

costruiamo un vero microelaboratore elettronico e impariamo a programmare

L. 4.000

SUPPLEMENTO AL N° 5 di Sperimentare
Spedizione in Abb. Postale Gruppo III/70



Giovanni Ghiringhelli e Giuseppe Fusaroli

**costruiamo un vero
microelaboratore
elettronico
e impariamo a programmare**



JACOPO CASTELFRANCHI EDITORE
Via dei Lavoratori, 124 - 20092 Cinisello Balsamo

SPERIMENTARE

Rivista mensile di elettronica pratica; Editore: J.C.E. - Direttore responsabile: RUBEN CASTELFRANCHI - Capo redattore: GIAMPIETRO ZANGA - Vice capo redattore: GIANNI DE TOMASI - Redazione: SERGIO CIRIMBELLI, DANIELE FUMAGALLI, TULLIO LACCHINI, MARTA MENEGARDO - Grafica e impaginazione: MARCELLO LONGHINI - Laboratorio: ANGELO CATTANEO, LORENZO BARRILE - Contabilità: ROBERTO OSTELLI, M. GRAZIA SEBASTIANI - Diffusione e abbonamenti: PATRIZIA GHIONI - Collaboratori LUCIO VISINTINI, FILIPPO PIPITONE, LUCIO BIANCOLI, FEDERICO CANCARINI, LODOVICO CASCIANINI, SANDRO GRISOSTOLO, GIOVANNI GIORGINI, ADRIANO ORTILE, AMADIO GOZZI, PIERANGELO PENSA, GIUSEPPE CONTARDI - Direzione, Redazione: Via dei Lavoratori, 124; 20092 Cinisello Balsamo - Milano. Telefono 6172671 - 6172641. - Amministrazione: Via Vincenzo Monti, 15 - 20123 Milano. - Autorizzazione alla pubblicazione: Tribunale di Monza, numero 258 del 28-11-1974. - Stampa: Tipo-Lito Elcograf s.p.a. 22050 Beverate (Como). - Concessionario esclusivo per la diffusione in Italia e all'Estero SODIP - Via Zuretti, 25 - 20125 Milano; SODIP - Via Serpieri, 11/5 - 00197 Roma. - Spedizione in abbonamento postale gruppo III/70.



JACOPO CASTELFRANCHI EDITORE

© Tutti i diritti di riproduzione e traduzione degli articoli pubblicati sono riservati.



Mensile associato all'USPI
Unione Stampa Periodica Italiana

PREFAZIONE

Questo libro vuole essere un contributo al passaggio dall' "era dei cervelli elettronici" a quella degli elaboratori elettronici.

*Questo trattato sul microelaboratore infatti è il frutto del lavoro di un gruppo di esperti italiani nel campo della divulgazione tecnica e della progettazione con i dispositivi elettronici che saranno i protagonisti della nostra vita di domani:
i MICROPROCESSORI.*

Con un taglio indirizzato specialmente a chi deve partire da zero si è voluto sfatare una volta per tutte il mito del "troppo difficile", del "queste sono cose per i soli addetti ai lavori" con una trattazione completa, giustamente approfondita, ma soprattutto facile da capire, divertente e, perchè no, entusiasmante anche perchè collegata alla costruzione di un vero e proprio microelaboratore elettronico sul quale verificare in pratica le nozioni apprese.

Ma tutto questo non toglie che anche l'esperto in elettronica non possa trovare in queste pagine la chiave per comprendere con naturalezza la filosofia dei moderni microelaboratori e imparare a programmare quasi senza accorgersene. E questa è una cosa positiva, più che positiva: troppo spesso si è accomunato il "tecnologicamente avanzato" con il "difficile", quasi che argomenti importanti come il microprocessore e la programmazione conservino il loro prestigio professionale solo se spiegati in maniera comprensibile a pochi.

La giusta prospettiva

La giusta prospettiva, il giusto punto di vista dal quale si è partiti è in sostanza la chiave della semplicità, vorremmo ribattere della naturalezza con la quale il lettore assimila passo dopo passo la filosofia del microelaboratore imparando a capirlo e a saperlo usare.

Solo un paragone. Vi è mai capitato di trovarvi ad osservare una immagine talmente da vicino che anche il particolare più definito non è altro che una macchia, un insieme di chiaroscuri senza significato. Certo che se in queste condizioni si vuole capire il significato di quella immagine, ciò che essa rappresenta, quello è senz'altro il punto di vista meno adatto.

Così per il microprocessore partire dal dettaglio infinitesimo, dal particolare squisitamente tecnico può essere un grosso errore tale da scoraggiare i più.

Ma se ci si allontana da quella immagine per vederla prima nel suo insieme, per capire la sua forma, la sua struttura globale, ecco che quelle macchie senza significato cominciano a prendere forma, ad assumere la loro posizione logica in un contesto che finirà per diventarci familiare. Vogliamo dire che il microelaboratore, il computer, se presentato da una giusta prospettiva non può essere non capito: siamo convinti che l'opera di un uomo non può non essere compresa da qualsiasi altro uomo, la chiave sta nel giusto meccanismo del trasferimento delle informazioni.

l'Ing. Giovanni Ghiringhelli e il Sig. Giuseppe Fusaroli, autori della trattazione, ringraziano il Sig. Giampietro Zanga, Direttore della Casa Editrice JCE, per aver creduto nell'iniziativa e per aver incoraggiato questo lavoro che si augurano possa incontrare il favore dei lettori.

Un particolare ringraziamento va anche al Sig. Marcello Marongiu per la consulenza prestata nella redazione dei testi e al Prof. Angelo Bressan e al Sig. Piero Dell'Orco per il contributo ai programmi applicativi riportati in appendice.

SOMMARIO

PREFAZIONE	pag.	5
CAPITOLO I - Cosa è un microelaboratore	»	9
Cosa fare con un computer: il limite è la fantasia	»	10
Come è composto l'AMICO 2000	»	10
Struttura di un microelaboratore	»	10
Come funziona un microelaboratore	»	11
Dal codice binario al codice esadecimale	»	12
Movimento e controllo dei dati in un microprocessore a 8 bit	»	13
CAPITOLO II - Il linguaggio del microelaboratore	»	15
Cosa è un programma	»	15
Il microelaboratore e il suo linguaggio	»	15
Un esempio pratico	»	15
Il program counter	»	18
Le istruzioni esaminate	»	19
Alcuni dettagli sulla CPU	»	19
CAPITOLO III - Montaggio e collaudo del microelaboratore	»	21
L'architettura dell'AMICO 2000A	»	21
Il montaggio	»	22
L'assemblaggio	»	22
Un primo collaudo	»	24
Completamento del montaggio	»	24
Introduciamo un primo programma	»	27
L'esecuzione del programma	»	29
Un programma più complesso: "Il gioco dei riflessi"	»	30
CAPITOLO IV - Il sistema di indirizzamento	»	35
La somma nel sistema binario	»	35
Esercizio con l'AMICO 2000	»	36
Somma esadecimale e somma decimale	»	37
La "pagina zero"	»	37
Il metodo di indirizzamento	»	39
Approfondimento hardware: il clock	»	39
Suddivisione dell'AMICO 2000	»	40
Applicazione	»	40

CAPITOLO V - L'uso del registratore a cassette	»	43
L'interfaccia per il registratore	»	43
Il montaggio	»	43
l'utilizzo del registratore e suo collegamento	»	44
Operazione di lettura del nastro	»	45
Operazione di registrazione sul nastro	»	47
Montaggio dell'espansione RAM	»	47
Utilizzo della cassetta registrata: "la tombola elettronica"	»	47
Software	»	48
Esercitazione	»	50
CAPITOLO VI - I numeri negativi	»	53
Sottrazione tramite i numeri negativi	»	54
L'Overflow	»	55
L'istruzione SBC	»	55
Status	»	56
Le istruzioni di Branch	»	56
Index X	»	58
Indirizzamento in pagina zero indicizzato	»	58
Il Flow-Chart	»	59
Le istruzioni del 6502	»	60
Esercizi	»	60
Un gioco: il "master mind"	»	62
CAPITOLO VII - Registro indice Y e Stack Pointer	»	63
Il registro indice Y	»	63
Un esempio pratico	»	64
Uso del registro indice Y	»	65
Indirizzamento assoluto indicizzato	»	65
Indirizzamento in pagina base	»	66
Pointer di Stack e Subroutine	»	66
Lancio di una Subroutine	»	67
Esempio di utilizzo di una Subroutine	»	67
Istruzioni che caricano lo Stack Pointer	»	68
Due esercizi pratici	»	68
Istruzioni di Shift e Rotazione	»	69
La moltiplicazione	»	71
Un gioco: "la corsa dei cavalli"	»	73
CAPITOLO VIII - Uso della porta utente	»	75
La Subroutine del monitor	»	77
L'Interrupt	»	82
Tipi di interrupt	»	83
Il funzionamento della interruzione	»	83
Il funzionamento di Reset	»	84
Esempio dell'uso di Interrupt	»	84
Funzionamento in Single Step	»	85
CAPITOLO IX - Ultime istruzioni di indirizzamento	»	87
Sistemi di indirizzamento del 6502	»	88
Un esercizio	»	88
Un gioco: il "tiro al bersaglio"	»	89
CAPITOLO X - Il Listing del monitor	»	91
Software	»	91
APPENDICE 1	»	95
Sommaro istruzioni 6502	»	95
APPENDICE 2	»	97
Programmi applicativi per il microcomputer AMICO 2000	»	97

COSA È UN MICROELABORATORE

Tutti, chi più chi meno, abbiamo sentito parlare di “cervelli elettronici” o meglio di elaboratori, computers, minicomputers e microcomputers.

Queste apparecchiature ci sono sempre state descritte come macchine complicatissime, riservate unicamente agli addetti ai lavori, mostri elettronici in grado di compiere le più disparate e complesse operazioni in tempi brevissimi.

Fino a poco tempo fa gli elaboratori si dividevano comunemente in due categorie principali: i grandi elaboratori (mainframes) ed i minielaboratori. In seguito la tecnologia dei circuiti integrati ha generato la sua terza grande rivoluzione dopo quella del transistor e del circuito integrato: il MICROPROCESSORE. Con questo dispositivo, che è un circuito integrato in grande scala (LSI = Large Scale Integration), ovvero moltissimi transistori, diodi e resistenze concentrati in uno stesso frammento di silicio di pochi millimetri quadrati (vedi figura) oggi chiunque, con un po' di studio e di esercizio è in grado di costruirsi un vero e proprio elaboratore elettronico di piccole dimensioni detto MICROCOMPUTER o MICROELABORATORE.

Beninteso, con il prefisso “micro” si vuole intendere piuttosto ridotto nelle dimensioni e nei costi che non nella potenza, visto che queste apparecchiature hanno una capacità di elaborazione realmente notevole.

Fino a poco tempo fa pensare di costruirsi un elaboratore elettronico poteva essere giusto una battuta di spirito o l'idea di un genio svitato e pieno di soldi.

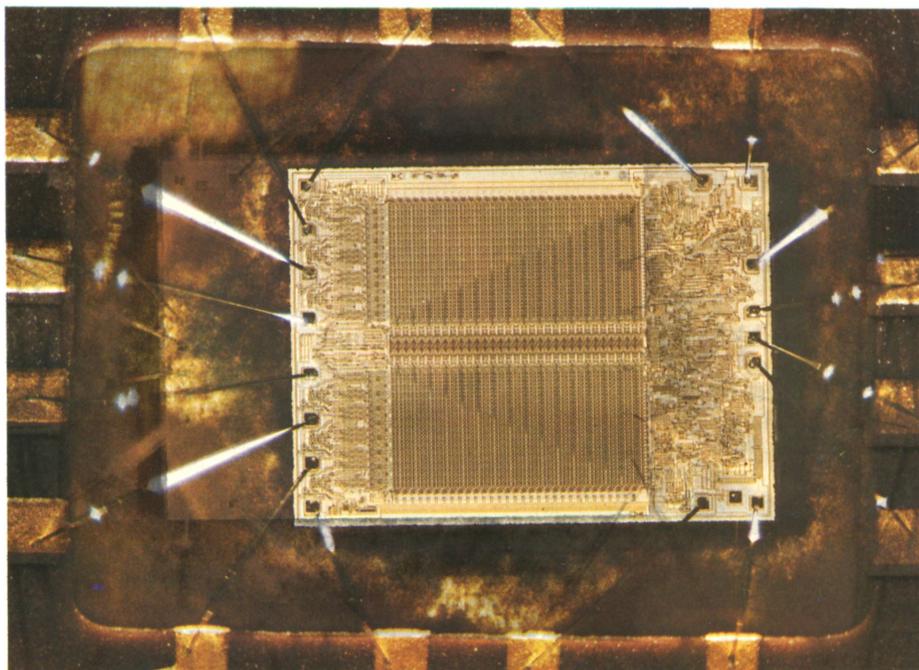
Oggi invece questo è possibile anche per te che stai leggendo queste righe e, te lo assicuriamo, anche se non sai assolutamente nulla di elaborazione dati.

Con questo libro infatti e con un po' di buona volontà si imparerà a costruire e a utilizzare un microcomputer componibile ed espandibile come potenza e prestazioni, basato su un potente microprocessore.

Non si tratta del solito corso teorico e noioso, al contrario: tutto ciò che tratterà la parte teorica sarà direttamente attinente all'argomento preso in considerazione senza inutili e spesso scoraggianti approfondimenti.

L'argomento sarà trattato tenendo sempre presenti i seguenti tre fini:

- A) *L'insegnamento*: struttura hardware (componenti, circuito stampato, etc.), software (programmi di base ed applicativi), istruzioni e linguaggi del microcomputer.
- B) *La costruzione*: il microcomputer, che si chiama AMICO 2000A, viene fornito montato e collaudato o in scatola di montaggio
- C) *Programmi applicativi*: giochi, mate-



Memoria RAM M4027 da 4096 bit della S.G.S. - A.T.E.S. il chip, la piastra al centro della foto, è la vera e propria memoria e misura 2,54 x 3,30 mm.

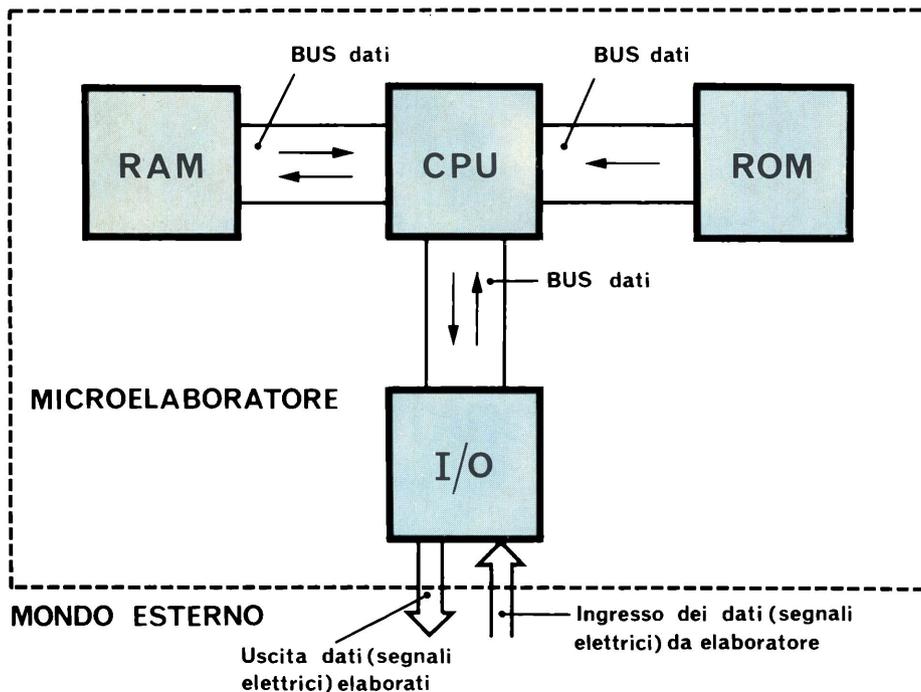


Fig. 1 - Schema a blocchi che mostra la struttura di un microelaboratore. Si noti che dalla ROM (memoria a sola lettura) i dati possono essere solo richiamati.

matica e programmi di utilità. Così come è stata strutturata, questa trattazione è adatta al principiante o a chi si diletta con l'hobby dell'elettronica, agli studenti di ogni disciplina ed in particolare modo di quelle scientifiche e tecniche, ai neodiplomati in elettronica o elettrotecnica che stanno per inserirsi nel mondo del lavoro ed anche al tecnico esperto che deve aggiornare le sue conoscenze sulle tecnologie più avanzate.

Cosa fare con un computer: il limite è la fantasia

Cosa significa possedere un elaboratore e saperlo usare?

Di primo acchito saremmo portati a pensare a cose strane, complicate, come il controllo della rotta di un missile, oppure poco allettanti come la gestione della Anagrafe Tributaria. Niente di tutto questo, fortunatamente il microelaboratore che costruirete non è destinato a risolvere questi problemi! Potremo invece realizzare un mare di cose nuove e divertenti come giochi elettronici di qualsiasi genere dettati dalla nostra fantasia, oppure eseguire delle complicate operazioni di calcolo matematico. L'elaboratore potrà essere utile in casa

per tenere sotto controllo programmato ed ottimizzato l'impianto di riscaldamento o per realizzare antifurti sofisticatissimi, o ancora per innaffiare automaticamente il nostro giardino.

Non sarebbe neanche tanto difficile utilizzare l'elaboratore come rubrica telefonica a chiamata diretta o, perché no, realizzare un controllo automatico del nostro plastico dei trenini elettrici.

Potremo infine operare in campo industriale realizzando semplici controlli di processo, automatismi, e tutta una serie di altre interessantissime applicazioni.

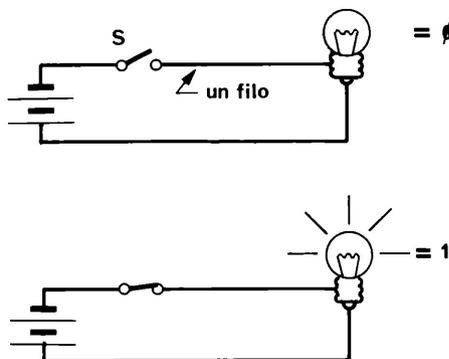


Fig. 2 - Rappresentazione pratica dei livelli logici 0 e 1. Un filo permette due combinazioni.

Come è composto l'AMICO 2000A

Il sistema a microprocessore che descriviamo in queste pagine è realizzato su un'unica scheda a circuito stampato ed è un vero e proprio microelaboratore completo ed autosufficiente in grado di eseguire programmi di piccola e media complessità. Con esso è possibile imparare a programmare e comprendere la filosofia dei microprocessori.

Il sistema AMICO 2000 è provvisto di una tastiera per l'immissione dei dati e dei programmi e di un visualizzatore a LED che funge da dispositivo di controllo e di "uscita" dati.

Le caratteristiche elettriche sono riportate a fine capitolo.

Struttura di un microelaboratore

Un microelaboratore è un sistema elettronico in grado di ricevere dall'esterno segnali elettrici, immagazzinarli, elaborarli secondo un certo programma, prendere delle decisioni e quindi emettere segnali elettrici utilizzabili all'esterno.

Essenzialmente un microelaboratore è composto da cinque "blocchi" funzionali:

CPU (Central Processing Unit): unità centrale di elaborazione.

RAM (Random Access Memory): memoria di lettura e scrittura.

ROM (Read Only Memory): memoria a sola lettura.

I/O (Input/Output): dispositivi di Ingresso/Uscita.

BUS: insieme di "fili" (o piste) sui quali si muovono i segnali elettrici e che collegano un "blocco" con l'altro.

La figura 1 mostra lo schema a blocchi del microelaboratore. Le frecce all'interno del BUS indicano il verso in cui si muovono i dati (sotto forma di gruppi di segnali elettrici) dentro l'elaboratore.

Per meglio comprendere come funziona ciascuno dei "blocchi" costituenti il microelaboratore faremo un diretto paragone con il corpo umano. Il cervello sarà allora costituito da CPU+ROM+RAM, mentre i sensi ed i movimenti del corpo possono essere assimilati al "blocco" di I/O (Ingresso/Uscita).

Allora avremo che:

CPU-analizza gli stimoli provenienti dall'esterno attraverso i sensi (vedi I/O); **ricerca** nella memoria ROM (vedi) e RAM (vedi) una risposta allo stimolo; **prende una decisione** in base a ciò che c'è nella memoria; **fa eseguire** l'azione.

ROM- può essere paragonata a tutto ciò che nella memoria umana è il bagaglio di conoscenza che derivano dalla educazione, dallo studio, e dalla esperienza. Esse *condizionano* il comportamento e *non possono essere modificate* (ciò non è assolutamente vero nel caso del cervello, ma lo assumeremo come tale per semplicità di trattazione).

RAM- Nella parte "RAM" del nostro cervello sono depositate, cancellate e nuovamente depositate tutte quelle informazioni che è necessario sapere momentaneamente o per uno specifico scopo, ma che non condizionano permanentemente il nostro comportamento.

I/O- gli input (ingressi) possono essere paragonati ai sensi tramite i quali si recepiscono le informazioni (vista, tatto, udito, etc), mentre gli output (uscite) sono le parole ed i movimenti del corpo.

BUS- i BUS dei dati non sono altro che *le interconnessioni nervose*, ovvero i mezzi che permettono le comunicazioni all'interno del corpo (leggi nel nostro caso microcalcolatore).

Un esempio:

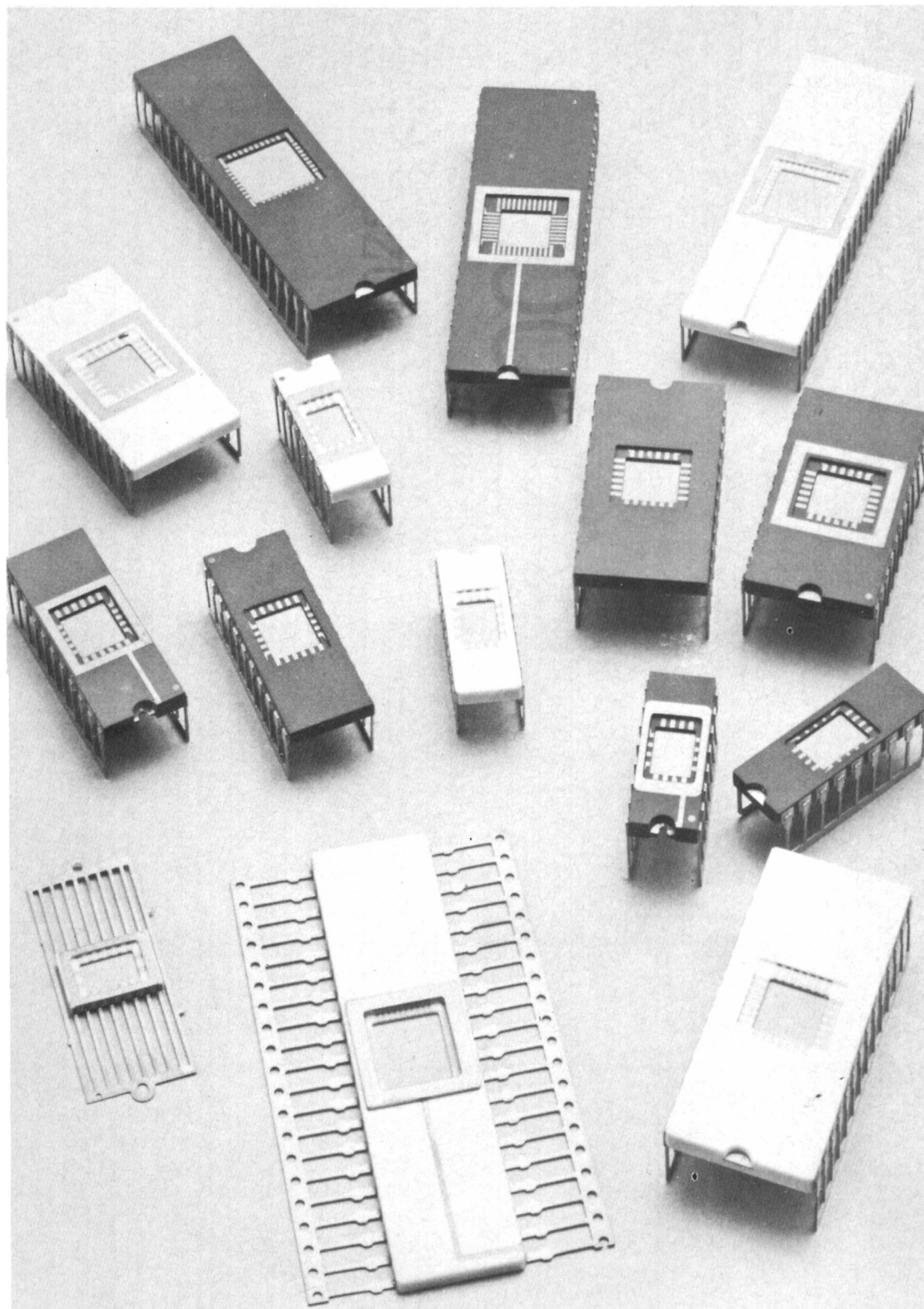
Sempre assimilando il funzionamento di un microelaboratore a quello di una persona e per meglio comprendere l'analogia consideriamo il seguente esempio: un autista che sta guidando la sua auto in una città che non conosce, chiede ad un passante dove si trovi una certa via. Quest'ultimo gli risponde che proseguendo diritto egli dovrà voltare a destra dopo il primo semaforo che incontra.

Allora: la CPU richiama dalla ROM tutte le nozioni necessarie per poter guidare l'automobile; ecco allora che quando scatta il rosso l'autista si ferma al semaforo (il suo comportamento è stato e sarà *sempre condizionato* dalla ROM ogni qualvolta si troverà di fronte ad un semaforo).

L'informazione momentanea "voltare a destra dopo il primo semaforo" è stata depositata nella RAM così che la CPU ne tiene conto solo *in quel determinato caso* dando il comando agli arti di eseguire la curva a destra dopo quel semaforo. Da allora in poi il nostro autista utilizzerà il programma "volta a destra dopo quel determinato semaforo" solo quando dovrà andare in quella certa via (infatti *non* volterà a destra ad ogni semaforo che incontra!).

Come funziona un microelaboratore

Abbiamo detto che all'interno di un microelaboratore circolano dei segnali elettrici che passano da un blocco all'altro attraverso il BUS dei dati.



Contenitori flat-pack e Dual-in-Line per circuiti integrati. LSI.

Questi particolari segnali elettrici si chiamano "bit" e non sono altro che livelli alti o bassi di tensione.

Con la parola "bit", contrazione delle due parole inglesi *binary digit*, si designa l'unità elementare di informazione ovvero uno dei due possibili stati o livelli logici 0 e 1.

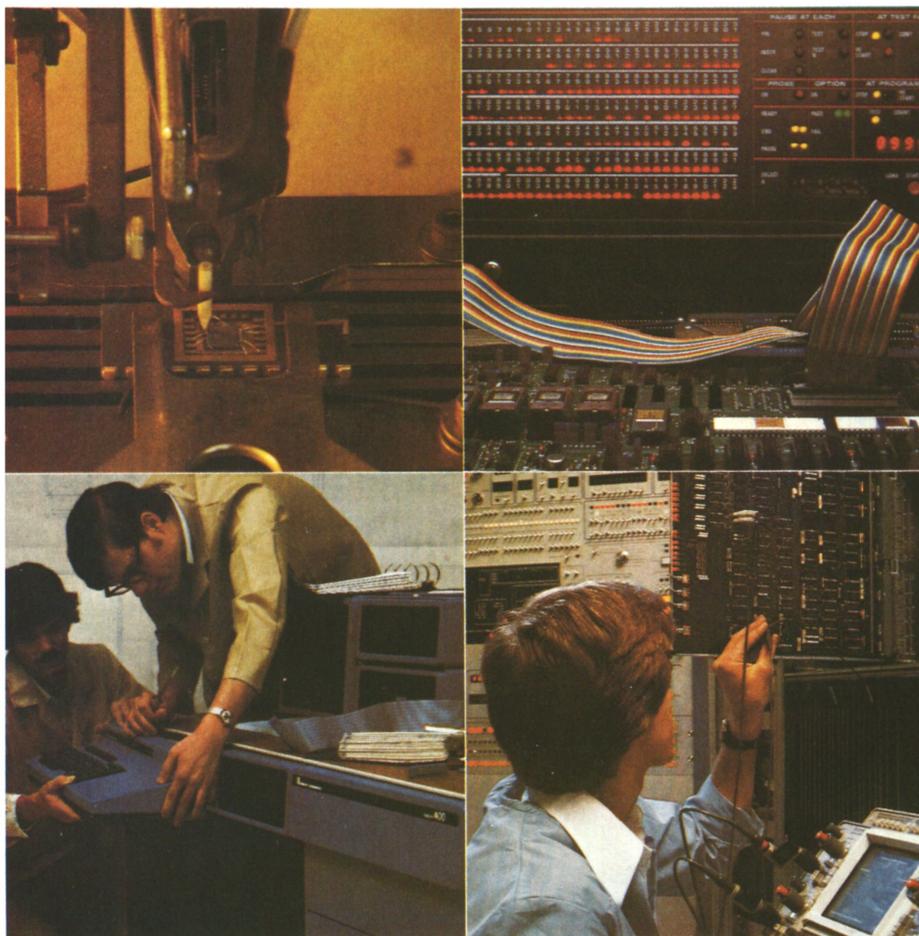
Si parla allora di livello "basso" o *stato logico "0"* (zero; barrato per non confonderlo con la lettera O) e di livello alto o *stato logico "1"* (uno).

Se consideriamo per esempio il semplicissimo circuito di fig. 2 a un solo filo (il ritorno di massa a terra non deve essere preso in considerazione) avremo uno stato logico "0" (lampadina spenta)

quando l'interruttore S è aperto, ed uno stato logico 1 (lampadina accesa) quando l'interruttore S è chiuso. Con un solo filo possiamo avere quindi due combinazioni (o 0 o 1) di un "bit". Se prendiamo ora in considerazione due fili, ovvero due "bit" (tralasciando sempre il ritorno di massa che è comune ad entrambi) possiamo avere quattro combinazioni diverse: 00; 10; 01; 11, come mostra la figura 3.

Siccome nell'elaboratore i segnali si muovono su più fili ognuno dei quali può portare il livello logico 0 o 1, avremo così che il numero di combinazioni possibili è dato dalla relazione:

$2^n =$ numero di combinazioni dove n è il numero dei fili.



Prove e collaudi hardware e software e sui sistemi elettronici di elaborazione dati.

Allora con 8 fili ovvero con una serie o "parola" di 8 bit (detta solitamente byte) è possibile avere $2^8 = 256$ diverse combinazioni e con 16 fili, ovvero 16 bit, è possibile avere $2^{16} = 65536$ diverse combinazioni. Presto vedremo perché sono particolarmente importanti i valori 8 e 16. La tabella 1 mostra le potenze crescenti da 2^0 a 2^{16} . Il tipo di numerazione che abbiamo appena esaminato e che utilizza i due stati 0 e 1 è detta numerazione binaria o *codice digitale binario*.

Con un numero binario, che è costituito da una serie di bit (o stati logici 0 e 1) è possibile rappresentare un numero decimale. Allora il numero binario di 4 bit 0110_2 (l'indice 2 rappresenta il sistema di conteggio binario - in base 2) equivale al numero decimale 6_{10} .

Per verificarlo basta usare la seguente formula di trasformazione:

$$0110 = (0 \times 2^3) + (1 \times 2^2) + (1 \times 2^1) + (0 \times 2^0) = 6$$

infatti l'esatto parallelo col sistema decimale mostra che il numero 4765 viene

ad esempio espresso in base 10 (cioè decimale):

$$4765 = (4 \times 10^3) + (7 \times 10^2) + (6 \times 10^1) + (5 \times 10^0)$$

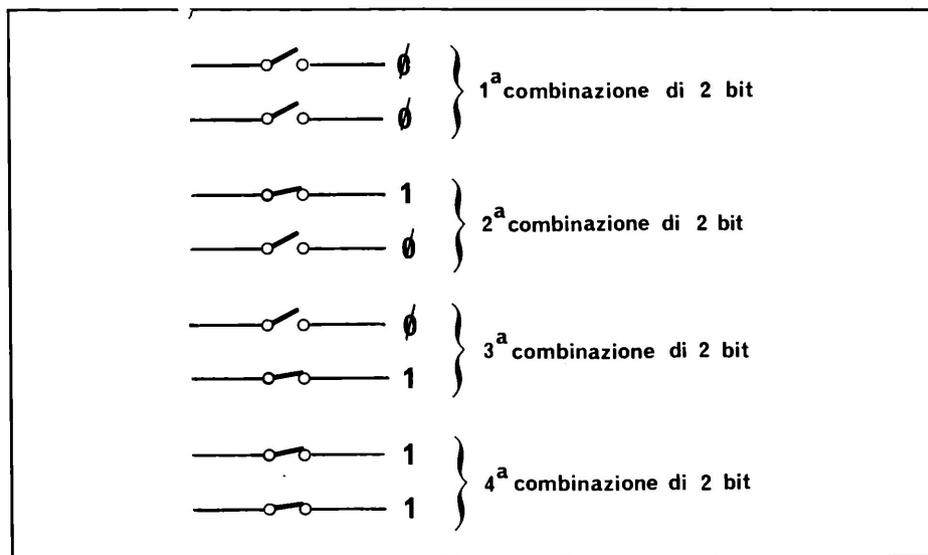


Fig. 3 - Con due fili (due bit) è possibile ottenere quattro differenti combinazioni.

Dal codice binario al codice esadecimale

Per poter comunicare con il microcalcolatore dovremo allora immettere i dati come serie di 0 e di 1 o parole di un certo numero di bit (nel nostro caso fino ad un massimo di 16 per volta).

In questo caso noi comunichiamo con l'elaboratore in "linguaggio macchina" ovvero nel linguaggio proprio della "macchina" calcolatore, che, come abbiamo visto "ragiona" in binario.

Sarebbe però troppo lungo, noioso e fonte di errori immettere parole fatte di 0 e 1, così che al posto del codice binario si può utilizzare un più semplice codice detto ESADECIMALE.

Questo codice fa riferimento al sistema di calcolo esadecimale che si fonda su una base, o radice, di 16, così come il sistema decimale si fonda su una base di 10 e quello binario su una base di 2.

Come si trasforma il codice binario in codice esadecimale?

È molto semplice: si considera una parola di 4 bit nelle sue possibili (e crescenti come valore) $2^4 = 16$ diverse combinazioni assegnando i valori dallo 0 al 9 per i primi dieci numeri e le lettere dalla A alla F per i successivi sei numeri binari. La tabella 2 mostra due successive trasformazioni: da binario a decimale ed a esadecimale.

Allora se dobbiamo scrivere e/o comunicare all'elaboratore ad esempio la "parola" di 16 bit:

0111101010111011

basterà scrivere 7ABB, cioè scomporre la parola in 4 gruppi di 4 bit ciascuno come mostrato in fig. 4.

In pratica ciò avviene attraverso una tastiera esadecimale (organo di input) a sedici tasti numerati da 0 a F presente

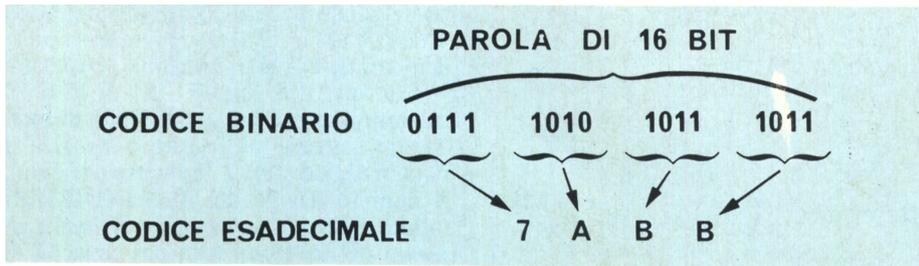


Fig. 4 - Trasformazione di una serie o PAROLA di 16 bit da binario ad esadecimale.

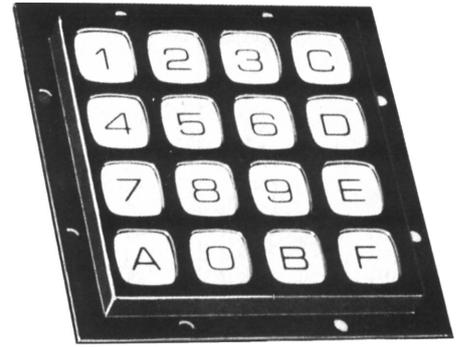


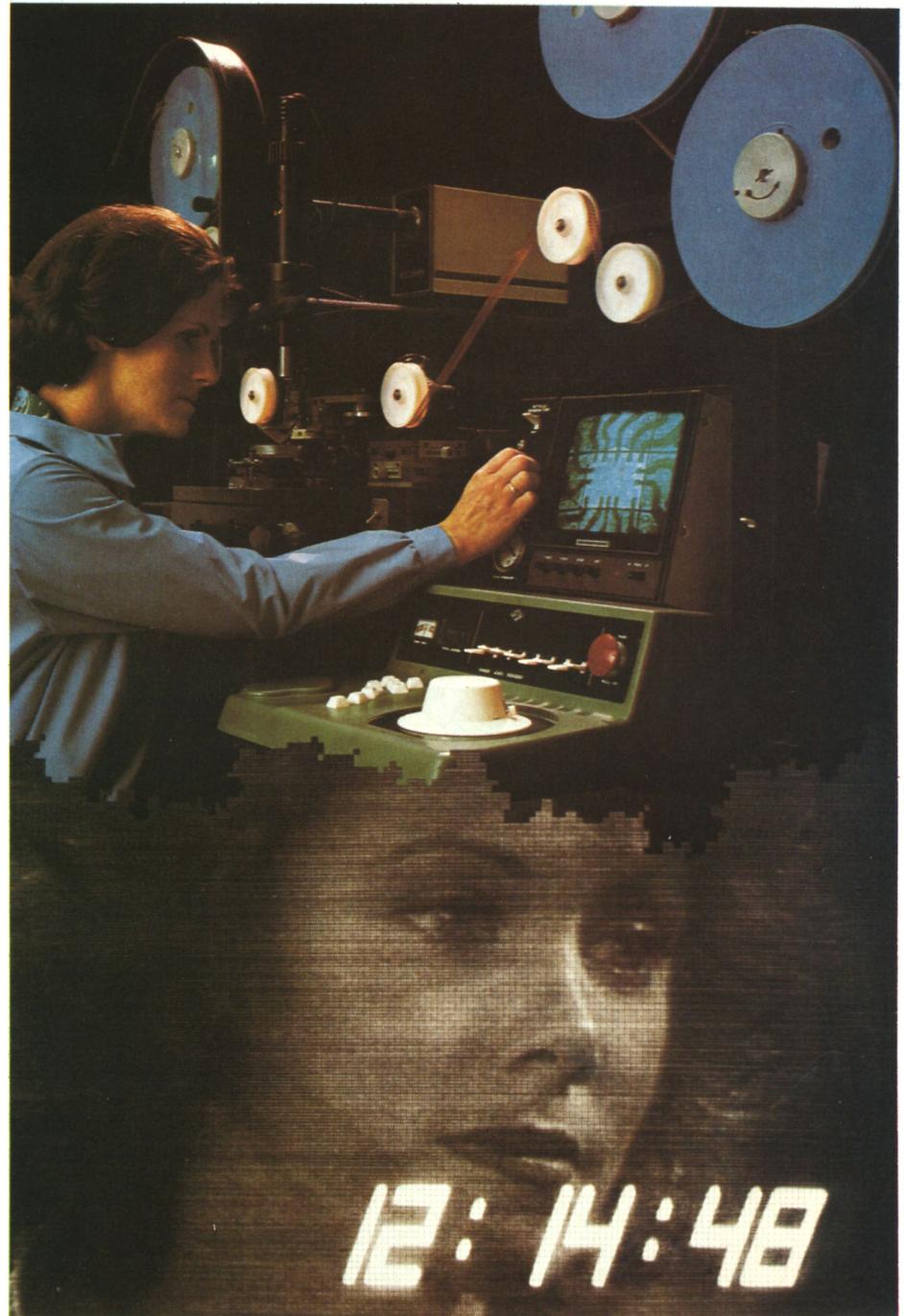
Fig. 5 - Tastiera esadecimale.

sul nostro microcomputer (vedi figura 5). Quando si preme un tasto un programma presente in ROM riconosce quale tasto è stato premuto e associa il valore binario corrispondente (4 bit) al carattere esadecimale introdotto (vedi tabella precedente). Per esempio premendo il tasto corrispondente al carattere esadecimale D il calcolatore riconoscerà il numero binario 1101.

un altro BUS detto BUS INDIRIZZI formato generalmente da 16 fili (nel nostro caso sarà infatti di 16 fili), per mezz-

Movimento e controllo dei dati in un microprocessore a 8 bit.

Il nostro microcalcolatore è basato su una CPU, ovvero su un **microprocessore** a 8 bit. Ciò significa che esso è in grado di trattare 8 bit contemporaneamente. I dati, sotto forma di parole costituite da 8 bit ovvero byte (8 bit = 1 byte), si muovono in ingresso ed in uscita nel BUS DATI formato da 8 fili. Esiste anche



Al centro dell'immagine una fase di fabbricazione di un circuito integrato.

TABELLA 1 - Potenze di 2

$2^0 = 1$
$2^1 = 2$
$2^2 = 4$
$2^3 = 8$
$2^4 = 16$
$2^5 = 32$
$2^6 = 64$
$2^7 = 128$
$2^8 = 256$
$2^9 = 512$
$2^{10} = 1.024$
$2^{11} = 2.048$
$2^{12} = 4.096$
$2^{13} = 8.192$
$2^{14} = 16.384$
$2^{15} = 32.768$
$2^{16} = 65.536$

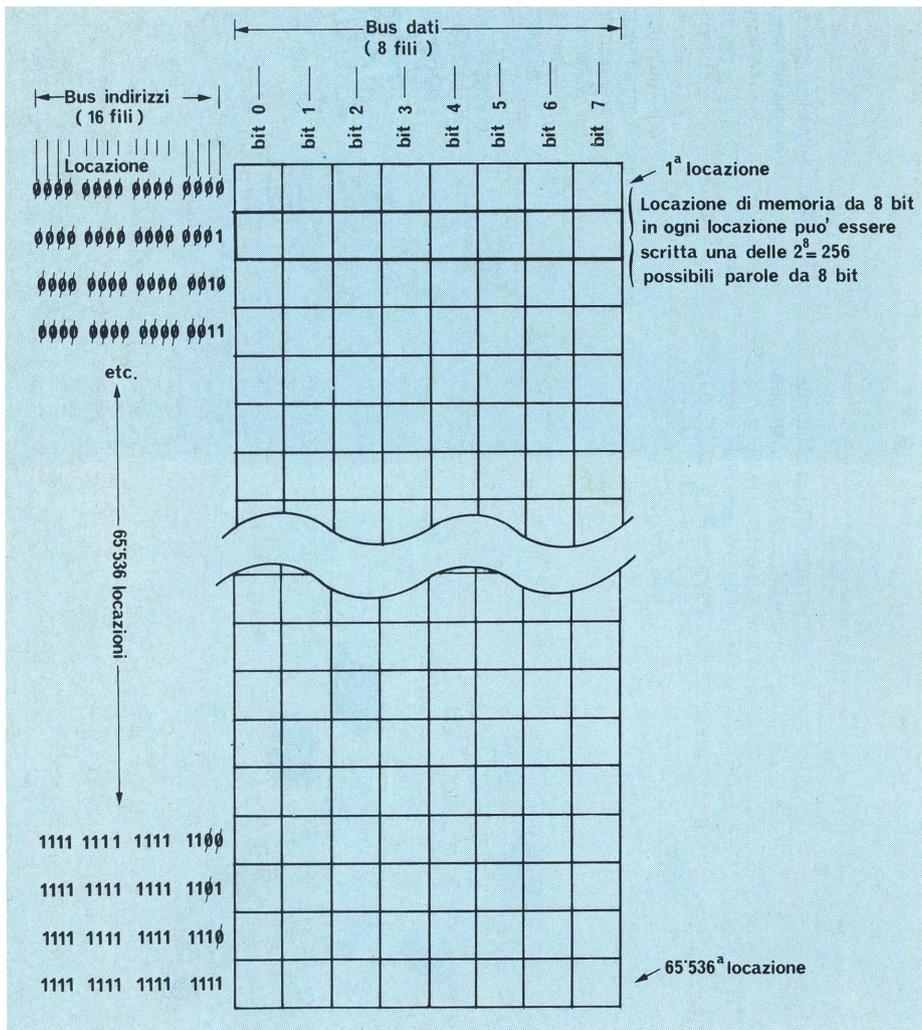


Fig. 6 - Schema che illustra il sistema di indirizzamento di memoria del nostro microcalcolatore. Sono indirizzabili $2^{16} = 65536$ locazioni di memoria da 8 bit (un Byte) ciascuna.

zo del quale la CPU *legge* (nella memoria ROM, RAM o dall'I/O) oppure *scrive* (nella memoria RAM) o *invia* (all'I/O) il dato.

Con un BUS INDIRIZZI di 16 fili il microprocessore ha la possibilità di indirizzare, ovvero di *leggere* o *scrivere* il dato in 65.536 (cioè 2^{16}) diverse posizioni. Il numero dei fili del BUS INDIRIZZI infatti definisce la *capacità di indirizzamento* del sistema, dove per capacità di indirizzamento si intende la quantità di **LOCAZIONI DI MEMORIA** che il microprocessore è in grado di leggere e/o di scrivere.

Per *locazione di memoria* si intende l'insieme di 8 celle elementari (ciascuna in grado di memorizzare un bit) in cui vengono conservati 8 bit ovvero 8 stati logici (esempio 10011010). Ad ognuna delle possibili 65536 locazioni corrisponde uno ed un solo indirizzo. In ogni locazione di memoria (che, come abbiamo visto contiene 8 bit, cioè un byte), attraverso il BUS DATI (che è formato da otto fili) si potrà depositare o leggere una delle $2^8 = 256$ diverse combinazioni binarie possibili con otto bit.

La figura 6 rappresenta schematicamente la memoria potenziale del nostro sistema a microprocessore (ovvero la massima capacità di memoria da esso indirizzabile)

Il microprocessore è quindi capace di leggere o scrivere parole da 8 bit in una qualsiasi delle 65536 locazioni dalla 0000000000000000 (0000 in esadecimale) alla 1111111111111111 (FFFF in esadecimale).

TABELLA 2 - Trasformazione da numerazione binaria decimale a esadecimale				
BINARIO		DECIMALE		ESADECIMALE
0000	=	0	=	0
0001	=	1	=	1
0010	=	2	=	2
0011	=	3	=	3
0100	=	4	=	4
0101	=	5	=	5
0110	=	6	=	6
0111	=	7	=	7
1000	=	8	=	8
1001	=	9	=	9
1010	=	10	=	A
1011	=	11	=	B
1100	=	12	=	C
1101	=	13	=	D
1110	=	14	=	E
1111	=	15	=	F

CARATTERISTICHE TECNICHE DEL MICROELABORATORE AMICO 2000

- CPU: microprocessore 6502
- Memoria RAM: fino a 2K byte sulla scheda
- Memoria ROM: 1K byte con Monitor e gestione cassette
- Tastiera esadecimale
- 7 tasti funzionali + deviatore per passo singolo
- Visualizzatore LED a 6 cifre
- Interfaccia parallelo 8 bit (Port di Input/Output)
- Interfaccia per registratore a cassette
- Clock quarzato da 1 MHz
- Regolatore di tensione incorporato
- Protezione contro l'inversione di polarità
- Alimentazione: 5 Volt, 800 mA max
- Espandibile: a mezzo connettore 40 poli
- Circuito stampato doppia faccia in vetronite
- Dimensioni: 300 x 160 mm.

IL LINGUAGGIO DEL MICROELABORATORE

Cosa è un programma

Partendo da certi input (dati in ingresso) e volendo ottenere certi output (dati in uscita) occorre far muovere i segnali così come noi vogliamo; questo risultato è ottenibile grazie ad un insieme di istruzioni (o comandi) che possiamo combinare logicamente secondo le nostre esigenze, insieme che chiameremo programma.

Tutti i progetti che non utilizzano il microprocessore infatti sono progetti "specifici", che esplicano cioè una ben determinata funzione (un frequenzimetro, un voltmetro digitale, un contatore, etc.) e realizzati dalle diverse combinazioni di circuiti integrati. Ogni progetto espleta solamente una determinata funzione, è uno strumento "specifico", in grado di svolgere solo quella specifica attività per la quale è stato disegnato.

Con il microelaboratore invece viene effettuato un salto di qualità rispetto a questo tipo di impostazione perchè, cambiando semplicemente programma, è possibile cambiare la funzione del microelaboratore, che infatti è uno strumento flessibile e di utilizzo universale, cioè non più specifico.

Sostituendo il programma sarà possibile cambiare in pochi istanti l'utilizzo del microelaboratore, passando per esempio da un gioco al controllo del riscaldamento domestico o di un processo produttivo.

Il microcomputer è in grado di eseguire un insieme di operazioni elementari (ad esempio $A + B$, $A \text{ and } B$, $A = A$) che, opportunamente disposte, permetteranno di creare applicazioni nei più svariati campi ed anche giochi di

vario genere. Nessun limite alle applicazioni tranne la fantasia e lo spirito di inventiva.

Il microelaboratore e il suo linguaggio

Riprendendo il paragone iniziato nel capitolo precedente tra microelaboratore e uomo, possiamo dire che volendo, ad esempio, far eseguire ad un nostro amico una addizione tra due numeri gli diremo: "Somma il numero A al numero B e scrivi il risultato su un foglio". Ci siamo avvalsi di un linguaggio che anche il nostro amico parla e capisce, e che associa ad un vocabolo come "somma" una operazione univocamente identificabile ed eseguibile su dei dati che pure vengono forniti in modo definito.

Anche il microelaboratore parla un suo linguaggio (linguaggio macchina) ed è in grado di fare esattamente ciò che vogliamo, se sappiamo parlargli nel suo linguaggio. Come avevamo accennato nel capitolo 1° il microelaboratore lavora in termini binari e pertanto il suo linguaggio originale è binario. D'altro canto abbiamo visto che sarebbe troppo lungo e fonte di molti errori utilizzare insieme di 0 e di 1 per parlare con il microelaboratore, essendo il "linguaggio" basato sul codice binario troppo diverso da quello che comunemente utilizziamo per risolvere i nostri problemi. Come fare allora per comunicare in modo più semplice con il nostro microelaboratore indicandogli le operazioni che deve svolgere?

In un primo momento viene analizzato il problema e sviluppato un programma nel nostro linguaggio, cioè in

termini colloquiali, come ad esempio "somma il numero A al numero B e scrivi il risultato". Successivamente questo viene tradotto in un linguaggio intermedio fra noi e la macchina, cioè un linguaggio che noi possiamo capire essendo esso "simbolico" o "mnemonico" in quanto ogni istruzione è formata da abbreviazioni di parole che associamo immediatamente ad una operazione che ci è ben chiara.

A questo punto è facile scrivere a fianco la traduzione esadecimale di questo linguaggio intermedio, cioè quella comprensibile direttamente dalla macchina. Noi forniremo pertanto i programmi in esadecimale, e la macchina provvederà alla ulteriore traduzione diretta in binario.

Il passaggio più complesso è pertanto quello da linguaggio intermedio (mnemonico) a linguaggio macchina in esadecimale, passaggio che viene effettuato o manualmente tramite una tabella, (vedi tabella 3) oppure, in maniera più evoluta tramite un apposito programma chiamato Assembler o traduttore.

Un esempio pratico

Per meglio comprendere tutto ciò facciamo un esempio applicativo piuttosto elementare, ma molto efficace ai nostri fini.

Supponiamo di voler eseguire la seguente operazione: $3 + 5 = 8$ e supponiamo anche di aver già introdotto il numero "3" (0000011) in binario e 03 in esadecimale) nella sesta locazione di memoria e il numero "5" (0000101 in binario e 05 in esadecimale) nella settima locazione di memoria.

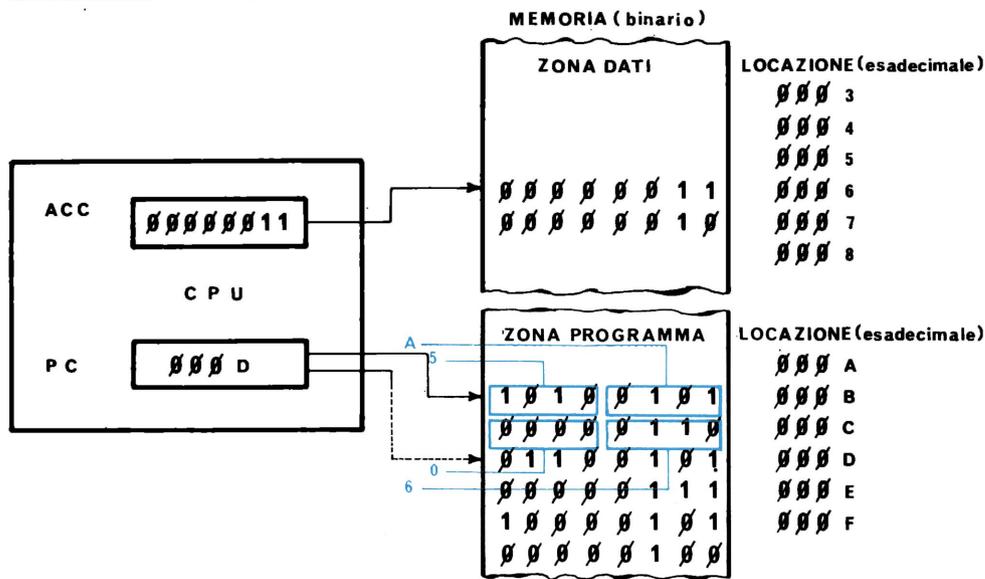
Tavola A - Rappresentazione dei "linguaggi" e del movimento dei dati all'interno del microelaboratore in base alle istruzioni contenute in un semplice programma che consente di sommare due numeri.

ISTRUZIONI	LINGUAGGIO DISCORSIVO (il nostro, non comprensibile dal microelaboratore)	LINGUAGGIO SIMBOLICO (intermedio tra il nostro e quello del micro)	LINGUAGGIO MACCHINA (è il linguaggio del microelaboratore)
1 ^a istruzione	Preleva dalla sesta posizione di memoria RAM il dato e trasportalo nel registro interno (ACCUMULATORE)* dell'unità di elaborazione (CPU).	LDA 06 (LDA sta per Load Accumulator e 06 indica in esadecimale la posizione di memoria nella quale si trova il dato da trasferire).	A5 06 (A5 è la traduzione di LDA in esadecimale).
2 ^a istruzione	Preleva il dato contenuto nella settima posizione di memoria RAM e sommalo al dato contenuto nel registro interno (ACCUMULATORE) della CPU.	ADC 07 (ADC sta per Add Memory to Accumulator con Carry; la funzione del Carry sarà spiegata in seguito. 07 indica, in esadecimale, la posizione di memoria che contiene il dato da sommare all'ACCUMULATORE.	65 07 (65 è la traduzione di ADC in esadecimale).
3 ^a istruzione	Trasferisci il risultato dall'ACCUMULATORE alla quarta locazione di memoria RAM.	STA 04 (STA significa Store Accumulator, istruzione di scrittura nella RAM). 04 indica in esadecimale la posizione di memoria nella quale deve essere ricopiato il contenuto dello ACCUMULATORE.	85 04 (85 è la traduzione di STA in esadecimale).

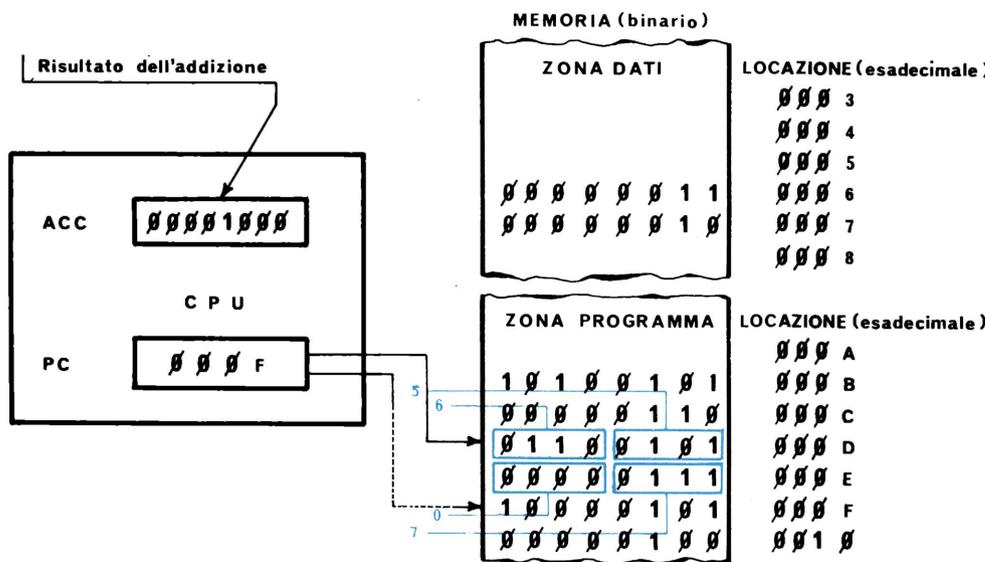
* Il significato del termine viene spiegato in questo capitolo.

RAPPRESENTAZIONE DELLO STATO DEL SISTEMA

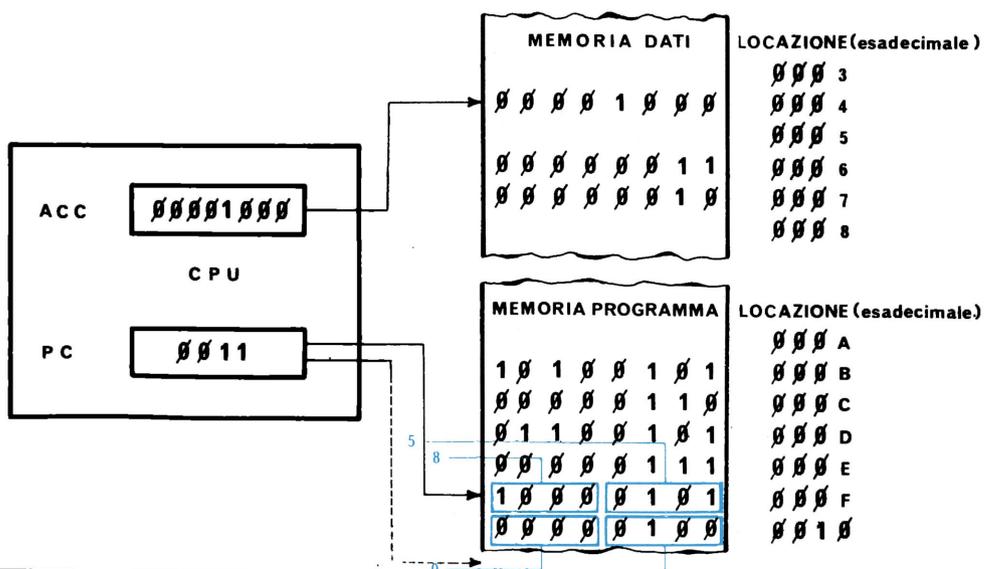
NOTE



Si tratta di una istruzione da due bytes, contenuta nella posizione di memoria 000B e 000C, posizioni indicate dal Program Counter (PC) prima della esecuzione. Eseguita l'istruzione il contenuto del PC è 000D che rappresenta l'indirizzo da cui verrà prelevata la prossima istruzione.

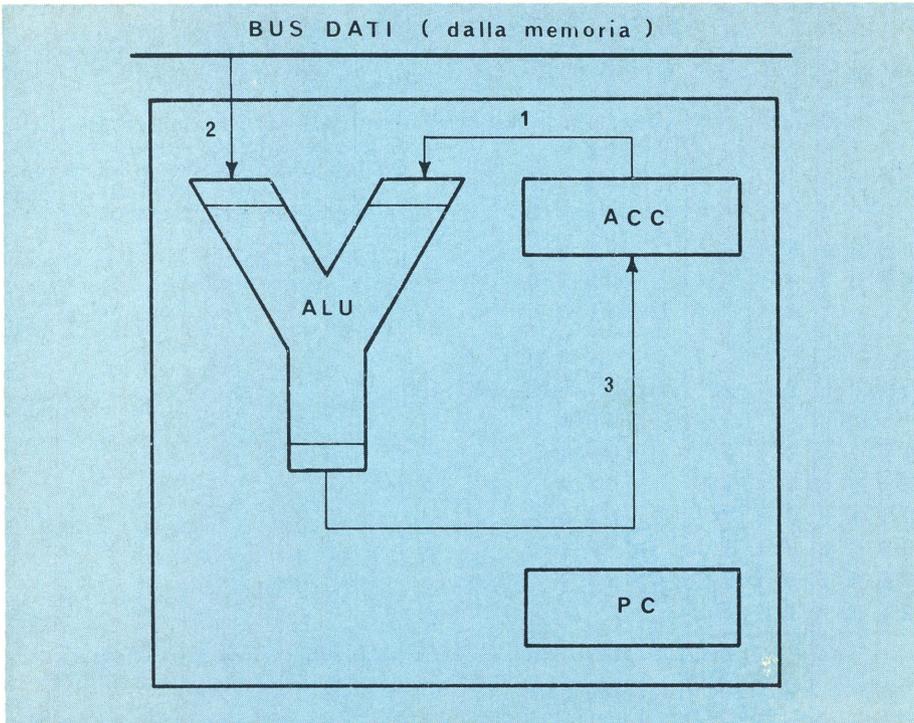


Anche questa è una istruzione da due bytes. Il dato contenuto nella settima posizione di memoria viene sommato al contenuto dell'ACCUMULATORE che, a istruzione eseguita, contiene il risultato dell'addizione.



Come si vede dalla figura a lato il contenuto dell'ACCUMULATORE è stato ricopiato nella locazione di memoria 0004. Il contenuto dell'ACCUMULATORE rimane invariato.

Le frecce tratteggiate relative alla posizione del Programm. Computer indicano la situazione ad istruzione eseguita.



Alcuni dettagli della CPU: Unità aritmetico-logica (ALU), Accumulatore (ACC) e Program Counter (PC).

Vogliamo ora che il microelaboratore esegua la somma e che depositi il risultato nella quarta posizione (o locazione) di memoria. Questo insieme di operazioni può essere rappresentato come nella tavola A.

La tavola A è da studiare con attenzione: in essa vengono spiegati alcuni

concetti fondamentali che, se ben compresi, permetteranno di seguire con molta facilità il resto della trattazione. In sostanza vengono esaminate tre delle più importanti istruzioni del microprocessore 6502 che è il "cervello" dell'AMICO 2000A. E così in seguito saranno analizzate via via tutte le istru-

zioni in modo da permettervi alla fine di programmare autonomamente il vostro microcomputer.

Il semplice insieme di istruzioni considerato nella tavola A permette quindi di sommare due numeri e di porre il risultato in memoria; già queste poche istruzioni possono costituire un "programma".

Un programma può risiedere sia in memoria ROM che in memoria RAM; il primo caso è classico per il programma chiamato MONITOR, che permette di eseguire alcune attività basilari come, ad esempio, quelle di input ed output dei

dati. I programmi che effettueremo noi a scopo didattico o per diletto, verranno invece solitamente immessi in RAM, memoria che può essere scritta e letta dall'utilizzatore e pertanto modificata a piacere.

Per inciso notiamo che i programmi contenuti nella memoria RAM vengono mantenuti finché permane l'alimentazione.

Togliendo l'alimentazione i programmi contenuti in RAM vanno a carte quarantotto!! Ciò non avviene per quelli contenuti in ROM.

Le operazioni di esecuzione del programma si svolgono in pratica nel modo seguente (Vedere la figura nella Tavola A).

La CPU legge la prima istruzione del programma, la interpreta e la esegue; a questo punto la CPU legge l'istruzione successiva, la interpreta ed esegue, e procede nello stesso modo in maniera sequenziale fino alla completa esecuzione del programma, cioè fino a che non troverà una istruzione di fine programma.

La CPU è in grado di capire ed eseguire un certo numero di ordini diversi, che costituiscono l'insieme delle possibili istruzioni, o set di istruzioni.

In particolare, per il microprocessore che è utilizzato nel microelaboratore AMICO 2000, il set è composto da 56 istruzioni, ma esistono anche dei microprocessori con oltre 150 istruzioni.

Tabella 3 - Traduzione delle istruzioni del microprocessore 6502 (che è la CPU utilizzata nel microcomputer AMICO 2000A) da rappresentazione in CODICE ESADECIMALE a LINGUAGGIO SIMBOLICO.

00 - BRK	24 - BIT - Zero Page
01 - ORA - (Indirect, X)	25 - AND - Zero Page
05 - ORA - Zero Page	26 - ROL - Zero page
06 - ASL - Zero Page	28 - PLP
08 - PHP	29 - AND - Immediate
09 - ORA - Immediate	2A - ROL - Accumulator
0A - ASL - Accumulator	2C - BIT - Absolute
0D - ORA - Absolute	2D - AND - Absolute
0E - ASL - Absolute	2E - ROL - Absolute
10 - BPL	30 - BMI
11 - ORA - (Indirect, Y)	31 - AND (Indirect), Y
15 - ORA - Zero Page, X	35 - AND - Zero Page, X
16 - ASL - Zero Page, X	36 - ROL - Zero Page, X
18 - CLC	38 - SEC
19 - ORA - Absolute, Y	39 - AND - Absolute, Y
1D - ORA - Absolute, X	3D - AND - Absolute, X
1E - ASL - Absolute, X	3E - ROL - Absolute, X
20 - JSR	
21 - AND - (Indirect, X)	

Il program counter

Evidentemente la CPU non può cominciare ad eseguire istruzioni partendo da un punto qualsiasi della memoria e seguendo un ordine casuale, ma deve iniziare l'esecuzione partendo da un punto ben preciso, che rappresenta l'inizio del programma (nel caso particolare del nostro esempio la locazione di memoria 000B).

