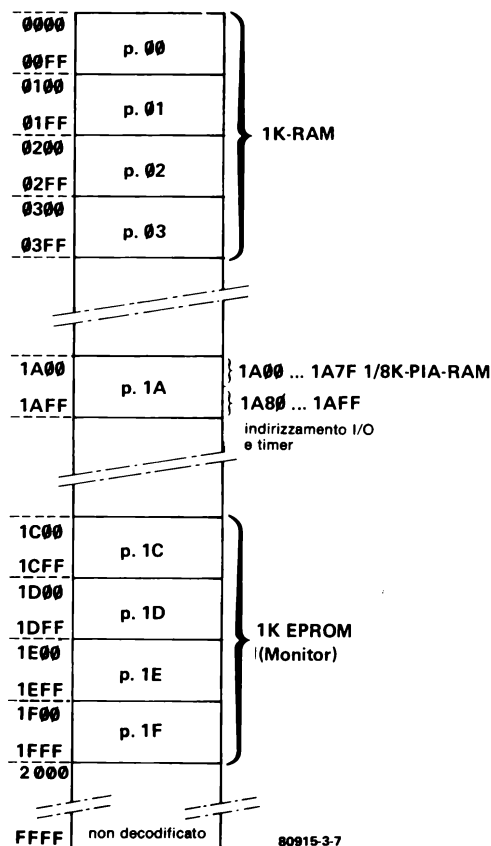
1. Stendere il diagramma di flusso grezzo (Flow Chart) (ad esempio fig. 6).
2. Sviluppare il diagramma di flusso particolareggiato con i codici OP, come ad esempio in fig. 4.
3. Introdurre il programma nel computer mediante la tastiera secondo il diagramma di flusso dettagliato.

Zero Page Addressing

Uno "Zero Page Address" è un indirizzo compreso nella Pagina Zero di memoria dello Junior-Computer. Lo "Zero Page Addressing" è un tipo particolare assoluto constavano di 3 bytes: codice OP - parte bassa del byte indirizzo ADL - parte alta del byte indirizzo ADH. Le istruzioni con Zero Page Addressing differiscono per il fatto che il byte indirizzo alto ADH è sempre 0. Il tipo di indirizzi per istruzioni di questo modo di indirizzamento è dunque: 0000.....00FF. Questi 256 indirizzi formano la Pagina 0 (Zero Page) dello Junior-Computer. I successivi 256 indirizzi a loro volta appartengono alla pagina 1, che segue alla pagina 0. Gli indirizzi in Pagina 1 vanno da 0100 a 01FF. E così via sino alla Pagina FF i cui indirizzi vanno da FF00 a FFFF. Il campo di indirizzi dello Junior-Computer consta dunque di 256 pagine ciascuna delle quali contiene 256 bytes. Rifacciamo il calcolo: verifichiamo infatti che: $256^2 = 65536$ celle di memoria indirizzabili. La struttura di queste pagine è illustrata in fig. 7.

Le pagine da 0 a 3 sono comprese nella RAM della piastra base dello Junior-Computer. In queste 4 pagine vengono depositati i dati quando introduciamo programmi dalla tastiera. La pagina 1A è associata alla PIA e comprende pure una RAM interna da 128 bytes, un timer ed altri registri. Nelle pagine 1C...1F è posto il programma Monitor dello Junior-Computer. Nel modello standard l'intero campo indirizzi non è completamente decodificato (vedi decodificazione degli indirizzi nel 1° capitolo). A causa appunto della decodificazione indirizzi non completa, l'indirizzo più alto dello Junior-Computer risulta 1FFF.

Ma torniamo ai nostri modi di indirizzamento. Dato che la CPU riconosce automaticamente che per un'istruzione della pagina 0 la parte alta del byte indirizzo ADH vale sempre 00 non è necessario introdurre questa parte del byte indirizzo nel computer. Le istruzioni con Page Zero Addressing sono quindi lunghe soltanto 2 bytes: codice OP-ADL. Al codice OP segue cioè immediatamente



80915-3-7

Figura 7. La memoria della CPU 6502 è suddivisa in 256 pagine. Ogni pagina è lunga 256 byte. La parte alta del byte indirizzo fornisce il numero della pagina (00...FF).

la parte bassa del byte indirizzi. Grazie al fatto che la parte alta del byte indirizzo ADH per queste istruzioni manca, ossia la CPU la pone automaticamente uguale a 0, è possibile risparmiare un buon numero di locazioni di memoria: 33% rispetto all'Absolute Addressing. Le istruzioni con Page Zero Addressing sono caratterizzate da una "Z". Un sommario di queste istruzioni è riportato nella tabella seguente.

Istruzioni di caricamento e scrittura:

LDAZ Code A5 (M → A) load accumulator with memory

LDXZ Code A6 (M → X) load index X with memory

LDYZ Code A4 (M → Y) load index with memory

STAZ Code 85 (A → M) store accumulator in memory

STXZ Code 86 (X → M) store index X in memory

STYZ Code 84 (Y → M) store index Y in memory

Istruzioni aritmetiche:

ADCZ Code 65 ($A+M+C \rightarrow A$) add memory to accumulator with carry
SBCZ Code E5 ($A-M-\bar{C} \rightarrow A$) subtract memory from accu w. carry
INCZ Code E6 ($M+1 \rightarrow M$) increment memory by one
DECZ Code C6 ($M-1 \rightarrow M$) decrement memory by one

Istruzioni logiche:

ORAZ Code 05 ($A \vee M \rightarrow A$) OR memory with accumulator
ANDZ Code 25 ($A \wedge M \rightarrow A$) AND memory with accumulator
EORZ Code 45 ($A \nabla M \rightarrow A$) Exclusive OR memory with accumulator

Avremo ancora tempo in altre parti di questo capitolo di prendere esauriente confidenza con queste istruzioni. Dato che lo Zero Page Addressing risulta molto simile allo Absolute Addressing, ci sembra inutile presentare un programma esemplificativo e ci rivolgiamo quindi subito ad un altro tipo di indirizzamento.

Relative Addressing

Il Relative Addressing è la modalità di indirizzamento con salti condizionati. Nella letteratura inglese i salti condizionati vengono definiti istruzioni di "Branch" (= diramazione). La CPU 6502 prevede due tipi di istruzioni Branch: salti condizionati e non condizionati. Nel caso dei salti condizionati il programma prende la diramazione indicata solo quando sono soddisfatte determinate condizioni: ad esempio quando il risultato di una operazione aritmetica risulta zero oppure diverso da zero, quando il Flag C è posto ad 1 o posto a 0, accetera. Si capisce quindi subito che i salti condizionati nel programma sono legati con lo stato dei Flag nel registro Processor Status. Nel 2° capitolo abbiamo già mostrato come vengono indicati i salti condizionati in un diagramma di flusso. Nel caso di salti non condizionati, al contrario di quelli condizionati, il salto viene sempre effettuato. Entrambi i tipi di istruzione di salto saranno ora considerati un pò più attentamente. Incominciamo dalle istruzioni di salto condizionato con Relative Addressing. La CPU 6502 prevede le seguenti istruzioni di salto condizionato:

- 1 BCC codice OP:90 Branch on Carry Clear = salta quando $C=0$
BCS codice OP:B0 Branch on Carry Set = salta quando $C=1$
- 2 BNE codice OP:D0 Branch not Equal = salta quando $Z=0$
BEQ codice OP:F0 Branch on Equal = salta quando $Z=1$
- 3 BPL codice OP:10 Branch on Plus = salta quando $M=0$
BMI codice OP:30 Branch on Minus = salta quando $M=1$
- 4 BVC codice OP:50 Branch on Over Flow Clear = salta quando $V=0$
BVS codice OP:70 Branch on Over Flow Set = salta quando $V=1$

Proviamo ora a scrivere di seguito alcuni programmi con queste istruzioni. Cominciamo dal programma in fig. 8. All'indirizzo 0200 si carica 0A nel registro Y, che viene successivamente decrementato di 1 con una istruzione DEY. Il nuovo contenuto del registro Y

è dunque 09. Quindi la CPU incontra l'istruzione BNE. Con essa si interroga se il registro Y è già 0. Se non è 0 il micro-computer salta a Start 1 e decrementa ancora una volta il registro Y di 1 unità. Il programma cioè continua a decrementare di 1 il registro Y fino a che esso non diventa 0. Quando è raggiunta questa condizione il Flag Z viene posto a 0 e la CPU abbandona il ciclo di programma. La successiva istruzione BRK termina il programma.

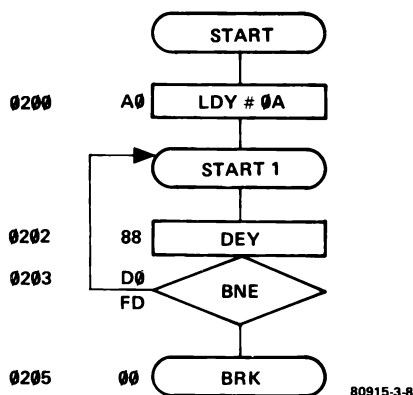


Figura 8. Diagramma di flusso di un programma con istruzione di salto condizionato. L'istruzione di salto condizionato è BNE, il codice OP D0, l'Offset relativo è FD (tre passi all'indietro).

In fig. 8 sotto l'istruzione BNE (D0) è posto il numero esadecimale FD. Questo numero indica di quante celle di memoria il programma deve saltare all'indietro per incontrare l'istruzione DEY. I salti in avanti si intendono positivi e i salti all'indietro negativi.

FD vale 11111101 che è il complemento a 2 del numero -3 . Ciò corrisponde esattamente al numero di passo di cui il computer deve saltare all'indietro nel programma. Perché? Immediatamente dopo l'esecuzione dell'istruzione BNE, il contatore di programma PC si trova all'indirizzo 0205. A questo indirizzo deve essere effettuato un salto all'indietro di -3 per giungere a Start 1. Il numero di passi di programma che la CPU deve far eseguire in corrispondenza ad un salto viene spesso denominato Offset. L'ampiezza dell'Offset varia fra 127 passi all'avanti (00...7F) ed al massimo 128 passi all'indietro ($-1...-128 = FF...80$). Nel calcolo dell'Offset dobbiamo partire dall'indirizzo al quale il contatore di programma punta quando viene eseguita l'istruzione di salto. A questo indirizzo troviamo sempre l'istruzione che segue l'istruzione di salto. Per rendere tutto questo più chiaro scriviamo il programma di fig. 8 come viene introdotto in memoria dello Junior-Computer:

0200	A0	LDY #
0201	0A	
0202	88	DEY
0203	D0	BNE
0204	FD	numero dei salti all'indietro
0205	00	BRK

In questa occasione abbiamo imparato pure l'istruzione DEY. Questa istruzione lunga 1 byte diminuisce il contenuto del registro Y di un'unità. Ritourneremo più avanti in modo più particolareggiato su queste istruzioni da 1 byte.

Calcolo dell'Offset con il Monitor

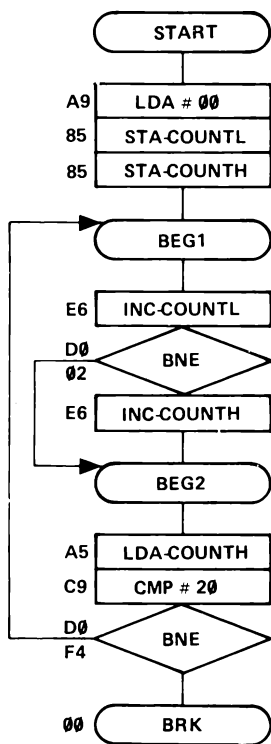
Il calcolo dell'Offset senza altri ausili porta via del tempo e non risulta neppure semplice. Perciò nel programma Monitor dello Junior-Computer è presente una Routine che calcola autonomamente il valore degli Offset. Si deve solo introdurre mediante la tastiera l'indirizzo dal quale parte il salto, quindi l'indirizzo al quale il salto deve arrivare. È il computer stesso allora a calcolare il relativo Offset. Se si provvede a stendere una Flow Chart come in figura 8, gli indirizzi di partenza e di fine dei salti di programma sono noti. Dato che si può saltare solo +127 passi in avanti e —128 passi all'indietro, è sufficiente indicare solo le parti basse dei byte indirizzo degli indirizzi di partenza e di arrivo. La Routine del Monitor per il calcolo dell'Offset si chiama BRANCH e inizia all'indirizzo 1FD5. Introducendo questo indirizzo il programma viene fatto partire con il tasto GO. Sul display compare 000000 e il programma attende l'introduzione degli indirizzi di partenza e di arrivo del salto. In fig. 8 l'istruzione BNE è posta all'indirizzo 0203 ossia l'indirizzo di partenza del salto. La parte bassa del byte indirizzo di 0203 è 03. Da questo indirizzo il salto deve portare a 0202, l'indirizzo di fine salto condizionato. La parte bassa del byte indirizzo 0202 è 02. Questi due bytes indirizzo bassi vengono introdotti nel computer; sul display dati compare il valore dell'Offset del relativo salto:

AD				XXXX XX	
1	F	D	5	1FD5	D8
GO				0000	00
0	3	0	2	0302	FD notare
RST					

Con la routine BRANCH è possibile calcolare il valore anche di diversi Offset l'uno dopo l'altro. Prima di ciascun calcolo si porta a 000000 il display premendo un qualsiasi tasto di comando. Questa rimessa a 0 del display è possibile anche quando nell'introduzione dei dati si è commesso qualche errore di battitura. Premendo il tasto RST si può abbandonare di nuovo il programma e lo Junior-Computer è pronto all'introduzione del programma di fig. 8.

AD					
0	2	0	0	0200	XX
DA		A	0	0200	A0 LDY #
+		0	A	0201	0A
+		8	8	0202	88 DEY
+		D	0	0203	D0 BNE
+		F	D	0204	FD Offset
+		0	0	0205	00 BRK

A prima vista questo programma sembra privo di scopo. Non è così! Questo programma può essere impiegato per generare determinati ritardi di tempo. Come sappiamo il computer per l'esecuzione di un'istruzione impiega un determinato tempo (vedi la tabella Instruction Code al termine del libro). Così è invece possibile con cicli di programma opportuni produrre ritardi di tempo di lunghezza qualsiasi.



COUNTL = 0000
COUNTH = 0001

80915-3-9

Figura 9. Diagramma di flusso di un contatore via Software. Il contenuto effettua il conteggio da 0000 a 2000. Il conteggio termina una volta raggiunto il massimo valore.

Un contatore via Software

Per prendere maggior confidenza con le istruzioni di salto condizionato, scriveremo ora un programma col quale è possibile realizzare un contatore via Software. Lo chiamiamo a questo modo perché per questo contatore non si richiedono né saldature né circuiti integrati. In questo programma incontriamo una nuova istruzione, l'istruzione CMP o di confronto.

Il nostro contatore in Software può contare da 0000 a 2000 (in esadecimale). Quando raggiunge il valore massimo il conto termina. Il diagramma di flusso del contatore è dato in fig. 9. Per rappresentare un numero decimale occorrono due bytes. Per il contatore occorre dunque riservare due celle di memoria. All'indirizzo 0001 è posta la parte alta del byte di conto COUNTH, e a 0000 la parte bassa COUNTL. Il programma pone inizialmente queste due celle del contatore a 0. COUNTL viene quindi incrementato di 1 e con la successiva istruzione BNE si verifica se deve avvenire un riporto a COUNTH. Si verifica un riporto quando il Flag Z è posto a 1. Se il contenuto di COUNTL è diverso da 0 il programma salta a BEG2. L'Accu viene quindi caricato del contenuto di COUNTH e con l'istruzione CMP confrontato con 20. In tal modo si controlla se il conteggio deve essere terminato o proseguito. L'istruzione CMP influenza lo stato di Flag Z nel registro Processor Status. Questo Flag vale 1 quando il contenuto dell'Accu è 20 e la successiva istruzione BRK termina allora il programma. In altri termini: dopo 256 incrementi COUNTL vale 00. Ogni volta si ha in corrispondenza un incremento di COUNTH. COUNTH continua ad essere incrementato fin quando il contenuto di questa cella di memoria è 20.

Per introdurre il nostro programma del contatore in Software nel Junior-Computer, procediamo ora come segue:

AD				xxxx	xx	
1	F	D	5	1FD5	D8	indirizzo di partenza di BRANCH;
GO				0000	00	
1	8	1	C	181C	02	Offset 1. BNE
2	0	1	6	2016	F4	Offset 2. BNE
RST	AD					
0	2	1	0	0210	xx	indirizzo di partenza del contatore;
DA		A	9	0210	A9	LDA #
+		0	0	0211	00	
+		8	5	0212	85	STA-
+		0	0	0213	00	
+		8	5	0214	85	STA-
+		0	1	0215	01	
+		E	6	0216	E6	INC-
+		0	0	0217	00	

12

+	D	0	11	0218	D0	BNE
+	0	2	10	0219	02	Offset
+	E	6	9	021A	E6	INC-
+	0	1	8	021B	01	1
+	A	5	7	021C	A5	
+	0	1	6	021D	01	LDA-
+	C	9	5	021E	C9	CMP #
+	2	0	4	021F	20	operando CMP #:
+	D	0	3	0220	D0	BNE
+	F	4	2	0221	F4	Offset
+	0	0	1	0222	00	BRK
AD				0222	xx	
0	2	1	0	0210	A9	indirizzo di partenza;
GO				0212	xx	inizio del programma

In questo programma sperimentale abbiamo inserito anche gli Offset. Dopo la prima istruzione BNE il programma salta di +2 passi in avanti e dopo la seconda istruzione BNE di -12 passi all'indietro. Gli Offset indicati per i 2 salti sono 02 ed F4.

L'istruzione CMP

Con l'istruzione CMP (CMP = CoMPare) si possono confrontare fra i due dati presenti in memoria dello Junior-Computer. La CMP prevede diversi tipi di indirizzamento. Nel programma di conteggio sopra indicato abbiamo fatto uso due volte dell'Immediate Addressing. CMP # ha il codice OP C9. L'istruzione CMP esegue una normale sottrazione senza tener conto dello stato del Flag C. Lo schema operativo di questa istruzione è: A—M. La CMP influenza solo i Flag nel registro di Processor Status ma non il contenuto dell'Accu. Il risultato della sottrazione condiziona lo stato dei seguenti tre Flag:

- Flag N = ad 1 quando $A < M$; in caso affermativo l'istruzione di salto è: BMI
- Flag Z = ad 1 quando $A = M$; in caso affermativo l'istruzione di salto è: BEQ
- Flag C = ad 1 quando $A \geq M$; in caso affermativo l'istruzione di salto è: BCS

Traducendo in parole queste relazioni matematiche si può anche dire:

- con l'istruzione CMP si può controllare se il contenuto dell'Accu è inferiore al contenuto di una determinata cella di memoria. Una successiva istruzione BMI determina se la condizione " $<$ " è soddisfatta oppure no.
- Con l'istruzione CMP si può controllare se il contenuto dell'Accu è uguale al contenuto di una determinata cella di me-

moria. Una successiva istruzione BEQ determina se la condizione "=" è soddisfatta oppure no.

- c) Con l'istruzione CMP si può controllare se il contenuto dell'Accu è maggiore od eguale al contenuto di una determinata cella di memoria. Una successiva istruzione BCS determina se la condizione " \geq " è soddisfatta oppure no.

Così abbiamo imparato alcune istruzioni di salto condizionato e l'istruzione CMP. Le istruzioni di salto in unione all'istruzione CMP consentono di scrivere i programmi in modo semplice e chiaro. Sull'argomento ora trattato ritorneremo spesso anche nel seguito, ma per intanto passiamo a descrivere le istruzioni di salto non condizionato.

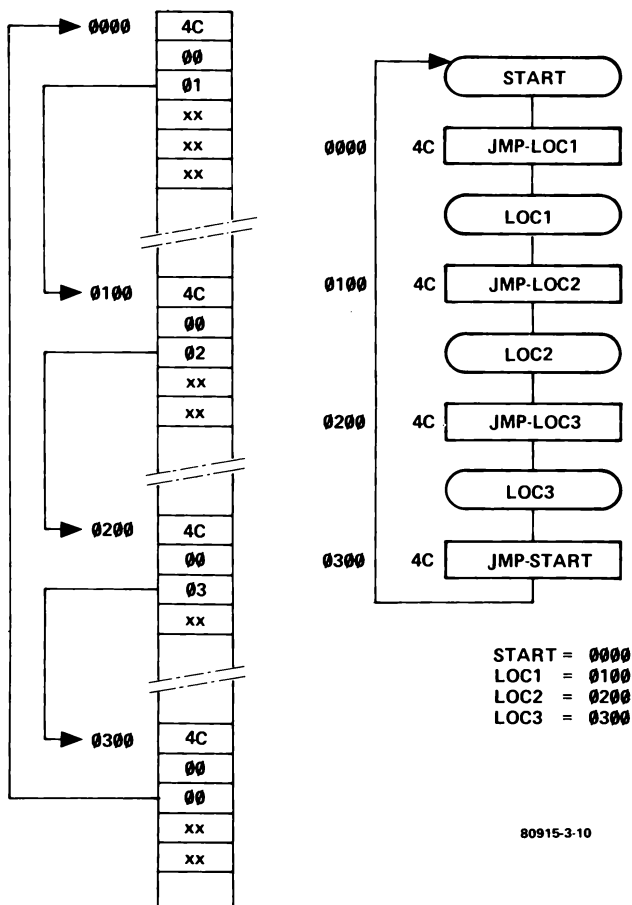


Figura 10. Un ciclo di programma consistente di 4 salti non condizionati (ciclo di programma senza uscita).

Istruzioni di salto non condizionato

Le istruzioni di salto non condizionato, al contrario dei salti condizionati, non dipendono né dallo stato dei Flag nel registro Processor Status né da altre condizioni. Quando il micro-processore incontra una istruzione di salto non condizionato salta senz'altro alla prossima cella di memoria, nella quale è posta una nuova istruzione. La CPU 6502 possiede due istruzioni di salto non condizionato: JMP e JMP-indiretto. L'istruzione di salto diretto od assoluto ha il codice OP 4C e l'istruzione di salto indiretto il codice OP: 6C. Spieghiamo prima l'istruzione di salto diretto od assoluto. Questa istruzione è lunga 3 bytes e fa passare il processore da un indirizzo assoluto ad un altro. Il modo di funzionare di questa istruzione di salto risulta chiaro osservando la fig. 10.

In essa è illustrato il programma che consiste essenzialmente di istruzioni di salto. Il programma salta da un indirizzo all'altro ed infine ritorna al punto di partenza. L'istruzione JMP è costituita come segue: codice OP - ADL - ADH. Il salto a programma indiretto verrà spiegato a proposito di un altro tipo di indirizzamento della CPU 6502, l'Indirect Addressing. Le istruzioni JMP vengono spesso impiegate quando in un microcomputer le "pagine" nella memoria non sono disposte ininterrottamente l'una dopo l'altra (cfr. fig. 7). Con le citate istruzioni è facile saltare da una pagina della memoria ad un'altra. Talvolta in un programma può essere necessario effettuare salti distanti più di +127 o -128 passi. In tal caso il programma salta ad una istruzione JMP situata in prossimità dell'istruzione di salto. In tal modo diventano possibili salti sopra l'intero campo di memoria indirizzabile. Trattando il Monitor di sistema nel 2° volume di questa serie incontreremo spesso tipi di salto che conducono ben oltre l'ampiezza di "Branch" della CPU 6502.

JSR e RTS: salti a sotto-programma e ritorno al programma principale

Supponiamo che uno studente abbia a risolvere un difficile problema di matematica. Per poter risolvere questo problema il suo istruttore lo ha fornito di un "formulario" che descrive punto per punto come l'esercizio matematico deve essere svolto. Ogni volta che lo studente avrà da risolvere un simile problema di matematica tornerà a consultare questo formulario. Il modo di procedere alla soluzione consiste di tre stadi successivi:

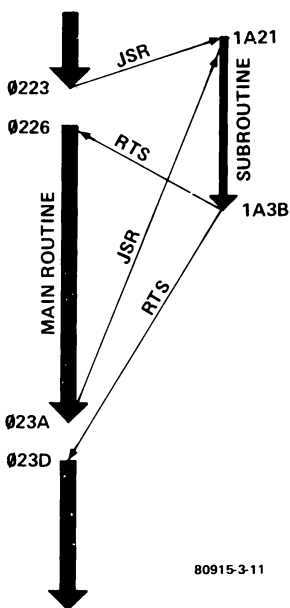
- 1) lo studente deve risolvere un problema di matematica (problema principale)
- 2) per risolverlo egli consulta un formulario per controllare come procedere (problema parziale)
- 3) dopo aver compreso chiaramente il procedimento che porta alla soluzione, lo studente può dedicarsi nuovamente al pro-

blema principale, la risoluzione del suo esercizio. (Ritorno dal problema parziale al problema principale).

Il micro-computer procede esattamente allo stesso modo. Durante l'esecuzione di un programma principale è spesso necessario percorrere più volte una sequenza di medesime istruzioni. Questa serie di istruzioni, che costituisce una parte del programma principale, vengono registrate in qualche parte della memoria e richiamate quando debbono essere impiegate per un dato compito. La procedura di esecuzione del programma per il computer assume lo schema seguente:

- 1) nel programma principale si deve risolvere un particolare problema.
- 2) Si effettua un salto in un sotto-programma (JSR = Jump to Subroutine) nel quale viene risolto una parte del problema (equivalente al formulario dello studente).
- 3) Si ritorna quindi al programma principale (RTS = ReTurn from Subroutine) per proseguire l'elaborazione, disponendo di una soluzione del problema.

Un sotto-programma o Subroutine è dunque una parte del programma principale. Mediante l'istruzione JSR si può richiamare quante volte necessario una stessa Subroutine. Un'istruzione RTS al termine della Subroutine riporta il processore entro il pro-



80915-3-11

Figura 11. Il programma principale effettua 2 salti ad una Subroutine (JSR). L'istruzione RTS al termine della Subroutine riporta il processore indietro al programma principale.

gramma principale. Si comprende dunque che con una Subroutine è possibile risparmiare molto spazio di memoria. Quando spazio, dipende da quanto spesso la Subroutine viene richiamata durante il programma principale.

La fig. 11 illustra il salto ad una Subroutine ed un successivo rientro al programma principale. La Subroutine occupa le posizioni di memoria 1A21...1A3B. L'indirizzo 1A21 contiene il primo codice operativo della Subroutine e all'indirizzo 1A3B è posta l'istruzione RTS con la quale il processore rientra nel programma principale. Questa Subroutine viene richiamata due volte dal programma principale. L'indirizzo 0223 ha il codice OP JSR. I due successivi indirizzi 0224 (ADL) nonché 0225 (ADH) indicano l'indirizzo di partenza della Subroutine (1A21 = ADH, ADL). Affinché il computer, dopo l'esecuzione della Subroutine, sappia a quale indirizzo deve rientrare nel programma principale, tale indirizzo di rientro deve essere registrato in qualche parte della memoria. L'istruzione JSR svolge automaticamente questo compito servendosi dello Stack, nel quale appunto viene depositato l'indirizzo di rientro. Dello Stack tratteremo fra poco in maggior dettaglio.

Dopo il rientro dalla Subroutine il programma principale prosegue dall'indirizzo 0226 nel modo usuale. All'indirizzo 023A il processore incontra ancora un'istruzione JSR, che lo riporta ancora una volta alla stessa Subroutine che aveva già eseguito la prima volta. Al termine della Subroutine è posta ancora l'istruzione RTS, che riporta il processore al punto corretto del programma principale. L'istruzione JSR è lunga 3 bytes ed ha Absolute Addressing. RTS è l'operazione inversa della JSR ed è lunga solo un byte.

Lo Stack ed il suo Pointer

Grazie alle Subroutine è possibile scrivere un programma principale in modo particolarmente chiaro: un numero più o meno grande di istruzioni viene raggruppato in una Subroutine che viene richiamata dal programma principale mediante un'istruzione JSR. Tuttavia è anche possibile richiamare una o più Subroutine dall'interno di un'altra Subroutine. Ciò vuol dire che le Subroutine possono essere "annidate" una entro l'altra. La fig. 12a mostra la struttura di alcune Subroutine annidate una dentro l'altra.

Per ogni salto in una Subroutine si hanno un indirizzo di salto ed un indirizzo di rientro. L'indirizzo di salto e l'indirizzo di partenza della Subroutine. L'indirizzo di rientro riporta il processore alla posizione corretta nel programma principale o nel sottoprogramma. Gli indirizzi di salto possono essere fissati direttamente dal programmatore, oppure, come mostreremo nel secondo volume di questa serie, egli può calcolarsi. Gli indirizzi di rientro vengono stabiliti dal computer mediante lo Stack (catasta). Questo è costituito nella CPU 6502 da una memoria situata in Pagina 1. Quando interviene un salto in una Subroutine, la CPU depone nello Stack

Figura 12a

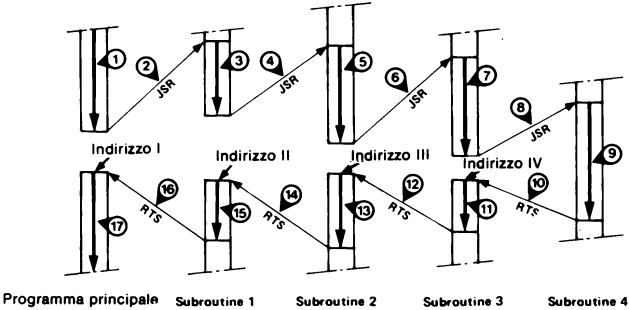
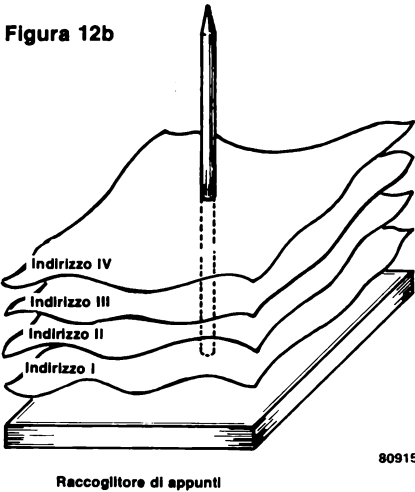


Figura 12b

80915-3-12a

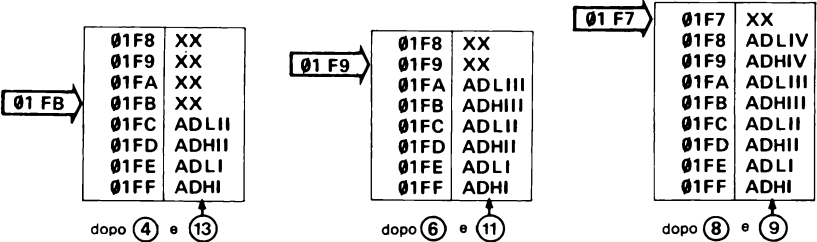


80915-3-12b

Figura 12c

01F8	XX
01F9	XX
01FA	XX
01FB	XX
01FC	XX
01FD	XX
01FE	ADLI
01FF	ADHI

Posizione di partenza: dopo ② e ⑮



80915-3-12c

Figura 12. Qui sono illustrate 4 Subroutine annidate una entro l'altra (a). Ad ogni salto alla Subroutine viene utilizzato lo Stack. Questo registro in Pagina 1 può venir paragonato ad un raccogliatore di appunti (b). La Fig. 12c mostra come ad ogni salto nella Subroutine gli indirizzi di rientro vengono depositati sullo Stack.

gli indirizzi di rientro. Un indirizzo di rientro consta di 2 bytes, il byte di indirizzo alto e quello basso. La gestione dello Stack è assicurata dallo Stackpointer. Questo è un "puntatore" che indica sempre la cella dello Stack in cui deve essere registrato il successivo indirizzo di rientro. Quando si hanno, come nella fig. 12 più Subroutine una dentro l'altra, lo Stack del basso verso l'alto. Ciò vuol dire che quanto più cresce lo Stack, tanto più bassi divengono i valori degli indirizzi in Pagina 1 nei quali sono registrati gli indirizzi di rientro.

