COSTRUIAMOCI UN VERO MICROELABORATORE

HOME COMPUTER AMICO 2000

Poter usare sempre meglio e a fondo il microelaboratore AMICO 2000A significa soprattutto conoscerne il linguaggio e le metodologie di programmazione. Per questo anche questa parte è dedicata al software. Verranno spiegati gli ultimi due registri della CPU 6502 ed il loro utilizzo. Questa volta poi entreremo nel vivo della programmazione spiegandovi il concetto di "Subroutine" e le istruzioni relative ad essa. Molti dei brevi programmi presentati in questo articolo non sono soltanto dimostrativi, ma possono essere conservati e utilizzati appunto come subroutine nella elaborazione di programmi più complessi: è il caso della SOMMA di due numeri da 16 bit o della MOLTIPLICAZIONE. Un giochino, sempre per rendere piacevole l'utilizzo del vostro microelaboratore, chiude questo articolo: si tratta della "Corsa dei cavalli", una piccola San Siro su scheda!

a cura della A.S.E.I. s.r.l. - parte settima

SOFTWARE

Registri

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 questa trattazione.

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 articolo questi sistemi di indirizzamento; per una prima spiegazione si veda comunque la tabella riassuntiva delle istruzioni del 6502 pubblicata nella sesta parte (Sperimentare n. 7/8 1979).

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 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.

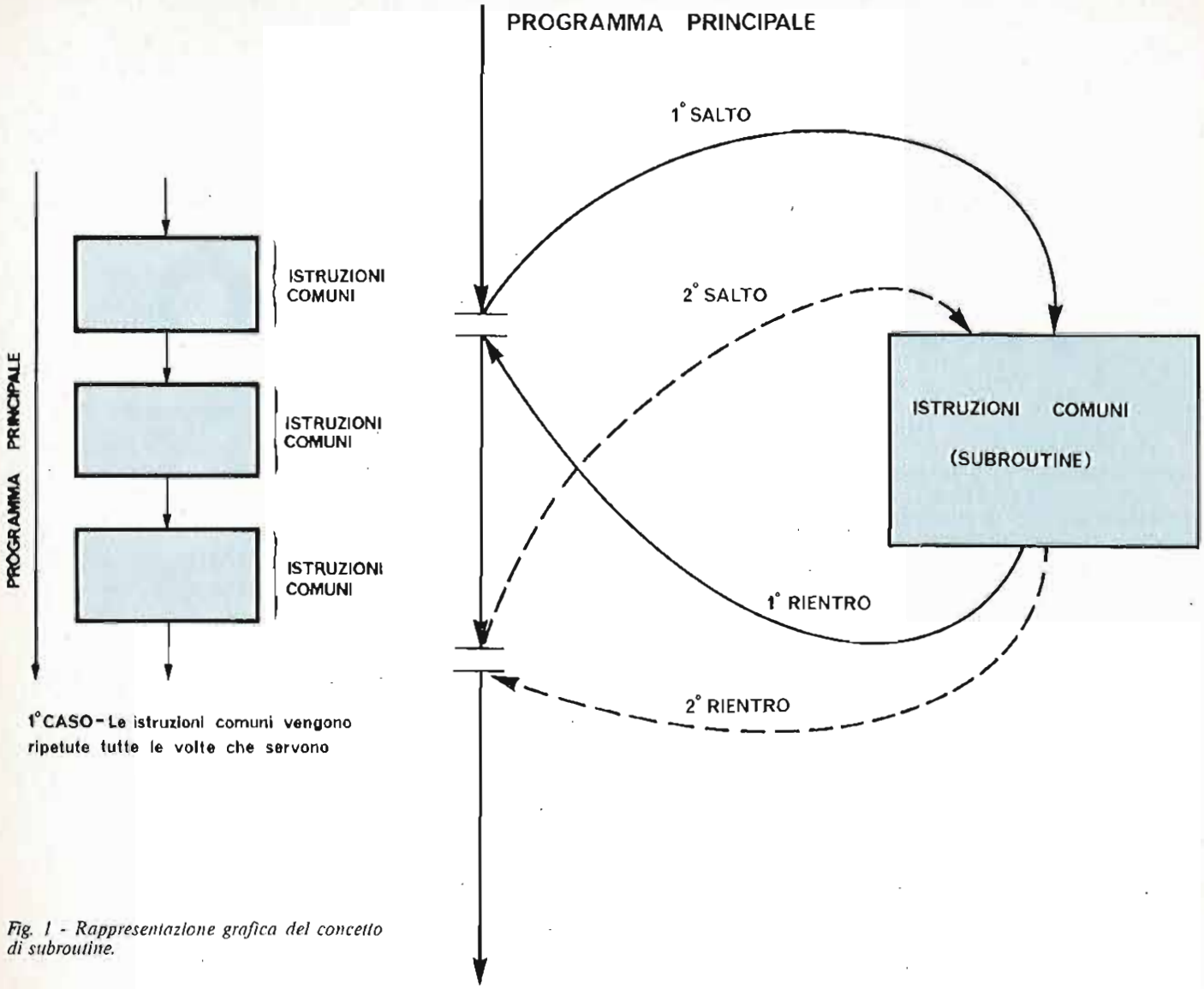
DEY (Decrement Y). Con questa istruzione si decrementa di uno (sempre con il sistema esadecimale) il contenuto del registro indice Y. Il suo sistema di indirizzamento è ovviamente implicito (quando si scrive questa istruzione non si deve precisare nulla di più alla CPU), il codice operativo è 88, dalla 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 (cioé -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.



