

Le pagine seguenti seguenti contengono la “tesina” che presentai alla commissione del mio esame di maturità. <https://www.vincenzov.net/tutorial/6502>.

Programmi in assembly per 6502 © 1983-2005 by Vincenzo Villa (<https://www.vincenzov.net>) is licensed under Creative Commons Attribution 4.0 International. To view a copy of this license, visit <https://creativecommons.org/licenses/by/4.0/>

VILLA VINCENZO

INDIRIZZO ELETTRONICO-INFORMATICO

=====

P R O G R A M M I

I N A S S E M B L Y

P E R 6 5 0 2

=====

SU MICROCOMPUTER AIM 65/40

PROGRAMMI ARITMETICI

PROGRAMMI DI CONVERSIONE

PROGRAMMI DI I/O

CONTROLLO DI PROCESSO

SCUOLA MEDIO SUPERIORE SPERIMENTALE DI VILLA GREPPI

ESAMI DI MATURITA' 4 LUGLIO 1983

I N D I C E G E N E R A L E

PREMESSA	pag. 5
PROGRAMMI ARITMETICI	" 7
ARITMETICA BINARIA	" 8
Somma e sottrazione in multipla precisione	9
Moltiplicazione di due bytes	" 12
Moltiplicazione in doppia precisione	" 14
Moltiplicazione in multipla precisione	" 16
Divisione di due bytes	" 18
Divisione con M bytes frazionari	" 20
Divisione M bytes	" 21
Radice quadrata in multipla precisione	" 22
ARITMETICA IN VIRGOLA MOBILE	" 25
Normalizzazione	" 25
Somma (con complemento a 2)	" 27
Moltiplicazione	" 29
ARITMETICA BCD	" 30
Somma M cifre	" 31
Moltiplicazione 8 cifre	" 32
CONVERSIONI	" 34
DA BCD A BINARIO	" 35
DA BINARIO A BCD	" 36
DA ESADECIMALI AD ASCII	" 38
INPUT E OUTPUT	" 40
OUTPUT SU STAMPANTE	" 41
Gestione con I/O ROM	" 41
Gestione senza I/O ROM	" 43
Gestione in modo grafico	" 45

Stampa grafici ad alta risoluzione (250 X 250 punti)	pag. 48
OUTPUT SU DISPLAY	" 50
L'INTEGRATO 6522 VIA (Descrizione generale)	" 52
Esempio di output	" 56
Esempio di input	" 57
INTERRUPT (caratteri generali)	" 59
Orologio	" 61
CONTROLLO DI PROCESSO (Velocità motore)	" 63
RILEVAZIONE DELLA VELOCITA'	" 66
CONTROLLO	" 69
DESCRIZIONE DELL'HARDWARE	" 70
PROGRAMMA DI CONTROLLO	" 73
Parte prima (input dei dati)	" 74
Parte seconda (preparazione al funzionamento)	79
Parte terza (controllo ed output)	" 82
VERSIONE BASIC PIU' ASSEMBLER	" 89
INTERPRETAZIONE DEI DATI	

Queste pagine presentano una serie di programmi in linguaggio macchina per il microprocessore 6502. Si tratta del lavoro svolto durante l'anno scolastico 1982-83 nelle ore di informatica-esercitazione.

Alla base di questi programmi vi é una conoscenza approfondita di tutto il set di istruzioni-macchina del 6502, del linguaggio di programmazione Assembler, dell'Editor, del Debug-monitor, delle principali routines del sistema, dell'uso degli interrupts e delle porte di input/output parallele dell'integrato 6522 VIA, oltre naturalmente alle nozioni di base di programmazione, quali l'uso dei diagrammi di flusso e di alcuni algoritmi fondamentali.

I programmi qui raccolti sono routines di utilità generale di uso frequente nei programmi complessi e vanno da programmi aritmetici (somme, prodotti, divisioni ecc.) a routines di I/O, dalla conversione tra i diversi codici usati dal microcomputer al conteggio di tempi.

Lo sbocco naturale di queste utility può essere considerata la realizzazione pratica del controllo in tempo reale di un fenomeno esterno al microcomputer (in pratica la stabilizzazione della velocità di un motore elettrico in CC), controllo realizzato interamente in linguaggio macchina, per le sue caratteristiche di velocità, e usando anche un programma in linguaggio BASIC.