In quale ordine vengono elaborate dal processore le Subroutine? Consideriamo ancora la fig. 12a. Vediamo che a un certo punto si lascia il programma principale e viene richiamata la Subroutine 1. Al momento del salto alla Subroutine 1, l'indirizzo di rientro (indirizzo I) viene posto nello Stack. A questo punto il processore interviene sullo Stackpointer per far sì che esso indichi la cella dello Stack nella quale potrà successivamente essere depositato un byte di un altro indirizzo di rientro.

Nel corso della Subroutine 1 viene effettuato un salto alla Subroutine 2. Per effetto di questo nuovo salto in una Subroutine viene nuovamente posto nello Stack il relativo indirizzo di rientro. La cosa si ripete sinché viene infine richiamata la quarta Subroutine. Anche in corrispondenza a questo salto il processore deposita il relativo indirizzo di rientro (indirizzo IV) sullo Stack. Notiamo che sinora sono state richiamate 4 Subroutine senza che la CPU abbia incontrato un'istruzione RTS. Alla fine della Subroutine 4 si trova la prima istruzione RTS che fa rientrare alla Subroutine 3. L'indirizzo di rientro corrispondente (indirizzo IV) viene rintracciato dalla CPU nello Stack. Lo stesso processo si verifica al termine della Subroutine 3. Anche lì troviamo un'istruzione RTS che riconduce alla Subroutine 2. L'indirizzo di ritorno (indirizzo III) viene letto dalla CPU nello Stack. E così via finché al termine l'ultimo indirizzo di rientro (indirizzo I) ci riporta al programma principale. Cosa ci insegna questo esempio? Molto semplice: quando più Subroutine risultano annidate una entro l'altra, per ogni Subroutine occorre depositare sullo Stack il relativo indirizzo di rientro.

Con un'immagine suggestiva possiamo paragonare lo Stack ad uno spillone porta appunti. (Fig 12b). Su un tale aggeggio possiamo immaginare infilzati uno sopra l'altro diversi foglietti con appunti. Su ognuno di questi foglietti è indicato un indirizzo di rientro. L'indirizzo di rientro relativo alla Subroutine richiamata per ultima si trova sopra tutti gli altri sul raccoglitore a spillo. Possiamo descrivere la struttura dello Stack in modo del tutto analogo (fig. 12c). La gestione dello Stack è assicurata dallo Stackpointer. Questo puntatore indica sempre un indirizzo nella pagina sul quale deve essere depositato un byte del successivo indirizzo di rientro. Immaginiamo che inizialmente ① lo Stackpointer indichi l'indirizzo 01FF. Questo è l'ultimo indirizzo della Pagina 1. Segue il richiamo della Subroutine 2. Prima che il processore effettui il sal-

to alla Subroutine, essa conserva l'indirizzo di ritorno ADHI, ADLI sullo Stack. Lo Stackpointer indica adesso l'indirizzo 01FD. Quando il processore lungo la Subroutine 1 ③ incontra una successiva istruzione JSR, esso conserva nuovamente l'indirizzo di ritorno della Subroutine 1 sullo Stack ④. L'indirizzo di ritorno per la Subroutine 1 è ADHII, ADLII. Lo Stackpointer viene nuovamente aggiornato, e mostra ora l'indirizzo 01FB al momento in cui il processore effettua il salto nella Subroutine 2. E così via, sin quando si incontra l'ultima istruzione JSR ⑧. Prima che il processore effettui il salto alla Subroutine 4 esso conserva l'indirizzo di ritorno depositandolo sullo Stack. Lo Stackpointer a questo punto indica l'indirizzo 01F7. Poiché ogni indirizzo di rientro consta di 2 bytes indirizzo, la parte alta e la parte bassa, nell'esempio citato risultano occupate nello Stack 8 celle di memoria.

Al termine della Subroutine 4, la CPU incontra una istruzione RTS che la riconduce nella Subroutine 3 ⑩. A quale indirizzo? I 2 bytes indirizzo dell'indirizzo di rientro sono recuperati dal processore nello Stack (ADHIV, ADLIV). Corrispondentemente lo Stackpointer viene aggiornato e mostra l'indirizzo 01F7. E così via, le varie istruzioni RTS al termine delle Subroutine riportano il processore sino al programma principale. A questo punto lo Stackpointer indica nuovamente la posizione iniziale: 01FF.

Riassumendo si può dunque dire: lo Stack rappresenta un raccoglitore di appunti (fig. 12b) con gli indirizzi di ritorno delle Subroutine. L'indirizzo di rientro della Subroutine verso la quale è stato effettuato l'ultimo salto è situato nella posizione superiore nello Stack, mentre l'indirizzo di rientro della prima Subroutine è la più bassa. Lo Stack (catasta) cresce dunque dal basso verso l'alto e viene "demolito" nell'ordine inverso. Nel gergo anglosassone si usa definire lo Stack del tipo LIFO (LIFO = Last In First Out). Ossia quel che è entrato per ultimo esce per primo.

Il meccanismo di funzionamento dello Stack con il suo Pointer non è forse tanto facile da comprendere. Perciò vogliamo spiegarlo meglio con un esempio pratico. In fig. 13 osserviamo che da un programma principale (Main Routine) si salta ad una Subroutine. All'indirizzo 0216 il processore incontra un'istruzione JSR (codice OP = 20). Salto a quale indirizzo? Agli indirizzi 0217 e 0218 è posto l'indirizzo di partenza della Subroutine: 0300. All'indirizzo 0217 è posto il byte indirizzo basso 00 ed all'indirizzo 0218 la parte alta 03. Come già detto, ad ogni istruzione JSR il processore "salva" l'indirizzo di rientro sullo Stack. La posizione iniziale dello Stackpointer è ancora 01FF. A questo indirizzo viene depositato il byte alto dell'indirizzo di rientro ed all'indirizzo successivo il byte basso (01FF-02, 01F2-18). L'indirizzo di rientro posto nello stack è dunque 0218. Dopo aver registrato l'indirizzo di rientro sullo stack, il processore salta alla Subroutine che inizia all'indirizzo 0300. A questo indirizzo è posta la prima istruzione della Subroutine. All'indirizzo 0306 il processore incontra di nuovo un'istruzione RTS

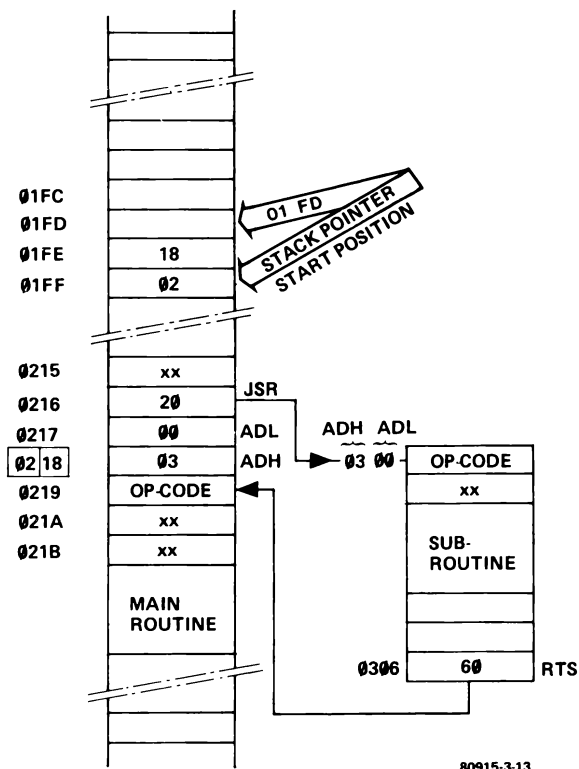


Figura 13. Esempio di programma con salto ad una Subroutine.

(codice OP 60). L'indicazione di dove si deve saltare all'indietro si trova sullo Stack: 0218. In questa posizione non troviamo però alcun codice OP bensì la parte alta del byte indirizzi dell'istruzione JSR. Niente paura!

La CPU incrementa automaticamente di 1 l'indirizzo di rientro sullo Stack ottenendo così il corretto indirizzo di rientro 0219. A questo indirizzo è posto un codice OP a cui segue l'istruzione JSR. Il programma principale prosegue poi nel modo consueto.

Dopo tutta questa teoria, piuttosto arida, è tempo ormai di scrivere alcuni programmi che contengono delle Subroutine. In particolare, considereremo alcune Subroutine del Monitor dello Junior-Computer. Sarà così possibile conoscere meglio il sistema Monitor e scrivere con poca fatica dei programmi dimostrativi.

Vogliamo dunque scrivere un programma che consenta di indicare sul display il contenuto di un tasto che sia stato premuto. Ai singoli tasti vengono associati numeri decimali come mostra la successiva tabella:

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

I valori dei tasti dati 0...F risultano immediati: il valore del tasto è uguale all'istestazione del tasto premuto. Anche ai tasti di funzione AD, PIU, GO e PC vengono associati valori numerici. Avendo così correlato univocamente i valori numerici con i relativi tasti, risulta facile far apparire sul display il valore di un tasto che sia stato premuto.

Prima di procedere alla stesura di questo programma, consideriamo un paio di Subroutine del sistema Monitor. Queste routine del Monitor possono essere inserite senza difficoltà in programmi autonomi senza che il programmatore abbia bisogno di sviluppare i relativi programmi ausiliari. All'indirizzo 1D8E inizia la Subroutine SCANDS. Questa Subroutine provvede a far comparire sul display a 6 cifre dello Junior-Computer il contenuto delle celle di memoria 00F9, 00FA e 00FB, come mostra la fig. 14.

I display a 7 segmenti possono rappresentare le cifre da 0 a F. Queste diverse possibilità si possono configurare con 4 bit. 4 bit costituiscono un mezzo byte. Perciò in una posizione di memoria è possibile depositare due numeri da indicare sul display. Per 6 display sono quindi richieste 3 posizioni di memoria. Come opera la Subroutine SCANDS per portare i numeri presenti nei 3 buffer di display sul display? Per primi, i 4 bit a sinistra nella cella di memoria 00FB vengono convertiti nel codice a 7 segmenti e trasferiti nel display 1 (Di1). Il Di1 viene inserito per circa 500 μ S. Poi tocca ai 4 bit a destra della cella 00FB. Anche questi 4 bit vengono convertiti nel codice a 7 segmenti e trasferiti su Di2. E così via sinché il contenuto di 3 buffer di display, con questo procedimento multiplex, è stato riportato su tutti i display. Alla Subroutine SCANDS segue la Subroutine AK. Anche la AK può essere richiamata con

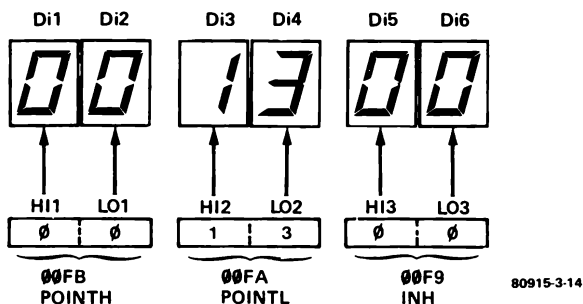


Figura 14. I buffer dati corrispondenti ai display Di1...Di6 in Pagina 0.

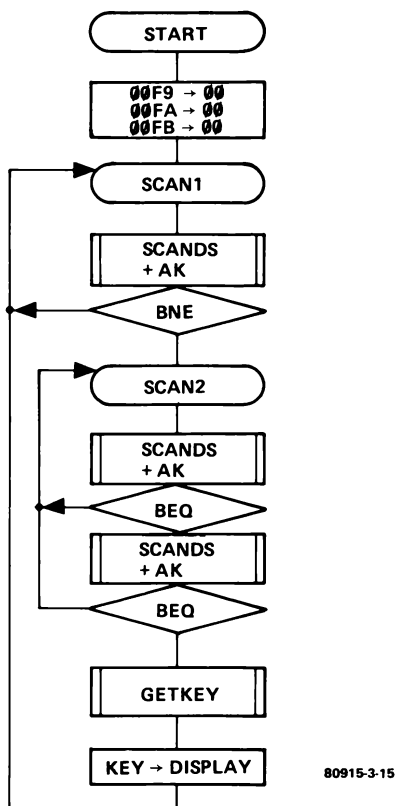


Figura 15. Il diagramma di flusso di un programma che indica sul display il valore di un tasto premuto. Vengono utilizzate Subroutine del programma Monitor.

un'istruzione JSR ed inizia all'indirizzo 1DB1. Mentre SCANDS serve alla gestione del display, la AK serve la tastiera. La Subroutine AK serve solo a stabilire se un tasto è stato premuto oppure no. Ciò vuol dire che AK non riconosce il valore di un tasto che sia stato premuto. Se dopo il rientro dalla Subroutine AK l'accumulatore è uguale a 0, ciò significa che nessun tasto è stato premuto. Se invece l'Accu risulta diverso da 0 significa che un qualche tasto è stato azionato. Con una successiva istruzione BEQ oppure BNE si può allora controllare se il tasto premuto deve venire decodificato o negato. Dopo il riconoscimento di un tasto attivato (cioè quando l'Accu risulta diverso da 0 dopo il rientro dalla AK) deve venir calcolato il suo valore esadecimale. A ciò provvede la Subroutine GETKEY del programma Monitor dello Junior-Computer. GETKEY inizia dall'indirizzo 1FD9 e può essere richiamata con un'istruzione JSR. Dopo il rientro dalla Subroutine GETKEY, nell'ac-

cumulatore del processore si trova il valore esadecimale del tasto premuto.

Il diagramma di flusso globale del programma che fa comparire sul display il valore di un tasto premuto è riportato in fig. 15. Il valore del tasto viene presentato sui due display di mezzo, mentre i due display esterni indicano 00. A tal fine il programma inizialmente carica nei due buffer di display 00F9 e 00FB i numeri esadecimali 00. Successivamente il processore incontra la SCAN 1. A questo punto vengono richiamate la Subroutine SCANDS e la successiva Subroutine AK. La SCANDS fa comparire il contenuto dei buffer di display 00F9, 00FB sul display, mentre la AK verifica se è stato premuto un qualsiasi tasto. Se risulta premuto un tasto, il programma dopo il rientro dall'AK salta nuovamente a SCAN 1. Il display viene nuovamente rinfrescato e con la AK si controlla se risulti ancora premuto un tasto della tastiera.

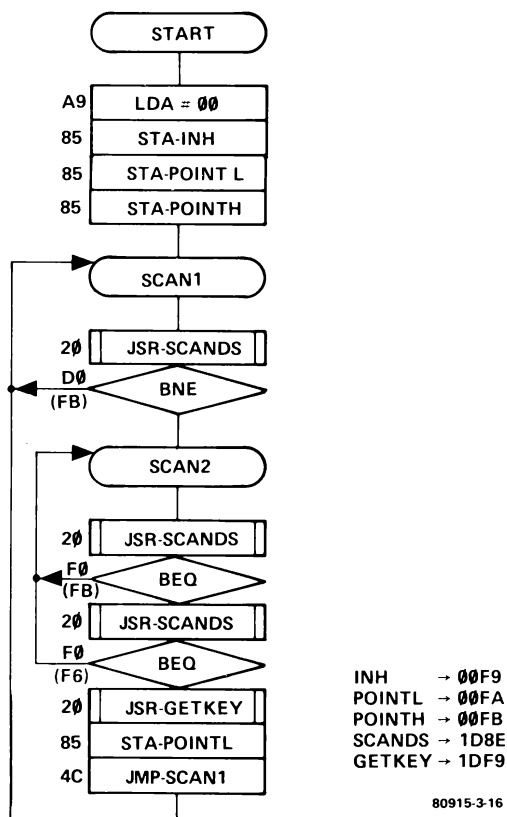


Figura 16. Diagramma di flusso dettagliato di fig. 15.

Solo quando da questo controllo risulta che il tasto è stato rilasciato, il processore salta l'esecuzione dell'istruzione di salto condizionato BNE e giunge a SCAN2. Le Subroutine SCANDS e AK vengono richiamate nuovamente. Il processore si mantiene in questo ciclo di programma, comprendente SCAN1, SCANDS + AK, BEQ, SCAN1, sin quando viene nuovamente premuto un tasto. Quando la Subroutine AK riconosce la presenza di un tasto premuto, la CPU abbandona il ciclo descritto e percorre ancora una volta le Subroutine SCANDS e AK. La successiva istruzione BEQ decide se debba avvenire il salto a SCAN2 oppure no.

A prima vista può sembrare senza senso il fatto di percorrere due volte successive le Subroutine SCANDS + AK. Una ragione c'è, ed è che il processore ha bisogno di alcuni millesecodi di tempo per percorrere SCAN + AK. In questo periodo di tempo gli eventuali rimbalzi del tasto premuto si sono esauriti, ed il processore è quindi in grado di determinare se si tratta di un tasto effettivamente premuto o di un impulso di disturbo. Senza questa Software anti-rimbalzo il computer in casi estremi potrebbe ritenere attivato un tasto che non è stato affatto premuto, o decodificare un tasto sbagliato.

Dopo il riconoscimento di un tasto premuto il processore salta alla Subroutine GETKEY e calcola il valore del tasto che è stato premuto. Al rientro da questa Subroutine il valore corrispondente al tasto posto nell'accumulatore e il processore trasferisce questo valore al buffer di display corrispondente ai due display di mezzo. Dopo aver posto il valore del tasto nel buffer di display, il programma salta nuovamente indietro a SCAN 1 e si ripete interamente il processo già descritto. Il computer si trova cioè in un ciclo senza uscita.

Il diagramma di flusso dettagliato è mostrato in fig. 16. L'indirizzo di partenza del programma è 0200 e gli indirizzi per i buffer di display nonché le Subroutine sono indicate accanto al diagramma di flusso. Pure le Subroutine del Monitor sono chiaramente indicate. Prima di inserire il programma nello Junior-Computer devono essere ancora calcolati gli Offset relativi ai vari salti. La descrizione dettagliata dei tasti da premere ordinatamente per l'inserimento del programma è la seguente:

AD				xxxx	XX	
1	F	D	5	1FD5	D8	
GO				0000	00	
0	B	0	8	0B08	FB	→ notare l'offset;
1	0	0	D	100D	FB	→ notare l'offset
1	5	0	D	150D	F6	→ notare l'offset
RST						
AD						
0	2	0	0	0200	XX	

DA	A	9	0200	A9	LDA #
+	0	0	0201	00	
+	8	5	0202	85	STAZ
+	F	9	0203	F9	INH
+	8	5	0204	85	STAZ
+	F	A	0205	FA	POINTL
+	8	5	0206	85	STAZ
+	F	B	0207	FB	POINTH
+	2	0	0208	20	JSR—
+	8	E	0209	8E	} SCANDS + AK
+	1	D	020A	1D	
+	D	0	020B	D0	BNE
+	F	B	020C	FB	
+	2	0	020D	20	JSR—
+	8	E	020E	8E	} SCANDS + AK
+	1	D	020F	1D	
+	F	0	0210	F0	BEQ
+	F	B	0211	FB	
+	2	0	0212	20	JSR—
+	8	E	0213	8E	} SCANDS + AK
+	1	D	0214	1D	
+	F	0	0215	F0	BEQ
+	F	6	0216	F6	
+	2	0	0217	20	JSR—
+	F	9	0218	F9	} GETKEY
+	1	D	0219	1D	
+	8	5	021A	85	STAZ
+	F	A	021B	FA	POINTL
+	4	C	021C	4C	JMP—
+	0	8	021D	08	} SCAN 1
+	0	2	021E	02	
AD			021E	02	
0	2	0	0	A9	indirizzo di partenza del programma;
GO			0000	00	Il programma lavora; il display visualizza tutti zeri sin quando non viene premuto un tasto Valore del tasto GO (!!!)
GO			0013	00	
6			0006	00	Valore del tasto 6
AD			0010	00	Valore del tasto AD
DA			0011	00	Valore del tasto DA
B			000B	00	Valore del tasto B

In questo modo è possibile presentare sul display il valore di qualsiasi tasto. Quando se ne ha abbastanza:
RST

Dopo introdotto l'indirizzo di Start, si inizializza il programma col tasto GO. Sul display compare una serie di 0. Solo quando si preme nuovamente il tasto GO compare il relativo valore (13) sul display. Mediante il tasto RST si può uscire dal programma.

Implied Addressing

25 istruzioni si riferiscono esclusivamente ai registri interni della CPU ed ai Flag. Con esse ad esempio è possibile scambiare fra di loro registri interni del processore. Sin qui abbiamo imparato istruzioni costituite da un codice OP ed un operando della lunghezza di 1 o 2 bytes. La CPU 6502 prevede anche istruzioni che sono lunghe solo 1 byte e constano solo del codice OP. Abbiamo già incontrato ed imparato alcune di tali istruzioni: CLC, SEC, DEY, RTS. Elenchiamo ora tutte le istruzioni che impiegano lo Implied Addressing:

BRK	Codice 00	BReak
CLC	Codice 18	Clear Carry
CLD	Codice D8	Clear Decimal mode
CLI	Codice 58	Clear Interrupt flag
CLV	Codice B8	Clear oVerflow flag
DEX	Codice CA	DEcrement X-register by one
DEY	Codice 88	DEcrement Y-register by one
INX	Codice E8	INcrement X-register by one
INY	Codice C8	INcrement Y-register by one
NOP	Codice EA	No OPeration* (eine brauchbare Instruktion)
PHA	Codice 48	PusH Accumulator on stack
PHP	Codice 08	PusH Processor-status on stack
PLA	Codice 68	PuL Accumulator from stack
PLP	Codice 28	PuIL Processor-status from stack
RTI	Codice 40	ReTURN from Interrupt
RTS	Codice 60	ReTURN from Subroutine
SEC	Codice 38	SEt Carry flag
SED	Codice F8	SEt Decimal flag
SEI	Codice 78	SEt Interrupt flag
TAX	Codice AA	Transfer Accumulator to X-register
TAY	Codice A8	Transfer Accumulator to Y-register
TSX	Codice BA	Transfer Stackpointer to X-register
TXA	Codice 8A	Transfer X-register to Accumulator
TXS	Codice 9A	Transfer X-register to Stack pointer
TYA	Codice 98	Transfer Y-register to Accumulator

Vediamo ora separatamente alcune di queste istruzioni. SED (SEt Decimal mode) e CLD (CLear Decimal mode). Con le istruzioni ADC ed SDC veniva realizzata l'addizione o la sottrazione del contenuto di una determinata cella di memoria dal contenuto dell'accumulatore. I numeri in tal caso erano espressi nel sistema esadecimale. È tuttavia possibile anche impiegare lo Junior-Computer come macchina di calcolo nel sistema decima-

le. Le relative istruzioni sono SED = Set Decimal Mode e Clear Decimal Mode. Il modo decimale deve essere selezionato solo immediatamente prima dell'esecuzione di operazioni aritmetiche ed al termine deve essere immediatamente rimosso. In caso contrario lo Junior-Computer continua ad operare esclusivamente nel sistema decimale e questo può comportare numerosi errori nel programma. Addizionando due numeri esadecimali si aveva l'attivazione di un Carry quando il risultato era maggiore di 11111111 = FF. Il Carry Flag viene gestito in modo del tutto diverso quando la macchina opera in modo decimale: ogni volta che il risultato è maggiore di 99 si ha un riporto sul nono bit, con la corrispondente messa ad 1 del Carry Flag.

Perciò diamo questo suggerimento: se il calcolo deve procedere nel modo binario all'inizio del programma deve trovarsi una istruzione CLD. Premendo il tasto RST lo Junior-Computer risulta automaticamente indirizzato al modo binario (CLD). Se invece il computer deve operare in modo decimale, all'inizio del programma si deve trovare un'istruzione SED che viene eliminata alla conclusione delle operazioni decimali con un CLD.

NOP (No OPeration)

Può spesso accadere di aver impostato un programma che occupa numerose posizioni di memoria e tuttavia non funziona. Non c'è che un modo per rimediare: eliminare gli errori. Potrà così capitare di accorgersi che questa o quella istruzione sono superflue, o che magari all'inizio del programma manca un'istruzione. Le conseguenze sono amare perché in questi casi non si può far altro che impostare completamente di nuovo il programma istruzione dopo istruzione. È consigliabile perciò inserire qua e là in posizioni importanti del programma alcune istruzioni NOP. Quando il micro-processore incontra un'istruzione NOP in effetti non esegue alcuna operazione. Dato che queste istruzioni NOP possono facilmente essere sostituite "scrivendovi sopra" altre istruzioni, è possibile, senza dover ricominciare tutto da capo, inserire o sostituire istruzioni in un programma. L'istruzione NOP non è dunque così priva di senso come può sembrare a prima vista.

Push-Pull-Transfer

Tutte le istruzioni dell'Implied Addressing il cui simbolo mnemonico cominci con P o T corrispondono al trasporto di dati da o verso i registri A, X, Y, S, o P. Tali istruzioni di Push-Pull-Transfer offrono molte nuove possibilità alla programmazione.

Spieghiamolo sulla base di un esempio. Ammettiamo di aver eseguito un salto ad una Subroutine. Il registro X risulti utilizzato sia nella Subroutine che nel programma principale. A tal fine è necessario conservare il contenuto del registro X, prima del salto

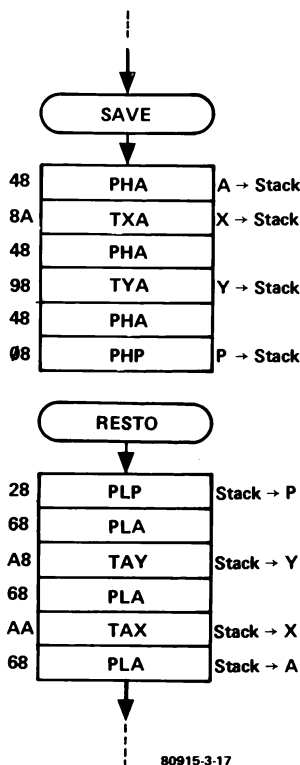
alla Subroutine, sullo Stack. Durante l'esecuzione della Subroutine il contenuto del registro è liberamente disponibile. Al termine della Subroutine riprendiamo questo registro dallo Stack e in tal modo lo riportiamo allo stato iniziale. Tutto ciò è realizzabile ad esempio con la seguente struttura di programma:

TXA copia il contenuto nel registro X nell'accumulatore
 PHA pone il contenuto dell'Accu sullo Stack
 JSR-XXXX il registro X è liberamente disponibile

.

RTS rientro dalla Subroutine al programma principale
 PLA recupera il precedente valore del registro X dallo Stack

TAX copia il contenuto dell'accumulatore nel registro X



80915-3-17

Figura 17. Con questa sequenza di programma è possibile conservare sullo Stack tutti i registri macchina della CPU 6502 e successivamente recuperarli (istruzioni Push-Pull). Dopo essere stati depositi sullo Stack, tutti questi registri sono liberamente accessibili.

	Il contenuto del registro X si sarebbe potuto conservare anche in altro modo:
STX-SAVX	conserva il contenuto del registro X nella cella di memoria SAVX
JSR-XXXX	il registro X è liberamente disponibile

RTS	rientro dalla Subroutine
LDX-SAVX	ricostituisci il precedente stato del registro X

Nel caso dell'indirizzamento assoluto le istruzioni LDX ed STX richiedono più bytes che la procedura Push-Pull. Lo Stack costituisce quindi una ideale memoria intermedia temporanea per dati di tutti i tipi.

Alle volte in alcune Subroutine è necessario poter disporre di tutti i registri interni del processore. Così come in precedenza abbiamo mostrato come il registro X può essere conservato depositandolo sullo stack, anche altri registri della CPU possono essere posti nello Stack e si rendono così disponibili in una Subroutine. La fig. 17 mostra l'andamento di un programma con il quale si possono conservare il contenuto dei registri A, X, Y e P sullo stack riportandoli successivamente nello stato iniziale. La Subroutine SAVE trasferisce i dati sullo Stack (Push) e la Subroutine RESTO li ripristina (Pull). Nel recupero dei registri dallo Stack si deve fare attenzione che ciò che era stato per ultimo sullo Stack deve essere il primo a venir recuperato. Ciò dipende, come abbiamo già chiarito parlando delle Subroutine, dal meccanismo dello Stackpointer. Lo Stackpointer indica sempre l'indirizzo al quale possono essere depositati dati. Quando si inizializza il computer (Reset) occorre sempre predisporre correttamente lo Stackpointer. Dopo il richiamo del Monitor con il tasto RST, lo Stackpointer indica sempre l'indirizzo 01FF. Questo indirizzo è il termine della Pagina 1 nella RAM dello Junior-Computer. Tuttavia è pure possibile posizionare lo Stackpointer ad un altro indirizzo in Pagina 1. Ciò avviene come segue:

LDX81	carica 81 nel registro X
TXS	poni lo Stackpointer all'indirizzo 0181

Lo Stackpointer indica così l'indirizzo 0181. Se sullo Stack vengono ulteriormente depositati indirizzi di rientro od altri dati, lo Stack si decrementa successivamente agli indirizzi 0181, 0180 eccetera. Dato che inizialmente lo Stackpointer indica l'indirizzo 01FF, sullo Stack possono essere depositati 256 dati o 128 indirizzi di rientro. Questi risultano ampiamente sufficienti se si tiene conto che il sistema Monitor dello Junior-Computer richiede solo 13 celle dello Stack.

Mediante la Subroutine STCHK (fig. 18) si può assicurare che il computer emetta un messaggio di errore quando viene superato

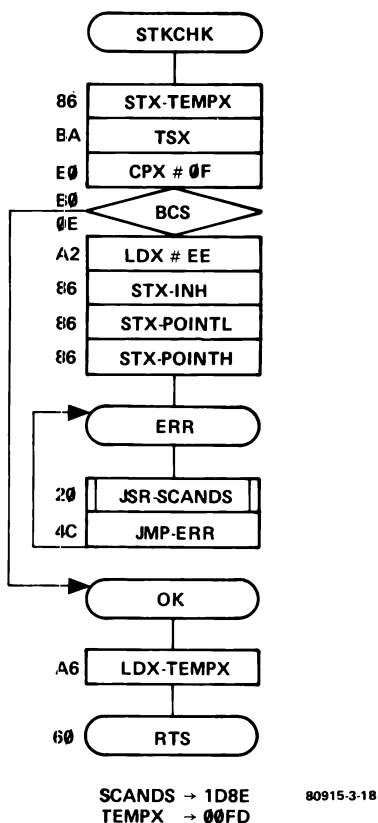


Figura 18. Questo programma fornisce un messaggio di errore sul display quando viene superato lo spazio di memoria disponibile sullo Stack.

lo spazio disponibile di memoria per lo Stack. Non appena lo Stack supera l'indirizzo 010F, sul display compare un messaggio di errore della forma EEEEEEE. Si deve allora premere il tasto RST per far uscire il computer dal ciclo della Subroutine ER. Si è scelto l'indirizzo 010F per comunicare al programmatore con un messaggio di errore che sullo Stack risultano disponibili solo pochi altri spazi di memoria.

Abbiamo illustrato solo le principali istruzioni con Implied Addressing. Altre istruzioni che impiegano lo stesso tipo di indirizzamento verranno considerate in altre sezioni di questo libro.