In concreto nella CPU si trova un importantissimo registro, chiamato Program Counter (contatore di programma) che contiene l'indirizzo della locazione di memoria dalla quale verrà

Segue Tabella 3

40 - RTI	A2 - LDX - Immediate
41 - EOR - (Indirect, X)	A4 - LDY - Zero Page
45 - EOR - Zero page	A5 - LDA - Zero Page
46 - LSR - Zero Page	A6 - LDX - Zero Page
48 - PHA	A8 - TAY
49 - EOR - Immediate	A9 - LDA - Immediate
4A - LSR - Accumulator	AA - TAX
4C - JMP - Absolute	AC - LDY - Absolute
4D - EOR - Absolute	AD - LDA - Absolute
4E - LSR - Absolute	AE - LDX - Absolute
50 - BVC	B0 - BCS
51 - EOR - (Indirect), Y	B1 - LDA - (Indirect), Y
55 - EOR - Zero Page, X	B4 - LDY - Zero Page, X
56 - LSR - Zero Page, X	B5 - LDA - Zero Page, X
58 - CLI	B6 - LDX - Zero Page, Y
59 - EOR - Absolute, Y	B8 - CLV
5D - EOR - Absolute, X	B9 - LDA - Absolute, Y
5E - LSR - Absolute, X	BA - TSX
60 - RTS	BC - LDY - Absolute, X
61 - ADC - (Indirect, X)	BD - LDA - Absolute, X
65 - ADC - Zero Page	BE - LDX - Absolute, Y
66 - ROR - Zero Page	C0 - CPY - Immediate
68 - PLA	C1 - CMP - (Indirect, X)
69 - ADC - Immediate	C4 - CPY - Zero Page
6A - ROR - Accumulator	C5 - CMP - Zero Page
6C - JMP - Indirect	C6 - DEC - Zero Page
6D - ADC - Absolute	C8 - INY
6E - ROR - Absolute	C9 - CMP - Immediate
70 - BVS	CA - DEX
71 - ADC - (Indirect), Y	CC - CPY - Absolute
75 - ADC - Zero Page, X	CD - CMP - Absolute
76 - ROR - Zero Page, X	CE - DEC - Absolute
78 - SEI	D0 - BNE
79 - ADC - Absolute, Y	D1 - CMP - (Indirect), Y
7D - ADC - Absolute, X	D5 - CMP - Zero page, X
7E - ROR - Absolute, X	D6 - DEC - Zero Page, X
81 - STA - (Indirect, X)	D8 - CLD
84 - STY - Zero Page	D9 - CMP - Absolute, Y
85 - STA - Zero Page	DD - CMP - Absolute, X
86 - STX - Zero Page	DE - DEC - Absolute, X
88 - DEY	E0 - CPX - Immediate
8A - TXA	E1 - SBC - (Indirect, X)
8C - STY - Absolute	E4 - CPX - Zero Page
8D - STA - Absolute	E5 - SBC - Zero Page
8E - STX - Absolute	E6 - INC - Zero Page
90 - BCC	E8 - INX
91 - STA (Indirect), Y	E9 - SBC - Immediate
94 - STY - Zero Page, X	EA - NOP
95 - STA - Zero page, X	EC - CPX - Absolute
96 - STX - Zero page, Y	ED - SBC - Absolute
98 - TYA	EE - INC - Absolute
99 - STA - Absolute, Y	F0 - BEQ
9A - TXS	F1 - SBC (Indirect), y
9D - STA - Absolute, X	F5 - SBC - Zero Page X
A0 - LDY - Immediate	F6 - INC - Zero Page, X
A1 - LDA - (Indirect, X)	F8 - SED
	F9 - SBC - Absolute, Y
	FD - SBC - Absolute, X
	FE - INC - Absolute, X

prelevata la prossima istruzione del programma che la CPU deve eseguire. In linguaggio più tecnico il Program Counter (abbreviato PC) "punta" (ovvero indica alla CPU) la prossima i-

struzione da eseguire. Per questo motivo si parla anche di Puntatore (in inglese "Pointer"). Dopo aver eseguito una istruzione del programma il Program Counter (PC)

si posiziona automaticamente sull'indirizzo della istruzione da eseguire subito dopo e non richiede pertanto alcun intervento da parte nostra.

In particolare riferendoci all'esempio riportato, il Program Counter (posto in partenza a 000B dall'utente) sarà 000B all'inizio del programma, 000D dopo aver eseguito la prima istruzione, 000F dopo aver eseguito la seconda e 0011 dopo aver effettuato la terza. Il fatto che il Program Counter incrementi di 2 è dovuto al tipo di istruzione che la macchina sta eseguendo: in pratica la maggior parte di istruzioni comporta un incremento di 2, ovvero esse occupano due locazioni di memoria consecutive (due byte). Esistono però anche istruzioni che occupano uno o tre byte; in questo caso il PC si incrementerà automaticamente di 1 e 3 rispettivamente, puntando sulla istruzione successiva.

Il contenuto del Program Counter può infine essere anche modificato dall'utilizzatore grazie ad alcune istruzioni che permettono di saltare da un punto all'altro del programma senza dover seguire una rigida sequenza di comandi.

Le istruzioni esaminate

Le tre istruzioni sin qui incontrate possono essere così esemplificate:

LDA M = M → A
ADC M = A + M → A

M = locazione di memoria qualsiasi

A = Accumulatore

+ = simbolo di somma

→ = Simbolo di trasferimento che indica il movimento del dato da a ...

Le due istruzioni LDA e STA consentono di muovere un dato (byte) tra memoria e Accumulatore. Queste operazioni sono fondamentali e ciò le rende tra le più utilizzate nei programmi.

Alcuni dettagli sulla CPU

Sino ad ora abbiamo accennato alla presenza, all'interno della CPU, dei seguenti elementi:

ACCUMULATORE (ACC)
PROGRAM COUNTER (PC)
ALU (Arithmetic-logic unit, unità aritmetico-logica).

L'Accumulatore ed il Program Counter sono due registri assai importanti, e l'ALU è il vero e proprio "cuore" della CPU.

Si può quindi per ora schematizzare la CPU come in figura 7. Facendo riferimento al nostro esempio analizziamo il flusso dei dati all'interno della CPU durante l'esecuzione dell'istruzione ADC 07 (cioè la somma).

Il primo dato, contenuto nell'accumulatore, viene trasferito ad uno dei due ingressi dell'ALU (freccia 1), immediatamente dopo, attraverso il Bus Dati viene presentato (freccia 2) al secondo ingresso dell'ALU il dato contenuto nella settima locazione di memoria (infatti l'istruzione era ADC 07, che vuol dire somma il contenuto della settima locazione all'accumulatore). Avendo a disposizione entrambi i dati necessari all'esecuzione dell'istruzione l'ALU esegue la somma al suo interno e trasferisce (freccia 3) il risultato dell'operazione nell'accumulatore, sostituendo il contenuto precedente. Tutte queste operazioni vengono eseguite in maniera automatica dalla CPU, e l'operatore non deve intervenire in alcun modo.

Fino a questo punto abbiamo chiarito i principi fondamentali della struttura del microelaboratore e del suo funzionamento. Dal capitolo III è necessario procedere ad approfondimenti dell'aspetto software che è la parte più importante di qualsiasi sistema a microprocessore.

La maniera migliore che consente una comprensione totale della materia è quella di poter disporre del microelaboratore per verificare la teoria immediatamente nella pratica.

Passiamo senz'altro, quindi, alla descrizione del montaggio e collaudo del microelaboratore AMICO 2000A la cui scatola di montaggio (o la scheda montata e collaudata) è disponibile per chi fosse interessato presso la A.S.E.L. s.r.l. di Milano come meglio specificato alla fine del libro.

MONTAGGIO E COLLAUDO DEL MICROELABORATORE

L'architettura dell'Amico 2000A

Con riferimento alla Fig. 8, nella scheda del microelaboratore sono chiaramente identificate diverse zone che costituiscono i cosiddetti "blocchi funzionali" dell'AMICO 2000A.

Essi sono così suddivisi:

Zona CPU: contiene il microprocessore 6502 e l'elettronica di supporto come l'oscillatore (74LS14), la logica per il funzionamento in single-step (74LS38)

e la logica per il "reset" iniziale (74LS14 e 74LS00).

Zona memoria e decodifica: contiene 2 K byte di memoria RAM ed 1 K byte di memoria PROM (Programmable Read Only Memory) che contiene il programma di controllo del registratore a cassette di cui si parlerà nel capitolo 5°.

Nella stessa zona sono anche presenti le PROM di decodifica 74S287.

Tastiera e display: questo settore contiene un display a sette segmenti a 6 cifre diviso in due zone, la prima delle quali, composta da 4 cifre, è il "display

indirizzi"; mentre la seconda, caratterizzata dalle due cifre più a destra, è il "display dati"

Anche la tastiera è suddivisa in due zone fondamentali. La zona con i tasti rossi è destinata all'introduzione dei dati in codice esadecimale (tasti da 0 ad F) mentre la zona con i tasti blu è invece una tastiera di comando che permette di selezionare a piacere una particolare funzione che si desidera eseguire.

Input/Output (I/O): in quest'area risiede un integrato LSI dotato di 3 porte da 8 bits ciascuna. Sono inoltre presenti

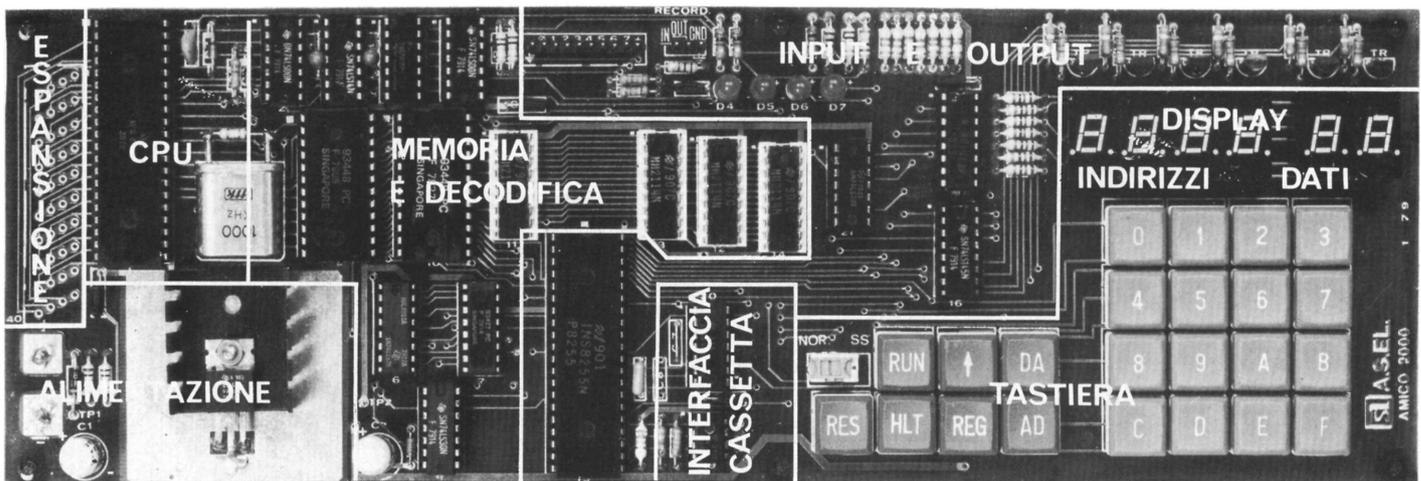


Fig. 8 - L'AMICO 2000/A montato. Le linee dividono la piastra in varie zone che corrispondono ai diversi blocchi funzionali del microcomputer. Prima del montaggio è necessario familiarizzarsi con i vari componenti.

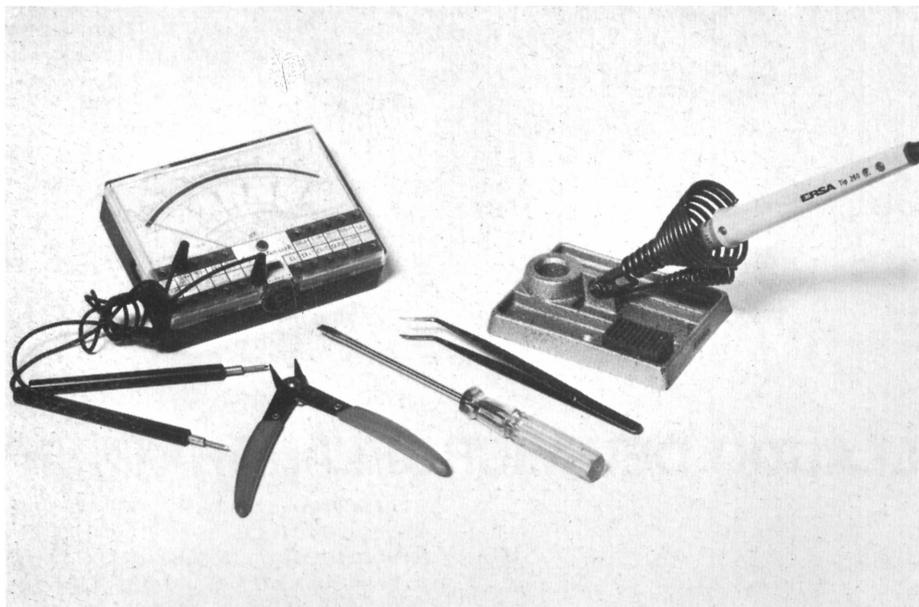


Fig. 9 - Attrezzatura indispensabile per il montaggio del microelaboratore. Raccomandiamo saldatore e stagno di qualità perché dalla bontà delle saldature dipende in massima parte il successo di un buon funzionamento.

degli integrati di decodifica e pilotaggio dei display (74LS145 e ULN2003).

I transistori TR2 ÷ TR7 vengono utilizzati per l'accensione sequenziale del display. In quest'area è anche presente una contattiera con 8 linee di input/output sfruttabili per comunicare con l'esterno.

Interfaccia Casseta: questa zona è già predisposta fin da ora per montare dei componenti che permettono di realizzare una interfaccia per registratore a cassetta. Ciò consentirà la memorizzazione di programmi e dati e la loro lettura ad alta velocità.

Alimentatore: comprende un regolatore integrato (TR1) che stabilizza la tensione a 5 Volt. Sull'ingresso è previsto un diodo in serie che viene utilizzato per protezione contro inversioni di polarità.

Analizzata in questo modo la funzione dei diversi "blocchi funzionali" presenti sulla piastra e riconoscibili anche sullo schema elettrico possiamo ora passare all'effettivo montaggio del microcomputer.

Il montaggio

Per assemblare correttamente il vostro AMICO 2000A occorre disporre innanzitutto di un saldatore da 20 - 30 W con una punta nuova e sottile, in modo da effettuare saldature precise evitando

pericolosi contatti cortocircuitanti. Raccomandiamo anche di utilizzare del filo di stagno sottile, con un diametro di un millimetro o poco più. In particolare consigliamo il tipo con fluxante interno.

È anche necessaria una normale pinzetta a becchi piatti e sottili, ed un tronchesino in mancanza del quale potete al limite usare un tagliaunghie (ma non le forbici!).

Infine, per il collaudo del buon montaggio, sarà molto utile un normale tester, con il quale controlleremo le tensioni di alimentazione e la continuità delle piste, per verificare l'assenza di eventuali cortocircuiti o interruzioni (vedi Fig. 9).

Dopo esservi accertati che nella scatola di montaggio siano presenti tutti i componenti riportati nella tabella 4 vi consigliamo di suddividerli in gruppi, omogenei, raggruppando tutte le resistenze, i diodi, i transistori, i condensatori e gli altri integrati.

Occorre prestare una notevole attenzione ai circuiti integrati che sono forniti su un supporto conduttore di protezione. Vi consigliamo di maneggiare il meno possibile questi componenti e di estrarli solamente quando dovete montarli sul circuito stampato.

Queste precauzioni coi circuiti integrati sono necessarie perché l'elettricità statica che si accumula sul corpo umano potrebbe danneggiare irreparabilmente i componenti MOS.

Raccomandiamo pertanto il montaggio a persone che abbiano un minimo di esperienza nella saldatura di circuiti integrati.

L'assemblaggio

Dopo aver suddiviso i componenti in gruppi possiamo cominciare a assemblare il nostro microcomputer a partire dai componenti passivi. Cominciamo ad identificare ed a dividere le resistenze.

Esse possono essere suddivise in cinque gruppi diversi a seconda del loro valore, riconoscibile attraverso la descrizione riportata nella tabella 5.

Effettuata anche questa ulteriore separazione possiamo cominciare a piegare i terminali (reofori) delle resistenze. Una volta piegati i reofori inseriamo la prima resistenza al suo posto come mostra la Fig. 10, saldiamo e tagliamo la parte eccedente dei terminali. Ripetiamo questa operazione fino a che non abbiamo saldato tutte le resistenze presenti nel kit cioè da R1 a R33.

Terminata questa operazione passiamo al montaggio degli zoccoli sui quali verranno poi collocati i circuiti integrati più delicati. La saldatura degli zoccoli è molto semplice: occorre solo evitare che le saldature dei piedini adiacenti si tocchino, creando in tal modo dei dannosi cortocircuiti.

Per effettuare un corretto montaggio degli zoccoli, che sono molto importanti, consigliamo di inserirli tutti curandone l'orientamento. L'orientamento dello zoccolo è lo stesso che dovrà avere l'integrato, ed il riferimento è un angolo smussato internamente in corrispondenza del piedino 1, mentre sull'integrato a questo piedino corrisponderà una tacca e/o un punto chiaro.

La Fig. 11 illustra la corrispondenza tra orientamento dell'integrato e dello zoccolo; l'angolo smussato dovrà essere in corrispondenza della tacca che identifica sulla serigrafia la posizione dell'integrato.

Cominciamo ad inserire gli zoccoli per gli integrati IC1 (40 piedini), IC2 (14 piedini), IC3 (14 piedini), IC4 (14 piedini), IC5 (14 piedini), IC6 (16 piedini), IC9 (24 piedini), IC11 e IC12 (18 piedini), IC15 (40 piedini), IC16 e IC17 (16 piedini), IC18/1-2-3-4-5-6 (14 piedini) saldandoli uno per volta e verificandone il corretto orientamento. In pratica vanno montati tutti con l'angolo interno smussato verso l'alto.

Finita questa operazione, e dopo esserci accertati di non aver creato dei contatti tra piedini adiacenti, possiamo passare al montaggio dei tasti. Facendo riferimento alla Fig. 10 inseriamo i tasti nella loro rispettiva posizione, voltiamo la piastra e saldiamo in un primo momento un solo piedino per ogni tasto. Senza tagliare né piegare i piedini ricapovolgiamo la piastra e verificiamo che i tasti siano tutti ben appoggiati sullo stampato; eventualmente ritocchiamo le saldature di quelli malposizionati, dopodiché,

quando siamo ben sicuri che tutto è in ordine, procediamo alla saldatura dei restanti piedini.

Nell'eseguire questo montaggio ricordatevi che i tasti, pur non essendo dei circuiti elettronici, temono il calore prolungato essendo composti di materiale plastico.

Terminato l'assemblaggio delle tastiere possiamo cominciare a montare diodi (D1-2-3 e lo Zener D8) e transistori. Per i primi il riferimento è una fascetta colorata che contraddistingue il catodo (anche qui pertanto il necessario rispettare la polarità seguente seguendo l'orientamento della serigrafia mentre per i transistori (montiamo inizialmente solo i TR2-3-4-5-6-7) possiamo fare riferimento alla Fig. 12 per una corretta disposizione dei piedini.

ci al gioco dei riflessi, diciamo che chi ferma il display su cifre comprese fra 30 e 40 ha i riflessi molto buoni, fra 50 e 70 sono normali, mentre oltre i 100... è meglio cominciare una cura di Gerovital!

Tabella 4 - Componenti e materiali dell'AMICO 2000/A nella versione minima con 1 k byte di RAM e senza l'interfaccia per registratore a cassetta.

6	:	resistori 3,9 k Ω
6	:	resistori 10 k Ω
1	:	resistore 220 k Ω
7	:	resistori 4,7 k
7	:	resistori 100 Ω
6	:	resistori 1,2 k Ω
2	:	cond. elettrolitici 47 μ F - 16 V
2	:	cond. elettrolitici 1 μ F - 16 V
1	:	cond. ceramico a disco 10 pF
3	:	condensatori a disco 47 nF oppure 100 nF
1	:	diodo 1N4001
2	:	diodi 1N4148
1	:	diodo zener 6,2 V-1 2 W
1	:	regolatore LM 340 T5 oppure μ A 7805
6	:	transistori BC 327
1	:	microprocessore 6502
2	:	integrati 74LS00
1	:	integrato 74LS14
1	:	integrato 74LS38 oppure 74LS03
1	:	integrato 74S287
1	:	integrato 93448
2	:	integrati TMS4045 oppure 2114
1	:	integrato 8255
1	:	integrato 74LS145
1	:	integrato ULN2003
6	:	display LED TIL312 oppure MAN72
1	:	quarzo 1 MHz
1	:	interruttore unipolare
23	:	tasti
6	:	piedini di gomma
1	:	dissipatore per TO220
2	:	capicorda
1	:	contattiera a 10 posti
2	:	zoccoli a basso profilo da 40 piedini
1	:	zoccolo a basso profilo da 24 piedini
2	:	zoccoli a basso profilo da 18 piedini
3	:	zoccoli a basso profilo da 16 piedini
10	:	zoccoli a basso profilo da 14 piedini
1	:	circuito stampato a doppia faccia in vetronite forato e serigrafato 300 mm x 100 mm

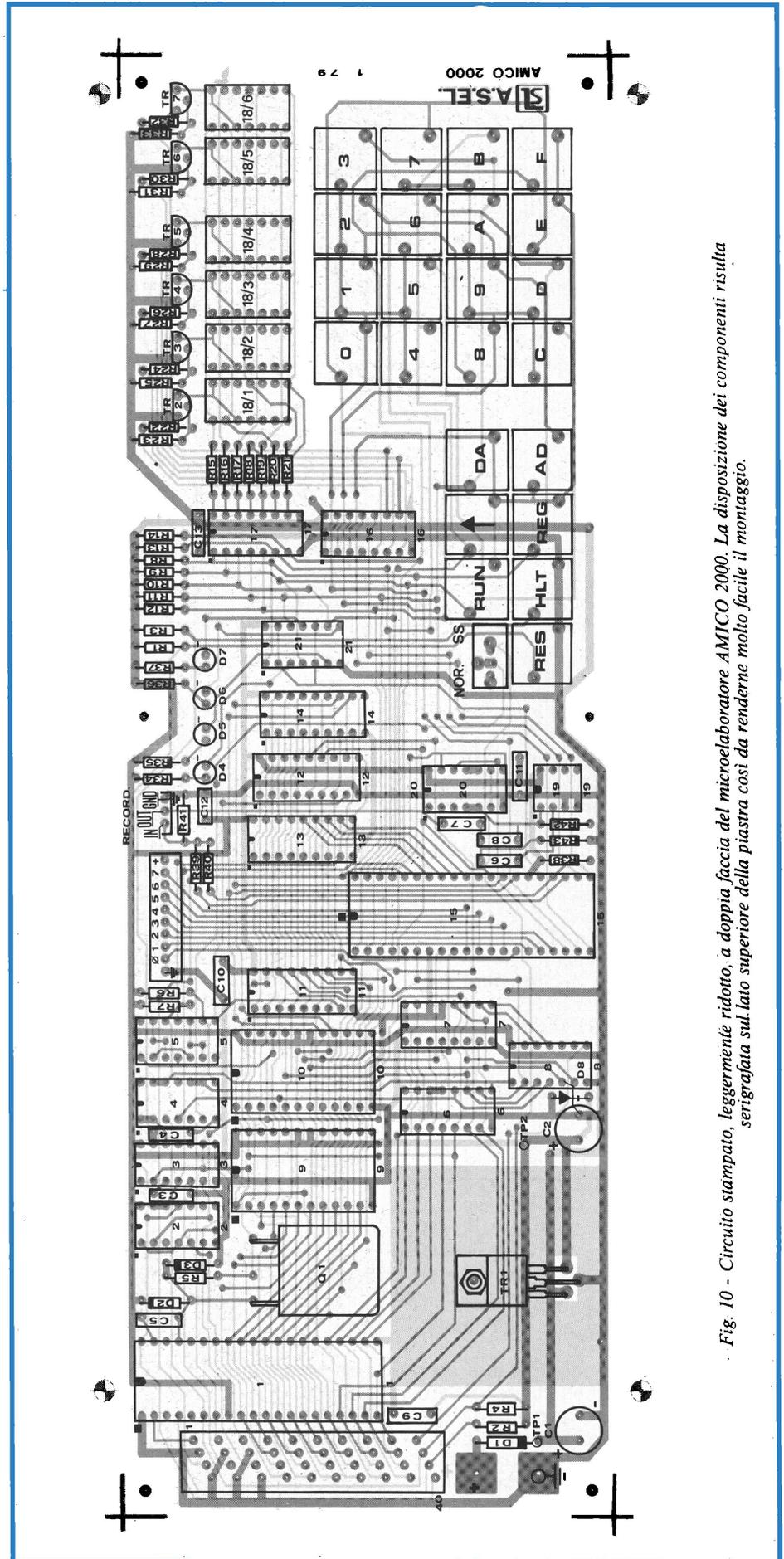


Fig. 10 - Circuito stampato, leggermente ridotto, a doppia faccia del microlaboratore AMICO 2000. La disposizione dei componenti risulta serigrafata sul lato superiore della piastra così da renderne molto facile il montaggio.

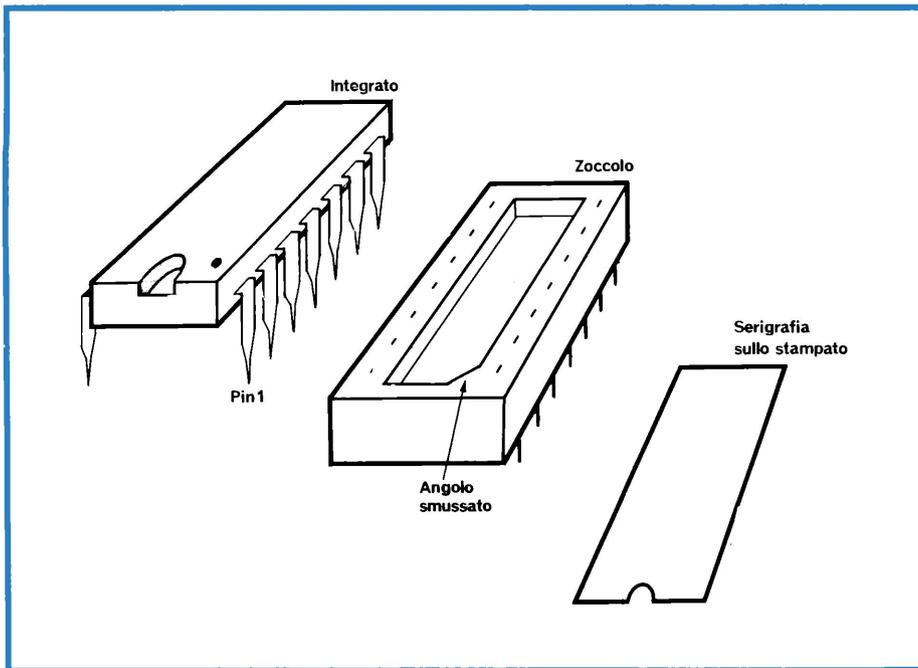


Fig. 11 - È indispensabile, per evitare dannose inversioni, montare gli zoccoli con lo stesso orientamento (ovvero tutti con l'angolo interno smussato verso l'alto) degli integrati da inserire. Il disegno mostra la corrispondenza fra "tacca di riferimento" dell'integrato e "angolo interno smussato" dello zoccolo.

timi condensatori bisogna prestare attenzione alla polarità, che è riportata sull'involucro dello stesso e sulla serigrafia dello stampato.

Concludendo il montaggio provvediamo a saldare anche il TR1, che richiede anche il dissipatore fornito con il kit (vedere Fig. 13).

Montiamo e saldiamo inoltre l'interruttore SS e la contattiera a 10 posti per le uscite digitali, ed anche i due capicorda per l'alimentazione. Da ultimo saldiamo il quarzo Q1 che va assicurato allo stampato con un pezzetto di biadesivo (vedere Fig. 13), che funge anche da isolante tra il corpo metallico del quarzo e le piste sottostanti.

Un primo collaudo

A questo punto mancano ancora gli integrati ed i display, che andranno inseriti negli zoccoli.

Prima di inserire questi delicati componenti è opportuno effettuare una prova preliminare che ci permetta di constatare l'effettiva assenza di cortocircuiti. Per effettuare questo primo collaudo è necessario disporre di un alimentatore, anche non stabilizzato, in grado di erogare una corrente di circa un Ampere ad una tensione compresa tra 8,5 e 12 V,

che andrà collegato ai due capicorda che si trovano sulla piastra. Chi non dispone di questo alimentatore può costruirselo secondo lo schema di Fig. 14, oppure acquistare il kit di montaggio.

Per questo primo collaudo occorre inoltre un comune tester, col quale cominciamo a controllare la tensione esistente tra il punto di prova TP1 e la massa (d'ora in poi faremo riferimento alla fotografia di Fig. 15). Dopo aver pertanto posizionato il puntale positivo rosso in TP1 e il puntale negativo sul punto "massa" misureremo la tensione: se questa è compresa tra 7,5 e 12 V (se la tensione è superiore agli 11 V fate attenzione alla temperatura di TR1 ed eventualmente limitatela aumentando le dimensioni del dissipatore) allora tutto è regolare e possiamo spostare il puntale

positivo del tester sul punto di prova TP2 nel quale dovremmo misurare una tensione compresa tra 4,8 e 5,2 V.

Se questi controlli non hanno dato risultato positivo verificate di non aver messo in cortocircuito con saldature maldestre delle piste o dei piedini. Se in TP2 avete trovato una tensione superiore a 5,2 V, con grande probabilità l'errore è nel montaggio del regolatore TR1. Se invece la tensione è 0, o comunque molto bassa può trattarsi sia del regolatore che di un cortocircuito. Potrebbero anche essere i condensatori elettrolitici montati al contrario, basterà toccarli: se scaldano parecchio dissaldateli e sostituiteli con dei nuovi.

Attenzione: non proseguite sino a che non avete trovato le tensioni corrette!

Completamento del montaggio

Dopo aver naturalmente tolto tensione provvediamo ad inserire nei loro zoccoli gli integrati IC2, IC3, IC4 ed IC5, che hanno 14 piedini, ponendo la massima attenzione all'orientamento (vedere Fig. 10). Controllate che tutti i piedini siano ben inseriti nello zoccolo e che l'integrato sia ben fermo nel supporto (vedere Fig. 16).

Inseriamo successivamente gli integrati IC6, IC16 ed IC17, a 16 piedini, e montiamo subito dopo IC9 a 24 piedini.

I quattro integrati rimanenti sono tutti

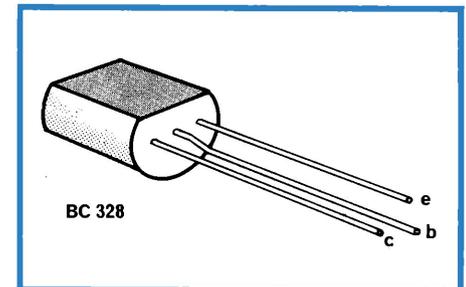
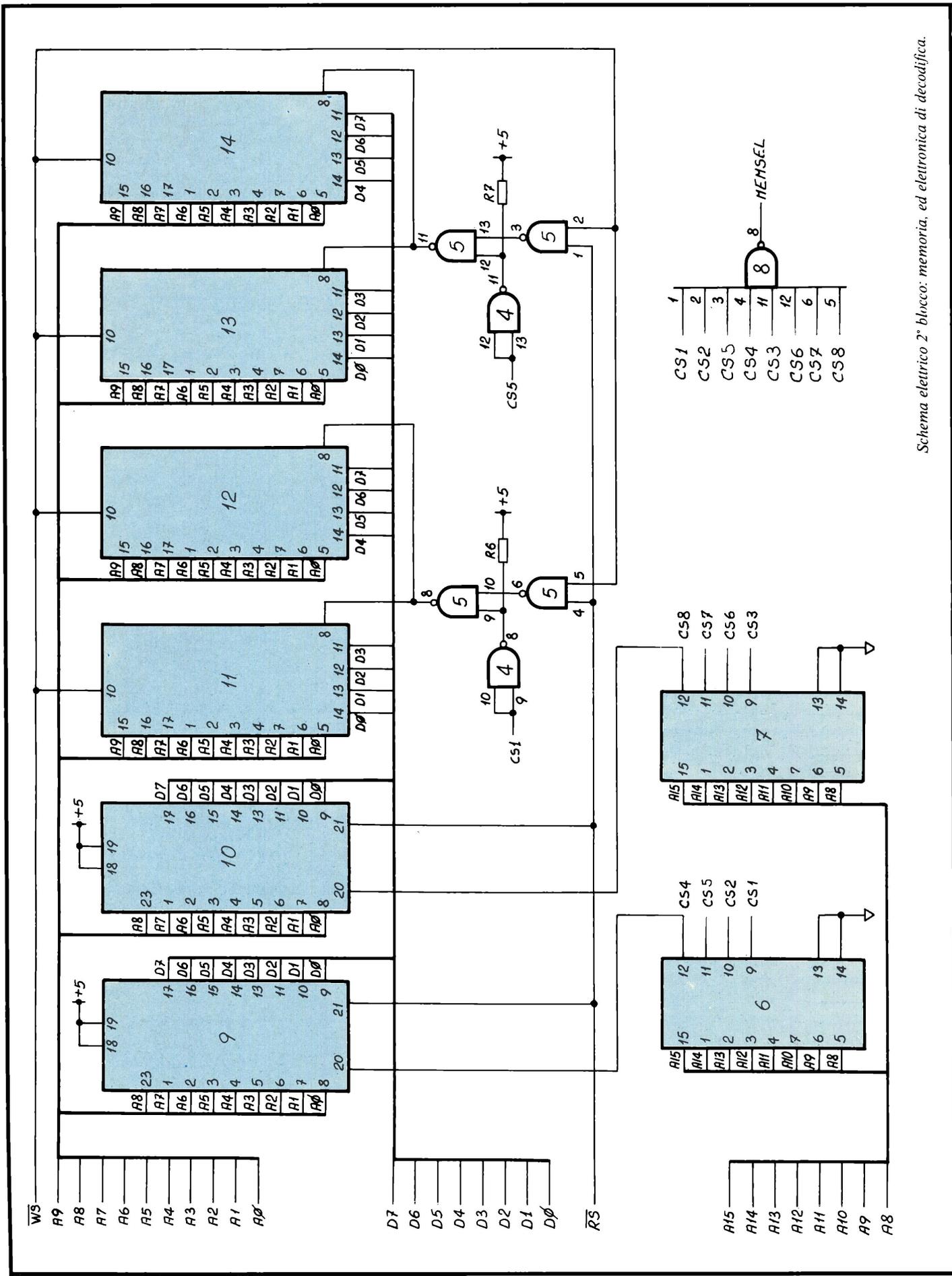


Fig. 12 - Disposizione dei piedini del transistor BC 328.

Tabella 5 - IDENTIFICAZIONE DELLE RESISTENZE CONTENUTE NELLA SCATOLA DI MONTAGGIO DELL'AMICO 2000/A SECONDO IL CODICE A COLORI

Valore	Codice colore			
	1 colore	2 colore	3 colore	4 colore
3,9 k Ω	arancio	bianco	rosso	oro
4,7 k Ω	giallo	viola	rosso	oro
10 k Ω	marrone	nero	arancio	oro
220 k Ω	rosso	rosso	giallo	oro
22 k Ω	rosso	rosso	arancio	oro
150 Ω	marrone	verde	marrone	oro
1,2 k Ω	marrone	rosso	rosso	oro



Schema elettrico 2° blocco: memoria, ed elettronica di decodifica.

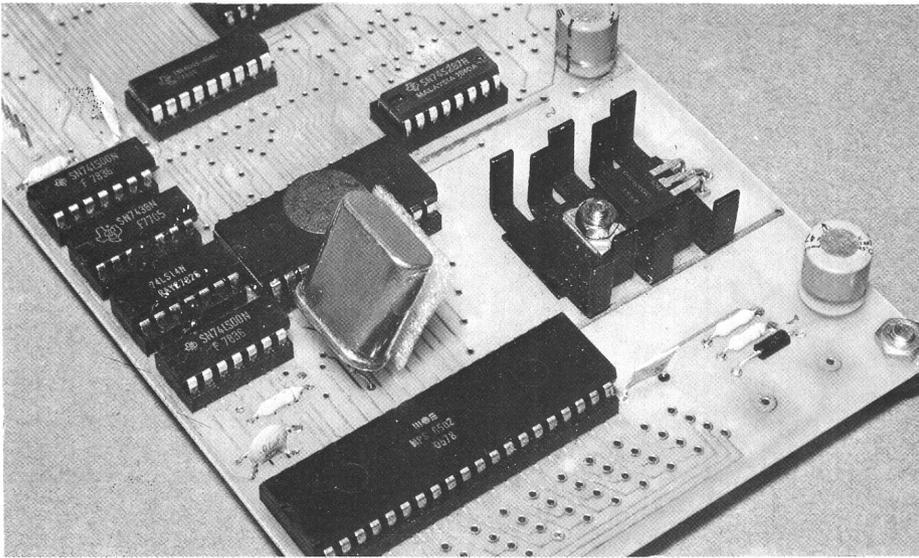


Fig. 13 - Particolare dissipatore del regolatore TR1 e del montaggio del quarzo con il biadesivo isolante.

punto dovrete vedere il display acceso, mostrando delle cifre e/o lettere che restano immutate se non si toccano i tasti. Le prime 4 cifre da sinistra (corrispondenti al display indirizzi) devono essere 0000, mentre le altre due devono essere due qualsiasi caratteri esadecimali. Per aiutarvi a riconoscere i caratteri del display fate riferimento alla tabella 8.

Se il display rimane spento o se le cifre continuano a cambiare significa che qualcosa non è a posto; conviene togliere l'alimentazione e controllare metodicamente l'orientamento degli integrati, dei transistori e dei display; ricontrollate anche le saldature, cercando eventuali contatti, fonti di corto circuiti.

A questo punto, dopo aver corretto gli eventuali errori, ripetete il ciclo di collaudo finale.

Se ancora una volta non ottenete i risultati previsti vuol dire che avete fatto qualche errore e vi consigliamo di rimettere tutto nella scatola e di spedirlo al servizio assistenza tecnica della A.S.E.L.

A questo punto, se tutto è a posto, premiamo uno per volta i tasti esadecimali (tastiera rossa). La cifra corrispondente al tasto premuto deve entrare nella posizione più a destra del display indirizzi, cioè quello a 4 cifre. Durante questa operazione il display dati cambia in maniera casuale.

Controlliamo ora l'ingresso della ta-

stiera esadecimale (rossa) nel display dati; premiamo 0000 (apparirà 0000 nel display indirizzi), e premiamo il tasto

DA. Ripetiamo ora l'operazione già svolta precedentemente, premendo uno alla volta i tasti rossi da 0 ad F. La cifra corrispondente entrerà nella posizione più a destra del display dati. Premiamo ora il tasto **↑**; ad ogni pressione il display indirizzi si incrementerà di uno (ricordati sempre che in esadecimale $09+1 = 0A$ e che $0F+1 = 10!$). Ancora una volta durante questa operazione il display dati varierà in maniera casuale. Se teniamo premuto il tasto **↑** il display indirizzi si incrementerà di uno una volta al secondo. Consideriamo ora

il tasto **REG**, del quale esamineremo

in seguito la specifica funzione operativa.

Premiamo il tasto **AD** e poi i tasti 00 F6, quindi il tasto **DA** e 9E; successivamente pigiamo **↑** (comparirà nel display indirizzi 00 F7) ed i tasti 01. A questo punto si preme il tasto **REG**. Nel display indirizzi deve apparire 019E (cioè i dati appena introdotti) mentre il display dati conterrà un numero casuale.

Effettuiamo ora la prova del tasto

HLT, premendo il quale il display si fermerà mettendo in evidenza una cifra a caso illuminata in modo piuttosto intenso (potrà capitare anche che il display rimanga spento, ma non preoccupatevi, premete di nuovo **RES** (reset e poi ancora **HLT**).

Effettuiamo ora la prova del tasto

HLT, premendo il quale il display

si fermerà mettendo in evidenza una cifra a caso illuminata in modo piuttosto intenso (potrà capitare anche che il display rimanga spento, ma non preoccupatevi, premete di nuovo **RES** (reset e poi ancora **HLT**).

Effettuiamo ora la prova del tasto

In questo modo abbiamo collaudato

tutta la tastiera, tranne il tasto **RUN**,

che proveremo facendo eseguire il nostro primo piccolo programma, in modo da verificare funzionamento globale nel nostro sistema.

Introduciamo un primo programma

Riprendendo quanto avevamo esposto nel secondo capitolo passiamo ad effettuare la somma di due numeri esadecimali. Per comprendere bene la funzione dei tasti che useremo facciamo riferimento alla tabella 6.

La tabella 7 mostra la sequenza di tasti che bisogna premere ordinatamente per

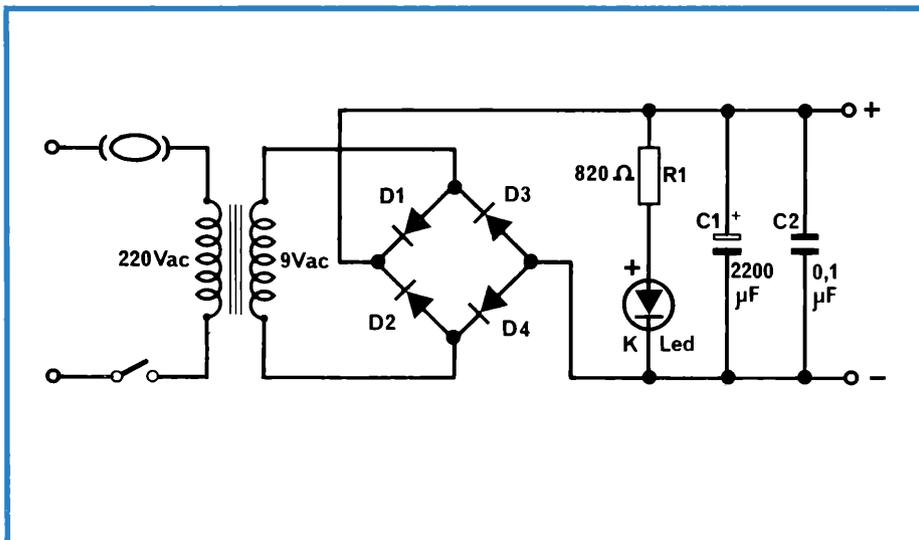
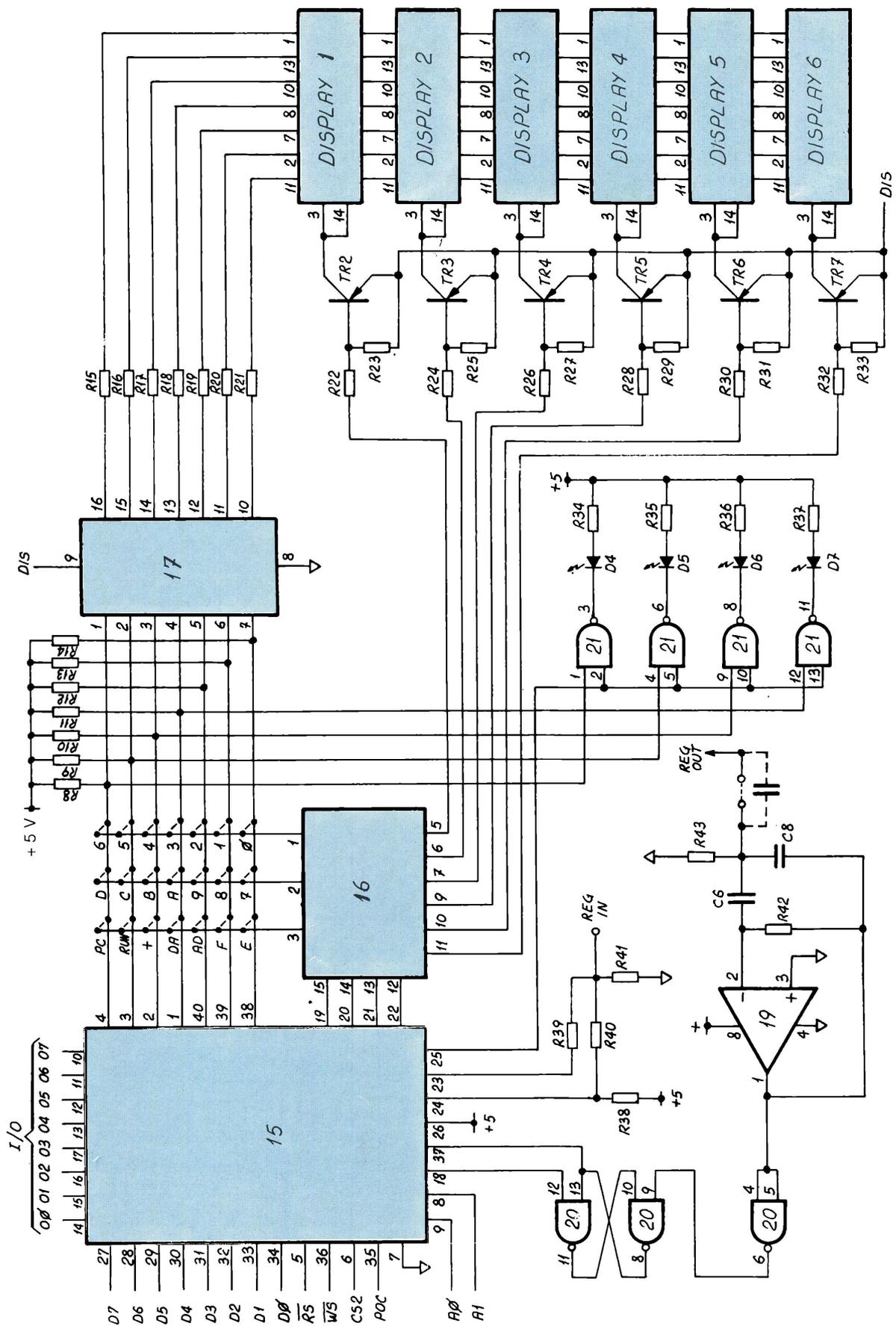


Fig. 14 - Schema dell'alimentatore da 1 A per il microcomputer AMICO 2000A. Questo alimentatore è anche disponibile in scatola di montaggio.



Schema elettrico 3° blocco display, tastiera, interfaccia cassetta e parte I/O.

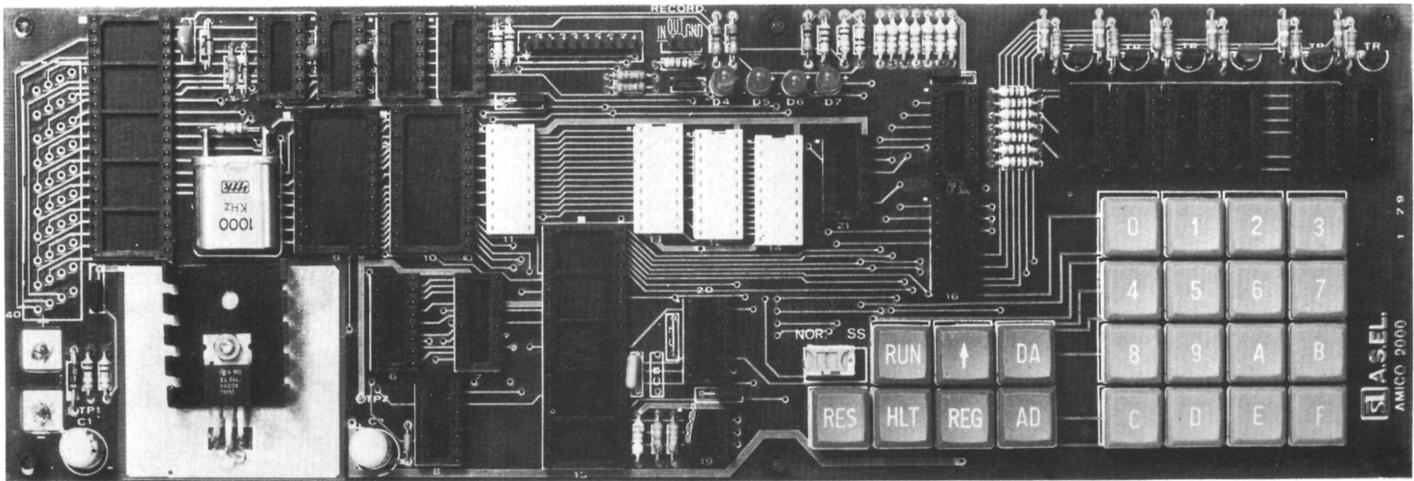


Fig. 15 - Piastra montata senza integrati inseriti. Controllare che sul punto di prova tp1 sia presente una tensione compresa fra 7,5 e 12 V e su tp2 una tensione compresa fra 4,8 e 5,2 V.

caricare il programma in memoria, e mette anche in evidenza il corrispondente movimento dei dati sul display.

Si noti che xx sta per numeri o lettere qualsiasi. Si veda inoltre la tabella 8 che mostra come vengono rappresentate cifre e lettere nel visualizzatore del micro-computer.

L'esecuzione del programma

Il programma che abbiamo appena finito di introdurre in memoria ha la funzione di sommare i due numeri presenti nelle locazioni di memoria 0006 e 0007 mettendo il risultato di questa operazione nella locazione di memoria 0004.

Prima di eseguirlo dovremo pertanto introdurre nelle locazioni 0006 e 0007 i due dati che vogliamo sommare. Per

fare ciò premiamo **AD** 0006 e **DA**

03; in tal modo avremo introdotto nella locazione 0006 il dato 03 (cioè il numero 3 in esadecimale, che corrisponde al 3 decimale). Successivamente premiamo

↑ e 02, con l'effetto di introdurre il

numero esadecimale 02, (corrispondente a 2 in decimale) nella locazione di memoria 0007 (infatti la funzione del tasto

↑ è, come abbiamo già detto, quella

di incrementare di uno il contenuto del display indirizzi, che sarà passato pertanto da 0006 a 0007).

Dopo aver inserito questi dati l'operazione che effettivamente ci accingiamo ad eseguire è 2+3. Probabilmente qualcuno di voi sarà deluso, essendo evidente

che non occorre un microelaboratore per effettuare una simile operazione, e che comunque l'intera procedura è molto complessa, ma non disperate; siamo appena agli inizi e questo programmino banale ha una sua precisa funzione didattica e di collaudo!

Per eseguire il programma non ci resta altro che indicare al nostro Amico 2000 l'indirizzo nel quale inizia il programma stesso e quindi dare il via all'operazione. Basterà pertanto premere **AD** 000A (sul display dati troveremo 18, che è proprio la prima istruzione del nostro program-

ma), e quindi premere il tasto **RUN**, che farà eseguire il programma a partire dalla istruzione 000A.

L'esecuzione del programma richiede solo una piccolissima frazione di secondo (un microelaboratore come l'AMICO 2000A è infatti in grado di eseguire mediamente 200 mila istruzioni al secondo) e apparentemente per voi non sarà cambiato nulla, poiché il display non è cambiato.

L'operazione di somma è però stata eseguita ed infatti il risultato potrà essere letto nella posizione di memoria

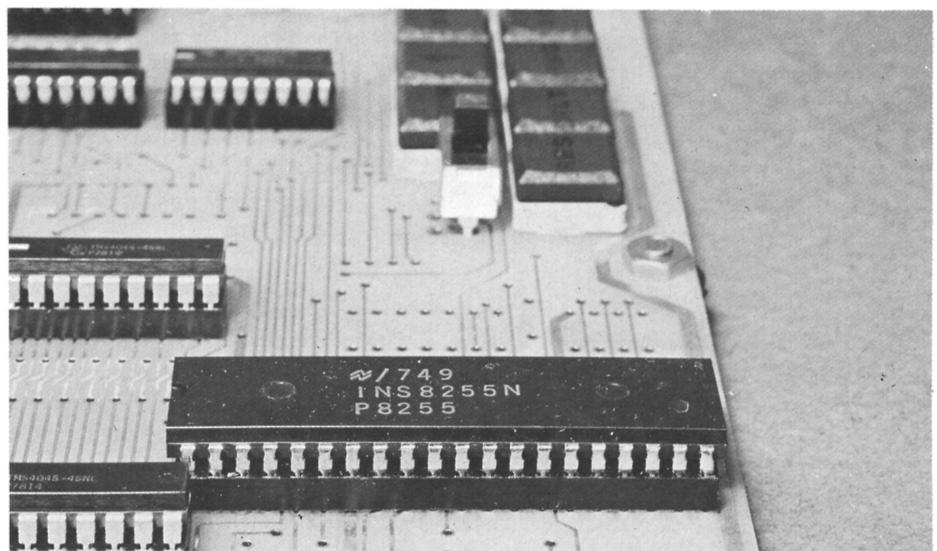


Fig. 16 - Integrato inserito nello zoccolo. Verificare che tutti i piedini siano ben inseriti

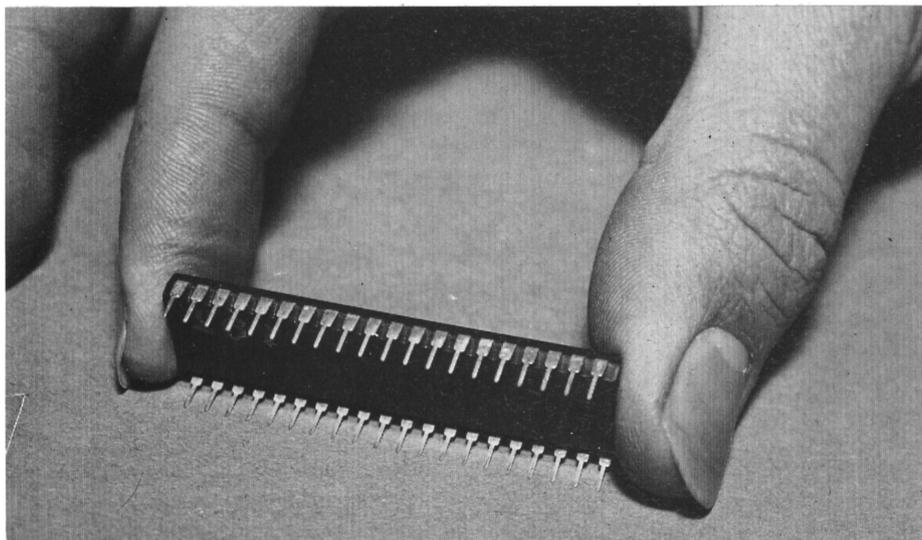


Fig. 17 - Particolare della piegatura dei piedini dei circuiti integrati che permette un perfetto inserimento degli stessi negli appositi zoccoli.

Tabella 6 - DESCRIZIONE DEI TASTI FUNZIONALI DELL'AMICO 2000

Tasto	Definizione
AD	AD = Address = Indirizzo Permette di selezionare l'indirizzo della locazione di memoria che si intende esaminare o modificare (si può modificare solo se si tratta di memoria RAM). Per introdurre l'indirizzo vengono utilizzati i tasti esadecimali (quelli rossi).
DA	DA = Dato Permette di modificare il contenuto di una locazione di memoria precedentemente selezionata. Per introdurre il dato si utilizza sempre la tastiera esadecimale. Attenzione: non si possono modificare le locazioni di memoria non coperte dalla RAM presente sul sistema. Nel caso del sistema minimo la RAM si trova compresa tra le locazioni 0000 e 03FF (1024ma locazione).
↑	Incremento indirizzo Questo tasto permette di esaminare la locazione successiva a quella sulla quale siamo posizionati. Nota bene: se l'ultimo tasto che abbiamo premuto prima del ↑ è AD , i tasti esadecimali premuti dopo vengono introdotti nel display indirizzi. Se invece l'ultimo tasto premuto prima di ↑ è stato DA , i valori esadecimali introdotti saranno relativi al dato, ovvero al contenuto della locazione di memoria aperta. Il tasto ↑ non modifica la funzione precedentemente selezionata.
REG	REG = Registro Program Counter Maggiori dettagli su questa funzione verranno spiegati nel corso della trattazione.
RUN	RUN = Via Permette di iniziare l'esecuzione del nostro programma a partire dalla locazione di memoria puntata dal display indirizzi.
HLT	HLT = Halt = Arresto Genera una interruzione a livello CPU.
RES	RES = Reset = Azzerramento Permette l'inizializzazione del sistema all'accensione, visualizza la locazione di memoria 0000, permette di arrestare l'esecuzione di un programma utente in qualsiasi momento passando il controllo del sistema al monitor.

0004. Per vedere il contenuto di questa posizione di memoria premiamo **AD**

e poi 0004; nel display dati apparirà 05, che è appunto l'atteso risultato della somma 03+02.

Con questo programma possiamo dunque eseguire delle somme di due numeri di 8 bits (valore massimo in decimale = 255). Per somme di numeri maggiori di 255 occorre scrivere un programma un po' più complesso. Questi programmi sono ad esempio già presenti in partenza nelle ROM delle calcolatrici tascabili, pertanto essi non richiedono l'operazione di inserimento manuale dei programmi stessi, che invece dobbiamo effettuare con l'AMICO 2000A. Questo non è in realtà un grande svantaggio perchè aggiungendo qualche integrato come vedremo nel quinto capitolo sarete in grado di registrare i programmi su cassette magnetiche e di richiamarli quando vorrete, saltando la fase di introduzione manuale. È comunque importante che vi rendiate conto della profonda differenza tra un elaboratore come questo ed una calcolatrice normale, differenza che è da un lato nella molto maggiore velocità di esecuzione (le calcolatrici tascabili programmabili sono notevolmente più lente), ma che consiste soprattutto nella flessibilità. Il microelaboratore AMICO 2000A infatti non solo potrà eseguire operazioni aritmetiche, ma sarà anche in grado di controllare tutta una serie di strumenti, cosa che certo non può fare una calcolatrice.

Vi ricordiamo peraltro che le applicazioni tipiche dell'AMICO 2000A, come dei microelaboratori in genere, non sono tanto orientate verso la sostituzione delle tradizionali calcolatrici, ormai diffusissime sul mercato, quanto piuttosto verso applicazioni più flessibili ed evolute.

Un programma più complesso: il gioco dei riflessi

Tutto ciò che abbiamo fatto fino ad ora ci ha permesso di prendere confidenza con i comandi dell'AMICO 2000A e soprattutto di comprenderne il funzionamento. Per trarre il massimo beneficio da ciò che faremo in futuro è indispensabile aver bene compreso tutto quello che abbiamo detto fino ad ora: questo vi permetterà piano piano di essere sempre più indipendenti per ciò che riguarda la creazione di programmi originali, che sono poi il "carburante" del nostro sistema.

Non vogliamo però togliervi il gusto di cominciare a giocare con il vostro microcalcolatore; abbiamo preparato perciò un semplice programma composto da 55 byte per trasformare il vostro AMICO 2000 in una macchina per la prova

Tabella 7 - PROGRAMMA PER ESEGUIRE UNA OPERAZIONE DI SOMMA CON IL MICROELABORATORE

Operaz. n.	Tasto da premere	Visualizzatore	Commento
1	RES	0000 XX	Azzerramento iniziale.
2	AD	0000 XX	AD = Address = indirizzo. L'elaboratore si prepara a ricevere un indirizzo di memoria.
3	0 0 0 A	000A XX	Indirizzo di partenza, del programma.
4	DA	000A XX	DA = Dato; l'elaboratore si prepara a ricevere un dato da depositare nella locazione di memoria 000A.
5	1 8	000A 18	Il numero 18, che è il codice esadecimale della operazione CLC (Clear Carry), cioè azzerramento del riporto è entrato nella locazione di memoria 000A.
6	↑	000B XX	L'elaboratore è pronto a ricevere un altro dato nella posizione di memoria successiva alla 000A.
7	A 5	000B A5	Il numero A5, che è il codice esadecimale della istruzione LDA (Load Accumulator) è entrato nella locazione di memoria 000B.
8	↑ 0 6	000C 06	06 è l'indirizzo di memoria del 1° addendo.
9	↑ 6 5	000D 65	Il numero 65 è il codice operativo dell'istruzione ADC.
10	↑ 0 7	000E 07	07 è l'indirizzo di memoria del secondo addendo.
11	↑ 8 5	000F 85	Il numero 85 è il codice operativo dell'istruzione STA.
12	↑ 0 4	0010 04	04 è la locazione di memoria in cui viene depositato il risultato della somma.
13	↑ 4 C	0011 4C	L'istruzione 4C corrisponde a JMP = salto (1).
14	↑ 2 2	0012 22	Questo salto serve a chiudere il programma ed a passare il controllo delle operazioni al Monitor, cioè al programma di gestione interna del microcalcolatore. L'indirizzo al quale inizia questo programma interno, che risiede in PROM, è appunto FE22.
15	↑ F E	0013 FE	

Nota 1: quando il programma arriva a questo punto, cioè quando trova una istruzione di JMP (codice 4C), legge il contenuto delle due locazioni di memoria successive al 4C e lo utilizza come indirizzo da cui preleva la prossima istruzione da eseguire. In questo caso riprenderà l'esecuzione all'istruzione contenuta in FE22.

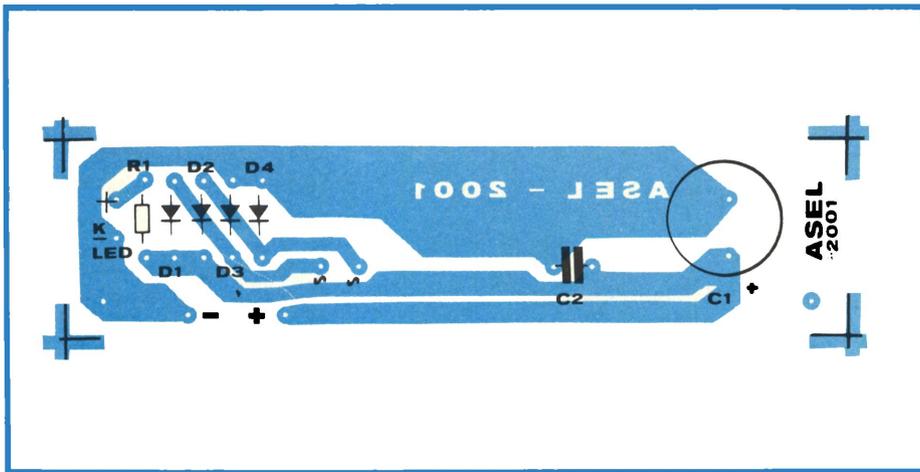


Fig. 18 - Serigrafia e traccia del circuito stampato dell'alimentatore da 1 A.

dei riflessi. Questo semplice programma può così cominciare a far parte della vostra biblioteca e al momento opportuno potrete anche registrarlo su cassetta magnetica per introdurlo automaticamente nella memoria RAM dell'elaboratore.

Per questa volta non preoccupatevi di capire ciò che state introducendo, ovvero il significato delle varie istruzioni ma cercate di comprendere la funzione e il perché dei tasti usati.

Come introdurre il programma: accendiamo la macchina, premiamo il tasto

[RES] poi [AD] e introduciamo l'indirizzo della locazione di RAM dalla quale partirà il nostro programma ovvero 0200,

quindi premiamo [DA], A5, poi [↑], F9,

ancor [↑] 2A e così via premendo sempre il tasto [↑] prima di introdurre i dati

fino all'ultimo che si troverà nella locazione di memoria 0236.

Ora torniamo alla locazione di memoria 0200 premendo [AD] 0200, quindi premiamo successivamente [↑] tante vol-

te quanti sono i dati introdotti verificando la corrispondenza fra ciò che appare sul display e la lista del programma.

Nella tabella 9 è riportata la lista del programma: sulla sinistra le prime quattro cifre indicano l'indirizzo che deve essere presente sui quattro digit del display indirizzi e sono riportate ogni

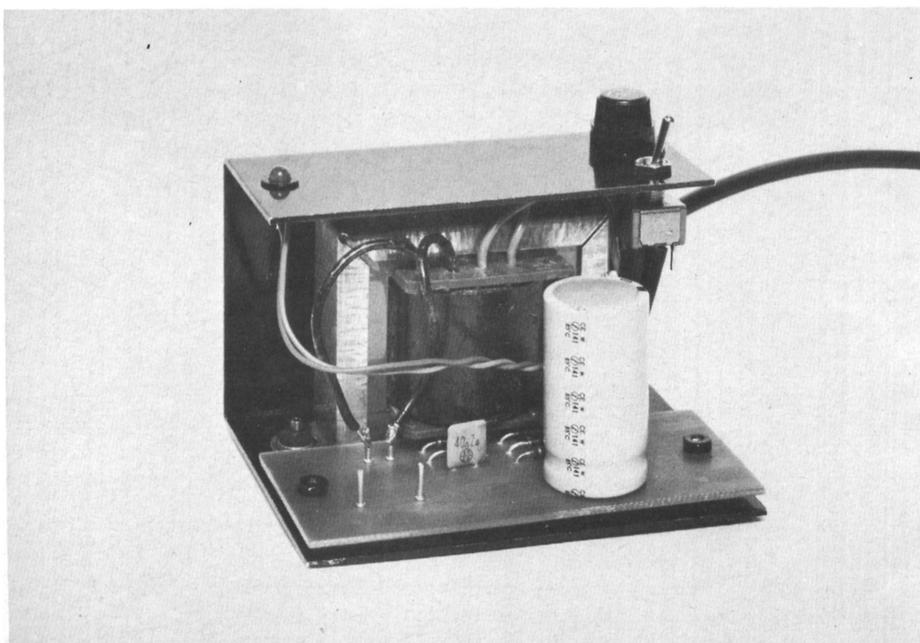


Fig. 19 - L'alimentatore dell'AMICO 2000/A montato.

Tabella 8 - Corrispondenza fra caratteri a sette segmenti (quelli dei display) e numeri esadecimali.

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

tanto per verificare la corrispondenza con il display dell'AMICO 2000; le due cifre a destra rappresentano il contenuto (o ciò che dobbiamo introdurre) della corrispondente locazione di memoria selezionata e sono in definitiva le istruzioni e i dati che immettiamo nell'elaboratore in codice esadecimale.

Dopo aver verificato che tutti i dati siano stati introdotti correttamente possiamo far "girare" il programma. Per far questo riportiamoci come abbiamo fatto prima all'indirizzo 0200 quindi premiamo il tasto **RUN**: il display si spegnerà per riaccendersi dopo qualche secondo. Appena esso si riaccende bisogna premere uno qualsiasi dei tasti rossi e sul display apparirà un numero proporzionale al tempo che è intercorso tra l'accensione del display e la pressione del tasto. Chi ha i riflessi più pronti visualizzerà numeri più bassi.

Ogni volta che si vuol far ripartire il programma basta premere RUN. Ricordatevi però che il programma rimane registrato nella RAM solo quando il microelaboratore è acceso; se, dopo aver programmato e giocato, spegnete la macchina il contenuto della RAM verrà perso così che riaccendendola dovrete reinserire daccapo l'intero programma.