Questi programmi sono stati realizzati usando il microcomputer AIM 65/40 del ROCKWELL INTERNATIONAL, ma la maggior parte di essi possono funzionare con qualsiasi macchina basata sul 6502; fanno eccezione i programmi di I/O che usano routines del sistema e/o sostegni hardware che non sono presenti su tutti i microcomputer.

La descrizione di ogni programma é realizzata tenendo conto in particolare della velocità di esecuzione, dell'occupazione di memoria e della flessibilità anche se difficilmente tutte queste caratteristiche possono aversi in un solo programma.

Ogni programma viene illustrato dai seguenti punti: a) descrizione degli obiettivi da raggiungere e dell'algoritmo più adatto allo scopo; b) schema blocchi seguito; c) listing completo con notazione simbolica ed esadecimale; d) occupazione di memoria e tempo di esecuzione (clock a 1 MHz); e) eventuali cenni su possibili miglioramenti o variazioni.

La successione dei programmi è divisa in tre sezioni principali:

- Programmi aritmetici su numeri binari interi sia in semplice precisione che in multipla (somma, prodotto, divisione, radice quadrata), su numeri decimali espressi nella notazione BCD, su numeri in virgola mobile.
- Conversioni di codice tra BINARIO - BCD - ASCII.
- Input e Output cioè uso della stampante, del display, della tastiera e della porta parallela di I/O.

BIBLIOGRAFIA ESSENZIALE

Rodnay Zaks - Programmazione del 6502 Ed. Jackson

Andronico...- Manuale di informatica Zanichelli

V.Alessandroni - Elettronica digitale e microprocessori Zanichelli

Rockwell International - AIM 65/40 User's manual

- R6500 Programming manual

- Assembler manual

- Listing I/O ROM

- Listing debug monitor

PROGRAMMI ARITMETICI

In questa sezione saranno esaminati programmi che, usando solo risorse interne del microprocessore 6502, permettono di eseguire somme, sottrazioni, prodotti ecc. di numeri contenuti nella memoria del sistema. Per comprenderli é necessaria una conoscenza almeno superficiale delle tecniche base di programmazione e dei modi in cui un numero può essere rappresentato con un codice binario; questi concetti sono comunque richiamati prima della descrizione dei programmi stessi.

I programmi sono divisi in tre sezioni: nella prima sono esaminati programmi che operano su numeri interi espressi in codice binario naturale; nella seconda sono introdotti programmi che lavorano su numeri con virgola mobile o in forme esponenziali; infine sono illustrati programmi di aritmetica decimale che lavorano su numeri espressi nella notazione BCD.

Lo scopo di questi programmi é innanzitutto preparare delle routines di uso il più generale possibile da usarsi in programmi più complessi, evitando di aggiungere ulteriori problemi di programmazione alle difficoltà già grosse di realizzare algoritmi molto lunghi. Un altro scopo non certo secondario si può trovare nell'acquisizione di una buona capacità di programmazione di un linguaggio per sua natura complesso come l'assembler.

ARITMETICA BINARIA

Somma e sottrazione

Moltiplicazione

Divisione

Radice quadrata

ARITMETICA IN VIRGOLA MOBILE

Somma

Moltiplicazione

ARITMETICA BCD

Somma e moltiplicazione

A R I T M E T I C A B I N A R I A

Il modo più semplice e compatto generalmente usato per rappresentare un numero in un sistema binario é il codice binario naturale.

Con esso sono rappresentabili in un byte (= 8 cifre binarie, la lunghezza di una parola del 6502 e della quasi totalità dei microprocessori) numeri compresi tra 0 e 255 (vedi tabella I);

NUMERO DECIMALE	BINARIO NATURALE	ESADECIMALE	COMPL. A 2
- 128	=	=	I 0000000
- 127	=	=	I 000000I
....	=	=	
+ I	=	=	I IIIIIII
0	00000000	00	0 0000000
I	0000000I	0I	0 000000I
2	000000IO	02	0 00000IO
....		...	
I27	0IIIIIII	7F	0 IIIIIII
...		...	=
254	IIIIIIIO	FE	=
255	IIIIIIII	FF	=

TAB. 1

generalmente sono però necessari numeri maggiori di 255: per questo sono usati numeri in doppia precisione ottenuti con 2 byte successivi (numeri fino a 65.536) o in multipla precisione usando N byte (numeri fino a $2^n - 1$, con n = numero di bit).