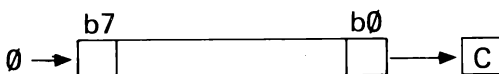
Istruzioni di spostamento e rotazione

In questo capitolo ci occuperemo di un nuovo gruppo di istruzioni, le istruzioni di spostamento e di rotazione della CPU 6502. Queste istruzioni servono soprattutto per poter formare con il

Software dei registri di spostamento di lunghezza qualsiasi. I registri a spostamento (shift-register) vengono utilizzati quando si devono trasferire serialmente tramite un conduttore dei dati a disposizione parallela. Un comune esempio, per chiarire quello che intendiamo, è offerto dal calcolatore tascabile: quando con la tastiera si inserisce un numero in un calcolatore di questo tipo, vediamo successivamente spostarsi le cifre corrispondenti ai tasti premuti da destra verso sinistra nel display. Vediamo dunque come risolvere con lo Junior-Computer il problema dello spostamento semplice con le istruzioni che andiamo a descrivere.

LSR-A - Logic Shift Right

Lo svolgimento dell'operazione descritta da questa istruzione si può rappresentare come segue:



L'istruzione LSR sposta il contenuto dell'accumulatore o di una cella di memoria di 1 bit verso destra. In ogni caso nel bit di ordine superiore, ossia il bit 7, viene sempre inserito uno 0. Mentre il valore del bit inferiore, ossia bit 0, viene spostato nel Carry Flag. Quando l'istruzione LSR viene ripetuta 8 volte successive, il contenuto dell'accumulatore o della cella di memoria è divenuto interamente 0. L'istruzione LSR influenza nel registro Processor Status la condizione dei seguenti Flag:

- Flag N : è sempre messo a 0 (N = 0)
- Flag Z : è posto a 1 quando il risultato dell'operazione di spostamento vale 0. Negli altri casi è sempre messo a 0.
- Flag C : dopo ogni istruzione LSR assume il valore del bit 0 dell'accumulatore o della cella di memoria interessata. Il bit 0 viene quindi sempre spostato nel Carry Flag.

ASL-A-Aritmetich Shift Left

L'andamento operativo di questa istruzione si può rappresentare come segue:

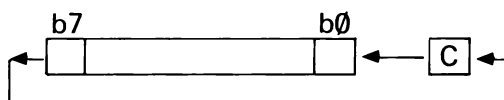


L'istruzione ASL sposta il contenuto dell'accumulatore o di una cella di memoria di 1 bit verso sinistra. In ogni caso il bit superiore, bit 7, viene spostato nel Carry Flag ed il bit 0 assume sempre il valore 0. L'istruzione ASL influisce nel registro Processor Status sulla condizione dei seguenti Flag:

- Flag N : viene posto ad 1 quando dopo l'istruzione ASL il bit 7 è uguale a "1", negli altri casi è sempre posto a 0.
- Flag Z : viene posto ad 1 quando il risultato dell'operazione di spostamento vale 0. In tutti gli altri casi è posto a 0.
- Flag C : dopo ogni istruzione ASL assume il valore del bit 7 nell'accumulatore o nella cella di memoria interessata. Il bit 7 viene quindi sempre spostato nel Carry Flag.

ROL - ROTate Left

L'andamento operativo di questa istruzione si può rappresentare come segue:

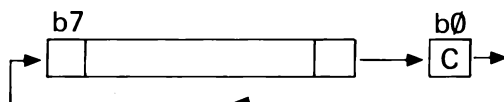


L'istruzione ROL sposta il contenuto dell'accumulatore o di una cella di memoria di 1 bit verso sinistra. In ogni caso il bit superiore, bit 7, viene spostato nel Carry Flag e il contenuto del Carry Flag nel bit 0. L'istruzione ROL influisce nel registro Processor Status sulla condizione dei seguenti Flag:

- Flag N : viene posto ad 1 quando il bit 6 prima dell'operazione di spostamento valeva 0; negli altri casi viene posto a 0.
- Flag Z : viene posto ad 1 quando il risultato dell'operazione di rotazione vale 0; negli altri casi viene posto a 0.
- Flag C : dopo ogni istruzione ROL assume il valore del bit 7.

ROR - Rotate Right

L'andamento operativo di questa operazione si può rappresentare come segue:



L'istruzione ROR sposta il contenuto dell'accumulatore o di una cella di memoria di 1 bit verso destra. In ogni caso il bit inferiore, bit 0, viene spostato nel Carry Flag ed il valore del Carry Flag nel bit 7. L'istruzione ROR influisce nel registro Processor Status sulla condizione dei seguenti Flag:

- Flag N : viene posto ad 1 quando il Carry Flag prima dell'operazione di spostamento si trova allo stato logico 1; negli altri casi viene posto a 0.
- Flag Z : viene posto ad 1 quando il risultato dell'operazione di rotazione vale 0.
- Flag C : dopo ogni istruzione ROR assume il valore del bit 0.

Si noti: con le istruzioni di Shift, come ASL ed LSR, alcuni bit dell'accumulatore o della cella di memoria interessata vanno persi. Con le istruzioni di rotazione, come ROR e ROL, non si perde alcuna informazione almeno sinché l'operazione di spostamento di riferisce ad un unico medesimo byte.

Accumulator Addressing

Introduciamo qui un nuovo tipo di indirizzamento: Accumulator Addressing. Come già dice il nome, questo tipo di indirizzamento si riferisce esclusivamente ad un registro interno della CPU, l'accumulatore. Il repertorio di istruzioni della CPU 6502 prevede solo 4 istruzioni che impiegano l'Accumulator Addressing. Queste sono istruzioni di spostamento e rotazione analoghe a quelle viste nelle sezioni precedenti. Si tratta delle seguenti 4 istruzioni:

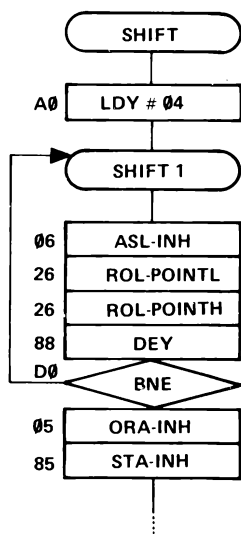
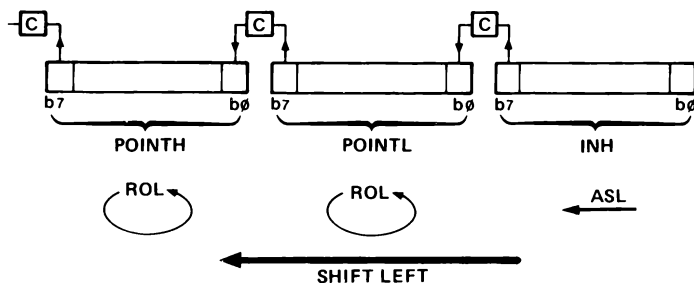
- ASL-A — 0A Arithmetic Shift Left Accumulator
LSR-A — 4A Logical Shift Right Accumulator
ROL-A — 2A ROTate Left Accumulator
ROR-A — 6A ROTate Right Accumulator

Combinazione di istruzioni di spostamento e rotazione

Come si è già ricordato, le istruzioni di spostamento e rotazione vengono per lo più impiegate per trasferire in modo seriale dati disposti in parallelo, o per riempire successivamente con delle cifre il display del calcolatore. Si era detto che la pressione di un tasto porta allo spostamento del valore corrispondente nel display. Vediamo come questo si può realizzare. Le cifre che si presentano sul display a 6 posti dello Junior-Computer, come sappiamo, si trovano nei 3 buffer di display POINTH (00FB), POINTL (00FA) e INH (00F9). Dato che la rappresentazione di un simbolo numerico di un numero esadecimale richiede solo 4 bit, con 1 byte si possono rappresentare 2 simboli esadecimali. Come si procede ora per riempire da destra a sinistra il display con le cifre? Per ogni simbolo numerico che viene spostato nel display devono essere resi li-

beri 4 bit. La fig. 19 dimostra come viene spostato di 4 bit verso sinistra il contenuto dei 3 buffer di display per far posto per un nuovo simbolo numerico in INH. Questo procedimento può descriversi nei particolari come segue:

- 1) Sposta il buffer di display INH di 1 bit a sinistra. L'istruzione corrispondente è ASLZ—06. *Risultato*: nel bit 0 di INH viene inserito uno 0, il bit 7 viene spostato nel Carry Flag.
- 2) Ruota il buffer di display POINTL di 1 bit verso sinistra. L'istruzione relativa è ROLZ—26. *Risultato*: nel bit 0 di POINTL viene inserito il valore del Carry Flag. Questo Carry Flag corrisponde al bit 7 di INH. Ciò equivale quindi a dire che il bit 7 di INH tramite il Carry Flag viene inserito nel bit 0 di POINTL. Contemporaneamente l'istruzione di rotazione sposta il bit 7 di POINTL nel Carry Flag.



80915-3-19

Figura 19. Lo spostamento e la rotazione di dati nei 3 buffer di display. La direzione di spostamento è da destra verso sinistra.

- 3) Ruota il buffer di display POINTH di 1 bit verso sinistra. L'istruzione corrispondente è ancora ROLZ. *Risultato*: nel bit 0 di POINTH viene inserito il valore del Carry Flag. Questo Carry Flag è tuttavia il bit 7 di POINTL. In modo equivalente si può dire che il bit 7 di POINTL viene spostato tramite il Carry Flag nel bit 0 di POINTH. Contemporaneamente l'istruzione di rotazione sposta il bit 7 di POINTH nel Carry Flag. Dato che oltre POINTH non è prevista alcuna cella di memoria, nella quale possa essere spostato tramite il Carry Flag il bit 7 di POINTH, il bit superiore (bit 7) in questione va perso. Non è poi in gran male dato che abbiamo a disposizione solo 6 display.
- 4) Ripeti le operazioni di spostamento e rotazione in tutto 4 volte. Il *risultato* di tutto ciò è:
 - I 4 bit alti di INH vengono spostati tramite il Carry Flag nei 4 bit bassi di POINTL.
 - I 4 bit alti di POINTL vengono spostati tramite il Carry Flag nei 4 bit bassi di POINTH.
 - I 4 bit alti di POINTH vengono successivamente trasferiti nel Carry Flag e persi.
 - Nei 4 bit bassi di INH a seguito dell'istruzione ASL si sono successivamente inseriti degli 0. Sopra a questi 4 zeri si può ora "ordinare" il nuovo numero esadecimale.

La sequenza di programma che effettua gli spostamenti, le rotazioni e l'ordinamento è anch'essa riportata in fig. 19. Con le conoscenze ora acquisite siamo in grado di programmare lo Junior-Computer come un piccolo calcolatore. Tutto quello che ci serve è già stato illustrato nei vari esempi di programmazione:

- Addizionare o sottrarre tramite le istruzioni aritmetiche della CPU 6502.
- L'identificazione di tasto premuto, con le Subroutine AK e GETKEY.
- Separare la scansione del display, con la SCANDS.
- Lo spostamento di dati o cifre nei buffer di display dello Junior-Computer.

Il programma esemplificativo che segue non può dunque incutere più alcun timore!

Lo Junior-Computer come piccolo calcolatore

Quello che oggi si può eseguire con un calcolatore tascabile è veramente stupefacente. Oltre alle 4 operazioni fondamentali essi prevedono pure diverse funzioni scientifiche, usano aritmetica a virgola mobile, gli esponenti e molte altre cose. In linea di principio è possibile programmare lo Junior-Computer in modo da trattare qualsiasi problema numerico. Per sviluppare simili programmi però occorre una pratica di programmazione un pò superiore a quella di cui finora disponiamo. Perciò per ora ci limiteremo a programmare lo Junior-Computer come semplice macchina per ese-

guire addizioni. Tanto per cambiare eseguiamo le addizioni nel sistema decimale. Una cosa comunque dovrà corrispondere con l'operazione di un comune calcolatore tascabile: la successione con cui premere i tasti per l'introduzione dei numeri e per l'esecuzione delle operazioni. Nel caso di un normale calcolatore, per eseguire un'addizione i tasti vanno premuti nel seguente ordine:

XX + YY = ZZ

XX → 1° numero

+ → segno dell'addizione

YY → 2° numero

= → segno di eguale

ZZ → risultato della somma

Poiché il display dello Junior-Computer ha solo 6 elementi, è possibile rappresentare solo numeri a 6 cifre. Per mantenere il programma il più semplice possibile, introdurremo le seguenti limitazioni:

- nel caso di Overflow del display per numeri che superano la sua capacità non deve comparire un messaggio di errore. Questo problema lo risolviamo come esercizio alla fine di questo capitolo.
- I due numeri da sommare quando vengono introdotti devono spostarsi sul display da destra verso sinistra. Prima dell'introduzione di un numero il display deve spegnersi automaticamente.
- Deve essere possibile annullare con un tasto Clear un eventuale cifra impostata errata.

Per eseguire un'addizione decimale si impiegano i seguenti simboli:

1) I tasti delle cifre da 0 a 9

2) I tasti di comando +; =; Clear

I tasti cifra 0...9 sono già presenti nella tastiera dello Junior-Computer. È anche presente il tasto comando + (\$12). Non abbiamo invece disponibili il segno di = e il tasto Clear.

Assumeremo quindi:

tasto DA → = → \$11

tasto AD → Clear → \$10

La fig. 20 presenta la Flow Chart della Routine di addizione. Considerando il programma principale che inizia da CLEAR1, possiamo constatare che esso è costituito da una successione di diverse Subroutine. Ciascuna di queste Subroutine svolge un determinato compito. Così è possibile introdurre maggior chiarezza nel programma principale. Se una Subroutine è richiesta più volte, essa può venir richiamata quando richiesto con l'istruzione JSR.

Se si considera globalmente la Flow-Chart della Routine di addizione, si può osservare facilmente che la struttura del programma è quella di un sistema a blocchi costruttivi: i blocchi costruttivi del

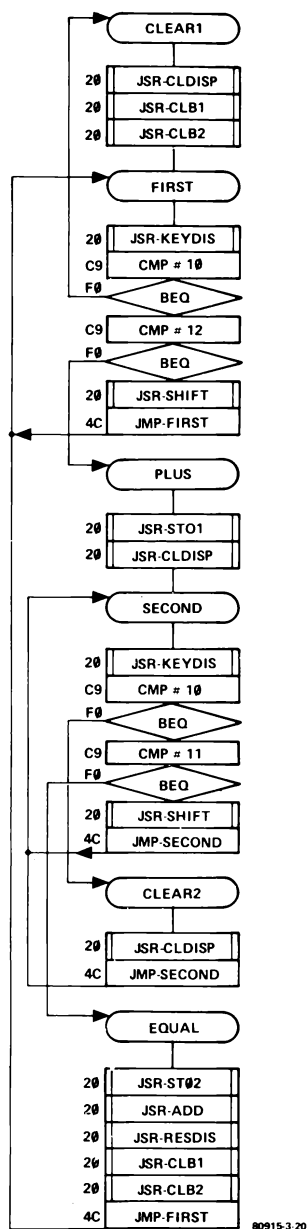


Figura 20. Programma principale per l'addizione di due numeri decimali a 6 cifre. Lo Junior-Computer viene utilizzato come un piccolo calcolatore.

programma principale solo le Subroutine. Prima di esaminare il programma in dettaglio, diciamo ancora qualcosa sulle celle buffer impiegate. Le celle buffer si trovano ai seguenti indirizzi:

POINTH 00FB	POINTL 00FA	INH 00F9	Display-Buffer
B12 0002	B11 0001	B10 0000	buffer per il 1° numero;
B22 0005	B21 0004	B20 0003	buffer per il 2° numero;
R2 0008	R1 0007	R0 0006	buffer per il risultato dell'addizione

La fig. 20 dunque illustra il diagramma di flusso dell'intero programma di addizione. Per poter eseguire in modo snello e facile il programma, ne descriviamo prima l'andamento a parole:

CLEAR1 : metti a 0 il buffer di display nonché i buffer numerici dei buffer da sommare (Buffer-Reset).
FIRST : salta alla Subroutine KEYDIS: questa Subroutine pilota il display dello Junior-Computer con la nostra vecchia Subroutine SCANDS. La Subroutine KEYDIS svolge anche le funzioni di riconoscimento dei tasti nonché l'eliminazione dei relativi contatti di rimbalzo. Il rientro dalla KEYDIS al programma principale è permesso solo quando è stato riconosciuto il tasto premuto. Quale tasto è stato premuto viene rivelato nel modo solito dalla Subroutine GETKEY. Successivamente al rientro da KEYDIS nel programma principale, il valore equivalente al tasto premuto si trova nell'accumulatore della CPU.

Ora il programma principale provvede ad interrogare per sapere quale tasto è stato premuto. Ciò è verificato in modo semplice dalla seguente istruzione CMP:

CMP#\$10 : si tratta del tasto CLEAR?

CMP#\$12 : si tratta del tasto +?

Se la risposta è no per entrambi questi due tasti di comando, non si può trattare altro che di un tasto cifra. Mediante la successiva Subroutine SHIFT il numero viene spostato nel display dello Junior-Computer e presentato sul display con la Subroutine KEYDIS. Che succede invece quando è stato premuto uno dei due tasti di comando CLEAR o + ? Se si tratta di CLEAR tutti i buffer vengono cancellati come avvenuto all'inizio della routine di addizione. Il display si riempie quindi nuovamente di 0 ed il numero appena introdotto viene cancellato. Se invece si tratta del tasto + il programma salta al Label PLUS.

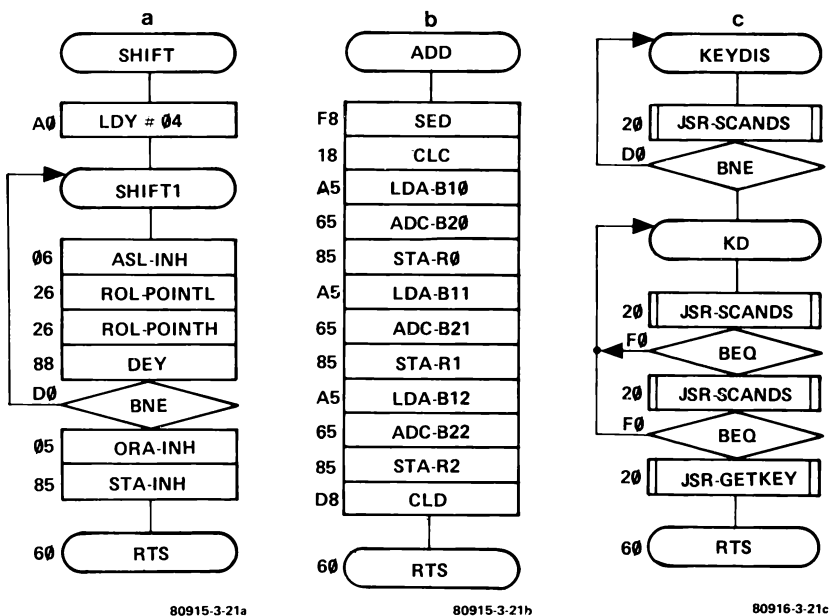


Figura 21. Le Subroutine per il programma di addizione di Fig. 20.

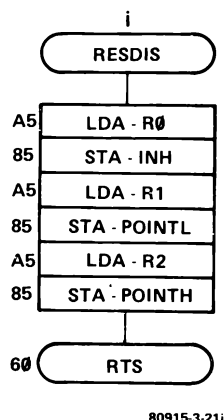
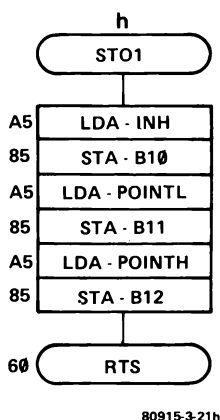
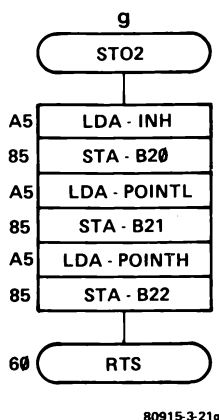
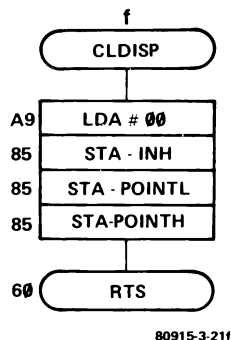
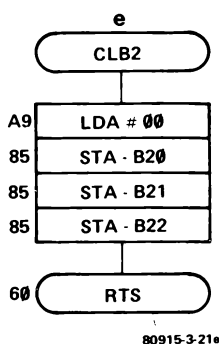
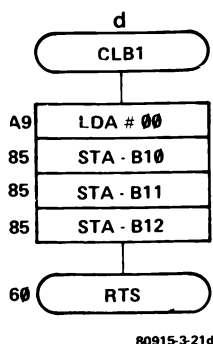
PLUS: salta alla Subroutine STO1. La STO1 trasferisce il numero appena introdotto dal buffer di display nel buffer del primo numero (Display-Buffer → buffer del primo numero). Il buffer di display è così libero per l'introduzione di un nuovo secondo numero. Prima occorre però rimetterlo a 0. La successiva Subroutine CLDISP esegue automaticamente questo compito.

SECOND: il programma principale esegue ancora un salto alla Subroutine KEYDIS. Questa provvede ancora a gestire il display e la tastiera dello Junior-Computer. Si può abbandonare la KEYDIS solo quando viene nuovamente premuto un tasto. Dopo il rientro dalla KEYDIS nel programma principale, il valore del tasto premuto si trova nell'accumulatore del micro-processore. Per la seconda volta mediante un ciclo di istruzioni viene richiesto quale tasto è stato premuto. Ciò si può determinare anche in questo caso con le seguenti istruzioni CMP:

CMP #\$10 : si tratta del tasto CLEAR?

CMP #\$12 : si tratta del tasto +?

Se non è nessuno di questi due tasti di comando, si può trattare solo di un tasto numerico. Mediante la successiva Subroutine SHIFT il secondo numero introdotto viene spostato nel display dello Junior-Computer e presentato sullo stesso con la Subroutine successiva KEYDES.



B10 → 0000
 B11 → 0001
 B12 → 0002

B20 → 0003
 B21 → 0004
 B22 → 0005

R0 → 0006
 R1 → 0007
 R2 → 0008

INH → 00F9
 POINTL → 00FA
 POINTH → 00FB

SCA*IDS → 1D8E
 GETKEY → 1DF9

Se risulta invece premuto il tasto CLEAR, il numero introdotto per ultimo viene cancellato. Questo compito è affidato alla Subroutine CLDISP. Il display viene così nuovamente riempito con zeri. Se invece è stato attivato il tasto =, per il programma principale ciò significa che è terminata l'introduzione dei numeri e deve essere eseguita la somma dei due numeri introdotti. Il programma salta quindi al Label EQUAL per eseguire questo compito.

EQUAL: il programma principale esegue adesso, con l'ausilio di alcune Subroutine, l'effettiva addizione e predispone lo Junior-Computer ad una nuova introduzione di numeri. Vediamo in dettaglio.

La Subroutine STO2 porta il numero introdotto per ultimo dal display buffer nel buffer numerico. La funzione logica relativa è: display buffer → buffer del 2° numero. I due numeri introdotti si trovano ora nei rispettivi buffer numerici ed il display è così libero per la presentazione del risultato. Nella successiva Subroutine ADD i due numeri vengono sommati in decimale ed il risultato posto nel buffer del risultato. Con la Subroutine RESDIS il contenuto del risultato viene trasferito nel display buffer dello Junior-Computer. Le Subroutine CLB1 e CLB2 rimettono a 0 i buffer numerici, e lo Junior-Computer è pronto per l'introduzione di due nuovi numeri. La presentazione del risultato è ancora una volta affidata alla Subroutine KEYDIS.

Osservazione: da questo programma di addizione si vede chiaramente come le Subroutine rendono più chiaro e comprensibile il programma principale.

Per scrivere un programma è quindi consigliabile applicare il già citato principio della scatola di costruzioni. In tal modo anche seconde o terze persone possono facilmente seguire l'andamento di un programma, anche complicato. In fig. 21 sono illustrate le Subroutine impiegate.

Indexed Addressing

In questo capitolo consideriamo i tipi di indirizzamento indicizzato della CPU 6502. L'indirizzamento indicizzato si può suddividere in 4 gruppi:

- Absolute Indexed, X Addressing
- Zero Page Indexed, X Addressing
- Absolute Indexed, Y Addressing
- (Indirect), Y - Indirect Indexed Addressing
- (Indirect), X - Indexed Indirect Addressing

Questi 4 gruppi di indirizzamento indicizzato verranno più attentamente considerati nel corso di questo capitolo. Particolare attenzione sarà dedicata ai tipi di indirizzamento indiretto che rendono la programmazione enormemente più semplice.

Absolute Indexed, X Addressing

Chiariamo l'Absolute Indexed, X Addressing sulla base di una istruzione di caricamento e scrittura ed una istruzione aritmetica. Prima di esaminare attentamente il programma che segue, dobbiamo ancora chiarire il termine "campo" (in inglese FIELD). Per campo s'intende un raggruppamento di dati (= numeri) depositati uno dietro l'altro in una qualsivoglia zona di memoria del microcomputer. Vediamolo con un esempio:

```

016A   AA
016B   BB

```

```

016C  1F
016D  9A
016E  4D

```

Il campo si estende da 016A a 016E e contiene 5 dati. In programmi di grandi dimensioni, ad esempio un Text-Editor, un BASIC Compiler eccetera, sono spesso presenti campi dati che occupano uno spazio di memoria di diversi K-bytes.

Per "operando" si deve intendere il nome di una cella di memoria. Alcuni esempi: PNH, POINTL, POINTH, B11, B12, B13, TEMPX, COUNTL, COUNTH... Un operando può anche consistere tuttavia di un numero esadecimale o un indirizzo esadecimale. Non vengono posti limiti alla lunghezza dei numeri. Se lo Junior-Computer opera in Decimal Mode, un operando può essere costituito da un numero decimale qualsiasi.

Nel programma che segue (fig. 22) vengono considerati 3 campi dati, ciascuno della lunghezza di 10 bytes. I campi si chiamano

```

FIELD1 $0120 ... $0129
FIELD2 $0011 ... $001A
FIELD3 $0300 ... $0309    con i dati:

```

FIELD1:	0120	FIELD2:	0011	FIELD3:	0300
0120	01	0011	10	0300	11
0121	02	0012	20	0301	22
0122	03	0013	30	0302	33
0123	04	0014	40	0303	44
0124	05	0015	50	0304	55
0125	06	0016	60	0305	66
0126	07	0017	70	0306	77
0127	08	0018	80	0307	88
0128	09	0019	90	0308	99
0129	0A	001A	A0	0309	AA

Il compito che questo programma svolge è quello di addizionare due dati posti in FIELD 1 e in FIELD 2 e di registrare il risultato in FIELD 3. Ad esempio:

- carica la 5^a cifra di FIELD 1 nell'accumulatore
- somma a questa la 5^a cifra di FIELD 2
- memorizza il risultato come 5^a cifra in FIELD 3

Naturalmente questo problema può essere affrontato senza alcuna difficoltà con il nostro vecchio Absolute Addressing. Ma questo tipo di soluzione risulta piuttosto scomoda. Per campi dati piuttosto lunghi questo piccolo programma di addizione diventerebbe infinitamente lungo. Adoperiamo perciò l'indirizzamento assoluto indicizzato, X della CPU 6502. Questo tipo di indirizzamento è lungo 3 bytes e si può descrivere come segue:

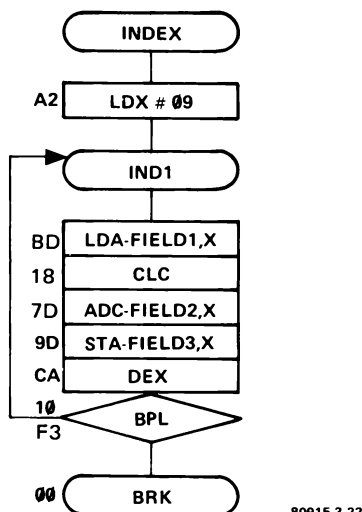


Figura 22. Programma per l'addizione di 2 campi. Il risultato viene registrato nel campo 3.

Denominazione: Esempio:
 codice operativo LDA-FIELD1, X carica l'accumulatore
 indicizzato

ADL parte bassa del byte indirizzo di FIELD 1

ADH parte alta del byte indirizzo di FIELD 1

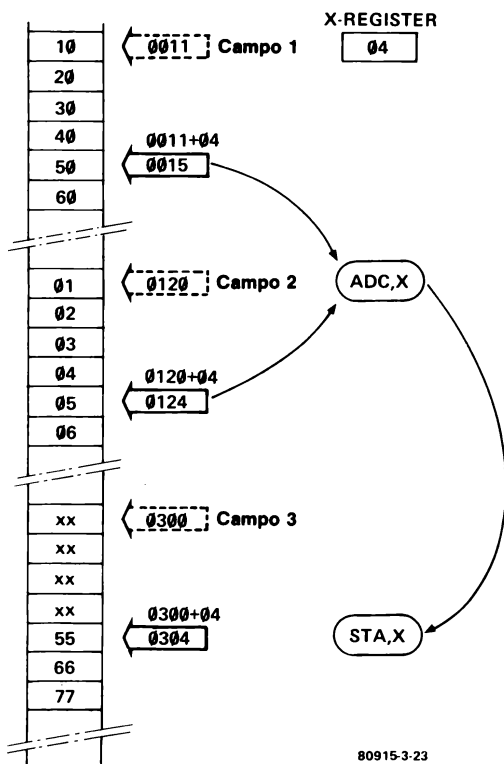
L'indirizzo effettivo a partire dal quale si effettua il caricamento nell'accumulatore *non* è l'indirizzo di partenza di FIELD 1 (ADH, ADL = 0120) bensì ADH, ADL *più* il contenuto del registro X. Se ad esempio il contenuto del registro X vale \$04 e l'indirizzo inizio di FIELD1 è 0120, nell'accumulatore vengono caricati i dati depositati all'indirizzo 0124. Questo indirizzo 0124 si calcola infatti da: ADH, ADL + X = 0120 + 04 = 0124. Nell'accumulatore, dopo le operazioni di caricamento indicizzato, si trova quindi \$05.

Dopo aver addizionato tutti i dati dei due campi e averli depositati in FIELD3, il registro X si è annullato. Il processore abbandona il ciclo di programma considerato ed interrompe il programma (BRK). La fig. 23 illustra un tratto istantaneo di questo programma di addizione.

Spesso si devono trasferire blocchi di dati da una zona di memoria ad un'altra. Con la terminologia introdotta, si può dire che si deve trasferire un campo in un'altra zona di memoria. La fig. 24 mostra cosa si intende con ciò. Il blocco dati da trasportare inizia a FROM-AD. L'indirizzo iniziale è 0172. Il blocco dati consta solo di 5 dati. Questi dati vengono trasferiti in una nuova zona di memoria dello Junior-Computer, che inizia all'indirizzo TOAD ovvero 03A0. Seguendo il diagramma di flusso, si vede che inizialmente il regi-

stro X vale 04. Perciò nell'accumulatore vengono caricati i dati depositati all'indirizzo $\text{FROMAD} + X = 0172 + 04 = 0176$. La successiva istruzione STA, X memorizza il contenuto dell'accumulatore all'indirizzo $\text{TOAD} + X = 03A0 + 04 = 03A4$. Il registro indice X viene ora incrementato di 1, consentendo così l'accesso ad una nuova cella di memoria. La successiva istruzione di salto BPL viene eseguita finché il contenuto del registro X è divenuto negativo. I valori che il registro X assume successivamente durante lo svolgimento del programma sono: 04, 03, 02, 01, 00. Al termine del programma MOVE si salta al Monitor dello Junior-Computer, che inizia all'indirizzo 1C1D. Dopo l'esecuzione del programma, lo Junior-Computer si rifà vivo illuminando il display. La successiva lista dei tasti attivati illustra come si introduce nello Junior-Computer il programma MOVE tramite la tastiera:

AD				xxxx	xx
0	2	0	0	0200	xx
DA		A	2	0200	A2 LDX#



80915-3-23

Figura 23. Un'istantanea dello svolgimento del programma di fig. 22 per $X = 04$.