1°CASO - Le istruzioni comuni vengono ripetute tutte le volte che servono

Fig. 1 - Rappresentazione grafica del concetto di subroutine.

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 inflenzati. 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=Ø e sia contenuto nella locazione ØØØ5 il dato Ø7.

Scriviamo il programma:

ØZØØ	Loop	C6	DEC $Ø5	Decrementa il contenuto della locazione di memoria ØØØ5
2		84	STY $Ø6	Memorizza il contenuto di Y nella ØØØ6
4		DØ	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.

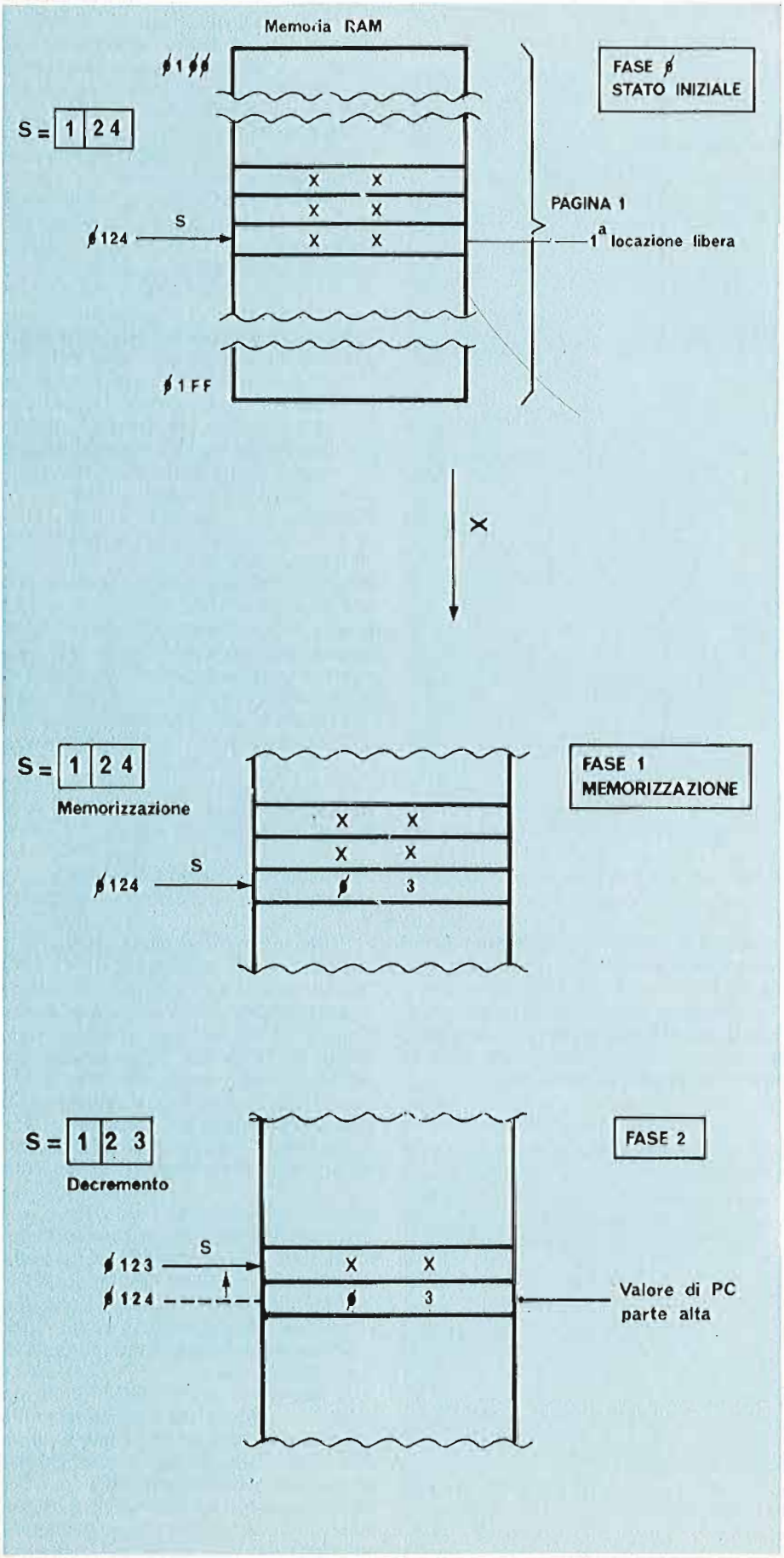
Poniamo che sia il contenuto della locazione di memoria ØØØ5 uguale a Ø1 e quello della locazione ØØØ6 uguale a Ø7.

Scriviamo il programma:

ØZØØ	Loop	C6	DEC $Ø5	Stesso commento programma precedente)
2		A4	LDY $Ø6	Carica in Y il contenuto della locazione di memoria ØØØ6 (che è 7 ≠ Ø!)
4		FØ	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

Rappresentazione grafica del funzionamento dello stack Pointer. (segue prossima pagina)



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 verifichiamo nella pratica, con l'AMICO 2000/A, la funzione e l'uso di alcune delle istruzioni appena viste scrivendo un programma che fa accendere 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 articolo.