Per rendere più leggibili i numeri binari é spesso usata la notazione esadecimale, in cui ogni gruppo di 4 bit é sostituito da una cifra esadecimale (I, 2 ... 9, A, B, C, D, E, F) come mostrato dalla tabella. Per distinguere i numeri binari ed esadecimali da quelli decimali, sono preceduti dai prefissi % e \$.

Nella rappresentazione di numeri negativi, dato che non esistono i simboli + e -, per indicare il segno può essere usato il bit più a sinistra (I per il negativo, 0 per il +); il numero vero e proprio é costituito dagli altri 7 bit: se il segno é positivo il numero é rappresentato nella sua forma normale; se é negativo col suo complemento

a 2.

Questo modo di rappresentare i numeri negativi può sembrare complesso ma si rivela utile al momento di usarli in somme e sottrazioni.

Dato che il numero rappresentabile con un byte é piuttosto piccolo (da - 128 a + 127) si usa anche qui la notazione in multipla precisione.

Per maggiori dettagli si può consultare il cap. I del libro "Programmazione del 6502".

SOMMA E SOTTRAZIONE

Nel microprocessore 6502 la somma é realizzata con l'istruzione ADC che somma il contenuto dell'accumulatore, l'operando specificato e il carry (cioé un bit che contiene l'eventuale riporto di una somma precedente). La somma sarà poi contenuta nell'accumulatore e l'eventuale riporto nel carry.

Il programma (progr.1) é formato da 3 istruzioni: la prima carica nell'accumulatore il primo addendo, la seconda realizza la somma con il secondo addendo (che può essere tanto un numero vero e proprio quanto l'indirizzo della locazione di memoria in cui tale numero si trova), la terza immagazzina l'accumulatore in una locazione specificata.

Inoltre, dato che prima di sommare i due numeri non si conosce il contenuto del carry é necessario azzerarlo con l'istruzione CLC.

Generalmente é necessario sommare numeri di più byte tra di loro: per farciò si usa lo stesso metodo usato nelle somme fatte manualmente. Si sommano le due cifre meno significative e si memorizzano il riporto e la somma; si passa poi alle cifre successive che si sommano tra loro e con il riporto precedente generando un eventuale riporto e così per tutte le cifre fino all'ultima.

Questa operazione é facilitata dal fatto che nei microprocessori la somma e la generazione del riporto sono automatiche e contemporanee.

Lo schema blocchi che ne consegue é quello rappresentato in fig. I,

Istruzioni	Commento
CLC	Cancella il CARRY
LDA OP1	Carica il primo addendo
ADC OP2	Somma il secondo addendo
STA SOM	Memorizza il risultato
BRK	Stop

PROG. 1

considerando che si sommano due numeri A e B di n byte ciascuno generando una somma S. Da notare che in un programma reale le lettere A, B ed S sono sostituite da un indirizzo fisso che indica dove è memorizzato il numero: così A_0 rappresenta l'indirizzo del byte meno significativo di A, A_1 il successivo e così via fino ad A_{n-1} che rappresenta l'ultimo byte.

Nel caso che la somma delle ultime cifre dia un riporto è necessario segnalarlo all'operatore. Si tratta di uno schema molto semplice che però presenta due grossi inconvenienti:

- 1) dovendo sommare numeri formati da molti byte il programma risulterà particolarmente lungo (la somma di due byte richiede come minimo tre istruzioni)
- 2) il numero di byte deve essere fissato unavolta per tutte ed una variazione di questo provoca necessariamente il cambiamento del programma.

Questi difetti possono essere risolti usando un algoritmo basato sull'indirizzamento indicizzato, un'indirizzamento in cui l'indirizzo reale di ogni singolo

byte di A, B ed S è ottenuto dalla somma automatica di un indirizzo base e di un indice variabile (nel 6502 i registri X o Y). Questo indice è

incrementato alla somma di ogni singolo byte, fino ad arrivare ad n: a questo punto il programma si ferma. Lo schema blocchi è riportato in fig.2. Esso presenta il difetto di essere più lento del precedente, non solo perché l'indirizzamento indicizzato è intrinsecamente più lento di quello diretto ma specialmente per la necessità di incrementare il registro indice e di confrontar-