+	0	4	0201	04	
+	B	D	0202	BD	LDA-,X
+	7	2	0203	72	ADL di FROMAD
+	0	1	0204	01	ADH di FROMAD
+	9	D	0205	9D	STA-,X
+	A	0	0206	A0	ADL di TOAD
+	0	3	0207	03	ADH di TOAD
+	C	A	0208	CA	DEX
+	1	0	0209	10	BPL
+	F	7	020A	F7	Offset
+	4	C	020B	4C	JMP
+	3	3	020C	33	
+	1	C	020D	1C	dell'indirizzo del Monitor;
AD			020D	1C	
0	2	0	0	0200	A2
GO			0200	A2	inizio programma

Prima di far partire il programma, occorre inizialmente caricare in memoria all'indirizzo 0172 i dati AA, BB, CC, DD e EE. Dopo lanciato il programma, si può controllare agli indirizzi 03A0...03A4 se effettivamente vi risulta depositata la copia del blocco di dati.

Se all'inizio del programma si carica \$FF nel registro X, è possibile copiare da una zona di memoria in un'altra un massimo di 256 bytes. Bisogna comunque fare attenzione che le due zone di memoria non si sovrappongano. Altrimenti nelle zone di sovrapposizione i dati che dovrebbero essere copiati vengono sovrascritti in modo errato, oppure distrutti.

Zero Page Indexed, X Addressing

Lo Zero Page Indexed, X Addressing è una particolare forma dell'Absolute Indexed, X Addressing. La struttura per lo Zero Page Indexed, X Addressing per una istruzione di caricamento è: LDA-operando, X codice-OP B5.

L'operando è il nome dell'indirizzo di una cella di memoria situata in Pagina 0. Il campo d'indirizzi risulta quindi da 0000 a 00FF, come già sappiamo dallo Zero Page Addressing.

Absolute Indexed, Y e Zero Page Indexed, Y

Non c'è bisogno di spendere molte parole per questi tipi di indirizzamento, dato che svolgono la stessa funzione dello Indexed, X Addressing. L'unica differenza fra lo Indexed, X e lo Indexed, Y sta naturalmente nel fatto che nel primo caso si impiega quale registro indice il registro X, e nel secondo il registro Y. Ci si potrà chiedere: perché due distinti registri indice? Il vantaggio dell'impiego

di due registri indici sta nel fatto che è così possibile “annidare” due cicli di programma in un solo programma mediante i due registri indici. Spiegheremo nel capitolo successivo come ciò funziona. In seguito mostreremo pure, sulla base di un esempio di trasformazione di codici, come diventi facile e chiaro scrivere programmi proprio perché la CPU 6502 dispone di due registri indici.

Modalità di indirizzamento indiretto

I tipi di indirizzamento indiretto sono illustrati alla fine del capitolo 3. Con ciò abbiamo quindi terminato la descrizione dei 13 modi di

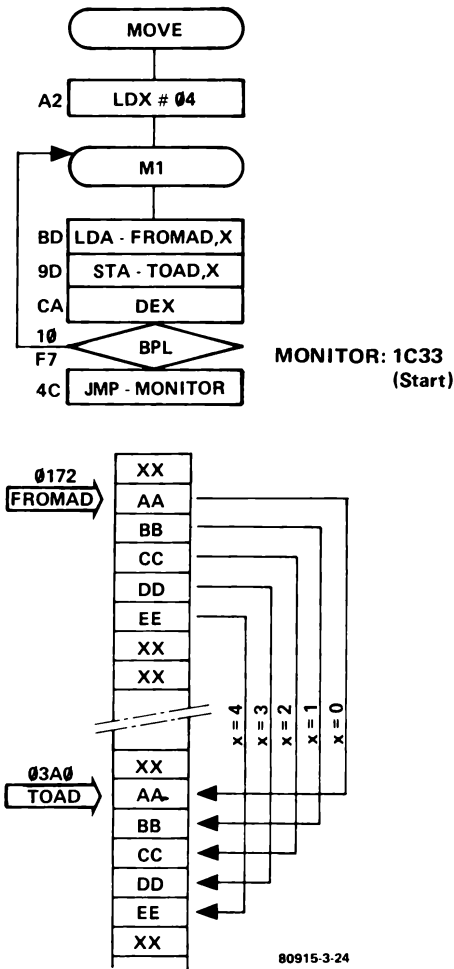


Figura 24. Questo programma copia dati da un campo di memoria in un altro.

indirizzamento della CPU 6502. Vogliamo comunque far notare che lo Indirect Addressing non è davvero un modo di indirizzamento facile da apprendere! Prenderemo quindi meglio confidenza con questo tipo di indirizzamento per mezzo di esempi. Quando però questo tipo di indirizzamento sarà stato interamente assorbito, il programmatore imparerà veramente ad apprezzare l'enorme robustezza dell'Instruction Set del 6502: grazie all'indirizzamento indiretto i programmi divengono corti e assai chiari. Altro vantaggio da non trascurare è che non occorre conoscere sin dall'inizio del programma gli indirizzi nei quali avverranno operazioni di lettura o scrittura. Il programmatore ha la possibilità di memorizzare successivamente questi indirizzi, in pagina zero, o addirittura trasferire al computer il compito di calcolare gli indirizzi richiesti. Il procedimento di indirizzamento indiretto di cui dispone costituisce sicuramente la ragione fondamentale del perché la CPU 6502 si sia candidata come uno dei micro-processori di maggior successo. La miglior prova sta nel suo impiego in calcolatori di processo, in calcolatori di navigazione e nella preferenza accordata dagli amatori dei computer. Mentre altri tipi di processori hanno 3, 6 o addirittura un paio di Pointer Register in più sul Chip della CPU, la 6502, con l'eccezione dello Stack Pointer, non ha alcun Pointer Register sul Chip. In compenso l'intera pagina zero può essere programmata quale Pointer Register. Dato che un Pointer risulta lungo 16 bit, è possibile registrare in pagina zero 128 Pointer! Cosa si può farne di questi Pointer lo vedremo tra breve.

Indirect Indexed Addressing

Prima di descrivere in dettaglio l'indirizzamento indiretto indicizzato, rammentiamoci un momento l'indirizzamento assoluto indicizzato. In tal caso viene impiegato quale registro indice il registro Y. Per cui è possibile confrontare lo Absolute Indexed, Y Addressing con lo Indirect Indexed Addressing.

La fig. 25 mostra nella parte superiore l'Absolute Indexed, Y Addressing, in cui si impiega Y quale registro indice. L'operazione eseguita è il caricamento indicizzato nell'accumulatore del contenuto di una cella di memoria. Il registro indice vale nel nostro esempio \$00. Nella memoria i singoli bytes sono collocati in successione come segue:

A0 codice op LDY

00 operando

B9 codice OP LDA-ABS, Y

1A parte bassa del byte indirizzo } operando

02 parte alta del byte indirizzo }

Il codice OP B9 incarica la CPU di caricare nell'accumulatore i dati posti all'indirizzo 021A. Infatti l'indirizzo effettivo dal quale deve essere iniziato il caricamento si calcola da: indirizzo = ADH,

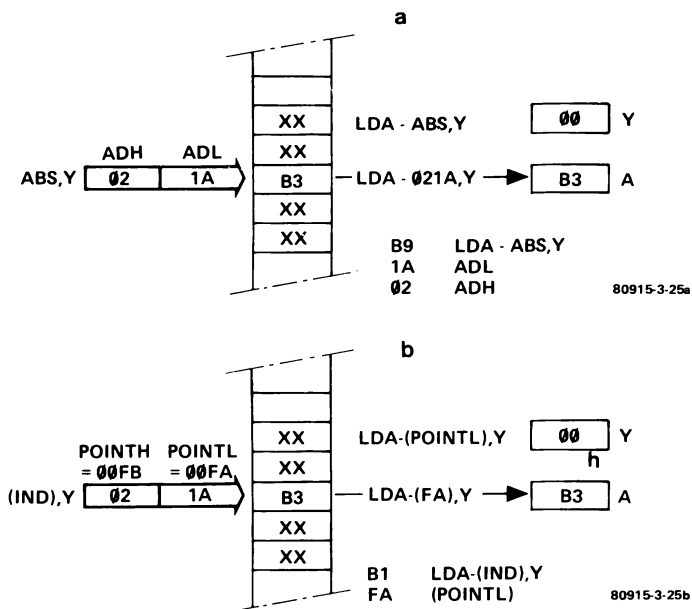


Figura 25. Il caricamento dell'accumulatore mediante Indirect Indexed Addressing. Il contenuto delle celle di memoria POINTH, POINTL costituisce un Pointer indirizzi che indica l'indirizzo dal quale i dati devono essere caricati nell'accumulatore.

$ADL+Y = 021A+00 = 021A$

Nulla di nuovo quindi, a parte il fatto che rispetto alla sezione precedente del libro impiegato il registro Y quale registro indice. Quando è stato eseguito il codice operativo B9, troviamo B3 nell'accumulatore della CPU (fig. 25a).

Nell'indirizzamento assoluto, che più propriamente dovrebbe chiamarsi indirizzamento diretto, l'indirizzo al quale si deve leggere o scrivere deve sempre essere noto direttamente in precedenza. L'indirizzo assoluto, cioè, deve seguire direttamente il codice operativo. Lo stesso vale anche per il tipo di indirizzamento indicizzato. Per questo indirizzamento si deve sommare il contenuto del registro indice al valore dell'indirizzo che segue direttamente il codice OP.

Nel caso dell'indirizzamento indiretto l'indirizzo a cui leggere o scrivere non è noto in precedenza. Lo si vede chiaramente dalla fig. 25b.

Anche in questo caso i dati posti all'indirizzo 021A vengono caricati nell'accumulatore. Ciò avviene tuttavia non come prima con l'Absolute Indexed, Y ma con lo Indirect Indexed, Y. La scrittura di questo tipo di indirizzamento è: (IND), Y. Tra le parentesi è posto un indirizzo indiretto presso il quale la CPU deve controllare da

quale indirizzo effettivo devono essere prelevati dei dati o a quale devono essere scritti i dati. Questo indirizzo indiretto per la CPU 6502 è *sempre* un indirizzo in pagina zero. Poiché l'indirizzo effettivo, cioè l'indirizzo i cui dati devono essere elaborati in qualche modo, nel caso della CPU 6502 è lungo 16 bit, occorre riservare a questo tipo di indirizzo 2 bytes in pagina zero. Sapendo ciò è ora facile descrivere lo Indirect Indexed Addressing. Cosa osserviamo in fig. 25b per l'indirizzamento indiretto indicizzato? Come già detto il contenuto della cella di memoria 021A viene caricato nell'accumulatore della CPU. Per comprendere meglio traduciamo in parole le singole operazioni che il micro-processore esegue nell'indirizzamento indiretto indicizzato:

1. In Pagina zero troviamo due indirizzi successivi detti POINTL e POINTH. I loro indirizzi esadecimali sono:

POINTL = 00FA

POINTH = 00FB

2. POINTL e POINTH sono celle di una RAM il cui contenuto può venire modificato a volontà. Tramite tastiera scriviamo in tali celle i dati seguenti:

dati in POINTL : 1A

dati in POINTH : 02

Leggendo uno dopo l'altro i 2 bytes, si possono interpretare ancora come un indirizzo assoluto. Questo indirizzo assoluto si compone di 1 byte basso, 1A, e un byte alto indirizzo 02. Il byte basso si trova nella cella di memoria POINTL e il byte alto indirizzo nella cella di memoria POINTH.

3. L'accesso alle celle di memoria POINTL e POINTH avviene tramite l'indirizzamento indiretto della CPU 6502. Il codice OP B1 sta per LDA-(IND), Y. Ciò significa che la CPU inizialmente accede ai due indirizzi indiretti POINTL e POINTH, e ne estrae l'indirizzo diretto dal quale deve essere iniziato il caricamento dati. L'indirizzo diretto funziona quindi come un indirizzo assoluto.

4. Al valore dell'indirizzo diretto o assoluto, posto nelle celle POINTL e POINTH, viene quindi sommato il contenuto del registro Y. Nel nostro esempio il contenuto del registro Y è 0. L'indirizzo diretto dal quale i dati devono essere prelevati e caricati nell'accumulatore si calcola quindi: $021A + Y = 021A + 00 = 021A$. Nell'accumulatore vengono quindi effettivamente caricati i dati posti alla cella di memoria con l'indirizzo 021A.
5. Dato che gli indirizzi indiretti per la CPU 6502 si trovano sempre in pagina 0, le istruzioni che impiegano lo Indirect Indexed Addressing sono lunghe solo 2 bytes. Quando effettuiamo tramite tastiera una operazione di caricamento indiretto in memoria, le cose vanno così:

B1 codice OP di LDA-(IND), Y

FA operando costituito dal primo byte di indirizzo indiretto

to in pagina zero, nel quale viene posto il byte basso indirizzo dell'indirizzo di caricamento diretto. La CPU estrae automaticamente da 00FB il byte alto di indirizzamento diretto. Scrivendo l'indirizzamento indiretto indicizzato nel solito modo, lo possiamo esprimere come segue:

A0 # 00 LDY00 carica 00 nel registro Y
B1FA LDA-(POINTL), Y carica indirettamente l'accumulatore

È così ora possibile fornire in modo semplice una definizione dell'indirizzamento indiretto indicizzato.

Osservazione: Le istruzioni che impiegano lo Indirect Indexed Addressing sono lunghe 2 bytes. Il primo byte è il codice operativo. Il successivo secondo byte è un indirizzo indiretto in pagina 0. Al contenuto di questa cella di memoria indiretta in pagina 0 viene sommato il contenuto del registro Y. Il risultato costituisce il byte passo indirizzo dell'indirizzo diretto o assoluto. Se nell'effettuare questa somma si trasferisce un riporto sul Carry Flag, esso viene automaticamente sommato al contenuto dell'indirizzo successivo indiretto in pagina 0. Il risultato costituisce il byte alto di indirizzo dell'indirizzo diretto o assoluto.

Trasferimento di blocchi con lo Indirect Indexed Addressing

Già in fig. 24 abbiamo eseguito un trasferimento di blocchi impiegando lo Absolute Indexed, X Addressing. Con tale programma è possibile copiare al massimo 256 bytes da una zona di memoria in un'altra. Spesso tuttavia è necessario trasferire un numero di dati superiore a 256 byte da una zona di memoria ad un'altra. Questo è possibile senza difficoltà con la mini Subroutine BLMOVE. Con questa Subroutine si possono trasferire da una zona ad un'altra di memoria sino a 255 blocchi di dati, ciascuno dei quali contiene 256 bytes. La Subroutine BLMOVE, con la quale è possibile trasferire un enorme numero di dati, risulta così breve grazie al tipo di indirizzamento indiretto indicizzato della CPU 6502.

Nell'esempio di fig. 26, per semplicità, mostriamo come trasferire solo due blocchi di dati, ciascuno di 256 bytes, da una zona di memoria ad un'altra. Facciamo a tal fine le seguenti ipotesi:

1. L'indirizzo di partenza del blocco di dati da trasferire sia 0200. Tale indirizzo diretto od assoluto sia posto nelle due celle di memoria BEG, e BEG+1 successive di pagina zero. BEG, e BEG+ successive di pagina zero. BEG, e BEG+1 hanno i seguenti indirizzi:

BEG = 0000 il contenuto è FRADL = 00

BEG+1 = 0001 il contenuto è FRADH = 02

2. Il campo di indirizzi nei quali debbono essere copiati i due dati di blocchi inizi all'indirizzo A800. Questo indirizzo indiretto o assoluto sia posto nelle due celle immediatamente successive

MOV ed MOV + 1 in pagina zero. MOV ed MOV+1 hanno attribuiti i seguenti indirizzi:

MOV = 0002 il contenuto è TOADL = 00

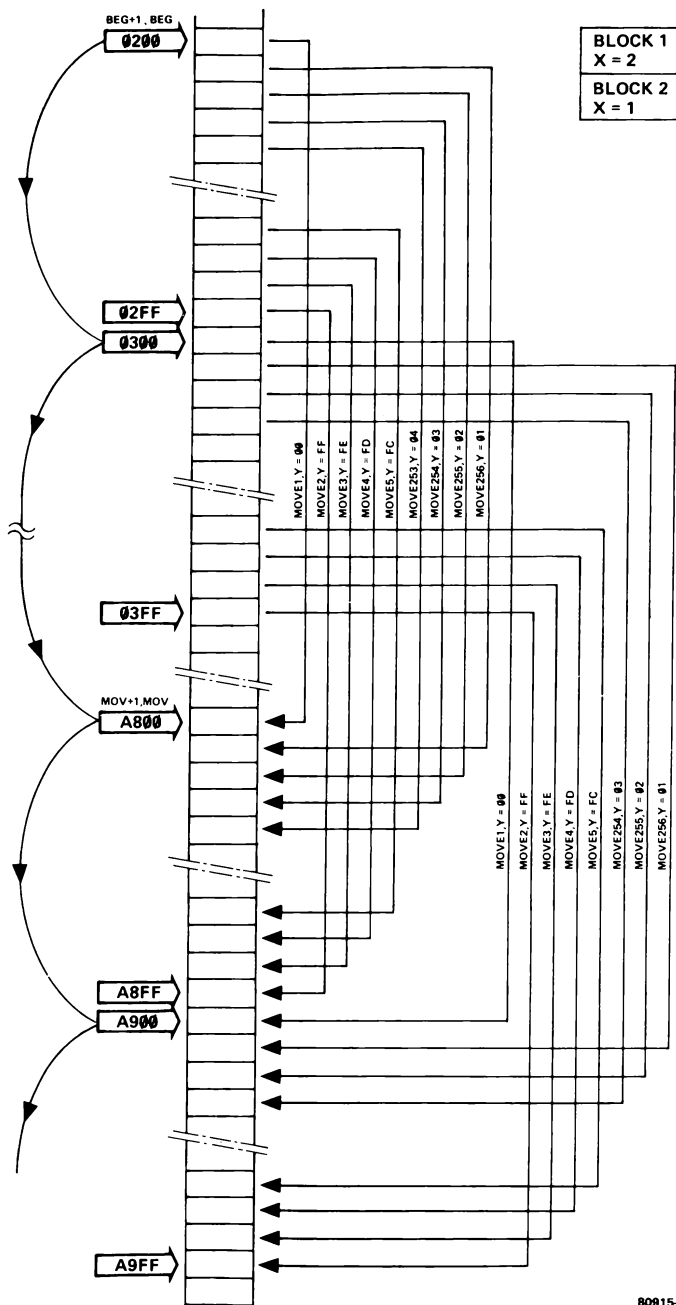
MOV+1 = 0003 il contenuto è TOADH = 08

3. Il numero dei blocchi da trasferire sta nella cella di memoria BLOCKS. Se vogliamo ad esempio trasferire 12 blocchi di dati, scriviamo in questa cella OC. Nell'esempio, poiché intendiamo trasferire solo due blocchi di dati, scriviamo nella cella di memoria BLOCKS il valore 02. Alla cella di memoria BLOCKS viene attribuito il seguente indirizzo:

BLOCKS = 0004 il contenuto è N = 02

Il caricamento dei bytes nelle celle di memoria BEG, BEG+1, MOV, MOV+1 e BLOCKS può venire effettuato sia a mano tramite la tastiera, come pure mediante un programma. Il programma DEF-MOV introduce automaticamente gli indirizzi diretti negli indirizzi indiretti appositamente ad essi riservati (0000...0004). Poi si ha un salto alla Subroutine BLMOVE, procedendo ad eseguire il trasporto dei dati nel modo richiesto. Come funziona in dettaglio la Subroutine BLMOVE? Vediamo di rispondere:

1. Il registro indice X viene programmato come contatore di blocchi. A tale scopo si carica in esso il contenuto della cella di memoria BLOCKS ricavando così l'informazione del numero di blocchi di dati che il programmatore intende trasferire.
2. Il registro Y viene impiegato quale vero e proprio registro indice. Il suo contenuto indica il numero della cella di memoria, entro un blocco di dati di 256 bytes, che viene richiamata. Se ad esempio il contenuto del registro Y è 21 (esadecimale) significa che viene richiamata la 33^a cella di memoria in questo blocco di dati.
3. I dati che debbono essere copiati da una zona di memoria in un'altra vengono caricati dalla CPU nell'accumulatore mediante l'Indirect Addressing. L'istruzione relativa è: LDA-(BEG), Y. Quindi la CPU trascrive con Indirect Indexed Addressing i dati dell'accumulatore, testé ricavati, nella zona di memoria. L'istruzione relativa è: STA(MOV), Y.
4. In funzione del valore del registro Y successivamente incrementato di 1, la CPU viene indirizzata ad una nuova cella di memoria interna al blocco di dati da 256 bytes. In questo programma il registro Y assume i valori: 00, FF, FE...02, 01, 00, FF, FE e così via.
5. Ogni volta che un blocco di dati da 256 byte viene trasportato da una zona di memoria in un'altra, il contenuto dei due indirizzi indiretti BEG+1 e MOV+1 viene incrementato di 1. In tal modo la CPU ricava l'informazione sul nuovo blocco di dati anch'esso da 256 bytes.
6. Dopo aver effettuato il trasporto da un blocco di dati da una zona di memoria ad un'altra, il registro X programmato quale contatore dei blocchi viene decrementato di 1. Solo quando



80915-3-26

Figura 26. Lo spostamento di due blocchi di dati da 256 byte ciascuno. Il programma che esegue questo trasferimento è mostrato in fig.27.

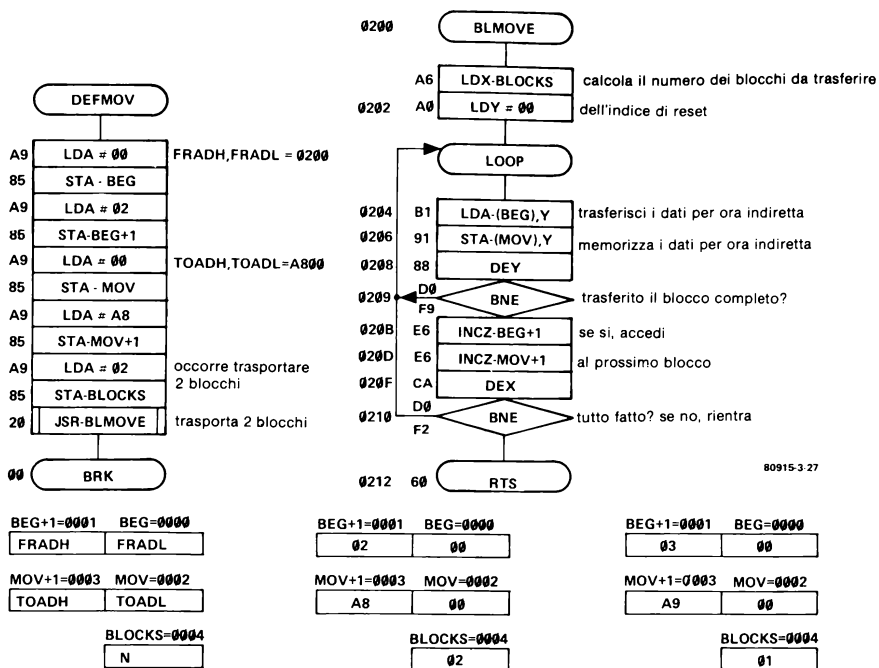


Figura 27. Con questo programma è possibile trasferire un massimo di 256 blocchi di memoria da 256 byte ciascuno. Ciò costituisce l'intero campo di memoria indirizzabile della CPU 6502. Il numero contenuto nella cella di memoria BLOCKS indica quanti blocchi dati debbono venir trasferiti.

tutti i blocchi di dati sono stati trasferiti, il contenuto del registro X diviene 00. A questo punto la CPU abbandona il ciclo di trasferimento LOOP e rientra con un salto nel programma principale,

Nella fig. 26 si ritrovano illustrati graficamente i 6 punti sopra descritti. Il meccanismo dello Indirect Indexed Addressing dovrebbe ora essere chiaro anche nei dettagli:

- Negli indirizzi indiretti BEG+1, BEG è memorizzato l'indirizzo diretto col quale il micro-processore vuole entrare in contatto. In seguito definiremo celle di memoria come BEG e BEG+1 come Pointer indirizzi (BEG+1, BEG = Pointer indirizzi). Un Pointer indirizzi consta di 2 celle di memoria indirette immediatamente successive in Pagina Zero. Il contenuto del Pointer indirizzi definisce direttamente la cella di memoria con la quale la CPU intende porsi attivamente in contatto tramite il bus indirizzi e il bus dati.
- Mediante il registro indice Y il micro-processore ha accesso a 256 celle di memoria. Queste 256 celle iniziano dall'indirizzo

definito dal Pointer indirizzi ($BEG + 1, BEG$) = 0200. L'indirizzo effettivo si calcola da: $BEG + 1, BEG + Y$. Se $Y = 00$ il caricamento indiretto porta nell'accumulatore i dati presenti all'indirizzo 0200, ma se $Y = FF$ nell'accumulatore finiscono i dati all'indirizzo 02FF.

- Per l'operazione di scrittura si impiega il Pointer indirizzi $MOV+1, MOV$. Anche a questo Pointer si somma il contenuto del registro indice Y . Le operazioni di caricamento e scrittura mediante l'indirizzamento indiretto con l'uso del registro Y , mostrate graficamente in fig. 26, dovrebbero ora essere chiare. Nel capitolo successivo descriveremo un nuovo tipo di indirizzamento della CPU 6502 nel quale si impiega ancora il concetto ora descritto del Pointer. Conviene rileggere ancora una volta con attenzione il presente capitolo se non si è ancora capito cos'è un Pointer indirizzi e come si può manipolare.

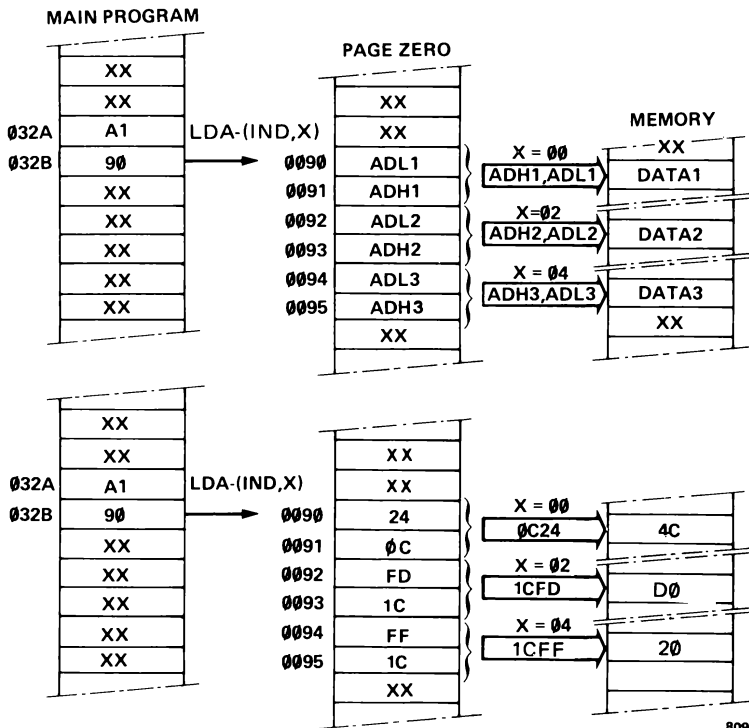
Indexed Indirect Addressing

Dopo aver descritto nel capitolo precedente l'Indirect Indexed Addressing in dettaglio, possiamo limitarci a brevi cenni per descrivere l'Indexed Indirect Addressing. Entrambi i tipi di indirizzamento hanno in comune l'impiego di celle di memoria in pagina zero quali Pointer indirizzi. Anche la lunghezza dell'istruzione è in entrambi i casi lunga solo 2 bytes. L'Indexed Indirect Addressing viene impiegato per prelevare, tramite 1 Pointer indirizzi, 1 byte dati dalla memoria o per modificare 1 byte dati nella memoria. I Pointer indirizzi sono ancora posti in celle di memoria successive in pagina 0. Come registro indice si impiega il registro X . Con l'aiuto della fig. 28 descriviamo ora il corso di un'operazione di caricamento indicizzato indiretto.

Nella parte superiore di questa figura è illustrato in generale l'Indexed Indirect Addressing, mentre la parte inferiore impiega dati concreti. All'indirizzo 032A la CPU trova il codice $OP A1 = LDA-(IND, X)$, cioè un'operazione di caricamento indiretto indicizzato dell'accumulatore. Gli indirizzi dei dati da caricare nell'accumulatore sono posti in pagina zero con Pointer. Il 1° Pointer indirizzi è posto nelle celle di memoria ed indirizzi 0090, 0091, il 2° Pointer indirizzi nelle celle 0092, 0093, ecc. In base a quale Pointer indirizzi debbono venir caricati i dati nell'accumulatore è stabilito dal registro X .

Ciò è chiaramente illustrato nella parte inferiore di fig. 28. In pagina zero sono registrati 3 Pointer indirizzi. I loro indirizzi sono 0C24, 1CFD e 1CFE. Se il contenuto del registro X è 00, nell'accumulatore vanno caricati i dati della cella di memoria 0C24. Se ad esempio prima dell'operazione di caricamento si è caricato 04 nel registro X , allora è la volta dei dati all'indirizzo 1CFE di venir caricati nell'Accu. Per effetto di un doppio incremento del registro X si ha quindi accesso al successivo Pointer, posto in pagina zero.

Come ben si vede da questo esempio, con l'Indexed Indirect Addressing si può programmare la pagina 0 come "Pointer Look Up Table". In pagina 0 cioè si possono porre Pointer indirizzi che indichino determinati dati in memoria. Questi Pointer con l'Indirect Indexed Addressing, tramite il registro X, possono essere richiamati come indici. L'Indirect Indexed Addressing viene in effetti usato relativamente poco. Quando tuttavia si implementano linguaggi di programmazioni superiori, ad esempio il Basic, questo tipo di indirizzamento viene spesso usato.



80915-3-28

Figura 27. Con questo programma è possibile trasferire un massimo di 256 blocchi di memoria da 256 byte ciascuno. Ciò costituisce l'intero campo di memoria indirizzabile della CPU 6502. Il numero contenuto nella cella di memoria BLOCKS indica quanti blocchi dati debbono venir trasferiti.

NMI, IRQ, RESET, ovvero i comando di Interrupt della CPU 6502

I concetti di Interrupt e di vetture sono stati descritti nell'ultima parte del capitolo 3. Con ciò vennero descritti tutti i possibili tipi di indirizzamento della CPU 6502. Si noti innanzitutto che qui abbiamo a che fare con un modo di indirizzamento indiretto simile al-

l'indirizzamento indiretto indicizzato. Disponiamo quindi già delle cognizioni preliminari e siamo in grado di comprendere facilmente che cosa è un Interrupt. Sugli Interrupts è stato scritto molto. Purtroppo nella letteratura si trovano solo descrizioni imprecise di questi comandi che per di più usualmente si riferiscono ad un singolo sistema. Per descrivere esattamente a che servono gli Interrupt la miglior cosa è di confrontarli con tipi di indirizzamento già noti. Vediamo un pò la fig. 29a.

Riconosciamo ancora il buon vecchio salto ad una Subroutine. L'istruzione per deviare dal programma principale in questa Subroutine è JSR. Con questa istruzione di salto è possibile richiamare un determinato sotto programma dalla memoria del computer. Dopo che il sottoprogramma o Subroutine è stato elaborato, il micro-processore incontra l'istruzione RTS con il codice OP 60.

Bild 29b.