Analizziamo brevemente di questo programma, il cui listing è riportato nella Tab. 1 riportata a pagina seguente, 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 articolo).

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

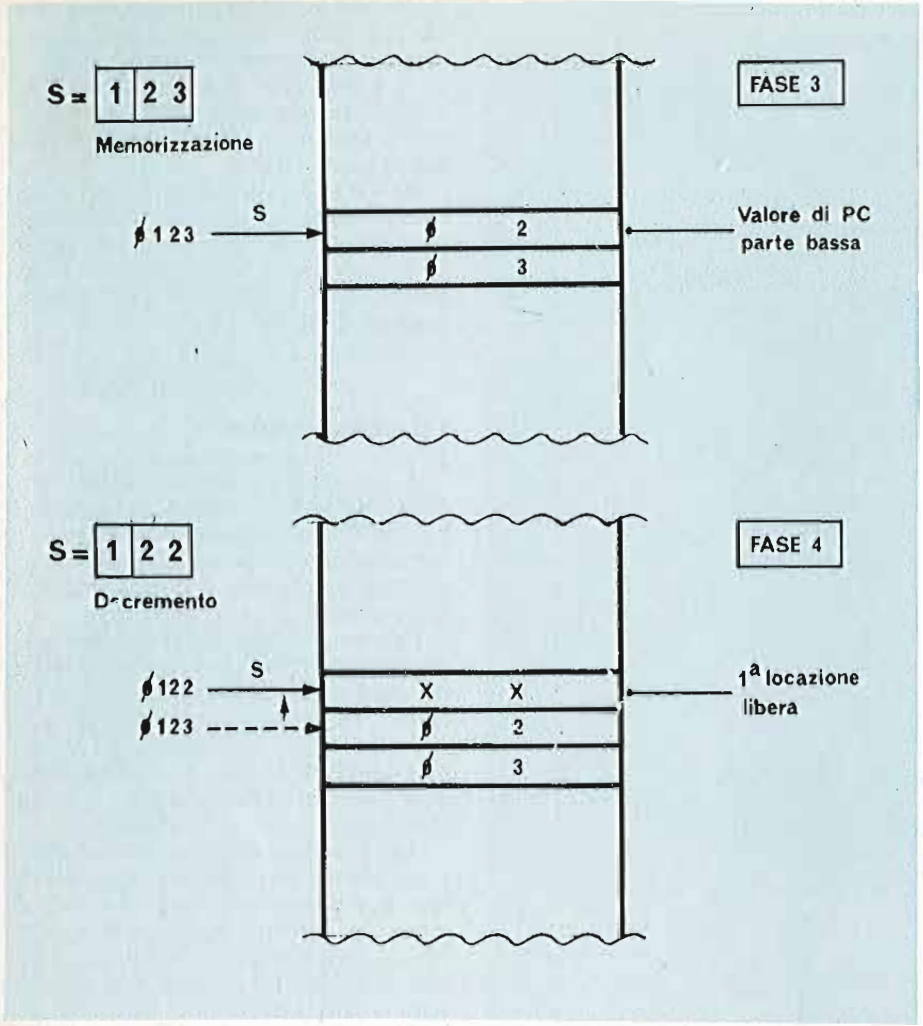


Fig. 2 - Rappresentazione grafica del funzionamento dello stack Pointer.

dell'intermittenza), mentre il tempo massimo di impulso si ha se scriviamo ØØ. Perche? Lasciamo a voi la risposta.

Come ultima particolarità facciamo notare che per passare alternativamente da ØØ a FF (cioé acceso/spento del display) eseguiamo una negazione tramite l'istruzione EOR =$FF del contenuto della locazione di memoria ØF (ricordiamo che ØØ EOR FF = FF; FF EOR FF = ØØ).

Dobbiamo puntualizzare infine per i lettori più sofisticati che lo stesso programma potrebbe essere scritto con un numero inferiore di istruzioni.

Tabella 1 - Programma per far lampeggiare il display

Ø2ØØ		A9 LDA #ØØ	1	2Ø JSR FF7E
1		ØØ	2	7E
2		85STA $ØF	3	FF
3		ØF	4	A4 LDY $1Ø
4	Loop 2	A5LDA $ØF	5	1Ø
5		ØF	6	88 DEY
6		A2LDX #Ø6	7	DØ BNE Loop 1
7		Ø6	8	F6
8		95STA 8E,X	9	A5 LDA $ØF
9		8E	A	ØF
A		CA DEX	B	49 EOR #$FF
B		DØ BNE Loop	C	FF
C		FB	D	85 STA $ØF
D		AØ LDY #$FF	E	ØF
E		FF	F	4C JMP Loop 2
F	Loop 1	84 STY $1Ø	Ø220	Ø4
Ø21Ø		1Ø	Ø221	Ø2

Uso del registro indice Y

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 che forma il 2° e 3° byte dell'istruzione, al contenuto del registro Y. Facciamo un esempio: sia Y = 3F.
l'operazione:

79 ADC Ø2Ø3,Y
Ø3
Ø2

esegue la somma del contenuto dell'accumulatore con il contenuto della locazione di memoria Ø2Ø3 + Y = Ø2Ø3 + + 3F = Ø242.

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 portate 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 ØØ14, ricordiamo che qui si è in pagina Ø).