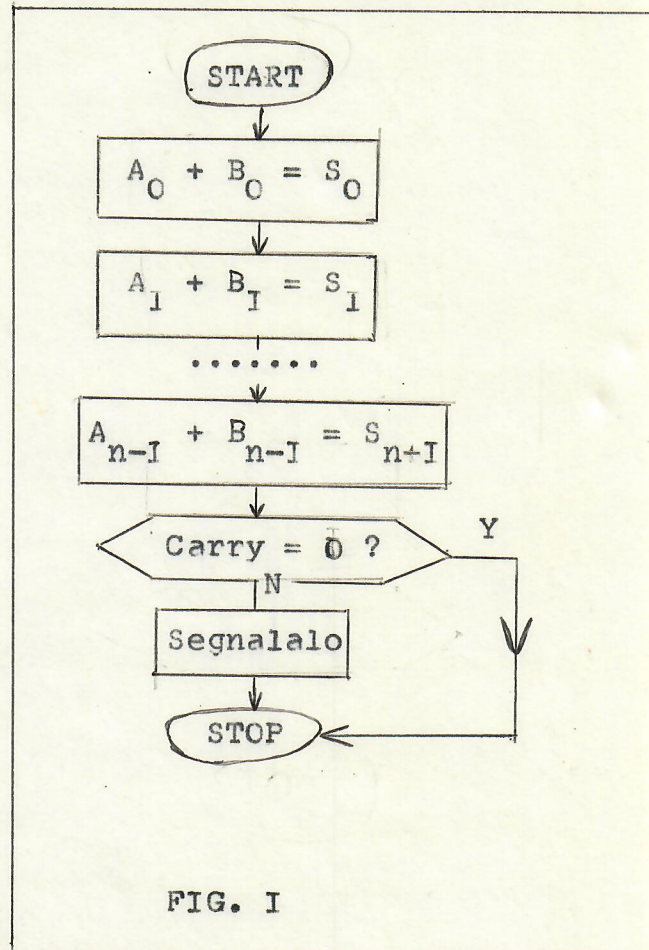


FIG. I

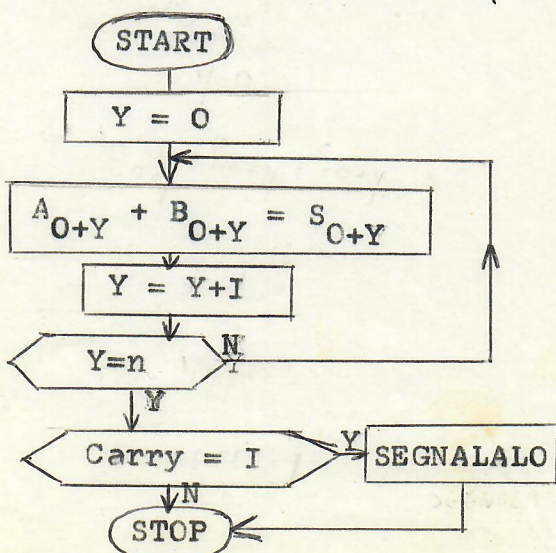


FIG 2

lo con n; il programmatore deve quindi saper scegliere volta per volta l'algoritmo più adatto al suo scopo, scegliendo tra velocità e flessibilità.

Un'altra modifica a questo programma può essere nell'uso dell'indiriz-

zamento indiretto indicizzato in cui cioè l'indirizzo non è definito da programma ma è contenuto in due locazioni di memoria consecutive, definite dal programmatore volta per volta.

Questo modo di indirizzare, a costo di una ulteriore piccola diminuzione di velocità, dà la possibilità di operare su numeri posti ad un indirizzo qualsiasi senza dover modificare il programma. Il listato 2 presenta un programma basato sull'algoritmo di fig.2 con l'uso dell'indirizzamento indiretto indicizzato. Gli indirizzi di A, B ed S sono posti rispettivamente nelle locazioni ADDRA, ADDRB e ADDR. Il numero di byte che formano i numeri deve essere

```

PAGE 0001
ADDR OBJECT    SOURCE
Somma multipla precisione
#1=Indirizzo di A
#3=Indirizzo di B
#3=Indirizzo della somma
#0=Numero di byte
        ADDRA=#1
        ADDRB=#3
        ADDR=#5
        PUTDIS=#F7D3
        *=$750
0750 4F 56      .BYT 'OVERFLOW'
        *=$800
0800 A4 00      LDY 0
0802 10         CLC
0803 B1 01      SU    LDA (ADDRA),Y
0805 71 03      ADC (ADDRB),Y
0807 91 05      STA (ADDR),Y
0809 00         DEY
080A 10 F7      BPL SU
080C A2 00      LDX #0
080E 50 00      BVC END
0810 BD 50 07   RISV LDA $750,X
0812 20 D3 F7   JSR PUTDIS
0816 E8         INX
0817 E0 00      CPX #0
0819 D0 F5      BNE RISV
081B 00         BRK
        END
Vincenzo Villa    V D    14/2/1983
        END
ERRORS=0000

```

PROG. 2

posto nella locazione numero 0. La segnalazione di un riporto all'ultima cifra è segnalata dalla scritta " OVERFLOW " sul display, secondo le modalità descritte in seguito, nella sezione di I/O.