A titolo di esempio, sempre riferendo-

ELENCO COMPONENTI DELL'AMICO 2000/A

Resistori (tutti da 1/4 W, tolleranza 5%)

R1-2-3-4	: 10 kΩ
R5	: 220 kΩ
R6-7	: 10 kΩ
R8-9-10-11-	
12-13-14	: 4,7 kΩ
R15-16-17-	
18-19-20-21	: 100 Ω
R22-24-26-	
28-30-32	: 1,2 kΩ
R23-25-27-	
29-31-33	: 3,9 kΩ
R34-35-	
36-37 ⁽¹⁾	: 82 Ω
R38 ⁽¹⁾	: 22 kΩ
R39 ⁽¹⁾	: 1,8 kΩ
R40 ⁽¹⁾	: 1,8 kΩ
R41 ⁽¹⁾	: 100 Ω
R42 ⁽¹⁾	: 33 kΩ
R43 ⁽¹⁾	: 3,3 kΩ

Condensatori

C1	: 47 μF - 16 V - elettrolitico
C2	: 47 μF - 16 V - elettrolitico
C3-4	: 1 μF - elettrolitico
C5	: 15 pF - ceramico a disco
C6-7 ⁽¹⁾	: 6,8 nF - polistirolo
C8 ⁽¹⁾	: 1 nF - polistirolo (opzionale)
C9-10-11 ⁽¹⁾ -	
12 ⁽¹⁾ -13	: 0,1 μF - ceramico a disco (oppure 0,047 μF)

Diodi

D1	: 1N4001
D2-3	: 1N4148
D4-5-6-7 ⁽¹⁾	: diodi LED
D8	: diodo zener da 6,2 V 1/2 W

Transistori

TR1	: LM340T5 (opp. μA 7805)
TR2-3-4	
5-6-7	: BC 327

Integrati

IC1	: 6502 - microprocessore CPU
IC2	: 74LS00 - quadruplo NAND a 2 ingressi
IC3	: 74LS14 - HEX Schmitt Trigger
IC4	: 74LS38 (opp. 74LS03) - quadruplo NAND a 2 ingressi a collettore aperto
IC5	: 74LS00 - quadruplo NAND a 2 ingressi
IC6	: 74S287 - PROM di decodifica (oppure 93427)
IC7 ⁽¹⁾	: 74S287 - PROM di decodifica (oppure 93427)
IC8 ⁽¹⁾	: 74LS30 - NAND a 8 ingressi
IC9	: 93448 - PROM programma MONITOR (oppure 74 S 474)
IC10 ⁽¹⁾	: 93448 - PROM gestione interfaccia cassette (oppure 74 S 474)
IC11-12	: TMS4045 (opp. 2114) RAM 1K x 4 statica
IC13-14 ⁽²⁾	: TMS4045 (opp. 2114) RAM 1K x 4 statica
IC15	: 8255 - tripla porta I/O a 8 bit
IC16	: 74LS145 - decodifica display
IC17	: ULN2003 (opp. MC 1413) - driver display LED
IC18-1-2-3-	
4-5-6	: TIL312 - display LED a 7 segmenti
IC19 ⁽¹⁾	: LM358 - convertitore di ingresso per interfaccia - cassetta magnetica
IC20 ⁽¹⁾	: 74LS132 - quadruplo NAND a 2 ingressi Schmitt Trigger
IC21 ⁽¹⁾	: 74LS38 - quadruplo NAND a due ingressi
SS	: interruttore unipolare
Q1	: quarzo da 1 MHz
TASTI	: 23 pezzi

NOTE:

(1) Questo componente viene fornito per completare il microcomputer AMICO 2000A con l'interfaccia per il registratore a cassette.

(2) Questi componenti vengono forniti per completare la memoria RAM dell'AMICO 2000/A.

Tabella 9 - Programma per il gioco dei riflessi

INDIRIZZI	DATI	INDIRIZZI	DATI
0200	A5		38
	F9		B5
	2A		FC
	65		69
	F9		00
	29	0220	95
	7F		FC
	85		E8
	FB		D0
	20		F7
	EB		D8
	FE		20
	D0		0C
	FB		FF
	E6		FD
	FA		ED
0210	D0		20
	F7		0C
	E6		FF
	FB		20
	D0		57
	F3	0230	FF
	85		C9
	F9		13
	A2		D0
	FD		F6
	F8		F0
		0236	C9

IL SISTEMA DI INDIRIZZAMENTO

La somma nel sistema binario

Abbiamo già visto nel capitolo precedente una operazione elementare, la somma binaria di due numeri. Le stesse regole che ci permettono di eseguire una somma nel nostro solito sistema decimale, sono usate nel sistema binario. Inoltre nel sistema binario, dato che vi sono due sole cifre (lo 0 e l'1) queste regole sono ancora più semplici.

Vediamole insieme e introduciamo il concetto di "Carry":

$0 + 0 = 0$	Riporto (Carry) = 0
$0 + 1 = 1$	Carry = 0
$1 + 0 = 1$	C = 0
$1 + 1 = 0$	C = 1

Assunte queste regole fondamentali supponiamo ora di dover eseguire la somma: $1 + 1 + 1$.

Il risultato di questa operazione è 1 con il $C = 1$. Perché?

Scomponendo abbiamo: $(1 + 1) + 1 = 0 + 1$ con $C = 1$ (per via della somma $1 + 1$); ora $0 + 1 = 1$. Il risultato della intera operazione quindi è 1 e il C rimane uguale a 1.

Utilizziamo queste regole per effettuare la somma:

$$0A_{16} + 07_{16} = 11_{16}$$

(N.B. - Si tratta di cifre esadecimali).

Trasformandole in binarie abbiamo:

$$\begin{array}{r} 0000 \quad 1010 \quad (0A) \\ 0000 \quad 0111 \quad (07) \end{array}$$

$$\hline 0001 \quad 0001 \quad (11)$$

Partendo dalla cifra a destra si ha:



Notiamo che la somma di ogni cifra viene fatta tenendo conto anche del Carry. Se ora ritorniamo al nostro elaboratore ci chiediamo: dove sta fisicamente il Carry?

Per rispondere dobbiamo introdurre un nuovo registro presente nella CPU. Fino ad ora abbiamo incontrato l'ACCUMULATORE, il PROGRAM COUNTER e l'UNITÀ ARITMETICOLOGICA (ALU).

Il REGISTRO DI STATO (Status in inglese) è il nuovo registro che contiene alcune informazioni sul progredire delle operazioni che il microprocessore sta eseguendo. Lo Status è formato da 8 bit di cui il primo è proprio il CARRY. Al momento attuale quindi la nostra CPU può essere rappresentata come in figura 20.

Vogliamo puntualizzare che: 1) Il Program Counter è un registro da 16 bit;

2) Dello Status Register (P) abbiamo definito solo il primo bit che è il Carry; gli altri bit dello Status verranno analizzati in seguito.

Il microprocessore 6502 (la CPU del nostro AMICO 2000/A) possiede alcune istruzioni che agiscono sul Carry. Le principali sono:

SEC; cioè SET CARRY FLAG. Questa istruzione mette a 1 il bit di Carry (Set in inglese). La sua traduzione in linguaggio macchina è **38**.

CLC; cioè CLEAR CARRY FLAG. Questa istruzione mette a 0 il bit di Carry (Clear in inglese). La sua traduzione in linguaggio macchina è **18**.

Riprendendo l'esercizio interrotto possiamo dire che il Carry all'ingresso della somma può essere da noi condizionato; lo possiamo infatti porre a 0 o a 1 a piacimento tramite una delle due istruzioni citate.

```

  1 0 0 1 1 1 1 1
  1 1 0 1 0 0 1 0
+
  1 0 1 1 1 0 0 0 1
  ↑ Carry

```

Cioè $9F + D2 = 71$ con il riporto di 1 (Carry di uscita = 1).

Gli esempi fino ad ora fatti sono stati sviluppati nel sistema Esadecimale. Le stesse cose però valgono anche nel nostro sistema Decimale che usiamo tutti i giorni.

Infatti se eseguiamo l'operazione:

```

8310 + 4110 = 1 2410
sistema  ↑   ↑ riporto
decimale

```

troviamo ancora il riporto, o carry, esattamente come abbiamo appena visto.

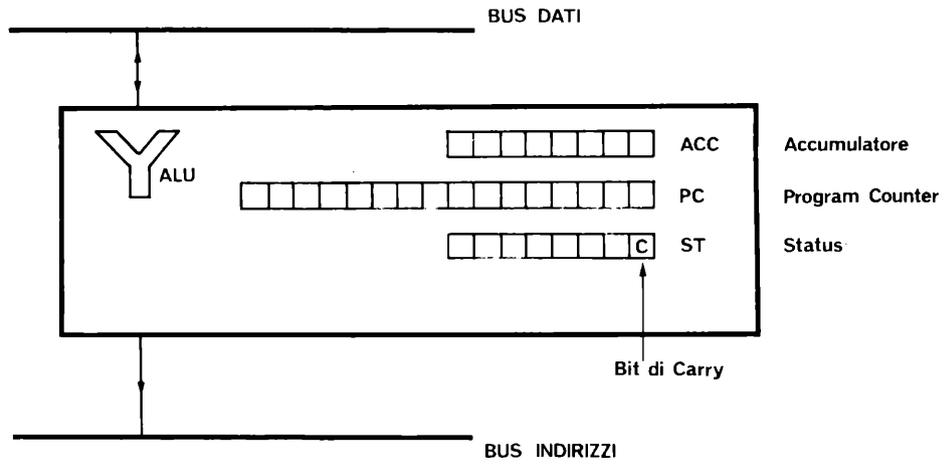


Fig. 20 - Alcuni particolari all'interno della CPU: Accumulatore, Program Counter e Status Register.

Esercizio con l'AMICO 2000A

Cerchiamo ora di mettere in pratica le varie nozioni che abbiamo appena appreso.

Il calcolo da fare è ancora il solito: la Somma, però una volta la faremo con il Carry di ingresso a 1, la volta successiva con il Carry di ingresso a 0.

Accendiamo la macchina e partiamo a scrivere il nostro programma per esempio dalla locazione di memoria 0201.

Il Carry all'uscita dell'operazione dipende dal risultato della operazione medesima.

Infatti nell'esercizio appena visto il (display)

Tasti	Indirizzi	Dati	Istruzione	Commenti
AD	0201	xx		
DA	0201	18	CLC	Azzerramento del Carry
↑	0202	A5	LDA 06	Carico l'addendo
↑	0203	06		
↑	0204	65	ADC 07	Sommo all'accumulatore il contenuto della locazione 07 e il Carry.
↑	0205	07		
↑	0206	85	STA 00	Metto il risultato nella locazione 0000.
↑	0207	00		
↑	0208	4C	JMPMONITOR	Istruzione di fine programma. Torno al programma di Monitor.
↑	0209	22		
↑	020A	FE		

Ora che avete inserito il programma possiamo immettere i dati da elaborare.

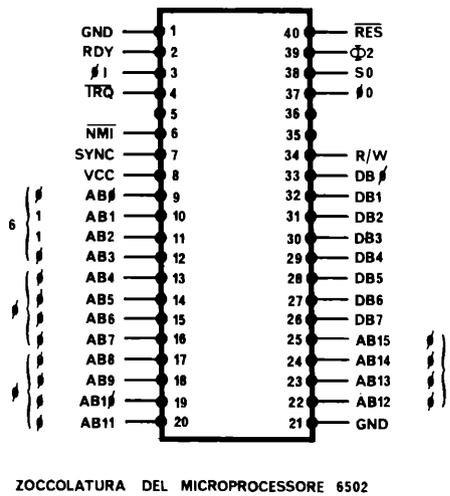


Fig. 21 - Piedini della CPU e loro funzioni.

Carry di uscita era uguale a zero, ma se avessimo fatto: $9F + D2$ avremmo ottenuto:

Tasti	Indirizzi	Dati
AD	0006	
DA	0006	(primo dato, p.e. 02)
↑	0007	(secondo dato, p.e. 03)

Ora carichiamo il PC di partenza del programma. Diciamo cioè al calcolatore da che punto deve partire l'esecuzione dello stesso.

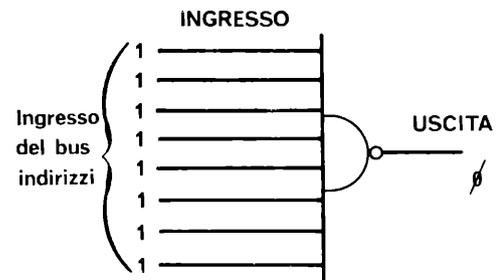
Allora premiamo AD 0201 poi RUN sul display comparirà il risultato 0000 05.

Il risultato è 5 perchè il Carry di ingresso è stato posto a 0 (prima istruzione di CLC).

Proviamo ora a porlo = 1. Per farlo sostituiamo CLC (18) con SEC (38).

Quindi sempre con la stessa procedura:

AD	0201	18
DA	0201	38
RUN		



L'uscita viene a 0 solo quando tutti gli ingressi sono a 1 in quanto la funzione logica impiegata è del tipo NAND.

Fig. 22 - Decodifica per l'indirizzo FF.

Sul display apparirà come risultato 06. Il Carry in ingresso è stato posto da noi a 1.

Provate ora voi stessi a cambiare i dati e ripetere più volte l'esercizio.

Per riassumere abbiamo:

02 +
03 +
Carry

= 05 se il Carry di ingresso = 0
= 06 se il Carry di ingresso = 1

Attenzione non spegnete a questo punto l'elaboratore perchè fra poco aggiungeremo alcune istruzioni al programma.

Somma esadecimale e somma decimale

Per ciò che riguarda la somma dobbiamo ancora spiegare la differenza fra somma esadecimale e somma decimale.

Abbiamo già visto nella parte prima che è:

$$10_{10} = 0A_{16}$$

Ora è evidente che se si esegue la operazione

$$05 + 05$$

il risultato sarà 10 se espresso in forma decimale e 0A se espresso in forma esadecimale.

Ma attenzione! La matematica non è una opinione.

$$05_{10} + 05_{10} = 10_{10} \text{ il che equivale a } 0A_{16}$$

$$05_{16} + 05_{16} = 0A_{16} \text{ il che equivale a } 10_{10}$$

Il problema è solo quello di sapere in che base si sta operando.

Il nostro AMICO 2000 può lavorare in entrambe le basi. Per farlo ha due istruzioni dedicate:

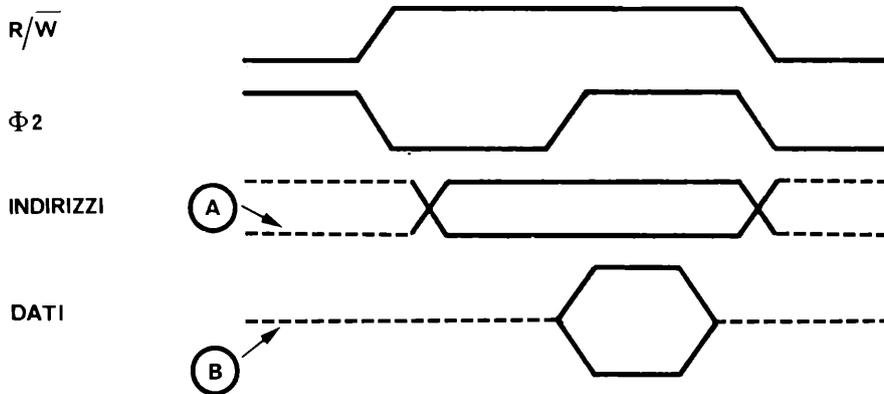
1) **SED** Set Decimal Mode. Codice operativo **F8**. Tutte le somme fatte dopo questa istruzione vengono eseguite in *decimale*.

2) **CLD** Clear Decimal Mode. Codice operativo **D8**. Tutte le somme fatte dopo questa istruzione vengono eseguite in *esadecimale*.

Aggiungiamo ora tornando al nostro computer queste ultime istruzioni al programma precedentemente introdotto. Vi abbiamo detto di non spegnere la macchina, se la avete spenta per qualsiasi ragione dovete reintrodurre il programma precedente prima di procedere alle modifiche.

Tasti	Indirizzi	Dati	Commenti
AD	0200	xx	Apro la locazione 0200
DA	0200	F8	Metto la macchina in calcolo decimale

TEMPORIZZAZIONE DI LETTURA



A. Nella zona tratteggiata è indifferente lo stato logico degli indirizzi. B. Nella zona tratteggiata non si deve prendere per valido il dato eventualmente presente sul bus indirizzi.

↑	0201	18	CLC Ripristino l'istruzione di Clear Carry
AD	0006	xx	Apro la locazione 0006
DA	0006	05	Introduco l'addendo 05
↑	0007	05	Introduco l'addendo 05
AD	0200	F8	Mi riporto alla locazione 0200
RUN			Faccio partire il programma

Il risultato sarà 10

Ora cambiamo modo di funzionamento.

Tasti	Indirizzi	Dati	Commenti
AD	0200	F8	Mi riporto alla locazione 0200 il cui contenuto è da modificare.
DA	0200	D8	D8 è il codice operativo della nuova istruzione che sostituisce alla precedente (F8).
RUN			Partenza programma

Il risultato sarà 0A.

Provate ora a cambiare i dati e i modi di funzionamento. Qui di seguito riportiamo qualche esercizio risolto.

$$13_{10} + 18_{10} = 31_{10}$$

$$13_{16} + 18_{16} = 26_{16}$$

Notate bene che non si può scrivere 0A₁₀. Il calcolatore comunque dà sempre un risultato anche se non attendibile. Esempio:

$$13_{10} + 0A_{10} = 23_{10} \text{ Risultano non attend.}$$

NON PERMESSO

La "pagina zero"

Abbiamo fin qui esaminato abbastanza approfonditamente le seguenti istruzioni:

Istruzione	Codice operativo
SEC	→ 38
CLC	→ 18
SED	→ F8
CLD	→ D8

Queste sono tutte istruzioni di un solo byte e che non richiedono altro per essere definite, sono cioè implicite.

Le istruzioni LDA e ADC invece necessitano di una ulteriore definizione. Si deve infatti precisare cosa bisogna caricare nell'accumulatore o cosa bisogna sommarli.

Nella tabella presentata nel capitolo secondo si trova:

A5	LDA	Zero page
65	ADC	Zero page

Cosa significa?

La pagina zero (*zero page*) della memoria è per definizione una zona della memoria che comprende tutte le locazioni comprese fra gli indirizzi 0000 e 00FF ed è quindi formata da 256 byte.

Vedremo quindi che è comodo lavorare con le locazioni di memoria in pagina zero. Perché?

Mappa della memoria dell'AMICO 2000/A

indirizzo

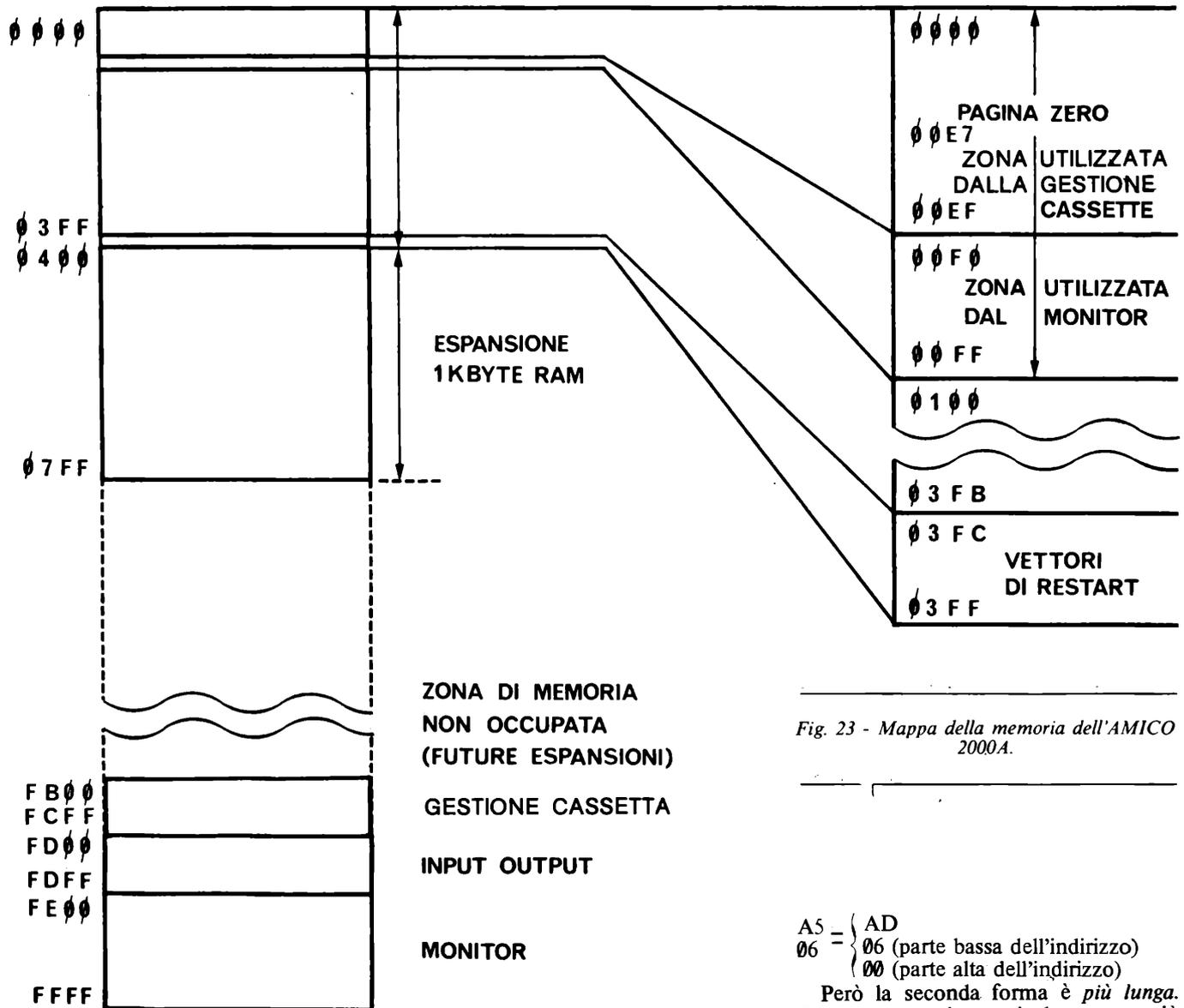


Fig. 23 - Mappa della memoria dell'AMICO 2000A.

Nota: la pagina 1 (da 0100 a 01FF) è normalmente utilizzata per lo Stack.

Perchè si sa già che il primo byte (la cosiddetta parte alta dell'indirizzo) è 00. Allora analizzando meglio questa zona di memoria:

$$\underbrace{00}_{1^{\circ} \text{ byte}} \quad \underbrace{00}_{2^{\circ} \text{ byte}} \quad \div \quad \underbrace{00}_{1^{\circ} \text{ byte}} \quad \underbrace{FF}_{2^{\circ} \text{ byte}}$$

Dove il 1° byte è la parte alta dello indirizzo e il 2° byte è la parte bassa dell'indirizzo.

Ciò che cambia è solo il secondo byte ed è solo quello che dobbiamo precisare.

L'istruzione che carica il contenuto della locazione di memoria 0006 nello accumulatore si scrive:

A5
06

Abbiamo allora che 00 è sottointeso ovvero c'è perchè sto lavorando con una istruzione (A5) che è in pagina zero.

Ma se voglio caricare il contenuto della locazione di memoria 0306 nell'accumulatore devo scrivere:

AD
06 (parte bassa dell'indirizzo)
03 (parte alta dell'indirizzo)

L'istruzione AD nella solita tabella cui abbiamo fatto cenno, viene indicata come "assoluto" (vedi: AD - LDA - Absolute) cioè ad essa deve seguire l'indirizzo completo della locazione di memoria da cui devo prelevare il contenuto.

Ovviamente per la pagina zero esiste l'eguaglianza:

$$A5 = \begin{cases} AD \\ 06 \end{cases} \begin{cases} AD \\ 06 \text{ (parte bassa dell'indirizzo)} \\ 00 \text{ (parte alta dell'indirizzo)} \end{cases}$$

Però la seconda forma è più lunga. Il programma che ne risulta occupa più memoria, è quindi meno efficiente.

A questo punto è necessario fare un altro esercizio.

Trasportiamo un dato dalla locazione 06 alla locazione 00. Cominciamo il programma a partire dalla locazione 0200. Vedere tabella A.

Facciamo partire il programma (come al solito si carica l'indirizzo di partenza 0200 e poi si preme **[RUN]**) e vedremo il contenuto della locazione di memoria 06 copiato nella locazione 00.

Per far ciò basterà come al solito selezionare **[AD] 0006**, vedere il contenuto sul display dei dati, poi selezionate **[AD] 0000** e verificate che in quella locazione si trovi lo stesso dato di prima.

Proviamo ora a modificare il programma come segue.

Tasti	Indirizzi	Dati	Commenti	TABELLA A
AD	0200	xx	Indirizzo di partenza	
DA	0200	A5	LDA Pagina 0. Contenuto della locazione 06 in Accumulatore.	
↑	0201	06	STA Pagina 0. Contenuto dell'accumulatore nella locazione 00.	
↑	0202	85		
↑	0203	00		
↑	0204	4C		
↑	0205	22	JMP MONITOR. Arresto del programma.	
↑	0206	FE		

Tasti	Indirizzi	Dati
AD	0200	A5
DA	0200	AD
↑	0201	06
↑	0202	00
↑	0203	85
↑	0204	00
↑	0205	4C
↑	0206	22
↑	0207	FE

Se facciamo partire il programma (premendo **AD** 0200 poi **RUN**) noteremo che il risultato è lo stesso, ma il programma è più lungo. Le stesse cose dette per la istruzione LDA valgono ovviamente per la istruzione STA. Anche per questa esiste un sistema di indirizzamento in pagina base (o pagina zero) con codice operativo 85 e un sistema di indirizzamento assoluto con codice operativo 8D.

Il precedente programma si può allora riscrivere anche così:

Tasti	Indirizzi	Dati
AD	0200	A5
DA	0200	AD
↑	0201	06
↑	0202	00
↑	0203	8D
↑	0204	00
↑	0205	00
↑	0206	4C
↑	0207	22
↑	0208	FE

Facendo partire il programma anche in questo caso vediamo che il risultato è sempre lo stesso.

Il metodo di indirizzamento

Abbiamo introdotto in sordina un concetto nuovo: IL METODO DI INDIRIZZAMENTO. Ne abbiamo visto due tipi: in pagina base e assoluto. Le stesse istruzioni cambiano codice operativo a seconda del metodo di indirizzamento usato. Dovrete esaminare con molta attenzione la tabella cui abbiamo fatto riferimento.

Sarà molto importante e utile quando si vorranno tradurre in linguaggio macchina dei programmi scritti in linguaggio simbolico.

Per ora fermiamoci qui con il software (il linguaggio di programmazione), cercate di impadronirvi delle poche, ma importanti nozioni e istruzioni che fino ad ora vi abbiamo insegnato provandole direttamente sul microcomputer. Tenete conto che la macchina fa esattamente ciò che voi scrivete sulla carta interpretando le istruzioni a una enorme velocità e soprattutto senza sbagliare.

Approfondimento hardware: il clock

Esaminiamo ora qualche aspetto fisico dello scambio di segnali fra la CPU (l'integrato n. 1 a 40 piedini) e gli altri componenti dell'AMICO 2000/A.

Il problema è: come può il 6502 (la CPU) leggere nella ROM e leggere e scrivere nella RAM?

Avete notato che nell'AMICO 2000/A c'è un quarzo da 1 MHz; questo significa che nel microcalcolatore c'è un oscillatore che, oltre a tutto, è preciso e stabile. Il clock (l'onda quadra) di uscita di questo oscillatore scandisce tutti i tempi principali della macchina. I segnali più importanti sono il 2 (OUT) cioè il piedino 39 dell'integrato e il R/W (cioè Rea-

d/Write, leggi/scrivi) piedino 34 dell'integrato (Fig. 21).

Gli altri fili interessati sono: i piedini 9÷20 e 22÷25 da cui la CPU fa uscire gli indirizzi della memoria interessata (AB0 = Address bit 0, cioè il bit meno significativo dell'indirizzo. AB15 = Address bit 15, cioè il bit più significativo).

Per rinfrescare le idee sul concetto dell'indirizzamento rimandiamo al I° capitolo.

Spieghiamo ora cosa si intende per "bit più o meno significativo".

Se ci rifacciamo alla numerazione decimale, quella che usiamo tutti i giorni, prendiamo in considerazione un numero qualsiasi, ad esempio il numero 35.417.

La variazione in più o in meno di una unità assume evidentemente un valore (o peso) diverso a seconda della posizione della cifra: in questo caso la cifra meno significativa è rappresentata dal 7 (ovvero dalla cifra più a destra del numero) quella più significativa dal 3 (ovvero da quella più a sinistra del numero).

Trasferendo analogamente lo stesso discorso al concetto dei 16 bit (dal bit 0 al bit 15) degli indirizzi, vediamo che il cambiamento di stato (da 0 a 1) del bit più significativo sposta l'indirizzamento di memoria dal primo blocco che va dalla 0 alla 32767^a locazione al secondo blocco che va dalla 32768^a alla 65535^a locazione.

Continuando la descrizione della CPU i piedini 26 ÷ 33 sono quelli tramite i quali essa presenta all'esterno un dato o lo preleva dall'esterno (DB0 = Data Bit 0, cioè il bit meno significativo del dato, DB7 = Data Bit 7, cioè il bit più significativo del dato).

Se la CPU vuole prelevare un dato dalla locazione di memoria 0006 (lo fa perchè noi glielo abbiamo indicato con un'istruzione) presenta il numero 0006 sui piedini di uscita dell'indirizzo, mette a 1 il piedino R/W, e quando la fase Φ 2 è alta (cioè a 1), legge sui fili dei dati quello che la memoria vi ha depositato (vedi figura 21).

Vediamo ora di rispondere a questa domanda: come fa la memoria a immettere i dati sui fili del bus dati?

Per far ciò esiste una decodifica che esamina gli indirizzi che escono dalla CPU. Se questi indirizzi sono quelli voluti la CPU segnala alla memoria che la stanno interrogando e che può immettere i suoi dati sul bus. In figura 22 è rappresentato il più semplice tipo di decodifica che riconosce un indirizzo formato da tutti i.

Se la CPU vuole scrivere un dato nella cella di RAM 0006 presenta all'uscita gli stessi dati del caso precedente ma ora tiene basso (ovvero a 0) il piedino R/W per indicare alla memoria che vuole scrivere. Ed essa lo fa. Il tutto come sempre viene scandito dal clock.

Questi sono i segnali che ora princi-

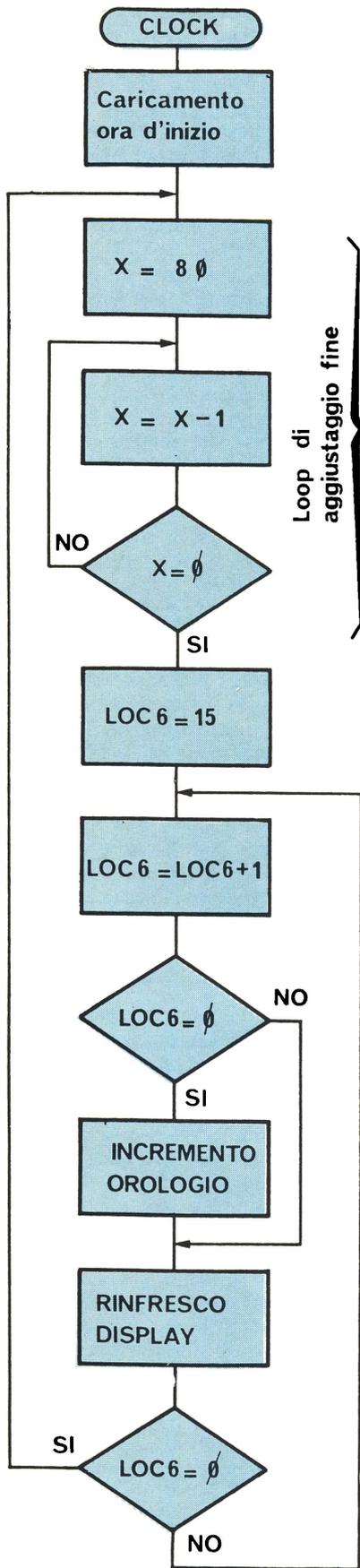


Fig. 24 - Flow chart del programma per realizzare un orologio digitale.

palmente interessano la nostra trattazione; in seguito approfondiremo le funzioni di altri piedini.

Suddivisione della memoria nell'AMICO 2000A

In figura 23 viene riportata la mappa della memoria, cioè si mostra come è posizionata la memoria di questo microelaboratore.

È importante conoscere bene quali locazioni di memoria sono occupate per evitare di scriverci sopra o di interessarle in qualche modo quando si scrivono dei programmi perché diversamente questi ultimi non potrebbero funzionare.

Senza entrare per ora in altri particolari facciamo notare la Pagina Zero (indirizzi 0000-00FF), il Monitor e la gestione della cassetta, che sono posizionati nelle ultime locazioni di memoria.

A proposito, vi ricordate che cosa è il Monitor? È quel programma, residente in ROM, che dà vita alla tastiera e al display, che permette cioè alla CPU di accendere i LED, di acquisire il tasto premuto e di interpretarne il significato. Esso permette inoltre l'esame del contenuto della memoria, la modifica di questo contenuto (se la cella esaminata, è di memoria RAM) e la partenza dei programmi.

Applicazione

Questa volta faremo funzionare il calcolatore da orologio. La calibrazione dell'orologio dovrete farla voi perché dipende dalla precisione del vostro quarzo agendo essenzialmente su una locazione di memoria.

Diamo una breve descrizione del funzionamento di questo programma la cui "costruzione" è stata fatta mediante una cosiddetta "flow chart" o carta di flusso di cui parleremo più diffusamente più avanti e che riportiamo in figura 24.

Il programma è costituito essenzialmente da un contatore, contenuto in tre byte successivi di RAM, gli F9 - FA - FB di pagina base. In F9 sono contenuti i secondi, in FA i minuti, in FB le ore. Ogni secondo noi andremo a sommare 1 della locazione F9 (incremento dei secondi); poi ci chiediamo: siamo arrivati a 60 secondi, azzeriamo i secondi e sommiamo 1 ai minuti. Confrontiamo i minuti con 60 e ripetiamo l'operazione già fatta per le ore; poi si torna all'inizio.

Come al solito il programma viene ca-

ricato da tastiera a partire dalla locazione 0300. Questo programma è stato scritto così come lo trovereste nei testi inglesi e come li scriveremo noi successivamente. In pratica si tratta di ripetere le solite operazioni iniziali selezionando la prima locazione di memoria (tasto **AD**), inserendo il primo dato o codice operativo dell'istruzione (tasto **DA**) procedendo poi ad inserire gli altri dati utilizzando il solito tasto con la freccia (**↑**).

Non analizziamo a fondo questo programma in quanto vi mancano ancora alcune nozioni, per ora ci limitiamo a sottolineare alcune particolarità:

- nella prima colonna è indicata la locazione di memoria in cui si va a porre l'istruzione;
- nella seconda colonna c'è il byte (dato) da introdurre nella locazione di memoria indicata;
- nella terza colonna è riportata l'istruzione in linguaggio mnemonico;
- la quarta colonna è dedicata al commento.

Analizzando ora qualche simbolo. Il simbolo = sta ad indicare che quello che segue è un numero, non una locazione di memoria; quindi se si scrive LDA = 00 si indica che carichiamo il numero 00 nell'accumulatore, in maniera "immediata" come si dice in gergo, ovvero DIRETTAMENTE. Se però scriviamo LDA 00 invece carichiamo nell'accumulatore il contenuto della locazione di memoria 0000.

In linguaggio macchina le due istruzioni si traducono come segue:

```
LDA # 00    A9
           00
LDA 00     A5    siamo in pagina base!
           00
```

Il simbolo \$ prima del numero sta ad indicare che il numero che segue è espresso in Esadecimale.

quindi sarà:
LDA # \$10 = A9 10
LDA # 10 = A9 0A

Ripetiamo che per caricare il programma si procede come al solito:

```
AD 0300    (partiamo da questa locazione)
DA 0300 F8
↑ 0301 A5
```

e via di seguito fino alla fine.

A questo punto dobbiamo caricare il numero dei secondi in un minuto, dei minuti in un'ora e delle ore in un giorno.

```
AD 0003 xx
DA 0003 60 secondi in un minuto
↑ 0004 60 minuti in un'ora
↑ 0005 24 ore in un giorno
```

Ora carichiamo l'ora di partenza (qualche decina di secondi avanti rispetto a quella esatta):

RES 0000 xx La pressione del tasto **RES** ci ha portato ad aprire la locazione di memoria 0000.

DA 0000 (ss) Inserisco i secondi

↑ 0001 (mm) Inserisco i minuti

↑ 0002 (hh) Inserisco le ore

Si carica ora il PC di partenza del programma **AD** 0300; si preme **RUN** nel momento in cui scocca l'ora da noi programmata.

Volendo cambiare l'ora fermiamo l'orologio tramite il tasto **RES** ripetendo le operazioni succitate introducendo i nuovi valori. Se l'orologio non precede giusto si può cambiare il contenuto della locazione 0312: diminuendo il valore contenuto si accelera l'orologio, aumentando il valore si rallenta l'orologio.

Programma di orologio			
Si carica dalla locazione 0300			
LOCAZ.	CODICE OPERAT.	ISTRUZIONE	COMMENTO
0300	F8	SED	
0301	A5	LDA \$00	CONTO IN DECIMALE
0302	00		SECONDI DI START
0303	85	STA \$F9	NEL CONTATORE
0304	F9		
0305	A5	LDA \$01	CARICO I MINUTI
0306	01		
0307	85	STA \$FA	
0308	FA		
0309	A5	LDA \$02	CARICO LE ORE
030A	02		
030B	85	STA \$FB	
030C	FB		
030D	A2	LDX ##95	AGGIUSTAGGIO FINE
030E	95		
030F	CA	DEX	
0310	D0	BNE 030F	
0311	FD		
0312	A9	LDA ##15	AGGIUSTAGGIO GROSSO
0313	15		
0314	85	STA \$06	CONTATORE IN LOCAZ.
0315	06		DI MEMORIA 06
0316	E6	INC \$06	CONTATORE DI ATTESA
0317	06		DI UN SECONDO
0318	D0	BNE 0334	RINFRESCO IL DISPLAY
0319	1A		SE NON E'PASSATO 1 SEC.
031A	A2	LDX ##00	INIZIO SCANSIONE
031B	D0		
031C	AD	LDY ##01	INCREMENTO
031D	01		
031E	18	CLC	
031F	98	TYA	
0320	75	ADC F9,X	PRELEVO L'INCREMENTO
0321	F9		
0322	D5	CMP 03,X	
0323	03		
0324	D0	BNE 032A	
0325	04		
0326	A9	LDA ##00	
0327	00		
0328	F0	BEG 032D	
0329	03		
032A	A0	LDY ##00	
032B	00		
032C	EA	NOP	
032D	95	STA F9,X	
032E	F9		
032F	E8	INX	
0330	E0	CPX ##03	
0331	03		
0332	D0	BNE 031F	
0333	EB		
0334	20	JSR SCADS	ACCENSIONE DISPLAY
0335	0C		
0336	FF		
0337	EA	NOP	EQUALIZZAZIONE
0338	A5	LDA \$06	VEDO SE HO FINITO
0339	06		L'INCREMENTO
033A	D0	BNE 0316	
033B	DA		
033C	4C	JMP 030D	
033D	0D		
033E	03		

L'USO DEL REGISTRATORE A CASSETTE

L'interfaccia per il registratore

A cosa serve l'interfaccia per il registratore e cosa è?

Si tratta innanzitutto di una circuiteria elettronica formata da logiche integrate, elementi discreti e un programma di gestione registrato su PROM che permette al nostro microelaboratore di comunicare i dati e di riceverli da un normale registratore a cassette. Abbiamo detto più volte che i programmi che noi scriviamo per far eseguire determinate funzioni all'elaboratore vengono generalmente introdotti in memoria RAM: questo tipo di memoria come sappiamo si cancella ogni volta che spegniamo la macchina, mentre mantiene indefinitamente i dati finché rimane accesa. Ora, dato che sarebbe almeno "scomodo" tenere sempre accesa la macchina e soprattutto poco pratico, si presenta la necessità di dover conservare questi dati su qualche supporto. Siccome i segnali che girano in un elaboratore non sono altro che livelli alti e bassi di tensione (gli zeri e gli uno) proprio come i fortissimo e i pianissimo di un brano musicale, ma senza livelli intermedi, possiamo allora registrarli in maniera sequenziale su un nastro magnetico alla stessa stregua di un brano musicale.

La funzione del circuito di interfaccia sarà allora quella di presentare al registratore i dati in modo sequenziale per permetterne la registrazione e di consentire all'elaboratore di interpretarli e di ritenerli in memoria nel posto giusto una volta che gli vengono ripresentati. In pratica, come vedremo più avanti nei particolari, sarà possibile trasferire su

nastro magnetico un programma precedentemente scritto nella RAM e viceversa. Quindi invece di avere ad esempio una biblioteca di programmi scritti su carta (come nel nostro caso abbiamo fatto fino ad ora) e doverli ogni volta reinserire a mano tramite la tastiera, li potremo avere registrati su cassette, numerati e titolati e sempre pronti all'uso.

Aggiungendo, come vedremo, solo qualche componente alla scheda AMICO 2000/A potremo collegarla ad un registratore ed eseguire tutte le operazioni di cui abbiamo parlato. I componenti per l'interfaccia registratore a cassette, così come le due RAM che servono per completare la scheda sono reperibili presso lo stesso fornitore dell'AMICO 2000/A.

Il montaggio

Per il montaggio dell'interfaccia cassette procederemo come quando abbiamo realizzato l'intera piastra AMICO 2000/A.

L'elenco dei componenti è riportato nella tabella 10.

Per il montaggio e il corretto posizionamento dei vari componenti ci riferiamo alla serigrafia del circuito stampato dell'AMICO 2000/A del capitolo 3°.

Dopo averne identificato il valore tramite la tabella 11 cominceremo per primo a saldare le resistenze da R34 a R43 prestando la solita attenzione nell'uso del saldatore e dello stagno. Facciamo notare che il valore di queste resistenze R34 ÷ R37 può essere indifferentemente di 82 o 100

Possiamo ora saldare gli zoccoli prestando attenzione all'orientamento degli

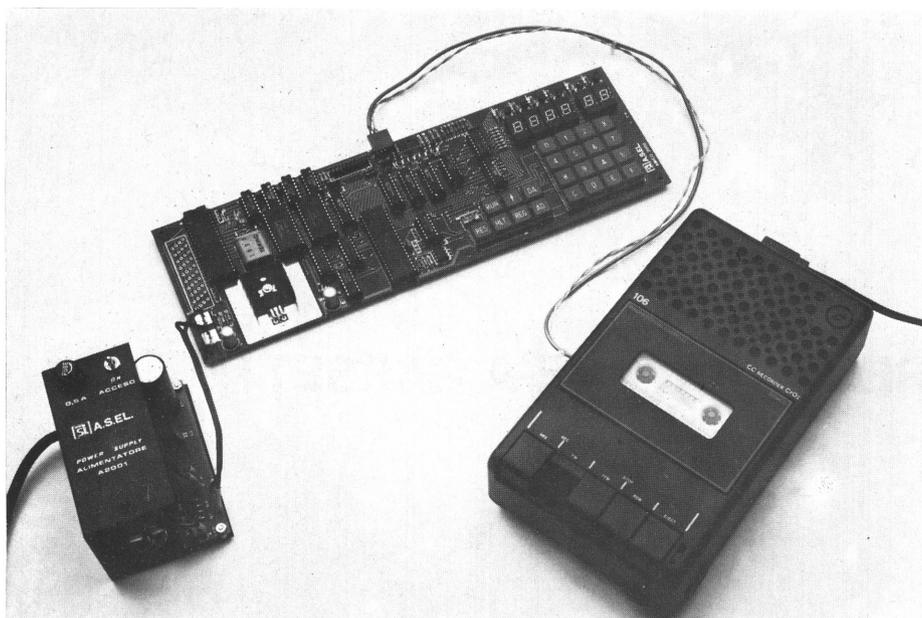
stessi (l'angolo interno smussato degli zoccoli deve corrispondere al punto o alla tacca riportata accanto o sul lato della serigrafia di ogni integrato) perché quello dovrà poi essere l'orientamento dell'integrato che ci va inserito. Cominciamo con lo zoccolo di IC19 (8 piedini), poi IC8, IC20, IC21 (14 piedini), indi IC7 (16 piedini) e IC10 (24 piedini). Se avete anche l'espansione RAM montate gli zoccoli degli integrati IC13, (16 piedini).

Una breve nota tecnica: IC8 nella parte della circuiteria della interfaccia cassette, ma viene fornito ugualmente per poter completare la scheda (IC8 serve per l'espansione del BUS esterno).

Ora possiamo saldare tutti i condensatori da C6 a C13. Facciamo notare subito che a seconda del tipo di registratore usato si può eliminare C8 e fare un cortocircuito al posto di C7. Inoltre può essere vantaggioso montare un condensatore da 0,22 µF in Milar fra i punti IN e GND riconoscibili nella piastra in mezzo in alto sotto la denominazione "RECORD".

Per ultimo salderemo i diodi LED D4 - 5 - 6 - 7 che come vedremo servono a sapere cosa sta accadendo quando è in funzione il registratore. Attenzione: montate i LED con la polarità corretta: in pratica basta far corrispondere il lato smussato del LED con il - della serigrafia.

A questo punto, prima di inserire gli integrati, controllate tutte le saldature, il corretto posizionamento di resistenze, condensatori e l'orientamento degli zoccoli degli integrati. Potete ora inserire con la dovuta attenzione tutti gli IC dopo averli ben identificati per evitare di scambiarli di posto. Ricordiamo ancora una volta che gli integrati devono essere orientati con la tacca in corrispondenza



L'AMICO 2000/A collegato al registratore.

del puntino sulla serigrafia e (se non avete commesso errori) dell'angolo smussato interno allo zoccolo: in pratica tutti con la tacca rivolta verso l'alto della scheda disposta orizzontalmente.

L'utilizzo del registratore e suo collegamento

Nella PROM IC10 risiede il programma di gestione della cassetta magnetica che consente le operazioni di registrazione e lettura. Come abbiamo precedentemente detto, questo programma vi permette di utilizzare il vostro normale registratore a cassette per memorizzare i programmi di maggior interesse, per conservarli e caricarli quando vi servono velocemente e senza errori.

Dobbiamo avvertire che va prestata una particolare attenzione al tipo di registratore e alle cassette utilizzate. Non che questi debbano essere ad alta fedeltà, ma è importante che il registratore sia a posto, con testine pulite e trascinamento del nastro uniforme (cinghia di trasmissione nuova e controllo di velocità di trascinamento efficiente). Il nastro deve essere di qualità e la meccanica della cassetta perfetta (non utilizzate cassette vecchie e impolverate).

Per le cassette consigliamo vivamente di utilizzare quelle da 5 o 10 minuti per lato, oppure le C60, ma non registratevi sopra più di tre o quattro programmi in quanto ogni volta dovete far passare tutto il nastro per richiamare l'ultimo programma registrato. È importante che sappiate che sul nastro vengono registrati ben 300 bit al secondo circa così che per un programma da 1 kbyte (8000 bit) che è molto lungo l'intera registrazione avviene circa 45 secondi comprendendo anche le code di inizio e fine programma. Se poi pensate che l'AMICO 2000/A con l'espansione RAM ha circa 1,5 kbyte di RAM nella quale possono essere scritti i programmi, vedete che qualche minuto è più che sufficiente per registrare numerosi piccoli programmi.

La registrazione, se la sensibilità di registrazione non è automatica, va fatta con il potenziometro del volume di registrazione a metà corsa; la lettura del nastro avviene invece con il potenziometro del volume a 2/3 circa. Su questi particolari comunque torniamo fra poco.

L'uscita verso il registratore è contrassegnata nell'AMICO 2000A dalla parola RECORD ed è posizionata in alto e in mezzo alla scheda (vedere sulla serigrafia).

Sotto la parola RECORD ci sono tre capicorda contrassegnati da:

- GND, che va collegato alla massa del registratore;
- IN, che va collegato all'ingresso del microfono del registratore;

Tabella 10 - Elenco componenti circuito per interfaccia cassette ed espansione memoria RAM

Resistori (tutti da 1/4 W, tolleranza 5%)

R34-R35- R36-R37	:	82 Ω
R38	:	22 kΩ
R39	:	1,8 kΩ
R40	:	1,8 kΩ
R41	:	100 Ω
R42	:	33 kΩ
R43	:	3,3 kΩ

Condensatori

C6-C7	:	6,8 nF - polistirolo
C8	:	1 nF - polistirolo
C11-C12	:	0,1 μF - ceramico a disco

Diodi

D4-D5-D6-D7	:	diodi LED
-------------	---	-----------

Integrati

IC7	:	74S287 (oppure 93427) - PROM di decodifica
IC8	:	74LS30 - NAND a 8 ingressi
IC10	:	93448 - PROM gestione interfaccia cassette
IC13-IC14	:	TMS4045 (opp. 2114) RAM 1K x 4 statica
IC19	:	LM358 - convertitore di ingresso per interfaccia cassette
IC20	:	74LS132 - quadruplo NAND a due ingressi Schmitt Trigger
IC21	:	74LS38 - quadruplo NAND a due ingressi

- OUT, che va collegato con l'uscita dell'altoparlante del registratore.

Il vostro registratore dovrebbe avere quindi una uscita per altoparlante supplementare che esclude quello incorporato. Se avete solo una piastra di registrazione non amplificata è possibile che non vada bene data la sua bassa tensione di uscita.

Per quanto riguarda il collegamento pratico al registratore dobbiamo vedere il tipo di prese che questo ha: in genere quelli portatili monofonici a cassetta hanno due prese standard, una a 7 poli e una cosiddetta punto-linea. La prima serve per l'ingresso del microfono, la seconda per collegare un altoparlante esterno.

Il capicorda GND (massa) va allora collegato al piedino 2 della presa a 7 poli tramite una adatta spina a 7 poli tipo Philips (Norme DIN) e al - della presa per l'altoparlante tramite adatta spina.

Il capicorda OUT con il piedino + della presa per altoparlante.

Il capicorda IN con il piedino 1 della presa a 7 poli.

La fig. 25 mostra come vanno fatti in pratica i collegamenti. Possibilmente, ma non è indispensabile dato l'alto livello dei segnali, il collegamento al piedino 1 va fatto con cavetto schermato.

Se avete prese di tipo diverso sarà sufficiente identificare i piedini che ci interessano per fare i collegamenti come sopra descritto.

Possiamo ora collegare il registratore, ma prima di farlo partire è necessario sapere alcune cose.

Noi possiamo accedere ai programmi che ci permettono di leggere e scrivere sul registratore posizionando il PC come segue:

SCRITTURA Program Counter FBBC

LETTURA Program Counter FC54

Ricordiamo che per posizionarci ad un certo indirizzo basta usare il tasto **AD** seguito dall'indirizzo stesso (in questo caso FBBC e FC54).

Tutto il programma utilizza le locazioni di memoria comprese fra la FB00 e la FCFF, cioè 512 locazioni di memoria.

A questo punto introduciamo l'uso del tasto **REG** del quale non abbiamo ancora definito la funzione e che ora ci sarà molto utile.

Questo tasto ha la funzione di richiamare un particolare indirizzo che noi abbiamo precedentemente memorizzato nelle locazioni di memoria 00F6 e 00F7. È chiaro che questo tasto può risultare molto utile quando si vuol far partire sempre un determinato programma. Basta infatti premerlo per richiamare sul display indirizzi la locazione di memoria in cui comincia il programma.

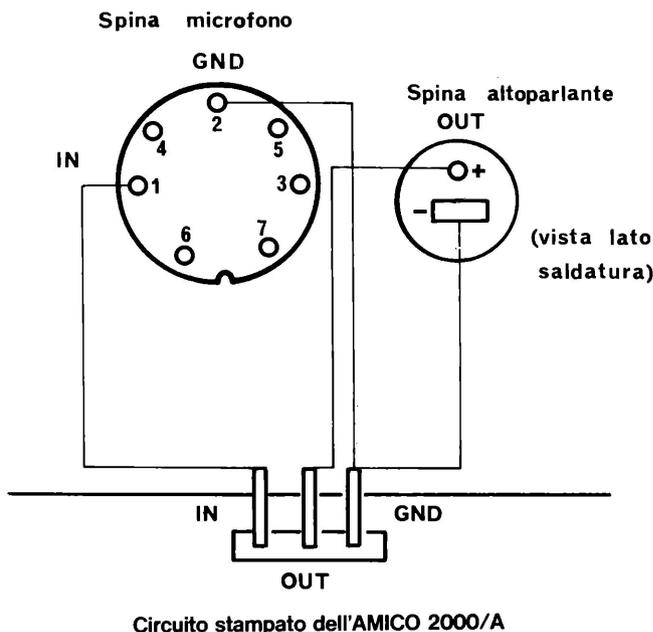


Fig. 25 - Collegamento pratico fra AMICO 2000A e prese di registrazione e uscita altoparlante del registratore a cassetta.

Operazione di lettura del nastro

Accendiamo allora il nostro AMICO 2000 A e impariamo subito a servirci del tasto **REG**

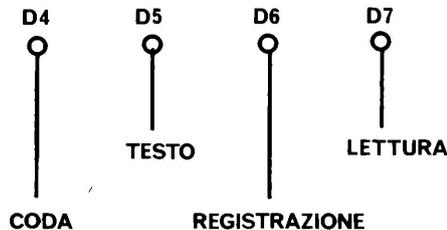


Fig. 26 - I quattro led D4, D5, D6, D7, servono a controllare le operazioni di lettura e scrittura del nastro. Ciò che sta avvenendo è indicato dall'accensione del rispettivo Led secondo le indicazioni riportate.

Carichiamo il Program Counter che vogliamo associare a REG (nel nostro caso quello di lettura della cassetta) nelle locazioni di memoria 00F6 e 00F7, premiamo i tasti:

AD **00F6** **DA** **54** **↑** **FC** **REG**

A questo punto vedrete sul display indirizzi FC54. Infatti avete introdotto la parte bassa dell'indirizzo (54) nella locazione 00F6 e la parte alta dell'indirizzo (FC) nella locazione 00F7.

Con la semplice pressione di **REG** possiamo ora richiamare l'indirizzo di partenza della routine di lettura del nastro.

Prendete ora la cassetta preregistrata fornita con il kit della interfaccia per registratore introducetela in modo da leggerla sul lato 1, riavvolgetela, fate partire il registratore in lettura (come se doveste ascoltare una musica) dopo aver posizionato a metà il potenziometro del volume e premete subito dopo il tasto **RUN** sull'AMICO 2000/A.

Tabella 11 - Identificazione delle resistenze contenute nel kit delle espansioni RAM e interfaccia registratore a cassetta.

Valore	Codice colore			
	1° colore	2° colore	3° colore	4° colore
82 Ω	grigio	rosso	nero	oro
100 Ω	marrone	nero	marrone	oro
1,8 kΩ	marrone	grigio	rosso	oro
3,3 kΩ	arancio	arancio	rosso	oro
22 kΩ	rosso	rosso	arancio	oro
33 kΩ	arancio	arancio	arancio	oro

CODA	BYTE DI START	IDENTIFICATORE	INDIRIZZO DI CARICAMENTO	NO. DI BYTES	TESTO	CHECKSUM	CODA
------	---------------	----------------	--------------------------	--------------	-------	----------	------

Fig. 27 - Disposizione fisica dei dati registrati sul nastro.

Dopo qualche secondo si devono accendere i LED relativi alla coda e alla lettura del programma registrato, rispettivamente D4 e D7 dell'AMICO 2000/A (vedi fig. 26). Se ciò non avvenisse aumentate il volume all'accensione dei LED.

Il lato 1 della cassetta contiene solo la registrazione di una lunga coda (tutti zeri) e serve proprio a tarare il volume di uscita del registratore.

Una volta regolato correttamente il volume potete fermare il microelaboratore premendo **RES** e il registratore tramite il suo tasto di STOP.

Vediamo ora come si introducono i parametri che permettono di caricare i programmi nella memoria RAM del microelaboratore cioè di trasferirli dal nastro nel quale sono registrati.

Girate allora la cassetta sul lato 2 e riavvolgetela.

I parametri da inserire nel microelaboratore sono sostanzialmente due (il NUMERO DEL PROGRAMMA e L'INDIRIZZO DAL QUALE SI VUOLE CHE QUESTO COMINCI AD ESSERE CARICATO) e vanno definiti come segue:

Premiamo ora **REG** e apparirà sul display **FC54 A9** che è l'inizio del programma di lettura. Si fa partire ora il registratore (tasto di ascolto) quindi si preme subito RUN per far eseguire il programma. A questo punto il display indirizzi e dati del calcolatore si spegne mentre dopo qualche secondo si accendono i LED D4 e D7, segno che l'elaboratore sta leggendo la coda del programma; dopo sei secondi circa si accendono i LED D5 e D7, segno che si è in fase di lettura del programma registrato.

Quando la lettura del testo è finita sul display appare **0000 01** (01 è il numero del programma caricato).

Appena ciò avviene spegnete il registratore agendo sul suo STOP. Se al posto di 01 dovesse apparire FF la lettura è avvenuta male. Bisogna allora ripetere la lettura nella maniera precedente fino ad ottenere la risposta voluta.

Per verificare che tutto sia avvenuto

nel modo corretto potete controllare che a partire dalla locazione **0100** siano presenti i dati relativi al programma N. 1 descritto al paragrafo "Utilizzo della cassetta registrata".

Vediamo ora di analizzare come sono fisicamente registrati i dati sul nastro dando un'occhiata alla fig. 27.

Il primo tratto del programma è una coda formata da tanti zeri che servono per permettere al lettore di sincronizzarsi; segue un byte che identifica la partenza del programma; segue poi un numero che dice di quale programma si tratta; segue l'indirizzo dal quale si dovrebbe caricare il programma (questo come abbiamo visto è modificabile con una istruzione specifica); segue un numero che indica il numero di parole che formano il programma; segue il testo del programma; segue una parola di controllo (checksum) ed infine una coda di chiusura.

Tutto questo, si noti bene, viene in gran parte fatto automaticamente quando dobbiamo registrare un programma dal microelaboratore sul nastro magnetico ad opera dell'apposito programma di scrittura, ma su ciò torneremo fra poco.

Un parametro molto utile è il numero di identificazione del programma, poiché se abbiamo più programmi registrati su uno stesso lato del nastro è possibile caricare quello desiderato, a selezionarlo ci penserà il microcomputer.

Supponiamo per esempio di aver registrato sul nastro i programmi 1 - 2 - 3 - 4 - 5 e di voler caricare quello N. 4.

Per farlo riavvolgiamo il nastro completamente e scriviamo **04** nella locazione di memoria **0000** e **FF** (perché vogliamo ricaricarlo a partire dall'indirizzo originale) nella locazione di memoria **0002**.

Facciamo partire il microelaboratore e il registratore come abbiamo precedentemente descritto e vedremo che dopo un certo tempo l'AMICO 2000/A mostrerà sul display dei dati il numero **04** segno che è stato caricato proprio quel programma, a questo punto fermeremo il registratore.

Attenzione però che prima di procedere alla lettura del nastro dovrete aver caricato precedentemente il Program Counter all'indirizzo di partenza del programma di lettura (indirizzo **FC54**).

Avvertiamo infine che se carichiamo un numero di programma non previsto sul nastro il microelaboratore continuerà la sua inutile ricerca fino alla fine del nastro. In questo caso per ripristinare il tutto basterà premere il tasto **RES**.

INDIRIZZO	DATO	SPIEGAZIONE
0000	xx	Numero del programma da caricare, xx è un numero diverso da FF e da 00.
0001	xx	Parte bassa dell'indirizzo dal quale si vuole che cominci il caricamento del programma nella RAM.
0002	xx	Parte alta dell'indirizzo dal quale si vuole che cominci il caricamento del programma nella RAM.
	FF	Se inseriamo come parte alta dell'indirizzo FF il programma viene caricato automaticamente a partire dall'indirizzo già registrato sul nastro.

Notare che xx è un simbolo grafico che indica un numero qualsiasi.

Allora, detto questo noi ad esempio vogliamo:

- 1 - Caricare il programma N. 1.
- 2 - Caricarlo in RAM a partire dall'indirizzo registrato sul nastro.

Premiamo quindi i tasti:

RES **DA** 01 (numero del programma) **↑** **↑** **FF**.

Il tasto **↑** viene premuto due volte perché non ci interessa dare la parte bassa dell'indirizzo in quanto ho deciso di far partire il programma dall'indirizzo già registrato inserendo FF.

Tabella 12 - Parametri richiesti dal programma di registrazione

Indirizzo	Spiegazione
0000 0001	DAL } DAH } Indirizzo della locazione di memoria dalla quale si vuole iniziare a registrare
0002 0003	AL } AH } Indirizzo dell'ultima locazione di memoria che si vuole registrare
0004	IDT Identificatore (N. del programma)

Operazione di registrazione sul nastro

Cambiate ora cassetta mettendone una delle vostre, riavvolgetela/e regolate il livello di registrazione a metà se questo non è automatico.

Serviamoci anche ora del tasto REG caricando il PC di partenza della routine di REGISTRAZIONE FBBC:

[AD] 00F6 [DA] BC [↑] FB

Caricheremo ora i parametri richiesti dal programma seguendo le indicazioni riportate in tabella 12.

Le sigle accanto all'indirizzo hanno un valore simbolico, non sono ovviamente dei dati: DA L, H significa che vogliamo registrare da (L = parte bassa o Low e H = parte alta o High dell'indirizzo); A L, H significa che vogliamo registrare fino a quella determinata locazione di memoria.

Facciamo un esempio: si vuole registrare il contenuto delle locazioni di memoria (ovvero il programma) che vanno dalla:

0010 (cioè DAH = 00 e DAL = 10) alla 001F (cioè AH = 00 e AL = 1F) e vogliamo chiamare il programma con il numero 13.

Procederemo allora come segue:

[AD] 0000
[DA] 10 (DAL)
[↑]
00 (DAH)
[↑]
1F (AL)
[↑]
00 (AH)
[↑]
13 (IDT)

Dopo aver fatto ciò andiamo alla locazione 0010 e scriviamo dei numeri progressivi da 00 a 15 da questa locazione alla 001F.

Per il momento supponiamo che i numeri che abbiamo scritto in queste locazioni costituiscano il programma che vogliamo trasferire sul nastro.

Per scrivere i numeri procediamo come ormai sappiamo: [AD] 0010 [DA] 00

[↑] 01 [↑] 02 [↑] 03 ... etc. fino a 15.

Se premiamo ora il tasto [REG] apparirà sul display FBBC D8 che è l'inizio del programma di registrazione.

Si fa partire ora il registratore in *Registrazione*, si aspetta che sia passata la coda non magnetica del nastro, quindi si preme [RUN].

Si accenderanno immediatamente i LED D6 e D4, poi per breve tempo i LED D6 e D5, quindi ancora D6 e D4 e infine il display del microelaboratore mostrerà 0000 indicando con questo che il programma è stato registrato. A questo punto si deve fermare anche il registratore premendo il suo tasto di STOP.

Per verificare di averlo caricato correttamente possiamo per prima cosa modificare il contenuto della locazione dalla 0010 alla 001F ponendole ad esempio tutte a FF. Ora possiamo caricare il programma registrato ripetendo l'intera procedura descritta in precedenza al paragrafo "operazione di lettura".

Ricontrolliamo il contenuto delle locazioni che abbiamo appena modificato: il contenuto dovrà essere quello del programma che abbiamo appena caricato.

Se vogliamo caricare il programma registrato a partire da una locazione di memoria diversa da quella registrata, ad esempio la 0200, procediamo come segue:

[AD] 0000 [DA] 13 [↑] 00 [↑] 02.

In tal modo il programma verrà caricato a partire dalla locazione 0200. Procediamo quindi all'operazione di lettura del nastro e verifichiamo alla fine che il testo registrato sia posizionato a partire dalla locazione 0200 fino alla 020F.

Montaggio dell'espansione RAM

È veramente elementare: si tratta, se non lo avete già fatto in precedenza, di montare gli zoccoli relativi agli integrati IC13 e IC14 e cioè le RAM statiche da 4 kbit TMS4045 (o 2114) e inserire le stesse badando al corretto orientamento.

Il collaudo è altrettanto semplice: basta portarsi all'indirizzo 0400, da questo fino a 07FF deve essere possibile scrivere dei dati.

Utilizzo della cassetta registrata: "La tombola elettronica"

Nel lato 2 della cassetta sono registrati due programmi: il primo è semplicemente una serie di numeri esadecimali dallo 0 al 32 registrati in locazione di memoria successive che vanno dalla locazione 0100 alla 0132.

Questo programma serve solo per controllare l'esattezza delle operazioni di caricamento in RAM.

Il programma numero 2 è invece la versione "anni 2000" del vecchissimo gioco della tombola. In pratica il nostro elaboratore farà le funzioni del sacchetto dal quale si estraggono i 90 numeri.

Dopo aver riavvolto il nastro selezioniamo, come già sappiamo fare, il programma N. 02 e lo carichiamo a partire dalla locazione già registrata sul nastro.

La prima istruzione del programma è stata caricata nella locazione 0200, mentre l'istruzione di inizio è posizionata nella locazione 0230.

Per far partire il programma facciamo allora:

[AD] 0230 [RUN]

apparirà sul display la parola VIA.

A questo punto per estrarre i numeri (che arrivano in modo del tutto casuale) basta premere il tasto F; il numero estratto compare sul display dati. Premendo successivamente F per 90 volte estrarremo tutti i numeri.

Il nostro programma però non si limita a far questo, infatti permette di eseguire in qualsiasi momento i seguenti controlli: *Numero delle estrazioni effettuate* (tasto [E]).

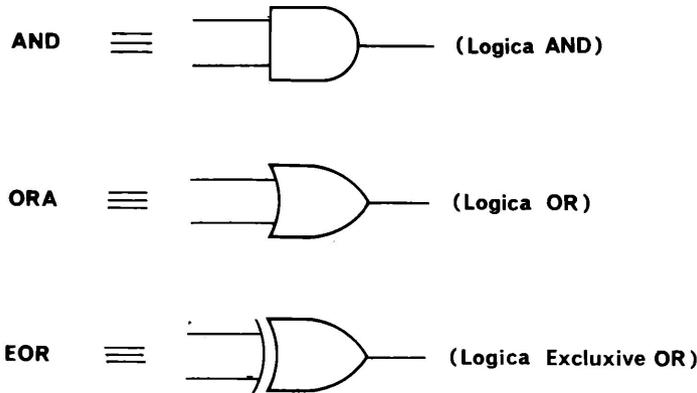


Fig. 28 - Funzioni logiche AND, OR, EOR.

Premendo in qualsiasi momento il tasto E comparirà sul display indirizzi il numero delle estrazioni effettuate seguito da un trattino e da una E. Ad esempio 09 - E che significa che fino a quel

momento sono stati estratti 9 numeri.

Verifica della avvenuta estrazione di un numero (tasto **D**)

Sempre in qualsiasi momento, volen-

do controllare se un certo numero, ad esempio il 24, è stato già estratto basterà battere sulla tastiera il numero richiesto (nell'esempio il 24) quindi premere D.

Se il numero è stato già estratto compare: 24 i (i sta per Sì).

Se il numero non è stato estratto compare: 24 o (o sta per No).

Se si preme il tasto D senza avere introdotto prima il numero, sul display comparirà 00.

Esame della sequenza dei numeri estratti (tasti **B** e **C**).

È possibile in ogni momento esaminare tutti i numeri nella stessa successione nella quale sono usciti. Per far ciò si preme prima il tasto B facendo comparire sulla prima cifra a sinistra del display indirizzi la lettera P che significa primo numero estratto. A questo punto premendo il tasto C compare sulle prime due cifre a sinistra del display indirizzi il primo numero estratto. Premendo successivamente il tasto C comparirà il 2°, 3°, 4°, ecc. numero estratto fino all'esaurimento.