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. 1).

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 accendiamo 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 locazion 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
```

Allora il Programm 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).

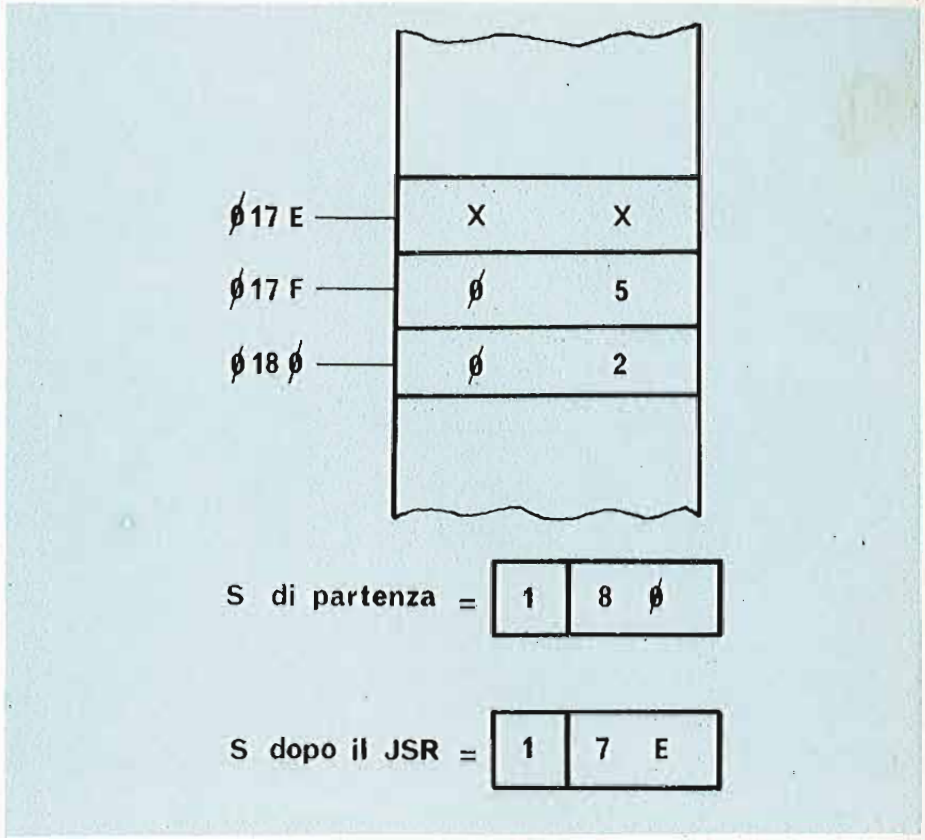


Fig. 3 - Esempio di salvataggio del PC e modifica del contenuto dello Stack Pointer dopo l'esecuzione dell'istruzione JSR.

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. 2).

La CPU incontra l'istruzione:

```
0300   20   JSR   0280
0301   80
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 2.

FASE 1 - La parte alta del PC (03) viene memorizzata (salvata) nella locazione puntata da S.

FASE 2 - S viene decrementato di uno.

FASE3 - 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.