In figura 3a è riportato uno schema che, con riferimento alla fig.2, indica il numero di cicli-macchina richiesti dal programma; esso è ottenuto sommando il numero di cicli richiesti dalle singole istruzioni che formano i vari blocchi. Il tempo di esecuzione, che dipende dal numero n di byte, è fornito dalla relazione $T = I_3 + 20n$, con T espresso in μ secondi; per confronto è riportato anche il diagramma dell'algoritmo di fig I ($T=I_1 + I_2n$).

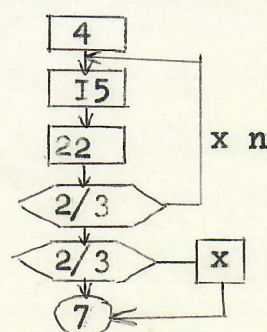


FIG 3a

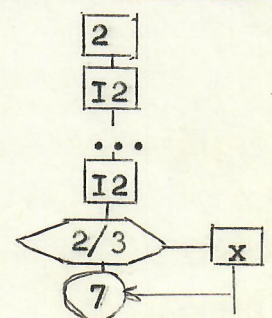


FIG 3b

Con $n = 10$ (numeri fino a circa $1,2 \times 10^{25}$) il tempo di esecuzione é di circa 0,211 mS per il diagramma a e 0,129 mS per il diagramma b; da questo calcolo é stato escluso il tempo usato per segnalare l'overflow, date le caratteristiche diverse dei programmi di I/O.

Il prog. 2 occupa sedici bytes di memoria, oltre naturalmente a quelli occupati da A,B ed S. Sono inoltre da aggiungere i sei bytes usati per l'indirizzamento .

Un programma del tipo di quello in fig. 1 occupa $9n$ più 2 bytes (con n = numero di bytes da sommare). Il programma di fig. 1 può quindi essere considerato conveniente fino alla somma di due o tre bytes.

Questo programma può funzionare tanto su numeri positivi che su numeri negativi espressi nella notazione del complemento a due, date le caratteristiche dell'aritmetica binaria.

Sostituendo l'istruzione ADC con SBC (sottrazione dall'accumulatore) il programma realizza una sottrazione in multipla precisione tanto su numeri positivi che negativi.

Dovendo lavorare su numeri con la virgola é possibile, usando questo programma, dedicare un certo numero di bytes alla parte intera ed un certo numero alla parte frazionaria.

MOLTIPLICAZIONE

Il 6502 non può eseguire direttamente il prodotto di due numeri, come del resto tutti i microprocessori oggi in commercio (escludendo il TMS 9980); esso deve quindi essere eseguito da un apposito programma.

Il primo algoritmo a cui si può pensare é quello per somme successive (fig. 4 $A \times B = P$) in cui cioè il moltiplicando é sommato a se stesso tante volte quanto é il moltiplicatore. Si tratta di un algoritmo poco efficiente perché, se é vero che per $A = 0$, il programma é subito bloccato, per $A = 255$ é necessaria la somma di B a se stesso per 255 volte,

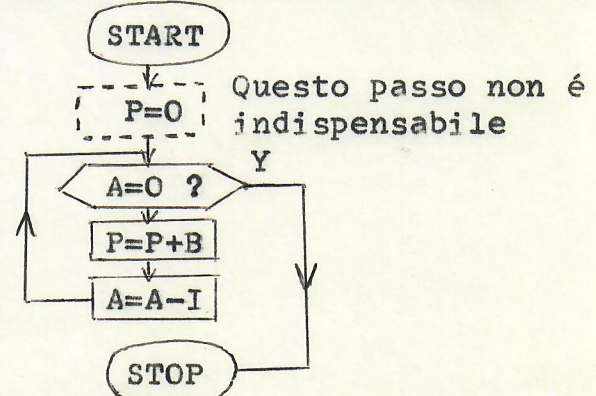


FIG. 4

somma che deve essere eseguita in doppia precisione dato che $255 \text{ per } 255 = 65025$, numero contenuto in due bytes. Il tempo di esecuzione può così variare da un minimo di tre microsecondi ad un massimo di 7,7 mS. Si deve inoltre notare che il tempo richiesto aumenta esponenzialmente con l'aumentare del numero di bytes (ad es. 2 bytes per due bytes richiedono fino a circa 2,5 S).