Premendo ulteriormente il tasto C comparirà sul display: U - - -

Questo significa che abbiamo visto tutti i numeri estratti.

Questa operazione di controllo può essere ripetuta quante volte si vuole.

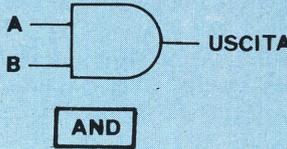
Fine del gioco

Dopo aver estratto il 90° numero, se premiamo ancora il tasto F comparirà sul display la parola FINE. Per ricominciare basterà premere **RUN**.

Un'ultima nota: alla estrazione degli ultimi dieci numeri, la ricerca da parte del calcolatore di quelli rimasti potrà essere più laboriosa. Può capitare di dover premere più di una volta il tasto F per estrarre il numero.

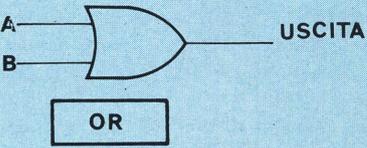
TABELLA 13 -
Tavola della verità per le funzioni logiche AND, OR, e exclusive OR

A	B	USCITA
0	0	0
0	1	0
1	0	0
1	1	1



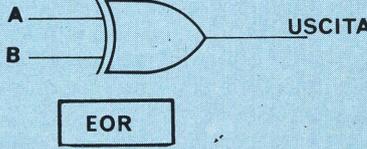
AND

A	B	USCITA
0	0	0
0	1	1
1	0	1
1	1	1



OR

A	B	USCITA
0	0	0
0	1	1
1	0	1
1	1	0



EOR

SOFTWARE

Esaminiamo in questo capitolo alcune istruzioni molto interessanti e di estrema utilità: AND, OR, EOR.

Sono istruzioni logiche che ricalcano esattamente le funzioni svolte dai circuiti integrati che già conoscete (Vedi fig. 28).

Per completezza riassumiamo le nozioni fondamentali di queste funzioni logiche.

Se si hanno 2 segnali di ingresso, A e B che entrano in una porta AND, l'uscita della porta risponde alla logica riportata nella tabella 13: cioè se i due segnali di ingresso sono a 1, l'uscita va a 1. Se un qualsiasi segnale di ingresso va a 0, l'uscita va a 0.

Per le altre due funzioni si veda ancora la tabella 13.

Facciamo notare sull'ultima funzione, che se i due ingressi sono uguali, l'uscita va a 0, se sono diversi l'uscita va a 1.

Il microprocessore è in grado di realizzare queste funzioni operando bit per bit.

Vediamo cosa significa.

Si abbia una parola in ACCUMULATORE (per esempio A5) e si voglia fare l'AND con F6: la CPU esegue l'operazione come rappresentato in Fig. 29.

Bit per bit vuol dire che si esegue l'operazione di AND sui bit di ciascuna colonna senza che ci sia influenza fra una colonna e le altre.

Proviamo ad eseguire praticamente l'operazione appena descritta con il microelaboratore. Scriviamo il programma quanto segue:

Indirizzo	Codice	Codice	Commenti
	Macchina	Mnemonic	
0200	A9	LDA #SA5	Carico A5 in accumulatore
1	A5		
2	29	AND #F6	Eseguo AND con il dato immediato F6
3	F6		
4	85	STA 00	Porto il risultato presente nell'accumulatore in 0000
5	00		
6	4C	JMP MONITOR	Istruzioni di arresto. Il controllo delle operazioni ritorna al programma di Monitor e rende operativa la tastiera e il display
7	22		
8	FE		

Introduciamo come al solito il programma a partire dalla locazione 0200:

AD 0200 DA A9 ↑ A5 ↑ e così via fino a introdurre FE. Facciamo partire il programma usando il tasto REG e cioè: AD 00F6 DA 00 ↑ 02 REG

Sul display compare: 0200 A9

Premiamo RUN comparirà sul display:

0000 A4 che è il nostro risultato.

Fate un po' di esercizi cambiando i dati di partenza alle locazioni 0201 e 0203. Per esempio eseguiamo 53 AND 07. Il risultato è 03. Premiamo allora i tasti:

AD 0201 DA 53 ↑ ↑ 07 REG
RUN

e si ottiene il risultato cercato.

Provate ora a verificare con l'AMICO 2000/A le seguenti uguaglianze:

- 14 AND 5E = 14
- 5A AND A5 = 00
- 2F AND 89 = 09
- D6 AND 77 = 56

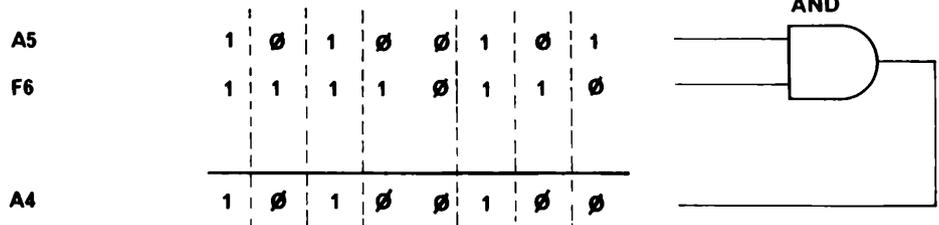


Fig. 29 - Come il microelaboratore esegue l'operazione logica AND.

Prestiamo ora attenzione a quanto segue: eseguendo l'operazione di AND fra un numero qualsiasi e 00 si ottiene come risultato 00. Mentre facendo AND fra un numero qualsiasi ed FF si ottiene come risultato lo stesso numero di partenza (Esempio 3A AND 00 = 00; 3A AND FF = 3A).

Passiamo ora ad analizzare l'istruzione ORA.

Lasciamo invariato il programma appena scritto cambiando la seconda istruzione.

0200	A9	LDA #SA5	
1	A5		
2	09	ORA #F6	
3	F6		
4	85	STA \$00	
5	00		
6	4C	JMP MONITOR	
7	22		
8	FE		

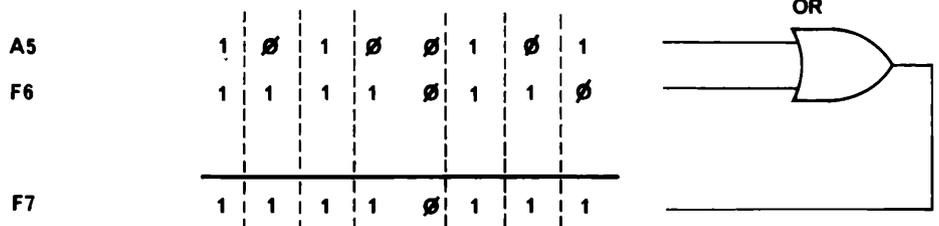


Fig. 30 - Come il microelaboratore esegue l'operazione logica OR.

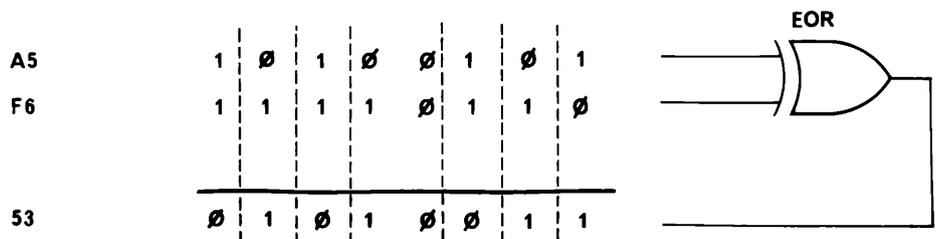


Fig. 31 - Come il microelaboratore esegue l'operazione logica EOR.

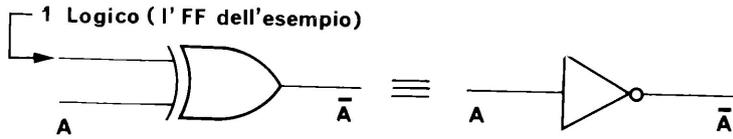


Fig. 32 - Operazione di EOR con 1 (FF).

La CPU esegue l'operazione come rappresentato in fig. 30. Verifichiamo anche questo risultato facendo girare il programma.

Vediamo anche in questo caso alcune importanti "curiosità". L'OR di un numero qualsiasi con 00 lascia invariato il numero. L'OR di un numero qualsiasi con FF dà come risultato FF.

Esempio:

55 OR 00 = 55

7B OR FF = FF

L'ultima operazione, EOR, è forse la

più inconsueta, ma è comunque molto utile. Il solito programma di prova si modifica agendo sulla seconda istruzione e cioè inserendo all'indirizzo 0202 il dato 49 (che è il codice macchina dell'istruzione EOR). La CPU esegue l'operazione come rappresentato in fig. 31.

Verificate questa operazione facendo girare il programma. Anche sull'EOR dobbiamo fare delle osservazioni particolarmente importanti.

Notiamo subito che eseguire l'EOR di un numero con 00 lascia invariato il numero stesso.

Eseguire invece l'EOR di un numero con FF, lo nega. Cosa vuol dire? Se facciamo A5 EOR FF risulta:

```
A5  1010 0101
FF  1111 1111
-----
     0101 1010
```

Notate che dove c'era lo zero nella parola di partenza, c'è 1 nel risultato e viceversa. Abbiamo cioè *negato* la parola.

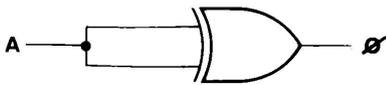


Fig. 33 - Operazione di EOR di un numero con se stesso

La stessa equazione può essere rappresentata con i simboli logici come in fig. 32.

Notiamo che se il segnale di ingresso (A) è 0 l'uscita (A) è 1, se A = 1 allora A = 0.

Ultima particolarità: l'EOR di una parola con se stessa dà per risultato 0 (vedi fig. 33). Verificalo.

Solitamente vengono usati dei simboli standard per indicare le tre operazioni appena analizzate. Essi sono:

∧ per AND

∨ per OR

⊕ per EOR

Detti A e B i due numeri si scrive allora:

$A \wedge B$; $A \vee B$; $A \oplus B$.

Le istruzioni svolte sono state tradotte in linguaggio macchina nel programma solo per il sistema di indirizzamento immediato. In questo sistema di indirizzamento il dato che dobbiamo utilizzare è contenuto nel secondo byte dell'istruzione.

Se si deve caricare il numero 12 nell'accumulatore, cioè se dobbiamo caricare l'accumulatore con il dato *immediato* 12, scriveremo in linguaggio mnemonico LDA #S12, che si traduce il codice macchina A9 12. Eseguita l'istruzione troviamo caricato in accumulatore proprio il numero 12.

Abbiamo già spiegato questo concetto. I codici operativi relativi alle istruzioni AND, OR, EOR nei tre modi di indirizzamento fino ad ora descritti sono riportati in tabella 14.

Un'ultima cosa. Le operazioni logiche descritte influenzano 2 altri bit dello Status register: il bit di ZERO e il bit di NEGATIVO.

Il registro di Status deve essere aggiornato allora come mostrato in fig. 34.

Se il risultato dell'ultima operazione eseguita è = 00, il bit Z viene messo automaticamente a 1.

Se il risultato dell'ultima operazione eseguita è negativo (ovvero quando il bit più significativo del risultato è = 1) il bit N viene messo automaticamente a 1.

Spiegheremo il concetto di numero negativo nella matematica binaria nel prossimo capitolo.

I bit dello Status sono molto importanti (ne vedremo ancora tre) perché servono a fare i salti condizionati che verranno analizzati anch'essi nel capitolo.

Tabella 14 - Codici operativi delle istruzioni AND, OR e EOR.						
Istruzione	Indirizzamento					
	immediato	#	pagina zero	#	assoluto	#
AND	29	2	25	2	2D	3
ORA	09	2	05	2	0D	3
EOR	49	2	45	2	4D	3

Nota: # è il numero di byte che formano l'istruzione.

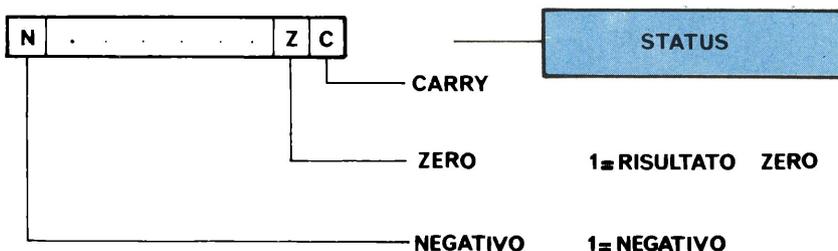


Fig. 34 - Bit della Status register fino ad ora esaminati.

Esercitazione

Vi proponiamo un semplice esercizio logico.

Implementiamo con un programma la funzione NAND (quella svolta dalla logica SN 7400 per intendersi).

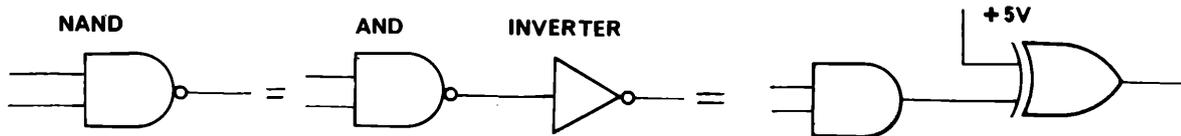


Fig. 35 - Equazione logica di NAND costruita con le funzioni AND e di inversione.

Notiamo che vale la equazione logica riportata in fig. 35. Per realizzare quindi la funzione NAND bisogna prima fare un AND, poi un EOR con FF.

Problema:

Eeguire il NAND fra il contenuto della locazione di memoria 0001 e quella della locazione 0002. Mettere il risultato nella locazione 0000.

Soluzione:

```
0200 A5 LDA S01
Indirizzamento in pagina zero
1 0
2 25 AND S02
Indirizzamento in pagina zero
3 02
4 49 EOR =SFF
Indirizzamento immediato
5 FF
6 85 STA S00
Indirizzamento in pagina zero
7 00
8 4C JMP MONITOR
9 22
A FE
```

Utilizzate questo programma caricando dei dati in 0001 e 0002.

Verificate che i risultati tornino:

```
27 NAND AE = D9
1E NAND 99 = E7
9B NAND 66 = FD
```

Esercizio. Eeguire il NOR fra il contenuto della locazione di memoria 0007 e il contenuto della locazione di memoria 0009. Il risultato va posto nella 0000. Provate ricordandovi che il NOR è un OR negato.

Codice oggetto ovvero lista delle istruzioni che compongono il programma della "fombola elettronica".

Nella prima colonna formata da quattro cifre compaiono gli indirizzi, su ogni riga 16 dati contenuti in 16 locazioni di memoria successive.

0200	=A2	00	F8	18	E8	69	99	C9	00	D0	F8	8A	60	AA	A9	00
0210	=F8	18	69	01	CA	D0	FA	60	EA	EA	EA	EA	EA	A9	00	AA
0220	=95	00	E8	E0	99	D0	F9	60	EA							
0230	=20	1D	02	A9	3E	85	8F	A9	30	85	90	A9	77	85	91	A9
0240	=01	85	08	EA	EA	EA	EA	E6	0F	20	7E	FF	A5	0F	F8	18
0250	=69	08	C9	91	90	10	29	0F	C9	02	D0	04	A9	01	D0	06
0260	=C9	01	D0	02	A9	02	85	0F	20	00	02	AA	B5	10	F0	06
0270	=4C	A6	03	EA	EA	EA	A5	0F	85	0E	20	2D	FF	D0	04	85
0280	=0D	F0	C6	A5	0D	D0	C2	E6	0D	A9	99	8D	03	FD	20	57
0290	=FF	C9	15	D0	4B	E6	0C	A5	0C	C9	5B	F0	26	A5	0E	20
02A0	=E0	03	A5	0A	85	93	A5	0B	85	94	20	D0	03	EA	EA	EA
02B0	=EA	EA	EA	EA	85	09	A5	0E	20	00	02	AA	A5	0C	95	10
02C0	=4C	49	02	C6	0C	A9	71	85	8F	A9	30	85	90	A9	37	85
02D0	=91	A9	79	85	92	A9	00	85	93	85	94	85	09	4C	49	02
02E0	=C9	14	D0	18	A5	0C	20	0D	02	20	E0	03	A5	0A	85	8F
02F0	=A5	0B	85	90	A9	40	85	91	A9	79	85	92	4C	75	03	C9
0300	=11	D0	0E	A9	01	85	08	20	D0	03	A9	73	85	8F	4C	75
0310	=03	C9	12	D0	36	A5	08	A2	00	E8	E0	5B	F0	16	D5	10
0320	=D0	F7	20	D0	03	8A	20	0D	02	20	E0	03	20	42	03	E6
0330	=08	4C	75	03	A9	3E	85	8F	A9	40	85	90	85	91	85	92
0340	=D0	EF	A5	0A	85	8F	A5	0B	85	90	60	C9	13	D0	31	A5
0350	=09	D0	05	4C	8B	03	EA	EA	20	00	02	AA	B5	10	F0	0D
0360	=A9	10	85	92	A9	00	85	91	85	09	4C	49	02	A9	5C	D0
0370	=F1	EA	EA	EA	EA	A9	00	85	09	4C	49	02	EA	EA	EA	EA
0380	=C9	19	D0	03	4C	30	02	C9	10	B0	DF	85	0A	A5	09	0A
0390	=0A	0A	0A	05	0A	85	09	20	E0	03	20	D0	03	20	42	03
03A0	=4C	49	02	EA	EA	EA	20	EB	FE	D0	02	85	0D	A5	0C	C9
03B0	=5A	D0	03	4C	76	02	4C	49	02	EA						
03C0	=EA	EA														
03D0	=A9	00	85	8F	85	90	85	91	85	92	60	EA	EA	EA	EA	EA
03E0	=A8	29	0F	AA	BD	EA	FF	85	0B	98	4A	4A	4A	4A	AA	BD
03F0	=EA	FF	85	0A	60	99	09	69	A2	19	69	98	00	FE	6F	F5

I NUMERI NEGATIVI

Tutte le operazioni fatte fino a questo momento hanno preso in considerazione solo numeri positivi; le nostre operazioni infatti hanno utilizzato cifre dalla 00 alla FF (numeri di 8 bit) senza che ci fosse ombra di segno. Una domanda che viene spontanea è: come si possono rappresentare i numeri con il segno?

Il sistema che esamineremo per rispondere a questa domanda si chiama **COMPLEMENTO A DUE** ed è universalmente usato.

I numeri che prendiamo in considerazione sono sempre di otto bit, ma nell'ambito di questi otto bit introdurremo anche il segno. Come si fa?

È molto semplice, basta prendere atto di questa convenzione:

i numeri che hanno il 1° bit (quello più significativo) uguale a 1 sono **NEGATIVI**; i numeri che hanno il 1° bit uguale a 0 sono **POSITIVI**.

Per chiarire le idee si osservi la fig. 36.

Bisogna prestare attenzione a questo particolare: in un numero formato da otto bit, sette di questi rappresentano il numero vero e proprio in valore assoluto, mentre l'ottavo - il più significativo - rappresenta il segno.

Abbiamo allora che con 7 bit possiamo rappresentare $2^7 = 128$ numeri diversi; considerando ora sia quelli positivi che quelli negativi essi raddoppiano e diventano 256, ritroviamo cioè il valore $2^8 = 256$ che già conosciamo per i numeri da 8 bit.

Secondo la convenzione suddetta diciamo dunque che i numeri FF (1111 1111), 82 (1000 0010), A0 (1010 0000), e 95 (1001 0101) sono numeri **negativi** (il primo bit è a 1) mentre i numeri 00 (0000 0000), 14 (0001 0100), 7E (0111 1110) e 39 (0011 1001) sono numeri **positivi** (il primo bit è a 0).

Facciamo notare che in questa convenzione il numero 0 è considerato come numero positivo.

Il problema immediatamente successivo da risolvere è il cambiamento di segno di un numero positivo e viceversa.

Per esempio il nostro numero 1 sappiamo che in esadecimale si scrive 01; il nostro -1 allora come si scrive in esadecimale con numeri da 8 bit?

La regola generale dice che bisogna prendere il numero positivo, negarlo bit per bit, quindi sommare 1. Cioè in pratica riferendoci alla fig. 37 abbiamo che secondo questa regola:

$$\begin{aligned} +1 &= 01 \\ -1 &= FF \end{aligned}$$

Attenzione, per evitare di farvi confondere le idee vi rammentiamo che se consideriamo il numero FF in valore assoluto (come abbiamo fatto fino ad ora), esso è il numero più grande da 8 bit e vale 255_{10} ; nella convenzione di complemento a due però, nella quale il primo bit è quello di segno, FF è il numero negativo -1.

Sembrirebbe ora che col nostro microcomputer si possano sommare e/o sottrarre numeri non superiori a +127

e non inferiori a -128. Questo avviene solo se lavoriamo con numeri contenuti in una sola locazione di memoria (8 bit); naturalmente nella matematica che ci costruiremo i numeri potranno essere contenuti in più locazioni di memoria: per esempio tre successive.

Riprendiamo ora il discorso del numero positivo e negativo e proviamo ad eseguire la somma:

$$12 + (-1).$$

Eseguiamola prima sulla carta:

$$\begin{array}{r} 0001 \ 0010 \ + \\ 1111 \ 1111 \ = \\ \hline \end{array} \quad \begin{array}{l} (12) \\ (FF) \end{array}$$

$$\boxed{1} \ 0001 \ 0001 \quad (11)$$

↑ Carry

Come risultato della operazione otteniamo 11. Il Carry esce dalla operazione a 1, ma in questo caso non se ne tiene conto.

In questo modo abbiamo sottratto 1 al numero 12.

Eseguiamo ora la stessa operazione sull'AMICO 2000A con questo semplice programma:

0200	18	CLC	Clear del Carry
1	D8	CLD	Funzionamento esadecimale
2	A9	LDA #12	Carico 12 in accumulatore in maniera immediata
3	12		
4	69	ADC #FF	Sommo FF in maniera immediata
5	FF		
6	85	STA 00	Porto il risultato nella locazione di memoria 00
7	00		
8	4C	JMP MONITOR	Torno al monitor. Stop.
9	22		
A	FE		

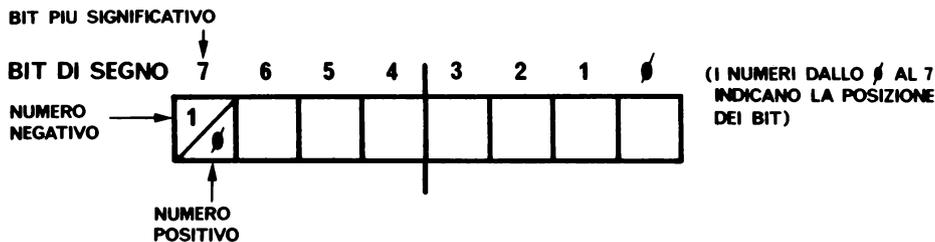


Fig. 36 - Segno di un numero di 8 bit (di cui sette costituiscono il valore assoluto del numero e uno, il più significativo, il segno).

Eseguendo il programma verrà immediatamente visualizzato il risultato.

Modifichiamo il programma per rendere più agevole il cambiamento dei dati:

```

0200 18 CLC
1 D8 CLD
2 A5 LDA 00          Carico il dato della locazione 00 in accumulatore
3 00
4 65 ADC 01          Sommo il contenuto della locazione 01
5 01
6 85 STA 00          Riporto il risultato nella locazione 00
7 00
8 4C JMP MONITOR
9 22
A FE

```

I dati vanno introdotti nelle locazioni 00 e 01. Scriviamo allora il programma, carichiamo il Program Counter per utilizzare il tasto REG (alle locazioni 00F6 e 00F7) in modo che richiami l'indirizzo 0200 poi premiamo i tasti nella sequenza

```

[RES] [DA] 12 [↑] FF
[REG] [RUN].

```

Il risultato (11) compare immediatamente nel display dati.

Per cambiare i dati di partenza basta introdurli alle locazioni 00 e 01, quindi far partire il programma.

A questo punto vi insegnamo un piccolo programma che permette di calcolare il numero negativo una volta dato il numero positivo. Attenzione! Se volete non guardate ciò che segue e provate a farlo voi; con un po' di pazienza dovreste

riuscire poichè avete tutte le cognizioni richieste.

Ad ogni modo il programma è quello riportato a fondo pagina.

In sostanza questo programma esegue la procedura di calcolo prima indicata: negazione del numero e somma con 1 trascurando il Carry.

Per usare il programma procediamo come al solito introducendo nella locazione di memoria 00 il numero del quale si vuole calcolare il negativo per ottenere nella stessa locazione 00 il risultato cercato.

Vi diamo alcuni numeri con il loro corrispondente negativo:

```

(p) (n)
00 00 (il negativo di 0 è ancora 0)
01 FF
0E F2
47 B9
7F 81

```

Notate che nell'ambito dei numeri da 8 bit non esiste un numero positivo che abbia come numero negativo 80.

```

0300 18 CLC
1 D8 CLD
2 A5 LDA 00          Prelevo il dato da trasformare dalla locazione 00
3 00
4 49 EOR #$FF       Nego il numero contenuto in accumulatore
5 FF
6 69 ADC #1          Sommo 1
7 01
8 85 STA 00          Riporto il dato nella locazione 00
9 00
A 4C JMP MONITOR. Stop.
B 22
C FE

```

Questo perchè i numeri positivi vanno, in decimale, da 0₁₀ a +127₁₀, quelli negativi da -1₁₀ a -128₁₀, in tutto 256 numeri (compreso lo zero).

Le particolarità da tener presenti sono dunque:

con 8 bit si possono rappresentare
128 numeri positivi 0 ÷ 127
128 numeri negativi -1 ÷ -128.

Il numero negativo corrispondente a 0 è ancora 0.

Il programma appena illustrato è ovviamente in grado di trasformare un numero negativo nel suo corrispondente positivo. Verificate i seguenti risultati:

```

(n) (p)
B9 47
C4 3C
98 68
8F 71

```

Sottrazione tramite i numeri negativi

A questo punto è evidente che la differenza fra due numeri si può sempre riportare alla somma fra due numeri.

Infatti $13 - 7A = 13 + (-7A) = 13 + + 86 = 99 = -67$.

Tenete sempre presente che si tratta di numeri esadecimali.

Verificate queste operazioni con i programmi appena scritti.

Può essere utile spesso poter disporre di un mezzo che ci permetta, senza fare fatica, di trasformare un numero negativo o positivo del sistema decimale in quello corrispondente nel sistema complemento a due.

Per far questo abbiamo scritto un programma di cui troverete a fine capitolo la lista delle istruzioni (codice oggetto).

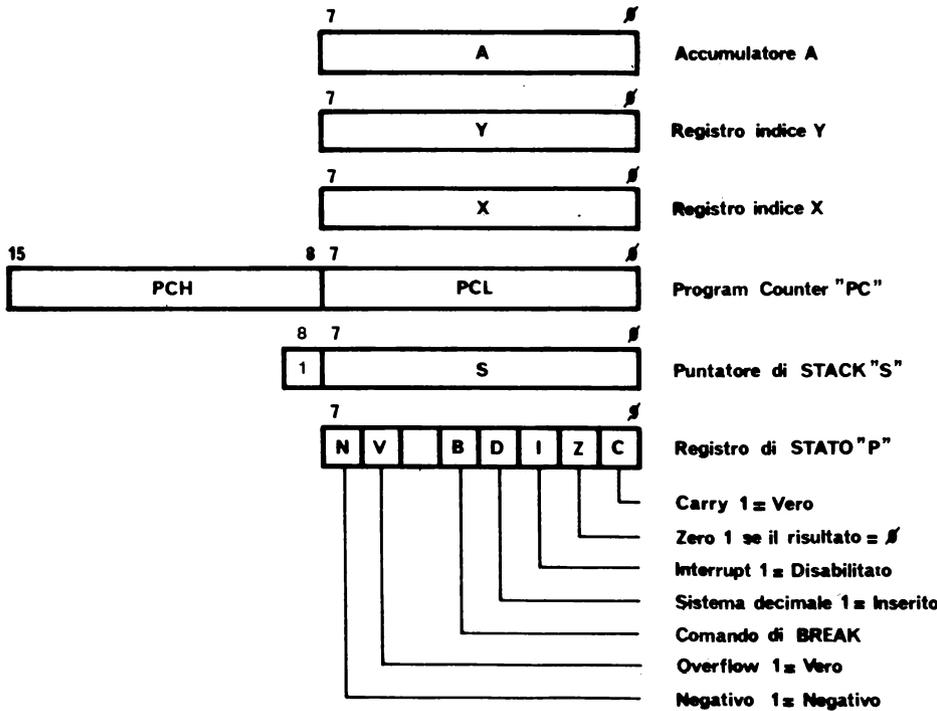
Il programma parte dalla locazione 0200 e arriva alla 02D3. Vi consigliamo di registrarlo su cassetta in modo da poterlo utilizzare ogni volta che ne avete bisogno. Attenzione, però, questo programma non è rilocabile, cioè deve essere sempre scritto a partire dalla locazione di memoria 0200.

Per utilizzarlo, una volta scritto, è molto semplice: premete i tasti [AD] 0230

```
[RUN].
```

Premete il tasto [C] e sul display indirizzi compariranno tre zeri. Scrivete ora il numero che intendete trasformare, premete il tasto [F] se volete farlo diventare negativo (e viceversa): contemporaneamente sul display dati compare il risultato. Da notare che questo programma è stato scritto per evitare che commettiate errori: tutti i numeri fuori della portata -128 ÷ +127 non vengono accettati.

SCHEMATIZZAZIONE DELLA CPU 6502



tati. Oltre a questo programma riportiamo nella tabella 1 una tavola di conversioni da numeri negativi (sopra) e positivi (sotto) in sistema decimale a numeri negativi e positivi in sistema complemento a due (e viceversa).

Usare queste tavole è molto semplice, facciamo un esempio.

Vogliamo sapere a cosa corrisponde il numero -77 (in decimale) nel sistema complemento a due.

Cerchiamo 77 all'interno della tavola in alto (numeri negativi), sulla riga del 77 a sinistra troviamo il numero B sulla colonna "1^a cifra"; sulla colonna del 77 troviamo in alto il numero 3 sulla riga "2^a cifra": la traduzione sarà quindi B3. Procedete all'inverso per trasformare un numero da complemento a due a decimale.

L'Overflow

Analizziamo ora un'altro bit dello status, quello di *overflow*. Il concetto di overflow nasce dall'introduzione dei numeri con segno appena fatta. Infatti abbiamo sempre detto che la somma di due numeri può generare un riporto (Carry) se il risultato eccede la dimensione degli otto bit che abbiamo a disposizione.

Nel caso di numeri col segno però abbiamo solo 7 bit per il numero, dato che il bit più significativo è stato utilizzato per il segno. Può succedere che sommando due numeri negativi si ottenga un numero positivo o sommando due numeri positivi si ottenga un numero negativo; questo perchè il risultato eccede

0	0	0	0	0	0	0	1	+1	Il numero precedente viene negato bit per bit, cioè dove c'era uno "zero", si scrive un "uno" e dove c'era un "uno", si scrive uno "zero". Si somma 1 al risultato della negazione. Risultato FF
1	1	1	1	1	1	1	0	(+1)	
+									
0	0	0	0	0	0	0	1	= +1	
1	1	1	1	1	1	1	1		

Fig. 37 - Trasformazione del numero 01 positivo in negativo (-1) nel sistema complemento a due: il risultato è FF.

i 7 bit andando ad influenzare l'ottavo bit, cioè quello di segno.

Chiariamo subito le idee con qualche esempio:

$$\begin{aligned} 1111\ 0111 &+ (F7 = -09) \\ 1111\ 1110 &= (FE = -02) \end{aligned}$$

$$\boxed{1} \quad 1111\ 0101 \quad (F5 = -0B)$$

Carry

Nel risultato il bit di segno (il primo a sinistra) è 1 e il numero è quindi negativo, infatti esso non eccede il limite dei 7 bit.

Vediamo ora la somma di due numeri negativi più grandi:

$$\begin{aligned} 1001\ 0110 &+ (96 = -6A) \\ 1000\ 1111 &= (8F = -71) \end{aligned}$$

$$\boxed{1} \quad 0010\ 0101 \quad (25 = +25)$$

Carry

Il risultato è un numero che ha cambiato segno. Il bit di segno infatti è 0, si è ottenuto cioè un numero positivo sommando due numeri negativi!

Operando in matematica con complemento a due bisogna tener conto di questi cambiamenti di segno per evitare errori. Si può dimostrare che, sviluppando una funzione di questo genere:

$V = A_7 \cdot B_7 \cdot R_7 + \bar{A}_7 \cdot \bar{B}_7 \cdot R_7$ in cui: A_7 è il bit più significativo del primo addendo; B_7 è il bit più significativo del secondo addendo; R_7 è il bit più significativo del risultato. La barretta sopra il segno del bit sta per negato; \cdot è il simbolo della operazione di AND; $\bar{}$ è il simbolo della operazione di OR.

- se $V = 0$ non ci sono stati errori nella operazione.

- se $V = 1$ c'è stato un cambiamento di segno nell'operazione.

Verifichiamo V per i due esempi appena fatti.

1° esempio: $A_7 = 1, B_7 = 1, R_7 = 1$
 $V = 1 \cdot 1 \cdot 1 + 0 \cdot 0 \cdot 1 = 1$

Il risultato è dunque esatto.

2° esempio: $A_7 = 1, B_7 = 1, R_7 = 0$
 $V = 1 \cdot 1 \cdot 0 + 0 \cdot 0 \cdot 0 = 0$

C'è un cambiamento di segno nel risultato. V si chiama "bit di overflow" e viene calcolato automaticamente dalla CPU ad ogni operazione che lo condiziona.

In generale, in forma molto semplice possiamo dire che la CPU pone $V = 1$ se il risultato dell'operazione eseguita esce dai limiti permessi per un numero con segno da 8 bit (7 di valore assoluto e 1 di segno) cioè se è al di fuori del campo $-128 \div +127$.

L'istruzione SBC

Questa istruzione esegue la differenza (sempre operando nella convenzione dei numeri negativi ovvero del complemento a due) fra il contenuto dell'accumulatore, il contenuto di una locazione di

memoria e il negato del bit di Carry. L'operazione in forma simbolica si scrive:

$$A - M - \bar{C} \rightarrow A$$

Cioè la differenza fra l'accumulatore, il dato in memoria, il negato del Carry, va a finire nell'accumulatore. Questa istruzione può operare sia in sistema decimale che esadecimale, a seconda se la CPU sta lavorando in maniera decimale o meno. \bar{C} (negato del bit di Carry) si chiama in inglese *borrow* ed è il riporto dell'operazione. La presenza del \bar{C} è un artificio del microprocessore per poter calcolare in codice binario complemento a due.

Tenete presente una regola molto importante: quando eseguite delle somme dovete sempre mettere, prima dell'esecuzione della operazione di somma, il $C = 0$.

Quando fate delle sottrazioni tramite l'istruzione SBC dovete sempre mettere, prima dell'esecuzione della differenza, il $C = 1$.

Ci sono ancora delle particolarità molto importanti da tener presenti: l'istruzione SBC mette a 1 il bit di Carry se il risultato dell'operazione è maggiore di 0; mette a 0 il bit di Carry se il risultato dell'operazione è minore di 0.

Il bit di overflow viene messo a 1 se il risultato dell'operazione non è compreso fra +127 e -128, altrimenti viene messo a 0.

Eseguiamo ora l'operazione di calcolo di numero negativo già descritta in precedenza utilizzando questa volta l'istruzione SBC. (Posizionatevi su una locazione di memoria a piacere per cominciare a scrivere il programma).

```
D8 CLD
38 SEC (notate che mettiamo il Carry a 1)
A9 LDA #00 Carico il numero 0 nell'accumulatore 00
E5 SBC $00 Sottraggo il dato da complementare 00
85 STA $00 Metto il risultato in locazione 00
4C JMP MONITOR, Stop
22
FE
```

Abbiamo fatto un programma che calcola $00 - \text{NUMERO} = -\text{NUMERO}$

Il numero (dato di ingresso) viene prelevato all'inizio dell'operazione dalla locazione di memoria 00 e alla fine viene ancora depositato nella locazione 00 con il segno cambiato.

Verificate con questo semplice programma le operazioni di calcolo precedentemente eseguite verificandone la corrispondenza.

Status (P)

I bit che fanno parte del registro di status sono riportati tutti nella tabella delle istruzioni del 6502; abbiamo già

visto alcuni di questi bit, comunque di seguito li riesaminiamo tutti insieme.

C = Carry

- Bit di Carry. È il riporto in varie operazioni, per esempio in quella di somma.

Z = Zero

- Questo bit dello stato viene messo a 1 se il risultato dell'ultima operazione eseguita è uno zero. Molte istruzioni modificano questo bit. Per esempio quella di Somma, Load, quelle logiche di AND, OR, EOR.

I = Richiesta di interrupt

- Se questo bit è a zero risulta abilitato l'interrupt in ingresso alla CPU. In un capitolo dedicato esamineremo questo aspetto dell'hardware.

D = Modo di funzionamento decimale

- Se questo bit è a uno la CPU sta operando in matematica decimale. Questo bit può essere posizionato a zero o a uno tramite due apposite istruzioni (CLD - SED).

B = Comando di BRK

- Questo comando sarà esaminato in seguito.

V = Bit di overflow

- È il bit esaminato nelle operazioni differenza. Molte operazioni lo condizionano.

N = Bit di negativo

- Viene messo a uno se il risultato dell'ultima operazione eseguita è negativo. In pratica è il bit più significativo del risultato dell'ultima operazione eseguita. Viene condizionato da molte operazioni sia aritmetiche che logiche.

I bit di status I e B sono legati essenzialmente alla circuiteria esterna al microelaboratore e non li esamineremo finché non avremo a che fare con questi dispositivi.

In questo momento i bit di status che maggiormente ci interessano sono quattro, e cioè: C - Z - V - N.

Questi bit in generale vengono condizionati automaticamente dalla istruzione che stiamo eseguendo e ci permettono, eseguita la istruzione, di prendere delle decisioni in base al risultato della operazione eseguita.

Sono proprio queste decisioni che rendono l'elaboratore "intelligente".

Le istruzioni che prendono una decisione in base ai risultati delle operazioni, sono le istruzioni di BRANCH (salto condizionato).

È una categoria di istruzioni (riportate a fondo pagina) molto importante, che permette di eseguire o non eseguire un salto da un punto all'altro del programma in base al risultato di una operazione precedente.

Vediamo ora quale è il significato di alcune di queste istruzioni indipendentemente dai bit di status, ovvero come se non dovessimo tener conto del valore di essi.

Ci spieghiamo meglio.

Spesso le istruzioni di Branch vengono utilizzate subito dopo le istruzioni di comparazione ("compare" in inglese) come CMP, CPX (che vedremo fra poco), CPY; di decremento come DEC, DEX (che vedremo fra poco), DEY o di incremento come INC, INX (che vedremo fra poco) e INY.

Tutte queste istruzioni eseguono una certa operazione: nella comparazione, ad esempio, la CPU esegue una sottrazione fra i due dati da comparare dando un certo risultato che influenza i bit di Status interessati (si veda la tabella riassuntiva delle istruzioni del 6502). In pratica quando i due dati che vengono comparati sono uguali il risultato di questa sottrazione è 0, ma noi non lo vediamo. Questo risultato però mette a 1 il bit Z dello Status.

Premesso ciò vediamo il significato per esteso delle seguenti istruzioni:

BEQ Salta se il risultato (della operazione precedente) è = 0

BMI Salta se il risultato (della operazione precedente) è < 0 (negativo)

BNE Salta se il risultato (della operazione precedente) è ≠ 0

BPL Salta se il risultato (della operazione precedente) è ≥ 0 (positivo)

Ricordiamo che 0 nel sistema di numerazione complemento a due è un numero positivo.

Visto questo, se a una comparazione facciamo seguire la istruzione BEQ otteniamo che verrà eseguito un salto quando i due dati messi a confronto saranno uguali (la differenza fa 0, quindi va a 1 il bit di Status Z, quindi BEQ esegue il salto).

Queste istruzioni sono:

BCC il salto viene eseguito se il bit di Carry	è = 0: codice op.	90
BCS il salto viene eseguito se il bit di Carry	è = 1: codice op.	B0
BEQ il salto viene eseguito se il bit di Zero	è = 1: codice op.	F0
BMI il salto viene eseguito se il bit di Negativo	è = 1: codice op.	30
BNE il salto viene eseguito se il bit di Zero	è = 0: codice op.	D0
BPL il salto viene eseguito se il bit di Negativo	è = 0: codice op.	10
BVC il salto viene eseguito se il bit di Overflow	è = 0: codice op.	50
BVS il salto viene eseguito se il bit di Overflow	è = 1: codice op.	70

Per riassumere diremo allora che se qualsiasi operazione eseguita immediatamente prima di BEQ dà come risultato \emptyset (potrà essere una somma, una differenza, una operazione logica tipo AND, OR, ecc.), verrà eseguito il salto.

Lo stesso ragionamento vale per le altre istruzioni: BMI ad esempio eseguirà un salto se il risultato di una operazione immediatamente precedente è negativo (cioè se il bit più significativo del risultato è = 1, cioè quando il bit di Status N = 1).

Una ultima nota sulla istruzione BVC: questa può essere usata per correggere un errore in un'operazione di sottrazione.

Facciamo notare che tutte le istruzioni di salto (Branch) sono formate sempre da due byte di cui il primo è il codice operativo (es. 90), il secondo rappresenta l'entità del salto, ovvero il numero di byte che si devono saltare nell'esecuzione del programma. Quest'ultimo numero è espresso in complemento a due, cioè il salto rispetto alla posizione in cui il Program Counter è in quel momento, può essere negativo (numero di byte indietro) o positivo (numero di byte in avanti). A ben comprendere e saper utilizzare le diverse istruzioni di Branch si imparerà man mano che verranno presentati esempi di programma. L'importante per ora è che impariate a memoria le condizioni che provocano il salto per ogni istruzione di Branch.

Vediamo subito di utilizzare alcune di queste istruzioni per costruire un programma che analizza un numero n.

Se $n \geq 0$ il microelaboratore dovrà scrivere $\emptyset\emptyset$ nella locazione di memoria $\emptyset\emptyset\emptyset$;

se $n < 0$ si scrive FF nella locazione di memoria $\emptyset\emptyset\emptyset$.

Questo programma può essere scritto sotto forma di svolgimento logico o diagramma di flusso come rappresentato in Fig. 38.

Vediamo in questa figura due simboli nuovi, il rettangolo e il rombo, usati rispettivamente per indicare una operazione da eseguire e una domanda che ci si pone.

Scriviamo ora il programma proposto. Il numero di cui vogliamo analizzare il segno sia contenuto nella locazione di memoria $\emptyset\emptyset\emptyset$.

$\emptyset32\emptyset$	A5	LDA $\emptyset1$
1	$\emptyset1$	
2	10	BPL AVA1
3	$\emptyset7$	
4	A9	LDA $\#\$FF$
5	FF	
6	IND1	STA $\emptyset\emptyset$
7	$\emptyset\emptyset$	
8	4C	JMP Monitor
9	22	
A	FE	
B	AVA1	A9 LDA $\#\emptyset\emptyset$
C	$\emptyset\emptyset$	
D	F0	BEQ IND1
E	F7	

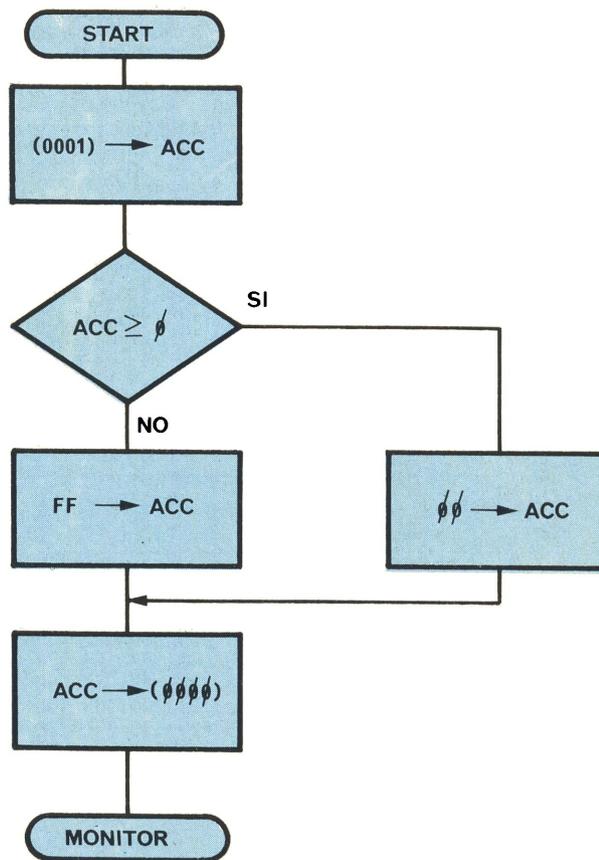


Fig. 38 - Diagramma di flusso (o flow-chart) del programma per il riconoscimento di numeri negativi e positivi.

Commentiamo di seguito per esteso il programma che è stato appena scritto.

1^a istruzione: LDA $\emptyset1$ - Si carica il contenuto della locazione di memoria $\emptyset\emptyset\emptyset1$ (pagina zero) nell'accumulatore. Automaticamente la CPU influenza i bit di stato Z e N, per cui nella istruzione successiva possiamo andarci a controllare uno (in particolare N) per decidere se eseguire il salto condizionato.

2^a istruzione: BPL AVA1 - Abbiamo "etichettato" questa istruzione con un nome di fantasia (AVA1). Questa procedura è molto comoda nella stesura dei programmi perché serve ad indicare dove dobbiamo eseguire il salto. Con l'istruzione BPL la CPU va a vedere se il bit di negativo (N) è = \emptyset . Notiamo che è l'istruzione che precede (la prima) che ha condizionato il bit N. Se N = \emptyset (contenuto dell'accumulatore ≥ 0) il salto viene eseguito, diversamente si prosegue nella esecuzione del programma con la istruzione immediatamente successiva. 3^a - 4^a - 5^a - 6^a istruzione - Nessun commento salvo che alla 4^a è stato attribuito il nome IND 1.

7^a istruzione: BEQ IND1 - Notiamo subito che la 6^a istruzione ha caricato in maniera immediata zero nell'accumulatore. Quindi il risultato della 6^a istruzione è sempre e comunque uno zero; allora dopo questa istruzione avrò sempre e

comunque Z = 1. È chiaro allora che l'istruzione BEQ in questa posizione equivale ad un salto incondizionato, cioè il salto viene sempre e comunque eseguito. (LDA $\#\emptyset\emptyset$ infatti genera sempre Z = 1, BEQ va a controllare che, se è uguale a uno, salta: quindi esegue sempre il salto in questo caso).

Analizziamo ora l'entità di questi salti. Nel microprocessore 6502 il Branch è sempre relativo alla posizione del Program Counter. Ciò significa che quando si esegue il Branch, il numero che segue il codice operativo del Branch viene sommato al valore del Program Counter in quel momento, quindi l'esecuzione del programma continua dal nuovo Program Counter generato.

Nel nostro esempio la CPU preleva dalla memoria alle locazioni $\emptyset322$ - $\emptyset323$ puntate dal Program Counter l'istruzione 10, $\emptyset7$ (BPL AVA1) posizionandosi subito dopo alla locazione successiva $\emptyset324$ e preparandosi, se non succede niente, ad eseguire la istruzione contenuta in questa locazione. Prima della esecuzione della BPL ci troviamo quindi nella seguente condizione:

PC = $\emptyset324$.

Se il salto deve essere eseguito bisogna posizionarsi sulla locazione $\emptyset32B$, ovvero il PC deve essere incremento di $\emptyset32B - \emptyset324 = \emptyset7$. Notiamo subito che $\emptyset7$ è il 2^o

Tabella 15 - Tavole di conversione da numeri in codice decimale a numeri in complemento a due e viceversa.

NUMERI NEGATIVI

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

NUMERI POSITIVI

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

byte dell'istruzione di branch. Esso rappresenta, come si suol dire, il "displacement" del Program Counter ed è relativo al P.C. medesimo, è cioè indipendente dalla locazione di memoria nella quale abbiamo cominciato a caricare il programma (per esempio il programma può girare anche se caricato alla locazione di memoria 0210 invece che alla 0320). Lo stesso conto per spostare il Program Counter può essere fatto nel salto negativo (ultima istruzione). In questo caso bisogna saltare dalla locazione 032F, dove si trova posizionato il P.C., alla 0326. Vale allora la relazione "displacement" = 0326 - 032F = -(032F - 0326) = -09 = F7, che è appunto il 2° byte della istruzione BEQ IND1. In pratica quando scriviamo un programma lasciamo uno spazio bianco dopo le istruzioni di Branch: il valore del salto viene definito a conclusione del programma.

Per calcolare i numeri positivi o negativi che rappresentano le entità del salto si può utilizzare con vantaggio un comodo programma (presente del Monitor) che fino ad ora non abbiamo adoperato, ma che è stato previsto proprio per questo scopo.

Questo programma parte dalla locazione di memoria FF97. Premiamo allora i tasti **AD** **FF97** **RUN**.

A questo punto riprendiamo per esempio il programma che abbiamo precedentemente analizzato: il primo salto (BPL AVA 1) doveva essere fatto dalla locazione 0322 (con questo metodo non dobbiamo tener conto della posizione del P.C. poiché ci pensa il programma) alla 032B.

Ora basterà semplicemente battere sulla tastiera nella successione:

- 1) la parte bassa (2° byte) dell'indirizzo di partenza: 22.
- 2) la parte bassa dell'indirizzo a cui si vuole saltare: 2B.

Sul display indirizzi comparirà 222B e sul display dati il numero 07, che è il valore relativo al salto nel sistema complemento a due.

Stessa procedura per il salto all'indietro: battete sulla tastiera 2D (l'istruzione BEQ IND1 è infatti all'indirizzo 032B poi 26 (infatti bisogna saltare alla locazione 0326) e otterrete il risultato cercato F7.

Il tutto, come si è visto, è molto semplice: ogni volta che serve basta richiamare questo programma che è sempre lì, registrato permanentemente nella PROM del Monitor.

Si fa notare infine che i salti in avanti e indietro possono essere fatti nel campo dei numeri positivi e negativi del sistema complemento a due; per salti più lunghi si associa alla istruzione di Branch quella di JMP (salto incondizionato) che consente di andare in qualsiasi punto dell'intera memoria.

In particolare osserviamo che il programma per il calcolo del salto che abbiamo descritto permette di eseguire salti in avanti fino a +129 posizioni e indietro fino a -126 posizioni (127+2=129 e -128+2=V = -126 poiché il programma tiene conto delle due posizioni più avanti del P.C.).

Index X

In questo paragrafo introduciamo un nuovo registro interno alla CPU: *il registro indice X*.

Fino ad ora della CPU abbiamo visto tre soli registri: l'Accumulatore A, il Program Counter P.C., e lo Status P. Vedremo che c'è ne sono ancora tre; cominciamo ad esaminare il primo di questi.

Il registro indice X è un registro di 8 bit sul quale è possibile eseguire dei calcoli, o, cosa più importante, indirizzare la memoria con metodi nuovi rispetto a quelli visti fino ad ora.

Analizziamo prima questo ultimo aspetto considerando il metodo di indirizzamento tramite il registro X.

Premettiamo che le istruzioni che usano questo tipo di indirizzamento sono sempre formate da due byte, di cui il primo è come sempre, il codice operativo, il secondo è un byte così detto di spostamento (in inglese *displacement*).

Indirizzamento in pagina zero indicizzato

Per i codici operativi di questa istruzione si faccia riferimento anche alla tabella riassuntiva di tutte le istruzioni del 6502 pubblicata in questo capitolo.

Usando il sistema di indirizzamento indicizzato, l'indirizzo della locazione di memoria sulla quale si vuole operare, viene calcolato dalla CPU sommando il contenuto del secondo byte dell'istruzione con il contenuto del registro indice, senza tener conto del Carry. Il risultato di questo calcolo (che avviene sempre in codice esadecimale) è un numero da 8 bit che rappresenta la parte bassa dell'indirizzo essendo la parte alta 00 perché si opera in pagina zero.

Per chiarire meglio le idee vediamo subito un esempio pratico.

Ammettiamo che sia X = 3B (registro indice X = 3B). Se si esegue la istruzione

B5 LDA 02, X
02

la CPU calcola l'indirizzo della locazione di memoria il cui contenuto viene portato in accumulatore eseguendo la operazione

$$3B + (\text{contenuto dell'indice X}) \\ 02 = (2^\circ \text{ byte dell'istruzione}) \\ 3D \quad (\text{indirizzo della locazione di memoria interessata})$$

N.B. B5 è il codice operativo della istruzione LDA con il sistema di indirizzamento in pagina zero indicizzato.

La domanda che sorge ora è l'uso che si può fare di questo sistema.

Innanzitutto premettiamo che esiste un certo numero di istruzioni che operano su X che sono:

CPX: Confronto di X con un numero.
DEX: Decremento di uno il valore di X ($X = X - 1$).

INX: Incremento di uno il valore di X ($X = X + 1$).

LDX: Carico X con un certo valore immediato o no.

STX: porto il valore di X in una certa locazione di memoria.

TAX, TXA: porto il contenuto dell'accumulatore in X e viceversa.

Detto questo vediamo di risolvere un semplice problema che utilizzi queste istruzioni: scrivere un programma che porti a zero tutte le locazioni di memoria comprese fra 0020 e 002F. Si noti che questo tipo di operazione viene normalmente fatto nei programmi di inizializzazione di un elaboratore all'atto della accensione.

Il diagramma di flusso del programma può essere quello rappresentato in fig. 39.

Scriviamo allora il seguente programma:

0020	A2	LDX #00	Carico X con 0 in modo immediato
1	00		
2	8A	TXA	Copio il contenuto di X in Accumulatore
3 Loop	95	STA 20,X	Memorizzo il contenuto dell'Acc. nella locazione di memoria X + 20
4	20		
5	E8	INX	Incremento X ($X = X + 1$)
6	E0	CPX #S10	Confronto X con 10, cioè $X = 10$?
7	10		
8	D0	BNE LOOP	Se X non è uguale a 10 torno ad eseguire la istruzione STA
9	F9		
A	4C	JMP Monitor	Stop
B	22		
C	FE		

Scrivete il programma nell'AMICO 2000A, fatelo girare e controllate che tutte le locazioni di memoria dalla 0020 alla 002F contengano ora 00.

Il programma può sembrare un tantino complicato, vediamo allora di descriverlo in modo discorsivo in modo da comprendere ogni passaggio.

Per prima cosa si è caricato nell'index X il numero 00 (dato immediato), quindi, siccome ci interessa che anche nell'accumulatore ci sia 00 si copia il contenuto di X in accumulatore.

Questo è stato fatto per risparmiare un byte di programmazione (l'alternativa era LDA 00, cioè A900 = 2 byte). A questo punto inizia un "loop", che viene identificato con una etichetta sulla prima istruzione, che è STA e cioè il trasferimento del contenuto dell'accumulatore nella locazione di memoria il cui indirizzo è 20 + X. Al primo passaggio di questo giro (loop) questa locazione sarà 0020 + 00 = 0020 e cioè la prima locazione di memoria che deve essere portata a zero. A questo si incrementa il contenuto del registro indice X di uno. Si noti che con questa istruzione (INX) il contenuto di X viene incrementato nel

sistema di numerazione esadecimale anche se la macchina è in funzionamento decimale. Dato che devo portare a zero 16 locazioni di memoria mi chiedo se $X = 10_{16}$ (questa uguaglianza verrà verificata al 17° passaggio, infatti $10_{16} = 17_{10}$) e cioè se ho finito il mio lavoro (ad ogni loop X si incrementa di uno perché si passa attraverso l'istruzione INX). La istruzione seguente è quella di Branch e viene eseguita se X non è uguale a 10. Ricordiamo che F9 significa -7, la CPU cioè riparte da 7 byte indietro cioè dalla prima istruzione del loop.

Notiamo che con questo programma si è eseguita una operazione di STA per 16 volte successive, sempre in posizioni diverse e con pochissime istruzioni. Abbiamo fatto un loop, come si dice in inglese, incrementando ad ogni giro l'indirizzo di memorizzazione del dato, cosa resa possibile dal sistema di indirizzamento indicizzato. Ripetiamo ancora una volta che stiamo operando con locazioni di memoria site in pagina zero.

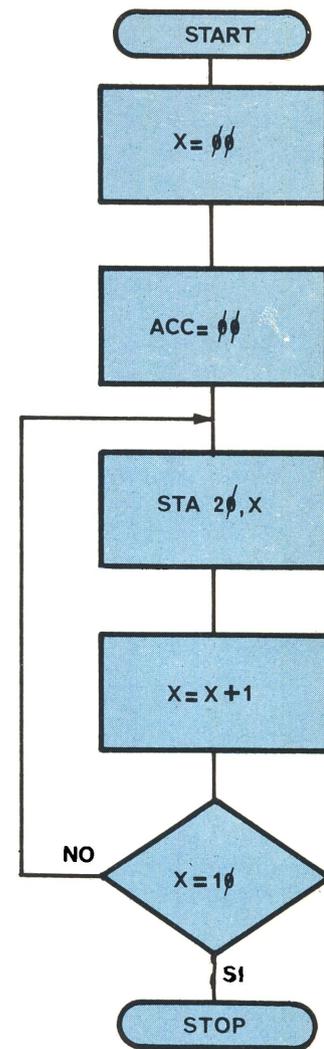


Fig. 39 - Diagramma di flusso per l'azzeramento di una zona di memoria.

Lo stesso programma può essere scritto almeno in altre due maniere diverse risparmiando qualche istruzione, provateci.

Il Folw-Chart

Le operazioni logiche che si eseguono nella stesura di un programma possono venir rappresentate tramite un sistema grafico detto dei diagrammi di flusso o flow-chart. Questo sistema è molto utile per rappresentazioni visivamente immediate e comprensibili di un programma e utilizza dei simboli grafici come quelli rappresentati in fig. 40

Alcuni di questi simboli sono già stati visti; di essi sono particolarmente interessanti i blocchi 2 e 3. Il 2 indica una operazione che bisogna eseguire che in generale può essere indicata come aritmetica o di movimento dati; questo blocco ha un ingresso e una uscita.

Il blocco 3 è quello decisionale; sta ad indicare una domanda che ci si pone,

domanda che ha sempre due risposte, SI e NO. Questo blocco ha dunque un ingresso e due uscite, una uscita per il SI e una per il NO.

Il blocco 1 caratterizza le operazioni di ingresso e di uscita dall'elaboratore. Ad esempio lo si usa per indicare il prelevamento di un dato da una tastiera. I blocchi 4 e 5 verranno descritti in una altra sede. L'incrocio indicato al punto 6 è chiaro: esso indica un punto di rientro su un altro flusso, quando, per esempio si effettua un rientro da un test (blocco 3). Il simbolo al blocco 7 indica l'inizio o la fine di un programma.

Generalmente quando si affronta un problema di programmazione per prima cosa si stende una sequenza logica di operazioni che risolvono il problema usando i diagrammi di flusso. Questi diagrammi sono immediatamente comprensibili e di facile correzione. Il vantaggio che offrono è che la stesura del diagramma è indipendente da come verrà scritto il programma effettivo; per

esempio è indipendente dal particolare set di istruzioni della CPU che poi eseguirà il programma.

Una volta verificata la correttezza delle operazioni logiche descritte nel diagramma di flusso si può cominciare a scrivere il programma stesso nel linguaggio della macchina che si vuole utilizzare. Il primo passo è sempre il più importante perché è quello che risolve effettivamente il problema. Il secondo è solo la fase realizzativa del tutto. Se infatti supponiamo di avere un programma scritto in linguaggio Assembler per un certo microprocessore diverso dal nostro, ci accorgeremo subito che è estremamente difficoltoso riscriverlo nel linguaggio del nostro computer. Se però abbiamo il flow-chart relativo a quel programma riscriverlo sarà una cosa molto semplice.

Le istruzioni del 6502

In questo capitolo è riportata una tabella che riassume tutte le istruzioni del microprocessore 6502 utilizzato nell'AMICO 2000. Di tutte le istruzioni riportate molte vi sono ormai note. Oltre che per ricordare i vari codici operativi, la tabella fornisce altre importanti indicazioni.

Vediamo come si legge.

Nella prima riga in alto sono riportati i vari sistemi di indirizzamento, alcuni dei quali sono stati già analizzati.

Nella seconda riga troviamo, dopo la rappresentazione mnemonica della istruzione e l'operazione ad essa associata, tre simboli di cui diamo il significato: OP sta per Codice Operativo N è il numero di byte che formano la istruzione:

* rappresenta il numero dei cicli macchina (Nel caso dell'AMICO 2000 il numero dei microsecondi impiegati per eseguire l'istruzione).

Nell'ultima colonna di questa stessa riga sono riportati i nomi dei bit dello Status che abbiamo già analizzato. Se l'istruzione influenza qualcuno di questi bit, in corrispondenza si vede un segno v, diversamente appare un trattino -.

Per meglio comprendere facciamo un esempio pratico.

Si voglia tradurre l'istruzione LDA *\$05.

Dalla tabella vediamo che si tratta di una istruzione di 2 byte, che viene eseguita in 2 μs, il cui codice (per lo indirizzamento immediato) è A9 e che influenza i bit N e Z.

La traduzione di questa istruzione in linguaggio macchina è quindi semplice: A9 05.

I simboli che compaiono nella colonna OPERAZIONE verranno spiegati via via che verrà analizzata l'istruzione.

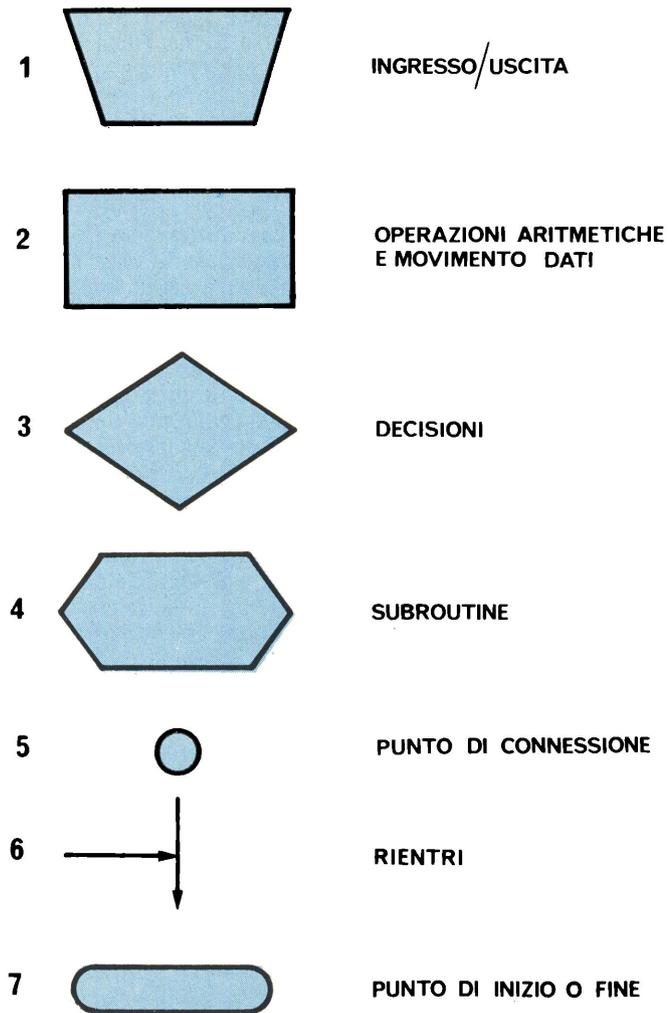


Fig. 40 - Simboli grafici usati nella costruzione di un programma per mezzo del flow-chart.

Esercizi

Questa volta vi faremo fare un po' di esercizi per abituarvi a prendere confidenza con le istruzioni fino ad ora imparate.

Per questi esercizi non vi daremo qui la soluzione, ma la pubblicheremo alla fine del prossimo capitolo.

1° esercizio - Scrivere un programma che esegue la somma di due numeri ciascuno da due byte (le somme che abbiamo fatto fino ad ora sono sempre state fra due numeri di 1 byte ciascuno) e positivi. Estendere poi il programma a numeri di più byte.

Per aiutarvi ricordiamo che se la somma di due numeri ha un riporto questo va automaticamente nel Carry. Se ad esempio scriviamo $93 + 75$, il risultato è di 1 68 cioè 68 e 1 nel Carry. Se ora sommiamo due numeri formati da quattro cifre come $1393 + 7275$ potremmo fare come segue:

13 93+
72 75=

scomponiamo

```

  93+
 75=
 1 68
   e
  13+
  72+
  1 =
   86

```

il risultato finale è 8668.

Utilizzare lo stesso metodo per scrivere il programma tenendo presente ancora una volta che il Carry è automatico e che in quattro successive locazioni di memoria potreste mettere rispettivamente il 1° e il 2° byte del primo numero e il 1° e il 2° byte del secondo numero; in altre due locazioni di memoria successive potreste mettere il 1° e il 2° byte del risultato.

Sviluppando il programma per la somma di numeri di più byte vi facciamo notare che un numero di 4 byte arriva a 4 miliardi!