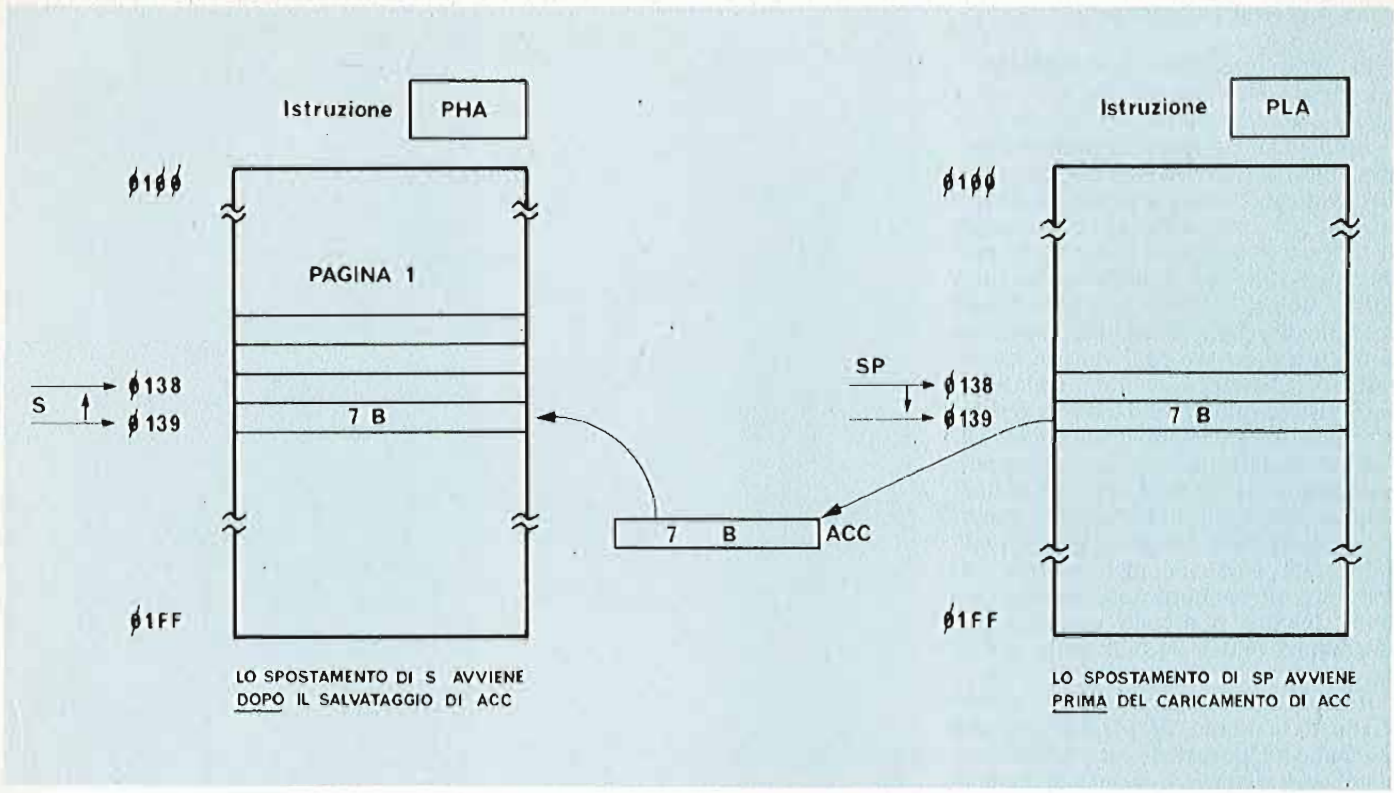


Fig. 4 - Rappresentazione grafica del funzionamento delle istruzioni PHA e PLA.

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. 2, 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!).

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 1ª, 2ª, 3ª, 4ª, 5ª, 6ª 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 corrispondete. 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 articolo.

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 1ª istruzione e X nella 2ª) 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 Ø18F, dove Ø1 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 Ø200.

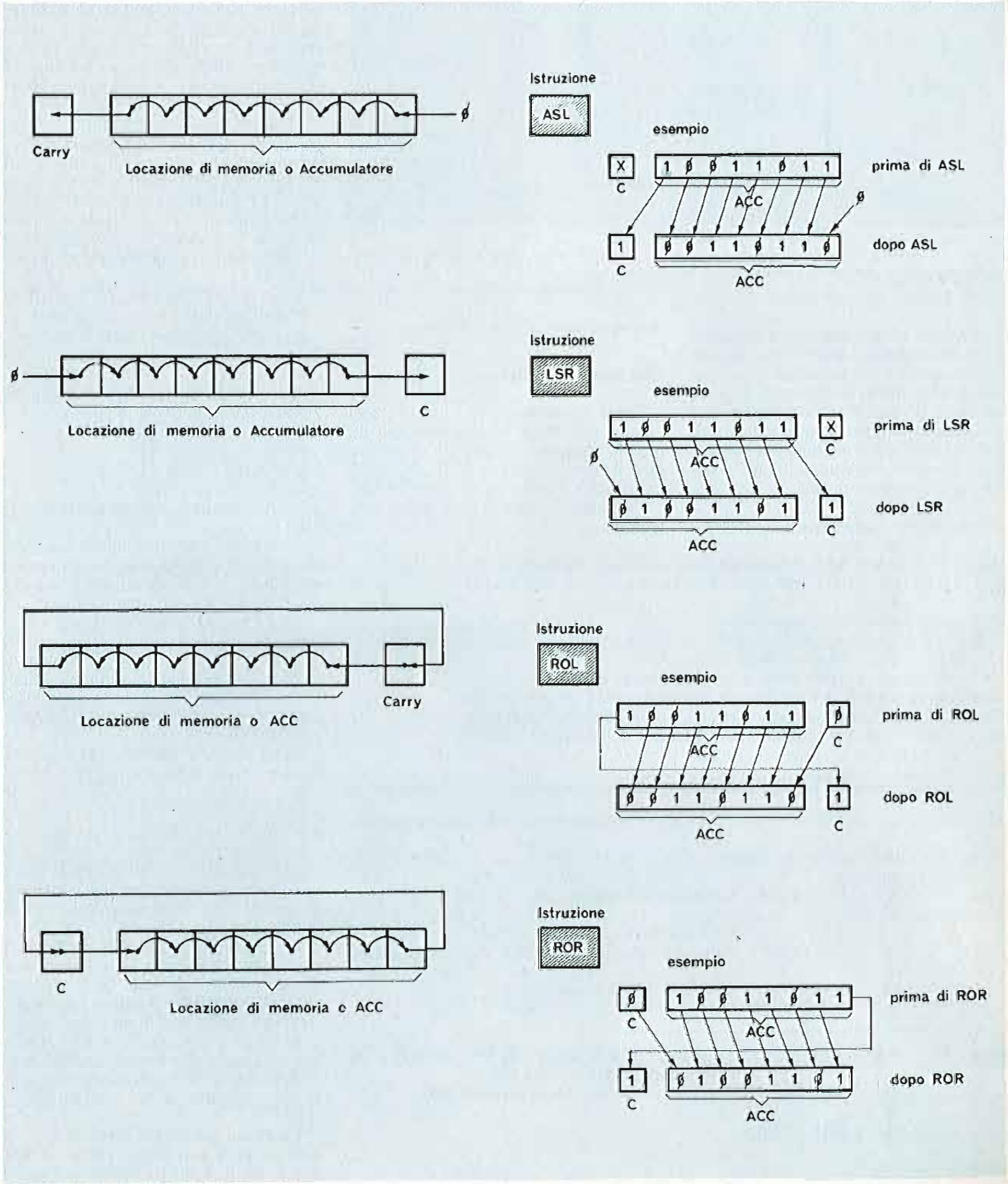


Fig. 5 - Rappresentazione grafica del funzionamento delle istruzioni ASL, LSR, ROL e ROR. Movimenti eseguiti sui bit.