Il problema è risolto ricorrendo all'algoritmo della moltiplicazione

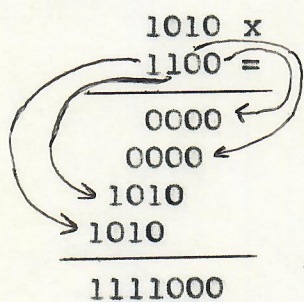


FIG. 5a

manuale opportunamente adattato al sistema binario. Si può cioè moltiplicare la prima cifra del moltiplicatore per il moltiplicando e quindi sommare al prodotto, inizialmente uguale a 0; quindi si sposta a sinistra il moltiplicando di una cifra e lo si moltiplica per la seconda cifra del moltiplicatore e si somma al prodotto; lo stesso ragionamento lo si deve fare per tutte le cifre.

Questo algoritmo è di facile applicazione su numeri binari dato che una singola cifra può essere solo 0 o 1; il moltiplicando può quindi essere semplicemente sommato al prodotto (nel caso che la cifra del moltiplicatore è 1) o non sommato (se la cifra del moltiplicatore è 0).

Un esempio (fig. 5a) può spiegare bene questa tecnica: si tratta del prodotto di due nibbles, che richiede due somme (al massimo possono essere 4) e 3

A=0000 1010
0 non somma
A=0001 0100
0 non somma
A=0010 1000
1 somma
P=0010 1000
A=0101 0000
1 somma
P=0111 1000

FIG. 5b: esempio di funzionamento

spostamenti verso sinistra del moltiplicando.

L'algoritmo che ne risulta è quello in fig. 6 da cui deriva il programma 3 che realizza il prodotto di due fattori posti nelle locazioni 0 e 2; il prodotto, formato da due bytes sarà posto nelle locazioni 3 e 4; da notare infine che il byte 1 deve essere azzerato perché in esso sarà spostato il moltiplicando.

Il tempo di esecuzione è raffigurato in fig. 7, con riferimento alla fig. 6: da esso si trae che il tempo totale va da un minimo di 0,26 mS (con il

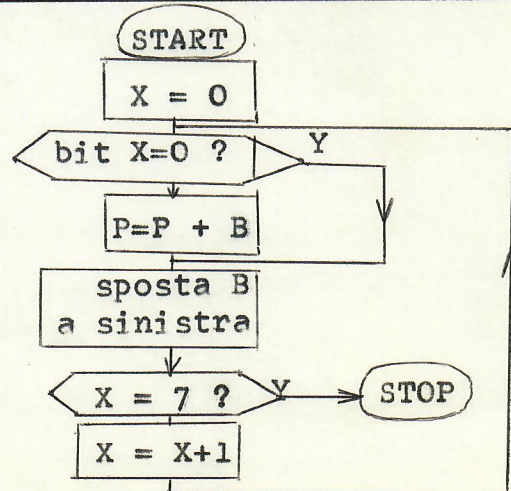


FIG. 6

moltiplicatore uguale 0) fino ad un massimo di 0,41 mS se il moltiplicatore è uguale a 255. Questa variabilità dipende dal numero di cifre uguali a 0 del moltiplicatore dato che da esso dipende il numero di volte per cui il moltiplicando è sommato.

Il tempo medio è così nettamente inferiore a quello di un programma legato allo schema di fig. 4 (0,33 contro 3,8 mS).

Il programma occupa 34 Bytes di programma.

Come già detto a proposito della somma è spesso necessario moltiplicare tra loro numeri più grossi di 255: se ad es. si devono moltiplicare tra loro numeri di 2 bytes (il prodotto sarà di 4 bytes) è necessario modificare il precedente programma nei seguenti punti: a) ricercare i bites I in due bytes anziché in uno; b) fare una somma in quadrupla precisione anziché in doppia; c) spostare

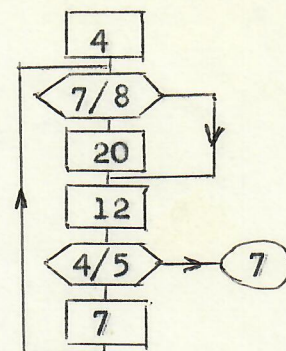


FIG. 7

il moltiplicando per 15 volte verso destra.