Tabella riassuntiva delle istruzioni del microprocessore 6502

MINEMOICO	ISTRUZIONE	OPERAZIONE	IMMEDIATE		ABSOLUTE		ZERO PAGE		ACCUM.		IMPLIED		(IND.) X		(IND.) Y		Z, PAGE, X		ABS, X		ABS, Y		RELATIVE		INDIRECT		Z, PAGE, Y		CONDITION CODES													
			OP	N	OP	N	OP	N	OP	N	OP	N	OP	N	OP	N	OP	N	OP	N	OP	N	OP	N	OP	N	OP	N	OP	N	N	Z	C	I	D	V						
A D C	A + M + C → A	(4) (1)	69	2	2	6D	4	3	65	3	2																								✓	✓	✓	-	-	✓		
A N D	A ∧ M → A	(1)	29	2	2	2D	4	3	25	3	2																									✓	✓	-	-	-		
A S L						0E	6	3	06	5	2	0A	2	1																					✓	✓	✓	-	-	-		
B C C	BRANCH 'ON C=0	(2)																																								
B C S	BRANCH ON C=1	(2)																																								
B E Q	BRANCH ON Z=1	(2)																																								
B I T	A ∧ M					2C	4	3	24	3	2																										M7	✓	-	-	-	M6
B M I	BRANCH ON N=1	(2)																																								
B N E	BRANCH ON Z=0	(2)																																								
B P L	BRANCH ON N=0	(2)																																								
B R K												00	7	1																												
B V C	BRANCH ON V=0	(2)																																								
B V S	BRANCH ON V=1	(2)																																								
C L C	0 → C											18	2	1																												
C L D	0 → D											08	2	1																												
C L I	0 → I											58	2	1																												
C L V	0 → V											B8	2	1																												
C M P	A - M	(1)	C9	2	2	CD	4	3	C5	3	2																															
C P X	X - M		E0	2	2	EC	4	3	E4	3	2																															
C P Y	Y - M		C0	2	2	CC	4	3	C4	3	2																															
D E C	M - 1 → M					CE	6	3	C6	5	2																															
D E X	X - 1 → X											CA	2	1																												
D E Y	Y - 1 → Y											88	2	1																												
E O R	A ∨ M → A	(1)	49	2	2	4D	4	3	45	3	2																															
I N C	M + 1 → M					EE	6	3	E6	5	2																															
I N X	X + 1 → X											E8	2	1																												
I N Y	Y + 1 → Y											CB	2	1																												
J M P	JUMP TO NEW LOC					4C	3	3																																		
J S R	JUMP SUB					20	6	3																																		
L D A	M → A	(1)	A9	2	2	AD	4	3	A5	3	2																															
L D X	M → X	(1)	A2	2	2	AE	4	3	A6	3	2																															
L D Y	M → Y	(1)	A0	2	2	AC	4	3	A4	3	2																															
L S R						4E	6	3	46	5	2	4A	2	1																												
N O P	NO OPERATION											EA	2	1																												
O R A	A ∨ M → A		09	2	2	0D	4	3	05	3	2																															
P H A	A → Ms	S-1 → S										48	3	1																												
P H P	P → Ms	S-1 → S										08	3	1																												
P L A	S+1 → S	Ms → A										68	4	1																												
P L P	S+1 → S	Ms → P										28	4	1																												
R O L						2E	6	3	26	5	2	2A	2	1																												
R O R						6E	6	3	66	5	2	6A	2	1																												
R T I	RTRN. INT.											40	6	1																												
R T S	RTRN SUB											60	6	1																												
S B C	A - M - C → A	(1)	E9	2	2	ED	4	3	E5	3	2																															
S E C	1 → C											38	2	1																												
S E D	1 → D											F8	2	1																												
S E I	1 → I											78	2	1																												
S T A	A → M					8D	4	3	85	3	2																															
S T X	X → M					8E	4	3	86	3	2																															
S T Y	Y → M					8C	4	3	84	3	2																															
T A X	A → X											AA	2	1																												
T A Y	A → Y											A8	2	1																												
T S X	S → X											BA	2	1																												
T X A	X → A											8A	2	1																												
T X S	X → S											9A	2	1																												
T Y A	Y → A											98	2	1																												

- (1) Aggiungere 1 a "N" se si passano i limiti di pagina.
 - (2) Aggiungere 1 a "N" se il BRANCH rimane nei limiti della pagina. Aggiungere 2 a "N" se il BRANCH avviene su un'altra pagina.
 - (3) Carry Negato = BORROW.
 - (4) Se si è in funzionamento "DECIMALE" il Flag Z non è valido. Bisogna eseguire un test dell'accumulatore per rivedere il risultato zero.
- X Registro Indice X
Y Registro Indice Y
A Accumulatore
M Locazione di memoria dell'indirizzo effettivo
Ms Locaz. di memoria puntata dallo STACK POINTER.

- + Somma.
- Differenza.
∧ AND.
V OR.
✓ OR esclusivo.
√ Modificato.
- Non modificato.
- M7 Bit 7 della loc. di memoria (MSB).
M6 Bit 6 della loc. di memoria.
N N. di cicli.
N di Bytes

METODI DI INDIRIZZAMENTO

IMM - Indirizzamento immediato. - L'operando è contenuto nel 2° byte dell'istruzione.

ABS - Indirizzamento assoluto. - Il secondo byte dell'istruzione contiene gli 8 bit più bassi dell'indirizzo effettivo. Il terzo byte contiene gli 8 bit più alti dell'indirizzo effettivo.

Z Page - Indirizzamento in pagina zero. - Il secondo byte dell'istruzione contiene gli 8 bit più bassi dell'indirizzo effettivo. Gli 8 bit più alti sono zero.

A - Accumulatore. - Istruzioni da 1 byte che agiscono sull'accumulatore.

Z, Page, X - Z, Page, Y - Indirizzamento indicizzato in pagina zero. - Il secondo byte della istruzione viene sommato al registro indice (il Carry viene ignorato) per formare il byte basso dell'indirizzo effettivo. Il Byte alto dell'indirizzo effettivo è zero.

ABS, X - ABS, Y - Indirizzamento indicizzato assoluto. - L'indirizzo effettivo viene calcolato sommando il registro indice al secondo e terzo byte dell'istruzione.

(IND, X) - Indirizzamento indicizzato indiretto. - Il secondo byte dell'istruzione viene aggiunto al registro indice X (sen

2° esercizio - Scrivere un programma che esegua la moltiplicazione di due numeri da 8 bit (1 byte).

Il metodo più immediato, per tutto ciò che fino ad ora avete appreso, è quello di caricare il 1° numero nell'index X e sommare l'altro tante volte quanto è il contenuto di X; questo è possibile se ad ogni loop si decrementa X. Attenzione però che X si decrementa (e incrementa) sempre in esadecimale.

3° esercizio - Scrivere un programma che carichi automaticamente dalla locazione 0000 alla 0050 numeri crescenti in codice esadecimale dallo 00 al 50 (in pratica sul display dati si dovranno vedere numeri sempre uguali alla parte bassa dell'indirizzo fino alla locazione 0050).

Programma 1 - Trasformazione di numeri decimali negativi e positivi in corrispondenti nel sistema complemento a due.

```
0200 =A5 11 FD 0C A2 00 F8 18 E8 69 99 C9 00 D0 F8 8A
0210 =18 D8 A6 10 F0 02 69 64 60 EA A8 29 0F AA BD EA
0220 =FF 85 0F 98 4A 4A 4A AA BD EA FF 60 EA EA EA
0230 =A9 00 AA 95 00 E8 ED 9F D0 F9 A5 11 20 1A 02 85
0240 =91 A5 0F 85 92 A5 12 20 1A 02 85 93 A5 0F 85 94
0250 =A5 10 D0 04 A9 3F D0 02 A9 06 85 90 20 7E FF EA
0260 =F8 20 2D FF D0 04 85 0A F0 D0 A5 0A D0 CC E6 0A
0270 =A9 99 8D 03 FD 20 57 FF C9 15 D0 09 A9 40 45 8F
0280 =85 8F 4C 99 02 C9 10 B0 44 48 A5 11 0A 0A 0A 0A
0290 =26 10 85 11 68 05 11 85 11 A9 01 25 10 85 10 F0
02A0 =1B A5 8F D0 0D A5 11 C9 27 90 05 A9 27 85 11 18
02B0 =90 0A A5 11 C9 28 90 F8 A9 27 D0 F1 20 0D 02 A6
02C0 =8F FD 05 49 FF 38 69 0D 85 12 4C 3A 02 C9 12 D0
02D0 =F9 4C 30 02
```

Un gioco: il "master mind"

Ed ora, dopo tanto lavorare con software, istruzioni, esercizi etc., ci sembra opportuno un attimo di relax con un giochino molto simpatico e molto conosciuto sotto il nome di "master mind".

Si tratta in sostanza di un programma che fa generare casualmente all'AMICO 2000A 4 numeri diversi (o meglio un numero di quattro cifre) che dobbiamo indovinare. Si tratta di un gioco interattivo cioè di un gioco in cui il microcalcolatore dialoga con noi dando delle risposte. Come? Spieghiamo subito il gioco. Il programma è riportato a fine capitolo. (viene dato il cosiddetto codice oggetto) e si carica a partire dalla locazione di memoria 0200.

Per farlo partire, una volta caricato, bisogna posizionarsi all'indirizzo 02A0 poi premere RUN.

A questo punto tutte le cifre del display lampeggiano con una frequenza abbastanza elevata: in questa situazione il microelaboratore sta scegliendo le cifre che compongono il numero.

Per arrestare questa operazione di ricerca dei numeri e per cominciare quindi il gioco si preme il tasto C, a questo punto tutte le cifre sono zeri.

Cominciamo il gioco battendo sulla tastiera una successione di quattro numeri qualunque (all'inizio tutti possono essere quelli buoni) che appariranno sul display indirizzi.

Premiamo ora il tasto F, sul display dati apparirà qualcosa (può rimanere anche 00):

- La cifra a destra del display dati indica quante cifre su quattro sono state indovinate;

- la cifra a sinistra del display dati indica quante cifre di quelle indovinate sono al loro giusto posto.

In base a queste indicazioni batteremo sulla tastiera altri numeri o li di-

Programma 2 - Gioco del "master mind". Il programma deve essere fatto partire (RUN) una volta caricato dalla locazione 02A0.

```
0200 =20 0C FF 20 2D FF D0 04 85 0D F0 F4 A5 0D D0 F0
0210 =E6 0D F8 A9 99 8D 03 FD 20 57 FF C9 15 F0 28 EA
0220 =EA EA EA C9 10 B0 D9 A2 04 06 FA 26 FB CA D0 F9
0230 =05 FA 85 FA 4C 00 02 EA EA EA 48 29 0F 95 01 68
0240 =4A 4A 4A 4A 95 00 60 A5 FA A2 03 20 3A 02 A5 FB
0250 =A2 01 20 3A 02 A9 00 85 09 85 0A A0 04 A2 01 B5
0260 =00 D5 04 D0 02 E6 09 D5 04 D0 02 E6 0A E8 ED 05
0270 =D0 F5 20 85 02 88 D0 E5 A5 09 0A 0A 0A 0A 05 0A
0280 =85 F9 4C 00 02 A2 01 85 00 95 FF E8 ED 09 D0 F7
0290 =85 FB 95 FF B5 F7 95 FB 60 EA EA EA EA EA EA
02A0 =A9 00 85 F9 85 FA 85 FB A2 05 18 F8 A5 0C 29 07
02B0 =75 00 29 0F 95 00 C6 0C E8 ED 09 D0 EF A9 00 85
02C0 =08 A2 05 B5 00 D5 01 D0 DE 48 18 69 01 29 0F 95
02D0 =01 68 E6 0B 4C C1 02 E8 ED 08 D0 E9 C6 0B F0 06
02E0 =20 85 02 4C C1 02 20 0C FF A9 99 8D 03 FD 20 57
02F0 =FF C9 12 D0 03 4C 00 02 4C A8 02
```

sporremo in maniera differente fino a che il display dati non indica 44: questo significherà che abbiamo trovato il numero esatto scelto dal microelaboratore (quattro cifre esatte tutte al loro posto).

Ricordate di battere sempre il tasto F dopo aver introdotto i nuovi numeri per avere la risposta del microelaboratore.

Per aiutarvi nei ragionamenti è meglio scrivere su un foglio di carta i numeri introdotti e la relativa risposta.

La bravura del giocatore sta nell'indovinare con il minor numero di tentativi

(qualche volta si tratterà anche di fortuna, quella c'entra sempre!).

Il programma è piuttosto lungo da inserire a mano, vi consigliamo quindi di registrarlo subito, dopo averlo caricato la prima volta e averne verificato il buon funzionamento.

Attenzione, un ultimo avvertimento: il numero generato dal micro ha sempre tutte le cifre diverse; nel tentare di indovinarlo inserite sempre NUMERI CON CIFRE UNA DIVERSA DALLA ALTRA.

REGISTRO INDICE Y E STACK POINTER

Mancano ancora all'appello i due ultimi registri presenti nella CPU 6502. Essi sono il *registro indice Y* e il *puntatore di catasta operativa* (Stack Pointer) S.

L'interno della CPU, una volta esaminati questi ultimi registri si presenta finalmente completo; si faccia anche riferimento alla schematizzazione della CPU 6502 apparsa nella parte sesta di questo capitolo.

Il registro indice Y

Si tratta di un registro ad 8 bit del tutto simile al registro indice X, di cui si è parlato nella sesta parte.

La differenza sostanziale del registro Y rispetto a quello X già trattato consiste nelle possibilità di indirizzamento diverse per i due registri. Esse sono: l'indirizzamento indiretto indicizzato (possibile tramite il registro Y); inoltre l'indirizzamento indicizzato in pagina zero è possibile per il registro Y solo su due istruzioni e su molte di più per il registro X.

Spiegheremo comunque in dettaglio in un prossimo capitolo questi sistemi di indirizzamento; per una prima spiegazione si veda comunque la tabella riassuntiva delle istruzioni del 6502 pubblicata nel sesto capitolo.

Vediamo ora le varie istruzioni relative all'indice Y:

CPY (Compare Y). Paragone fra il contenuto di Y e un numero.

Questo paragone lascia invariato Y e condiziona semplicemente i bit dello Stato. In pratica viene eseguita nella CPU l'operazione $Y - M$ dove M è il numero

da paragonare a Y, il risultato di questa operazione non compare da nessuna parte in CPU, ma se questo risultato è zero automaticamente si ha $Z = 1$ (bit di zero dello Status), se il risultato è negativo si ha $N=1$ (bit di negativo dello Status), se il risultato è positivo si ha $C = 1$ (bit di Carry dello Status).

Il dato da paragonare con Y può essere scelto in tre sistemi diversi di indirizzamento.

Immediato (Codice operativo C0); il dato è nel secondo Byte dell'istruzione stessa.

Per esempio:

```
C0 CPY # $7F
7F
```

esegue il confronto fra il registro indice Y e il dato "immediato" 7F.

Pagina zero (Codice operativo C4); il dato di paragone è contenuto in una locazione di memoria in pagina zero.

Per esempio:

```
C4 CPY $04
04
```

Il contenuto del registro indice Y viene paragonato al contenuto della locazione di memoria 0004.

Assoluto (Codice operativo CC); il dato da paragonare è contenuto in una qualsiasi delle possibili 65.536 locazioni di memoria indirizzabili dal 6502, della quale viene dato l'indirizzo nella seconda parte dell'istruzione.

Per esempio:

```
CC CPY $35D6
D6
35
```

Con questa istruzione il contenuto del registro indice Y viene paragonato al contenuto della locazione di memoria numero 35D6.

DEY (Decrement Y). Con questa istruzione si decrementa di uno (sempre

con il sistema esadecimale) il contenuto del registro indice Y. Il suo sistema di indirizzamento è ovviamente implicito (quando si scrive questa istruzione non si deve precisare nulla di più alla CPU), il codice operativo è 88, dalla operazione vengono condizionati i bit di zero e di negativo dello Status.

Esempio: se si ha $Y=1$, eseguendo successivamente queste due operazioni:

```
88 DEY
88 DEY
```

si ha dopo il 1° DEY: $Y=0$, con $N=0$ e $Z=1$;

dopo il 2° DEY: $Y=FF$ (cioè -1), con $N=1$ e $Z=0$.

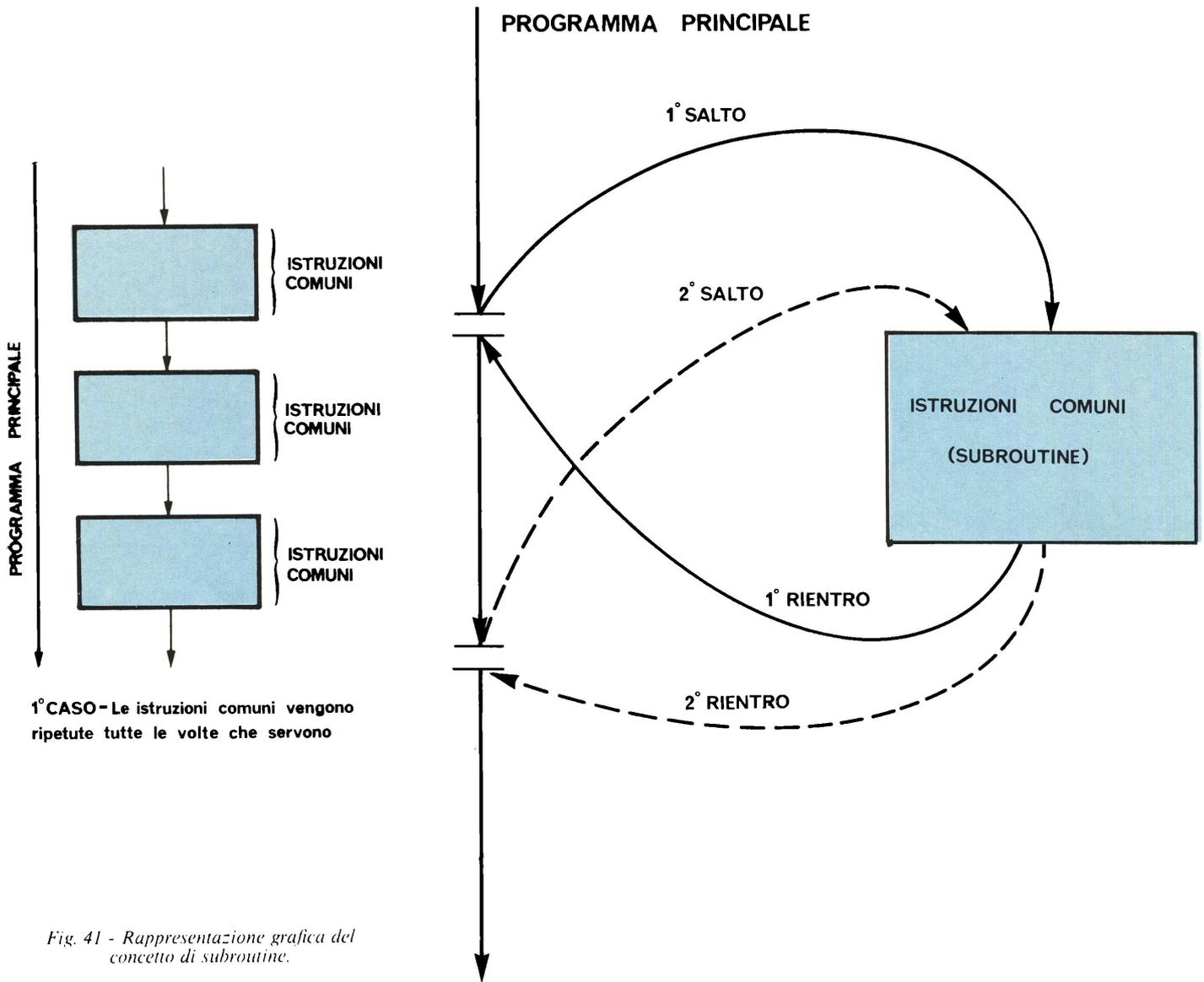
INY (Increment Y). Valgono le stesse cose dette per l'istruzione precedente. Il contenuto di Y viene incrementato di 1 in esadecimale, il codice operativo dell'istruzione è C8, vengono influenzati N e Z.

LDY (Load Y). Con questa istruzione si carica un valore nel registro Y. Il valore da caricare può essere identificato con cinque diversi sistemi di indirizzamento, alcuni dei quali sono stati già analizzati. Vengono influenzati i bit N e Z.

STY (Store Y). Con questa istruzione si porta il valore di Y in memoria, lasciando Y invariato. Questo trasferimento può essere fatto operando in tre diversi sistemi di indirizzamento mentre i bit di stato non vengono influenzati. Quest'ultima particolarità è molto importante e deve essere tenuta presente per tutte le istruzioni che non influenzano i bit di stato. Con gli esempi che seguono vogliamo dimostrare l'effetto di questa caratteristica.

1° esempio: l'istruzione STY **non influenza** i bit di stato.

Poniamo che sia $Y=0$ e sia contenuto nella locazione 0005 il dato 07.



1° CASO - Le istruzioni comuni vengono ripetute tutte le volte che servono

Fig. 41 - Rappresentazione grafica del concetto di subroutine.

Scriviamo il programma:

0200	Loop C6	DEC \$05	Decrementa il contenuto della locazione di memoria 0005
1	05		
2	84	STY \$06	Memorizza il contenuto di Y nella 0006
3	06		
4	D0	BNE	Salta se il risultato di DEC non è uguale a zero (e NON se il contenuto del registro Y non è uguale a zero!)
5	FA	Loop	

2° esempio: l'istruzione LDY influenza i bit di stato N e Z. Poniamo che sia il contenuto della locazione di memoria 0005 uguale a 0 e

quello della locazione 0006 uguale a 07.

Scriviamo il programma:

0200	Loop C6	DEC \$05	Stesso commento programma precedente)
1	05		
2	A4	LDY \$06	Carica in Y il contenuto della locazione di memoria 0006 (che è 7 ≠ 0!)
3	06		
4	F0	BEQ	Salta se il bit di Z è = 1
5	FA	Loop	

In questo esempio le cose vanno molto diversamente, infatti se LDY non influenzasse i bit di stato faremmo subito un Loop in quanto il contenuto della locazione di

memoria 0005 (che è 01) va subito a zero. Ma l'istruzione LDY carica in Y un numero, 07, diverso da zero, ponendo Z = 0 (bit delle Status di zero = 0).

È allora **solo** quest'ultima istruzione che determina la condizione di salto e **non** la prima (DEC).

TAY e TYA (Transfer A → Y e Y → A). Queste istruzioni servono a trasferire il contenuto dell'accumulatore nel registro Y o viceversa. L'indirizzamento è implicito come al solito vengono influenzati i bit di Status N e Z.

Un esempio pratico

A questo punto verificiamo nella pratica, con l'AMICO 2000/A, la funzione e l'uso di alcune delle istruzioni appena viste scrivendo un programma che fa ac-

cendere e spegnere il display con una intermittenza di un secondo circa.

Facciamo notare che in questo programma sono contenute delle subroutine del monitor cui si fa riferimento nel paragrafo hardware del prossimo capitolo.

Analizziamo brevemente di questo programma, il cui listing è riportato nella Tabella 16 i punti che maggiormente ci interessano.

Alla locazione 0208 comincia il primo Loop che mi permette di caricare su sei locazioni di memoria successive un numero che è alternativamente 00 oppure FF (se è 00 il display è spento, se è FF ogni cifra indica il numero 8; si rimanda sempre alla spiegazione hardware del prossimo capitolo).

Notiamo alla locazione 0208 il sistema di indirizzamento indicizzato in pagina

zero. Questo Loop viene eseguito 6 volte (si vede l'istruzione A2 06).

Loop di ritardo. Alla locazione 020D carichiamo il numero FF nel registro Y; vogliamo far notare che prima di entrare nella subroutine di rinfresco del display (JSR FF7E) salviamo il contenuto del registro Y nella locazione di memoria 0010 (istruzione STY \$10) perché la subroutine stessa modifica, per come è stata costruita, anche il contenuto di Y. Al ritorno dalla subroutine quindi dobbiamo ripristinare questo contenuto (istruzione LDY \$10) decrementandolo poi fino ad arrivare a zero (istruzioni DEY e BNE Loop1).

Questo Loop di rinfresco viene eseguito FF volte (255 volte) e quindi per un certo tempo, che dipende dal tempo impiegato dalla CPU a eseguire la routine di rinfresco display, il display stesso rimane acceso o spento a seconda che sia stato scritto FF o 00.

Si potrà quindi modificare la cadenza di accensione cambiando il valore contenuto nella locazione di memoria 020E: il tempo minimo di impulso si ha se scriviamo 01 (il display in pratica lo vedremo sempre acceso data l'elevata frequenza dell'intermittenza), mentre il tempo massimo di impulso si ha se scriviamo 00. Perché? Lasciamo a voi la risposta.

Come ultima particolarità facciamo notare che per passare alternativamente da 00 a FF (cioè acceso/spento del display) eseguiamo una negazione tramite l'istruzione EOR #\$FF del contenuto della locazione di memoria 0F (ricordiamo che 00 EOR FF = FF; FF EOR FF = 00).

Dobbiamo puntualizzare infine per i lettori più sofisticati che lo stesso programma potrebbe essere scritto con un numero inferiore di istruzioni.

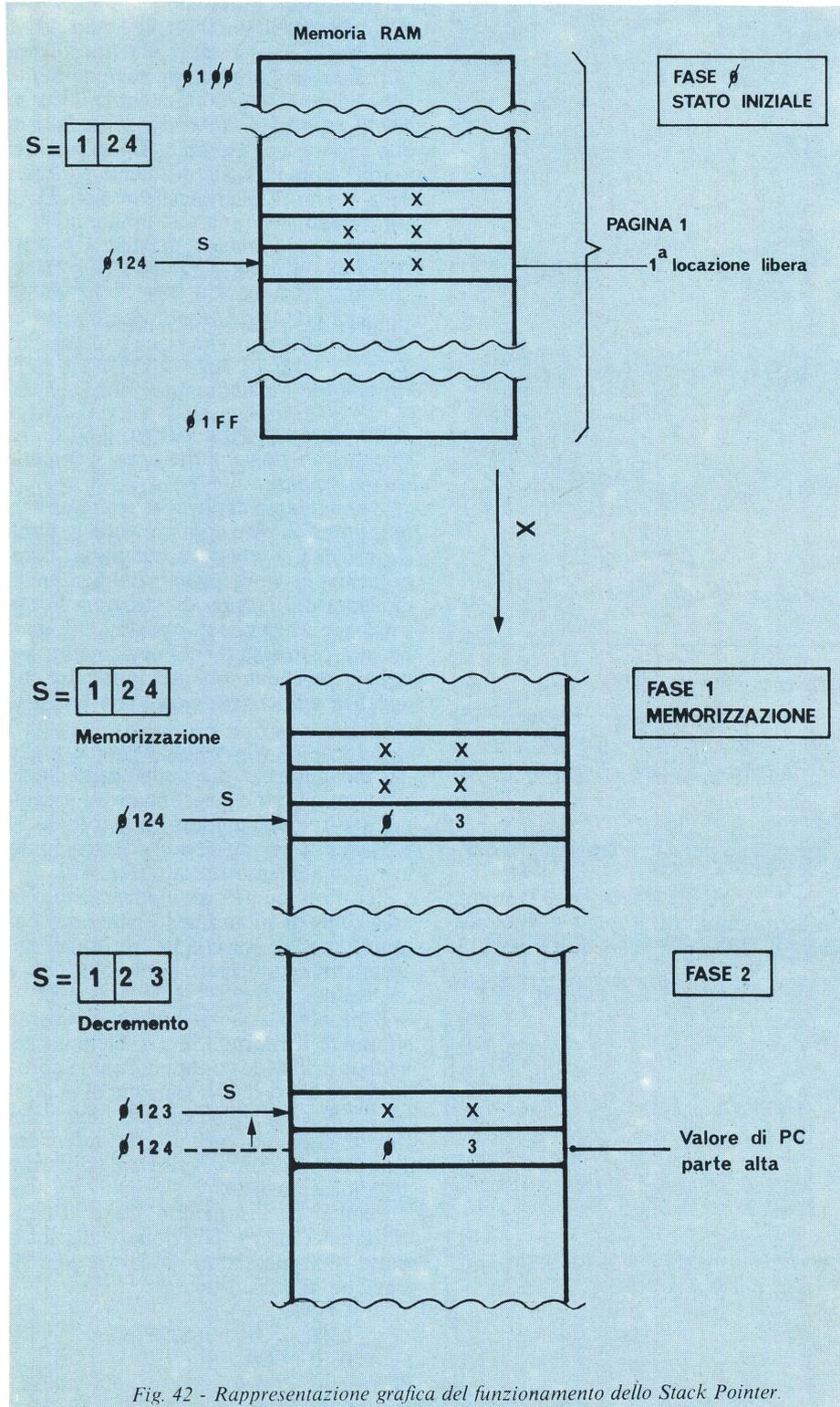


Fig. 42 - Rappresentazione grafica del funzionamento dello Stack Pointer.

L'uso principale di questo registro è dettato dal suo stesso nome, lo si usa infatti per "puntare" in una certa zona di memoria, ciò permette alcuni nuovi sistemi di indirizzamento.

Esaminiamo, per ora, due di questi indirizzamenti rimandando gli altri ad un paragrafo successivo.

Indirizzamento assoluto indicizzato

Effettuato tramite il puntatore Y.

Nella tabella riassuntiva delle istruzioni del 6502 è indicato come ABS.Y.

Con questo sistema la CPU calcola l'indirizzo della locazione di memoria interessata sommando il valore assoluto

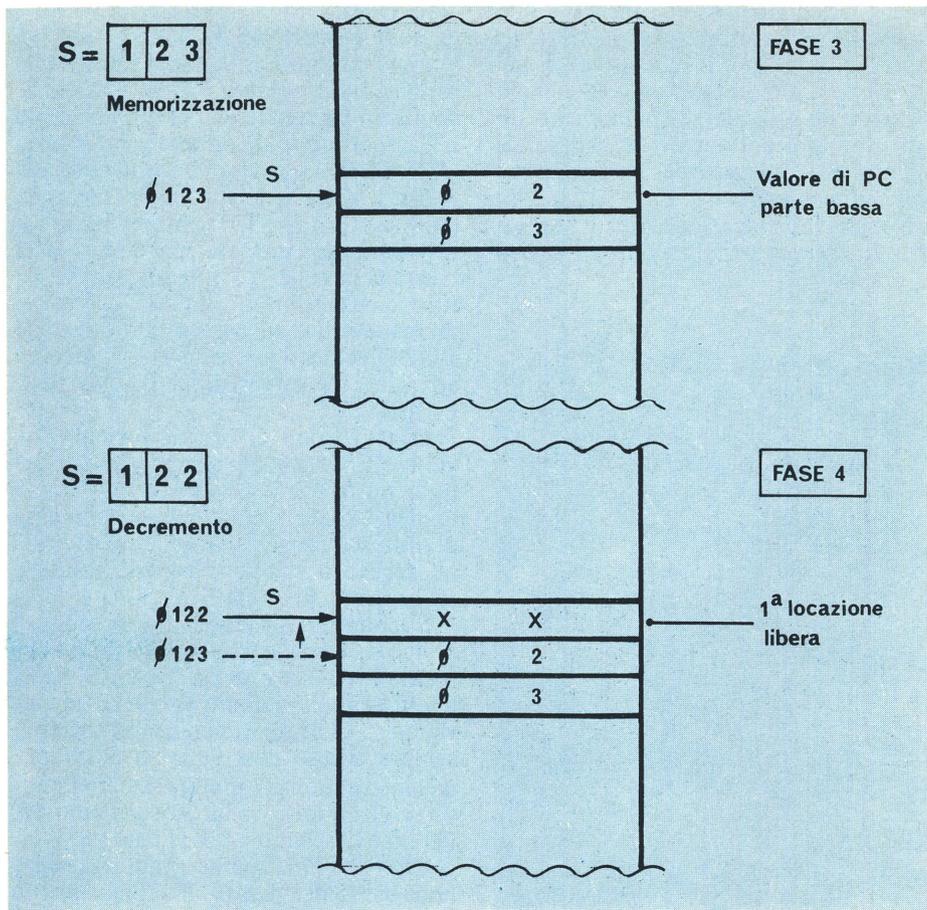


Fig. 42 - Rappresentazione grafica del funzionamento dello Stack Pointer.

che forma il 2° e 3° byte dell'istruzione, al contenuto del registro Y. Facciamo un esempio: sia $Y = 3F$. l'operazione:

79 ADC 0203, Y
03
02

esegue la somma del contenuto dell'accumulatore con il contenuto della locazione di memoria $0203 + Y = 0203 + 3F = 0242$.

Il risultato della somma va a finire nell'accumulatore.

Indirizzamento in pagina base

Effettuato tramite il puntatore Y. Nella tabella è indicato come Z.PAGE. Y.

Esiste solo per due istruzioni: la LDX e la STX.

L'indirizzo della locazione di memoria con il contenuto della quale si vuole caricare X (LDX) o nella quale si vuole

portare X (STX) viene calcolato sommando, senza tenere conto del Carry, l'indirizzo di pagina zero indicato nel secondo byte dell'istruzione e il registro Y.

Per esempio, se si ha $Y = FF$, l'istruzione:

96 STX 15, Y
15

memorizza il contenuto del registro indice X nella locazione di memoria $15 + Y = 15 + FF = 14$ (cioè 0014, ricordiamo che qui si è in pagina 0).

Un altro uso altrettanto importante di Y è come contatore.

Si può per esempio caricarlo con un valore ed eseguire una operazione molte volte successive decrementando il valore di Y fino a farlo arrivare a zero. Questo può servire, per esempio, a generare dei ritardi (come il ritardo della partenza della sirena di allarme di un sistema di antifurto).

Pointer di Stack e Subroutine

Per introdurre il concetto di STACK, spieghiamo brevemente cosa si intende per Subroutine.

Ammettiamo di dover eseguire un certo numero di istruzioni più volte in punti diversi di uno stesso programma. L'approccio più immediato sarebbe quello di scrivere il gruppo di istruzioni in oggetto ogni volta che ne abbiamo bisogno; questo comporterebbe ovviamente un notevole dispendio di memoria oltre che di fatica a riscrivere ogni volta le stesse cose.

Un sistema più evoluto è invece quello di abbandonare momentaneamente il programma principale, andare ad eseguire le istruzioni comuni, poi tornare al programma principale nello stesso punto in cui lo abbiamo lasciato (Fig. 41).

Per fare questo si memorizzano in una zona di memoria le istruzioni comuni, mentre si esegue un salto dal programma principale ogni volta che è necessario.

Il problema che si presenta è come effettuare il ritorno. Se per esempio accediamo alle locazioni comuni (Subroutine) a partire dalla locazione di memoria 0300, dobbiamo tornare, una volta che le abbiamo eseguite, a riprendere il nostro programma principale a partire dalla locazione 0303.

Perché? Se l'istruzione "Salta alla subroutine" (JSR) è posizionata alla locazione 0300, possiamo ipotizzare una situazione di questo tipo:

	0300	20	JSR 0280
	0301	80	
PC →	0302	02	
	0303	xx	

Tabella 16 Programma per far lampeggiare il display

0200	A9 LDA #00	1	20 JSR FF7E
1	00	2	7E
2	85 STA \$0F	3	FF
3	0F	4	A4 LDY \$10
4 Loop 2	A5 LDA \$0F	5	10
5	0F	6	88 DEY
6	A2 LDX = 06	7	D0 BNE Loop 1
7	06	8	F6
8 Loop	95 STA 8E,X	9	A5 LDA \$0F
9	8E	A	0F
A	CA DEX	B	49 EOR #SFF
B	D0 BNE Loop	C	FF
C	FB	D	85 STA \$0F
D	A0 LDY #SFF	E	0F
E	FF	F	4C JMP Loop 2
F Loop 1	84 STY \$10	0220	04
0210	10	0221	02

Allora il Program Counter (PC), dopo che il processor ha letto l'istruzione (che è sempre formata da 3 byte) e prima di averla eseguita si trova a puntare la locazione 0302; a questo punto il processor va a eseguire la subroutine che comincia all'indirizzo 0280 e dopo averla completata deve ricaricare il valore 0303 nel PC per proseguire l'esecuzione del programma dal punto in cui era stato interrotto. Il valore 0303 è l'indirizzo dell'istruzione successiva al JSR. Tutta la procedura descritta avviene ogni volta che si incontra la istruzione JSR, dovunque essa si trovi.

Perché ciò avvenga il problema principale quindi è: dove salviamo (memorizziamo) il valore del PC prima di eseguire il salto?

Nello STACK.

Cosa è lo Stack?

Non è altro che una zona di memoria RAM situata in pagina 1 (indirizzi da 0100 a 01FF). È all'interno di questa zona che viene salvato (memorizzato) automaticamente il PC su due locazioni di memoria successive (ricordiamo che il PC è un registro di 16 bit = 2 byte).

A questo punto abbiamo bisogno dello *Stack Pointer* (Puntatore dello Stack), un registro da 8 bit (+ 1 bit sempre a 1 perché siamo in pagina 1) che contiene l'indirizzo della 1ª locazione di memoria dello Stack nella quale dobbiamo memorizzare o dalla quale dobbiamo prelevare il PC che ci interessa salvare.

L'unico problema ora è quello di assegnare un valore iniziale (ovvero posizionare) allo Stack Pointer - che d'ora in avanti indicheremo anche con il solo simbolo S -; la cosa nel caso dell'AMICO 2000/A viene fatta dallo stesso programma del monitor, ma può essere fatta dal programmatore con delle istruzioni specifiche, come vedremo nel paragrafo successivo.

Vediamo ora di chiarire con degli esempi pratici tutto ciò che finora è stato spiegato.

Lancio di una Subroutine

Esiste una istruzione specifica, come abbiamo precedentemente accennato, che permette di eseguire il salto a subroutine presenti in memoria. Essa è: JSR (codice operativo 20).

Questa istruzione è seguita da 2 byte contenenti l'indirizzo in cui comincia la subroutine interessata.

Vediamo come opera la CPU.

Ammettiamo che sia S (Stack Pointer) = 124 (Fase zero della Fig. 42).

La CPU incontra l'istruzione:

```
0300 20 JSR 0280
0301 80
```

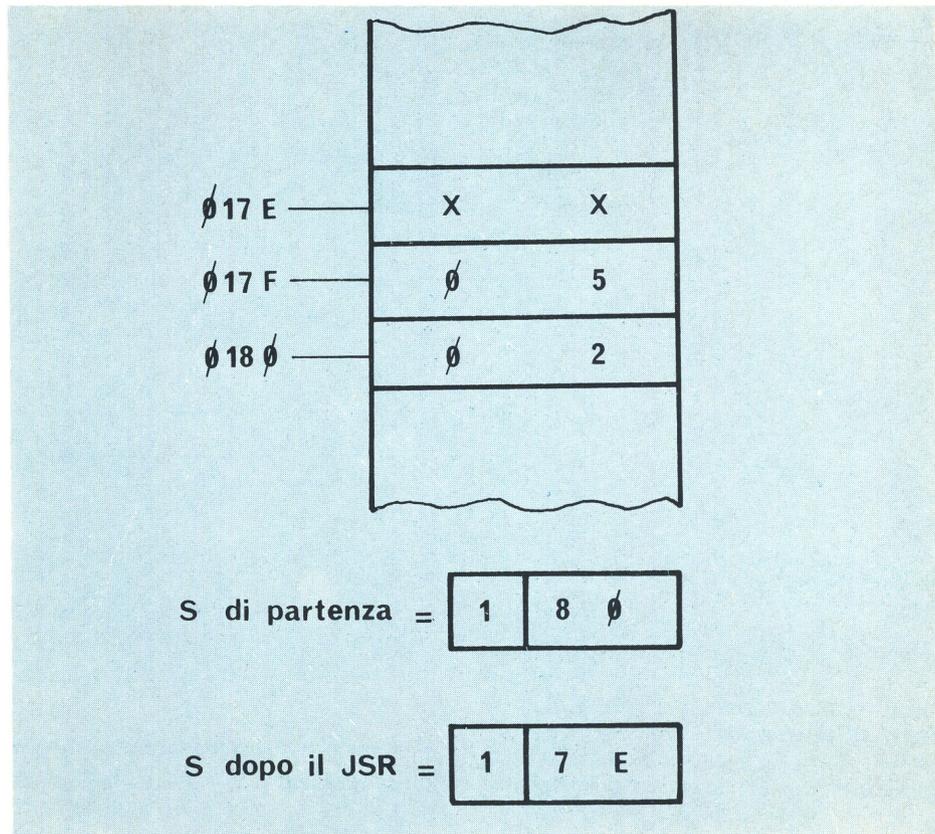


Fig. 43 - Esempio di salvataggio del PC e modifica del contenuto dello Stack Pointer dopo l'esecuzione dell'istruzione JSR.

```
0302 02.
```

```
0303 xx
```

Dopo che l'istruzione è stata letta si ha il Program Counter:

PC = 0302

A questo punto la CPU salva automaticamente nello Stack il valore del program Counter.

Questo avviene in varie fasi come descritto nella Figura 42.

FASE 1 - La parte alta del PC (03) viene memorizzata (salvata) nella locazione puntata da S.

FASE 2 - S viene decrementato di uno.

FASE 3 - La parte bassa del PC (02) viene memorizzata nella locazione puntata da S.

FASE 4 - S viene ancora una volta decrementato di uno.

Avvenuto ciò il PC viene caricato con il valore presente nella istruzione JSR (0280) e l'esecuzione del programma riprende da lì.

Per tornare indietro, per uscire cioè dalla subroutine, si usa l'istruzione RTS, codice operativo 60, che è l'istruzione che chiude tutte le subroutine.

Come opera l'istruzione RTS?

Quando la CPU incontra la RTS, automaticamente decrementa il valore di S. Quindi carica la parte bassa dell'indirizzo del PC con il contenuto della locazione di memoria puntata da S.

Successivamente si decrementa ancora S e viene caricata la parte alta del PC con il contenuto della locazione di memoria puntata, ripristinando il PC dal quale siamo partiti.

Il procedimento è sostanzialmente l'inverso di quanto abbiamo illustrato nella Fig. 42, in particolare alla fine di RTS sarà ancora S = 124.

Abbiamo visto come la CPU abbia salvato in una zona di memoria ben identificata (tramite l'S) il punto da cui è partita e come quindi sia in grado di riprendere dallo stesso punto l'esecuzione del programma principale.

Esempio di utilizzo di una Subroutine

Come esempio possiamo costruire un semplicissimo programma che scrive una parola sul display dell'AMICO 2000/A.

Esiste una subroutine nel programma di Monitor che accende il display secondo una configurazione assegnata. In pratica allora nel programma che stiamo per scrivere utilizzeremo una parte di un programma precedentemente scritto per un suo preciso scopo, ma che è comunque a nostra disposizione (bontà delle subroutine!).

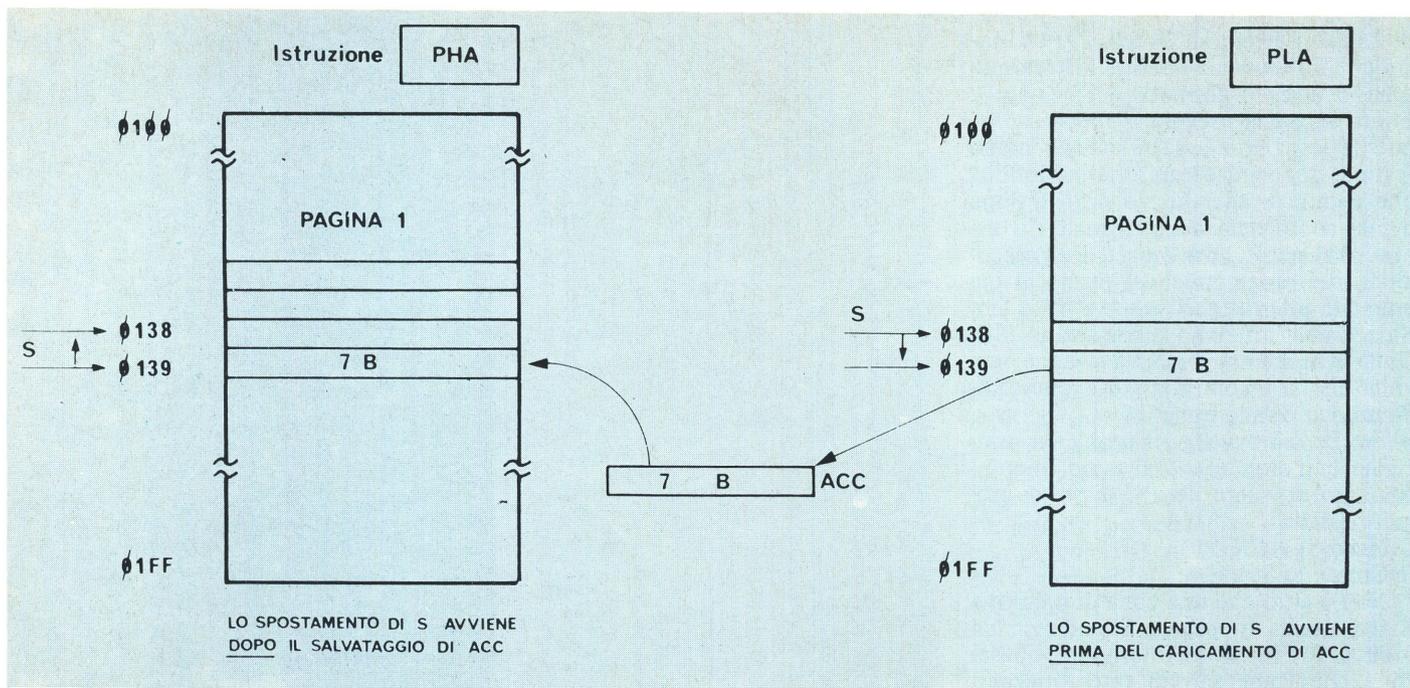


Fig. 44 - Rappresentazione grafica del funzionamento delle istruzioni PHA e PLA.

Il programma è il seguente:

```

0300 Loop 20 JSR FF7E
1       7E
2       FF
3       4C JMP Loop
4       00
5       03
  
```

È quasi banale, ma molto utile: questo programma non fa altro che visualizzare in continuazione (tramite la subroutine di rinfresco del display posizionata a FF7E) il contenuto delle locazioni di memoria 8F, 90, 91, 92, 93, 94 (situate in pagina zero) sulla 1^a, 2^a, 3^a, 4^a, 5^a, 6^a cifra del display.

L'istruzione JMP Loop mantiene costantemente il processor nella condizione di far eseguire in continuazione la subroutine di rinfresco.

In definitiva per scrivere un qualcosa su ogni digit del display basta memorizzare un certo dato nella locazione di memoria corrispondente. C'è una procedura che permette di controllare uno per uno i sette segmenti di ogni cifra (digit) e che spiegheremo nella parte hardware del prossimo capitolo.

Per ora vi diamo i dati per scrivere la parola ASEL sul display.

Caricate:

Indirizzo	Dato	Commento
008F	77	lettera A
90	6D	lettera S
0091	79	lettera E
92	38	lettera L
93	00	cifra spenta
94	00	cifra spenta

A questo punto potete far partire il programma dalla 0300.

Ora, se siete bravi, potete tentare di arricchire questo programma facendo accendere e spegnere il display in modo intermittente con i loop di ritardo che abbiamo già esaminato.

Provate senza disperare alle prime difficoltà.

Istruzioni che caricano lo Stack Pointer

Abbiamo visto l'uso dello Stack Pointer nella gestione delle subroutine. Vogliamo ora analizzare le istruzioni che ci permettono di caricare un certo valore nell'S, cioè di modificarne direttamente il contenuto. Questa operazione serve essenzialmente all'accensione della macchina. Nel caso dell'AMICO 2000/A ciò avviene automaticamente nel programma di Monitor che carica S = 1FF. In generale non avremo mai bisogno di modificare S, salvo che in casi particolari.

Passiamo ad analizzare le istruzioni.

TSX (Transfer Stack Pointer in X). Codice operativo BA.

Questa operazione trasferisce il contenuto dello Stack Pointer nel registro indice X.

TXS (Transfer X to Stack Pointer). Codice operativo 9A.

Questa operazione trasferisce il contenuto del registro indice X nello Stack Pointer.

Come sempre in tutte le istruzioni di trasferimento il registro di partenza (S nella 1^a istruzione e X nella 2^a) rimane invariato dopo l'esecuzione dell'istruzione.

Vi renderete subito conto che l'unico mezzo per caricare un valore nell'S consiste nel trasferirlo dal registro indice X.

Ricordiamo ancora una volta che lo Stack Pointer S è un registro a 8 bit che punta in memoria su una locazione di pagina 1. Se cioè si ha S = 8F, la locazione di memoria puntata è la 018F, dove 01 è la pagina 1 e 8F è il contenuto dell'S.

Ora comincia a diventare chiara la ragione per cui i nostri programmi cominciano tutti da 0200.

Conviene *sempre* tener libera la pagina 0 (locazioni 0000 ÷ 00FF) per eseguire i calcoli poiché l'indirizzamento in pagina zero è molto semplice e richiede istruzioni di soli 2 byte. La pagina 1 invece è generalmente dedicata allo Stack cioè a quella zona di memoria relativa alle subroutine e, come vedremo a salvataggi di parametri diversi.

Due esercizi pratici

Prima di analizzare le altre istruzioni relative allo Stack Pointer facciamo un piccolo esercizio. Vogliamo esaminare dove il microelaboratore AMICO 2000/A ha lo Stack Pointer in un certo istante.

Possiamo scrivere questo breve programma:

Possiamo scrivere questo breve programma:

```

0200 BA TSX Prelevo il valore dell'S e lo metto in X.
1 86 STX $00 Porto X nella locazione di memoria 00.
2 00
3 4C JMP Monitor
4 22
5 FE

```

Essendo questo programma in generale vedremo apparire sul display dati FF (ovvero S = FF). Cioè lo Stack Pointer è puntato proprio in cima alla sua zona di azione: questo perché, come abbiamo accennato, viene caricato così nel programma di Monitor dell'AMICO 2000/A.

Vediamo ora un altro esempio scrivendo questo programma:

```

0200 A2 LDX # $80 Carico in X il numero 80.
-1 80
2 9A TXS Lo trasferisco nello Stack Pointer.
3 20 JSR $0300 Salto alla subroutine che inizia alla locazione 0300
4 00
5 03
6 xx

```

```

0300 BA TSX Trasferisco in X il valore che S ha assunto dopo
1 86 STX $00 l'esecuzione dell'istruzione JSR
2 00 Memorizzo questo valore nella loc. 0000
3 4C JMP Monitor
4 22
5 FE

```

Dopo aver fatto girare il programma troveremo che lo Stack Pointer posizionato a 80, si è decrementato di due posizioni (leggeremo infatti 7F sul display dati alla locazione 0000).

Ciò è dovuto alla esecuzione della istruzione JSR che ha salvato il PC di partenza nelle locazioni 0180 e 017F. Se infatti andiamo ad esaminare il contenuto delle locazioni di memoria 017F e 0180 troveremo nella prima il valore 05 (PC parte bassa) e nella seconda 07 (PC parte alta) (vedi Fig. 43).

Vogliamo farvi notare infine che questa non è una vera subroutine (infatti non ne usciamo con una istruzione RTS!) ma solo un esempio dimostrativo di come avviene il salvataggio nello Stack.

Esaminiamo ora altre quattro istruzioni molto importanti relative allo Stack.

PHA (Push Accumulator). Codice operativo 48.

Con questa istruzione il contenuto dell'accumulatore viene salvato nella locazione di memoria puntata dallo Stack Pointer quindi S viene decrementato (vedi Fig. 44).

Quindi se S = 39 (cioè punta alla locazione 0139 pagina 1) e ACC = 7B dopo l'esecuzione di PHA si ha:

S = 38, ACC = 7B

e il contenuto della locazione di memoria 0139 è = 7B.

PHP (Push Status). Codice operativo 08. Avviene la stessa cosa dell'istruzione precedente ma viene salvato il registro di Status invece dell'Accumulatore.

PLA (Pull Accumulator). Codice operativo 68.

Con questa istruzione il contenuto dell'S viene incrementato di uno, quindi viene prelevato il contenuto della locazione puntata che viene portato in ACC (vedi Fig. 44).

Se si ha S = 38, Loc. di memoria 0139 = 7B dopo l'istruzione PLA abbiamo:

S = 39, ACC = 7B.

PLP (Pull Status). Codice operativo 28.

Avviene la stessa cosa dell'istruzione precedente ma viene caricato il registro di Status.

L'uso di queste istruzioni è evidente: se si deve salvare il contenuto dell'accumulatore perché dobbiamo utilizzarlo in altre istruzioni, lo si può fare con questa semplice operazione di *un solo byte*.

Chiedetevi ora perché in una subroutine è "proibito" eseguire un PHA senza farlo seguire da un PLA prima di uscire dalla subroutine stessa con l'istruzione RTS.

Facciamo adesso un esempio per vedere come si modifica in pratica il registro S. Per non andare ad interferire con lo Stack del Monitor, spostiamo anche l'S in una zona della pagina 1.

Scriviamo il programma:

```

0200 A2 LDX # $39
1 39
2 9A TXS Carico lo Stack Pointer
3 A9 LDA # $7B Carico 7B nell'accumulatore
4 7B
5 48 PHA Porto il contenuto dell'ACC nello Stack
6 BA TXS Porto S in X
7 86 STX $00 Memorizzo X in 0000 per esaminarlo
8 00
9 4C JMP Monitor
A 22
B FE

```

Eseguendo questo programma si troverà 38 (valore attuale dello S) nella locazione di memoria 0000; se si va ad esaminare la 0139, si troverà il 7B che vi abbiamo caricato. Tenete comunque presente che, quando si torna al Monitor lo S viene ancora modificato dal programma di monitor stesso e viene riportato al valore originario.

Istruzioni di Shift e Rotazione

Tratteremo ora una classe di istruzioni che di primo acchito può sembrare oscura, ma che all'uso pratico si rivela di grande utilità, come avremo modo di vedere nel corso di questo stesso articolo. Si tratta delle istruzioni di SHIFT e ROTAZIONE.

Sono quattro istruzioni:

ASL (Arithmetic Shift Left) Shift (spostamento) a sinistra aritmetico.

LSR (Logical Shift Right) Shift a destra logico.

ROL (Rotate Left) Ruota a sinistra comprendendo il Carry.

ROR (Rotate Right) Ruota a destra comprendendo il Carry.

I codici operativi delle varie istruzioni vanno ricercati sulla tabella riassuntiva delle istruzioni del 6502.

Per una perfetta comprensione del modo di agire di queste istruzioni si veda la Figura 45.

L'uso di queste istruzioni verrà chiarito man mano che si procede nella trattazione; diamo di seguito solo un esempio. Vi siete mai chiesti come si fa a moltiplicare un numero per 10 nel nostro solito sistema decimale?

In effetti si esegue uno *Shift* a sinistra di un posto aggiungendo uno zero alla fine del numero.

Per esempio: $739 \times 10 = 7390$

La stessa cosa avviene per i numeri binari. Moltiplicare per 2 (che è la base della numerazione binaria, come 10 lo

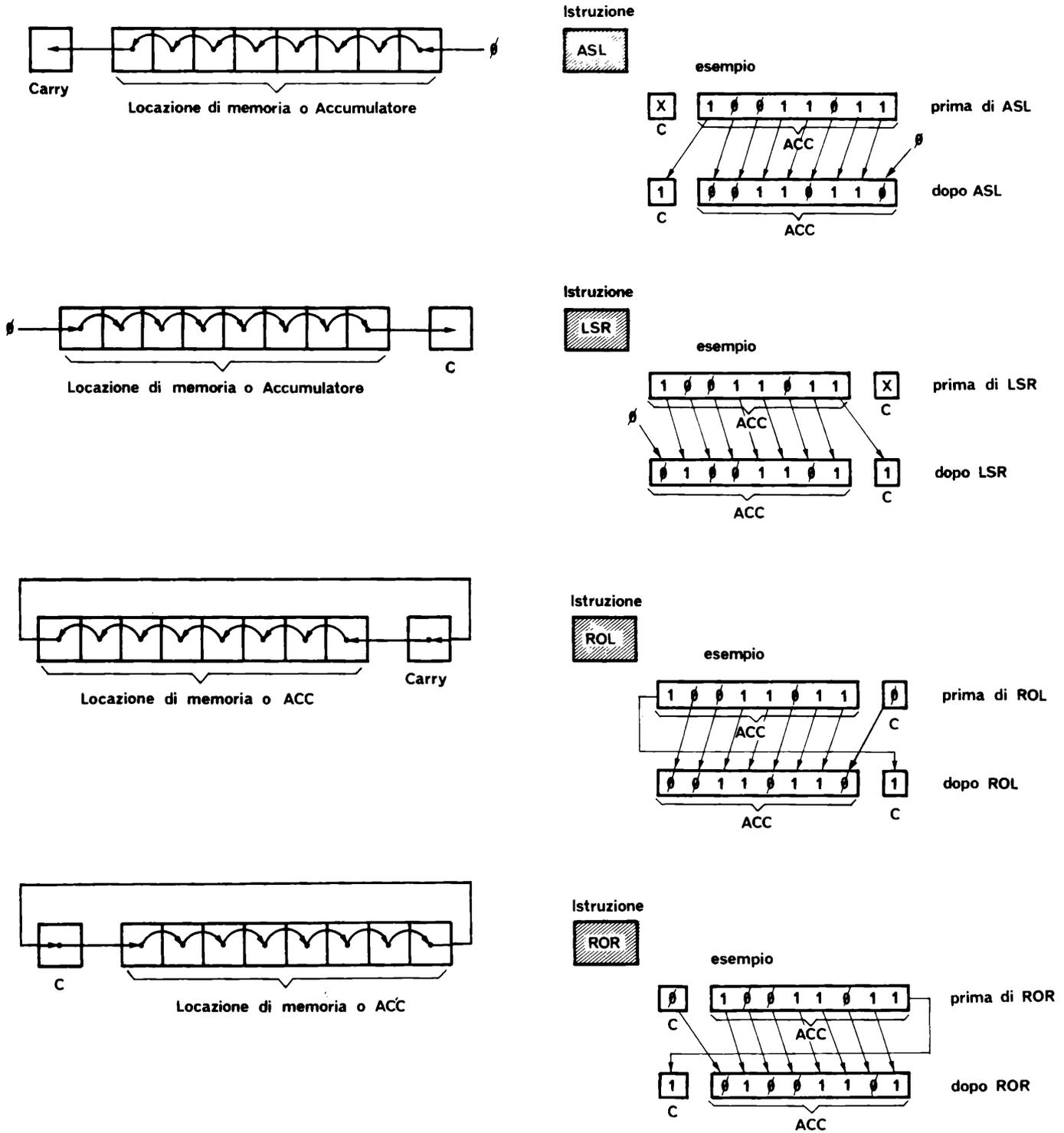
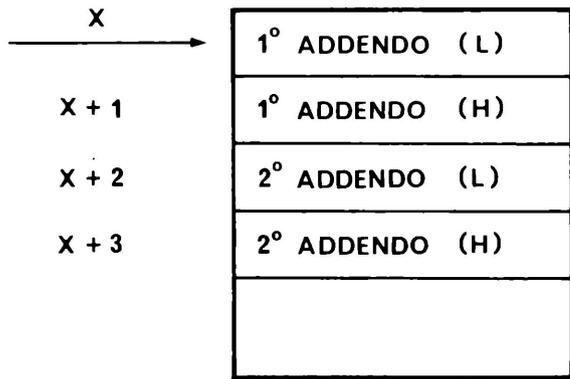


Fig. 45 - Rappresentazione grafica del funzionamento delle istruzioni ASL, LSR, ROL e ROR. Movimenti eseguiti sui bit.



L=Low = parte bassa
H=High=parte alta

esempio :

1 3 9 F
H L

Fig. 46 - Lo schema rappresenta come sono assegnate quattro successive locazioni di memoria per eseguire una somma di due numeri da 16 bit ciascuno.

è per quella decimale) un numero binario equivale ad eseguire uno Shift a sinistra di un posto, introducendo uno 0 in coda (ASL).

Dividerlo per 2 equivale ad uno Shift a destra, introducendo uno 0 all'inizio (LSR).

Eseguiamo per esempio $3B \times 2 = 76$ prelevando il dato dalla locazione di memoria 0001.

Scriviamo il semplice programma:

```
0200 A5 LDA $00
1 00
2 0A ASL A*
```

*Si vuole spostare a sinistra il contenuto dell'Accumulatore.

```
3 85 STA $01
4 01
5 4C JMP Monitor
6 22
7 FE
```

Partiamo introducendo 3B nella locazione 0000, eseguiamo il programma e troviamo 76 nella 0001. La 0000 rimane invariata.

E per moltiplicare un numero N per 5? Si potrà fare:

$$N \times 5 = (N \times 2) \times 2 + N$$

Perciò per eseguire $0C \times 5 = 3C$, con il moltiplicando in 0000 e il risultato posto ancora in 0000, si scrive il programma:

```
0200 A5 LDA $00
1 00
2 0A ASL A
3 0A ASL A
4 18 CLC Pulisco il Carry
5 D8 CLD Lavoro in
    esadecimale
6 65 ADC $00
7 00
8 85 STA $00
9 00
A 4C JMP Monitor
B 22
C FE
```

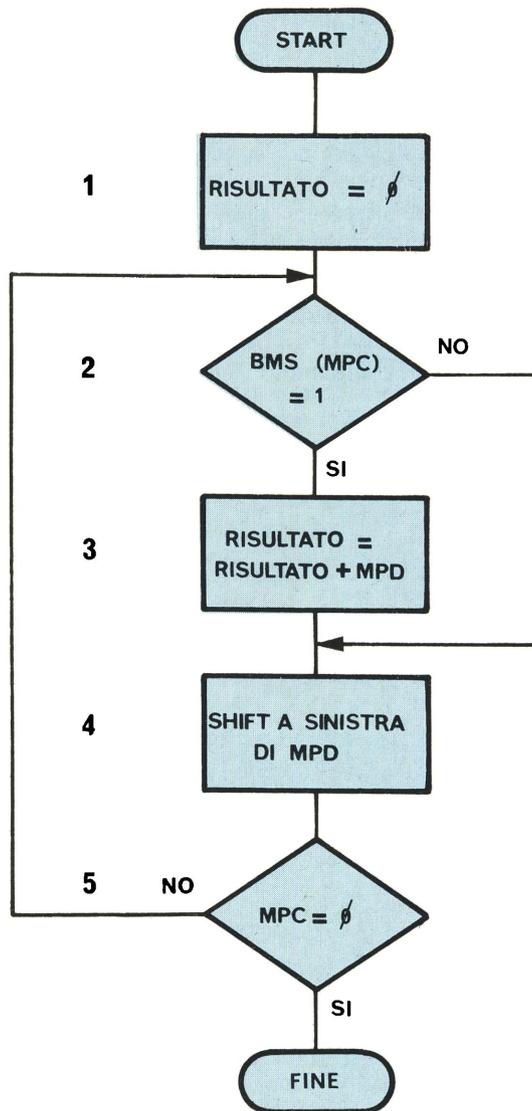
Notiamo che tutte queste operazioni di prodotto sono binarie (si lavora in esadecimale) e che presuppongono che il risultato rimanga nell'ambito degli 8 bit.

La moltiplicazione

Passiamo ora ad un'operazione matematica più seria, il prodotto di due numeri binari da 8 bit, che dà come risultato un numero binario di 16 bit.

Per capire quest'ultima affermazione facciamo un semplice esempio con i numeri decimali. Se si moltiplicano due numeri decimali da 2 cifre ciascuno si ottiene un numero decimale da 4 cifre.

Infatti il prodotto più grande che si può fare con due numeri da due cifre è: $99 \times 99 = 9801$, che è un numero da 4 cifre.



BMS = Bit meno significativo
MPC = Moltiplicatore
MPD = Moltiplicando

Fig. 47 - Diagramma di flusso del programma per eseguire una moltiplicazione.

La stessa cosa avviene per i numeri esadecimali.

Per eseguire l'operazione di prodotto cominciamo col costruire una routine che *somma* due numeri da 16 bit, operazione binaria.

Supponiamo che i due numeri siano posizionati in pagina 0 secondo la tabella di Fig. 46.

Il registro X è posizionato sul primo numero, ovvero X contiene l'indirizzo della locazione di memoria che contiene la parte bassa (meno significativa) del 1° addendo (1° ADDENDO L).

Il risultato della somma va messo al posto del secondo addendo, cioè sostituisce il valore del secondo addendo.

Scriviamo ora un programma che utilizzeremo come subroutine e che potrete conservare nella vostra biblioteca matematica.

0200	18	CLC	Carry = 0
1	D8	CLD	Funzionamento esadecimale
2	B5	LDA 0,X	Carico in ACC il 1° ADDENDO L
3	00		
4	75	ADC 2,X	Gli sommo il 2° ADDENDO L
5	02		
6	95	STA 2,X	Metto la parte bassa (L) del risultato nella locazione del 2° ADDENDO
7	02		
8	B5	LDA 1,X	Si ripetono le stesse operazioni per la parte alta. Notiamo l'assenza della operazione CLC
9	01		
A	75	ADC 3,X	
B	03		
C	95	STA 3,X	Risultato H a posto
D	03		
E	60	RTS	Ritorno dalla subroutine

Con questo programma abbiamo messo il Carry = 0, poi sommato le parti basse degli addendi, quindi, *tenendo conto del Carry*, abbiamo sommato le parti alte degli addendi.

Allo stesso modo a scuola ci hanno insegnato che:

$$\begin{array}{r} 27\ 84\ + \\ 15\ 52\ = \\ \hline \text{riporto}\ 1 \\ 43\ 36 \end{array}$$

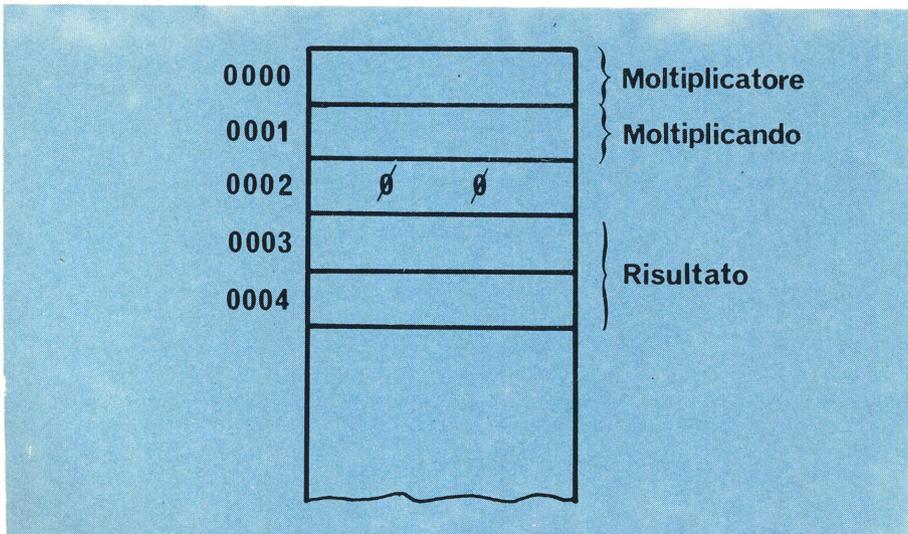


Fig. 48 - Locazioni di memoria usate nel programma per eseguire la moltiplicazione.

Per provare la nostra subroutine scriviamo ora il programma che segue:
0200 A2

LDX = 00 Lavoriamo con le prime 4 locazioni di RAM

a eseguire la subroutine

Per provare la nostra subroutine scriviamo ora il programma che segue:

0200 A2 LDX # 00 Lavoriamo con le prime 4 locazioni di RAM

1	00	
2	20	JSR 0300 Andiamo
3	00	a eseguire la subroutine
4	03	
5	4C	JMP Monitor
6	22	
7	FE	

Mettiamo i dati da sommare alle locazioni di memoria:

0000	1° addendo parte bassa
0001	1° addendo parte alta
0002	2° addendo parte bassa
0003	2° addendo parte alta

Il risultato compare nelle:

0002	somma parte bassa
0003	somma parte alta

Verificate che:

$$\begin{array}{l} 0013 + 0013 = 0026 \\ 3214 + 1F6A = 517E \\ A999 + 0001 = A99A \end{array}$$

Facciamo notare che con questo programma la somma più grande che si può eseguire è FFFF + FFFF = FFFE con Carry = 1; cioè abbiamo oltrepassato il limite dei 16 bit perché abbiamo un riporto (Carry) diverso da 0. Dal punto di vista generale è importante andare a controllare il Carry per vedere se c'è un riporto; nel caso specifico del prodotto, come vedremo, ciò non avverrà perché in effetti sommiamo un numero da 16 bit con uno da 8 bit.

Affrontiamo ora il problema del prodotto.