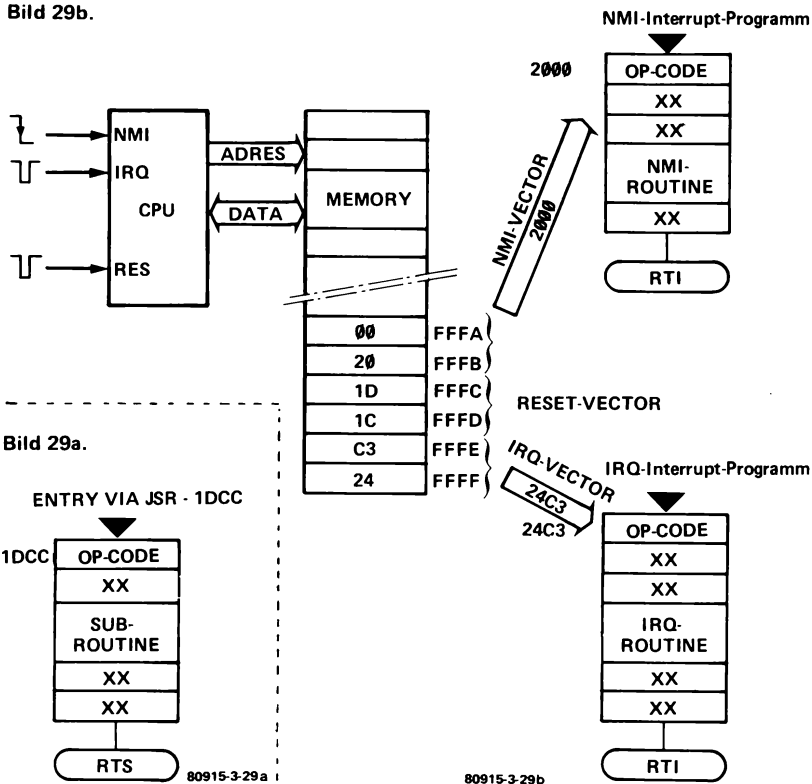


Figura 29. Vengono messi a confronto il salto ad una Subroutine ed il salto in una routine Interrupt. Il vettore IRQ o NMI fissa l'indirizzo di partenza della corrispondente Routine Interrupt. L'istruzione RTI riporta il processore al programma principale.

L'istruzione RTS riconduce il processore nuovamente nel programma principale. Affinché il processore sappia a quale indirizzo deve rientrare nel programma principale, lo stesso programma prima del salto alla Subroutine ha provveduto a depositare il precedente valore del contatore di programma sullo Stack.

Un Interrupt procede in maniera del tutto analoga. Non c'è qui un salto con l'istruzione JSR ad una Subroutine, ma un salto tramite un piedino del CI del micro-processore in una Routine di Interrupt. Con l'istruzione JSR si effettua un salto in un sotto programma *via Software*. Mediante un Interrupt è invece possibile effettuare il salto *via Hardware*, ossia tramite un segnale digitale applicato dall'esterno al processore, verso una Subroutine digitale chiamata Routine di Interrupt (fig. 29b).

La CPU 6502 è dotata per il salto in una Routine di Interrupt di due connessioni sui piedini:

NMI-PIN = Non Maskable Interrupt

IRQ-PIN = Interrupt ReQuest

Non Maskable Interrupt significa: Interrupt Non Mascherabile. Ciò vuol dire che quando al piedino NMI della CPU è applicato un segnale digitale il processore *dove* saltare incondizionatamente ad una Routine di Interrupt. La CPU deve quindi in tal caso abbandonare il programma in corso di svolgimento e servire l'Interrupt. *Interrupt ReQuest* significa: Richiesta di Interrupt. Se al piedino IRQ della CPU è applicato un segnale digitale il processore *può* in tal caso saltare in una Routine di Interrupt. Se o meno debba effettuare il salto alla Routine di Interrupt viene stabilito in base allo stato dell'Interrupt Flag nel registro Processor Status. L'Interrupt Flag può essere messo ad 1 o a 0 con le seguenti istruzioni:

CLI, codice OP 58 I = 0 Interrupt Enable

SEI, codice OP 78 I = 1 Interrupt Disable

Se l'Interrupt Flag nel registro Processor Status è posto a 0 (I=0), la CPU accetta, tramite il relativo pin IRQ, l'Interrupt. Ossia, se al piedino IRQ è applicato un corrispondente segnale digitale, il micro-processore esegue l'Interrupt. Se invece l'Interrupt Flag è messo a 1 (I=1), il processore ignora il segnale applicato al piedino IRQ e non esegue alcun interrupt. Si suole anche dire che l'Interrupt emesso tramite il piedino IRQ della CPU viene "mascherato" dall'Interrupt Flag nel registro Processor Status. Quindi un Interrupt che viene emesso dal piedino IRQ può essere mascherato ed un Interrupt che viene emesso dal piedino NMI non può essere mascherato; oltre a ciò i due piedini di Interrupt hanno un'altra sostanziale differenza.

Il piedino NMI è sensibile ai *fronti*, mentre il piedino IRQ è sensibile al *livello* delle onde quadre applicate. Bisogna prestar bene attenzione a ciò quando si desidera, tramite uno di tali piedini, effettuare i salti in una Routine Interrupt. Che fa dunque la CPU dopo aver ricevuto ad uno dei due piedini di Interrupt un segnale di emissione di un Interrupt?

Consideriamo prima il piedino NMI. Un fronte di discesa a questo ingresso provoca un Non Maskable Interrupt. Come già sappiamo la CPU deve intervenire subito su questo Interrupt, ossia non deve considerare l'Interrupt Flag nel registro Processor Status. Dopo il passaggio del fronte negativo (a conformazione TTL) il processore viene avviato alla Routine di Interrupt. Come procede? Per avviarsi alla Routine di Interrupt il microprocessore verifica nelle celle di memoria FFFA ed FFFB in quale posizione di memoria incomincia la Routine di Interrupt. All'indirizzo FFFA si trova il byte basso dell'indirizzo di partenza della Routine di Interrupt ed all'indirizzo FFFB la parte bassa dell'indirizzo. Nel nostro esempio in queste celle troviamo i seguenti dati:

FFFA 00 byte basso di indirizzo

FFFB 20 byte alto di indirizzo

Con questi due byte il micro-processore costruisce il vettore NMI. Resta inteso che per vettore e Pointer di indirizzo intendiamo la stessa cosa. Così come nello Indirect Indexed Addressing il Pointer di indirizzo era posto in pagina 0, i vettori Interrupt sono posti al termine della pagina FF.

Il vettore NMI nel nostro esempio indica l'indirizzo 2000. A questo indirizzo troviamo il codice OP della Routine NMI. Il microprocessore elabora ora nel modo solito questa Routine. Osserviamo, a proposito, che nella Routine NMI è possibile anche effettuare salti di programma ad altre Subroutine, così come abbiamo visto che in una Subroutine potevano essere previsti salti ad una o più Subroutine. Come fa il processore a capire che la Routine di Interrupt è terminata? Similmente all'istruzione RTS, che fa rientrare la CPU dalla Subroutine nel programma principale, l'istruzione RTI riporta il processore alla giusta posizione nel programma. La RTI ha il codice OP 40.

Come già detto, l'ingresso IRQ della CPU è sensibile al livello. Se a questo ingresso si applica per un tempo definito un segnale a stato logico 0, viene emesso un Interrupt ReQuest. (Per quanto tempo il pin IRQ debba essere messo a massa affinché la CPU riconosca l'IRQ, non è per ora importante; ritorneremo altrove sull'argomento). Non appena la CPU ha riconosciuto un Interrupt ReQuest, controlla nel registro Processor Status se lo I-Flag è a 0 o ad 1. Se l'Interrupt Flag I = 0, la CPU esegue l'Interrupt, ossia salta alla Routine IRQ. L'indirizzo di partenza della Routine IRQ viene fornito al micro-processore da un vettore posto agli indirizzi FFFE ed FFFF. A questi due indirizzi nel nostro esempio troviamo i dati seguenti:

FFFE C3 byte basso di indirizzo

FFFF 24 byte alto di indirizzo

Con questi due byte il micro-processore costruisce il vettore IRQ. Il vettore IRQ indica quindi l'indirizzo 24C3. A questo indirizzo si trova posto il primo codice OP della Routine IRQ. Il processore rimane entro tale Routine finché incontra l'istruzione RTI. La RTI ri-

porta il processore nel programma principale.

Dobbiamo ancora rispondere ad una domanda che riguarda l'Interrupt: come fa la CPU dopo l'elaborazione della Routine Interrupt a rientrare al punto giusto nel programma principale? La risposta è molto semplice: ricordiamoci dell'istruzione JSR. Quando il micro-processore incontra questa istruzione mette in salvo l'indirizzo di rientro sullo Stack. Solo dopo di ciò effettua il salto alla Subroutine. Al termine della Subroutine si trova l'istruzione RTS. Questa istruzione consente alla CPU di recuperare l'indirizzo di rientro precedentemente salvato sullo Stack, per ritrovare l'indirizzo corretto nel programma principale.

In una sequenza di Interrupt si procede in maniera del tutto analoga. Quando il micro-processore riconosce un Interrupt, provvede, come nel caso dell'istruzione JSR, a salvare i byte alto e basso dell'indirizzo di rientro sullo Stack. Nel caso dell'Interrupt, oltre all'indirizzo di rientro, viene conservato sullo Stack un altro registro della CPU molto importante: il registro Processor Status. Ne consegue che dopo il rientro dalla Routine di Interrupt anche il registro Processor Status viene riportato allo stato iniziale quando il processore effettua il rientro dalla Routine di Interrupt. L'istruzione che riporta la CPU al termine della Routine di Interrupt nel programma principale è la RTI = ReTurn from Interrupt.

Riassumendo, tra l'istruzione JSR ed una richiesta di Interrupt si possono osservare le seguenti analogie:

Istruzione JSR	Richiesta di Interrupt
----------------	------------------------

JSR: codice OP20	fronte impulso negativo al pin NMI o livello logico 0 al pin IRQ
------------------	--

RTS: codice OP60	RTI: codice OP40
------------------	------------------

L'esatto svolgimento di una sequenza di Interrupt la spiegheremo nel paragrafo successivo. Qui volevamo soltanto descrivere *in generale* che cos'è un Interrupt e come il micro-processore reagisce ad una richiesta di Interrupt. Nella fig. 29 si vede un altro collegamento portato ad un piedino della CPU: RES. Questo piedino costituisce l'ingresso di Reset del microprocessore. Quando si fornisce la tensione di alimentazione allo Junior-Computer, il micro-processore non sa riconoscere da solo in quale posizione del programma Monitor deve iniziare l'elaborazione. Così come un Flip-Flop può essere rimesso a 0 con l'ingresso di Reset la CPU viene avviata nel programma Monitor. L'inizio di questo processo è simile a quello di una sequenza di Interrupt. Se l'ingresso RES assume per un determinato tempo lo stato logico 0, la CPU riconosce una richiesta di Reset. Essa allora controlla anche nelle celle di memoria FFFC ed FFFD l'indirizzo indicato dal vettore di Reset. Per lo Junior-Computer questo è sempre 1C1D. Da questo indirizzo parte il Monitor del sistema, ossia la parte di programma che gestisce la scansione del display e l'interrogazione della tastiera. Dobbiamo ancora rispondere ad un'ultima interessante domanda: come fa la CPU a ricavare dalle celle di memoria FFFA...FFFF i

vettori di Interrupt e di Reset, se lo Junior-Computer non ha una memoria per questi indirizzi? Se osserviamo il circuito dello Junior-Computer, riconosciamo facilmente che solo le linee indirizzo A0...A9 vengono impiegate per l'indirizzamento di celle di memoria nelle EPROM, RAM e PIA. I segnali CS vengono ricavati dalle linee indirizzo A10...A12, mentre le linee A13...A15 non vengono decodificate. La conformazione di bit posta sulle linee di indirizzo A13...A15 non ha quindi alcun effetto per l'indirizzamento degli elementi di memoria dello Junior-Computer. Non fa quindi differenza se la CPU ricavi il vettore di Reset dagli indirizzi FFFC, FFFD o dagli indirizzi 1FFC, 1FFD. Lo stesso vale per i vettori di Interrupt. Dato che per il modo di funzionare dello Junior-Computer non è necessario decodificare l'intero campo indirizzi, i vettori di Reset e di Interrupt sono posti nella pagina 1F della EPROM. Ciò significa che benché la CPU nel ricavare tali vettori si indirizzi alla pagina FF che non è presente, essa ricava i vettori dalla pagina 1F. Ciò è comunque esatto solo per la configurazione base dello Junior-Computer. Nell'espansione del computer si deve osservare che i citati vettori sono posti nelle corrette celle di memoria al termine del campo di memoria, ossia in pagina FF.

Descrizione dettagliata dell'IRQ e NMI

Gli Interrupt IRQ e NMI hanno tecnicamente il medesimo svolgimento. Essi differiscono fra loro per il fatto che il primo può essere mascherato e l'altro no. Vediamo l'esempio pratico in fig. 30. Ivi sono descritte alcune sezioni dell'intero campo di memoria indirizzabile della CPU 6502.

In pagina 1 è posto lo Stack. Il programma, che il processore ha in corso di svolgimento, si trova in pagina 3. I vettori mediante i quali il processore viene attivato alle Routine IRQ e NMI sono posti al termine della pagina FF. Come si vede dalla figura, il vettore NMI indica l'indirizzo 2000 ed il vettore IRQ l'indirizzo 24C3. Da questi due indirizzi partono le Routine di Interrupt. Immaginiamo ora che il processore stia elaborando il suo programma in pagina 3. Non c'è nulla di speciale: esso deve semplicemente eseguire istruzione dopo istruzione. Ad un certo punto essa arriva all'indirizzo 0343. I tale locazione la CPU ritrova un codice OP. Dalla figura vediamo che si tratta di un codice OP della lunghezza di 1 o 2 bytes senza che nulla cambi.

Supponiamo che, mentre la CPU si sta interessando del primo operando, si verifichi un NMI. C'è un Interrupt del quale la CPU deve occuparsi immediatamente. Che cosa succede adesso? La risposta è che la CPU esegue dapprima l'istruzione che inizia all'indirizzo 0343. Solo dopo essa si occupa dell'Interrupt. Quando l'istruzione è stata eseguita, il contatore di programma indica il codice OP immediatamente successivo, che sta all'indirizzo 0346. Solo ora la CPU si interessa della richiesta di NMI. La sequenza dell'Interrupt si svolge in diversi tratti che ora descriviamo:

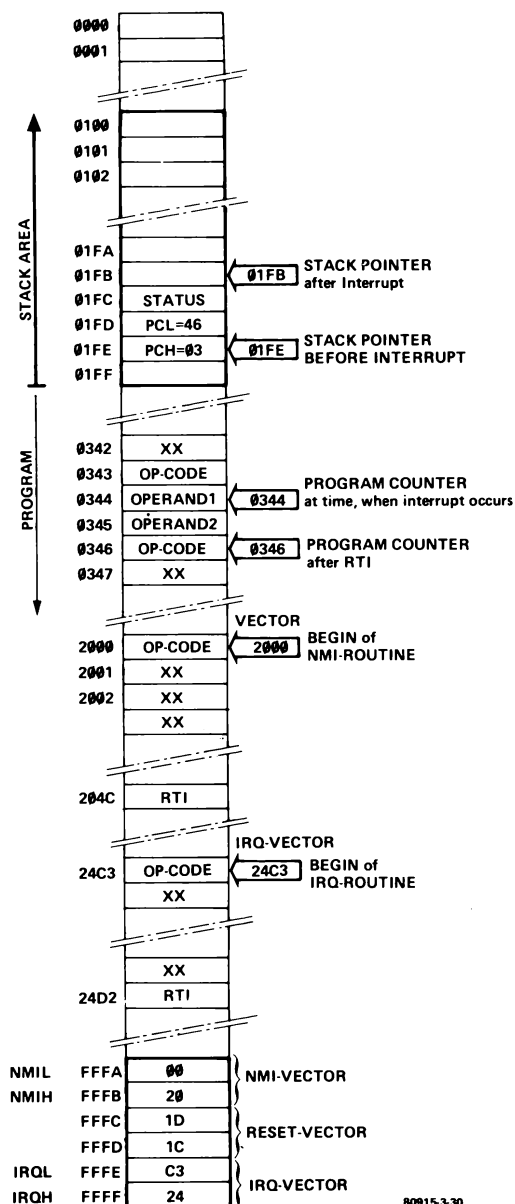


Figura 30. Su di un estratto della memoria è facile seguire l'andamento di un Interrupt. Viene anche illustrato come il contatore di programma ed il contenuto del registro Processor Stats vengono depositi sullo Stack dopo il riconoscimento dell'Interrupt.

- 1) Il micro-processore ferma il contatore di programma. Il suo contenuto al momento è l'indirizzo 0346.
- 2) Il micro-processore depone la parte alta e la parte bassa del byte del contatore di programma sullo Stack. Poiché lo Stack Pointer inizialmente indicava l'indirizzo 01FF, ora esso indirizza la cella di memoria 01FC.
- 3) Dopo aver deposto l'indirizzo di rientro della Routine di Interrupt sullo Stack, la CPU salva il contenuto del registro Processor Status sul medesimo Stack. Ora lo Stack Pointer indica l'indirizzo 01FB.
- 4) Nei passi successivi, il processore ricava dalle celle di memoria FFFA e FFFB il vettore NMI e lo trasferisce sul bus indirizzi. Viene così fissato l'indirizzo di partenza 2000 della Routine NMI.
- 5) Al termine della Routine NMI il processore incontra l'istruzione RTI. Questa istruzione lo riporta nel programma principale. Il contatore di programma ed il registro di Processor Status vengono riportati al loro stato iniziale. La sequenza di Interrupt è così conclusa.

Cosa avviene nel caso in cui l'avvio di un Interrupt proviene non da un NMI ma da un IRQ? Come già sappiamo un'istruzione IRQ può essere eseguita solo dopo aver posto a 0 lo I-Flag nel registro Processor Status. Ciò significa che lo I-Flag deve essere 0 prima che la linea IRQ venga portata al livello logico 0. La sequenza dell'IRQ decorre quindi esattamente come la sequenza precedentemente descritta del NMI. Anche in questo caso, nell'ordine, vengono deposti sullo Stack il PCH, il PCL e il contenuto del registro P. Però il vettore di Interrupt non viene ricavato dagli indirizzi FFFA e FFFB, ma dagli indirizzi FFFE e FFFF. A questi indirizzi sono infatti registrati il byte basso e quello alto del vettore IRQ. Successivamente il vettore IRQ viene come al solito trasferito sul bus indirizzi e il processore viene così avviato alla Routine IRQ.

È chiaro pertanto che durante un IRQ non può venir inizialmente un altro IRQ, poiché il micro-processore all'inizio della sequenza dell'Interrupt ha posto automaticamente lo I-Flag ad 1. Invece è possibile durante il tempo in cui la CPU elabora la Routine IRQ venga generato un NMI, in quanto un NMI deve essere immediatamente eseguito senza badare allo stato del I-Flag. In un grande elaboratore può addirittura succedere che durante una Routine IRQ vengano generate diversi NMI successivi uno dietro l'altro. Questi NMI successivi indirizzano il micro-processore tramite un vettore NMI a *diverse* Routine NMI. Come ciò funzioni, lo vedremo più avanti, quando discuteremo lo Jump Indirect, ovvero il salto a programma indiretto.

Osservazioni: analogamente al modo in cui entro una Subroutine possono effettuarsi salto a diverse altre Subroutine, in una Routine di Interrupt, mediante Interrupt successivi, possono essere effettuati salti ad altre Routine di Interrupt. Durante una Routine di

Interrupt spesso sono utilizzati i registri della CPU Accumulatore, X e Y. Poiché al riconoscimento di un Interrupt solo il registro Processor Status viene conservato automaticamente sullo Stack, anche questi altri registri debbono venire conservati via Software. È quindi bene introdurre una Routine di Interrupt nel modo seguente:

```
SAVE   PHA    save Accu on Stack
        TXA   }
        PHA   } save X-Register on Stack
        TYA   }
        PHA   } save Y-Register on Stack
```

```
        .
        .
        .
        .
```

Interruptroutine

```
        .
        .
        .
        .
```

```
RESTO  PLA   }
        TAY  } restore Y-Register
        PLA   }
        TAX  } restore X-Register
        PLA   }
        PLA   restore Accu
```

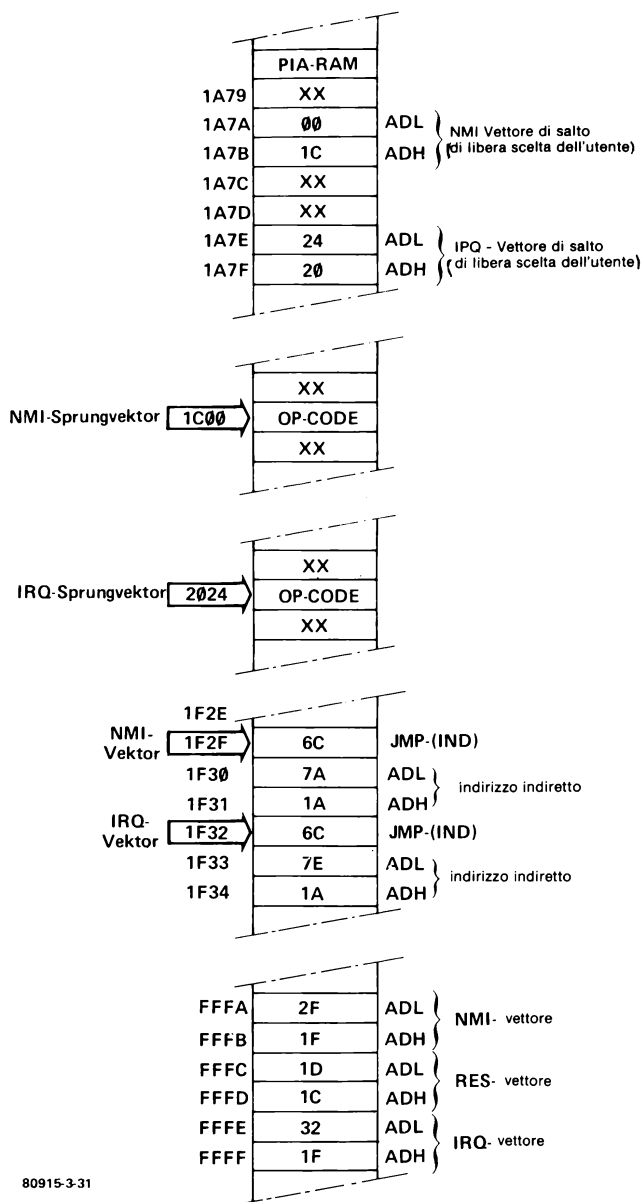
```
END    RTI    return from interrupt
```

Come mostra questa sequenza di programma, all'inizio della Routine di Interrupt tutti i registri della CPU vengono memorizzati, ed al termine riportati nuovamente allo stato iniziale. Il ripristino dello stato iniziale di tutti i registri della CPU viene effettuato prima dell'istruzione RTI. Come si è già ricordato nel corso di questo paragrafo, è possibile tramite un NMI annidare una dentro l'altra diverse Routine di Interrupt. Come è possibile, se è disponibile solo un vettore NMI depositato nelle celle di memoria FFFA ed FFFB? La soluzione sta in un salto di indirizzo indiretto. Come ciò avviene lo illustriamo nel paragrafo seguente.

Jump Indirect

Nei capitoli precedenti abbiamo mostrato come è possibile eseguire operazioni indirette di caricamento e di memorizzazione: analogamente è possibile effettuare salti nel programma in modo indiretto. Questo procedimento è mostrato in fig. 31. Agli indirizzi 1F2F e 1F32 troviamo il codice OP 6C. Questo codice OP esegue un salto a programma indiretto. La scrittura simbolica relativa è: JMP-(IND) codice OP 6C

Un salto indiretto a programma è lungo 3 byte. Ricordiamoci il salto diretto a programma: JMP codice OP 4C. Quando ad esempio si diceva JMP-1C00 oppure 4C001C, veniva effettuato un salto di-



80915-3-31

Figura 31. Se il vettore di Interrupt indica una istruzione JMP-(IND), con un unico vettore è possibile richiamare un numero qualsiasi di routine Interrupt. In questo caso il vettore diretto di Interrupt deve essere registrato nella parte finale della RAM del PIO

retto all'indirizzo 1C00, al quale si trovava un altro codice OP. Le cose vanno diversamente nel caso del salto indiretto a programma. Quando diciamo JMP-(1A7A), la CPU controlla nella cella di memoria 1A7A quale è la parte bassa del byte indirizzo dell'indirizzo di salto diretto.

Dalla cella di memoria immediatamente successiva, cioè 1A7B, essa preleva poi la parte alta del byte indirizzo dell'indirizzo di salto diretto. Detto in altro modo: nel salto indiretto il microprocessore salta a due celle di memoria successive, dalle quali preleva l'indirizzo di salto diretto. Con i 2 byte indirizzo posti in tali celle esso compone un vettore di salto. Questo vettore di salto indica l'indirizzo al quale deve essere effettuato il salto diretto. Nello Junior-Computer le celle agli indirizzi 1A00...1A7F funzionano da RAM. E il motivo c'è: nelle celle di una RAM, come è noto, è possibile modificare a volontà il contenuto. Perciò anche gli effettivi vettori di salto posti nelle celle 1A7A, 1A7B e 1A7E, 1A7F possono essere arbitrariamente modificati. Ciò è proprio quello che ci serve quando dobbiamo, tramite *un* vettore NMI, saltare in *diverse* Routine NMI con differenti indirizzi di partenza. Il programmatore deve quindi curare che:

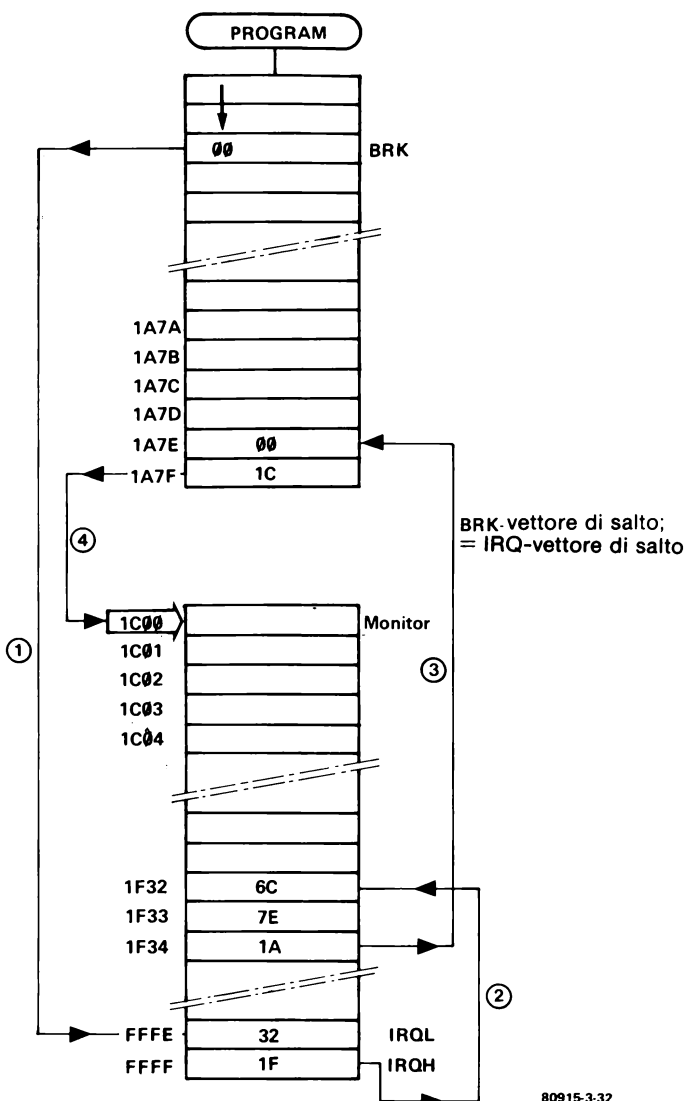
- prima che venga generato un NMI, nelle celle di memoria 1A7A, 1A7B deve essere memorizzato un vettore di salto. Questo vettore indica l'indirizzo di partenza di una Routine Interrupt. È a questo indirizzo che viene condotto il processore quando viene attivato lo NMI.
- Il vettore NMI deve sempre indicare un codice OP del tipo JMP-(IND) quando mediante *un* NMI o IRQ si deve saltare a *diverse* Routine Interrupt che iniziano a differenti indirizzi.

Descrivendo la programmazione della PIA mostreremo, sulla base di un esempio pratico, come si possa modificare a volontà da programma il vettore NMI.

Osserviamo in conclusione che un JMP-(IND) non deve essere utilizzato introdurre un tale tipo di salto in un programma al posto di un'istruzione di salto diretto con il codice OP 4C. Ciò risulta vantaggioso quando si controllano grossi programmi, perché in tal modo il programmatore può imprimere il suo programma in una EPROM senza conoscere preventivamente gli indirizzi di salto diretto. In tal modo è possibile depositare gli indirizzi di salto diretto in celle della RAM in via sperimentale e successivamente modificarli secondo necessità. Dopo il controllo finale del programma nella EPROM, le istruzioni JMP-(IND) codice OP possono venir sostituite da normali istruzioni JMP, dato che sia i salti diretti che quelli indiretti hanno entrambi una lunghezza di 3 byte.

Il comando BRK

Sinora abbiamo descritto solo degli Interrupt che vengono generati esternamente mediante segnali logici. Abbiamo cioè applica-



80915-3-32

Figura 32. Il comando BRK è la versione in Software di un IRQ. Nella versione standard dello Junior-Computer, il vettore BRK (= vettore IRQ) deve sempre indicare l'indirizzo 1C00. A tale indirizzo nel Monitor è presente una Save-routine che conserva il contenuto di tutti i registri della CPU e successivamente salta alla routine tastiera/display.

to segnali TTL agli ingressi NMI oppure IRQ. Abbiamo detto che il salto ad una Routine di Interrupt avviene via Hardware collegata allo Junior-Computer. Con la CPU 6502 è tuttavia anche possibile generare un Interrupt via Software. L'istruzione che attiva in tale Interrupt è il comando BRK. Il suo codice OP è: BRK—00. (BRK = BReak = interruzione).

Quando il micro-processore incontra un'istruzione BRK, esso reagisce nell'identico modo che se gli fosse stato applicato esternamente un segnale IRQ, dato che il comando BRK lavora insieme al vettore IRQ.

Se il micro-processore incontra il codice OP 00, esso verifica nelle celle di memoria FFFE ed FFFF a quale punto del programma debba saltare. L'istruzione BRK esegue quindi un salto indiretto a programma. La differenza con il salto indiretto a programma JMP-(IND) con il codice operativo 6C è la seguente:

Codice OP	Istruzione	Lunghezza	Vettore di salto	Flag
6C	JMP-(IND)	3 byte	arbitrario	nessuno
00	BRK	1 byte	vettore IRQ	B-Flag

La fig. 32 illustra con un esempio pratico il comando BRK. Il vettore IRQ, che lavora insieme al comando BRK, indica l'indirizzo 1F32. Ivi è posto il codice OP 6C che genera un JMP-(IND). Questo salto indiretto a programma porta il micro-processore agli indirizzi 1A7E ed 1A7F. Nello Junior-Computer questi due indirizzi sono le ultime due celle di memoria nella RAM del PIA. Se il programmatore ha deposto colà l'indirizzo 1C00, il comando BRK invia il processore al programma Monitor dello Junior-Computer. I programmi dimostrativi presentati in questo capitolo sono stati spesso terminati con l'istruzione BRK. La fig. 32 chiarisce come questo Stop di programma funzioni. Il vettore IRQ nel Monitor dello Junior-Computer riferisce ad un'istruzione JMP-(IND). Dato che gli indirizzi indiretti sono celle della RAM in cui sono posti gli indirizzi di salto diretto, è possibile con un solo vettore Interrupt effettuare salti a numerosi punti del programma. Il vettore IRQ, come si sa, è posto nelle celle 1A7E e 1A7F della RAM. Queste due celle sono gli indirizzi indiretti in cui è posto l'effettivo vettore IRQ. Poiché il vettore IRQ è posto in celle della RAM, è possibile modificare tale vettore a mano o per Software. Se trasferiamo questo concetto al comando BRK, possiamo dire che con un unico comando BRK si può saltare a diverse Routine BRK, dato che anche il comando BRK impiega il vettore IRQ.