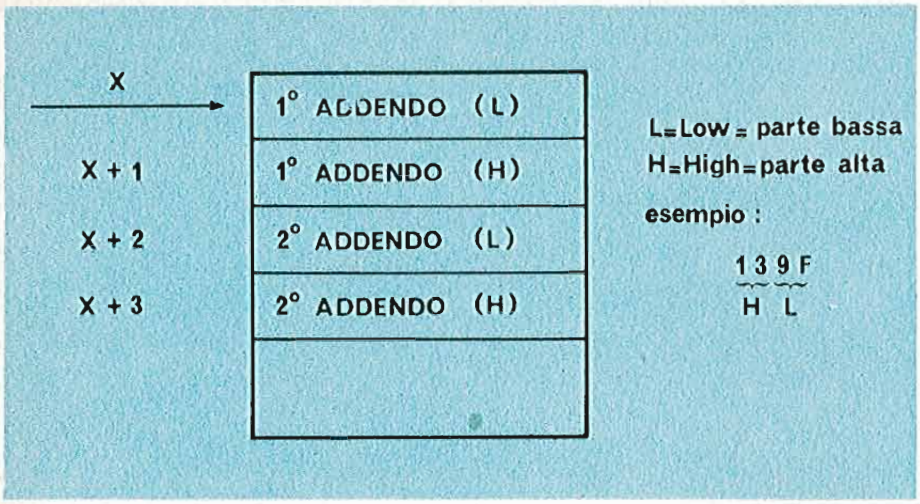


Fig. 6 - Lo schema rappresenta come sono assegnate quattro successive locazioni di memoria per eseguire una somma di due numeri da 16 bit ciascuno.

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:

```
0000    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
                            l'esecuzione dell'istruzione JSR
   1    86      STX $00     Memorizzo questo valore nella loc. 0000
   2    00
   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 01 (PC parte alta) (vedi Fig. 3).

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. 4).

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. 4).

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 perchè 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    TSX          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.

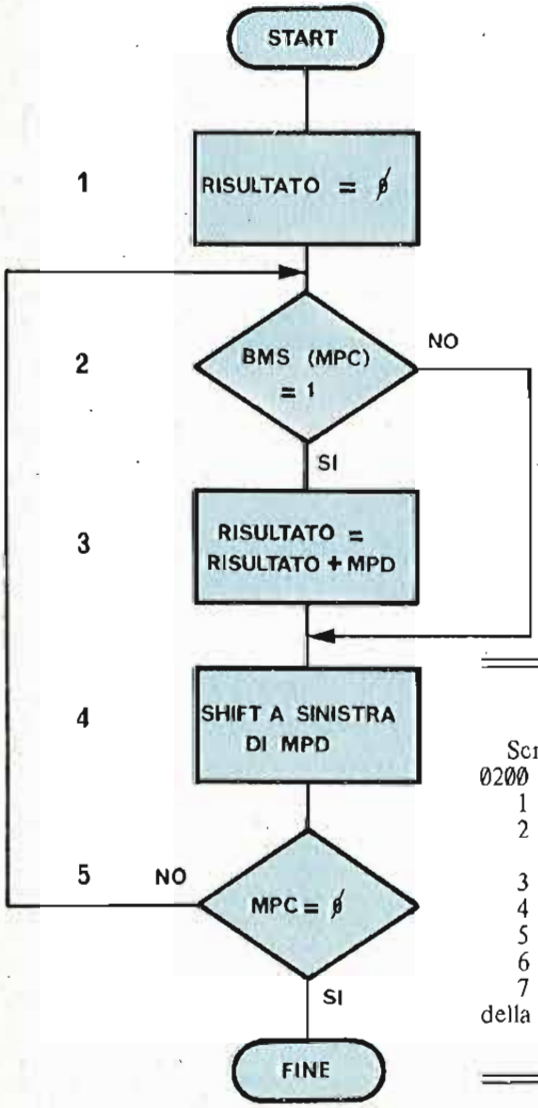


Fig. 7 - Diagramma di flusso del programma per eseguire una moltiplicazione.

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
```

della numerazione binaria, come 10 lo

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 x 5 = (N x 2) x 2 + N

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 (Aritmetic 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 5.

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 x 10 =

7390 ← 0

La stessa cosa avviene per i numeri binari. Moltiplicare per 2 (che è la base della numerazione binaria, come 10 lo 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 x 2 = 76 prelevando il dato dalla locazione di memoria 0001.

Perciò per eseguire 0C x 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 x 99 = 9801, che è un numero da 4 cifre.

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. 6.