Il programma che ne deriva è il progr.4 ; esso ha un tempo variabile fra I e I,7 mS (mediamente 1000 volte più veloce di un algoritmo basato su somme successive). Dall'esame del programma 4 appaiono alcuni difetti che ne rendono difficile l'applicazione come routine di uso generale o che sono poco razionale dal punto di vista dell'occupazione di memoria; in particolare: I) il programma tra le labels SUI e DUE è formalmente uguale alla parte compresa fra SU2 ed END? cambiando solo gli indirizzi su cui le istruzioni agiscono.

PAGE 0001

ADDR OBJECT SOURCE

;Moltiplicazione di 2 numeri di 8 bit
;(risultato di 16 bit)
;Fattori \$0 e \$2 Prodotto \$3 e \$4
;(\$1,\$3 e \$4 devono essere azzerati)

```

0000 D8          CLD
0001 A2 01       LDX ##01
0003 8A          TXA
0004 25 00       AND $00
0006 F0 0D       BEQ ROL1
0008 18          CLC
0009 A5 02       LDA $02
000B 65 04       ADC $04
000D 85 04       STA $04
000F A5 01       LDA $01
0011 65 03       ADC $03
0013 85 03       STA $03
0015 18          CLC
0016 26 02       ROL $02
0018 26 01       ROL $01
001A 8A          TXA
001B 30 05       BMI END
001D 0A          ASL A
001E AA          TAX
001F 4C 03 00    JMP SU
0022 00          END
          BRK
          END
  
```

ERRORS=0000

Vincenzo Villa

V D

3/2/1983

PROG. 3

Si tratta quindi di un doppione eliminabile con l'uso di indirizzamento

indicizzato; ciò permette anche di ampliare a piacere il numero di bytes da moltiplicare. 2) Le souboutines ADD (che realizza la somma) e ROLS (che sposta il moltiplicando a sinistra) possono essere sostituite con routines con indirizzamento indicizzato sul tipo di prog.2, con i vantaggi tipici di questo tipo di programma. 3) E' necessario che subito davanti al moltiplicando vi siano due bytes azzerati per spostarci il moltiplicando durante la routine ROLS. 4) Moltiplicatore, moltiplicando e risultato devono essere in locazioni ben determinate ed invariabili della memoria: questo problema è superato con l'uso dell'indirizzamento indiretto indicizzato. 5) Il moltiplicando è trasferito di due bytes sulla sinistra. Il fatto che il prodotto sia sommato a ciò che era contenuto prima in quelle locazioni non può essere considerato un difetto del programma, in quanto permette di risolvere facilmente espressioni del tipo $P_t = AB + CD + EF + \dots$. Se il programmatore vuole, deve azzerare queste locazioni con un programma a parte. Questi problemi sono stati risolti con il programma 5. In esso sono state usate le seguenti variabili: ADDRA contiene l'indirizzo di un fattore (16 Bits); ADDRB l'indirizzo dell'altro fattore; ADDRH e

```

PAGE 0001
ADDR OBJECT SOURCE
; Moltiplicazione di 2 numeri di 16 bit
; (Risultato 32 bit)
; Fattori $0$1 e $4$5
; $2 $3 $6 $7 $8 e $9 = 0
; Risultato $6 $7 $8 e $9
0000 D8 CLD
0001 A2 01 LDX #1
0003 8A SU1 TXA
0004 25 01 AND 1
0006 F0 03 BEQ ROL1
0008 20 2C 08 JSR ADD
000B 20 46 08 ROL1 JSR ROLS
000E 8A TXA
000F 30 05 BMI DUE
0011 0A ASL A
0012 AA TAX
0013 4C 03 08 JMP SU1
0016 A2 01 DUE LDX #1
0018 8A SU2 TXA
0019 25 00 AND 0
001B F0 03 BEQ ROL2
001D 20 2C 08 JSR ADD
0020 20 46 08 ROL2 JSR ROLS
0023 8A TXA
0024 30 05 BMI END
0026 0A ASL A
0027 AA TAX
0028 4C 18 08 JMP SU2
002B 00 END BRK
002C 18 ADD CLC
002D A5 05 LDA 5
002F 65 09 ADC 9
0031 85 09 STA 9
0033 A5 04 LDA 4
0035 65 08 ADC 8
0037 85 08 STA 8
0039 A5 03 LDA 3
003B 65 07 ADC 7
003D 85 07 STA 7
003F A5 02 LDA 2
0041 65 06 ADC 6
0043 85 06 STA 6
0045 60 RTS
0046 18 ROLS CLC
0047 26 05 ROL 5
0049 26 04 ROL 4
004B 26 03 ROL 3
004D 26 02 ROL 2
004F 60 RTS
; Vincenzo Villa V D
END
ERRORS=0000