Siamo così giunti al termine del 3° capitolo. Abbiamo considerato tutti i tipi di indirizzamento della CPU 6502; nella parte finale abbiamo forse esagerato con la teoria, e presentato pochi programmi dimostrativi. Il lettore sarà tuttavia compensato nel capitolo successivo. Troverà in questo aggiunte al capitolo 3 e molti programmi dimostrativi, che certamente troverà interessanti.

Suggerimenti e programmi dimostrativi

Prima di descrivere qualche altro programma, diamo una serie di suggerimenti per inserire programmi nello Junior-Computer:

- 1) Prima di inserire con la tastiera un programma nello Junior-Computer, è bene preparare inizialmente una Flow Chart grezza (globale), e quindi una più dettagliata. Specie nel caso di programmi di grandi dimensioni il diagramma di flusso globale è molto importante, poiché con esso si può, per così dire, tradurre il problema in parole.
- 2) Prima di introdurre programmi con salti condizionati, è necessario calcolare i relativi Offset. A tal fine si può impiegare convenientemente la Subroutine BRANCH (indirizzo di partenza 1FD5) al termine del programma Monitor. A tale scopo è sufficiente introdurre la parte bassa del byte indirizzo nel computer.
- 3) I programmi vengono svolti correttamente solo se essi sono stati correttamente introdotti nella memoria dello Junior-Computer. Abbiamo a disposizione i seguenti indirizzi di memoria nel modello standard dello Junior-Computer:
una RAM da un K con gli indirizzi 0000...03FF. Rappresentano 4 pagine da 256 byte l'una:

pagina 00 : 0000...00FF

pagina 01 : 0100...01FF

pagina 02 : 0200...02FF

pagina 03 : 0300...03FF

In questo campo di memoria non si vedono impiegare i seguenti indirizzi che sono riservati al Monitor di servizio:

in pagina 00 : le celle 00E1...00FF

in pagina 01 : le celle 01AF3...01AFF

Le ultime 13 celle di memoria in pagina 1 sono riservate allo System Stack. In pagina 1A è posto il PIA con il Timer. Ivi sono pure liberamente disponibili 128 byte di RAM : 1A00...1A7F. Però gli indirizzi 1A7A, 1A7B nonché uno 1A7F sono previsti per il vettore NMI, rispettivamente IRQ, e non dovrebbero quindi essere utilizzati dal programmatore in un programma.

- 4) Dopo il lancio del programma Monitor (con il tasto RST), il programmatore dovrebbe sempre caricare nelle celle di memoria 1A7A, 1A7B nonché 1A7E, 1A7F i vettori IRQ e NMI. Così lo Junior-Computer è sempre predisposto per lo "Step by Step Mode" ed il comando BRK:

Tasten				Adresse	Daten	
RST				xxxx	xx	
AD				xxxx	xx	
0	2	0	0	0200	xx	
DA		2	0	0200	20	JSR- CLEAR1
+		9	4	0201	94	ADL di CLDISP
+		0	2	0202	02	ADH di CLDISP
+		2	0	0203	20	JSR-
+		8	2	0204	82	ADL di CLB1
+		0	2	0205	02	ADH di CLB1
+		2	0	0206	20	JSR-
+		8	8	0207	8B	ADL di CLB2
+		0	2	0208	02	ADH di CLB2
+		2	0	0209	20	JSR- FIRST
+		6	F	020A	6F	ADL di KEYDIS
+		0	2	020B	02	ADH di KEYDIS
+		C	9	020C	C9	CMP #
+		1	0	020D	10	con 10
+		F	0	020E	F0	BEQ
+		F	0	020F	F0	F0 passi più indietro è posto CLEAR 1
+		C	9	0210	C9	CMP #
+		1	2	0211	12	con 12
+		F	0	0212	F0	BEQ
+		0	6	0213	06	(Offset) 06 passi più avanti è situato PLUS
+		2	0	0214	20	JSR-
+		4	9	0215	49	ADL di SHIFT
+		0	2	0216	02	ADH di SHIFT
+		4	C	0217	4C	JMP- (direttamente)
+		0	9	0218	09	ADL di FIRST
+		0	2	0219	02	ADH di FIRST
+		2	0	021A	20	JSR- PLUS
+		A	A	021B	AA	ADL di STO1
+		0	2	021C	02	ADH di STO1
+		2	0	021D	20	JSR-
+		9	4	021E	94	ADL di CLDISP
+		0	2	021F	02	ADH di CLDISP
+		2	0	0220	20	JSR- SECOND
+		6	F	0221	6F	ADL di KEYDIS
+		0	2	0222	02	ADH di KEYDIS
+		C	9	0223	C9	CMP #
+		1	0	0224	10	con 10
+		F	0	0225	F0	BEQ
+		0	A	0226	0A	0A passi più avanti (Offset) è situato CLEAR2
+		C	9	0227	C9	CMP #
+		1	1	0228	11	con 11
+		F	0	0229	F0	BEQ

+	0	C	022A	0C	(Offset); 0C passi più avanti è situato EQUAL
+	2	0	022B	20	JSR-
+	4	9	022C	49	ADL di SHIFT
+	0	2	022D	02	ADH di SHIFT
+	4	C	022E	4C	JMP-
+	2	0	022F	20	ADL di SECOND
+	0	2	0230	02	ADH di SECOND
+	2	0	0231	20	JSR- CLEAR2
+	9	4	0232	94	ADL di CLDISP
+	0	2	0233	02	ADH di CLDISP
+	4	C	0234	4C	JMP-
+	2	0	0235	20	ADL di SECOND
+	0	2	0236	02	ADH di SECOND
+	2	0	0237	20	JSR- EQUAL
+	9	D	0238	9D	ADL di STO2
+	0	2	0239	02	ADH di STO2
+	2	0	023A	20	JSR-
+	5	9	023B	59	ADL di ADD
+	0	2	023C	02	ADH di ADD
+	2	0	023D	20	JSR-
+	B	7	023E	B7	ADL di RESDIS
+	0	2	023F	02	ADH di RESDIS
+	2	0	0240	20	JSR-
+	8	2	0241	82	ADL di CLB1
+	0	2	0242	02	ADH di CLB1
+	2	0	0243	20	JSR-
+	8	B	0244	8B	ADL di CLB2
+	0	2	0245	02	ADH di CLB2
+	4	C	0246	4C	JMP-
+	0	9	0247	09	ADL di FIRST
+	0	2	0248	02	ADH di FIRST
+	A	0	0249	A0	LDY #; Subroutine SHIFT
+	0	4	024A	04	04 nel registro indice Y
+	0	6	024B	06	ASLZ SHIFT1
+	F	9	024C	F9	INH con l'indirizzo 00F9
+	2	6	024D	26	ROLZ
+	F	A	024E	FA	POINTL con l'indirizzo 00FA
+	2	6	024F	26	ROLZ
+	F	B	0250	FB	POINTH con l'indirizzo 00FB
+	8	8	0251	88	DEY - decrementa Y di 1
+	D	0	0252	D0	BNE
+	F	7	0253	F7	F7 passi più indietro è situato SHIFT1
+	0	5	0254	05	ORAZ
+	F	9	0255	F9	svolge la funzione OR su INH bit per bit
+	8	5	0256	85	STAZ
+	F	9	0257	F9	Contenuto dell'Accu. a INH (indirizzo 00F9)
+	6	0	0258	60	RTS
+	F	8	0259	F8	SED calcola in decimale Subroutine ADD
+	1	8	025A	18	CLC
+	A	5	025B	A5	LDAZ
+	0	0	025C	00	ADL di B10 (indirizzo 0000); B10 nell'Accu.
+	6	5	025D	65	ADCZ

+	0	3	025E	03	ADL di B20 (Indirizzo 0003); Accu = B10 + B20
+	8	5	025F	85	STAZ
+	0	6	0260	06	ADL di R0; Accu → R0
+	A	5	0261	A5	LDAZ
+	0	1	0262	01	ADL di B11 (indirizzo 0001); B11 nell'accumulatore
+	6	5	0263	65	ADCZ
+	0	4	0264	04	ADL di B21 (indirizzo 0004); Accu = B11 + B21
+	8	5	0265	85	STAZ
+	0	7	0266	07	ADL di R1 (indirizzo 0007); Accu → R1
+	A	5	0267	A5	LDAZ
+	0	2	0268	02	ADL di B12 (indirizzo 0002); B12 in Accu
+	6	5	0269	65	ADCZ
+	0	5	026A	05	ADL di B22 (indirizzo 0005); Accu = B12 + B22
+	8	5	026B	85	STAZ
+	0	8	026C	08	ADL di R2 (indirizzo 0008); Accu → R2
+	D	8	026D	D8	CLD calcolare in binario!
+	6	0	026E	60	RTS
+	2	0	026F	20	JSR-Subroutine KEYDIS
+	8	E	0270	8E	ADL di SCANDS } nel Monitor
+	1	D	0271	1D	ADH di SCANDS } (1D8E)
+	D	0	0272	D0	BNE
+	F	B	0273	FB	(offset) FB passi più indietro è posto KEYDIS
+	2	0	0274	20	JSR- KD
+	8	E	0275	8E	ADL di SCANDS } nel Monitor
+	1	D	0276	1D	ADH di SCANDS } (1D8E)
+	F	0	0277	F0	BEQ
+	F	B	0278	FB	(offset); FB passi più indietro è posto KD
+	2	0	0279	20	JSR-
+	8	E	027A	8E	ADL di SCANDS } nel Monitor
+	1	D	027B	1D	ADH di SCANDS } (1D8E)
+	F	0	027C	F0	BEQ
+	F	6	027D	F6	(offset); F6 passi più indietro è posto KD
+	2	0	027E	20	JSR-
+	F	9	027F	F9	ADL di GETKEY } nel Monitor
+	1	D	0280	1D	ADH di GETKEY } (1DF9)
+	6	0	0281	60	RTS
+	A	9	0282	A9	LDA #; Subroutine CLB1
+	0	0	0283	00	00 → Accu
+	8	5	0284	85	STAZ
+	0	0	0285	00	Accu → B10 (= 00)
+	8	5	0286	85	STAZ
+	0	1	0287	01	00 → B11
+	8	5	0288	85	STAZ
+	0	2	0289	02	00 → B12
+	6	0	028A	60	RTS
+	A	9	028B	A9	LDA #; Subroutine CLB2
+	0	0	028C	00	00 → Accu
+	8	5	028D	85	STAZ

+	0	3	028E	03	.00 → B20
+	8	5	028F	85	STAZ
+	0	4	0290	04	00 → B21
+	8	5	0291	85	STAZ
+	0	5	0292	05	00 → B22
+	6	0	0293	60	RTS
+	A	9	0294	A9	LDA #; Subroutine CLDISP
+	0	0	0295	00	00 → Accu
+	8	5	0296	85	STAZ
+	F	9	0297	F9	00 → INH (indirizzo 00F9)
+	8	5	0298	85	STAZ
+	F	A	0299	FA	00 → POINTL (indirizzo 00FA)
+	8	5	029A	85	STAZ
+	F	B	029B	FB	00 → POINTH (indirizzo 00FB)
+	6	0	029C	60	RTS
+	A	5	029D	A5	LDAZ; Subroutine STO2
+	F	9	029E	F9	INH (indirizzo 00F9) → Accu
+	8	5	029F	85	STAZ
+	0	3	02A0	03	Accu (= INH) → B20 (indirizzo 0003)
+	A	5	02A1	A5	LDAZ
+	F	A	02A2	FA	POINTL (indirizzo 00FA) → Accu
+	8	5	02A3	85	STAZ
+	0	4	02A4	04	Accu (= POINTL) → B21 (indirizzo 0004)
+	A	5	02A5	A5	LDAZ
+	F	B	02A6	FB	POINTH (indirizzo 00FB) → Accu
+	8	5	02A7	85	STAZ
+	0	5	02A8	05	Accu (= POINTH) → B22 (indirizzo 0005)
+	6	0	02A9	60	RTS
+	A	5	02AA	A5	LDAZ; Subroutine STO1
+	F	9	02AB	F9	INH (indirizzo 00F9) → Accu
+	8	5	02AC	85	STAZ
+	0	0	02AD	00	Accu (= INH) → B10 (indirizzo 0000)
+	A	5	02AE	A5	LDAZ
+	F	A	02AF	FA	POINTL (indirizzo 00FA) → Accu
+	8	5	02B0	85	STAZ
+	0	1	02B1	01	Accu (= POINTL) → B11 (indirizzo 0001)
+	A	5	02B2	A5	LDAZ
+	F	B	02B3	FB	POINTH (indirizzo 00FB) → Accu
+	8	5	02B4	85	STAZ
+	0	2	02B5	02	Accu (= POINTH) → B12 (indirizzo 0002)
+	6	0	02B6	60	RTS
+	A	5	02B7	A5	LDAZ; Subroutine RESDIS
+	0	6	02B8	06	R0 (indirizzo 0006) → Accu
+	8	5	02B9	85	STAZ
+	F	9	02BA	F9	Accu (= R0) → INH (indirizzo 00F9)
+	A	5	02BB	A5	LDAZ
+	0	7	02BC	07	R1 (indirizzo 0007) → Accu
+	8	5	02BD	85	STAZ
+	F	A	02BE	FA	Accu (= R1) → POINTL (indirizzo 00FA)
+	A	5	02BF	A5	LDAZ

+	0	8	02C0	08	R2 (indirizzo 0008) → Accu
+	8	5	02C1	85	STAZ
+	F	B	02C2	FB	Accu (= R2) → POINTH (indirizzo 00FB)
+	6	0	02C3	60	RTS

Figura 1. Impostazione da tastiera del programma di addizione decimale del capitolo 3. Questo programma occupa 196 byte. Il diagramma di flusso dettagliato è illustrato nelle figg. 20 e 21a...21i nel capitolo 3.

RST		AD		XXXX XX
1	A	7	A	1A7A XX
DA		0	0	1A7A 00
+		1	C	1A7B 1C
+				1A7C XX
+				1A7D XX
+		0	0	1A7E 00
+		1	C	1A7F 1C

Ora non troviamo più ostacoli per l'inserimento del programma di addizione del cap. 3 (figure 20 e 21) nello Junior-Computer. L'indirizzo di partenza di tale programma è 0200, la digitatura da tastiera del programma è mostrata in figura 1. Nella figura 2 sono illustrati i principali Label e sezioni del programma:

Adresse 0200 ... 0248: programma principale, vedi fig. 20;
 Adresse 0249 ... 0258: Subroutine SHIFT dalla fig. 21a
 Adresse 0259 ... 026E: Subroutine ADD dalla fig. 21b
 Adresse 026F ... 0281: Subroutine KEYDIS dalla fig. 21c
 Adresse 0282 ... 028A: Subroutine CLB1 dalla fig. 21d
 Adresse 028B ... 0293: Subroutine CB2 dalla fig. 21e
 Adresse 0294 ... 029C: Subroutine CLDISP dalla fig. 21f
 Adresse 029D ... 02A9: Subroutine STO2 dalla fig. 21g
 Adresse 02AA ... 02B6: Subroutine STO1 dalla fig. 21h
 Adresse 02B7 ... 02C3: Subroutine RESDIS dalla fig. 21i

Dobbiamo ancora calcolare gli Offset dei salti condizionati del programma. A tal fine inizializziamo ancora la Subroutine BRANCH con l'indirizzo 1FD5 e inseriamo la parte bassa dell'indirizzo di partenza e di arrivo del salto:

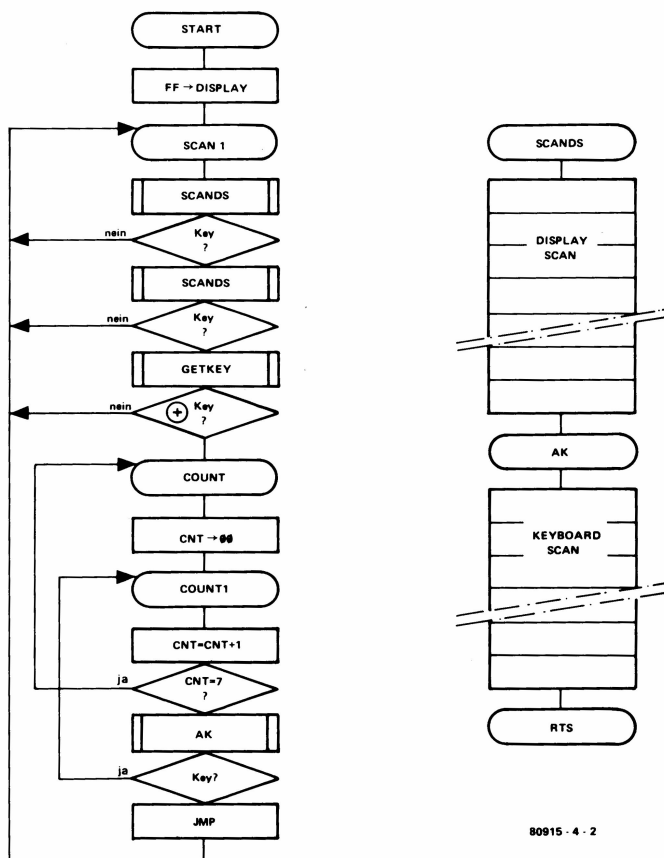
RST		AD		XXXX XX
1	F	D	5	1FD5 D8
GO				0000 00
0	E	0	0	0E00 F0 F0 all'indirizzo 020F
1	2	1	A	121A 06 06 all'indirizzo 0213
2	5	3	1	2531 0A 0A all'indirizzo 0226
2	9	3	7	2937 0C 0C all'indirizzo 022A
5	2	4	B	524B F7 F7 all'indirizzo 0253

7	2	6	F	726F	FB	FB	all'indirizzo	0273
7	7	7	4	7774	FB	FB	all'indirizzo	0278
7	C	7	4	7C74	F6	F6	all'indirizzo	027D

RST

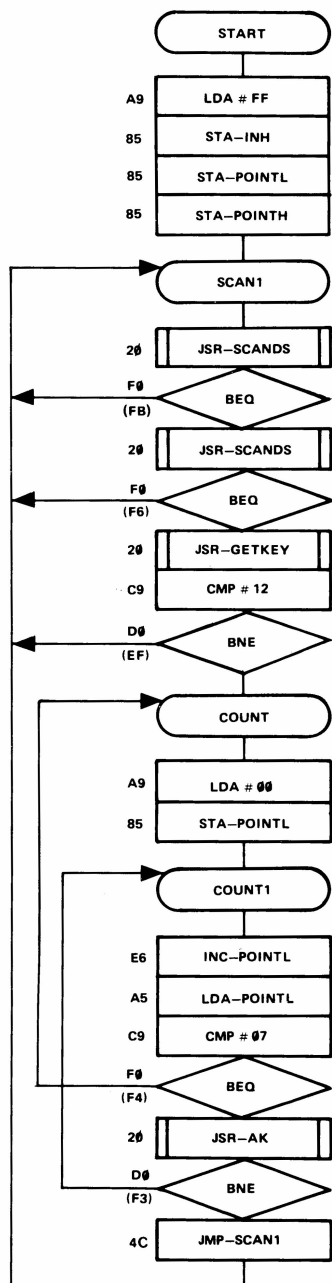
Osserviamo a proposito che premendo un qualsiasi tasto di funzione durante lo svolgimento della Routine BRANCH si mette il display a 000000 (= Reset).

Dopo aver introdotto con la tastiera l'intero programma di addizione nello Junior-Computer, possiamo farlo partire con il tasto GO. Subito dopo il computer è pronto per l'introduzione dei numeri. Mostriamo con un esempio come si svolge l'addizione di alcuni numeri:





80915 - 4 - 2

Figura 2. Diagramma di flusso globale del "lancio del dado". In questo programma vengono richiamate diverse Subroutine del programma Monitor.



80915 - 4 - 3

Figura 3. Diagramma di flusso dettagliato del "lancio del dado".

AD	0 2 0 0		introdurre indirizzo di partenza;
GO	000000		inizio programma;
2	4 5 6	002456	1° numero;
+	1 3 2	000000	+
4	1 3 2	004132	2. numero
DA	(= )	006588	risultato
AD	(= )	000000	Reset
1	9 8 5 3 1	198531	1. numero
+		000000	+
8	3 2 7 0 2	832702	2. numero
DA		031233	risultato
AD		000000	risultato

La Routine principale del programma di addizione è disposta in modo che si possono immediatamente correggere numeri eventualmente digitati in modo errato. Basta premere il tasto CLEAR. Con il tasto CLEAR l'ultimo numero introdotto viene cancellato, e lo Junior-Computer è pronto ad accogliere un nuovo numero.

Giochiamo a dadi con lo Junior-Computer

Dopo aver imparato come inserire nei programmi le Subroutine SCANDS, GETKEY e AK non sarà difficile "abusare" dello Junior-Computer impiegandolo come un dado. Dobbiamo tuttavia fornire prima un paio di chiarimenti. È noto che un dado ha 6 facce. Perciò con un contatore di Software che sappia contare da 1 a 6 si possono rappresentare tutte le facce del dado. Il "rotolamento" del dado lo avvieremo col tasto +. Sinché questo tasto rimane premuto, il display rimarrà spento solo dopo il rilascio del tasto dovrà comparire FFXXFF sul display, dove XX rappresenta il risultato del lancio del dado.

In fig. 2 mostriamo la struttura del programma per il lancio del dado. Innanzitutto i Buffer di display vengono caricati con \$FF e presentati, mediante la Subroutine SCANDS, sul display. Questa Subroutine verifica nel solito modo lo stato della tastiera, controllando se un tasto è stato premuto. Se un tasto risulta attivo, la Subroutine GETKEY decodifica il tasto. Dopo il riconoscimento del tasto attivo, il programma controlla se il tasto + è premuto. Viene accettato solo questo tasto, ed ogni altro tasto ignorato.

Il contatore via Software che rappresenta il dado inizia al Label COUNT. Inizialmente il programma pone il contenuto del contatore a 0, per poi immediatamente incrementarlo. In un ciclo di correzione, il micro-computer controlla se il contenuto del contatore ha già raggiunto il valore di 7. Se si è raggiunto questo valore non ammesso, il programma riporta il contatore ad 1, e ricomincia un nuovo ciclo di conteggio.

Ora si deve solo stabilire quanto a lungo deve durare il conteggio.

Tasten				Adresse Daten	
RST		AD		xxxx	xx
0	2	0	0	0200	xx
DA		A	9	0200	A9 A9 LDA # START
+		F	F	0201	FF FF → Accu
+		8	5	0202	85 STA Z
+		F	9	0203	F9 FF → INH (00F9)
+		8	5	0204	85 STA Z
+		F	A	0205	FA FF → POINTL (00FA)
+		8	5	0206	85 STA Z
+		F	B	0207	FB FF → POINTH (00FB)
+		2	0	0208	20 JSR- SCAN1
+		8	E	0209	8E ADL } di SCANDS (Monitor)
+		1	D	020A	1D ADH } (indirizzo 1D8E)
+		F	0	020B	F0 BEQ
+		F	B	020C	FB (Offset); FB passi più indietro per SCAN1
+		2	0	020D	20 JSR-
+		8	E	020E	8E ADL } di SCANDS (Monitor)
+		1	D	020F	1D ADH } (indirizzo 1D8E)
+		F	0	0210	F0 BEQ
+		F	6	0211	F6 F6 passi più indietro per SCAN1
+		2	0	0212	20 JSR-
+		F	9	0213	F9 ADL } di GETKEY (Monitor)
+		1	D	0214	1D ADH } (indirizzo 1DF9)
+		C	9	0215	C9 CMP #
+		1	2	0216	12 con 12
+		D	0	0217	D0 BNE
+		E	F	0218	EF EF passi più indietro per SCAN1
+		A	9	0219	A9 LDA # COUNT
+		0	0	021A	00 00 → Akku
+		8	5	021B	85 STA Z
+		F	A	021C	FA 00 → POINTL (00FA)
+		E	6	021D	E6 INC Z COUNT1
+		F	A	021E	FA POINTL + 1 → POINTL
+		A	5	021F	A5 LDA Z
+		F	A	0220	FA POINTL → Accu
+		C	9	0221	C9 CMP #
+		0	7	0222	07 con 07
+		F	0	0223	F0 BEQ
+		F	4	0224	F4 F4 passi più indietro per COUNT
+		2	0	0225	20 JSR-
+		B	1	0226	B1 ADL } di AK (Monitor)
+		1	D	0227	1D ADH } (indirizzo 1DB1)

+		D	0	0228	D0	BNE	
+		F	3	0229	F3	F3 passi più indietro per COUNT1	
+		4	C	022A	4C	JMP-	
+		0	8	022B	08	ADL	} di SCAN 1
+		0	2	022C	02	ADH	
AD							
0	2	0	0	0200	A9	Indirizzo iniziale	
GO						Inizio programma	
+				FF04	FF	Il dado è tratto	
+				FF01	FF		
+				FF06	FF		
+				FF02	FF		
usw.							

Figura 4. Schema dell'introduzione da tastiera del programma del "lancio del dado" di fig. 3.

Sin quando il tasto + resta premuto il conteggio deve andare avanti e - come già richiesto - il display rimane spento. A tal fine viene richiamata la Subroutine AK. Questa Subroutine si collega direttamente alla Subroutine SCANDS e verifica se un tasto risulta premuto. La AK *non* provvede a pilotare il display a 7 segmenti. La fig. 2 mostra come le Subroutine SCANDS e AK sono collegate l'una all'altra nel Monitor Junior-Computer.

Ma torniamo al nostro dado. Sin quando il tasto + resta premuto, la CPU permane nel ciclo di conteggio COUNT. Dato che in questo ciclo di programma viene richiamata solo la Subroutine AK, il display resta spento. Solo quando si rilascia il tasto + il microprocessore salta a SCAN1. Qui ritrova la Subroutine SCANDS e mostra quindi sul display il numero risultato del lancio.

La fig. 3 presenta il diagramma di flusso dettagliato del programma "lancio del dado". Gli indirizzi di partenza delle singole Subroutine sono pure chiaramente indicati. Il programma parte ancora una volta all'indirizzo 0200, e può essere facilmente introdotto nello Junior-Computer tramite la tastiera (fig. 4).

Considerando i programmi sin qui scritti, a prima vista essi ci sembrano privi di senso! È molto più facile ed economico realizzare un dado "digitale" con un paio di CI TTL o MOS. Non c'è proprio bisogno di ricorrere ad un computer. Ma non è questo il punto. I programmi che abbiamo descritto hanno esclusivamente un carattere didattico.

Osservazione: le Subroutine semplificano i programmi principali. È facile in questo modo seguire l'andamento del programma principale, e introdurre successivamente in modo semplice le adatte modifiche.

La Subroutine LENACC (calcolo della lunghezza del codice OP)

La Subroutine LENACC (fig. 6) la si ritrova in ogni Assembler o Disassembler. Vedremo nel secondo volume che cosa intendiamo con Assembler o Disassembler. Qui basti sapere che la scrittura di un Assembler o Disassembler richiede molta esperienza e pratica di programmazione. Perché presentiamo allora la Subroutine LENACC? È semplice: per mostrare che con le conoscenze acquisite del capitolo 3 si possono già scrivere o seguire programmi di una certa complessità.

Che cosa realizza la Subroutine LENACC? Essa converte un qualsiasi codice OP posto nell'Accu della CPU in una larghezza in byte e memorizza l'informazione nelle celle di memoria BYTES. Come già si sa i singoli codici della CPU 6502 possono suddividersi in 3 gruppi:

1. codici OP lunghi *un* byte
2. codici OP lunghi *due* byte
3. codici OP lunghi *tre* byte

		cifra esadecimale destra							
		0	1	2	3	4	5	6	7
cifra decimale sinistra	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)	
	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 =	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 =	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 =	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)	

		cifra esadecimale destra							
cifra decimale sinistra;	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)			LDA ABS (3)	LDX ABS (3)		A
	B CLV (1)	LDA ABS,Y (3)	TSX (1)			LDY ABS,X (3)	LDX ABS,X (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)		F

Figura 5. Tabella dei codici OP della CPU 6502. L'intestazione delle colonne, ossia il Nibble basso del codice OP ha particolare importanza per il programma LENACC. In questa routine il codice OP viene convertito nella lunghezza di una istruzione.

Il codice OP20 = JSR ha la lunghezza di tre byte. Quando carichiamo questo codice OP nell'Accu e successivamente richiamiamo la Subroutine LENACC, il micro-processore "calcola" tramite questa Subroutine la lunghezza del codice OP20 che è di tre byte. Scrivendo nel solito modo, il tutto ha la forma seguente:

```
LDA # 20          carica nell'accumulatore JSR-codice OP
JSR LENACC        calcola la lunghezza dell'istruzione JSR
BRK               interrompi il programma
```

Prima del richiamo della Subroutine LENACC il contenuto delle celle di memoria BYTES è qualsiasi (BYTES = XX = Don't Care). Al termine ed al rientro dalla Subroutine, nelle celle BYTES sta la lunghezza del codice operativo 20 : BYTES = 03.

Se avessimo nell'Accu un altro codice OP, ad esempio A9 = LDA #, prima di saltare alla Subroutine LENACC, nelle celle di memoria BYTES troveremmo 02, dato che il codice OP A9 ha la lunghezza di due byte.

In fig. 5 sono riassunti in una tabella tutti i codici operativi della CPU 6502. A prima vista sembra quasi impossibile riconoscere da questa tabella la lunghezza dei diversi codici OP. Ma consideriamo la prima colonna della tabella dei codici OP. I singoli codici OP di questa tabella giacciono tra 00 F0. Se riscriviamo questa colonna più in dettaglio otteniamo il seguente risultato:

codice OP simbolo mnemonico lunghezza

00	BRK	1 Byte
10	BPL	2 Bytes
20	JSR	3 Bytes
30	BMI	2 Bytes
40	RTI	1 Byte
50	BVC	2 Bytes
60	RTS	1 Byte
70	BVS	2 Bytes
90	BCC	2 Bytes
A0	LDY#	2 Bytes
B0	BCS	2 Bytes
C0	CPY#	2 Bytes
D0	BNE	2 Bytes
E0	CPX#	2 Bytes
F0	BEQ	2 Bytes

Come ora si vede facilmente, nella prima colonna della tabella dei codici OP sono riportate istruzioni lunghe 1, 2 o 3 byte. Poiché la frequenza delle istruzioni lunghe 2 byte è preponderante, possiamo dire che le istruzioni lunghe 1 o 3 byte in questa tabella costituiscono l'eccezione! Sono queste eccezioni che dobbiamo "filtrare" tramite la Subroutine LENACC. Torneremo comunque più tardi su questo argomento.