Il registro X è posizionato sul primo numero, ovvero X contiene l'indirizzo della locazione di memoria che contiene la parte bassa (meno siginficativa) 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
   7    02                del 2° ADDENDO
   8    B5    LDA 1,X    } Si ripetono le stesse operazioni per la parte alta.
   9    01                } Notiamo l'assenza della operazione CLC
   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:

```
          27 84 +
          15 52 =
riporto    1
          43 36
```

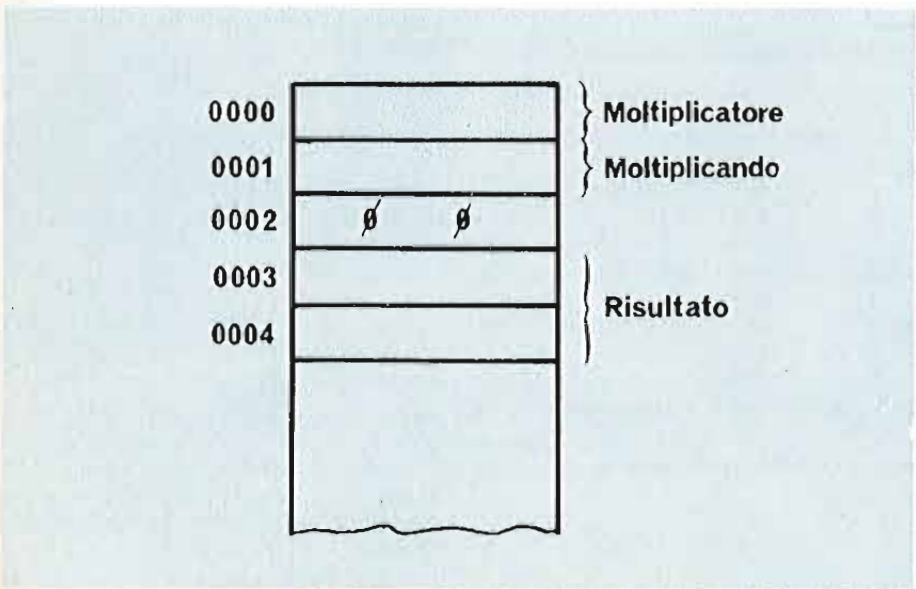


Fig. 8 - Locazioni di memoria usate nel programma per eseguire la moltiplicazione.

Per provare la nostra subroutine scriviamo ora il programma che segue:

```
0200    A2    LDX #00
```

Lavoriamo con le prime 4 locazioni di RAM

```
   1    00
   2    20    JSR 0300
```

Andiamo a eseguire la subroutine

```
   3    00
   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:

```
0013  +  0013  =  0026
3214  +  1F6A  =  517E
A999  +  0001  =  A99A
```

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 moltiplica sulla carta quando usiamo numeri nel sistema decimale.

```
 57 x
 14 =
228   1° passo si fa 57 x 4
570   2° passo si fa 57 x 1 e il risultato lo si moltiplica x 10 (Shift a sinistra)
798   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:

```
(CD)        11001101  x
(8D)        10001101  =
            11001101
           00000000
          11001101
         11001101
        00000000
       00000000
      00000000
     11001101
(70E9) 1110000111101001
       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. 7.

Commentiamolo: *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 si, 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. 8:

```
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             } Azzeramento del risultato
   8    95   STA 3,X  }
   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  }
   2    00             } Eseguo lo shift del moltiplicando
   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 verso sinistra; infatti il 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 ø2 øA) 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 necessaria 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 articolo 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 "monovrare" a vostro piacimento ogni singolo segmento delle cifre del display.

Un gioco: la corsa dei cavalli

Questa volta vi presentiamo un giochetto originale che utilizza i tre segmenti parallelo di ogni cifra come se fossero dei "cavalli" in corsa. La probabilità di vincere che ha ogni cavallo è causale e dipende in primo luogo dallo stesso programma; il bello è che si può intervenire direttamente "frustando" ogni cavallo per farlo correre di più. 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

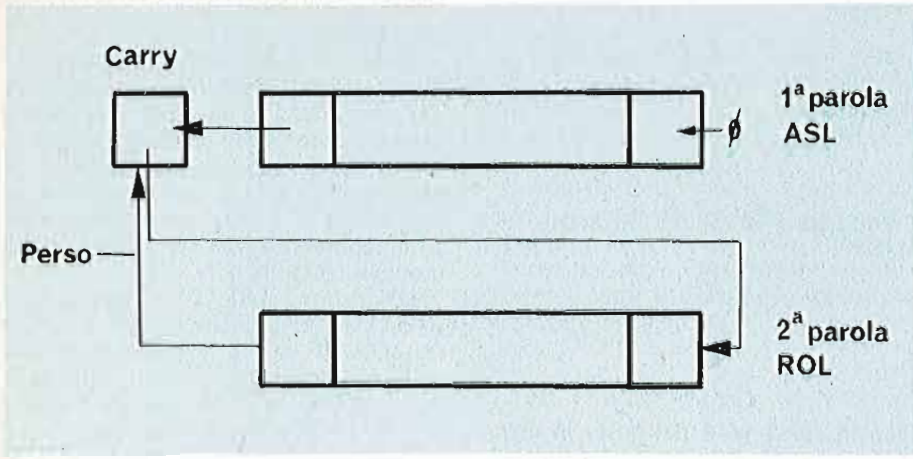


Fig. 9 - Combinazioni dell'uso delle due istruzioni ASL e ROL per riempire in modo automatico due successive locazioni di memoria con il risultato della moltiplicazione che cresce di un bit ad ogni loop. Si noti che il bit che esce a sinistra e va nel Carry (nella istruzione di ROL) viene perso perché subito dopo sostituito con quello proveniente dall'istruzione ASL.