Vediamo prima come si esegue una moltiplicazione sulla carta quando usiamo numeri nel sistema decimale.

$$\begin{array}{r} 57\ x \\ 14\ = \\ \hline 228 \\ 570 \\ \hline 798 \end{array}$$

1° passo si fa 57 x 4
2° passo si fa 57 x 1 e il risultato lo si moltiplica x 10 (Shift a sinistra)
3° passo si sommano i due risultati parziali

La stessa cosa viene fatta in codice binario, tenendo conto che qui si moltiplica 0 x 0 o per 1; moltiplichiamo CD x 8D.

```

(CD)      11001101 x
(8D)     10001101 =
-----
          11001101
          00000000
          11001101
          11001101
          00000000
          00000000
          00000000
          11001101
(70E9) 111000011101001
-----
                15 bit

```

Sono 8 righe in cui si ha 0 nel posto in cui il moltiplicatore aveva 0, altrimenti si ha il moltiplicando moltiplicato ogni volta per 2 (cioè spostato ogni volta di una posizione)

Risultato = somma di tutti i parziali

Facciamo ora un programma che esegue queste operazioni. Notiamo ancora una cosa, che basta continuamente "shiftare" a sinistra il moltiplicando e sommarlo ad un risultato (che all'inizio abbiamo posto = 0) solo se il bit corrispondente del moltiplicatore è = 1. Vediamo il flow chart alla Fig. 47.

Commentiamo: *blocco 1* - pongo il risultato uguale a 0; *blocco 2* - mi chiedo se il bit meno significativo del moltiplicatore BMS (MPC) è = 1 (cioè eseguo uno shift del moltiplicatore a destra di una posizione portando il bit meno significativo nel Carry, quindi eseguo una istruzione di salto condizionato dal bit di Carry); *blocco 3* - se sì, sommo il moltiplicando al risultato, se no, passo direttamente al blocco 4; *blocco 4* - eseguo uno shift a sinistra di una posizione del moltiplicando; *blocco 5* - mi chiedo se tutto il moltiplicatore, dopo lo shift del blocco 2, è arrivato ad essere uguale a zero.

Prima di scrivere il programma poniamoci nelle condizioni riportate in Fig. 48.

```

0200 A2 LDX #01 Punto X sulla locazione 0001
1 01
2 A9 LDA #00
3 00
4 95 STA 1,X Porto 0 nella locazione 0002
5 01
6 95 STA 2,X
7 02
8 95 STA 3,X } Azzeramento del risultato
9 03
A loop 56 LSR FF,X Porto il bit meno signif. del moltiplicatore nel Carry
B FF
020C 90 BCC NOSOM Se è = 0 (Carry = 0) non faccio la somma
D 03
E 20 JSR 0300 Eseguo la somma tramite la subroutine
F 00
0210 03
1 Nosom 16 ASL 0,X } Eseguo lo shift del moltiplicando
2 00
3 36 ROL 1,X }
4 01
5 B5 LDA FF,X Vedo se ho il moltiplicatore = 0
6 FF
7 D0 BNE Loop Se no continuo il Loop
8 F1
9 4C JMP Monitor
A 22
B FE

```

Facciamo subito qualche commento chiarificatore.

Abbiamo messo a zero la locazione di memoria 0002 (istruzione alla 0200) per permetterci di eseguire lo shift del moltiplicando che era inizialmente un numero di 8 bit alla fine, dopo la moltiplicazione x 2, diventa un numero da 15 bit come mostrato nell'esempio della moltiplicazione binaria.

Nell'istruzione LSR FF,X all'indirizzo 020A facciamo notare che FF + X = 00 (essendo FF = -1 e X = 1) è l'indirizzo della locazione di memoria in pagina zero il cui contenuto viene spostato ("shiftato") di un bit a destra (nel Carry).

Rimane infine da spiegare lo shift di più byte. L'operazione di shift a sinistra di due parole avviene con l'istruzione ASL e ROL secondo quanto mostrato dalla fig. 9. Questa operazione è necessa-

ria perché, come abbiamo detto, ad ogni loop il moltiplicando cresce di un bit.

Per provare questo programma verifichiamo l'esattezza della seguente moltiplicazione: CD x 8D = 70E9.

Per fare ciò, riferendoci alla fig. 8 inseriremo 8D alla locazione 0000, CD alla 0001 e troveremo la parte bassa del risultato alla locazione 0003, la parte alta del risultato alla locazione 0004.

Verifichiamo ora l'esattezza di queste altre moltiplicazioni:

```

A9 x B1 = 74D9
02 x 04 = 0008
10 x 10 = 0100
34 x 97 = 1EAC
12 x 13 = 0156

```

E con questo abbiamo terminato questo altro importante blocco di software; nel prossimo capitolo esamineremo anche alcuni importanti aspetti dell'hardware che ci consentiranno di utilizzare la "porta utente" parallela da 8 bit presente sull'AMICO 2000A.

Sempre nell'ambito dell'hardware vi daremo tutte le informazioni necessarie per "manovrare" a vostro piacimento ogni singolo segmento delle cifre del display.

Un gioco:
"la corsa dei cavalli"

Ora vi presentiamo un giochetto originale che utilizza i tre segmenti paralleli di ogni cifra come se fossero dei "cavalli" in corsa. La probabilità di vincere che ha ogni cavallo è casuale e dipende in primo luogo dallo stesso programma; il bello è che si può intervenire direttamente "frustando" ogni cavallo per farlo correre di più. A attenzione però, se lo si frusta troppo (e anche questo non è prevedibile a priori in quanto deciso casualmente dal programma) potrebbe come si dice in gergo "rompere" e attardarsi nella corsa invece di accelerare, andateci piano allora.

Il programma viene dato come al solito nel codice oggetto e parte dalla locazione 0200. Si può giocare in tre; la frusta è rappresentata dai tasti:

per il cavallo in alto, per quello di mezzo e per quello in basso. Pigiando

ripetutamente ognuno di questi tasti dopo aver fatto partire il programma (e quindi la corsa) si accelera l'andatura del rispettivo cavallo. All'ultimo giro si accende un "1" sull'ultimo display a destra che rappresenta la barriera di arrivo, la corsa si ferma quando il primo cavallo tocca questa barriera.

Per ripartire bisogna premere il tasto RES e riportarsi alla locazione di partenza 0200.

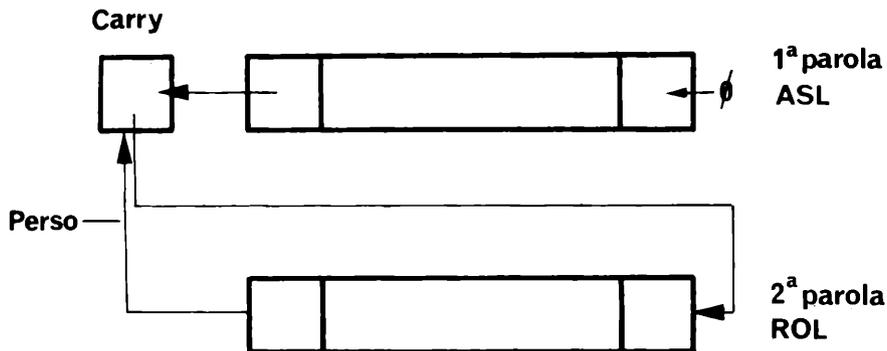


Fig. 49 - Combinazioni dell'uso delle due istruzioni ASL e ROL per riempire in modo automatico due successive locazioni di memoria con il risultato della moltiplicazione che cresce di un bit ad ogni loop. Si noti che il bit che esce a sinistra e va nel Carry (nella istruzione di ROL) viene perso perchè subito dopo sostituito con quello proveniente dall'istruzione ASL.

```

Programma della "Corsa dei cavalli"

0200 =D8 A2 13 BD D9 02 95 7C CA 10 F8 A9 89 8D 03 FD
0210 =A2 09 AD 00 B9 7C 00 84 FC 20 3B FF C8 CD 06 90
0220 =F3 20 EB FE A5 8F 30 E3 A2 03 CA 30 DE D6 86 D0
0230 =F9 86 99 A4 99 B6 83 B9 ED 02 35 7C 95 7C E8 96
0240 =83 B9 ED 02 49 FF 15 7C 95 7C E0 05 30 2B D0 06
0250 =A5 8F FD 1B D0 23 A2 02 38 B5 83 E9 06 95 83 CA
0260 =10 F6 A2 06 B5 7C 95 76 A9 80 95 7C CA D0 F5 C6
0270 =8F D0 06 A5 81 09 06 85 81 B9 89 00 F0 0A 20 C5
0280 =02 29 3C D0 1A 99 89 00 20 C5 02 29 38 85 9A B9
0290 =8C 00 30 0B 29 38 C5 9A B0 05 A9 FF 99 89 00 20
02A0 =EB FE AD FF A6 99 3D F0 02 F0 01 88 98 55 89 85
02B0 =9A 20 C5 02 38 29 01 65 9A 18 A6 99 75 8C 95 8C
02C0 =95 86 4C 2A 02 38 A5 92 65 95 65 96 85 91 A2 04
02D0 =85 91 95 92 CA 10 F9 60 00 80 80 80 80 80 80 80
02E0 =FF FF FF 80 80 80 00 00 00 80 80 80 08 FE BF F7
02F0 =01 02 04
  
```

SOLUZIONE DEGLI ESERCIZI DEL SESTO CAPITOLO.

- 1° Esercizio. La soluzione di questo problema viene data nel testo di questo stesso articolo.
- 2° Esercizio. Questa la soluzione al problema utilizzando le istruzioni note fino alla volta scorsa:

```

0200 A9 LDA #00 Azzeraimento del risultato
1 00
2 D8 CLD
3 A6 LDX $00 Il moltiplicando va nel registro indice X
4 00
5 F0 BEQ Fine
6 06
7 Loop 18 CLC
8 65 ADC $01 Sommo il moltiplicatore ad ogni loop
9 01
A CA DEX
B D0 BNE Loop Faccio un numero di loop pari al moltiplicando
C FA
D Fine 85 STA $02 Risultato a posto
E 02
F 4C JMP Monitor
0210 22
I FE
  
```

In questo programma si ha:
 Il 1° numero da moltiplicare nella locazione 0000.
 Il 2° numero da moltiplicare nella locazione 0001.

Il risultato nella locazione 0002.
 I numeri trattati da questo programma sono in esadecimale: i numeri che si possono moltiplicare sono quelli il cui prodotto è minore o uguale a 255.

3° Esercizio. Di seguito una delle soluzioni possibili:

```

0200 A2 LDX #00
1 00
2 Loop 8A TXA
3 95 STA 00,X
4 00
5 E8 INX
6 E0 CPX #$51
7 51
8 D0 BNE Loop
9 F8
A 4C JMP Monitor
B 22
C FE
  
```

USO DELLA PORTA UTENTE

Nell'AMICO 2000A c'è un integrato che gestisce le operazioni di input e output: si tratta del dispositivo 8255 il cui schema appare in Fig. 1.

Questo integrato dispone di tre porte ovvero di tre mezzi per comunicare col mondo esterno costituiti da 8 bit ciascuno. Di queste tre porte A, B e C, le prime due sono utilizzate per la gestione del display e della tastiera e sono controllate dal programma di monitor, la C invece è detta "PORTA UTENTE" perché utilizzabile a nostro piacimento. Si tratta solo di sapere come.

Ognuna di queste tre porte ha un indirizzo della locazione di memoria nella quale il monitor (per la A e B) o l'utilizzatore (per la C) scrive una ben determinata parola (ovvero 8 bit) che fa funzionare queste porte in un altrettanto ben determinato modo; vediamo quali sono gli indirizzi di memoria interessati dall'8255:

Indirizzo della Porta A = FD00

Indirizzo della Porta B = FD01

Indirizzo della PORTA C (PORTA UTENTE) = FD02

La porta utente dell'8255 può essere utilizzata previa programmazione del modo di funzionamento: per far ciò esiste un *registro di definizione del modo di funzionamento* che nel nostro caso ha indirizzo FD03.

Fisicamente la porta utente C sono nell'AMICO 2000A quegli otto capicorda posti in alto e in mezzo nella piastra sui quali sono presenti quegli otto bit paralleli (nel senso che si leggono o si scrivono allo stesso tempo) di cui abbiamo parlato.

Prima di passare ad esaminare come si usa il registro di definizione premettiamo che la porta C è divisa in due parti: la porta CH (H sta per High = alto) ovvero i bit 7, 6, 5, 4 e la porta CL (L sta

per Low = basso) ovvero i bit 3, 2, 1, 0 come mostra la Fig. 51.

Queste due parti della porta C hanno la particolarità di poter funzionare una in uscita e una in entrata (indifferentemente CH o CL) oppure tutte e due in uscita o in entrata; questo naturalmente dipende da ciò che abbiamo programmato cioè da cosa viene scritto nel registro che definisce il modo di funzionamento.

Detto questo vediamo subito che scrivere una parola nel registro di definizione del modo di funzionamento, ovvero all'indirizzo FD03, non significa altro se non determinare la funzione delle varie porte e cioè se la CPU in queste porte deve leggere un dato che le viene presentato (condizione della porta di input) o se essa stessa deve presentare un dato sulle porte (condizione di output).

La Fig. 52 rappresenta il registro del modo di funzionamento; notiamo che i bit sulle posizioni 2, 5, 6 e 7 sono "fissi" ovvero sono stati scritti così dal nostro programma di monitor perché l'8255 potesse lavorare nel progetto dell'AMICO 2000A in un certo modo.

Vediamo come usare (programmare) gli altri bit:

B4 corrisponde alla porta A: si scrive 0 in Output e 1 in Input

B1 corrisponde alla porta B: si scrive 0 in Output e 1 in Input

B3 corrisponde alla porta CH: si scrive 0 in Output e 1 in Input

B0 corrisponde alla porta CL: si scrive 0 in Output e 1 in Input

Come esempio nella Fig. 53 sono riportate le configurazioni utilizzate dal monitor che sono:

89 per eseguire la routine del display

99 per eseguire la routine della tastiera.

Per lavorare con la porta C si dovrà agire allora nel seguente modo:

1) Decido come usare la porta C scrivendo una parola di definizione alla locazione FD03 (decido cioè se utilizzarla come uscita o ingresso dati);

2) Scrivo (se la porta C è stata posta in Output) o leggo (se la porta C è stata posta in input) il dato nella locazione FD02 (che è appunto l'indirizzo relativo alla porta C).

È possibile usare la porta C indirizzando uno per uno, ovvero singolarmente, i bit della porta utente. Per far questo ci si serve soltanto del registro di definizione (non della locazione relativa alla porta C) e si procede come segue.

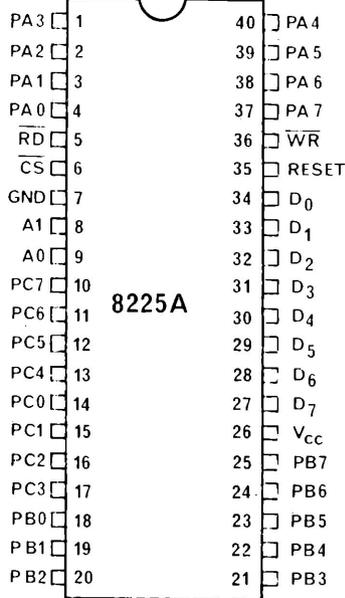
Prima di tutto si definisce il modo di funzionamento delle varie porte scrivendo una opportuna parola nella locazione FD03.

Poi e questa è una particolarità dell'integrato 8255, scrivo ancora nella locazione FD03 una configurazione (una parola) tale da permettermi l'operazione desiderata secondo la tabella che segue e con riferimento alla fig. 54.

B7 = 0 con ciò seleziono questo particolare modo di funzionamento che mi permette di usare la stessa locazione FD03.

B6 - B5 - B4 = 0 o 1 è indifferente nulla cambia.

CONFIGURAZIONE
PIEDINI 8255A



FUNZIONE DEI PIEDINI

D ₇ -D ₀	Bus dati bidirezionale
RESET	Ingresso di azzeramento
CS	Selezione
RD	Ingresso impulso di lettura
WR	Ingresso impulso di scrittura
A ₀ , A ₁	Indirizzi delle porte
PA ₇ -PA ₀	Linee di uscita della Porta A
PB ₇ -PB ₀	Linee di uscita della Porta B
PC ₇ -PC ₀	Linee di uscita della Porta C

B₃ - B₂ - B₁ = 000 = Uscita sul bit 0
 001 = Uscita sul bit 1
 010 = Uscita sul bit 2
 011 = Uscita sul bit 3
 100 = Uscita sul bit 4
 101 = Uscita sul bit 5
 110 = Uscita sul bit 6
 111 = Uscita sul bit 7

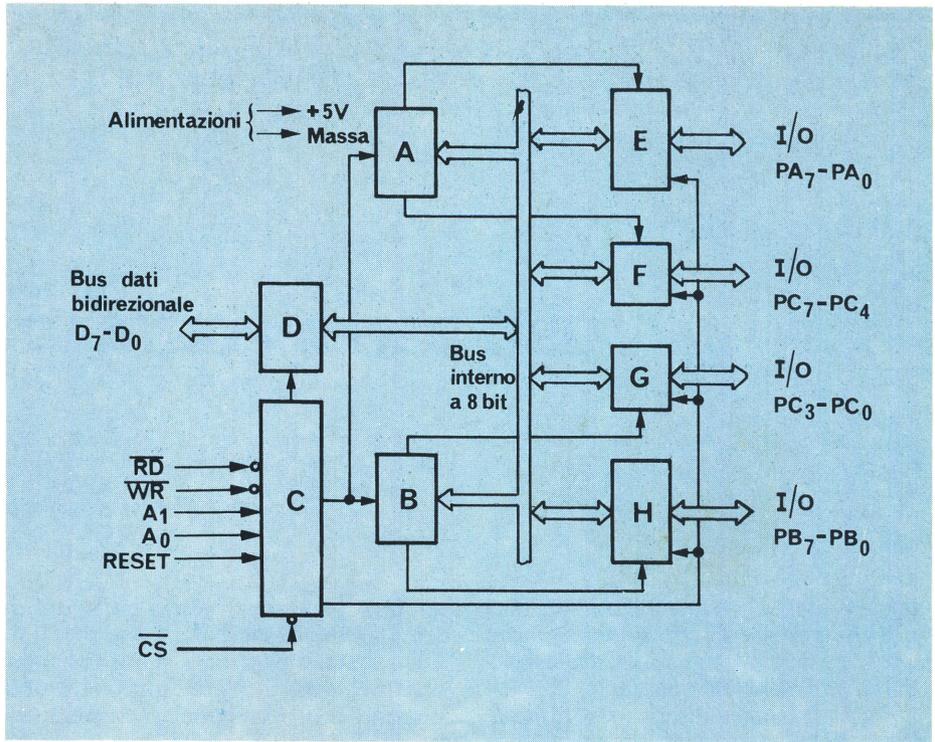
Selezione del bit della porta C
 su cui si intende operare.

B₀ = 1 se si vuole scrivere un 1 sul bit scelto.

B₀ = 0 se si vuole scrivere uno 0 sul bit scelto.

Facciamo, per passare alla pratica, il seguente esempio. Si voglia realizzare un programma che generi un impulso al bit 6 della porta C.

Innanzitutto programmeremo il modo di funzionamento in maniera che la porta CH (perché il 6° bit è in questa parte della porta C) si trovi in condizione di Output. Scriveremo allora, ad esempio il dato 81 nella locazione di memoria FD03 ponendoci nelle condizioni illustrate dalla Fig. 55.



A = Controllo gruppo A; B = Controllo gruppo B; C = Logica di controllo lettura-scrittura; D = Buffer dati; E = Porta A (8 bit); F = Porta CH (parte alta 4 bit); G = Porta CL (parte bassa 4 bit); H = Porta B (8 bit).

Fig. 50 - Schema a blocchi dell'integrato 8255A.

Ora scriverò nella stessa locazione FD03, secondo quanto abbiamo spiegato poc'anzi, prima una parola che metta a 1 il bit 6, poi (e qui il tempo di durata dell'impulso dipende dal tempo di esecuzione della istruzione) una parola che metta a 0 lo stesso bit 6 della porta C: si faccia riferimento alla Fig. 56.

Compreso tutto ciò posso scrivere il programma come segue:

Da notare che in questo programma non si effettua alcuna operazione di "store" all'indirizzo della porta C (FD02), infatti in questo modo di funzionamento, come abbiamo detto, si opera esclusivamente con il registro di definizione del modo di funzionamento alla locazione FD03.

Con questo programma si ottiene un impulso di circa 10 μS (cioè il tempo di

```

0200      A9  LDA #$81
1         81
2         8D  STA $FD03
3         03
4         FD
5         A9  LDA #$0D
6         0D
7         8D  STA $FD03
8         03
9         FD
A         A9  LDA #$0C
B         0C
C         8D  STA $FD03
D         03
E         FD
F         4C  JMP Monitor
0210      22
1         FE
    
```

Definizione del modo di funzionamento

Metto a 1 il bit 6 (Set del bit 6)

Metto a 0 il bit 6 (Clear del bit 6)

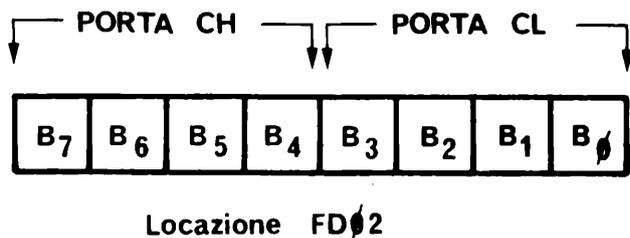


Fig. 51 - Configurazione della porta utente C all'indirizzo FD02.

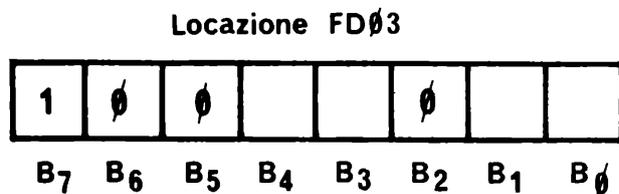


Fig. 52 - Locazione FD03 del registro del modo di funzionamento.

durata di esecuzione delle istruzioni) al bit 6 della porta C ogni volta che facciamo partire il programma.

Se si desidera ottenere un tempo di durata dell'impulso più lungo è necessario inserire un loop di ritardo tra il Set (portare a 1) e il Clear (portare a 0) del bit. Se si vuole poi che l'impulso sia ripetitivo si utilizza l'istruzione di JMP; vediamo di seguito il programma arricchito di queste altre prestazioni:

Note: 1) Dopo il primo decremento il contenuto di X è = FF (infatti $0 - 1 = -1 = FF$); 2) Continuo a decrementare fino a che $X = 0$. Questo avviene dopo 256 decrementi. Durante questo periodo di tempo il bit 6 è a 1; 3) Nota bene: entriamo in questo Loop con $X = 0$ (risultato del loop precedente).

La durata di ogni semionda si calcola dal tempo impiegato dal microprocessore

ad eseguire le istruzioni DEX e BNE ($4\mu S$ in totale) moltiplicato per il numero dei loop (256); allora $4 \times 256 = 1,024 \text{ mS}$, cioè una frequenza di circa 500 Hz.

Si può naturalmente diminuire il tempo di ciclo caricando un valore minore da decrementare o aumentarlo servendosi di due loop concatenati che fanno uso dei due registri X e Y:

```
LDY #$yy (yy = numero che
LDX #$00 determina il
Loop DEX ritardo)
BNE Loop
DEY
BNE Loop
```

Il ritardo generato da questa routine è uguale a $1,024 \text{ msec}$ moltiplicato per il numero caricato nel registro Y all'inizio della routine; il tempo di ritardo massimo è quindi di circa 262 millisecondi ($1,024 \times 256$).

Facendo girare il programma appena descritto sull'AMICO 2000A è possibile vedere l'onda quadra generata se si è in possesso di un oscilloscopio collegando il probe al piedino 6 della porta utente.

Se viene inserito un ritardo maggiore nel programma possiamo osservare con un semplice voltmetro (tester commutato sui 10 Vcc fondo scala) una tensione che varia periodicamente fra il piedino 6 e massa.

Una volta che si è in possesso delle tecniche per lavorare sulla porta utente, non esistono più problemi a collegarsi all'esterno a qualsiasi altro dispositivo elettronico in grado di presentare alla sua uscita livelli alti e bassi di tensione o di saperli leggere.

0200	LDA	#S81		
1	81		} Definizione del modo di funzionamento	
2	8D	STA FD03		
3	03			
4	FD			
5	A2	LDX #\$00	} Preparo il loop di ritardo caricando 00 nel registro X	
6	00			
7	Onda	A9	} Set del bit 6	
8		0D		
9		8D		
A		03	} Decremento X ⁽¹⁾	
B		FD		
C	Loop1	CA	} Salto al Loop 1 quando X ≠ 0 ⁽²⁾	
D		D0		
E		FD		
F		A9	} Clear del bit 6	
0210		0C		
1		8D		
2		03		
3		FD	} Ripeto la routine di ritardo ⁽³⁾	
4	Loop2	CA		
5		D0		
6		FD		
7		4C	} Ritorno all'inizio del set bit 6 per generare un'onda quadra continua	
8		07		
9		02		

Le Subroutine del monitor

Nel programma di Monitor, quello che permette al microelaboratore di funzionare, sono impiegate alcune routine che possono essere convenientemente utilizzate nei vostri programmi.

Vediamo ora quali sono e come si usano.

Routine di TASTO ATTIVO
Routine di IDENTIFICAZIONE DEL TASTO

Routine 1 di RINFRESCO DISPLAY
Routine 2 di RINFRESCO DISPLAY

A) Routine di tasto attivo - Ha come indirizzo di partenza la locazione FEEB. Si esce dalla subroutine con 00 in Accumulatore se non vi è alcun tasto premuto e con 01 in Accumulatore se c'è qualche tasto premuto.

Questa subroutine si usa per far succedere qualcosa quando si preme un tasto qualunque. Alla fine di questa subroutine si fa un test sul contenuto dell'accumulatore utilizzando una istruzione di BEQ o BNE.

Normalmente però è più utilizzata la routine che segue, di identificazione del

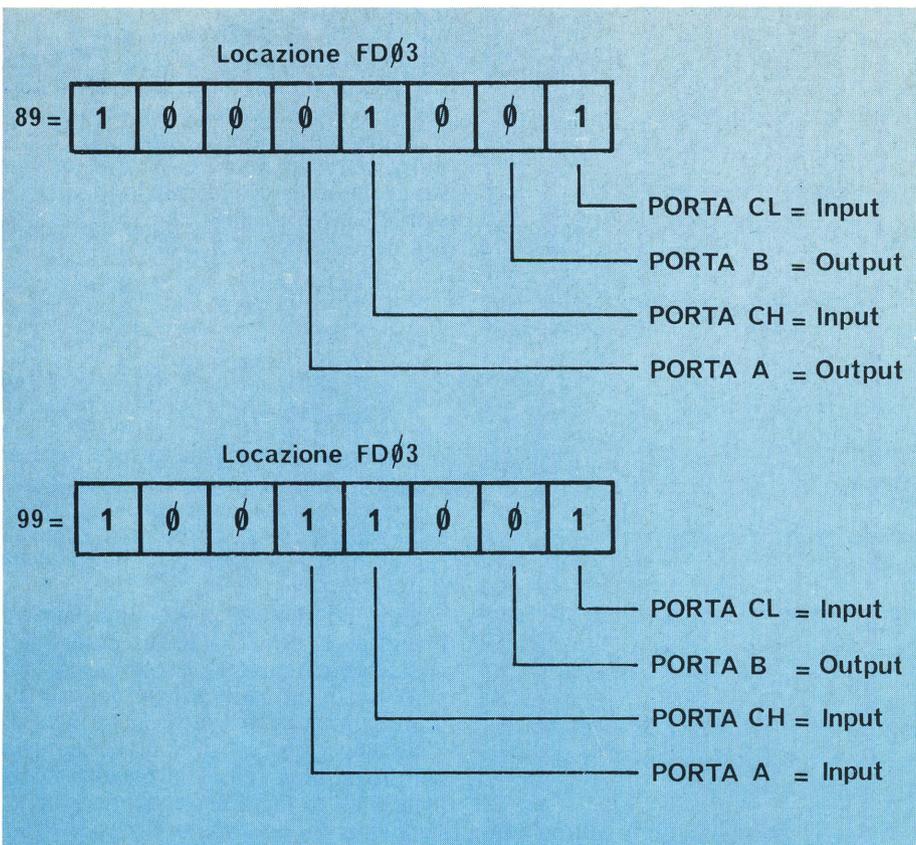


Fig. 53 - Due configurazioni del registro del modo di funzionamento utilizzate dal programma di monitor per eseguire la routine del display e quella della tastiera.

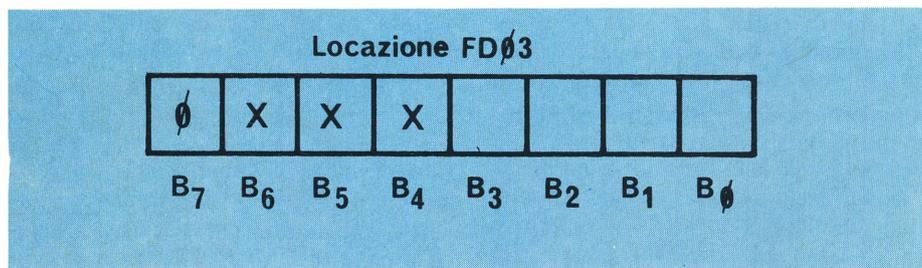
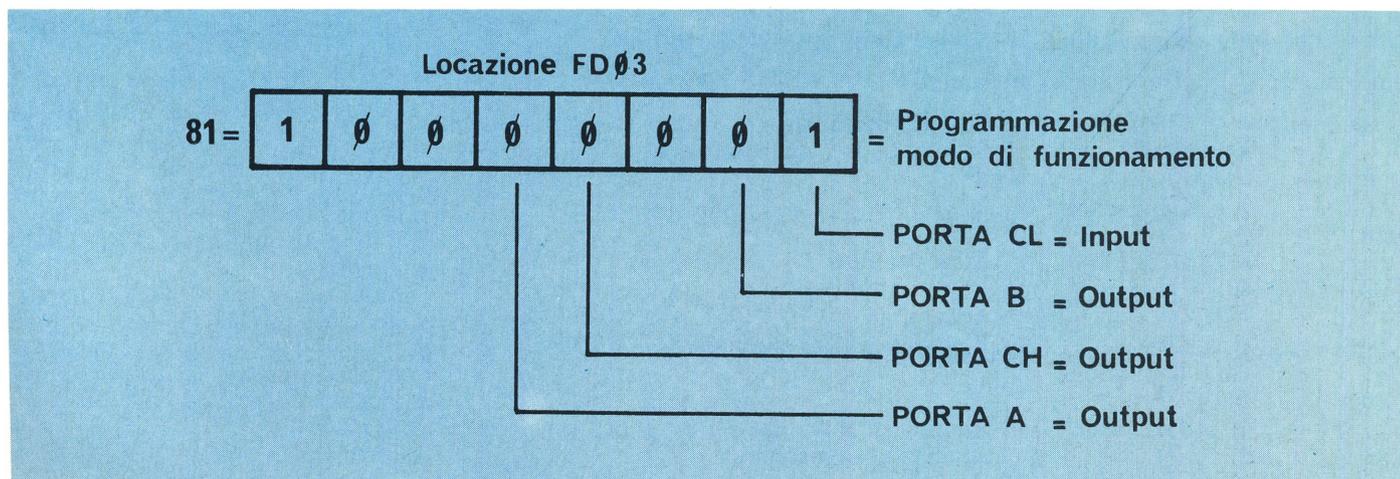


Fig. 54 - Uso del registro del modo di funzionamento per usare la porta C indirizzando i suoi bit uno per uno.



tasto, perché permette di assegnare ad ogni tasto una determinata funzione da eseguire.

B) *Routine di identificazione del tasto* - Ha come indirizzo di partenza la locazione FF57. Prima di entrare in questa routine è bene inizializzare l'integrato 8255 per essere certi che esso sia in grado di trattare i segnali così come la routine di identificazione del tasto richiede: per far questo basta caricare il numero 99 nel registro di definizione del modo di funzionamento.

Si possono usare le seguenti istruzioni:

```
A9 LDA #99
99
8D STA $FD03
03
FD
```

Si esce dalla routine con il valore del tasto contenuto in Accumulatore secondo quanto riportato nella Tabella 17. Nella tabella sono riportate due serie di valori diversi per ogni tasto che dipendono dal modo di funzionamento in cui è stato posto il processor: decimale (mediante l'istruzione SED) o esadecimale (mediante l'istruzione CLD). Se non vi è alcun tasto premuto o più di un tasto premuto il contenuto dell'accumulatore vale 21 se in funzionamento decimale o 15 se in funzionamento esadecimale. Per usare questa routine si fa una comparazione in uscita con il numero corrispondente al tasto che interessa seguita da una istruzione di Branch.

C) *Routine 1 rinfresco display* - Ha come indirizzo di partenza la locazione FF06.

Durante l'esecuzione la routine preleva il contenuto esadecimale della locazione di memoria indirizzata dalle loca-

Fig. 55 - Programmazione del modo di funzionamento per porre la porta CH in condizione di Output.

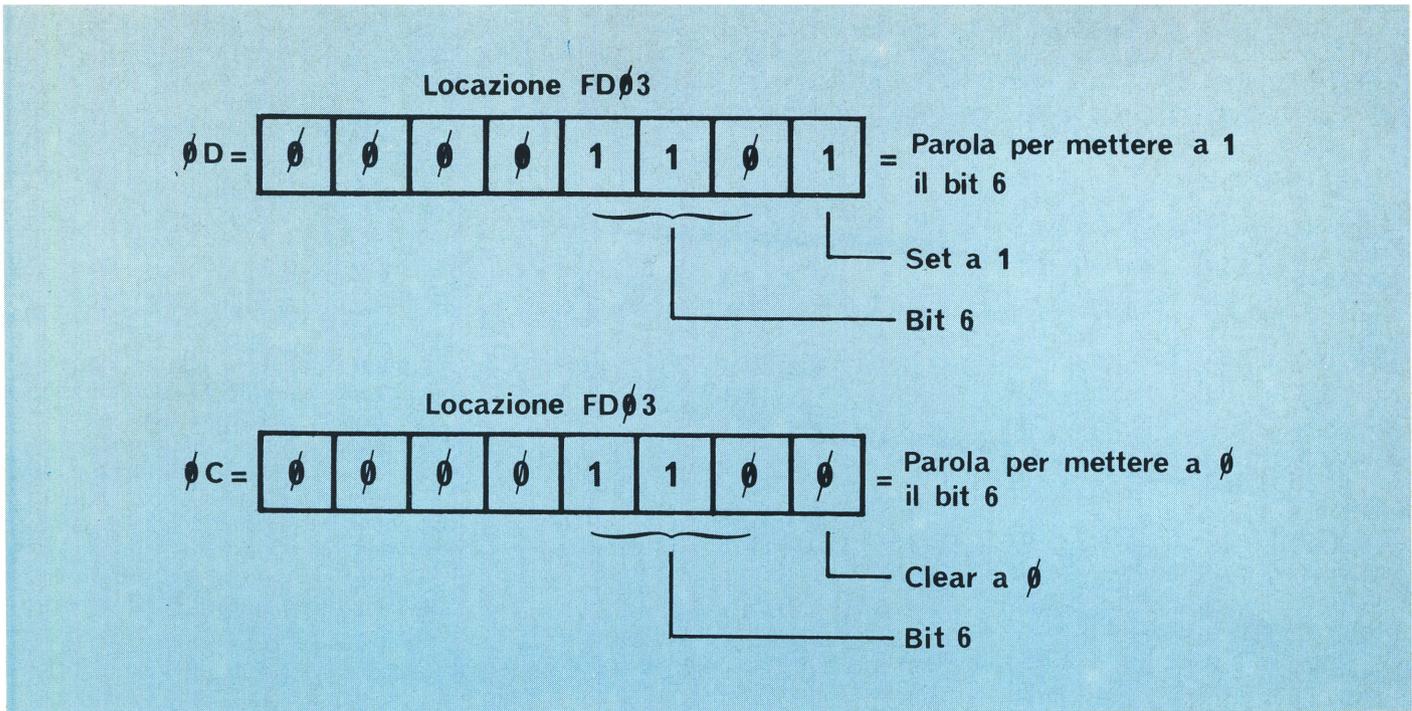


Fig. 56 - Le due parole 0D e 0C scritte alla locazione FD03 servono a portare a 1 o a 0 il bit 6 della porta C.

zioni di pagina base 00FA (parte bassa) e 00FB (parte alta). In altre parole il contenuto delle locazioni 00FA e 00FB diventa l'indirizzo di una locazione di cui si vuole vedere il contenuto.

Per utilizzare convenientemente questa routine è evidente che è necessario caricare in precedenza queste locazioni di memoria.

Per utilizzare in pratica questa routine facciamo l'esempio di un programma che carica sul display indirizzi la locazione 0202. È chiaro che sul display apparirà il contenuto di memoria relativo all'indirizzo 0202; vogliamo poi che questo contenuto sia 04 (vedi Listing a lato).

Attenzione! A questo punto il programma sembrerebbe terminato, ma se si facesse partire l'esecuzione vedremo un lampo di luce nel display e poi più niente.

Infatti per mantenere la configurazione di cifre fissa sul display è necessario "rinfrescare" continuamente il display stesso: si completa perciò il programma con una istruzione di JMP che fa ripetere in continuazione la subroutine:

```
020E 4C JMP 020B
F 0B
0210 02
```

Ora si può far partire il programma.

Vediamo ora in particolare come funziona questa subroutine: viene prelevato il contenuto delle locazioni di memoria puntate da 00FA e 00FB che viene trasportato nella locazione 00F9. Successivamente il contenuto delle locazioni di memoria 00F9, 00FA e 00FB viene trasferito sul display come mostra la Fig. 57.

Listing

0200	A9	LDA #S02	Carico 02 nell'accumulatore
1	02		
2	85	STA \$FA	Memorizzo 02 alla locazione 00FA
3	FA		
4	85	STA \$FB	Memorizzo 02 alla locazione 00FB
5	FB		
6	A9	LDA #S04	Carico 04 in ACC
7	04		
8	8D	STA \$0202	Memorizzo 04 nella locazione 0202
9	02		
A	02		
B	20	JSR FF06	Salto alla subroutine 1 di rinfresco del display
C	06		
D	FF		

Tabella 17 - Dati presenti in Accumulatore al ritorno della subroutine di identificazione dei tasti secondo il tasto premuto.

TASTO	DATO ¹	TASTO	DATO ²	DATO ³
0	00	A	10	A
1	01	B	11	B
2	02	C	12	C
3	03	D	13	D
4	04	E	14	E
5	05	F	15	F
6	06	AD	16	10
7	07	DA	17	11
8	08	↑	18	12
9	09	RUN	19	13
		REG	20	14
		NESSUN TASTO	21	15

1 = Funzionamento decimale e esadecimale
2 = Funzionamento decimale
3 = Funzionamento esadecimale

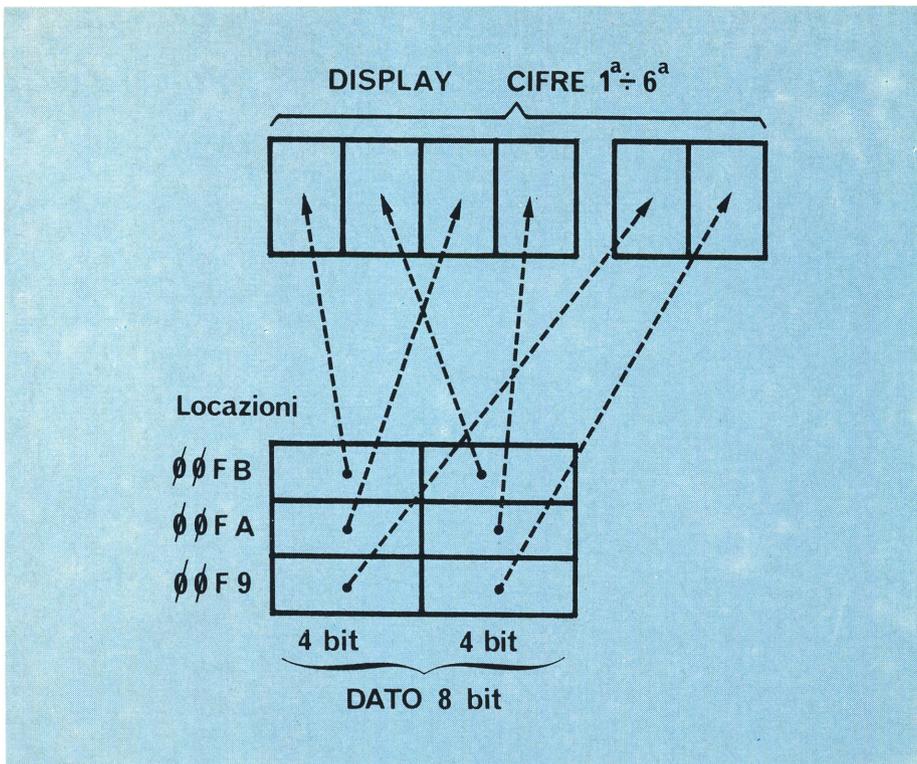


Fig. 57- Corrispondenza fra contenuto delle locazioni in pagina base F9, FA e FB e le sei cifre del display.

In questa stessa routine di rinfresco display si può entrare alla locazione FF0C invece che alla FF06, saltando in questo modo il prelevamento del dato e la modifica del contenuto di 00F9. In questo modo possiamo visualizzare ciò che vogliamo direttamente sui sei digit del display.

D) *Routine 2 di rinfresco display* - Ha come indirizzo di partenza la locazione FF7E.

Questa routine porta sul display il contenuto delle sei successive locazioni di memoria così come segue:

008F	1 ^a cifra a sinistra del display
0090	2 ^a cifra
0091	3 ^a cifra
0092	4 ^a cifra
0093	5 ^a cifra
0094	6 ^a cifra

Il caricamento di queste locazioni di memoria deve essere fatto tenendo conto della corrispondenza esistente tra i singoli bit e i segmenti delle cifre del display secondo quanto riportato nella Fig. 58. Il bit a 1 corrisponde a segmento acceso.

Nella tabella 18 inoltre è riportato un

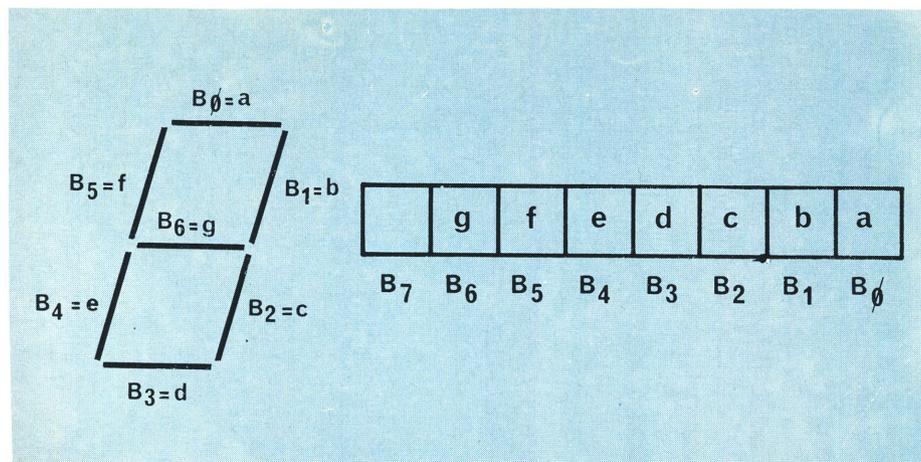


Fig. 58 - Corrispondenza fra i vari segmenti di ogni cifra e i bit della locazione di memoria ad essa relativa.

esempio di alfabeto che può essere visualizzato dall'AMICO 2000A. È chiaro che ognuno poi potrà scrivere i segni più strani scrivendo delle parole opportune che mettano a 1 i bit corrispondenti segmenti che si vuole far accendere. Si fa notare infine che il bit 7 non viene utilizzato.

Vediamo subito un programma esemplificativo che presenti sul display: -ASEL-.

Facciamo riferimento alla Tabella 2 e carichiamo i dati:

Locazione	Dato	Rappresentazione
008F	C0	-
0090	F7	A
1	ED	S
2	F9	E
3	B8	L
4	C0	-

Scriviamo ora il seguente semplice programma di uso generale che permette di visualizzare ciò che abbiamo scritto nelle diverse locazioni:

```

0200      20 JSR FF7E
          Salto alla subroutine di
          rinfresco
1         7E
2         FF
3         4C JMP 0200
          Eseguo in continuazione
          la subroutine precedente
4         00
5         02
  
```

Tabella 18 - Esempio di alfabeto per display a sette segmenti e dati relativi per generare quel carattere.

Numeri	Lettere	
1 = 86	A = F7	n = 54
2 = DB	b = 7C	o = BF
3 = CF	C = B9	P = F3
4 = E6	d = 5E	q = 67
5 = ED	E = F9	r = 50
6 = FD	F = F1	S = ED
7 = 87	G = BD	t = 78
8 = FF	H = F6	U = BE
9 = EF	I = 86	Y = EE
0 = BF	J = 9E	
- = C0	L = B8	
		Cifra
		Spenta = 00

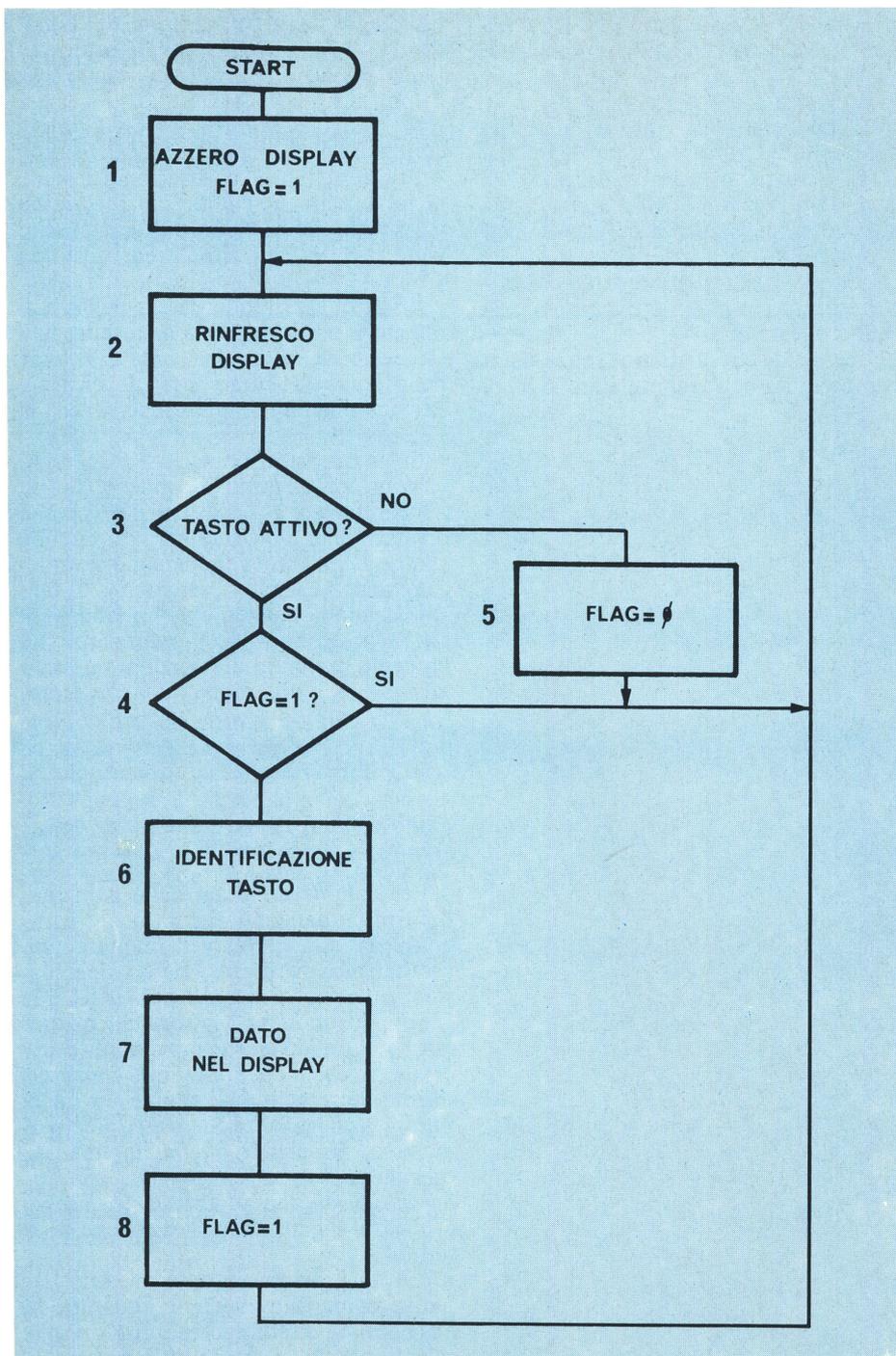


Fig. 59 - Flow chart del programma per l'uso delle routine di identificazione del tasto e di tasto attivo.

Vediamo ora un programma che mostri l'uso della routine di identificazione del tasto.

0200	F8	SED	Metto la macchina in modo decimale
1	20	JSR FF57	Salto alla routine di identificazione del tasto
2		57	
3		FF	
4	C9	CMP # \$05	Confronto il contenuto dell'ACC con 05
5		05	
6	D0	BNE Loop	Se non è uguale a 05 salto al Loop
7		F9	
8	4C	JMP Monitor	Se è uguale a 05 passo il controllo del micro al monitor (il display si accende)
9		22	
A		FE	

Facendo partire questo programma il display rimarrà spento finché non si preme il tasto 5, solo in questo caso si salta il Monitor che riprende il controllo del display facendolo accendere.

Ora esamineremo un programma più complesso e completo che mette in evidenza l'uso delle due routine di tasto attivo e di identificazione del tasto. Con questo programma vogliamo che il display mostri tutte le sue cifre a zero e che sul display compaia il valore dei tasti da 0 a F quando questi vengano premuti. Prima di dare la lista delle istruzioni è opportuno esaminare il flow chart per meglio comprenderne la costruzione (Fig. 59).

Blocco 1: Si azzerò il display e si pone un FLAG a 1, con questa operazione in pratica si memorizza il dato 01 in una certa locazione di memoria: lo scopo, come vediamo più avanti, è quello di controllare se in questa locazione di memoria c'è 1 o 0, cioè se è stata modificata in qualche modo.

Blocco 2: Si salta alla routine di rinfresco del display.

Blocco 3: Da questo blocco e fino al 5 il programma che è stato scritto ha la funzione di permettere da una parte che l'uso del tasto RUN che fa partire il programma non venga identificato, (cioè nel pur brevissimo tempo durante il quale questo tasto rimane premuto il programma continua a rinfrescare il display azzerato), dall'altra che ogni tasto premuto venga identificato una sola volta come spieghiamo più avanti.

Quando allora entriamo nel blocco 3 con il tasto RUN premuto alla domanda "TASTO ATTIVO?" si esce con SI e si entra nel blocco 4.

Blocco 4: Qui ci si chiede se il FLAG è a 1: tutte le volte che passiamo al blocco 3 mentre RUN rimane premuto il FLAG sarà ancora a 1 in quanto niente lo ha modificato. In questo caso si esce dal SI e si torna alla routine di rinfresco del display che mostra ancora tutti zeri.

Blocco 5: Passando attraverso il blocco 3 dopo aver rilasciato il tasto RUN nessun tasto sarà attivo e quindi si uscirà dal NO entrando nel blocco 5 che pone il FLAG a 0 e torna a rinfrescare il display.

A questo punto, posto il FLAG a 0, quando andiamo a premere uno dei tasti della tastiera esadecimale entriamo nel blocco 3, usciamo con SI entrando nel blocco 4 e da questo usciamo con NO entrando nel blocco successivo.

Blocco 6: Si entra nella routine di identificazione del tasto.

Blocco 7: Il dato (cioè il valore dello stesso tasto) viene visualizzato nel display e posto nella prima cifra a destra.

Blocco 8: Il FLAG viene ripristinato a 1 per far in modo che il tasto identificato e che abbiamo premuto (ad esempio il 5) venga riconosciuto una sola volta. In pratica infatti anche nel pur

breve tempo durante il quale il tasto rimane premuto il programma gira numerosissime volte, se non mettessimo ancora il FLAG a 1 il numero 5 corrispondente al tasto premuto riempirebbe tutte le sei cifre del display invece dell'ultima a destra. Quando però il tasto 5 viene rilasciato ecco che il FLAG va nuovamente a zero e il programma è pronto ad accettare un nuovo tasto.

In pratica le cifre che inseriremo, ovvero i tasti che premeremo entreranno dall'ultima cifra a destra e scorreranno fino alla prima a sinistra fino a sostituire tutti gli zeri del display.

Cercate di comprendere bene l'uso del FLAG perché questo artificio viene usato molto di frequente nella compilazione dei programmi.

Ora possiamo scrivere il programma; badate che questo programma contiene diversi passaggi interessanti dal punto di vista ottimizzazione del numero delle istruzioni impiegate e va quindi studiato con attenzione.

Per meglio comprendere tutti i passaggi di questo programma esaminiamo di seguito queste note:

1) Uscendo da questa routine senza alcun tasto premuto nell'accumulatore si ha 0,

uso questo stesso 0 per azzerare il FLAG. Se invece c'è qualche tasto premuto continua l'esecuzione del programma e si va a controllare il FLAG.

2) Carico il registro X con il valore contenuto nella locazione 0000 (il FLAG).

3) Questo blocco di istruzioni permette di far entrare nell'ultima cifra a destra il tasto premuto: viene eseguito quattro volte perché ogni cifra è composta da 4 bit (es.: 5 = 0101).

4) In questo modo carico nell'Accumulatore la nuova parola da inserire nella locazione di pagina base F9 (cioè nelle ultime due cifre a destra del display). Per una miglior comprensione si veda la Fig. 60.

5) Le due istruzioni CLC e BCC equivalgono ad un salto incondizionato, le abbiamo usate al posto del JMP perché diversamente il programma non sarebbe stato rilocabile (ovvero sarebbe stato legato alla locazione di partenza 0?00).

Facciamo notare che saltando ad IND2 (locazione 0?06) realizziamo un notevole risparmio di istruzioni ripassando su passi di programma già scritti.

0200	A9	LDA	#\$00	Carico 00 in Acc.
1	00			
2	85	STA	\$FB	} Azzero il display
3	FB			
4	85	STA	\$FA	
5	FA			
6	IND 2	85	STA \$F9	
7	F9			
8	A9	LDA	#\$01	} FLAG = 1
9	01			
A	IND 1	85	STA \$00	
B	00			
C	LOOP	20	JSR SCANS	Rinfresco display
D	0C			
E	FF			
F	20	JSR	TESTAS	Salto alla routine di tasto attivo (MONITOR)
0210	EB			
1	FE			
2	F0	BEQ	IND 1	Se ACC = 0 non vado avanti, ma azzero il FLAG (1)
3	F6			
4	A6	LDX	\$00	Vedo se il FLAG = 0 (2)
5	00			
6	D0	BNE	LOOP	Se non è uguale a 0 torno in Loop diversamente continuo il program.
7	F4			Funzionamento in esadecimale
8	D8	CLD		
9	A9	LDA	#\$99	
A	99			
B	8D	STA	CONTR	Inizializzazione dell'8255
C	03			
D	FD			
E	20	JSR	TASTO	Salto alla subroutine di identificazione del tasto e torno col valore del tasto premuto in ACC.
F	57			
0220	FF			Uso Y come contatore
1	A0	LDY	#04	
2	04			
3	LOOP 1	06	ASL \$F9	} Eseguo 4 volte lo shift per far entrare il tasto premuto nel display (3)
4	F9			
5	26	ROL	\$FA	
6	FA			
7	26	ROL	\$FB	
8	FB			
9	88	DEY		
A	D0	BNE	LOOP 1	
B	F7			
C	05	ORA	\$F9	Eseguo l'operazione di OR fra il contenuto di F9 e l'accumulatore (4)
D	F9			
E	18	CLC		(5)
F	90	BCC	IND 2	Rimetto a posto il digit meno significativo e vado a porre il FLAG = 1
0230	D5			

L'Interrupt

Per completare il set delle istruzioni del 6502 rimangono ancora pochi particolari da esaminare; vediamo ora una caratteristica hardware che ha ripercussioni anche nel software: l'INTERRUPT.

Supponiamo che il nostro calcolatore stia percorrendo un suo programma principale, che stia per esempio contando quante volte si apre e chiude un interruttore collegato ad una sua linea di ingresso. Supponiamo che, mentre sta eseguendo questo lavoro debba, contemporaneamente, tenere acceso un display a 6 cifre aggiornando il dato che vi è scritto.

Il vero problema è dato appunto dal "contemporaneamente" in quanto la macchina in effetti può eseguire un solo calcolo per volta: deve perciò passare in continuazione dall'esecuzione di un programma all'esecuzione di un altro per farli avanzare di pari passo tutti e due.

Questo passaggio è possibile se si interrompe periodicamente il lavoro della CPU tramite un temporizzatore esterno, che fornisce ad un piedino della CPU un segnale ad onda quadra di interruzione.

Il processor quando riceve questo segnale di interruzione, abbandona l'esecuzione del programma principale per andare a servire le necessità del programma secondario.

Un altro esempio di interruzione può essere quello di un interruttore di fine corsa che in chiusura deve interrompere qualsiasi calcolo stia eseguendo la CPU perché diversamente c'è un pistone che

va a sbattere rovinando la macchina controllata. Alla chiusura del fine corsa allora viene generato il segnale di interruzione e la macchina va immediatamente ad eseguire il programma che contiene i vari controlli per l'arresto del pistone.

Da tutto ciò ci si può rendere conto dell'importanza fondamentale di questo segnale che permette di controllare fenomeni fra loro asincroni e completamente indipendenti l'uno dall'altro.

Tipi di Interrupt

Il microprocessore 6502 ha due diversi tipi di ingresso di interrupt tramite piedini marcati con le sigle $\overline{\text{IRQ}}$ e $\overline{\text{NMI}}$.

La differenza sostanziale fra questi due piedini è che $\overline{\text{IRQ}}$ è sensibile al livello di ingresso, vale a dire che si genera una interruzione se il piedino $\overline{\text{IRQ}}$ va a 0 Volt mentre $\overline{\text{NMI}}$ è sensibile ai fronti vale a dire che si genera una interruzione se su questo piedino si presenta un fronte di discesa, cioè se la sua tensione passa da 5 V a 0 Volt e solo a questo passaggio (vedi Fig. 61).

In pratica il controllo dello stato di questi due piedini viene fatto in continuazione dalla CPU e in particolare prima della esecuzione di ciascuna istruzione. Nota Bene: per il piedino $\overline{\text{NMI}}$ il fronte di discesa viene memorizzato da un circuito interno che permette il funzionamento sul solo fronte.

Un'altra differenza fra i due tipi di interrupt è che mentre l'interruzione generata su $\overline{\text{IRQ}}$ può essere mascherata (ignorata dalla CPU), la CPU serve sempre quella che nasce su $\overline{\text{NMI}}$.

Il 3° bit dello Status (bit I) serve proprio a bloccare la interruzione generata su $\overline{\text{IRQ}}$. Se viene posto a 1 da programma, l'interruzione non passa, se viene posto a 0 l'interrupt è abilitato.

Normalmente all'accensione della macchina di ha I = 1.

Per il bit I esistono due istruzioni specifiche: CLI (Clear Interrupt) codice op. 58, che pone I = 0.

SEI (Set Interrupt) codice op. 78, che pone I = 1.

Il bit I viene anche automaticamente messo a 1 (cioè viene disabilitato l'interrupt) dalla CPU, quando essa va a servire una interruzione; questo perché diversamente la CPU continuerebbe a servire la stessa interruzione (ricordiamoci che siamo sensibili al livello) (vedi Fig. 62).

La messa a 0 di I avviene invece tramite l'istruzione RTI (ritorno da Interrupt) codice op. 40 o tramite la CLI.

Come già detto invece l'interruzione su $\overline{\text{NMI}}$ nasce solo su un fronte di discesa su questo piedino e non può venire bloccata dalla CPU; perché avvenga una successiva richiesta di interruzione il

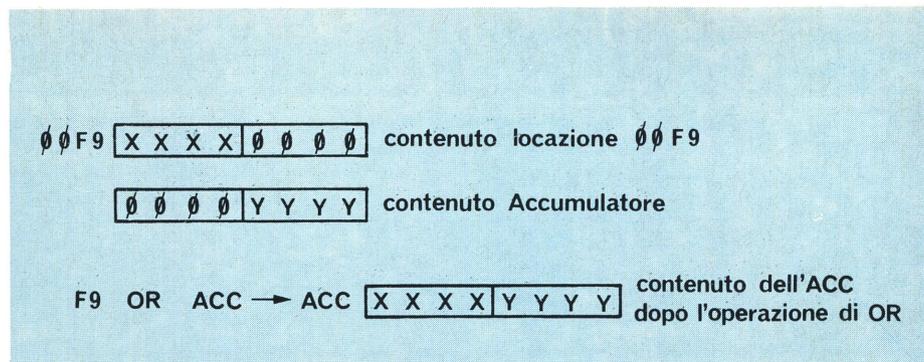


Fig. 60 - Operazione di OR fra il dato contenuto nella locazione 00F9 e quello contenuto in accumulatore.

piedino $\overline{\text{NMI}}$ deve tornare alto, quindi ancora basso.

Il funzionamento della interruzione

Come risponde la CPU ad una interruzione?

Essa interrompe l'esecuzione del programma che sta eseguendo, salva nello Stack lo Status e il PC che vi era in

macchina al momento dell'interruzione, questo per poter poi tornare al programma principale al punto dove era stato arrestato. La CPU quindi preleva da locazione fissa della memoria il punto da cui parte la routine di gestione dell'interruzione (ovvero il nuovo programma da eseguire) che finisce sempre con una istruzione di RTI. Con RTI viene eseguita la procedura esattamente inversa ripristinando i valori (presenti nello Stack)

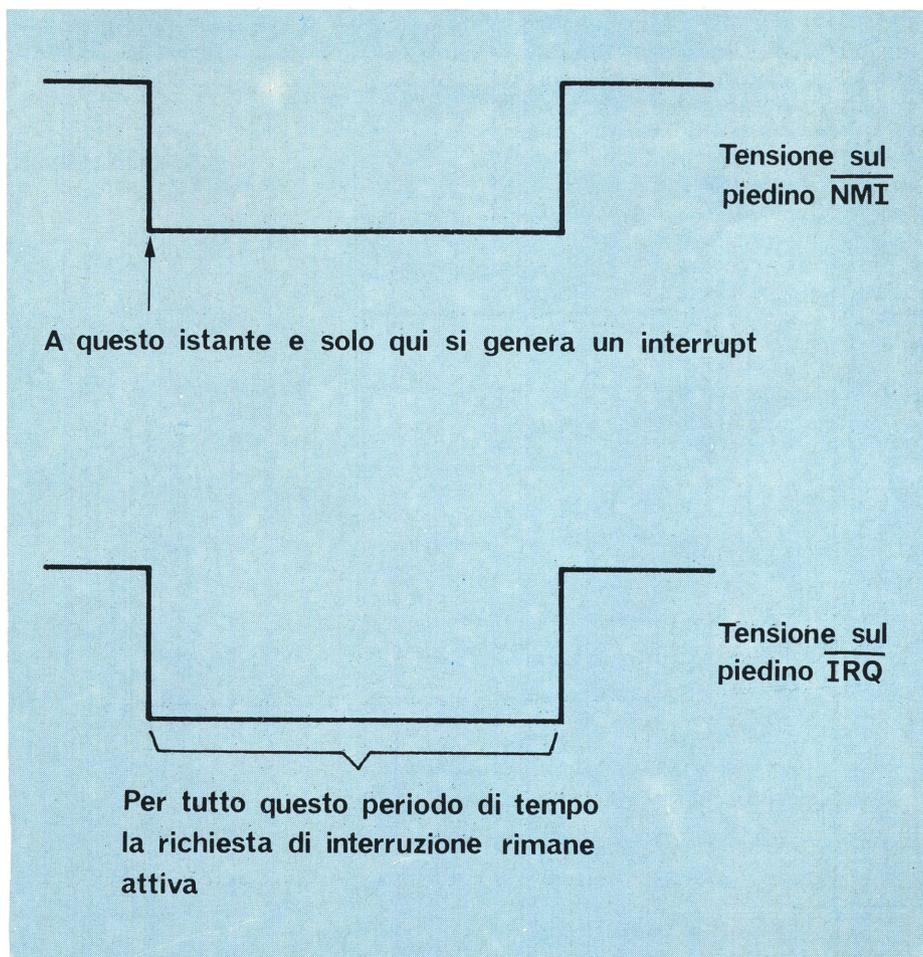


Fig. 61 - Differenza fra le richieste di interruzione nel 6502.

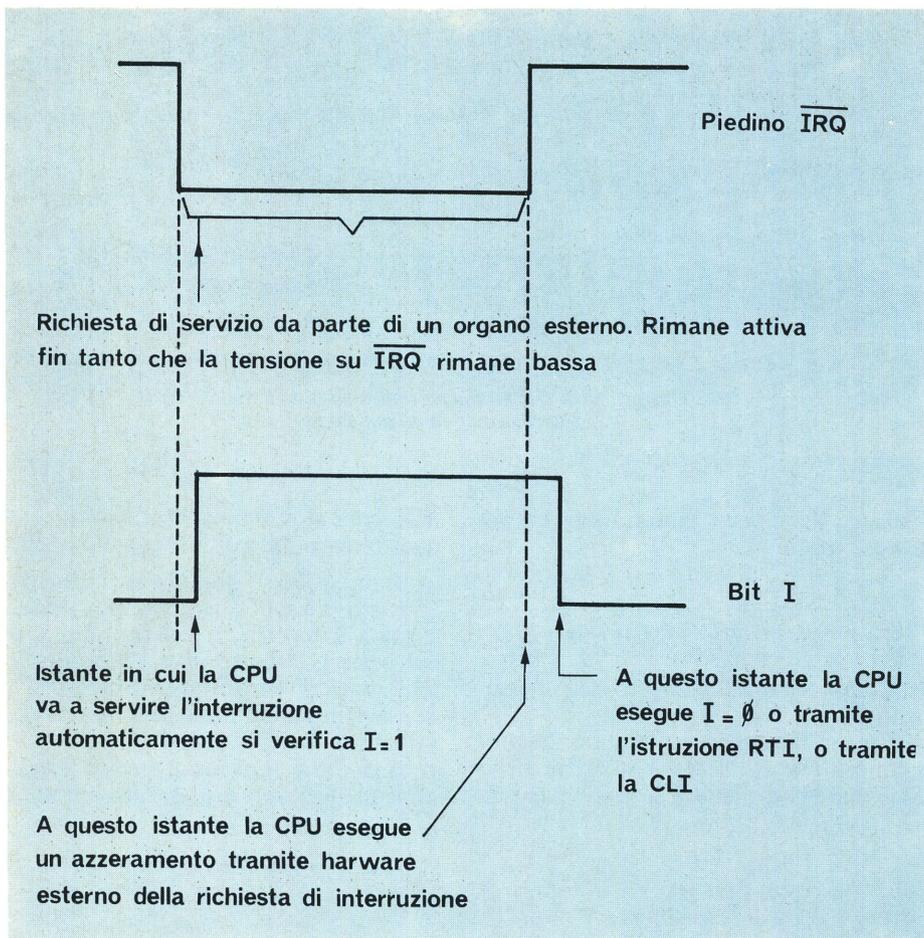


Fig. 62 - Funzionamento dell'Interrupt.