Nella seconda colonna della tabella dei Codici Operativi, troviamo 16 istruzioni della CPU 6502 elencate una sotto l'altra. Tutte le

istruzioni di questa colonna hanno per modo di indirizzamento le Indirect Indexed, Y oppure Indexed Indirect, X. Come abbiamo appreso dal capitolo 3, tutte le istruzioni che si servono di questa modalità di indirizzamento hanno la lunghezza di 2 byte. Possiamo dunque stabilire che i codici OP il cui Nibble basso vale "1" hanno sempre una lunghezza di 2 byte. Le istruzioni nella secon-

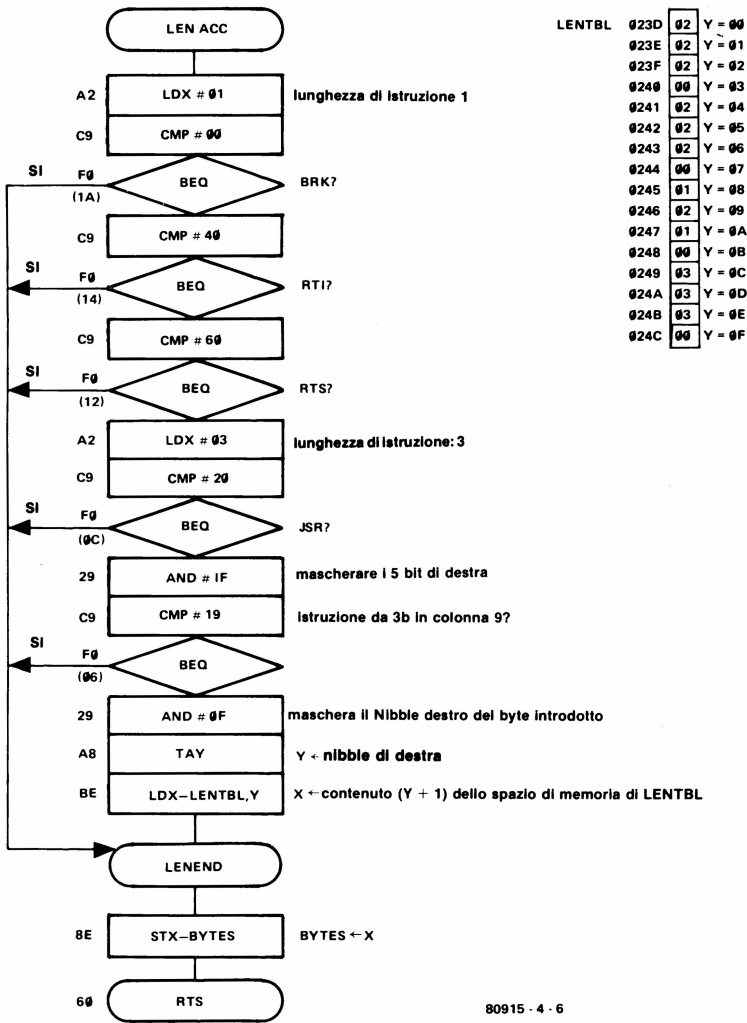


Figura 6. La Subroutine LENACC calcola la lunghezza di ogni istruzione della CPU 6502. Un'istruzione può essere lunga 1, 2 o 3 byte. Istruzioni illegali vengono sempre convertite a lunghezza 00.

da colonna hanno i seguenti codici OP: 01, 11, 21, 31, 41, 51, 61, 71, 81, 91, A1, B1, C1, E1, F1. Nella terza colonna della tabella dei codici OP troviamo solo una istruzione: LDY #. Il relativo codice OP è A2. Possiamo quindi stabilire che i codici OP il cui Nibble basso è "2" hanno una lunghezza di 2 byte. Nelle colonne 4, 8 e 16 non troviamo Codici Operativi. Nella 5ª colonna si trovano i Codici Operativi che hanno indirizzamento in pagina 0 o Zero Page, X Addressing. Come sappiamo dal capitolo 3, tutte le istruzioni che si servono di questo tipo di indirizzamento hanno una lunghezza di 2 byte. Possiamo quindi stabilire: i codici OP il cui Nibble basso è "4" hanno una lunghezza di 2 byte.

Le istruzioni nella 5ª colonna hanno i seguenti codici OP: 24, 84, 94, A4, B4, C4, E4.

Nella 6ª colonna troviamo ancora 16 istruzioni della CPU 6502. Tutte le istruzioni di questa colonna hanno indirizzamento in pagina 0, o Zero Page, X Addressing. Le operazioni che si servono di questo tipo di indirizzamento hanno una lunghezza di 2 byte. Resta quindi stabilito: i Codici Operativi il cui Nibble basso è "5" hanno una lunghezza di 2 byte. Le istruzioni della 6ª colonna hanno i seguenti codici OP: 05, 15, 25, 35, 45, 55, 65, 75, 85, 95, A5, B5, C5, D5, E5, F5.

Nella 7ª colonna troviamo ancora codici OP per i quali vale quanto detto per i codici OP della 6ª colonna. Essi hanno tutti una lunghezza di 2 bytes ed un indirizzamento in pagina 0, Zero Page, X o Zero Page, Y.

Possiamo dunque affermare: i codici OP, il cui Nibble basso è "6" hanno una lunghezza di 2 byte. Le istruzioni della 7ª colonna hanno i seguenti Codici Operativi: 06, 16, 26, 36, 46, 56, 66, 76, 86, A6, B6, C6, D6, E6, F6.

Nella 9ª colonna troviamo 16 istruzioni. Tutte le istruzioni di questa colonna hanno per tipo di indirizzamento l'Implied Addressing. Le istruzioni che si servono di questo indirizzamento sono lunghe un byte. Si può quindi dire che: i Codici Operativi il cui Nibble basso è "8" hanno una lunghezza di un byte. Le istruzioni della 9ª colonna hanno i seguenti Codici Operativi: 08, 18, 28, 38, 48, 58, 68, 78, 88, 98, A8, B8, C8, D8, E8, F8.

Nella 10ª colonna riscontriamo ancora alcune irregolarità. Per renderle evidenti listiamo qui di seguito in modo esplicito la 10ª colonna:

codice OP	simbolo mnemonico	lunghezza
09	ORA#	2 Bytes
19	ORA-ABS,Y	3 Bytes
29	AND#	2 Bytes
39	AND-ABS,Y	3 Bytes
49	EOR#	2 Bytes
59	EOR-ABS,Y	3 Bytes
69	ADC#	2 Bytes

79	ADC-ABS,Y	3 Bytes
99	STA-ABS,Y	3 Bytes
A9	LDA#	2 Bytes
B9	LDA-ABS,Y	3 Bytes
C9	CMP#	2 Bytes
D9	CMP-ABS,Y	3 Bytes
E9	SBC#	2 Bytes
F9	SBC-ABS,Y	3 Bytes

Se si esamina globalmente questa tabella è facile accorgersi che vi si succedono alternativamente due modi di indirizzamento: Immediate Addressing e Absolute Indexed, Y Addressing. L'Immediate Addressing ha una lunghezza di due byte, mentre l'Absolute Indexed, Y Addressing una lunghezza di tre byte. Come si può riconoscere in questa tabella quale istruzione è lunga 2 e quale tre byte? In base al Nibble basso del codice OP non è possibile riconoscere se l'istruzione è lunga due o tre byte. Invece tale informazione sulla lunghezza è contenuta nel Nibble alto del codice OP! Se il Nibble alto è un numero pari (il bit basso del Nibble alto è allora 0), si tratta di un'istruzione della lunghezza di due byte, ovvero un'istruzione con Immediate Addressing. Se il Nibble alto dell'istruzione è invece un numero dispari (il bit basso del Nibble superiore è allora 1), si tratta di un'istruzione lunga tre byte, ovvero un'istruzione con Absolute Indexed, Y Addressing. Con il seguente schema di programma si possono separare le istruzioni della 10ª colonna lunghe due byte da quelle di tre byte:

LDY # 00

LDA-(POINTL), Y carica il codice operativo nell'accumulatore

AND # 1F maschera i bit da 0 a 4

CMP # 19

BEQXX salta quando la lunghezza è tre byte; non effettuare il salto se la lunghezza è due byte.

La mascheratura con 1F ha per conseguenza che per i Codici Operativi 09, 29, 49, 69, 89, A9, C9, E9 nell'accumulatore è posto il numero esadecimale 09. Quando troviamo 09 nell'Accu della CPU significa che l'istruzione ha una larghezza di due byte. Per i codici OP 19, 39, 59, 79, 99, B9, D9, F9, dopo la mascheratura con 1F, abbiamo nell'Accu il numero esadecimale 19. Ciò significa che si tratta di una istruzione con Absolute Indexed, Y Addressing. Questa istruzione è lunga tre byte. Dunque mediante la mascheratura con AND # 1F è facile stabilire se si tratta di Immediate- o Absolute Indexed, Y Addressing.

Ora comprendiamo meglio la Subroutine LENACC. In questa per prima cosa viene caricato nell'Accu della CPU il codice OP di cui si deve determinare la lunghezza. Quindi il programma determina se si tratta di una istruzione BRK, RTI oppure RTS. Queste istruzioni sono lunghe un byte e costituiscono le eccezioni nella colonna 1. Successive interrogazioni mediante le : CMP # 00 (istru-

zione BRK?), CMP # 40 (istruzione RTI?) e CMP # 60 (istruzioni RTS?) separano le istruzioni irregolari della 1^a colonna da quelle regolari lunghe due byte.

Successivamente il programma controlla se si tratta di un'istruzione JSR. Questa istruzione è lunga tre byte e costituisce pure essa una irregolarità della 1^a colonna. Per verificare anche questa eccezione, il programma effettua un test con CMP # 20. In tal modo vengono "filtrate" tutte le irregolarità della 1^a colonna. Anche nella 10^a colonna della tabella dei Codici Operativi troviamo diverse irregolarità: anche qui occorre separare le istruzioni lunghe due byte da quelle di tre byte. Ciò avviene come prima grazie alla sequenza di programma:

AND # 1F

CMP # 19

BEQ LENEND

Comunque, se all'inizio della Subroutine non è stata caricata nell'accumulatore alcuna istruzione BRK, RTI, RTS, JSR e nessuna istruzione con Absolute, Y Addressing, può trattarsi solo di una istruzione regolare. Come si è detto prima l'informazione sulla lunghezza di una istruzione regolare si trova nel Nibble basso del Codice Operativo. Ecco perché le lunghezze delle istruzioni regolari sono riportate in una tabella di "Look Up". Per ricavare l'informazione sulla lunghezza di queste istruzioni è necessario la sequenza di programma:

AND # 0 ricava il Nibble basso del codice OP

TAY impiega tale Nibble quale indice

LDX-LENTBL, Y ricava la lunghezza dalla tabella di Look Up

STX-BYTES memorizza la lunghezza in byte

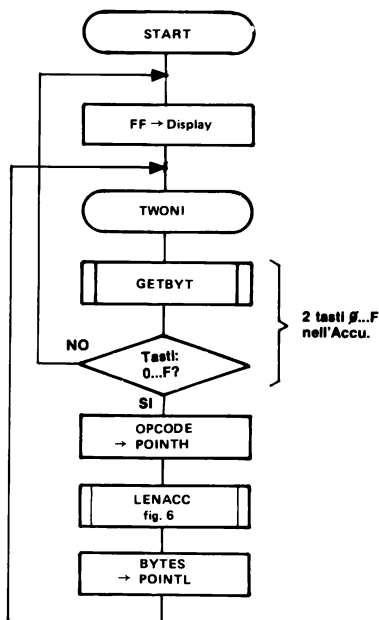
Ogni volta che si abbandona la Subroutine, la lunghezza dell'istruzione si trova nella cella di memoria BYTES. La tabella di Look Up LENTBL può essere registrata in una posizione qualsiasi del campo di memoria.

Osservazione: Le conversioni di codici si possono effettuare facilmente mediante una tabella Look Up. Con la Subroutine LENACC il codice OP, tramite tale tabella Look Up, viene convertito in una informazione di lunghezza.

Per concludere scriviamo un programma che impiega ancora la Subroutine LENACC. Questo programma presenta sul display a 6 cifre dello Junior-Computer un codice operativo e la sua lunghezza. Nei due display a sinistra compare il codice OP introdotto con la tastiera, e nei due display di mezzo la sua lunghezza in byte. I due display a destra mostrano sempre FF. Subito dopo la partenza il display viene rimesso a 0 (FFFFFF). Il diagramma di flusso grezzo della Routine per il calcolo della lunghezza delle istruzioni è dato in fig. 7. All'inizio si carica FF nei buffer di display. Quindi il programma salta al programma Monitor dello Junior-Computer, verso la Subroutine GETBYT. Questa Subroutine gestisce la tastiera ed il display del computer, ed attende l'introduzione di un

byte dati. Dopo l'introduzione di un byte dati (con i tasti dati 0...F), il programma esce dalla Subroutine GETBYT con il byte dato posto nell'accumulatore. Se si sono premuti i tasti dati 0...F, dopo il rientro della Subroutine il Flag N è posto ad 1. Azionando un tasto comandi, quale AD, DA, +, eccetera, invece la CPU esce dalla Subroutine GETBYT con il Flag N rimesso a 0. È quindi in tal modo facile determinare se sulla tastiera è stato premuto un tasto dati o un tasto comandi.

Se il programmatore ha attivato due tasti dati, ha introdotto in tal modo nello Junior-Computer un codice OP. Dopo il rientro dalla Subroutine GETBYT questo codice OP viene trasferito nel buffer di display POINTH. Successivamente il programma salta alla Subroutine LENACC. Con questa calcola la lunghezza del codice OP appena inserito. Dopo il rientro dalla LENACC, la lunghezza dell'istruzione è in BYTES e viene poi ricopiata nel buffer di display POINTL. Il programma salta ancora a TWONI ed attende l'introduzione di un nuovo codice OP. Se però nella Subroutine GETBYT viene azionato un tasto di comando, il programma annulla il display e su questo compare come al principio FFFFFF. Il programmatore a questo punto può reintrodurre un nuovo codice OP.



80915 - 4 - 7

Figura 7. Diagramma di flusso globale della routine per il calcolo della lunghezza delle istruzioni.

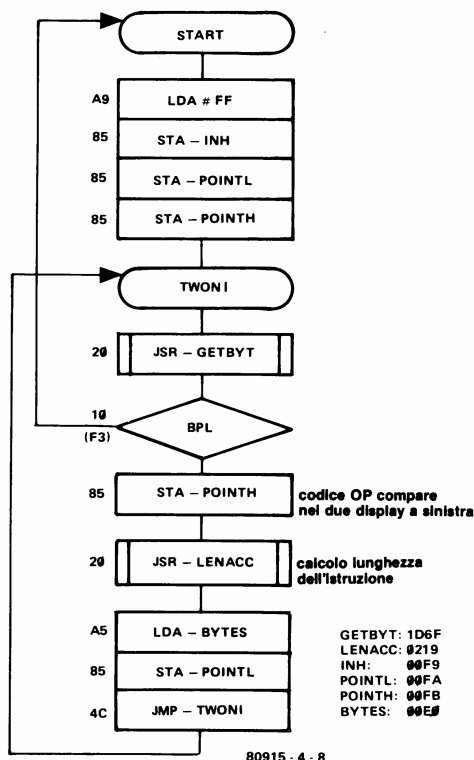


Figura 8. Diagramma di flusso dettagliato nella routine per il calcolo della lunghezza delle istruzioni.

La fig. 8 mostra il diagramma di flusso dettagliato corrispondente a fig. 7. Sono ivi indicati pure gli indirizzi delle Subroutine e delle celle buffer impiegate. La sequenza di azionamento dei tasti per l'introduzione di questo programma con la Subroutine LENACC nello Junior-Computer è riportata in fig. 9. La conversione del codice OP in una lunghezza di istruzione avviene, come detto, con una Lookup Table. Dato che in alcune colonne di fig. 5 non compare alcun codice OP, nella Lookup Table in corrispondenza a queste posizioni è posta la lunghezza di istruzione 00. Se viene introdotto un codice OP illegale, sul display compare quale lunghezza di istruzione 00. Ciò vale comunque solo per codici OP illegali delle colonne 3, 7, B ed F.

E così eccoci giunti al termine di questo libro. Disponiamo ormai del corredo di conoscenza necessarie per poter effettuare il libro JUNIOR-COMPUTER 2. In esso affineremo ulteriormente le conoscenze teoriche e pratiche. Molti interessanti programmi atten-

dono colà il lettore. Dato che nel 2° volume la teoria sarà scritta a caratteri piccoli e la pratica a caratteri grandi, esso dovrebbe risultare ancora più interessante che lo JUNIOR-COMPUTER 1. Ricordatevi che è tipico dei computer che, quando si impara l'arte della programmazione, è proprio al principio che la mela risulta più acerba ...

Tasten				Adresse Daten		
RTS	AD			xxxx	xx	
0	2	0	0	0200	xx	Indirizzo iniziale
DA		A	9	0200	A9	LDA # START
+		F	F	0201	FF	FF → Accu
+		8	5	0202	85	STAZ
+		F	9	0203	F9	Accu → INH (indirizzo 00F9)
+		8	5	0204	85	STAZ
+		F	A	0205	FA	Accu → POINTL (indirizzo 00FA)
+		8	5	0206	85	STA Z
+		F	B	0207	FB	Accu → POINTH (indirizzo 00FB)
+		2	0	0208	20	JSR- TWONI
+		6	F	0209	6F	ADL } di GETBYT (Monitor; ADH } indirizzo 1D6F)
+		1	D	020A	1D	
+		1	0	020B	10	BPL; è attivato 1 dei tasti 0...E?
+		F	3	020C	F3	se no, F3 passi indietro (= Start)
+		8	5	020D	85	STAZ (Byte nell'Accu)
+		F	B	020E	FB	Accu (= Codice OP) (indirizzo 00FB)
+		2	0	020F	20	JSR-
+		1	9	0210	19	ADL } di LENACC ADH }
+		0	2	0211	02	
+		A	5	0212	A5	LDAZ
+		E	0	0213	E0	BYTES (indirizzo 00E0) → Accu
+		8	5	0214	85	STAZ
+		F	A	0215	FA	Accu → POINTL (indirizzo 00FA)
+		4	C	0216	4C	JMP ABS
+		0	8	0217	08	ADL } dell'indirizzo di salto ADH } (TWONI)
+		0	2	0218	02	
+		A	2	0219	A2	LDX #
						Subroutine LENACC
+		0	1	021A	01	01 → X; Lunghezza d'istruzione 1
+		C	9	021B	C9	CMP #
+		0	0	021C	00	con 00
+		F	0	021D	F0	BEQ BRK?
+		1	A	021E	1A	1A passi più oltre è posto LENEND

+	C	9	021F	C9	CMP #
+	4	0	0220	40	con 40
+	F	0	0221	F0	BEQ RTI?
+	1	6	0222	16	16 passi più oltre è posto LENEND
+	C	9	0223	C9	CMP #
+	6	0	0224	60	con 60
+	F	0	0225	F0	BEQ RTS?
+	1	2	0226	12	12 passi più oltre è posto LENEND
+	A	2	0227	A2	LDX #
+	0	3	0228	03	03 → X; lunghezza d'istruzione 3
+	C	9	0229	C9	CMP #
+	2	0	022A	20	con 20
+	F	0	022B	F0	BEQ JSR?
+	0	C	022C	0C	0C passi più oltre è posto LENEND
+	2	9	022D	29	AND #
+	1	F	022E	1F	mascherare con 1F i 5 bit di destra
+	C	9	022F	C9	CMP #
+	1	9	0230	19	con 19
+	F	0	0231	F0	BEQ; istruzione da 3 bit; colonna 9
+	0	6	0232	06	06 passi più oltre è posto LENEND
+	2	9	0233	29	AND #
+	0	F	0234	0F	mascherare con 0F il nibble destro
+	A	8	0235	A8	TAY; nibble destro → Y
+	B	E	0236	BE	LDX ABS,Y; (Y + 1) locazione di memoria di LENTBL
+	3	D	0237	3D	ADL } di LENTBL
+	0	2	0238	02	ADH } (look up table)
+	8	E	0239	8E	STX ABS LENEND
+	E	0	023A	E0	ADL BYTES
+	0	0	023B	00	ADH BYTES
+	6	0	023C	60	RTS indietro al programma principale
+	0	2	023D	02	Colonna 0; Y = 00 LENTBL
+	0	2	023E	02	Colonna 1; Y = 01
+	0	2	023F	02	Colonna 2; Y = 02
+	0	0	0240	00	Colonna 3; Y = 03
+	0	2	0241	02	Colonna 4; Y = 04
+	0	2	0242	02	Colonna 5; Y = 05
+	0	2	0243	02	Colonna 6; Y = 06
+	0	0	0244	00	Colonna 7; Y = 07
+	0	1	0245	01	Colonna 8; Y = 08
+	0	2	0246	02	Colonna 9; Y = 09
+	0	1	0247	01	Colonna A; Y = 0A
+	0	0	0248	00	Colonna B; Y = 0B

+		0	3	0249	03	Colonna C; Y = 0C
+		0	3	024A	03	Colonna D; Y = 0D
+		0	3	024B	03	Colonna E; Y = 0E
+		0	0	024C	00	Colonna F; Y = 0F
AD						
0	2	0	0			Indirizzo iniziale
GO						Inizio programma
		A	9	A902	FF	
		0	3	0300	FF	
		D	2	D202	FF	
		9	E	9E03	FF	
		D	5	D502	FF	
		usw.				

Figura 9. Schema dell'impostazione da tastiera delle routine di fig. 6 e 8.

Appendici

1 Codici OP in Ordinemamento esadecimale

00	BRK	20	JSR	40	RTI	60	RTS
01	ORA (IND,X)	21	AND (IND,X)	41	EOR (IND,X)	61	ADC (IND,X)
02	-	22	-	42	-	62	-
03	-	23	-	43	-	63	-
04	-	24	BIT Z	44	-	64	-
05	ORA Z	25	AND Z	45	EOR Z	65	ADC Z
06	ASL Z	26	ROL Z	46	LSR Z	66	ROR Z
07	-	27	-	47	-	67	-
08	PHP	28	PLP	48	PHA	68	PLA
09	ORA IMM	29	AND IMM	49	EOR IMM	69	ADC IMM
0A	ASL A	2A	ROL A	4A	LSR A	6A	ROR A
0B	-	2B	-	4B	-	6B	-
0C	-	2C	BIT ABS	4C	JMP ABS	6C	JMP IND
0D	ORA ABS	2D	AND ABS	4D	EOR ABS	6D	ADC ABS
0E	ASL ABS	2E	ROL ABS	4E	LSR ABS	6E	ROR ABS
0F	-	2F	-	4F	-	6F	-
10	BPL	30	BMI	50	BVC	70	BVS
11	ORA (IND),Y	31	AND (IND),Y	51	EOR (IND),Y	71	ADC (IND),Y
12	-	32	-	52	-	72	-
13	-	33	-	53	-	73	-
14	-	34	-	54	-	74	-
15	ORA Z,X	35	AND Z,X	55	EOR Z,X	75	ADC Z,X
16	ASL Z,X	36	ROL Z,X	56	LSR Z,X	76	ROR Z,X
17	-	37	-	57	-	77	-
18	CLC	38	SEC	58	CLI	78	SEI
19	ORA ABS,Y	39	AND ABS,Y	59	EOR ABS,Y	79	ADC ABS,Y
1A	-	3A	-	5A	-	7A	-
1B	-	3B	-	5B	-	7B	-
1C	-	3C	-	5C	-	7C	-
1D	ORA ABS,X	3D	AND ABS,X	5D	EOR ABS,X	7D	ADC ABS,X
1E	ASL ABS,X	3E	ROL ABS,X	5E	LSR ABS,X	7E	ROR ABS,X
1F	-	3F	-	5F	-	7F	-

80	-	A0	LDY IMM	C0	CPY IMM	E0	CPX IMM
81	STA (IND,X)	A1	LDA (IND,X)	C1	CMP (IND,X)	E1	SBC (IND,X)
82	-	A2	LDX IMM	C2	-	E2	-
83	-	A3	-	C3	-	E3	-
84	STY Z	A4	LDY Z	C4	CPY Z	E4	CPX Z
85	STA Z	A5	LDA Z	C5	CMP Z	E5	SBC Z
86	STX Z	A6	LDX Z	C6	DEC Z	E6	INC Z
87	-	A7	-	C7	-	E7	-
88	DEY	A8	TAY	C8	INX	E8	INX
89	-	A9	LDA IMM	C9	CMP IMM	E9	SBC IMM
8A	TXA	AA	TAX	CA	DEX	EA	NOP
8B	-	AB	-	CB	-	EB	-
8C	STY ABS	AC	LDY ABS	CC	CPY ABS	EC	CPX ABS
8D	STA ABS	AD	LDA ABS	CD	CMP ABS	ED	SBC ABS
8E	STX ABS	AE	LDX ABS	CE	DEC ABS	EE	INC ABS
8F	-	AF	-	CF	-	EF	-
90	BCC	B0	BCS	D0	BNE	F0	BEQ
91	STA (IND),Y	B1	LDA (IND),Y	D1	CMP (IND),Y	F1	SBC (IND),Y
92	-	B2	-	D2	-	F2	-
93	-	B3	-	D3	-	F3	-
94	STY Z,X	B4	LDY Z,X	D4	-	F4	-
95	STA Z,X	B5	LDA Z,X	D5	CMP Z,X	F5	SBC Z,X
96	STX Z,Y	B6	LDX Z,Y	D6	DEC Z,X	F6	INC Z,X
97	-	B7	-	D7	-	F7	-
98	TYA	B8	CLV	D8	CLD	F8	SED
99	STA ABS,Y	B9	LDA ABS,Y	D9	CMP ABS,Y	F9	SBC ABS,Y
9A	TXS	BA	TSX	DA	-	FA	-
9B	-	BB	-	DB	-	FB	-
9C	-	BC	LDY ABS,X	DC	-	FC	-
9D	STA ABS,X	BD	LDA ABS,X	DD	CMP ABS,X	FD	SBC ABS,X
9E	-	BE	LDX ABS,Y	DE	DEC ABS,X	FE	INC ABS,X
9F	-	BF	-	DF	-	FF	-

In questa tabella sono riportati i codici OP in ordine esadecimale (00...FF). Per completezza, abbiamo riportato in questa tabella anche i codici OP "vuoti" (ad esempio: 1A, 1B ecc.). Il codice OP è

seguito dal simbolo mnemonico della data istruzione. Esse constano di tre lettere maiuscole (ad esempio: ADC). Dopo il simbolo mnemonico seguono ulteriori lettere che descrivono il tipo di indirizzamento dell'istruzione. Il loro significato è il seguente:

IM	Immediate Addressing (# = IM)
ABS	Absolute Addressing
Z	Zero Page Addressing
A	Accumulator Addressing
(IND,X)	Indexed Indirect Addressing
(IND),Y	Indirect Indexed Addressing
Z,X	Zero Page Indexed,X Addressing
Z,Y	Zero Page Indexed,Y Addressing
ABS,X	Absolute Indexed,X Addressing
ABS,Y	Absolute Indexed,Y Addressing
IND	Indirect Addressing

2 Lista delle Istruzioni

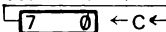
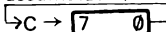
L'elenco che segue presenta le 56 istruzioni del micro-processore 6502 in ordine alfabetico. Un certo numero di tali istruzioni ammette diverse possibilità di indirizzamento, per cui in totale sono disponibili 151 codici OP diversi.