PROGRAMMA DELLA "Corsa dei cavalli"

```
Ø2ØØ  D8 A2 13 BD D9 Ø2 95 7C CA 1Ø F8 A9 89 8D Ø3 FD
Ø21Ø  A2 Ø9 AØ ØØ B9 7C ØØ 84 FC 2Ø 36 FF C8 CØ Ø6 9Ø
Ø22Ø  F3 2Ø EB FE A5 8F 3Ø E3 A2 Ø3 CA 3Ø DE D6 86 DØ
Ø23Ø  F9 86 99 A4 99 B6 83 B9 ED Ø2 35 7C 95 7C E8 96
Ø24Ø  83 B9 ED Ø2 49 FF 15 7C 95 7C EØ Ø5 3Ø 2B DØ Ø6
Ø25Ø  A5 8F FØ 1B DØ 23 A2 Ø2 38 B5 83 E9 Ø6 95 83 CA
Ø26Ø  1Ø F6 A2 Ø6 B5 7C 95 76 A9 8Ø 95 7C CA DØ F5 C6
Ø27Ø  8F DØ Ø6 A5 81 Ø9 Ø6 85 81 B9 89 ØØ FØ ØA 2Ø C5
Ø28Ø  Ø2 29 3C DØ 1A 99 89 ØØ 2Ø C5 Ø2 29 38 85 9A B9
Ø29Ø  8C ØØ 3Ø ØB 29 38 C5 9A BØ Ø5 A9 FF 99 89 ØØ 2Ø
Ø2AØ  EB FE AØ FF A6 99 3D FØ Ø2 FØ Ø1 88 98 55 89 85
Ø2BØ  9A 2Ø C5 Ø2 38 29 Ø1 65 9A 18 A6 99 75 8C 95 8C
Ø2CØ  95 86 4C 2A Ø2 38 A5 92 65 95 65 96 85 91 A2 Ø4
Ø2DØ  B5 91 95 92 CA 1Ø F9 6Ø ØØ 8Ø 8Ø 8Ø 8Ø 8Ø 8Ø 8Ø
Ø2EØ  FF FF FF 8Ø 8Ø 8Ø ØØ ØØ ØØ 8Ø 8Ø 8Ø Ø8 FE BF F7
Ø2FØ  Ø1 Ø2 Ø4
```

SOLUZIONE DEGLI ESERCIZI DELLA PARTE SESTA

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:

```
Ø2ØØ      A9 LDA #ØØ      Azzeramento del risultato
   1      ØØ
   2      D8 CLD
   3      A6 LDX $ØØ      Il moltiplicando va nel registro indice X
   4      ØØ
   5      FØ BEQ Fine
   6      Ø6
   7 Loop 18 CLC
   8      65 ADC $Ø1      Somma il moltiplicatore ad ogni loop
   9      Ø1
   A      CA DEX
   B      DØ BNE Loop     Faccio un numero di loop pari al moltiplicando
   C      FA
   D Fine 85 STA $Ø2      Risultato a posto
   E      Ø2
   F      4C JMP Minitor
Ø21Ø      22
   1      FE
```

accelerare, andateci piano allora.

Il programma viene dato a fine articolo, come al solito nel codice oggetto e parte dalla locazione Ø2ØØ. Si può giocare in tre; la frusta è rappresentata dai tasti:

1 per il cavallo in alto, 2 per quello di mezzo e 3 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 Ø2ØØ.

Questo programma è stato rielaborato per l'AMICO 2000A da uno dei nostri lettori sulla scorta di informazioni tratte da *"The first book of Kim-1"*.

Ringraziamo per questo il Sig. Dell'Orco e speriamo vivamente che non sia il caso isolato perché siamo molto soddisfatti dell'entusiasmo col quale tutti voi ci state seguendo: vuol dire che le nostre fatiche (e sono tante ve lo assicuriamo) sono state fruttifere di soddisfazioni, ne siamo contenti e orgogliosi.

In questo programma si ha:
Il 1° numero da moltiplicare nella locazione ØØØØ.
Il 2° numero da moltiplicare nella locazione ØØØ1.
Il risultato nella locazione ØØØ2.
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:

```
Ø2ØØ      A2 LDX #ØØ
   1      ØØ
   2 Loop 8A TXA
   3      95 STA ØØ,X
   4      ØØ
   5      E8 INX
   6      EØ CPX #$51
   7      51
   8      DØ BNE Loop
   9      F8
   A      4C JMP Monitor
   B      22
C         FE
```

ERRATA CORRIGE - N. 7-8 Sperimentare '79

- **Contrariamente a quanto indicato a pag. 613, 3ª colonna, 2ª riga, la locazione di memoria di partenza del programma 1 è la Ø23Ø.**
- **A pag. 617, 1ª colonna, 12ª riga, è stato indicato erroneamente lo Status con la lettera S, dizione esatta secondo la terminologia americana è P.**
- **Nella Tabella delle istruzioni (pag. 620) del 6502, bisogna correggere il codice operativo delle due istruzioni seguenti:**
  BCS, codice operativo BØ (non 8Ø)
  CLV, codice operativo B8 (non 88)