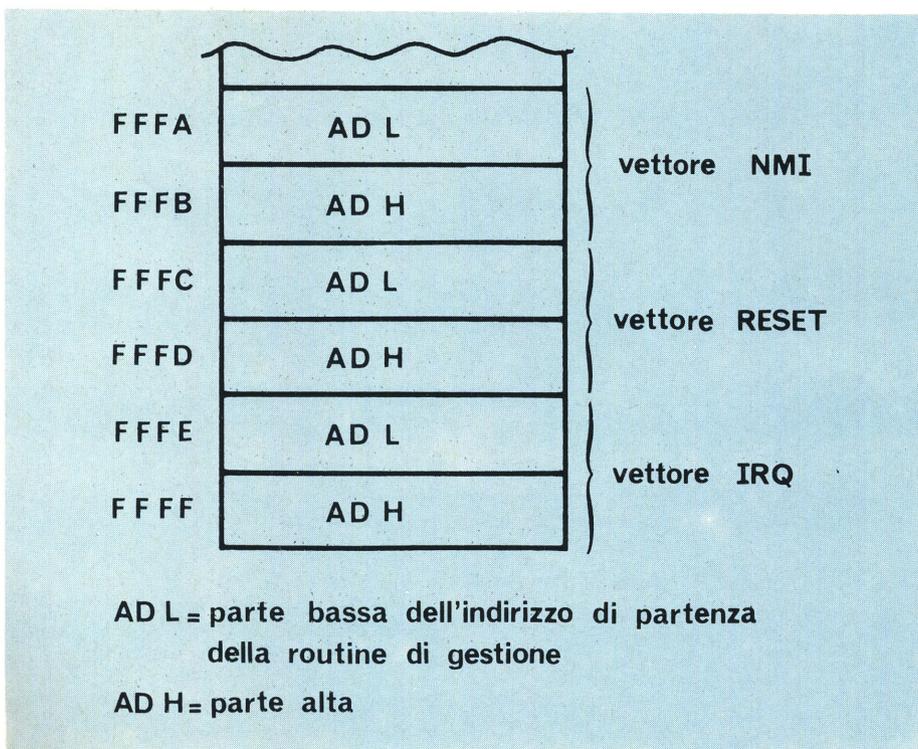


Fig. 63 - Posizionamento in memoria dei vettori di partenza della routine di Interrupt.

dello Status e del PC di partenza.

Si noti che questa procedura automatica non prevede il salvataggio dell'ACC., di X e di Y, salvataggi che, se necessario, devono essere eseguiti dallo stesso programma di gestione dell'interrupt.

I vettori di partenza delle routine di Interrupt sono posizionati in memoria secondo quanto riportato nella Fig. 63. Questi cosiddetti vettori non sono altro che due locazioni di memoria successive che contengono l'indirizzo di memoria in cui inizia il programma di gestione dell'interrupt; nel caso dell'AMICO 2000 sono già scritti nella PROM del Monitor (vedi Fig. 64).

Notiamo subito che gli indirizzi in cui sono memorizzati i vettori di restart sono fissi e posizionati in fondo alla memoria (le ultime 6 parole dei 64k indirizzabili). È perciò che generalmente l'ultimo k della memoria di un 6502 è di memoria ROM.

Il funzionamento di Reset

Nella nomenclatura dei piedini del microprocessore 6502, oltre a quelli indicati con NMI e \overline{IRQ} che abbiamo già esaminato ne esiste uno che si chiama \overline{RES} (il numero 40).

La sua funzione è quella di permettere che la CPU cominci a funzionare da un punto ben determinato. Quando questo piedino viene portato a livello logico 0 (meno di 0.8 V) il processor arresta qualsiasi attività e rimane in attesa che questo livello logico torni a 1. Appena questo avviene, automaticamente il PC viene caricato con i valori presenti nella tabella di Restart (vettore RESET), quindi la macchina inizia le sue operazioni dall'istruzione puntata dal vettore di reset stesso.

È evidente che in qualsiasi momento, l'esecuzione di un programma può essere interrotta agendo su questo piedino.

Una operazione di questo genere comunque viene eseguita ogni volta che si dà alimentazione al microelaboratore; infatti, si vada a vedere lo schema elettrico, un gruppo di ritardo formato dalla resistenza R1 e dal condensatore C3 permette all'alimentatore di stabilizzarsi prima di portare a livello logico 1 il piedino 40 della CPU 6502.

Esempio dell'uso di Interrupt

Per usare l'Interrupt nell'AMICO 2000, che ha i vettori di partenza fissi sulla PROM del Monitor, si ricorre ad un artificio di programmazione presente nel monitor stesso.

0200	A9	LDA	#\$00	}	Caricamento del vettore di restart di NMI (locazione 0300)
1	00				
2	8D	STA	\$03FC		
3	FC				
4	03				
5	A9	LDA	#\$03		
6	03				
7	8D	STA	\$03FD		
8	FD				
9	03				
A	Loop	18	CLC	}	Il processor continua a girare in questo punto (il display si spegne)
B		90	BCC Loop		
C		FD			

Scriviamo la routine che serve l'interruzione:

0300	A9	LDA	#\$88	}	Carico 88 sul display		
1	88						
2	85	STA	\$FB				
3	FB						
4	85	STA	\$FA				
5	FA						
6	85	STA	\$F9				
7	F9						
8	A9	LDA	#\$10				
9	10						
A	85	STA	\$00	}	Parametri della routine di ritardo (durata accensione display)		
B	00						
C	A9	LDA	#\$00				
D	00						
E	85	STA	\$01				
F	01						
0310	Loop 1	20	JSR SCANS			}	Routine rinfresco display
1		0C					
2		FF					
3		C6	DEC \$01				
4		01					
5		D0	BNE Loop 1				
6		F9					
7		C6	DEC \$00				
8		00					
9		D0	BNE Loop 1				
A		F5		}	Ritorno al programma principale		
B		40	RTI				

Le routine di $\overline{\text{NMI}}$ e di $\overline{\text{IRQ}}$ si riducono ad un salto indiretto sul contenuto delle locazioni di memoria RAM (quindi modificabili dall'utente) che sono:

03FC = ADL di $\overline{\text{NMI}}$

03FD = ADH di $\overline{\text{NMI}}$

03FE = ADL di $\overline{\text{IRQ}}$

03FF = ADH di $\overline{\text{IRQ}}$

Quindi nei nostri programmi utente dobbiamo inserire, prima di usare l'interrupt, i nostri specifici vettori di restart in queste locazioni di RAM (ovvero l'indirizzo dal quale deve partire la routine dell'interrupt).

Per fare un esempio dell'uso di interrupt dobbiamo avere a disposizione un segnale hardware che generi l'interrupt stesso. Osservando lo schema elettrico dell'AMICO 2000, si può vedere che il tasto di HALT (HLT) quando è premuto genera su NMI (piedino 6) un fronte di discesa di tensione. Possiamo perciò sfruttare questo tasto per generare una interruzione.

Facendo girare il programma che segue vedremo il display spegnersi per accendere per circa 16 secondi tutti 8 sul display ogni volta che viene premuto il tasto di HLT.

Analizziamo il programma:

1) Si caricano i vettori di restart, ovvero la locazione dalla quale deve partire

la routine che serve l'interruzione.

2) In questo Loop il programma gira su se stesso tenendo spente tutte le cifre del display.

A questo punto l'unico sistema che consente di intervenire sulla macchina è il ricorso all'interrupt. Useremo l' $\overline{\text{NMI}}$ per via della presenza del tasto HLT.

3) Alla locazione 0300 comincia la routine di interrupt.

Si noti che per il loop di ritardo non si usano i registri indice X e Y poiché la routine di scansione del display li modifica. Durante il loop di ritardo il display rimane acceso mostrando tutti 8.

Come ultima nota avvertiamo che se si preme ancora il tasto HLT mentre il display è acceso esso continuerà a rimanere acceso perché non si esce più dalla routine di interrupt.

Funzionamento in Single Step

Sull'AMICO 2000A esiste un interruttore per il funzionamento della macchina in "Single Step" o passo singolo che fino ad ora non abbiamo utilizzato.

Con questo tipo di funzionamento è possibile esaminare istruzione per istruzione lo svolgersi di un programma fermandosi ad esaminare per ogni passo (*step*) il contenuto dell'accumulatore e degli altri registri di macchina. Ciò è molto utile quando, ad esempio, non si riesce a comprendere perché un certo programma non va come vorremmo e scoprire quindi dove è stato fatto l'errore.

Per poter utilizzare la macchina in single step prima di tutto è necessario caricare i dati 00 e FE rispettivamente alle locazioni 03FC e 03FD. Si provvederà quindi a scrivere il programma sul quale si intende operare in single step badando che il programma stesso non vada ad interessare le locazioni 03FC e 03FD.

Si porta ora sul display indirizzi la locazione di partenza del programma quindi si sposta a destra l'interruttore del single step.

A questo punto premendo il tasto RUN vediamo che sul display appare l'indirizzo della seconda istruzione del programma e, sui due digit dei dati, il contenuto esadecimale della locazione di memoria corrispondente.

Procedendo in questo modo (premando ad ogni istruzione RUN) si può esaminare lo sviluppo del programma, i vari salti o condizionamenti.

Se si desidera esaminare, dopo l'esecuzione di una determinata istruzione, il contenuto di un qualsiasi registro o

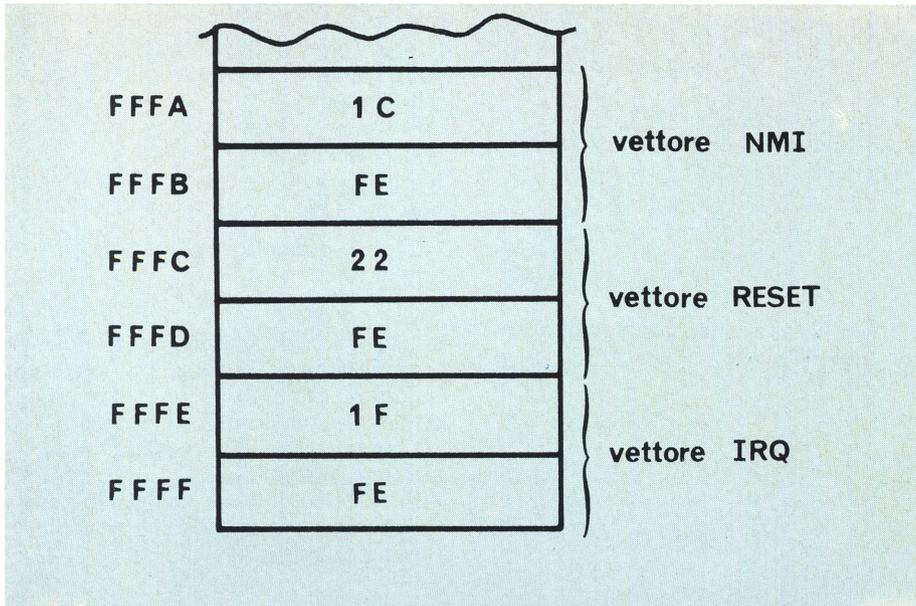


Fig. 64 - Vettori di restart dell'AMICO 2000/A.

locazione di memoria della macchina sarà sufficiente portarsi all'indirizzo corrispondente al registro o alla locazione desiderata secondo quanto indicato di seguito:

- 00F3 = Accumulatore
- 00F4 = Registro Y
- 00F5 = Registro X
- 00F6 = Program Counter (parte bassa)
- 00F7 = Program Counter (parte alta)
- 00FD = Status
- 00FE = Stack Pointer

Nel caso si desideri modificare uno qualsiasi di questi registri si procede al solito modo.

Per riprendere l'esecuzione in single step del programma è sufficiente richiamare il Program Counter premendo il tasto REG e di seguito il tasto RUN passando così alla prossima istruzione.

Si badi bene che *nessuna* delle operazioni di analisi dei vari registri o locazioni modifica di per se stessa i dati del programma.

ULTIME ISTRUZIONI DI INDIRIZZAMENTO

In questo ultimo capitolo dedicato all'analisi del software di base della CPU 6502 spiegheremo l'utilizzo delle ultime istruzioni del microprocessore, mentre verranno analizzati nuovamente i sistemi di indirizzamento propri di questa CPU la cui comprensione permette il più completo impiego dello stesso microelaboratore.

Esaminiamo dapprima le seguenti istruzioni:

BIT (compara i bit di una locazione di memoria con il contenuto dell'accumulatore). Codici operativi: 24 con indirizzamento in pagina zero; 2C con indirizzamento assoluto.

Questa istruzione esegue l'AND fra il contenuto dell'accumulatore e il contenuto di una locazione di memoria il cui indirizzo è indicato nella seconda parte dell'istruzione.

Il risultato della operazione di AND non compare da nessuna parte, non va a interessare (modificare) nessun registro, ma influenza soltanto alcuni bit dello Status.

In particolare, se il risultato dell'operazione logica AND è stato uno 00, viene automaticamente posto il bit Z = 1, altrimenti si ha Z = 0.

Il secondo effetto di questa istruzione è quello di portare il bit più significativo del contenuto della locazione di memoria interessata (M7) nel bit N dello Status e il bit successivo (M6) nel bit V dello Status (si veda la Fig. 65).

Facciamo notare ancora che la caratteristica più importante di questa istruzione è proprio il fatto che la sua esecuzione non va ad interessare nessun registro e nessuna locazione di memoria salvo, ovviamente, lo Status.

Con questo, se durante la stesura di un programma vogliamo conoscere il segno (positivo o negativo) del conte-

nuto di una locazione di memoria possiamo eseguire un'istruzione BIT, che porta il bit di segno (M7) nello Status, permettendoci di conoscerlo senza altre operazioni.

Vediamo con un esempio pratico di analizzare l'istruzione di BIT:

0200	A9	LDA	#SA5	Numero arbitrario caricato in accumulatore
1	A5			
2	24	BIT	\$00	
3	00			
4	08	PHP		Portiamo lo Status nello Stack, per poi riprenderlo in accumulatore ⁽¹⁾
5	68	PLA		
6	85	STA	\$01	Memorizzo della locazione 0001 lo Status per poi esaminarlo
7	01			
8	4C	JMP	Monitor	
9	22			
020A	FE			

⁽¹⁾ Questo è l'unico modo per poter trasferire lo Status nell'accumulatore.

Introducendo un numero nella locazione di memoria 0000 e facendo partire il programma, troveremo nella locazione 0001 il contenuto dello Status.

Negli esempi di seguito useremo le parentesi per indicare che si tratta del contenuto di quella locazione: (0000) = equivale a: "il contenuto della locazione di memoria 0000 è".

Per esempio se si ha (0000) = 5A troviamo (0001) = 76.

Infatti 5A = 01011010 e i primi due bit sono 0 e 1, quindi N = 0, V = 1 nello Status.

Inoltre 5A AND A5 = 0, quindi Z = 1 nello Status.

Se si ha (0000) = 5B si ha (0001) = 74, cioè Z = 0, in quanto 5A AND 5B ≠ 0.

Verificate i seguenti risultati utilizzando il precedente programma sull'AMICO 2000/A:

(0000) = 77; (0001) = 74
(0000) = 01; (0001) = 34

Ovviamente questi risultati valgono per i tre bit di stato interessati, gli altri non vengono influenzati dall'istruzione. Vediamo ora un altro utilizzo pratico della istruzione di BIT per quanto riguarda in particolare la modifica del bit Z dello Status.

Quando vogliamo sapere se uno qualsiasi degli 8 bit di una locazione di memoria è a 0 o a 1, dobbiamo procedere ad una cosiddetta mascheratura della parola, cioè ad una operazione di AND fra la parola stessa e una parola che ha tutti i bit a 0 salvo quello che ci interessa mascherare e che viene posto a 1.

Una delle ragioni per cui potrebbe interessarci l'analisi di un bit specifico è per esempio quando vogliamo tenere 8 flag contemporaneamente in una stessa locazione di memoria, sia per risparmiare RAM, sia per risparmiare istruzioni quando vogliamo esaminare più di un flag alla volta.

Vediamo subito un esempio pratico: si vuole esaminare se il bit 4 di una certa locazione di memoria è a 1.

Facendo riferimento alla Fig. 66 eseguiamo una mascheratura della locazione di memoria M con la parola 10 contenuta in accumulatore.

Il risultato, che mediante l'istruzione di BIT non compare da nessuna parte, influenza il bit di Status Z. A questo punto, se il bit Y che volevamo analizzare è uguale a 0, il risultato dell'operazione di BIT è uguale a 0 e quindi $Z = 1$; viceversa, se Y è uguale a 1.

Lo stesso risultato lo avremmo ottenuto se invece della istruzione di BIT avessimo utilizzato l'istruzione AND che però avrebbe modificato il contenuto dell'accumulatore.

BRK (istruzione di Break). Codice operativo 00. È un'istruzione che opera in maniera simile all'interrupt. Si ha il salvataggio in Stack del (PC) + 2 e dello Status, quindi il PC viene caricato con il contenuto delle locazioni di memoria FFFE e FFFF.

L'uso di questa istruzione è piuttosto complicato ed esula dai propositi di questa trattazione.

NOP (nessuna operazione). Codice operativo EA.

Questa istruzione non fa nulla e vi chiederete allora a che cosa serve. Facciamo un semplice esempio: si abbia un programma in cui momentaneamente si deve levare una istruzione per controllare se è quella che non lo fa funzionare. Sarebbe poco pratico riscrivere tutto il programma, fare la prova, poi riscriverlo nuovamente: è molto più semplice sostituire l'istruzione dubbia con una che non fa niente! Possiamo suggerire anche l'impiego di questa istruzione, scritta diverse volte di seguito, in quei punti del programma nei quali si pensa di dover eseguire delle modifiche, sempre per non dover riscrivere tutto ogni volta.

Sistemi di indirizzamento del 6502

Riprendiamo in questa sede in maniera organica questo importantissimo aspetto del software del microprocessore.

Indirizzamento immediato. L'operando è contenuto nel secondo byte dell'istruzione.

Esempio. Se si vuole caricare il numero 1B nell'accumulatore si scrive:

LDA = 1B

e si traduce in:

- A9 1° byte - Codice operativo di LDA
1B in maniera immediata.
1B 2° byte - Dato immediato da caricare nell'accumulatore.

Indirizzamento assoluto. Il secondo byte dell'istruzione contiene la parte bassa dell'indirizzo della locazione di memoria interessata; il terzo byte la parte alta dell'indirizzo.

Si voglia per esempio paragonare il contenuto del registro indice X con il contenuto della locazione di memoria 0354; si scrive:

CPX \$ 0354

e si traduce in:

- EC 1° byte - Codice operativo di CPX con indirizzamento assoluto.
54 2° byte - Parte dell'indirizzo della locazione di memoria interessata.
03 3° byte - Parte alta dell'indirizzo della locazione di memoria interessata.

Indirizzamento dell'Accumulatore. È relativo alle istruzioni di un solo byte che operano sull'accumulatore stesso (sono le istruzioni di Shift e Rotazione).

Indirizzamento in pagina zero. Il 2° byte dell'istruzione contiene la parte bassa dell'indirizzo della locazione di memoria interessata. La parte alta dell'indirizzo è sott'intesa in quanto è sempre 00.

Esempio. Si voglia eseguire uno Shift a destra della locazione di memoria 0013 e cioè:

LSR \$ 13

Questa istruzione si traduce in:

- 46 1° byte - Codice operativo di LSR con indirizzamento in pagina zero.
13 2° byte - Parte bassa dell'indirizzo della locazione di memoria interessata.
La parte alta dell'indirizzo è sott'intesa perché è 00.

Indirizzamento in pagina zero indicizzato (Z. PAGE,X - Z. PAGE,Y). Il secondo byte dell'istruzione viene sommato, senza tener conto del Carry, al contenuto del registro indice indicato nella istruzione. Il risultato di questa operazione è la parte bassa dell'indirizzo della locazione di memoria interessata. La parte alta dell'indirizzo è sempre zero.

Esempio. Se scriviamo:

15 ORA 2B,X (con p.es. X = 02)
2B

Eseguiamo l'OR fra il contenuto dell'accumulatore e il contenuto della locazione di memoria 002B + 02 = 002D

Indirizzamento indicizzato assoluto (ABS,X - ABS,Y). L'indirizzo della locazione di memoria interessata viene calcolato, senza tenere conto del Carry, sommando il contenuto del registro indice indicato nell'istruzione al 2° e al 3° byte dell'istruzione stessa.

L'istruzione dell'esempio precedente l'avremmo scritta:

1D ORA 002B,X (con X = 02)

2B

00

e l'indirizzo calcolato sarebbe ancora 002B + 02 = 002D.

Indirizzamento indicizzato indiretto. (IND,X). Il secondo byte dell'istruzione viene aggiunto al contenuto del registro indice X (senza tenere conto del Carry). Il risultato punta su una locazione in pagina zero, che contiene gli 8 bit più bassi dell'indirizzo effettivo. Il byte successivo della pagina zero contiene gli 8 bit più alti dell'indirizzo effettivo.

Esempio. Supponiamo che X = 04 e che (0006) = 34 e (0007) = 02.

L'istruzione:

21 AND (02,X)

02

calcola l'indirizzo della locazione di memoria dalla quale prelevare il dato eseguendo 02 + X = 06.

Nella locazione 0006 è contenuta la parte bassa (34) dell'indirizzo della locazione di memoria effettiva interessata dalla operazione di AND, mentre la parte alta (02) dell'indirizzo è contenuta nella locazione di memoria successiva 0007; l'indirizzo della locazione di memoria effettivamente interessata è quindi lo 0234.

Indirizzamento indiretto indicizzato (IND),Y. Il secondo byte dell'istruzione punta su una locazione in pagina zero. Il contenuto di questa locazione di memoria viene aggiunto al registro indice Y. Il risultato sono gli 8 bit più bassi dell'indirizzo effettivo. Il Carry della prima somma viene aggiunto al contenuto della successiva locazione di memoria di pagina zero e il risultato sono gli 8 bit più alti dell'indirizzo effettivo.

Esempio. Supponiamo che Y = 04 e che (0002) = FC e (0003) = 01.

L'istruzione:

11 AND (02),Y

02

preleva il byte puntato dal secondo byte dell'istruzione (02), lo somma a Y ottenendo la parte bassa dell'indirizzo della locazione di memoria interessata; la parte alta si ottiene sommando l'eventuale Carry alla locazione di memoria successiva a quella puntata dal secondo byte dell'istruzione; nel nostro caso si esegue 01FC + 04 = 0200, che è l'indirizzo effettivo da cui prelevare il dato.

Un esercizio

Per mettere alla prova la vostra capacità di apprendimento delle istruzioni del microelaboratore vi diamo le specifiche per la costruzione di un programma

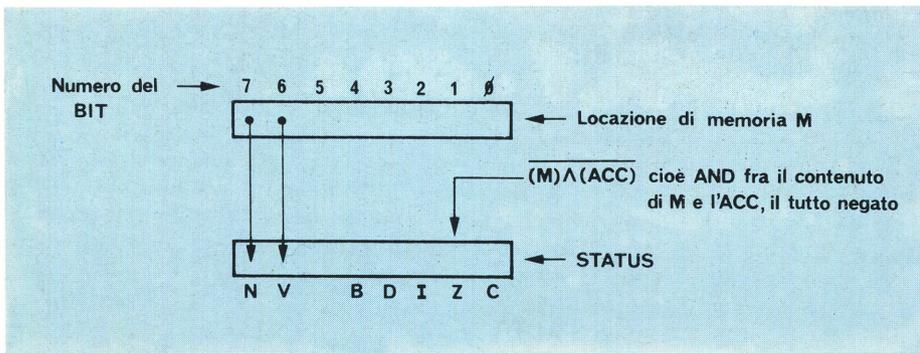


Fig. 65 - Effetti dell'istruzione BIT sui bit dello Status.

che vi consentirà di utilizzare diversi metodi di indirizzamento che abbiamo rivisto in questo articolo. Come al solito, prima di scrivere la prima istruzione chiaritevi le idee costruendo un flow-chart dell'intero programma, verificandone sulla carta la validità logica; passa-

te poi a scrivere il vero programma facendovi guidare dal flow-chart precedentemente ideato.

Passiamo alla descrizione di ciò che il vostro microelaboratore deve fare: cercare fra la locazione FE00 e la FE20 il byte 85; porre nella locazione di me-

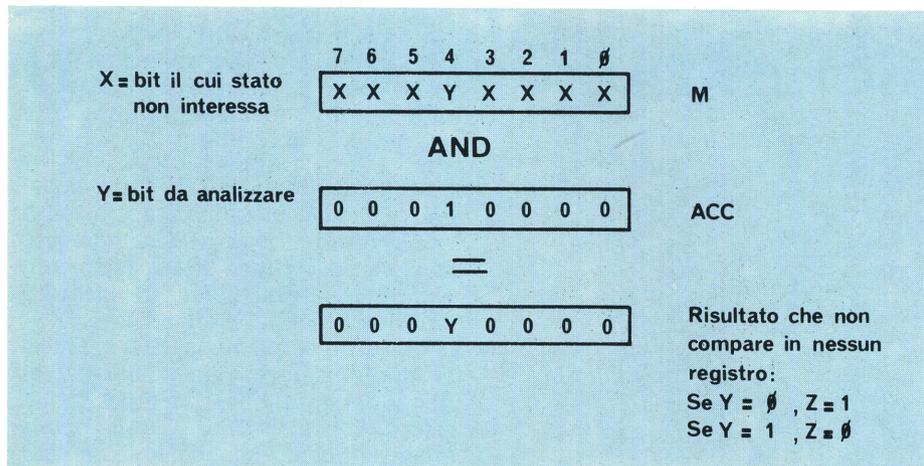


Fig. 66 - Esecuzione dell'operazione di mascheratura.

```

Programma del tiro a bersaglio. Locazione di partenza del programma 0200.

0200 =A7 00 86 F9 8A FA 86 FB A9 13 85 D0 CA 8A D1 86
0210 =D7 A5 D2 10 0D 20 8A 07 29 3F 09 0C 85 D3 29 08
0220 =E5 D7 CA D4 D0 06 A5 D3 85 D4 CA D7 A5 D1 30 06
0230 =CA D5 D0 06 CA D1 A9 08 85 D5 D8 20 DA FE 20 57
0240 =FE D5 D6 F0 12 85 D6 C9 10 80 0C A5 D1 10 08 A2
0250 =0A 8A D1 86 D7 CA D0 A9 89 8D 03 FD A7 05 A0 13
0260 =A9 00 F4 D1 D0 03 8D 80 02 F4 D7 D0 02 49 21 C9
0270 =20 D0 10 A5 D7 30 0C F8 18 A5 F9 A9 01 85 F9 A9
0280 =FE 85 D7 8D 00 FD 8C 01 FD CA D8 D0 FC 88 88 CA
0290 =10 0E C9 FE D0 04 A5 D4 85 D5 A5 D1 25 D0 30 03
02A0 =4C 11 07 20 0C FE 20 57 FE C9 13 D0 FA 4C 00 02
02B0 =01 40 08 08 08 08 38 A5 92 A5 95 A5 9A 85 91 A7
02C0 =04 85 91 95 92 CA 10 F9 AD 92 00 60
>

```

memoria 0000 il numero delle volte che questo byte è stato trovato; nelle locazioni successive di pagina zero 01 - 02, 03 - 04, etc. (a coppie) l'indirizzo (prima la parte bassa, poi quella alta) in cui è stato trovato il byte in questione.

Cercate come sempre di ottimizzare il programma usando il minor numero di istruzioni.

Un gioco: il "tiro al bersaglio"

Un po' di relax con questo gioco che consiste di un bersaglio mobile che parte dalla destra del display per poi "sparire" al di là dell'ultima cifra a sinistra: è proprio in questo punto che il proiettile, che si fa partire con la semplice pressione di un tasto qualsiasi, deve raggiungere il bersaglio mobile per colpirlo facendo accendere tutti i segmenti di quest'ultima cifra. I colpi a disposizione sono 20 e si possono sparare anche in successione, ma senza sprecarli.

IL LISTING DEL MONITOR

Software

nel corso della trattazione riguardante il software dell'AMICO 2000/A abbiamo spesso fatto riferimento al programma di monitor. Quest'ultimo è assimilabile al cosiddetto *sistema operativo* dei grandi elaboratori e come questo provvede a gestire il funzionamento dell'intero microcomputer. Senza di esso insomma il microelaboratore non sarebbe altro che un insieme di circuiti integrati senza vita.

Del programma di monitor abbiamo analizzato nel nono capitolo di questo libro un importante gruppo di subroutine che ci permettevano di usare il port esterno di I/O, la tastiera e il display come periferiche dello stesso sistema in modo da farle funzionare come è richiesto dai programmi che scriviamo.

Per poter usare ora la scheda AMICO 2000 al massimo della sua potenzialità, per poter scrivere programmi complessi e per poter riconfigurare, se fosse necessario, la mappa della memoria del microelaboratore è necessario conoscere il programma di monitor.

Consigliamo però a tutti, indipendentemente dal fatto che lo utilizzino o meno, di cercare di comprendere il funzionamento del programma principale e di ciascuna subroutine: questo sarà un ottimo esercizio per apprendere come si scrive un programma complesso e per tenere a mente quei piccoli "trucchi" di software che spesso fanno risparmiare tempo e spazio di memoria. Fisicamente, lo ricordiamo, il programma di monitor risiede in una PROM (memoria a sola lettura programmabile) tipo 93448, integrato IC9 e occupa 512 byte.

Pubblichiamo così come lo ha scritto la stampante, il listing del programma.

POSIZIONI RISERVATE AL MONITOR E ALLOCATE IN PAGINA BASE

```

00F0 =CONT
00F1 =NON USATA
00F2 =FLATAS
00F3 =LAST
00F3 =ACC
00F4 =REGY
00F5 =REGX
00F6 =PCL
00F7 =PCH
00F8 =INL
00F9 =INH
00FA =POINTL
00FB =POINTH
00FC =TEMPO
00FD =REGP
00FE =UTILE
00FF =MODO
    
```

DEFINIZIONI DELL'INPUT/OUTPUT (8255)

```

FD00 =PORTA
FD01 =PORTB
FD02 =PORTC
FD03 =DEF
    
```

ROUTINE DI ENTRATA DA INTERRUPT NON MASCHERABILE O DA BREAK

```

FE00 85 F3 SALVA STA ACC SALVO ACCUM.
FE02 68 PLA
FE03 85 FD SALVA1 STA REGP SALVO STACK
FE05 68 PLA
FE06 85 F6 SALVA1 STA PCL SALVO P.C.
FE08 85 FA STA POINTL
FE0A 68 PLA
FE0B 85 F7 STA PCH
FE0D 85 FB STA POINTH
FE0F 84 F4 STY REGY SALVO Y
FE11 86 F5 STX REGX SALVO X
FE13 BA TSX
FE14 86 FE STX UTILE SALVO STATUS
FE16 20 DAFE JSR INIZIO
FE19 4C 30FE JMP ATTESA
    
```

RILANCIO DEI VETTORI DI RESTART

FE1C 6C FC03 NMIRIL (JMP) #03FC
 FE1F 6C FF03 IRGRIL (JMP) #03FE

ROUTINE DI ENTRATA DA RESET

FE22 A2 00 RESET LDX ##00 RIENTRO FREDDO
 FE24 86 FA STX POINTL
 FE26 86 FB STX POINTH
 FE28 A2 FF RESET1 LDX ##FF RIENTRO CALDO
 FE2A 9A TXS
 FE2B 86 FE STX UTILE
 FE2D 20 DAFE JSR INIZIO

ROUTINE PRINCIPALE

FE30 20 06FF ATTESA JSR SCANS
 FE33 D0 05 BNE AVA1
 FE35 85 F2 STA FLATAS
 FE37 4C 30FE JMP ATTESA
 FE3A A5 F2 AVA1 LDA FLATAS
 FE3C D0 F2 BNE ATTESA
 FE3E E6 F2 INC FLATAS
 FE40 20 06FF JSR SCANS
 FE43 F0 EB BEQ ATTESA
 FE45 20 06FF JSR SCANS
 FE48 F0 E6 BEQ ATTESA
 FE4A 20 57FF JSR TASTO LEGGO IL TASTO
 FE4D C9 15 CMP ##15
 FE4F 10 DF BPL ATTESA
 FE51 A2 15 LDX ##15 PULSO IL DISPLAY
 FE53 A0 FF LOOP1 LDY ##FF
 FE55 88 LOOP2 DEX
 FE56 D0 FD BNE LOOP2
 FE58 CA DEX
 FE59 D0 F8 BNE LOOP1
 FE5B C9 14 CMP ##14
 FE5D F0 5D BEQ DISPC DISPLAY P.C.
 FE5F C9 10 CMP ##10
 FE61 F0 2C BEQ ADMODE INPUT INDIRIZZO
 FE63 C9 11 CMP ##11
 FE65 F0 2C BEQ DAMODE INPUT DATO
 FE67 C9 12 CMP ##12
 FE69 F0 2F BEQ NEXT PROSSIMO DATO
 FE6B C9 13 CMP ##13
 FE6D F0 4A BEQ RUN START PROGRAMMA
 FE6F 0A ASL A
 FE70 0A ASL A
 FE71 0A ASL A
 FE72 0A ASL A
 F073 85 FC STA TEMPO DEPOS. TEMPOR.
 F075 A2 04 LDX ##04
 FE77 A4 FF DATO LDY MODO
 FE79 D0 0A BNE ADDR TEST DEL MODO
 FE7B B1 FA LDA (POINTL),Y
 FE7D 06 FC ASL TEMPO
 FE7F 2A ROL A
 FE80 91 FA STA (POINTL),Y
 FE82 4C 8AFE JMP DATO1
 FE85 0A ADDR ASL A SHIFT CARAT.
 FE86 26 FA ROL POINTL SHIFT IND.LOW
 FE88 26 FB ROL POINTH SHIFT IND.HIGH
 FE8A CA DATO1 DEX
 FE8B D0 EA BNE DATO QUATTRO VOLTE
 FE8D F0 08 BEQ DATO3
 FE8F A9 01 ADMODE LDA ##01
 FE91 D0 02 BNE DATO2
 FE93 A9 00 DAMODE LDA ##00
 FE95 85 FF DATO2 STA MODO
 FE97 4C 30FE DATO3 JMP ATTESA
 FE9A A2 FF NEXT LDX ##FF
 FE9C 86 F0 STX CONT
 FE9E 20 50FF LOOP3 JSR INCPT
 FEA1 20 06FF LOOP4 JSR SCANS
 FEA4 20 57FF JSR TASTO
 FEA7 C9 12 CMP ##12
 FEA9 D0 07 BNE AVA2
 FEAB C6 F0 DEC COUNT
 FEAD D0 F2 BNE LOOP4
 FEAF 4C 9EFE JMP LOOP3
 FEB2 A2 FF AVA2 LDX ##FF
 FEB4 86 F0 STX COUNT
 FEB6 4C 30FE JMP ATTESA
 FEB9 4C C7FE RUN JMP RUNMOD

ROUTINE PER IL CARICAMENTO DEL P.C. SUL DISPLAY

FEBE A5 F6 DISPC LDA PCL
 FEBE 85 FA STA POINTL
 FEC0 A5 F7 LDA PCH
 FEC2 85 FB STA POINTH
 FEC4 4C 30FE JMP ATTESA

ROUTINE DI START DEL PROGRAMMA UTENTE

FEC7 A6 FE RUNMOD LDX UTILE
 FEC9 9A TXS
 FECA A5 FB LDA POINTH
 FECC 48 PHA
 FECD A5 FA LDA POINTL
 FECE 48 PHA
 FED0 A5 FD LDA REGP
 FED2 48 PHA
 FED3 A6 F5 LDX REGX
 FED5 A4 F4 LDY REGY
 FED7 A5 F3 LDA ACC
 FED9 40 RTI

ROUTINE DI INIZIALIZZAZIONE

FEDA A2 01 INIZIO LDX ##01
 FEDC 86 FF STX MODO
 FEDE A2 99 LDY ##99
 FEED 8E 03FD STX DEF
 FEE3 A2 07 LDY ##07
 FEE5 8E 01FD STX PORTB
 FEE8 D8 CLD
 FEE9 78 SEI
 FEEA 60 RTS

ROUTINE DI TASTO ATTIVO

FEEB A0 03 TESTAS LDY ##03 TRE RIGHE
 FEED A2 01 LDY ##01 DIGIT ZERO
 FEEF A9 FF RETASR LDA ##FF
 FEF1 8E 01FD ENDTAS STX PORTB CARICO PORT B
 FEF4 E8 INX
 FEF5 E8 INX NUOVO DIGIT
 FEF6 2D 00FD AND PORTA LEGGO PORT A
 FEF9 88 DEY
 FEFA D0 F5 BNE ENDTAS TEST PER FINE
 FEFC A0 07 LDY ##07 RIPRISTINO
 FEFE 8C 01FD STY PORTB
 FFF0 09 80 ORA ##80 MASCHERA
 FFF3 49 FF EOR ##FF
 FFF5 60 RTS

ROUTINE DI SCANSIONE DEL DISPLAY

FFF6 A0 00 SCANS LDY ##00 PRELEVO DATO
 FFF8 B1 FA LDA (POINTL),Y
 FFFA 85 F9 STA INH
 FFFC A9 89 SCANDI LDA ##89 INZ. 8255
 FF0E 8D 03FD STA DEF
 FF11 A2 09 LDY ##09 PRESET CONT.
 FF13 A0 03 LDY ##03
 FF15 B9 F800 SCANS1 LDA INL,Y PRELEVO BYTE
 FF18 4A LSR A
 FF19 4A LSR A
 FF1A 4A LSR A
 FF1B 4A LSR A ISOLO MSD
 FF1C 20 35FF JSR CONDAT SUL DISPLAY
 FF1F B9 F800 LDA INL,Y PRELEVO BYTE
 FF22 29 0F AND ##0F ISOLO LSD
 FF24 20 35FF JSR CONDAT SUL DISPLAY
 FF27 88 DEY
 FF28 D0 EB BNE SCANS1
 FF2A 8E 01FD STX PORTB DIGIT OFF
 FF2D A9 99 LDA ##99 INIZ. 8255
 FF2F 8D 03FD STA DEF

```

FF32 4C EBFE JMP TESTAS
FF35 84 FC CONDAT STY TEMPO SALVO Y
FF37 A8 TAY
FF38 B9 EAFF LDA TAB,Y LEGGO TABELLA
FF3B AD 00 DD LDY ##00 SEGMENTI OFF
FF3D 8C 00FD STY PORTA
FF40 8E 01FD STX PORTB NUOVO DIGIT
FF43 8D 00FD STA PORTA SEGMENTI ON
FF46 A0 7F LDY ##7F RITARDO
FF48 88 CONVE1 DEY
FF49 DO FD BNE CONVE1
FF4B E8 INX
FF4C E8 INX NUOVO DIGIT
FF4D A4 FC LDY TEMPO RIPRISTINO Y
FF4F 60 RTS

```

ROUTINE DI INCREMENTO PUNTATORE

```

FF50 E6 FA INCPT INC POINTL
FF52 DO 02 BNE INCPT1
FF54 E6 FB INC POINTH
FF56 60 INCPT1 RTS

```

ROUTINE DI IDENTIFICAZIONE DEL TASTO

```

FF57 A2 01 TASTO LDX ##01 DIGIT 0
FF59 AD 01 TASTO1 LDY ##01 RIGA 1
FF5B 20 EFFE JSR RETAST
FF5E DO 07 BNE INTAST TEST PER TASTO
FF60 ED 07 CPX ##07 TEST PER DIGIT 2
FF62 DO F5 BNE TASTO1
FF64 A9 15 LDA ##15 15=NESSUN TASTO
FF66 60 RTS
FF67 AD FF INTAST LDY ##FF
FF69 0A INTAS1 ASL A
FF6A B0 03 BCS INTAS2
FF6C C8 INY
FF6D 10 FA BPL INTAS1
FF6F 8A INTAS2 TXA
FF70 29 0F AND ##0F ISOLO MSD
FF72 4A LSR A DIVIDO PER 2
FF73 AA TAX
FF74 98 TYA
FF75 10 03 BPL INTAS4
FF77 18 INTAS3 CLC
FF78 69 07 ADC ##07 MOLT.(X-1)*AC
FF7A CA INTAS4 DEX
FF7B DO FA BNE INTAS3
FF7D 60 RTS

```

ROUTINE DI RINFRESCO DISPLAY

```

FF7E A9 89 START LDA ##89
FF80 8D 03FD STA CONTR
FF83 A2 09 LDX ##09
FF85 A0 00 LDY ##00
FF87 B9 8FDD LOOP LDA #008F,Y
FF8A 84 FC STY TEMPO
FF8C 20 3BFF JSR DD
FF8F C8 INY
FF90 CD 06 CPY ##06
FF92 90 F3 BCC LOOP
FF94 60 RTS
FF95 FF FF LOCAZIONI LIBERE

```

ROUTINE DI CALCOLO DEI JMP

```

FF97 D8 CJMP CLD
FF98 18 CLC
FF99 A5 FA LDA POINTL
FF9B E5 FB SBC POINTH
FF9D 85 F9 STA INH
FF9F C6 F9 DEC INH
FFA1 20 0CFF JSR SCAND1
FFA4 20 57FF JSR TASTO
FFA7 C5 F3 CMP LAST
FFA9 FD EC BEQ CJMP
FFAB 85 F3 STA LAST
FFAD C9 10 CMP ##10
FFAF BD E6 BCS CJMP
FFB1 0A ASL A

```

```

FFB2 0A ASL A
FFB3 0A ASL A
FFB4 0A ASL A
FFB5 A2 04 LDX ##04
FFB7 0A LOOPN ASL A
FFB8 26 FA ROL $FA
FFBA 26 FB ROL $FB
FFBC CA DEX
FFBD DO F8 BNE LOOPN
FFBF FO D6 BEQ CJMP

```

LOCAZIONI DI MEMORIA LIBERE

```

FFC1 FF FF
FFC3 FF FF
FFC5 FF FF
FFC7 FF FF
FFC9 FF FF
FFCB FF FF
FFCD FF FF
FFCF FF FF
FFD1 FF FF
FFD3 FF FF
FFD5 FF FF
FFD7 FF FF
FFD9 FF FF
FFDB FF FF
FFDD FF FF
FFDF FF FF
FFE1 FF FF
FFE3 FF FF
FFE5 FF FF
FFE7 FF FF
FFE9 FF

```

TABELLA PER IL PILOTAGGIO DEL DISPLAY

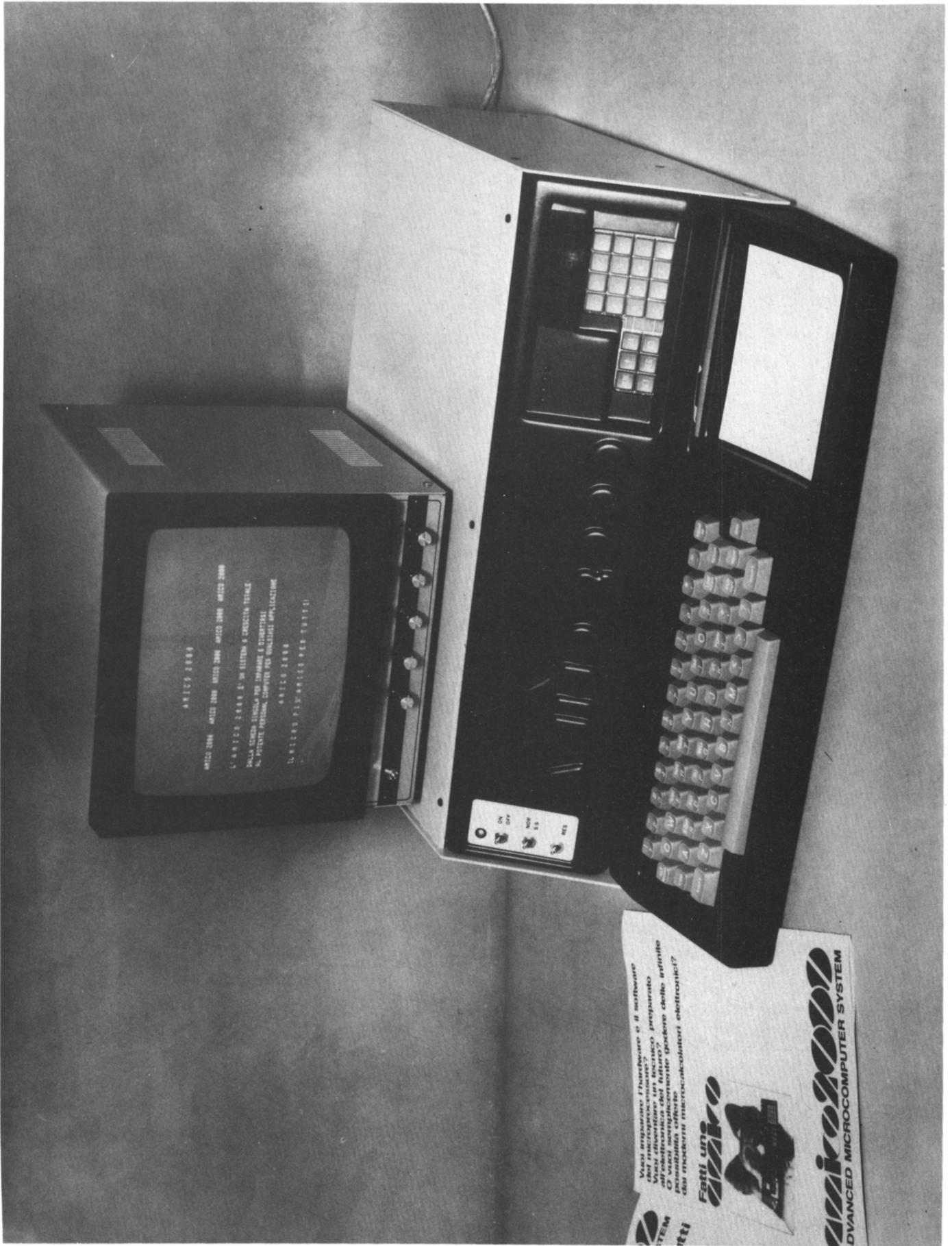
FFEA	BF	TAB	0
FFEB	86		1
FFEC	DB		2
FFED	CF		3
FFEE	E6		4
FFEF	ED		5
FFF0	FD		6
FFF1	87		7
FFF2	FF		8
FFF3	EF		9
FFF4	F7		A
FFF5	FC		B
FFF6	B9		C
FFF7	DE		D
FFF8	F9		E
FFF9	F1		F

VETTORI DI RESTART

```

FFFA 1C FE NMI
FFFC 22 FE RESET
FFFE 1F FE IRQ

```



Il sistema AMICO 2000 nella sua configurazione completa di contenitore e monitor TV.

SOMMARIO ISTRUZIONI 6502

Per i modi di indirizzamento e le influenze sui bit di stato si rimanda alla tabella riassuntiva delle istruzioni.

ADC - Add Memory to Accumulator with carry.
Somma il contenuto della locazione di memoria specificata con il contenuto dell'accumulatore sommando anche il bit di carry (riporto). Il risultato dell'operazione viene riportato in accumulatore.

AND - AND Memory with accumulator.
Esegue una operazione di AND logico fra il contenuto dell'accumulatore e il contenuto della locazione di memoria specificata. Il risultato dell'operazione viene riportato in accumulatore.

ASL - Shift Left ONE BIT (Memory or Accumulator)
Esegue uno spostamento verso sinistra del contenuto dell'accumulatore o della locazione di memoria specificata. Il bit più significativo del contenuto di partenza viene portato nel carry. Il bit che si libera viene riempito con uno zero.

BCC - BRANCH ON CARRY CLEAR
Esegue il salto specificato se il bit di carry (C) = 0.

BCS - BRANCH ON CARRY SET.
Esegue il salto specificato se il bit di carry (C) = 1

BEQ - BRANCH ON result zero.
Esegue il salto specificato se il bit di zero (Z) = 1

BIT - Test bits in memory with accumulator.
Esegue un AND logico fra il contenuto dell'accumulatore e il contenuto della locazione di memoria specificata. Il risultato dell'operazione non compare in registri della macchina, però, se il risultato è = 0, il bit di zero dello status viene messo a 1, altrimenti si ha Z = 0. I bit 6 e 7 vengono trasferiti nello status nel seguente modo N = M7; V = M6.

BMI - BRANCH ON Result minus.
Esegue il salto specificato se il bit di negativo (N) è = 1

BNE - BRANCH ON result not zero.
Esegue il salto specificato se il bit di zero (Z) = 0.

BPL - BRANCH ON result plus.
Esegue il salto specificato se il bit di negativo (N) = 0.

BRK - Force BREAK.
Forza una Interruzione, non mascherabile tramite il bit I dello Status. Salva automaticamente il PC e P. Mette a 1 il bit I dello Status.

BVC - BRANCH on overflow Clear.
Esegue il salto specificato se il bit di overflow (V) è = 0.

BVS - BRANCH on overflow Set.
Esegue il salto specificato se il bit di overflow (V) e = 1.

CLC - CLEAR CARRY FLAG
Forza a 0 il bit di carry (C) dello status.

CLD - CLEAR DECIMAL Mode
Forza a 0 il bit di Decimal (D) dello status. Le operazioni aritmetiche svolte successivamente, sono effettuate in binario.

CLI - CLEAR interrupt disable bit.
Forza a 0 il bit di interrupt (I) dello status. Dopo questa istruzione la CPU è abilitata a servire le Interruzioni che dovessero arrivarle.

CLV - CLEAR OVERFLOW FLAG
Forza a 0 il bit di overflow (V) dello status.

CMP - COMPARE MEMORY and Accumulator.
Compara il contenuto della locazione di memoria specificata con il contenuto dell'accumulatore. Come risultato dell'operazione vengono influenzati i bit N, Z, C dello status.

CPX - Compare Memory and Index x.
Compara il contenuto della locazione di memoria specificata con il contenuto dal registro x. Come risultato dell'operazione vengono influenzati i bit N, Z, C dello status.

CPY - COMPARE Memory and index y.
Confronta il contenuto della locazione di memoria specificata con il contenuto del registro indice y. Come risultato dell'operazione vengono influenzati i bit N, Z, C dello status.

DEC - Decrement memory by ONE.
Leva una unità al contenuto della locazione di memoria specificata.

DEX - Decrement index x by ONE.
Leva una unità al contenuto del registro indice x.

DEY - Decrement index y by ONE.
Leva una unità al contenuto del registro indice y.

EOR - Exclusive OR Memory with accumulator.
Esegue una operazione di OR esclusivo fra il contenuto della locazione di memoria specificata e il contenuto dell'accumulatore. Il risultato dell'operazione viene riportato in accumulatore.

INC - INCREMENT Memory by ONE
Somma uno al contenuto della locazione di memoria specificata.

INX - Increment index x by ONE.
Somma uno al contatore del registro indice x.

INY - Increment index y by ONE.
Somma uno al contenuto del registro indice y.

JMP - JUMP to new location.
Esegue un salto incondizionato alla locazione di memoria specificata come indirizzo assoluto.

JSR - Jump to new location Saving Return Address.
Salta ad un nuovo indirizzo, salvando in STACK il PC di partenza.

LDA - Load accumulatore with memory
Carica in accumulatore il contenuto della locazione di memoria specificata.

LDX - Load index x with memory.
Carica nel registro indice x il contenuto della locazione di memoria specificata.

LDY - Load index y with memory.
Carica nel registro indice y il contenuto della locazione di memoria specificata.

LSR - Shift right one bit (memory or accumulator).
Esegue uno spostamento verso destra del contenuto dell'accumulatore o della locazione di memoria specificata. Il bit

meno significativo del contenuto di par-tenza viene portato nel carry. Il bit che si libera viene riempito con uno zero.

NOP - NO operation
Non esegue alcuna operazione significativa.

ORA - OR Memory with accumulator.
Esegue una operazione di OR logico fra il contenuto dell'accumulatore e il contenuto della locazione di memoria specificata. Il risultato dell'operazione viene riportato in accumulatore.

PHA - Push accumulator on stack.
Esegue la memorizzazione dell'accumulatore nello stack. Lo stack pointer viene decrementato di uno.

PHP - Push processor status on Stack.
Esegue la memorizzazione dello Status nello stack.
Lo stack pointer viene decrementato di uno.

PLA - Pull accumulator from stack.
Esegue il caricamento dell'accumulatore con il contenuto della locazione di stack puntata dallo stack pointer. Lo stack pointer viene incrementato di uno.

PLP - Pull processor status from stack.
Esegue il caricamento dello status con il contenuto della locazione di Stack puntata dallo stack pointer. Lo stack pointer viene decrementato di uno.

ROL - Rotate one bit left (memory or accumulator).
Esegue la rotazione verso sinistra del contenuto dell'accumulatore o della locazione di memoria specificata. Il Carry viene caricato con M7, e il contenuto del carry si porta in M0.

ROR - Rotate one bit right (memory or accumulator).
Esegue la rotazione verso destra del contenuto dell'accumulatore o della locazione di memoria specificata. Il carry viene caricato con M0 e il contenuto del carry si porta in M7.

RTI - Return from interrupt.
Esegue il ritorno dalla interruzione ripristinando dallo stack, lo status e il PC.

RTS - Return from subroutine.
Esegue il rientro da una subroutine ripristinando dallo stack il PC.

SBC - Subtract memory from accumulator with borrow.
Esegue la sottrazione aritmetica fra il contenuto dell'accumulatore e il conte-

nuto della locazione di memoria specificata, sottraendo anche il carry negato. Il risultato dell'operazione viene riportato in accumulatore.

SEC - Set carry FLAG.
Forza a 1 il bit di Carry (C) dello status.

SED - Set decimal mode.
Forza a 1 il bit di decimal (D) dello status. Le operazioni aritmetiche svolte successivamente, sono effettuate nel sistema decimale.

SEI - Set interrupt disable status.
Forza a uno il bit di disabilitazione dell'interruzione (I) dello status. Dopo questa istruzione la CPU non risponde più alle richieste di interruzione.

STA - Store accumulator in memory.
Esegue la scrittura del contenuto dell'accumulatore nella locazione di memoria specificata.

STX - Store index x in memory.
Esegue la scrittura del contenuto del registro indice x nella locazione di memoria specificata.

STY - Store index y in memory.
Esegue la scrittura del contenuto del registro indice y nella locazione di memoria specificata.

TAX - Transfer Accumulator to index x.
Copia il contenuto dell'accumulatore nel registro indice x. L'accumulatore rimane invariato.

TAY - Transfer Accumulator to index y.
Copia il contenuto dell'accumulatore nel registro indice y. L'accumulatore rimane invariato.

TYA - Transfer index y to accumulator.
Copia il contenuto del registro y nell'accumulatore. Il registro y rimane invariato.

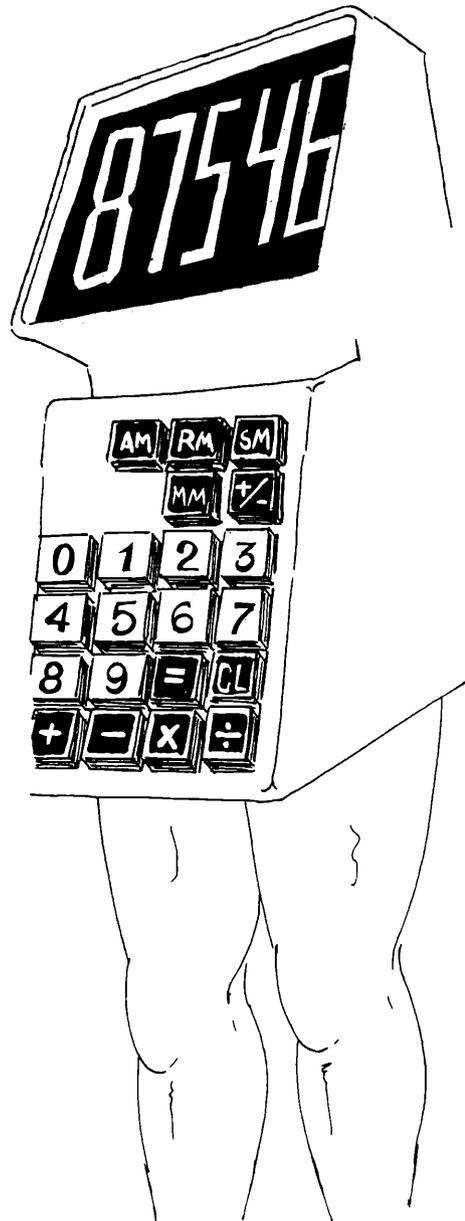
TSX - Transfer stack pointer to index x.
Copia il contenuto dello Stack pointer nel registro indice x. Il registro S rimane invariato.

TXA - Transfer index x to accumulator.
Copia il contenuto del registro x nell'accumulatore.
Il registro x rimane invariato.

TXS - Transfer index x to stack pointer.
Copia il contenuto del registro x nello stack pointer. Il registro x rimane invariato.

PROGRAMMI APPLICATIVI PER IL MICROCOMPUTER

Calcolatrice elettronica



Questo programma permette di simulare sull'AMICO 2000 il funzionamento di una semplice calcolatrice aritmetica. Utilizzando la tastiera opportunamente letta dal programma è possibile introdurre le cifre ed eseguire le varie operazioni. Ovviamente leggeremo i risultati sul display a 6 cifre del sistema. Si noti che le funzioni realizzate coincidono praticamente con quelle offerte dalle calcolatrici commerciali più economiche. Unica differenza è l'adozione della virgola fissa invece che di quella mobile. Infatti lo scopo di questo programma è essenzialmente di dimostrare come un sistema a microprocessore, se corredato del software opportuno, possa realizzare le funzioni più diverse. Perciò, e per la limitatezza del display e della tastiera disponibile, non si è ritenuto utile realizzare delle funzioni più complesse.

Precisiamo che il programma non è rilocabile, cioè deve essere caricato nell'esatta zona di memoria indicata nel listing.

Una volta caricato tutto il codice oggetto bisognerà ricontrrollare tutto molto attentamente per eliminare gli inevitabili errori di battitura. Quindi si potrà impostare l'indirizzo 04 DC (entry point del programma) premendo il tasto RUN seguito da quello **CL** e **SM**. Se tutto è a posto dovrà apparire uno zero sulla cifra più a destra, e tutto è pronto per effettuare le operazioni descritte in seguito. Se ciò non avviene, o se comunque il funzionamento è imperfetto rispetto a qualche funzione, ciò è certamente dovuto all'errata inserzione di qualche codice.

Ricontrollate allora il contenuto della memoria cella per cella.

Uso

La tabella riportata specifica come il programma interpreti i vari tasti, ovviamente quelli numerici coincidono. A sinistra in alto c'è la funzione d'origine del tasto, a destra in basso quella assegnata dal programma della calcolatrice.

RUN	↑	DA
AM	RM	SM
	REG	AD
	MM	+/-

0	1	2	3
0	1	2	3
4	5	6	7
4	5	6	7
8	9	A	B
8	9	=	CL
C	D	E	F
+	-	x	:

Da notare come si siano sfruttati tutti i tasti possibili.

I numeri sono rappresentati da 5 cifre, con la sesta a sinistra riservata per il segno. Poichè la virgola è fissa, bisognerà tenere presente che un'operazione con risultato minore della cifra meno significativa (più a destra) produrrà uno zero sul display (underflow). Può, secondo i casi, essere conveniente considerare

le due cifre corrispondenti ai dati (due cifre a destra) come cifre decimali per ogni numero introdotto, e così via.

Vediamo ora l'uso dei vari tasti:

CL Azzerò il display, elimina errori e cancella ogni tasto impostato. Non azzerò la memoria.

+/- Cambia il segno del numero caricato sul display, può essere usato in qualsiasi momento.

= **+** **-** **x** **:**
Hanno il significato usuale.

SM Trasferisce in memoria il valore presente sul display. Il contenuto precedente della memoria è perso.

RM Trasferisce sul display il contenuto della memoria. Il contenuto precedente del display è perso.

AM Somma il contenuto del display con quello della memoria, mette il risultato in memoria e lascia invariato il display.

MM Moltiplica il contenuto del display con quello della memoria, mette il risultato in memoria e lascia invariato il display.

Uso delle 4 operazioni

Si inseriscono al massimo 5 cifre dopo di che le ulteriori vengono ignorate. Sono possibili operazioni continuate: Esempio: $8 + 9 : 2 \times 10 = 80$.

Basta premere **=** solo alla fine; un tasto di operazione provoca l'esecuzione dell'operazione precedente e ne mostra il risultato parziale.

Notare che le operazioni vengono eseguite nell'ordine in cui sono scritte; quindi la scrittura matematica esatta sarebbe:

$$(8 + 9) : 2 \times 10 = 80$$

Per eseguire un nuovo conto bisogna prima cancellare il risultato precedente, altrimenti il vecchio numero viene semplicemente spostato a sinistra.

Esempio:
 $4 \times 2 = 8$; premendo ora 43 sul display appare 438.

Notare infine che il risultato di una divisione è arrotondato per difetto.

Esempio:
 $19 : 5 = 3$ $31 : 6 = 5$

Uso della memoria

In generale si utilizza per calcolare espressioni con parentesi o nelle quali comunque la precedenza delle operazioni non sia la stessa normalmente seguita dal calcolatore.

Programma																
0200	=38	A2	03	B5	15	F5	12	95	0F	CA	D0	F7	60	A9	00	D6
0210	=13	69	00	46	13	06	16	69	00	46	16	85	1C	60	D8	20
0220	=0D	02	20	00	02	90	02	38	60	A0	18	18	A2	06	36	15
0230	=CA	D0	FB	20	00	02	90	10	A2	03	B5	0F	95	15	CA	D0
0240	=F9	A9	01	18	65	1B	85	1B	88	D0	E0	A9	01	C5	19	90
0250	=D8	D0	0E	A9	86	C5	1A	90	D0	D0	06	A9	9F	C5	1B	90
0260	=C8	06	19	46	1C	66	19	18	60	D8	20	0D	02	A9	18	85
0270	=19	A9	00	85	10	85	11	85	12	18	A9	01	24	15	F0	0B
0280	=A2	03	B5	0F	75	15	95	0F	CA	D0	F7	A0	06	A2	00	76
0290	=10	E8	88	D0	FA	C6	19	D0	E0	A2	03	B5	0F	D0	21	CA
02A0	=D0	F9	A9	01	C5	13	90	18	D0	0E	A9	86	C5	14	90	10
02B0	=D0	06	A9	9F	C5	15	90	08	06	13	46	1C	66	13	18	60
02C0	=38	60	85	39	F0	13	18	A0	03	B9	35	00	79	32	00	99
02D0	=32	00	88	D0	F4	C6	39	10	EB	60	86	3C	A0	03	B6	35
02E0	=96	38	88	D0	F9	A0	09	A2	03	18	B5	38	75	35	95	35
02F0	=CA	D0	F7	88	D0	F1	A6	3C	60	D8	A9	00	85	33	85	34
0300	=85	35	85	36	85	37	A9	01	85	38	A2	03	85	2F	29	0F
0310	=20	C2	02	20	DA	02	B5	2F	4A	4A	4A	4A	20	C2	02	20
0320	=DA	02	CA	D0	E7	60	F8	A9	00	85	30	85	31	85	32	85
0330	=36	85	37	A9	01	85	38	A2	03	A9	08	85	39	56	32	90
0340	=0F	AD	03	18	B9	35	00	79	2F	00	99	2F	00	88	D0	F4
0350	=AD	03	B9	35	00	79	35	00	99	35	00	88	D0	F4	C6	39

Programma

```

036D =D0 DB CA D0 D4 60 B5 01 10 14 0A 4A 95 01 A0 03
0370 =E8 B5 00 49 FF 95 00 88 D0 F6 69 01 95 00 60 D8
0380 =A2 0F 20 66 03 A2 12 20 66 03 A2 03 18 B5 0F 75
0390 =12 95 15 CA D0 F7 70 1F A2 15 20 66 03 A9 7F 25
03A0 =16 C9 10 B0 14 C9 01 D0 0E A9 86 C5 17 90 0A D0
03B0 =06 A9 9F C5 18 90 02 18 60 38 60 A2 04 A0 00 94
03C0 =1F CA D0 FB A2 06 94 8E CA D0 FB 60 85 24 A2 06
03D0 =A0 00 98 B4 8E 95 8E CA D0 F8 A5 24 85 94 60 A0
03E0 =04 A2 03 18 36 20 CA D0 FB 88 D0 F5 18 AA 65 23
03F0 =85 23 8A 60 E6 20 20 1F 06 60 EA EA 00 FE 6F F5
0400 =84 24 A2 03 B5 20 95 2F CA D0 F9 20 F9 02 06 33
0410 =66 20 66 33 A4 24 A2 03 B5 32 99 00 00 88 CA D0
0420 =F7 60 A2 03 B9 00 00 95 32 88 CA D0 F7 06 33 26
0430 =20 46 33 20 26 03 A2 03 B5 2F 95 20 CA D0 F9 60
0440 =A9 00 85 8F A9 F9 85 90 A9 50 85 91 85 92 85 94
0450 =A9 DC 85 93 60 EA EA EA A5 1F F0 4F C9 0E D0 0C
0460 =A0 18 20 00 04 20 69 02 A0 15 D0 2A C9 0F D0 15
0470 =4C 44 06 A0 15 06 19 66 16 46 19 20 00 04 20 1E
0480 =02 A0 1B D0 11 A0 15 C9 0D D0 03 20 F4 03 20 00
0490 =04 20 7F 03 A0 18 90 05 20 2F 06 D0 06 20 22 04
04A0 =20 59 06 A9 00 85 1F A9 06 85 1E 60 EA EA EA A6
04B0 =1F F0 05 48 20 58 04 68 85 1F C9 0E D0 04 A0 15
04C0 =D0 12 C9 0F D0 0C A0 1B A9 00 85 16 85 17 85 18
04D0 =F0 02 A0 12 20 00 04 A9 00 85 1E 60 20 95 05 D8
04E0 =EA EA 20 7E FF A9 98 8D 03 FD 20 EB FE D0 05 85
04F0 =1D 4C DF 04 A5 1D D0 E7 E6 1D EA EA 20 7E FF A9
0500 =98 8D 03 FD 20 EB FE F0 D7 EA EA 20 7E FF A9 98
0510 =8D 03 FD 20 EB FE F0 C8 20 57 FF C9 15 F0 C0 A2
0520 =15 A0 FF 88 D0 FD CA D0 F8 C9 0A 10 1B A6 1E E0
0530 =05 F0 AC A6 1E D0 03 20 BB 03 E6 1E 20 DF 03 20
0540 =35 FF 20 CC 03 4C E0 04 C9 0A D0 06 20 58 04 4C
0550 =DF 04 C9 0B D0 06 20 95 05 4C DF 04 C9 10 D0 06
0560 =20 F4 03 4C DF 04 C9 11 D0 06 20 DF 05 4C DF 04
0570 =C9 12 D0 06 20 E9 05 4C DF 04 C9 13 D0 06 20 F6
0580 =05 4C DF 04 C9 14 D0 06 20 FA 05 4C DF 04 20 AF
0590 =04 4C DF 04 EA 20 BB 03 84 1E 84 1F A9 BF 85 94
05A0 =60 EA EA EA A0 06 A2 03 B5 20 29 0F 20 35 FF 99
05B0 =8E 00 88 CA CA B5 20 4A 4A 4A 4A 20 35 FF 99 8E
05C0 =00 88 CA CA CA D0 E1 20 C8 05 60 A2 00 A9 BF E8
05D0 =E0 06 F0 0A D5 8E D0 06 A0 00 94 8E F0 F1 60 A2
05E0 =04 B5 1F 95 24 CA D0 F9 60 A2 04 85 24 95 1F CA
05F0 =D0 F9 20 59 06 60 A0 0C D0 02 A0 0E A2 04 B5 1F
0600 =95 28 CA D0 F9 98 20 AF 04 20 E9 05 20 58 04 20
0610 =DF 05 A2 04 85 28 95 1F CA D0 F9 20 59 06 60 A9
0620 =01 24 20 D0 05 A9 00 85 8F 60 A9 40 85 8F 60 A9
0630 =00 85 92 85 93 85 94 A9 DC 85 8F A9 9C 85 90 A9
0640 =50 85 91 60 A5 23 D0 0E A5 22 D0 0A A5 21 D0 06
0650 =20 40 04 4C A3 04 4C 73 04 20 A4 05 20 1F 06 60
    
```

Esempio:
per calcolare:

$$4 + (8 \times 2) + 15 + 4 = 22$$

$$(-7 + 4) \times (-9 + 11)$$

premere i tasti: con sequenza da sinistra a destra

7	+/-	+	4	=	SM
CL	9	+/-	+	1	1
=	MM	CL	8	x	2
+	4	=	:	RM	=
+/-	SM	CL	1	5	AM
CL	4	AM	RM		

Notare il risultato arrotondato della divisione. Per azzerare la memoria è sufficiente premere i due tasti: **CL** **SM**

Segnalazioni di errore

Se il risultato dell'operazione è troppo grande per essere contenuto sul display, si vede la scritta OVR (overflow). Se l'operazione eseguita è matematicamente scorretta si vede la scritta ERROR. Premere sempre **CL** dopo ogni errore. Notare che le operazioni sulla memoria non danno segnalazioni di errore e quindi bisogna usare una certa cautela.