Mnemonici e loro descrizione	Tipi di indirizzamento	Codice OP (hex)	Numero degli impulsi di clock	Numero di byte	Flag modificati dall'istruzione
ADC Add memory to accumulator with carry $A + M + C \rightarrow A$ (1) (4)	IMM ABS Z (IND,X) (IND),Y Z,X ABS,X ABS,Y	69 6D 65 61 71 75 7D 79	2 4 3 6 5 4 4 4	2 3 2 2 2 2 3 3	NV ---- ZC
AND "AND" memory with accumulator $A \wedge M \rightarrow A$ (1)	IMM ABS Z (IND,X) (IND),Y Z,X ABS,X ABS,Y	29 2D 25 21 31 35 3D 39	2 4 3 6 5 4 4 4	2 3 2 2 2 2 3 3	N ---- Z -
ASL Shift left one bit (accu or memory) $C \leftarrow \boxed{7} \leftarrow 0$	ABS Z A Z,X ABS,X	0E 06 0A 16 1E	6 5 2 6 7	3 2 1 2 3	N ---- ZC
BCC Branch on carry clear (2) Branch on $C = 0$	REL	90	2 3, dopo eseguita l'istruzione	2	-----
BCS Branch on carry set (2) Branch on $C = 1$	REL	B0	2 3, dopo eseguita l'istruzione	2	-----
BEQ Branch on result zero (2) Branch on $Z = 1$	REL	F0	2 3, dopo eseguita l'istruzione	2	-----
BIT Test bits in memory: $A \wedge M$ $M_7 \rightarrow N; M_6 \rightarrow V$	ABS Z	2C 24	4 3	3 2	$M_7 M_6$ ---- Z -
BMI Branch on result minus (2) Branch on $N = 1$	REL	30	2 3, dopo eseguita l'istruzione	2	-----

Mnemonici e loro descrizione	Tipi di indirizzamento	Codice OP (hex)	Numero degli impulsi di clock	Numero di byte	Flag modificati dall'istruzione
BNE Branch on result not zero (2) Branch on Z = 0	REL	D0	2 3, dopo eseguita l'istruzione	2	-----
BPL Branch on result plus (2) Branch on N = 0	REL	10	2 3, dopo eseguita l'istruzione	2	-----
BRK Force break forced interrupt	IMP	00	7	1	---1-1--- B I
BVC Branch on overflow clear (2) Branch on V = 0	REL	50	2 3, dopo eseguita l'istruzione	2	-----
BVS Branch on overflow set (2) Branch on V = 1	REL	70	2 3, dopo eseguita l'istruzione	2	-----
CLC Clear carry flag 0 → C	IMP	18	2	1	-----0 C
CLD Clear decimal mode; 0 → D	IMP	D8	2	1	----0--- D
CLI Clear interrupt flag; 0 → 1, Enable IRQ	IMP	58	2	1	-----0-- I
CLV Clear overflow flag; 0 → V	IMP	B8	2	1	0----- V
CMP Compare memory and accumulator A-M	IMM ABS Z (IND,X) (IND),Y Z,X ABS,X ABS,Y	C9 CD C5 C1 D1 D5 DD D9	2 4 3 6 5 4 4 4	2 3 2 2 2 2 3 3	N-----ZC
CPX Compare memory and index X X-M	IMM ABS Z	E0 EC E4	2 4 3	2 3 2	N-----ZC
CPY Compare memory and index Y M-Y	IMM ABS Z	C0 CC C4	2 4 3	2 3 2	N-----ZC

Mnemonici e loro descrizione	Tipi di indirizzamento	Corice OP (hex)	Numero degli impulsi di clock	Numero di byte	Flag modificati dall'istruzione
DEC Decrement memory by one $M-1 \rightarrow M$	ABS Z Z,X ABS,X	CE C6 D6 DE	6 5 6 7	3 2 2 3	N - - - - - Z -
DEX Decrement index X by one $X-1 \rightarrow X$	IMP	CA	2	1	N - - - - - Z -
DEY Decrement index Y by one $Y-1 \rightarrow Y$	IMP	88	2	1	N - - - - - Z -
EOR "Exclusive or" memory with accumulator $A \vee M \rightarrow A$ (1)	IMM ABS Z (IND,X) (IND),Y Z,X ABS,X ABS,Y	49 4D 45 41 51 55 5D 59	2 4 3 6 5 4 4 4	2 3 2 2 2 2 3 3	N - - - - - Z -
INC Increment memory by one $M + 1 \rightarrow M$	ABS Z Z,X ABS,X	EE E6 F6 FE	6 5 6 7	3 2 2 3	N - - - - - Z -
INX Increment index X by one $X + 1 \rightarrow X$	IMP	E8	2	1	N - - - - - Z -
INY Increment index Y by one $Y + 1 \rightarrow Y$	IMP	C8	2	1	N - - - - - Z -
JMP Jump to new location $(PC + 1) \rightarrow PCL$ $(PC + 2) \rightarrow PCH$	ABS IND	4C 6C	3 5	3 3	- - - - -
JSR Jump to new location saving return address $PC + 2 \downarrow$ $(PC + 1) \rightarrow PCL$ $(PC + 2) \rightarrow PCH$	ABS	20	6	3	- - - - -

Mnemonici e loro descrizione	Tipi di indirizzamento	Codice OP (hex)	Numero degli impulsi di clock	Numero di byte	Flag modificati dall'istruzione
LDA Load accumulator with memory M → A (1)	IMM ABS Z (IND,X) (IND),Y Z,X ABS,X ABS,Y	A9 AD A5 A1 B1 B5 BD B9	2 4 3 6 5 4 4 4	2 3 2 2 2 2 3 3	N - - - - - Z -
LDX Load index X with memory M → X (1)	IMM ABS Z Z,Y ABS,Y	A2 AE A6 B6 BE	2 4 3 4 4	2 3 2 2 3	N - - - - - Z -
LDY Load index Y with memory M → Y (1)	IMM ABS Z Z,X ABS,X	A0 AC A4 B4 BC	2 4 3 4 4	2 3 2 2 3	N - - - - - Z -
LSR Shift right one bit (memory or accumulator) 0 → 7 0 → C	ABS Z A Z,X ABS,X	4E 46 4A 56 5E	6 5 2 6 7	3 2 1 2 3	0 - - - - - ZC N
NOP No operation	IMP	EA	2	1	- - - - -
ORA "OR" memory with accumulator AVM → A	IMM ABS Z (IND,X) (IND),Y Z,X ABS,X ABS,Y	09 0D 05 01 11 15 1D 19	2 4 3 6 5 4 4 4	2 3 2 2 2 2 3 3	N - - - - - Z -
PHA Push accumulator on stack A ↓	IMP	48	3	1	- - - - -
PHP Push processor status on stack; P ↓	IMP	08	3	1	- - - - -
PLA Pull accumulator from stack A ↑	IMP	68	4	1	N - - - - - Z -

Mnemonici e loro descrizione	Tipi di indirizzamento	Codice OP (hex)	Numero degli impulsi di clock	Numero di byte	Flag modificati dall'istruzione
PLP Pull processor status from stack; P↑	IMP	28	4	1	condizione originaria del registro di stato
ROL Rotate one bit left (memory or accumulator) 	ABS Z Z,X ABS,X A	2E 26 36 3E 2A	6 5 6 7 2	3 2 2 3 1	N - - - - - ZC
ROR Rotate one bit right (memory or accumulator) 	ABS Z A Z,X ABS,X	6E 66 6A 76 7E	6 5 2 6 7	3 2 1 2 3	N - - - - - ZC
RTI Return from interrupt PC↑; P↑	IMP	40	6	1	condizione originaria del registro di stato
RTS Return from subroutine PC↑; PC + 1 → PC	IMP	60	6	1	- - - - -
SBC Subtract memory from accumulator with borrow (3) A-M- \bar{C} → A (1)	IMM ABS Z (IND,X) (IND),Y Z,X ABS,X ABS,Y	E9 ED E5 E1 F1 F5 FD F9	2 4 3 6 5 4 4 4	2 3 2 2 2 2 3 3	N - - - - - ZC
SEC Set carry flag 1 → C	IMP	38	2	1	- - - - - 1
SED Set decimal mode	IMP	F8	2	1	- - - - 1 - - - D
SEI Set interrupt 1 → I, Disable IRQ	IMP	78	2	1	- - - - - 1 - - I
STA Store accumulator in memory A → M	ABS Z (IND,X) (IND),Y Z,X ABS,X ABS,Y	8D 85 81 91 95 9D 99	4 3 6 6 4 5 5	3 2 2 2 2 3 3	- - - - -

Mnemonici e loro descrizione	Tipi di indirizzamento	Codice OP (hex)	Numero degli impulsi di clock	Numero di byte	Flag modificati dall'istruzione
STX Store index X in memory; $X \rightarrow M$	ABS Z Z,Y	8E 86 96	4 3 4	3 2 2	-----
STY Store index Y in memory; $Y \rightarrow M$	ABS Z Z,X	8C 84 94	4 3 4	3 2 2	-----
TAX Transfer accumulator to index X $A \rightarrow X$	IMP	AA	2	1	N-----Z-
TAY Transfer accumulator to index Y $A \rightarrow Y$	IMP	A8	2	1	N-----Z-
TSX Transfer stack pointer to index X $S \rightarrow X$	IMP	BA	2	1	N-----Z-
TXA Transfer index X to accumulator $X \rightarrow A$	IMP	8A	2	1	N-----Z-
TXS Transfer index X to stack pointer $X \rightarrow S$	IMP	9A	2	1	N-----Z-
TYA Transfer index Y to accumulator $Y \rightarrow A$	IMP	98	2	1	N-----Z-

Note

- (1) Aggiungere 1 ad N quando viene superato il limite della pagina
- (2) Aggiungere 1 ad N quando il salto ha luogo nella medesima pagina.
- (3) Borrow = non-Carry (C).
- (4) Aggiungere 2 ad N quando il salto è ad un'altra pagina.

3 Hex-dump del programma Monitor

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1C00:	85	F3	68	85	F1	68	85	EF	85	FA	68	85	F0	85	FB	84
1C10:	F4	86	F5	BA	86	F2	A2	01	86	FF	4C	33	1C	A9	1E	8D
1C20:	83	1A	A9	04	85	F1	A9	03	85	FF	85	F6	A2	FF	9A	86
1C30:	F2	D8	78	20	88	1D	D0	FB	20	88	1D	F0	FB	20	88	1D
1C40:	F0	F6	20	F9	1D	C9	13	D0	13	A6	F2	9A	A5	FB	48	A5
1C50:	FA	48	A5	F1	48	A6	F5	A4	F4	A5	F3	40	C9	10	D0	06
1C60:	A9	03	85	FF	D0	14	C9	11	D0	06	A9	00	85	FF	F0	0A
1C70:	C9	12	D0	09	E6	FA	D0	02	E6	FB	4C	33	1C	C9	14	D0
1C80:	0B	A5	EF	85	FA	A5	F0	85	FB	4C	7A	1C	C9	15	10	EA
1C90:	85	E1	A4	FF	D0	0D	B1	FA	0A	0A	0A	0A	05	E1	91	FA
1CA0:	4C	7A	1C	A2	04	06	FA	26	FB	CA	D0	F9	A5	FA	05	E1
1CB0:	85	FA	4C	7A	1C	20	D3	1E	A4	E3	A6	E2	E8	D0	01	C8
1CC0:	86	E8	84	E9	A9	77	A0	00	91	E6	20	4D	1D	C9	14	D0
1CD0:	2A	20	6F	1D	10	F7	85	FB	20	6F	1D	10	F0	85	FA	20
1CE0:	D3	1E	A0	00	B1	E6	C5	FB	D0	07	C8	B1	E6	C5	FA	F0
1CF0:	D9	20	5C	1E	20	F8	1E	30	E9	10	3E	C9	10	D0	0A	20
1D00:	20	1E	10	C9	20	47	1E	F0	C1	C9	13	D0	14	20	20	1E
1D10:	10	BB	20	5C	1E	20	F8	1E	A5	FD	85	F6	20	47	1E	F0
1D20:	A9	C9	12	D0	07	20	F8	1E	30	A0	10	0D	C9	11	D0	09
1D30:	20	83	1E	20	EA	1E	4C	CA	1C	A9	EE	85	FB	85	FA	85
1D40:	F9	A9	03	85	F6	20	8E	1D	D0	FB	4C	CA	1C	A2	02	A0
1D50:	00	B1	E6	95	F9	C8	CA	10	F8	20	5C	1E	20	8E	1D	D0
1D60:	FB	20	8E	1D	F0	FB	20	8E	1D	F0	F6	20	F9	1D	60	20
1D70:	5C	1D	C9	10	10	11	0A	0A	0A	0A	85	FE	20	5C	1D	C9
1D80:	10	10	04	05	FE	A2	FF	60	A0	00	B1	FA	85	F9	A9	7F
1D90:	8D	81	1A	A2	08	A4	F6	A5	FB	20	CC	1D	88	F0	0D	A5
1DA0:	FA	20	CC	1D	88	F0	05	A5	F9	20	CC	1D	A9	00	8D	81
1DB0:	1A	A0	03	A2	00	A9	FF	8E	82	1A	E8	E8	2D	80	1A	88
1DC0:	D0	F5	A0	06	8C	82	1A	09	80	49	FF	60	48	84	FC	4A
1DD0:	4A	4A	4A	20	DF	1D	68	29	0F	20	DF	1D	A4	FC	60	A8
1DE0:	B9	0F	1F	8D	80	1A	8E	82	1A	A0	7F	88	10	FD	8C	80
1DF0:	1A	A0	06	8C	82	1A	E8	E8	60	A2	21	A0	01	20	B5	1D
1E00:	D0	07	E0	27	D0	F5	A9	15	60	A0	FF	0A	B0	03	C8	10
1E10:	FA	8A	29	0F	4A	AA	98	10	03	18	69	07	CA	D0	FA	60
1E20:	20	6F	1D	10	21	85	FB	20	60	1E	84	F7	84	FD	C6	F7

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1E30:	F0	12	20	6F	1D	10	0F	85	FA	C6	F7	F0	07	20	6F	1D
1E40:	10	04	85	F9	A2	FF	60	20	A6	1E	20	DC	1E	A2	02	A0
1E50:	00	B5	F9	91	E6	CA	C8	C4	F6	D0	F6	60	A0	00	B1	E6
1E60:	A0	01	C9	00	F0	1A	C9	40	F0	16	C9	60	F0	12	A0	03
1E70:	C9	20	F0	0C	29	1F	C9	19	F0	06	29	0F	AA	BC	1F	1F
1E80:	84	F6	60	A5	E6	85	EA	A5	E7	85	EB	A4	F6	B1	EA	A0
1E90:	00	91	EA	E6	EA	D0	02	E6	EB	A5	EA	C5	E8	D0	EC	A5
1EA0:	EB	C5	E9	D0	E6	60	A5	E8	85	EA	A5	E9	85	EB	A0	00
1EB0:	B1	EA	A4	F6	91	EA	A5	EA	C5	E6	D0	06	A5	EB	C5	E7
1EC0:	F0	10	38	A5	EA	E9	01	85	EA	A5	EB	E9	00	85	EB	4C
1ED0:	AE	1E	60	A5	E2	85	E6	A5	E3	85	E7	60	18	A5	E8	65
1EE0:	F6	85	E8	A5	E9	69	00	85	E9	60	38	A5	E8	E5	F6	85
1EF0:	E8	A5	E9	E9	00	85	E9	60	18	A5	E6	65	F6	85	E6	A5
1F00:	E7	69	00	85	E7	38	A5	E6	E5	E8	A5	E7	E5	E9	60	40
1F10:	79	24	30	19	12	02	78	00	10	08	03	46	21	06	0E	02
1F20:	02	02	01	02	02	02	01	01	02	01	01	03	03	03	03	6C
1F30:	7A	1A	6C	7E	1A	B1	E6	A0	FF	C4	EE	F0	0D	D1	EC	D0
1F40:	0A	88	B1	EC	AA	88	B1	EC	A0	01	60	88	88	88	D0	E9
1F50:	60	38	A5	E4	E9	FF	85	EC	A5	E5	E9	00	85	ED	A9	FF
1F60:	85	EE	20	D3	1E	20	5C	1E	A0	00	B1	E6	C9	FF	D0	1D
1F70:	C8	B1	E6	A4	EE	91	EC	88	A5	E7	91	EC	88	A5	E6	91
1F80:	EC	88	84	EE	20	83	1E	20	EA	1E	4C	65	1F	20	F8	1E
1F90:	30	D3	20	D3	1E	20	5C	1E	A0	00	B1	E6	C9	4C	F0	16
1FA0:	C9	20	F0	12	29	1F	C9	10	F0	1A	20	F8	1E	30	E6	A9
1FB0:	03	85	F6	4C	33	1C	C8	20	35	1F	F0	EE	91	E6	8A	C8
1FC0:	91	E6	D0	E6	C8	20	35	1F	F0	E0	38	E5	E6	38	E9	02
1FD0:	91	E6	4C	AA	1F	D8	A9	00	85	FB	85	FA	85	F9	20	6F
1FE0:	1D	10	F2	85	FB	20	6F	1D	10	EB	85	FA	18	A5	FA	E5
1FF0:	FB	85	F9	C6	F9	4C	DE	1F	FF	FF	2F	1F	1D	1C	32	1F

4 Collegamenti dei connettori

Connettore di espansione

32a	Massa	32a	o	o	32c	32c	Massa
31a	RAM-R/W	31a	o	o	31c	31c	non collegato
30a	Φ 1	30a	o	o	30c	30c	EX
29a	K1	29a	o	o	29c	29c	R/W
28a	non collegato	28a	o	o	28c	28c	K2
27a	Φ 2	27a	o	o	27c	27c	non collegato
26a	A1	26a	o	o	26c	26c	A0
25a	A3	25a	o	o	25c	25c	A2
24a	A5	24a	o	o	24c	24c	A4
23a	A7	23a	o	o	23c	23c	A6
22a	A9	22a	o	o	22c	22c	A8
21a	A11	21a	o	o	21c	21c	A10
20a	A13	20a	o	o	20c	20c	A12
19a	A15	19a	o	o	19c	19c	A14
18a	-5 V	18a	o	o	18c	18c	K3
17a	K4	17a	o	o	17c	17c	+12 V
16a	verso 16c	16a	o	o	16c	16c	
15a	K5	15a	o	o	15c	15c	K6
14a	K7	14a	o	o	14c	14c	SO
13a	non collegato	13a	o	o	13c	13c	non collegato
12a	IRQ	12a	o	o	12c	12c	NMI
11a	non collegato	11a	o	o	11c	11c	non collegato
10a	D7	10a	o	o	10c	10c	D6
9a	D5	9a	o	o	9c	9c	D4
8a	D3	8a	o	o	8c	8c	D2
7a	D1	7a	o	o	7c	7c	D0
6a	non collegato	6a	o	o	6c	6c	non collegato
5a	RES	5a	o	o	5c	5c	RDY
4a	Massa	4a	o	o	4c	4c	Massa
3a	non collegato	3a	o	o	3c	3c	non collegato
2a	non collegato	2a	o	o	2c	2c	non collegato
1a	+5 V	1a	o	o	1c	1c	+5 V

Connettore di porta

non collegato 30	o	o	31 non collegato
non collegato 28	o	o	29 non collegato
PB3 26	o	o	27 non collegato
PB1 24	o	o	25 PB2
PB7 22	o	o	23 PB0
PB5 20	o	o	21 PB6
non collegato 18	o	o	19 PB4
non collegato 16	o	o	17 +5 V
non collegato 14	o	o	15 non collegato
non collegato 12	o	o	13 non collegato
PA7 10	o	o	11 non collegato
PA5 8	o	o	9 PA6
PA3 6	o	o	7 PA4
PA1 4	o	o	5 PA2
+5 V 2	o	o	3 PA0
		o	1 Massa

PASCAL

IMPARIAMO IL PASCAL

Compattezza, concisione, chiarezza e notevoli potenzialità scientifiche, oltre a prestarsi ottimamente per calcoli gestionali e ad essere usato anche con i microcomputer, sono le caratteristiche che decretano il successo del PASCAL come linguaggio di programmazione. Non vi era però finora un testo che insegnasse a tutti a programmare in PASCAL: o perché i libri esistenti sono troppo concisi, o troppo semplici, oppure perché richiedono la conoscenza di altri linguaggi di programmazione, o, non ultimo, perché in inglese.

Queste sono proprio le lacune che colma il libro: un libro di divulgazione, incentrato sull'auto-apprendimento, che non ha nulla di accademici non funzionali al lettore, riportandolo "a scuola". I capitoli sono il più possibile organici, in modo che la loro consultazione sia semplice ed agevole. Un riassunto di quanto si apprenderà è posto all'inizio e non in fondo al capitolo, perché il lettore possa subito avere un metro valutativo con cui verificare passo passo il suo apprendimento. E poi, ci sono consigli, problemi, esercizi affinché il libro sia "usato" e non letto, perché si possa sapere come si usa un'istruzione piuttosto che conoscerne le differenze semantiche tra linguaggio e linguaggio. Con un lavoro graduale, partendo senza alcuna conoscenza di programmazione, dopo circa due settimane dovreste conoscere abbastanza bene il PASCAL. Un buon risultato, no?!

IMPARIAMO IL PASCAL

EDIZIONE ITALIANA

FLAVIO WALDNER

GRUPPO EDITORIALE JACKSON



Pagine 162 Prezzo Lit. 10.000

Per ordinare il volume utilizzate l'apposito tagliando d'ordine inserito in fondo alla rivista.
Formato 15 x 21
Codice 501A

SOMMARIO

- Da non trascurare
- Come si descrive la sintassi del linguaggio
- Come si scrive in PASCAL
- Il programma e le dichiarazioni in generale
- Le dichiarazioni ed i tipi standard
- I tipi speciali e subrange
- Gli statements di assegnazione
- Gli statements di ripetizione
- Gli statements logici
- I dati strutturati - Generalità
- Il tipo array
- Il tipo record
- Il tipo set
- Il tipo file
- Il tipo pointer
- Le procedure e le funzioni
- Procedure ricorrenti input ed output
- I diagrammi di struttura

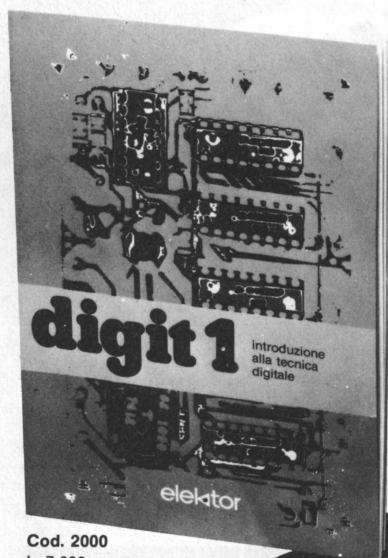


GRUPPO EDITORIALE JACKSON

DIVISIONE LIBRI

PER ORDINAZIONE UTILIZZATE L'APPOSITO TAGLIANDO INSERITO IN FONDO A QUESTO LIBRO.

Bastano questi due libri per imparare veramente l'elettronica digitale



Cod. 2000
L. 7.000
(Abb. L. 6.400)

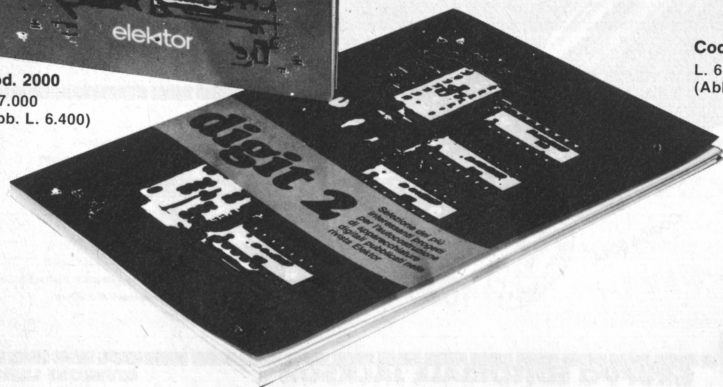
Per imparare fattivamente l'elettronica digitale occorre unire teoria e sperimentazione pratica. Il **DIGIT 1** porta il lettore ad impadronirsi dei concetti fondamentali senza ricorrere a formule noiose ed astratte ma con spiegazioni chiare e semplici. Esperimenti pratici utilizzando una originale piastra sperimentale a circuito stampato, fornito a richiesta, consentono un'introduzione passo-passo alla teoria di base e alle applicazioni dell'elettronica digitale.

È però solo realizzando praticamente delle applicazioni, che il lettore può dirsi veramente padrone delle tecniche digitali. Questo è lo scopo del **DIGIT 2**, che costituisce il naturale prosieguo del volume precedente, al fine di quell'unità didattica di cui si è parlato.

I circuiti pratici presentati nel **DIGIT 2** sono oltre 50, tutti interessantissimi che spaziano dal frequenzimetro al generatore di onde sinusoidali-triangolari-rettangolari, fino all'impianto semaforico o alla pistola luminosa.

Una serie di pratiche e divertenti realizzazioni, insomma, per arricchire il proprio laboratorio, la propria casa o semplicemente per divertirsi, ma soprattutto per imparare veramente l'elettronica digitale.

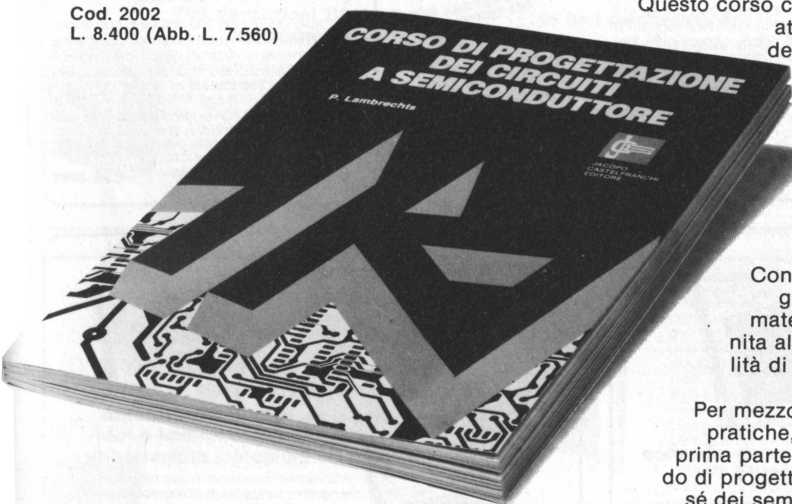
Cod. 6011
L. 6.000
(Abb. L. 5.300)



PER ORDINAZIONE UTILIZZATE L'APPOSITO TAGLIANDO INSERITO IN FONDO A QUESTO LIBRO.

La guida pratica per progettare e calcolare da soli i circuiti elettronici

Cod. 2002
L. 8.400 (Abb. L. 7.560)



Questo corso costituisce la guida attraverso i meandri della moderna tecnica circuitale dei semiconduttori. A differenza delle trattazioni sinora apparse in questo settore, la materia viene trattata con molta semplicità.

Con un minimo di grigia teoria e di arida matematica, viene fornita al lettore la possibilità di progettare circuiti a semiconduttore.

Per mezzo di chiare nozioni pratiche, già alla fine della prima parte il lettore è in grado di progettare e calcolare da sé dei semplici stadi amplificatori. Vengono considerate le tecniche circuitali tipiche della

moderna tecnologia dei circuiti integrati fra le quali: l'accoppiamento in corrente continua, l'indipendenza delle funzioni circuitali della variazione delle caratteristiche nei singoli esemplari, come pure l'uso di componenti attivi in sostituzione di induttanze, capacità e resistenze. Chiaramente si deve fare un cenno sulla teoria dei semiconduttori. Si parlerà, perciò, anche delle proprietà fondamentali dei più importanti componenti.

Il corso, inoltre, esamina i problemi di fondo che sorgono nel progetto di circuiti più complicati. Dato che le complesse funzioni di tali circuiti si ottengono in pratica combinando tra loro i circuiti fondamentali, viene mantenuta la semplicità della tecnica di progetto e di calcolo.

PER ORDINAZIONE UTILIZZATE L'APPOSITO TAGLIANDO INSERITO IN FONDO A QUESTO LIBRO.

LIBRI IN

Le Radiocomunicazioni



Ciò che i tecnici, gli insegnanti, i professori, i radioamatori, gli studenti, i radiooperatori debbono sapere sulla propagazione e ricezione delle onde em, sulle interferenze reali od immaginarie, sui radiodisturbi e loro eliminazione, sulle comunicazioni extra-terrestri.

Oltre 100 figure, tabelle varie e di propagazione.

L. 7.500 (Abb. L. 6.750)

Cod. 7001

Alla ricerca dei tesori

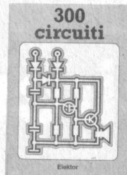
Il primo manuale edito in Italia che tratta la prospezione elettronica. Il libro, in oltre 110 pagine ampiamente illustrate spiega tutti i misteri di questo hobby affascinante. Dai criteri di scelta dei rivelatori, agli approcci necessari per effettuare le ricerche, dal mercato dei rivelatori di seconda mano alla manutenzione del detector fino alle norme del codice che il prospektore deve conoscere. Il libro analizza anche ricerche particolari come quelle sulle spiagge, nei fiumi, nei vecchi stabili, in miniere ecc.

L. 6.000 (Abb. L. 5.400)

Cod. 8001



300 circuiti



300 Circuiti

Il libro raggruppa 300 articoli in cui vengono presentati schemi elettrici completi e facilmente realizzabili, oltre a idee originali di progettazione circuitale. Le circa 270 pagine di **300 Circuiti** vi ripropongono una moltitudine di progetti dal più semplice al più sofisticato con particolare riferimento a circuiti per applicazioni domestiche, audio, di misura, giochi elettronici, radio, modellismo, auto e hobby.

L. 12.500 (Abb. L. 11.250)

Cod. 6009



Transistor cross-reference guide

Il volume raccoglie circa 5.000 tipi diversi di transistor prodotti dalle principali case europee americane (Motorola, Philips, General Electric, R.C.A., Texas Instruments, Westinghouse, AEG-Telefunken) e fornisce di essi l'indicazione di un eventuale prodotto equivalente giapponese (Toshiba, Nec, Hitachi, Mitsubishi, Matsushita, Fujitsu, Sony, Sanyo). Di ogni transistor inoltre, vengono forniti i principali parametri elettrici e meccanici.

L. 8.000 (Abb. L. 7.200)

Cod. 6007

Manuale di sostituzione dei transistori giapponesi

Manuale di intercambiabilità fra transistori delle seguenti Case giapponesi: Sony, Sanyo, Toshiba, Nec, Hitachi, Fujitsu, Matsushita, Mitsuhashi. Il libro ne raccoglie circa 3.000.

L. 5.000 (Abb. L. 4.500)

Cod. 6005

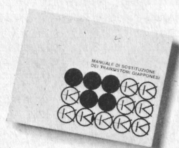


Tabelle equivalenze semiconduttori e tubi elettronici professionali

Un libro che riempie le lacune delle pubblicazioni precedenti sull'argomento. Sono elencati i modelli equivalenti Siemens per quanto riguarda:

- Transistori europei, americani e giapponesi
- Diodi europei, americani e giapponesi
- Diodi controllati (SCR-thyristors)
- LED
- Circuiti integrati logici, analogici e lineari per radio-TV
- Circuiti integrati MOS
- Tubi elettronici professionali e vidicons.

L. 5.000 (Abb. L. 4.500)

Cod. 6006

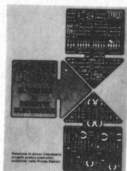
VETRINA

Selezione di progetti elettronici

Una selezione di interessanti progetti pubblicati sulla rivista "Elektor". Ciò che costituisce il "trait d'union" tra le varie realizzazioni proposte e la varietà d'applicazione, l'affidabilità di funzionamento, la facilità di realizzazione, nonché l'elevato contenuto didattico.

L. 9.000 (Abb. L. 8.100)

Cod. 6008



TV SERVICE 100 riparazioni TV illustrate e commentate



Dalle migliaia di riparazioni che si effettuano in un moderno laboratorio TV, sono assai poche quelle che si discostano dalla normale "routine" e sono davvero gratificanti per il tecnico appassionato. Cento di queste "perle" sono state raccolte in questo libro, e proposte all'attenzione di chiunque voglia per hobby o per mestiere il Servizio di Assistenza TV.

L. 10.000 (Abb. L. 9.000)

Cod. 7000

Accessori elettronici per autoveicoli



In questo volume sono trattati progetti di accessori elettronici per autoveicoli quali: l'amplificatore per autoradio, l'antifurto, l'accensione elettronica, il pluri lampeggiatore di sosta, il temporizzatore per fergicristallo ed altri ancora.

L. 6.000 (Abb. L. 5.400)

Cod. 8003

Le luci psichedeliche

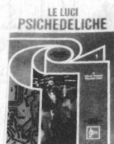
Questo volume propone numerosi progetti per costruire apparecchi psichedelici di ogni tipo.

Tutti gli apparecchi descritti sono stati provati e collaudati e sono corredati da ampie descrizioni, schemi elettrici e di montaggio.

Questo libro, tratta anche teoria e realizzazioni di generatori psichedelici sino a 6 kW di potenza, flash elettronici, luci rotanti etc.

L. 4.500 (Abb. L. 4.000)

Cod. 8002



TTL IC cross reference manual

Il prontuario fornisce le equivalenze, le caratteristiche elettriche e meccaniche di pressoché tutti gli integrati TTL sinora prodotti dalle principali case mondiali.

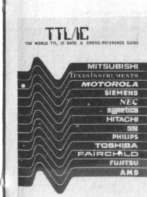
I dispositivi Texas, Fairchild, Motorola, National, Philips, Signetics, Siemens, Fujitsu, Hitachi, Mitsubishi, Nec, Toshiba, Advanced Micro Deviced, sono confrontati tra loro all'interno di ogni famiglia proposta.

Per facilitare la ricerca o la sostituzione del dispositivo in esame, è possibile anche, dopo aver appreso ad integrare la nomenclatura degli IC, consultare il manuale a seconda delle funzioni svolte nei circuiti applicativi.

Rappresenta, quindi, un indispensabile strumento di lavoro per tutti coloro che lavorano con i TTL.

L. 20.000 (Abb. L. 18.000)

Cod. 6010



Appunti di elettronica Vol. 1

Il libro è costituito da una raccolta di fogli ognuno dei quali tratta un singolo argomento.

Una particolare ed elegante confezione, studiata appositamente per rispondere alle precise finalità dell'opera, fa sì che tutti i fogli possano essere asportati e consultati separatamente.

Esposizione generale-Elettricità-Parametri principali-Fenomeni alternati sinusoidali-Oscillazioni-Analisi delle oscillazioni-Tensione costante e corrente continua-Tensione variabile unidirezionale-Corrente variabile unidirezionale-Tensione alternata-Corrente alternata-Resistenza statica e resistenza differenziale.

L. 8.000 (Abb. 7.200)

Cod. 2300



IMPORTANTE

PER ORDINAZIONE UTILIZZATE L'APPOSITO TAGLIANDO INSERITO IN FONDO A QUESTO LIBRO.

Molti associano alla parola "computer" il concetto: meno lavoro, più tempo libero. In termini più attuali e più realistici tale formula dovrebbe esprimersi in: più lavoro nello stesso tempo. Il crescente numero dei "fans" dei micro-computer porta oggi ad ancora un'altra conclusione: più lavoro nel tempo libero!

Prima di premere il primo tasto, v'è tuttavia il grosso problema della scelta: infatti l'offerta di computer già montati o in kit, di libri e di riviste è assai abbondante. Ci sono già state persone che non hanno saputo distinguere il bosco davanti a molti alberi: nel nostro caso, hanno abbandonato il progetto oppure si sono dovuti accorgere che il nuovo home-computer era stato un bidone.

Elektor vuole perciò dare una mano a tutti coloro che iniziano in questo campo:

- Con lo Junior-Computer viene offerto un micro-computer economico su una sola piastra per l'autocostruzione, con un corso di istruzione in 4 volumi.
- Il sistema base "Junior-Computer" è predisposto per ogni tipo di espansione richiesto dalle singole applicazioni specifiche o generali.

Dunque: buon lavoro!

Journal of Interpersonal Violence

Volume 1