Operazioni aritmetico-logiche



Programma

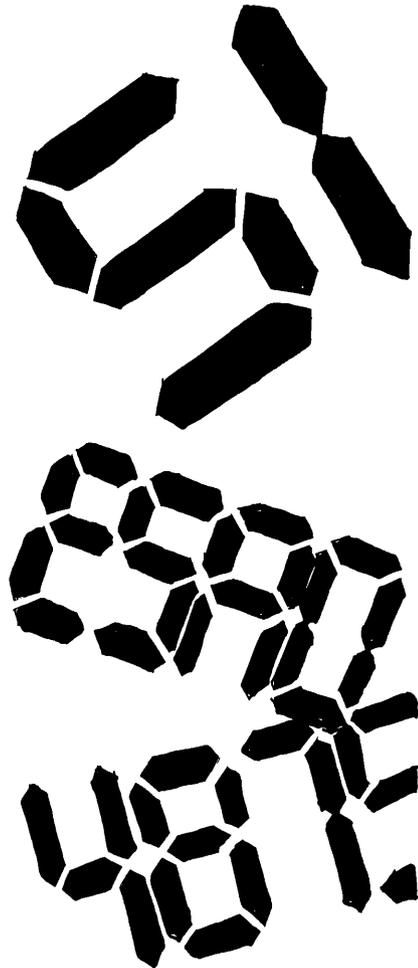
0200	=A9	73	85	8F	A9	31	85	90	A9	3F	85	91	85	94	A9	37
0210	=85	92	A9	78	85	93	20	50	FF	20	7E	FF	20	DA	FE	20
0220	=57	FF	C9	14	D0	F0	A5	FA	45	FB	85	FB	A9	79	85	8F
0230	=85	91	A9	6D	85	90	A9	3D	85	92	A9	3E	85	93	A9	30
0240	=85	94	20	DB	03	A2	94	A9	00	95	00	CA	D0	FB	A5	FA
0250	=29	0F	C9	05	30	03	4A	10	F9	C9	00	F0	24	C9	01	F0
0260	=37	C9	02	F0	4E	C9	03	F0	62	A9	7B	85	8F	A9	3F	85
0270	=90	A9	31	85	91	20	DB	03	A5	FB	45	FA	85	0A	4C	DC
0280	=02	A9	5F	85	8F	A9	5E	85	90	85	91	20	DB	03	A5	FB
0290	=18	65	FA	85	0A	4C	DC	02	A9	6D	85	8F	A9	3F	85	90
02A0	=A9	78	85	91	85	92	20	DB	03	A5	FB	38	E5	FA	85	0A
02B0	=4C	DC	02	A9	5F	85	8F	A9	37	85	90	A9	5E	85	91	20
02C0	=DB	03	A5	FA	25	FB	85	0A	4C	DC	02	A9	3F	85	8F	A9
02D0	=31	85	90	20	DB	03	A5	FA	05	FB	85	0A	A9	00	85	05
02E0	=85	F9	E6	06	D0	02	E6	07	A5	07	C9	3C	D0	06	20	E3
02F0	=03	4C	2C	02	20	0C	FF	D0	04	85	03	F0	E5	A5	03	D0
0300	=E0	E6	03	20	DA	FE	20	57	FF	C9	11	10	D5	C9	10	D0
0310	=05	E6	07	20	DB	03	F0	C7	E6	05	A2	04	06	F9	CA	D0
0320	=FB	05	F9	85	F9	A5	05	C9	02	D0	B7	A5	F9	C5	0A	F0
0330	=21	A9	7B	85	8F	A9	31	85	90	85	91	A9	5F	85	92	A9
0340	=78	85	93	A9	3F	85	94	20	DB	03	20	E3	03	E6	07	4C
0350	=2C	02	A9	7B	85	8F	A9	6D	85	90	A9	5F	85	91	A9	78
0360	=85	92	85	93	A9	3F	85	94	20	DB	03	A9	00	85	93	85
0370	=94	A5	07	C9	0B	30	12	C9	15	30	24	C9	24	30	34	C9
0380	=33	30	44	20	E3	03	4C	2C	02	A9	38	85	8F	A9	3F	85
0390	=90	A9	5E	85	91	A9	7B	85	92	20	DB	03	4C	00	02	A9
03A0	=7C	85	8F	A9	7B	85	90	85	92	A9	37	85	91	20	DB	03
03B0	=4C	00	02	A9	6D	85	8F	A9	3E	85	90	A9	71	85	91	85
03C0	=92	20	DB	03	4C	00	02	A9	73	85	8F	A9	3F	85	90	85
03D0	=92	A9	39	85	91	20	DB	03	4C	00	02	20	7E	FF	E6	08
03E0	=D0	F9	60	A9	31	85	8F	A9	30	85	90	85	94	A9	73	85
03F0	=91	A9	7B	85	92	A9	78	85	93	20	DB	03	00	FE	6F	F5

Il programma parte dalla locazione 0200 e rimane in attesa con la scritta "PRONTO". Premendo REG si ha l'invito a fare una operazione "ESEGUI": "add", "sott", "and", "or", "eor" fra due numeri esadecimali di due cifre ciascuno (display 1-2 e 3-4 da sinistra); se si scrive il risultato (due cifre) sulla tastiera esso compare sui display 5 e 6. Se non è quello richiesto il programma risponde "errato", invita a riprovare con "ripeti" e ripropone la stessa operazione come prima. Per rivedere di che operazione si tratta basta premere il pulsante AD. Se il risultato è corretto si ha immediatamente la conferma "esatto" cui segue una valutazione: "lode", "bene", "suff", "poco" in base al tempo impiegato. Il programma passa quindi di nuovo in attesa con "PRONTO".

Se si indugia più di 50 secondi e in ogni caso dopo un minuto buono il programma ripropone pazientemente la stessa operazione di prima.

Impossibile non imparare!

Traduzione da notazione binaria in esadecimale e viceversa



Programma

```

0200 =A9 78 85 8F A9 31 85 90 A9 5F 85 91 A9 5E 85 92
0210 =A9 06 85 93 A9 37 85 94 20 7E FF E6 04 20 DA FE
0220 =20 57 FF C9 0E F0 06 C9 0D F0 6F D0 EB 20 2C 03
0230 =A9 7B 85 8F A9 6D 85 90 A9 5F 85 91 A9 5E 85 92
0240 =20 1A 03 20 2C 03 A2 03 A5 04 85 02 85 03 46 02
0250 =90 09 A9 06 95 8F CA 10 F5 30 04 A9 3F 90 F5 20
0260 =7E FF E6 04 20 DA FE 20 57 FF C9 10 10 F1 85 05
0270 =A8 B9 EA FF 85 94 20 1A 03 A5 03 29 0F C5 05 F0
0280 =06 A9 00 85 94 F0 D8 A9 00 85 8F 85 90 A9 6D 85
0290 =91 A9 32 85 92 20 1A 03 F0 A9 20 2C 03 A9 5E 85
02A0 =8F A9 3E 85 90 A9 5F 85 91 A9 38 85 92 20 1A 03
02B0 =20 22 03 A5 04 29 0F 85 05 A8 B9 EA FF 85 94 A9
02C0 =00 85 02 85 06 20 7E FF E6 04 20 2D FF D0 04 85
02D0 =07 F0 F2 A5 07 D0 EE E6 07 20 DA FE 20 57 FF C9
02E0 =02 10 E2 A8 06 02 05 02 85 02 98 D0 04 A9 3F D0
02F0 =02 A9 06 A6 06 95 8F E6 06 A5 06 C9 04 D0 C6 20
0300 =1A 03 A5 05 C5 02 F0 05 20 22 03 D0 B2 A9 6D 85
0310 =93 A9 32 85 94 20 1A 03 F0 96 20 7E FF E6 01 D0
0320 =F9 60 A2 04 A9 00 95 8F CA 10 FB 60 A2 05 D0 F4

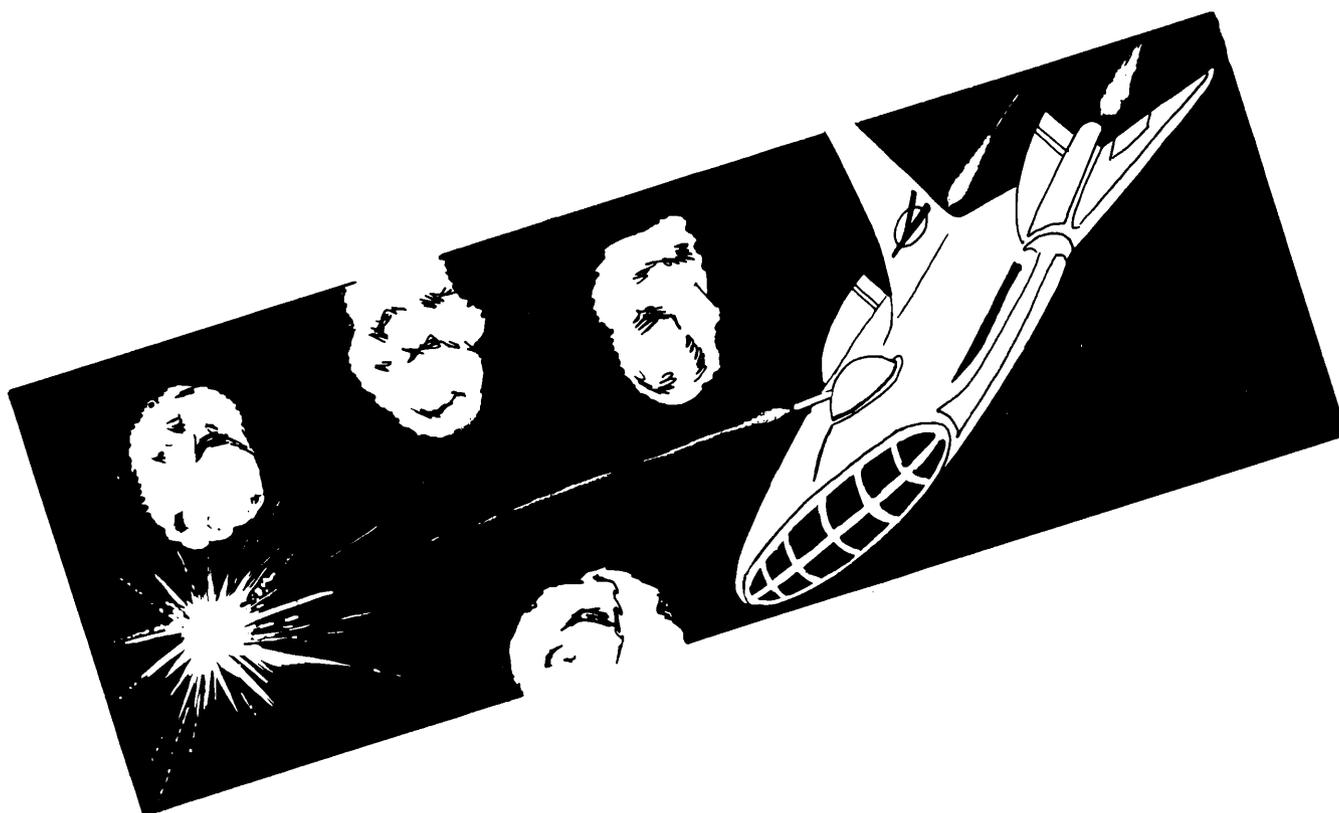
```

Il programma parte dalla locazione 0200 e scrive sul display "trad in".

Premendo il tasto D compare la scritta "dual", seguita da una cifra esadecimale da tradurre in binario con le cifre 0 e 1 soltanto, che si scrivono da sinistra a destra. Se la combinazione duale è errata il programma la rifiuta (si spengono i 4 bit); se è esatta appare la conferma "SÌ" sul display-dati e segue quindi un'altra cifra esadecimale.

Se si preme invece il tasto E compare la scritta "esad" seguita da una stringa di 4 bit da tradurre in cifra esadecimale. Se la cifra scritta è errata il programma la rifiuta spegnendola; se è esatta appare la conferma "SÌ" sul display-indirizzi e segue quindi un'altra combinazione duale.

Asteroidi



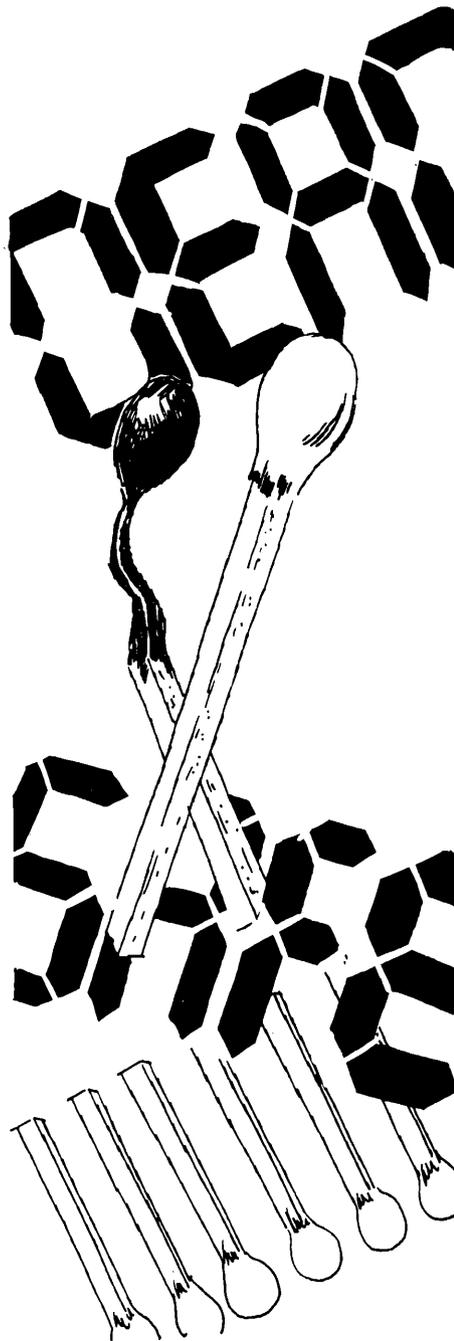
Mentre il giocatore sta pilotando la sua astronave incontra una cortina di asteroidi. Il pilota può manovrare la nave premendo il tasto "O" per muovere a sinistra o il tasto "3" per muovere a destra. Se l'astronave sarà colpita il display indicherà a quale punto del gioco è avvenuto l'impatto; più alto sarà il numero visualizzato maggiore sarà stata l'abilità del pilota.

Il programma parte dalla locazione 0200.

Programma

0200	=A9	00	85	F9	85	FA	85	FB	A2	06	BD	CE	02	95	E2	CA
0210	=10	F8	A5	E8	49	FF	85	E8	A2	05	20	48	02	20	97	02
0220	=CA	D0	F7	20	DA	FE	20	57	FF	C9	15	10	E5	C9	00	F0
0230	=06	C9	03	F0	0A	D0	DB	06	E7	A9	40	C5	E7	D0	D3	46
0240	=E7	D0	CF	38	26	E7	D0	CA	A9	89	8D	03	FD	A9	09	8D
0250	=01	FD	A9	20	85	E0	A0	02	A9	00	85	E1	.B1	E2	25	E0
0260	=F0	07	A5	E1	19	E4	00	85	E1	88	10	F0	A5	E1	C4	E8
0270	=D0	08	A4	E0	C4	E7	D0	02	09	08	8D	00	FD	20	08	03
0280	=EA	EA	EA	EA	EA	EA	EA	A9	00	8D	00	FD	EE	01	FD	EE
0290	=01	FD	46	E0	D0	C0	60	C6	E9	D0	1A	A9	30	85	E9	8A
02A0	=48	A2	FD	F8	38	B5	FC	69	00	95	FC	E8	D0	F7	D8	68
02B0	=AA	E6	E2	A5	E2	C9	30	F0	09	A0	00	A5	E7	31	E2	D0
02C0	=07	60	A9	00	85	E2	F0	F1	20	0C	FF	4C	C8	02	D5	02
02D0	=08	40	01	04	FF	00	00	00	04	00	08	00	06	12	00	11
02E0	=00	05	00	2C	00	16	00	29	00	16	00	28	00	26	00	19
02F0	=00	17	00	38	00	2E	00	09	00	18	00	24	00	15	00	39
0300	=00	0D	00	21	00	10	00	00	A5	00	85	00	01	EA	EA	EA
0310	=CE	00	01	D0	F5	60										

21 fiammiferi



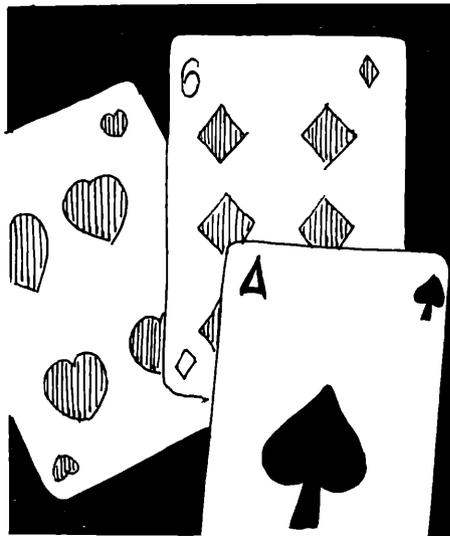
Ci sono 21 fiammiferi. Ogni giocatore può coglierne, al suo turno, 1 - 2 o 3. Chi rimane con l'ultimo fiammifero perde. Il giocatore compete con l'AMICO 2000 e deve fare la sua mossa per primo. L'indirizzo di partenza è 0200 ed il display mostra da sinistra le mosse del giocatore (2 digit) quelle del calcolatore (2 digit) ed il totale dei fiammiferi ancora in gioco (2 digit). Le mosse devono essere impostate da tastiera (tasti 1-2-3) dopo qualche secondo anche l'AMICO 2000 farà il suo gioco. Il display mostra la parola "DEAD" (perso) in caso di sconfitta "SAFE" per la vittoria.

Programma

```

0200 =A9 00 85 FB A9 21 85 F9 20 0C FF 20 57 FF C9 04
0210 =10 F6 C9 00 F0 F2 85 FB F8 38 A5 F9 E5 FB 85 F9
0220 =20 7E FF D0 FB A9 08 85 EE 20 86 02 EA EA EA EA
0230 =EA 2C 00 01 EA EA A5 EE C6 EE D0 ED C6 F9 A5 F9
0240 =29 10 4A 4A 4A 18 65 F9 E6 F9 29 03 D0 02 A9 01
0250 =AE EA EA ED A0 B0 02 A9 02 85 FA A5 F9 38 E5 FA
0260 =85 F9 C9 01 F0 04 30 10 B0 9E A9 DE 85 FB A9 AD
0270 =85 FA 20 0C FF 18 90 FA A9 5A 85 FB A9 FE 85 FA
0280 =A9 00 85 F9 F0 EC A9 60 8D 00 01 20 0C FF CE 00
0290 =01 D0 F8 60
    
```

Il gioco del 21



L'AMICO 2000 usa per questo gioco un "vero" mazzo di carte. Questo significa che dopo l'uscita di quattro assi questa carta non comparirà fino alla successiva "mischiata" del mazzo. Il gioco parte dalla loc. 0200 e per circa 2 secondi il display mostrerà la parola "carte" questo indica che l'AMICO 2000 ha mischiato il mazzo; ora il display mostra la parola "PUN" (puntata) seguita da un punto di domanda (?) ciò significa che il computer vi chiede quanto volete puntare.

Inizialmente disporrete di 20 gettoni così il display indicherà "Pun? 20". Ora potete fare la vostra giocata che non dovrà superare i 9 gettoni (il computer ignora l'impostazione dello 0). Dopo aver effettuata la puntata l'AMICO 2000 mostra a sinistra del display le due carte del giocatore, e a destra, una delle sue, l'altra carta dell'AMICO 2000 è "coperta" quindi non compare, questa diventerà trasparente quando sarà il suo turno di gioco.

Lo scopo del gioco è quello di avvicinarsi il più possibile al 21 considerando che le figure indicate con una "F" valgono 10 e gli assi indicati con la "A" possono valere 1 oppure 11. Le altre carte hanno il proprio valore quindi il 6 vale 6 il 4 vale 4 e così via.

Per avere la terza carta premere sulla tastiera il "3" per la carta il "4" e via di seguito. Quando il giocatore riterrà di non dover chiedere altre carte dovrà premere lo "0" l'AMICO 2000 riporterà sul display la scritta "TOT-XX" dove le XX indicheranno la somma dei valori delle carte. A propria volta l'AMICO 2000 giocherà cercando di battervi. Se il giocatore (o l'AMICO 2000) supera il 21 il display mostrerà la parola "SBALLA" e la mano sarà automaticamente vinta dal computer (o dal giocatore). Cinque carte chiamate senza

superare il 21 danno automaticamente la vittoria. Diversamente vince il punteggio che più si avvicina al 21. Il valore della puntata fatta all'inizio della mano sarà sommata agli altri gettoni in caso di vincita o sottratta in caso di

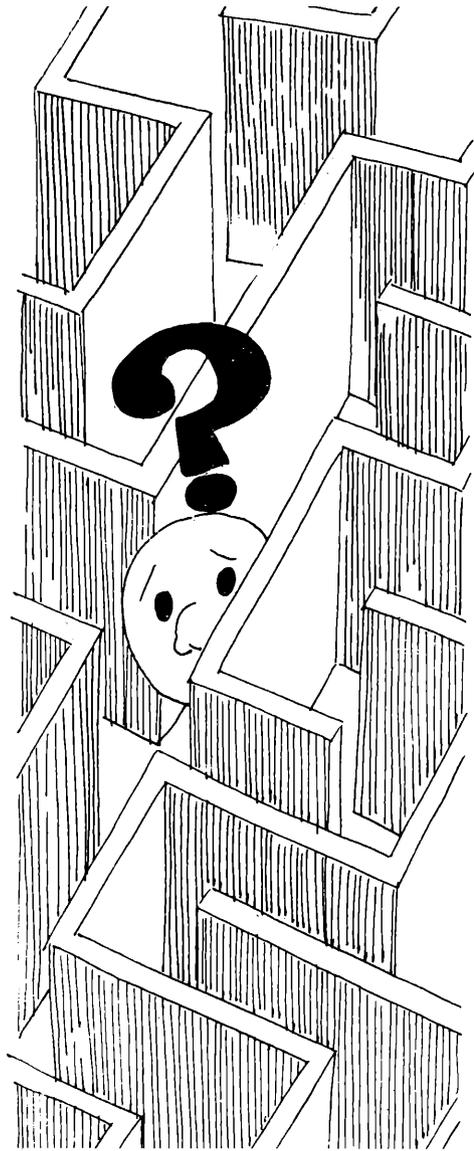
sconfitta. In caso di parità l'ammontare dei gettoni non cambia.

Per avere buone possibilità di vittoria è utile tenere a mente quanti assi o figure sono stati estratti dal mazzo con il quale state giocando.

Programma

0200	=A2	33	8A	95	40	CA	10	FA	A2	02	BD	BB	03	95	75	CA
0210	=10	F8	20	EC	03	85	80	D8	A6	76	EO	09	B0	34	A0	D8
0220	=20	57	03	A0	33	84	76	20	30	03	38	A5	81	65	82	65
0230	=85	85	80	A2	04	B5	80	95	81	CA	10	F9	29	3F	C9	34
0240	=B0	E5	AA	B9	40	00	48	B5	40	99	40	00	68	95	40	88
0250	=10	D5	A0	DE	20	57	03	A5	77	20	A6	03	20	30	03	C9
0260	=0A	B0	F9	AA	86	79	CA	30	F3	E4	77	B0	EF	A2	0B	A9
0270	=00	95	90	CA	10	FB	20	78	03	20	8F	03	20	78	03	20
0280	=64	03	86	7A	20	28	03	20	30	03	AA	CA	30	11	E4	96
0290	=D0	F5	20	78	03	C9	22	B0	40	EO	05	F0	53	D0	E8	A5
02A0	=95	48	A2	00	20	0F	03	A2	04	A9	00	95	90	CA	10	FB
02B0	=68	85	95	A6	7A	20	6D	03	20	92	03	20	28	03	A5	9A
02C0	=C9	22	B0	29	65	9B	A6	91	D0	18	C9	22	90	02	A5	9A
02D0	=C9	17	B0	2C	20	8F	03	D0	E2	20	28	03	20	55	03	20
02E0	=28	03	A5	77	F8	38	E5	79	85	77	4C	17	02	20	55	03
02F0	=20	28	03	A5	77	F8	18	65	79	A0	99	90	01	98	D0	E8
0300	=A2	03	20	0F	03	A5	9A	C5	97	F0	DF	B0	D5	90	E4	B5
0310	=97	F8	18	75	98	C9	22	B0	02	95	97	D8	B5	97	48	A0
0320	=E2	20	57	03	68	20	A6	03	A0	80	20	30	03	88	D0	FA
0330	=84	89	A0	13	A2	05	A9	89	8D	03	FD	B5	90	8D	00	FD
0340	=8C	01	FD	E6	7B	D0	FC	88	88	CA	10	EF	20	DA	FE	20
0350	=57	FF	A4	89	60	A0	E6	84	74	A0	05	B1	74	99	90	00
0360	=88	10	F8	60	A6	76	C6	76	B5	40	4A	AA	18	D0	01	
0370	=38	BD	BE	03	BC	CB	03	60	20	64	03	E6	96	A6	96	94
0380	=8F	A0	10	90	02	84	98	18	F8	65	97	85	97	D8	60	20
0390	=64	03	C6	99	A6	99	94	96	A0	10	90	02	84	98	18	F8
03A0	=65	9A	85	9A	D8	60	48	4A	4A	4A	4A	A8	B9	EA	FF	85
03B0	=94	68	29	0F	A8	B9	EA	FF	85	95	60	03	00	20	01	02
03C0	=03	04	05	06	07	08	09	10	10	10	10	F7	DB	CF	E6	ED
03D0	=FD	87	FF	EF	F1	F1	F1	F1	B9	F7	D0	F8	F9	CD	F3	9C
03E0	=D4	D3	F8	DC	F8	CD	ED	FC	F7	B8	B8	F7	38	A5	92	65
03F0	=95	65	96	85	91	A2	D4	B5	91	95	92	CA	00	FE	6F	F5
0400	=00	60														

Il labirinto



L'AMICO 2000 crea un labirinto dal quale il giocatore deve uscire.

Il giocatore è presentato, sul display, da un segmento centrale che lampeggia. Per muoversi il giocatore utilizzerà i tasti nel seguente ordine:

TASTO 9 = in alto
 TASTO 1 = in basso
 TASTO 4 = a sinistra
 TASTO 6 = a destra

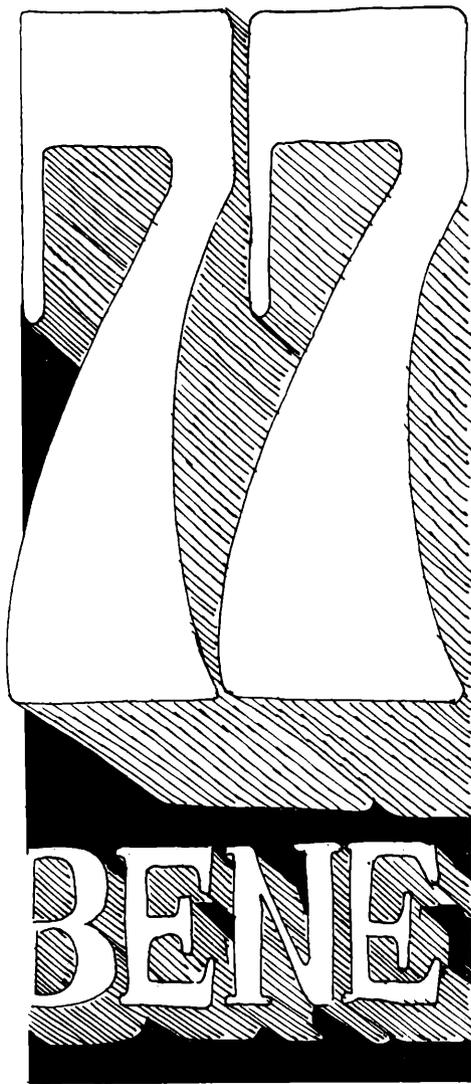
Attenzione che il computer non sposterà dal centro il segmento che rappresenta il giocatore. Ma sposterà tutto il labirinto del quale se ne vedrà di volta in volta solo una parte.

Quando sul display resterà solo il segmento che lampeggia, senza altro intorno significa che il giocatore è uscito dal labirinto. Il programma parte da 0200 - RUN per un altro labirinto.

Programma

0200	=E6	D0	20	EB	FE	D0	F9	A2	07	26	D0	9D	17	BC	08	03
0210	=BD	10	03	59	DE	02	99	DE	02	C8	C8	BD	18	03	59	DE
0220	=02	99	DE	02	CA	10	E2	A2	02	D8	30	D4	BD	D8	02	95
0230	=D2	CA	10	F8	A0	0B	B1	D2	99	D8	00	88	10	F8	A2	0A
0240	=A4	D4	A9	FF	38	36	D9	36	D8	2A	88	D0	F7	29	07	A8
0250	=B9	C6	02	95	D8	CA	CA	10	E7	C6	D5	10	0A	A9	05	85
0260	=D5	A5	DE	49	40	85	DE	A9	89	8D	03	FD	A0	09	A2	0A
0270	=B5	D8	8D	00	FD	8C	01	FD	C6	D6	D0	FC	C8	C8	CA	CA
0280	=10	EE	20	DA	FE	20	57	FF	C5	D7	F0	CD	85	D7	A2	04
0290	=DD	CE	02	F0	05	CA	10	F8	30	BC	CA	30	8D	BC	D3	02
02A0	=B9	D8	00	3D	D7	02	D0	B1	CA	10	04	C6	D4	D0	85	D0
02B0	=04	E6	D4	D0	F8	CA	D0	06	C6	D2	C6	D2	D0	EF	E6	D2
02C0	=E6	D2	D0	E9	F0	F2	00	08	40	48	01	09	41	49	13	09
02D0	=01	06	04	06	06	04	08	01	08	40	40	DA	02	08	FF	FF
02E0	=04	00	F5	7F	15	00	41	FE	5F	04	51	7D	5D	04	C1	B6
02F0	=15	14	77	D5	14	54	7D	5E	05	00	FD	FF	00	00	00	00
0300	=00	00	00	00	00	00	00	05	08	10	10	14	18	17	10	
0310	=01	04	80	10	80	02	40	40	02	02	40	01	10	04	80	10

Duesette



Scopo del gioco è stimolare la conoscenza delle tecniche di presentazione su display a 7 segmenti oltre che della notazione binaria ed esadecimale.

Il programma è rilocabile. Parte dalla prima istruzione presentando un rimescolamento di segmenti su tutti i display.

Premendo REG restano fissate due configurazioni diverse sui due display-dati.

Il gioco consiste nell'introdurre delle cifre sulla tastiera esadecimale fino a che la configurazione di destra risulti uguale a quella di sinistra che fa da modello. Avvenuto ciò il programma scrive "bene" e sottolinea lampeggiando l'uguaglianza raggiunta.

Ogni giocatore dispone di 4 mosse controllate dall'accensione di un segmento sugli altri display. Premendo a caso i tasti esadecimali è ben difficile indovinare. Tener presente che ogni cifra

introdotta modifica il byte già presente nella locazione 0094 con l'istruzione ASL e vi si sovrappone con l'istruzione ORA.

Se un giocatore, studiando bene la situazione prevede di non poter vincere in quattro mosse può sempre peggiorare le cose per l'avversario.

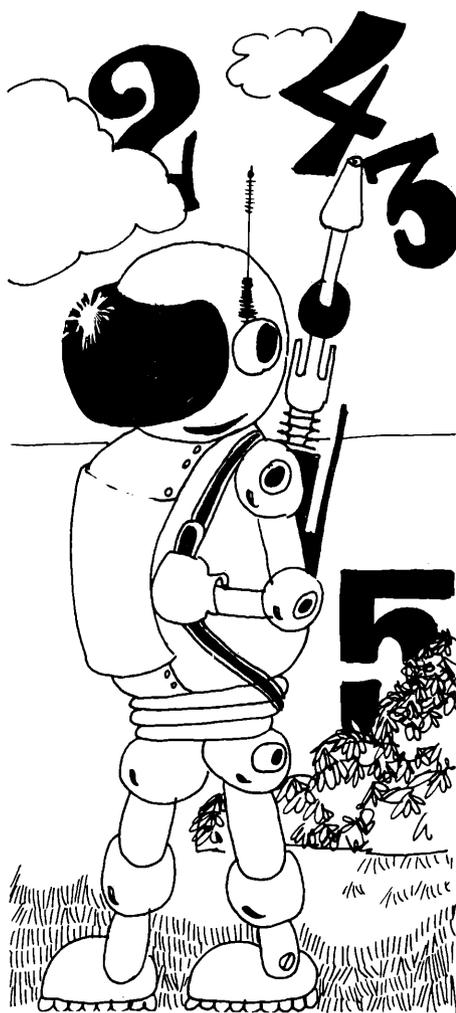
Tener presente la possibilità di vittoria anche quando si raggiunge una configurazione uguale a quella del primo display a sinistra (settematto). Anche in questo caso il programma scrive "bene" indicando due configurazioni diverse.

Per ricominciare una nuova partita premere RUN per mezzo secondo.

Programma

0200	=A9	0A	85	00	E6	04	20	7E	FF	A6	00	F6	8E	CA	86	00
0210	=D0	F2	D8	20	DA	FE	20	57	FF	C9	14	F0	02	D0	E1	A5
0220	=04	85	93	49	55	85	94	A2	03	A9	00	95	8F	CA	10	FB
0230	=A9	03	85	05	20	7E	FF	20	2D	FF	D0	04	85	03	F0	F4
0240	=A5	03	D0	F0	E6	03	20	DA	FE	20	57	FF	C9	10	10	E4
0250	=06	94	05	94	85	94	A5	94	0A	85	01	A5	93	0A	C5	01
0260	=F0	10	A6	05	F6	8F	A5	01	C5	8F	F0	06	C6	05	10	C4
0270	=30	BE	20	DA	FE	20	57	FF	C9	13	F0	84	20	7E	FF	E6
0280	=04	D0	F9	A2	7C	86	8F	CA	86	90	86	92	A9	37	85	91
0290	=9A	CA	D0	FD	E6	04	D0	FC	BA	CA	D0	F4	F0	D4		

Caccia al numero



Il programma parte dalla 1ª istruzione ed è rilocabile. Premendo REG, dopo il RUN, il display presenta ritmicamente serie dei segmenti orizzontali inferiori e in alternanza genera un segnale in uscita il cui periodo è proporzionale a un numero esadecimale di due cifre che rappresenta il “nascondiglio”.

Un semplice altoparlante collegato un capo a massa e uno ai piedini della porta C permette di avvertire una nota ritmica, che varia di potenza col numero dei piedini collegati in parallelo.

Si cerca ora di indovinare il numero predetto introducendolo dalla tastiera. Quando le due cifre risultano scritte sul display-dati ne deriva un segnale acustico di frequenza tanto più alta quanto più piccola è la distanza assoluta dal numero-nascondiglio. Ciò rende possibile la sua localizzazione.

Durante l'avvicinamento però, ogni passo che non sia più breve del precedente provoca la “fuga” della preda verso un altro nascondiglio, generalmente più in basso nella pagina. Questo fatto è segnalato dalla scomparsa delle cifre sul display e il nuovo segnale acustico che

si accompagna serve ora di localizzazione, come all'inizio.

Un cambiamento di nascondiglio può avvenire anche all'atto di scrivere una singola cifra, se si verificano certe condizioni imprevedibili e persino mentre si è... in agguato, trattandosi di una preda

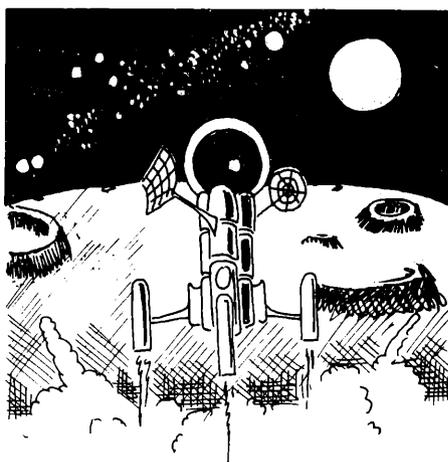
che non ama rimanere a lungo nello stesso buco.

Il gioco richiede molta pazienza e perspicacia, ma grande è la soddisfazione che si prova udendone gli schiamazzi, quando si riesce a inchiodare la preda nel suo nascondiglio!

Programma

0200	=E6	F9	E6	FA	E6	FB	20	0C	FF	E6	04	20	DA	FE	20	57
0210	=FF	C9	14	F0	02	D0	E9	A9	7F	85	08	A2	05	A9	08	95
0220	=8F	CA	10	FB	A5	04	85	07	85	03	A9	00	85	06	E6	04
0230	=20	7E	FF	E6	01	D0	F9	A9	92	8D	03	FD	A0	FF	A6	03
0240	=8C	02	FD	CA	D0	FD	C8	F0	F5	E6	01	D0	EF	20	DA	FE
0250	=20	57	FF	C9	10	10	07	AA	45	04	F0	BB	D0	0A	A5	04
0260	=29	2F	29	2F	F0	B1	D0	C6	8A	A2	04	06	F9	CA	D0	FB
0270	=05	F9	85	F9	29	0F	A8	B9	EA	FF	A6	06	95	93	A9	01
0280	=C5	06	F0	06	E6	06	D0	A6	D0	9E	A5	F9	38	E5	07	F0
0290	=1F	10	05	49	FF	18	69	01	AA	38	E5	03	10	05	49	FF
02A0	=18	69	01	C5	08	10	05	85	08	8A	D0	DC	A9	00	F0	B4
02B0	=A9	92	8D	03	FD	A0	FF	A6	00	8C	02	FD	CA	D0	FD	C8
02C0	=F0	F5	E6	00	D0	EF	A9	10	85	01	20	7E	FF	C6	01	D0
02D0	=F9	F0	D0													

Atterraggio lunare



Il programma inizia dalla locazione 0200.

Alla partenza ti trovi a 4500 piedi di altezza e stai scendendo. La discesa della tua navicella è provocata dall'attrazione della forza di gravità. Quando vuoi potrai vedere la quantità di carburante premendo il tasto "C" (carburante). Inizialmente disporrai di 800 lt. di carburante.

Gli ultimi due display a destra ti mostrano la velocità di discesa (o ascensione) della tua navicella.

È possibile regolare la discesa o la ascensione agendo sui tasti da "1" a "9" che agiscono sulla potenza dei motori.

ATTENZIONE: premendo il pulsante "0" i motori si spegneranno e non sarà possibile riaccenderli.

La potenza dei motori ottenuta con il tasto "1", minimo, è decisamente bassa per tanto essi consumeranno pochissimo carburante, ma la forza di gravità attirerà la tua navicella molto velocemente verso la luna.

La potenza dei motori ottenuta premendo il tasto "9", massimo, è molto alta consuma molto carburante, ma vince la forza di gravità e riduce prontamente la velocità di discesa.

Premendo il pulsante "5" esso contrasta esattamente la forza di gravità e con esso è possibile proseguire la discesa o l'ascensione a velocità costante.

Se si termina il carburante i comandi relativi ai motori diverranno inoperativi.

Per un atterraggio sicuro la velocità di discesa dovrà essere di 05 o inferiore a tale valore.

Dopo l'atterraggio i motori si spegneranno automaticamente e di relativi controlli diverranno inoperativi, ma premendo il tasto "C" potrai vedere la quantità di carburante rimasta.

Premere "RUN" per effettuare un nuovo volo.

Suggerimenti per un atterraggio sicuro

- Per conservare il carburante inizialmente premi "1", scenderai molto velocemente.
- Quando la velocità sarà di "90" premi il tasto "5" per stabilizzare questa velocità.
- Quando l'altitudine sarà di circa 1500

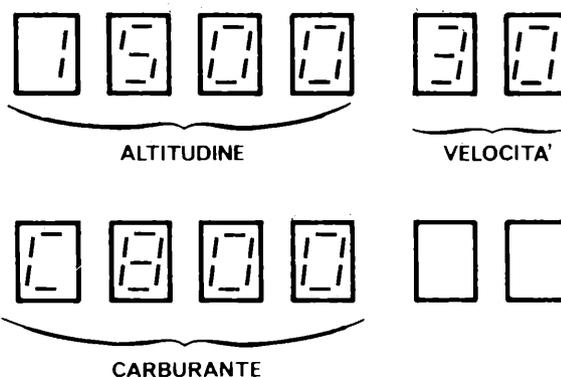
piedi premi il tasto "9" per ridurre la velocità di discesa.

- Quando la velocità sarà scesa al valore 15 puoi stabilizzarla premendo il tasto "5". Poi sta a te proseguire felicemente il volo.
- Dopo aver visualizzato con "C" il carburante premere "A" per tornare a visualizzare l'altitudine.

Programma

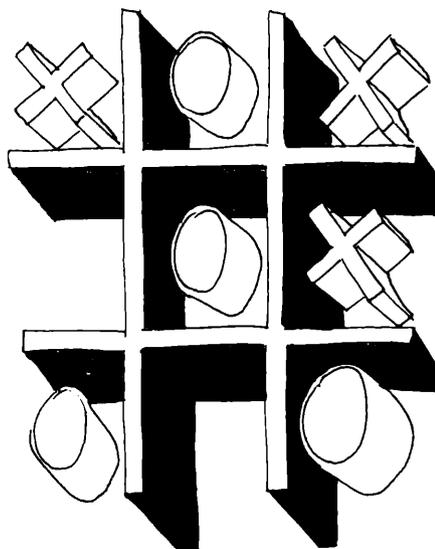
0200	=A2	0D	BD	CC	02	95	D5	CA	10	F8	A2	05	AD	01	F8	18
0210	=B5	D5	75	D7	95	D5	CA	88	10	F6	B5	D8	10	02	A9	99
0220	=75	D5	95	D5	CA	10	E5	A5	D5	10	0D	A9	00	85	E2	A2
0230	=02	95	D5	95	DB	CA	10	F9	38	A5	E0	E5	DD	85	E0	A2
0240	=01	B5	DE	E9	00	95	DE	CA	10	F7	B0	0C	A9	00	A2	03
0250	=95	DD	CA	10	FB	20	BD	02	A5	DE	A6	DF	09	CO	A4	E1
0260	=F0	20	F0	9C	F0	A4	A2	FE	AD	5A	18	A5	D9	69	05	A5
0270	=D8	69	00	B0	04	A2	AD	AD	DE	98	A4	E2	F0	04	A5	D5
0280	=A6	D6	85	FB	86	FA	A5	D9	A6	D8	10	05	38	A9	00	E5
0290	=D9	85	F9	A9	02	85	E3	D8	20	0C	FF	20	57	FF	C9	13
02A0	=F0	C0	B0	03	20	AD	02	C6	E3	D0	ED	F0	B7	C9	0A	90
02B0	=05	49	0C	85	E1	60	AA	A5	DD	F0	FA	86	DD	A5	DD	38
02C0	=F8	E9	05	85	DC	A9	00	E9	00	85	DB	60	45	01	00	99
02D0	=81	00	99	97	02	08	00	00	01	01						

DISPLAY



Esempio di indicazioni sul display durante il gioco dell'atterraggio lunare.

Filetto

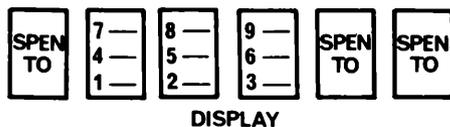


Questa è una versione del "filetto" per l'AMICO 2000, il giocatore gareggia con il computer. La matrice del gioco è quella riportata nella figura a piede pagina.

Per vincere è necessario, come sempre, riuscire a posizionare tre segmenti in fila, orizzontali, verticali od obliqui. Le mosse dell'AMICO 2000 sono rappresentate da segmenti lampeggianti, quelle del giocatore da segmenti fissi. Premendo RUN il computer gioca per primo volendo il giocatore la prima mossa premerà "1". Il programma parte da 0100.

Il computer possiede in questo programma un "quoziente d'intelligenza" che inizialmente è del 75% (loc. 0311 - 0C) che sale se la macchina perde e scende se vince. Il giocatore può comunque, alla fine di ogni gioco, modificarlo premendo "REG" il display mostra la parola "ODDS" seguita dal valore del "Q-I" che il calcolatore ha in quel momento : impostare quindi in tastiera il nuovo valore (00 minimo - 0F massimo) e premere "DA" per tornare al gioco.

7	8	9
4	5	6
1	2	3

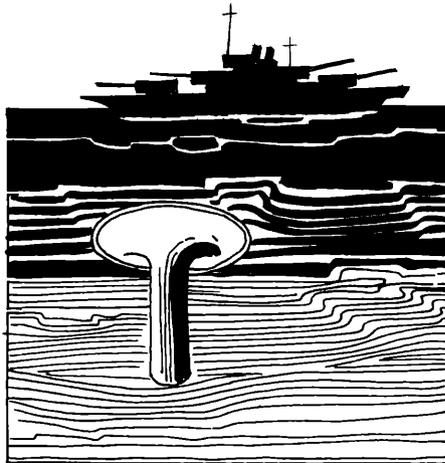


Corrispondenza fra tasti e segmenti delle cifre nel "gioco del filetto".

Programma

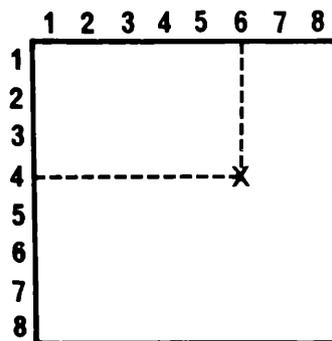
0100	=4C	10	03	EA	EA	EA	A9	20	15	BF	95	BF	6D	EA	EA	08
0110	=08	08	4D	4D	4D	01	01	01	01	04	07	01	02	03	01	03
0120	=02	05	08	04	05	06	05	05	03	06	09	07	08	09	09	07
0130	=85	D9	A2	09	A5	D9	35	DB	24	D9	D0	03	CA	D0	F5	6D
0140	=B5	BF	D0	02	F6	DB	6D	95	BF	A0	08	A9	0D	99	C8	0D
0150	=BE	17	01	2D	8A	03	BE	1F	01	2D	8A	03	BE	27	01	2D
0160	=8A	03	88	D0	E6	6D										
0200	=A9	0D	A2	1D	95	B4	CA	D0	FB	A9	05	85	BB	A0	04	2D
0210	=F2	03	A2	04	D5	BB	F0	F7	CA	D0	F9	99	BB	0D	88	D0
0220	=EE	E6	B6	A0	04	2D	F2	03	A2	05	D5	B6	F0	F7	CA	D0
0230	=F9	99	B6	0D	88	D0	EE	A9	03	A0	08	D9	C8	0D	F0	05
0240	=88	D0	F8	F0	15	BE	17	01	2D	06	01	BE	1F	01	2D	06
0250	=01	BE	27	01	2D	06	01	4C	FE	02	A2	09	A9	C0	35	BF
0260	=F0	0E	CA	D0	F7	A2	09	2D	06	01	CA	D0	FA	4C	15	03
0270	=E6	B5	A5	DB	D0	17	2D	A6	03	F0	FB	C9	0A	B0	F7	AA
0280	=B4	BF	D0	F2	A9	4D	2D	47	01	E6	DB	D0	AA	2D	4C	03
0290	=E6	D1	D0	F9	A9	08	2D	C8	03	A9	02	2D	C8	03	A9	04
02A0	=2D	C8	03	A9	01	2D	C8	03	A9	C0	2D	3D	01	D0	43	A9
02B0	=3D	2D	3D	01	D0	3C	A9	08	2D	3D	01	D0	35	2D	B3	03
02C0	=29	0F	C5	D2	B0	1F	A4	B5	C0	01	D0	04	29	01	D0	17
02D0	=C0	04	D0	06	24	C4	3D	0D	7D	07	A9	02	2D	3D	01	D0
02E0	=11	A0	05	D0	02	A0	D9	B6	B6	B5	BF	F0	05	88	D0	F7
02F0	=F0	F3	A9	8D	2D	47	01	C6	DB	A9	0C	4C	39	02	A5	DB
0300	=D0	04	C6	D2	1D	0F	E6	D2	A9	1D	C5	D2	9D	F4	B0	05
0310	=A9	0C	85	D2	D8	2D	A6	03	A0	01	C9	13	F0	28	88	C9
0320	=12	F0	23	C9	14	D0	EE	A9	0D	85	FB	A9	D5	85	FA	A5
0330	=D2	85	F9	2D	0C	FF	2D	EB	FE	2D	57	FF	C9	11	F0	D5
0340	=B0	E5	85	D2	9D	E1	84	DB	4C	0D	02	EA	A9	89	8D	03
0350	=FD	E6	DA	A0	0D	A2	0B	B9	C0	0D	85	FC	F0	14	29	2D
0360	=F0	04	24	DA	7D	0C	A5	FC	29	4D	D0	0A	A5	DA	29	08
0370	=F0	04	A9	0D	F0	03	B9	0F	01	84	FC	2D	3B	FF	C8	C0
0380	=09	F0	06	E0	11	F0	CE	D0	CE	6D	B5	BF	85	D9	24	D9
0390	=3D	06	7D	08	A9	0D	F0	06	A9	04	D0	02	A9	01	18	79
03A0	=C8	0D	99	C8	0D	6D	2D	4C	03	2D	DA	FE	F0	F8	2D	57
03B0	=FF	AA	6D	D8	38	A9	D4	65	D7	65	D8	85	D3	A2	04	B5
03C0	=D3	95	D4	CA	1D	F9	6D	EA	85	D9	A2	09	16	DB	16	DB
03D0	=CA	D0	F9	A0	08	A5	D9	D9	C8	0D	D0	12	BE	17	01	2D
03E0	=4D	01	BE	1F	01	2D	4D	01	BE	27	01	2D	4D	01	88	D0
03F0	=E4	6D	2D	B3	03	29	0E	05	B6	F0	F7	C9	0D	FE	6F	F5

Battaglia navale

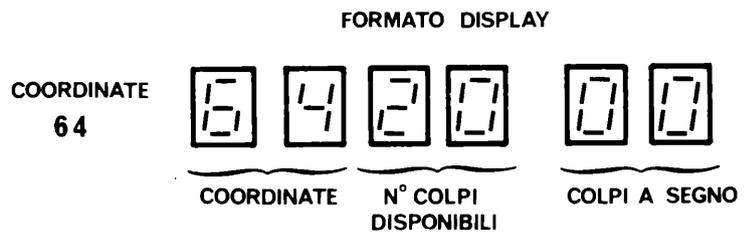


L'AMICO 2000 posiziona casualmente in un quadrato 8x8 quadretti una nave da 3 quadretti. La nave potrà essere in posizione orizzontale, verticale od obliqua. Il display mostra da sinistra (2 digits) le coordinate, il numero dei colpi ancora disponibili, e quante volte la nave è stata colpita. Per ogni battaglia sono disponibili 20 colpi, se, dopo i venti colpi la nave non è stata colpita il computer ne rivela le coordinate. Dopo l'impostazione delle coordinate premere il tasto "F" (fuoco), affondando la nave il display mostra la parola "DEAD" e di seguito il numero di colpi impiegati. Per una nuova battaglia AD 0200 - RUN.

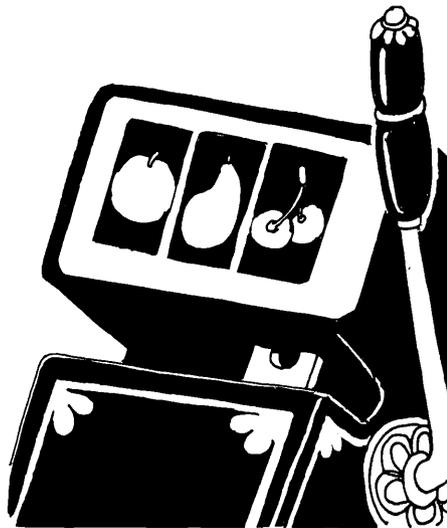
Programma																
0200	=A9	02	85	00	A9	00	85	E8	A2	99	A9	02	95	00	CA	D0
0210	=F9	A9	11	85	E7	85	E3	A2	07	18	A0	07	A9	00	91	E7
0220	=88	10	F8	F8	A5	E7	69	10	85	E7	CA	10	EC	F8	38	A5
0230	=EA	65	ED	65	EE	85	E9	A2	04	B5	E9	95	EA	CA	10	F9
0240	=38	E9	99	B0	E8	A5	EB	29	06	C9	00	F0	41	C9	02	F0
0250	=36	C9	04	F0	19	18	A0	02	A6	E9	B5	00	C9	02	F0	B1
0260	=A9	01	95	00	8A	69	09	AA	88	10	EF	4C	95	02	A0	02
0270	=A6	E9	B5	00	C9	02	F0	99	A9	01	95	00	8A	38	E5	E3
0280	=AA	88	10	EE	4C	95	02	A9	10	85	E3	4C	6E	02	A9	01
0290	=85	E3	4C	6E	02	A9	20	85	FA	A9	00	85	F9	85	E4	85
02A0	=FB	85	E6	D8	20	0C	FF	20	57	FF	C9	0F	F0	37	C9	09
02B0	=10	F1	C9	00	F0	ED	85	E5	A5	E6	C9	01	F0	16	E6	E6
02C0	=06	E5	06	E5	06	E5	06	E5	A5	E5	85	FB	20	2D	FF	D0
02D0	=FB	4C	A3	02	18	A5	E5	65	FB	85	FB	C6	E6	20	2D	FF
02E0	=D0	FB	4C	A3	02	A5	FB	C5	E4	F0	07	AA	B5	00	C9	01
02F0	=F0	17	F8	A5	FA	38	E9	01	F0	36	85	FA	D8	A5	FB	85
0300	=E4	20	2D	FF	D0	FB	4C	A3	02	E6	F9	A5	F9	C9	03	F0
0310	=08	20	2D	FF	D0	FB	4C	F2	02	F8	A9	21	38	E5	FA	85
0320	=F9	D8	A9	DE	85	FB	A9	AD	85	FA	20	0C	FF	4C	2A	03
0330	=AD	02	A2	99	B5	00	C9	01	F0	06	CA	D0	F7	4C	48	03
0340	=8A	99	F9	00	88	4C	3A	03	20	0C	FF	4C	48	03		



Esempio di rappresentazione sul display di una fase del gioco della battaglia navale.



Slot machine



All'inizio del gioco sul display a destra sarà visualizzato il numero "25" che rappresenta i gettoni che l'AMICO 2000 ti dà per cominciare a giocare; premendo un tasto qualsiasi i tre display che simulano "le ruote" cambieranno velocemente il loro contenuto e, dopo qualche secondo si fermeranno, uno per volta in successione da sinistra a destra formando una combinazione di segni. Se la combinazione è vincente sarà incrementato il totale dei gettoni.

Ad ogni giocata sarà sottratto dal totale un gettone.

La combinazione che "paga" meglio è costituita da tre segmenti al centro dei display in questo caso saranno accreditati 15 gettoni; ci sono, naturalmente, altre combinazioni che incrementano il totale.

La massima vincita è rappresentata da 99 gettoni; se il giocatore dovesse restare senza gettoni l'AMICO 2000... non fa credito.

Programma	
0200	=A9 25 85 05 20 BA 02 A9 00 85 06 20 8D 02 D0 FB
0210	=E6 09 20 8D 02 F0 F9 A9 03 85 06 F8 38 A5 05 E9
0220	=01 85 05 20 BA 02 26 09 20 8D 02 C6 08 D0 F9 A6
0230	=06 A5 09 29 06 09 40 95 01 46 09 46 09 C6 06 D0
0240	=E7 A5 04 C5 03 D0 37 C5 02 D0 33 A2 10 C9 40 F0
0250	=0D A2 08 C9 42 F0 07 A2 06 C9 44 F0 01 CA 86 07
0260	=A9 80 85 08 20 8D 02 C6 08 D0 F9 C6 07 F0 9C 18
0270	=F8 A5 05 69 01 B0 94 85 05 20 BA 02 D0 E2 A2 03
0280	=C9 46 F0 DA 20 8D 02 A5 05 D0 80 F0 F7 A6 06 10
0290	=02 F6 02 CA 10 FB A9 89 8D 03 FD AD 0A A2 04 B5
02A0	=00 8C 01 FD 8D 00 FD D8 A9 89 E9 01 D0 FC 8D 01
02B0	=FD C8 C8 CA 10 E9 20 2D FF 60 A5 05 29 0F AA BD
02C0	=EA FF 85 00 A5 05 4A 4A 4A 4A AA BD EA FF 85 01
02D0	=60

Combinazioni vincenti della slot machine

= 2 GETTONI	= 2 GETTONI	= 2 GETTONI	= 2 GETTONI
= 2 GETTONI	= 2 GETTONI	= 2 GETTONI	= 2 GETTONI
= 7 GETTONI	= 2 GETTONI	= 2 GETTONI	= 2 GETTONI
= 2 GETTONI	= 2 GETTONI	= 15 GETTONI	= 4 GETTONI

E se vuoi crescere fino al Personal Computer

CEDOLA PER L'ACQUISTO DI UN MICROCOMPUTER
"AMICO 2000"

(Da spedire in busta chiusa a **A.S.EL. s.r.l.** Via Cortina D'Ampezzo 17, 20139 MILANO
- Tel. 02-5391719)

Sono interessato ad acquistare:

Microcomputer AMICO 2000/1K in scatola di montaggio ⁽¹⁾
(fornito con 1 Kbyte di memoria RAM e interfaccia per registratore a cassette):
Lit. 284.400 (IVA inclusa).

Microcomputer AMICO 2000/2 montato e collaudato ⁽²⁾
(caratteristiche come il precedente): **Lit. 348.000** (IVA inclusa).

Prego spedirmi direttamente quanto ordinato (pagherò contrassegno al ricevimento del pacco) al sottoindicato indirizzo o di segnalarmi il vostro rivenditore più vicino alla mia città.

ATTENZIONE: la segnalazione avverrà solo se il distributore locale è presente, **in ogni altro caso si procederà all'invio del materiale ordinato** con pagamento in contrassegno.

Sono interessato inoltre a maggiori dettagli sulle schede di espansione del sistema AMICO 2000.

Scrivere in stampatello e compilare in ogni voce:

NOME COGNOME.....

CODICE FISCALE TELEFONO.....

VIA

C.A.P.....CITTA'

IMPORTANTE: La merce viaggia a rischio e pericolo del Commitente: è possibile assicurarla aggiungendo Lit. 2.000 per ogni 50.000 di valore assicurato.

(1) La scatola di montaggio è coperta da una forma di "funzionamento garantito" per cui in caso di insuccesso nella realizzazione è possibile inviare la piastra con tutti i componenti al costruttore, che la sostituirà con una montata e collaudata dietro il pagamento di una quota fissa di Lit. 50.000 qualora il difetto risieda in un errore di montaggio.

(2) Il microcomputer montato e collaudato è garantito per 3 mesi.





è un sistema espandibile

Se vuoi approfondire le tue conoscenze nel settore dei computer o se desideri un potente sistema di elaborazione, l'AMICO 2000 ti aiuta a crescere fino al personal computer.

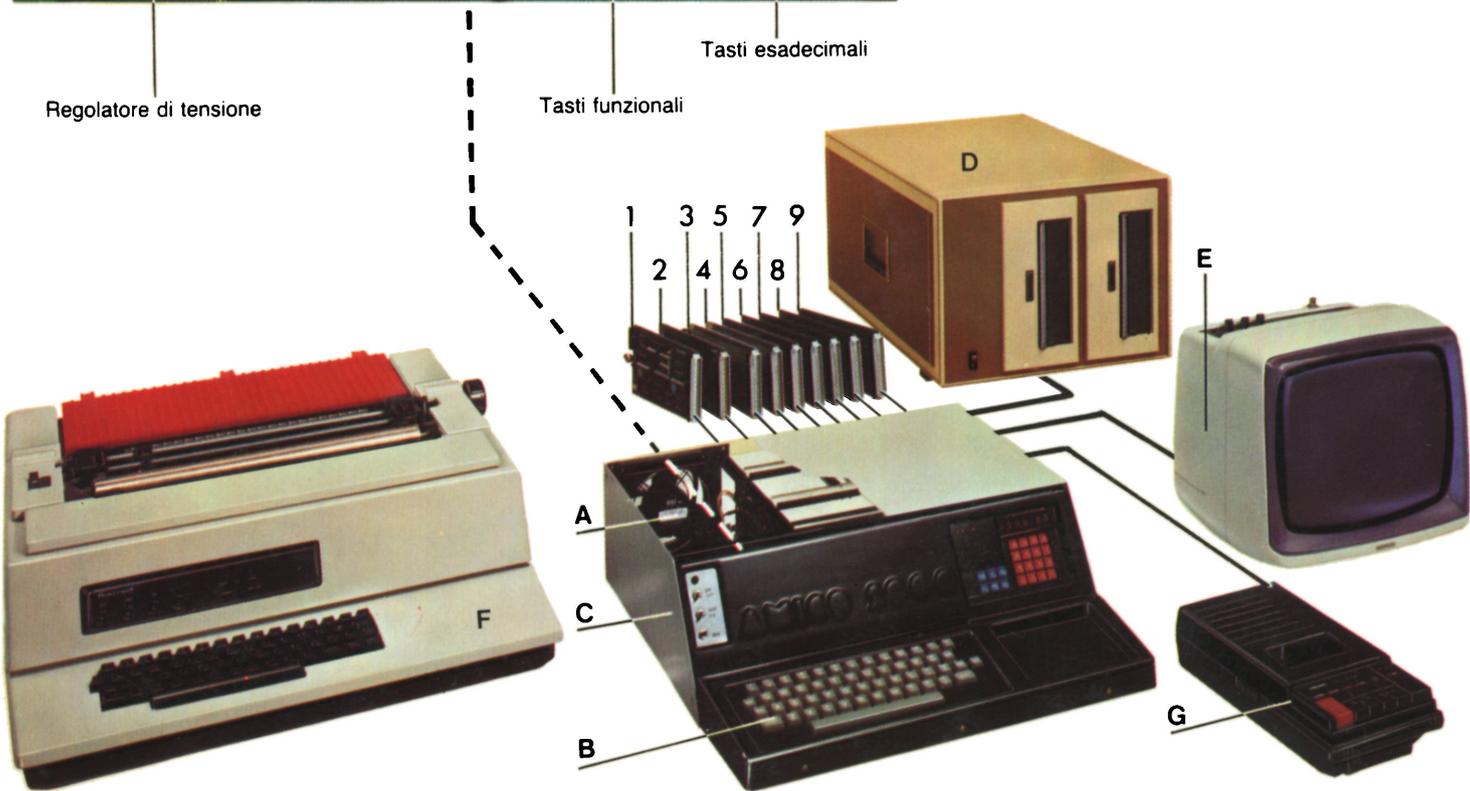
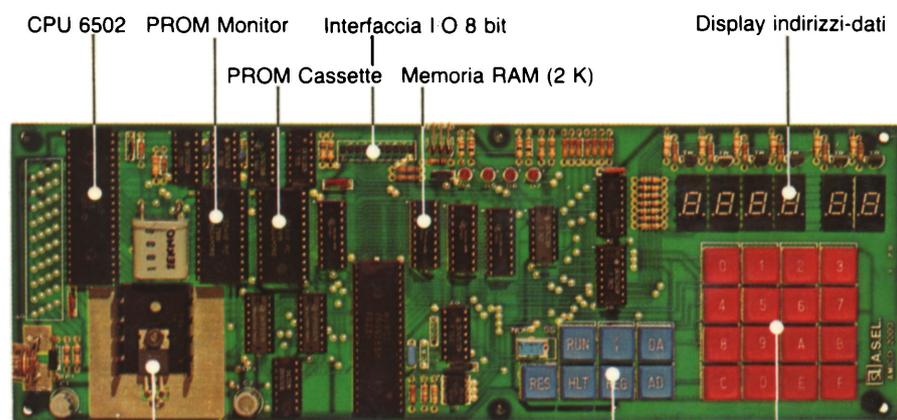
Dalla scheda base puoi arrivare a realizzare un potente sistema di elaborazione espandendolo con i moduli aggiuntivi e compatibilmente anche con le tue esigenze di ordine economico.

L'AMICO 2000 è un sistema professionale: ogni scheda di espansione è realizzata con i componenti più avanzati su formato normalizzato europeo EUROCARD (160 x 100 mm).

Ogni configurazione intermedia fino alla massima trova posto in un razionale contenitore completo di cestello portaschede e di tastiera alfanumerica.

legenda sistema espanso

- A Alimentatore di potenza
- B Tastiera alfanumerica standard ASCII
- C Contenitore metallico per l'intero sistema completo di cestello porta schede
- D Unità a floppy disk
- E Monitor TV ad alta definizione e linearità
- F Stampante
- G Registratore a cassette
- 1 Interfaccia video
- 2 MiniBASIC e BASIC da 8K
- 3 interfaccia per doppio registratore a cassette
- 4 Espansione RAM 16K byte
- 5 Interfaccia floppy disk
- 6 Interfaccia di comunicazione seriale
- 7 I/O analogico
- 8 I/O digitale
- 9 Altre espansioni



espandibilità del sistema