```

21/2/1983

PROG. 4

ADDRL contengono l'indirizzo del prodotto: esso é diviso in due parti ciascuna delle quali é formata dal numero di bytes che formano moltiplicando e moltiplicatore (il risultato é quindi composto dal doppio di queste cifre); ADDR5 é l'indirizzo di una serie di bytes liberi usati per spostare il moltiplicando, evitando così di dover porre n bytes azzerati davanti al moltiplicando; infine L contiene la lunghezza espressa in numero di bytes di tutti i numeri su cui il programma agisce.

Il programma é sostanzialmente legato allo schema a blocchi di Fig. 6, tenenda presente che ogni blocco deve essere eseguito in multipla precisione, secondo quanto determinato da L.

Esaminando il programma si possono riconoscere i vari blocchi segnati da una label:

Con SU0 il programma cancella le locazioni che riceveranno parte del moltiplicando quando questo é spostato verso sinistra. Questa locazione non deve essere necessariamente a fianco del moltiplicando ma può essere definita in qualunque posizione di memoria. Le quattro istruzioni dopo la label MOLT servono per spostare un byte di B in una locazione nella quale sarà esaminata per ricercare i bit 1

PAGE 0001

ADDR OBJECT SOURCE

Moltiplicazione M byte
 #0 e #1 = Indirizzo di A
 #2 e #3 = Indirizzo di B
 #4 e #5 = Indirizzo del prodotto basso
 #6 e #7 = Indirizzo del prodotto alto
 #8 e #9 = Indirizzo di memoria libera
 #A = Lunghezza di A e B (in byte)

ADDRA=#0
 ADDR8=#2
 ADDRRL=#4
 ADDRRLH=#6
 ADDR5=#8
 L=#A
 PB=#B
 CONT=#C

0000	C6	0A		DEC	L
0001	A5	0A		LDA	L
0004	85	0C		STA	CONT
0006	A4	0A		LDY	L
0008	A9	00		LDA	#0
000A	91	00	SU0	STA	(ADDR5),Y
000C	88			DEY	
000D	10	FB		BPL	SU0
000F	A4	0C	MOLT	LDY	CONT
0011	B1	02		LDA	(ADDR8),Y
0013	85	0B		STA	PB
0015	C6	0C		DEC	CONT
0017	A9	01	SEMM	LDA	#1
0019	AA		INI	TAX	
001A	25	0B		AND	PB
001C	F0	17		BEQ	ROLS
001E	A4	0A		LDY	L
0020	18			CLC	
0021	B1	00	SU1	LDA	(ADDRA),Y
0023	71	04		ADC	(ADDRRL),Y
0025	91	04		STA	(ADDRRL),Y
0027	88			DEY	
0028	10	F7		BPL	SU1
002A	A4	0A		LDY	L
002C	B1	00	SU2	LDA	(ADDR5),Y
002E	71	06		ADC	(ADDRRLH),Y
0030	91	06		STA	(ADDRRLH),Y
0032	88			DEY	
0033	10	F7		BPL	SU2
0035	A4	0A	ROLS	LDY	L
0037	18			CLC	
0038	B1	00	SU3	LDA	(ADDRA),Y
003A	2A			ROL	A
003B	91	00		STA	(ADDRA),Y
003D	88			DEY	
003E	10	F8		BPL	SU3
0040	A4	0A		LDY	L
0042	B1	00	SU4	LDA	(ADDR5),Y
0044	2A			ROL	A
0045	91	00		STA	(ADDR5),Y
0047	88			DEY	
0048	10	F8		BPL	SU4
004A	8A			TXA	
004B	8A			ASL	A
004D	D0	CB		BNE	INI

Con la label SEMM inizia il programma vero e proprio, con la ricerca dei bit 1 e 0 e la decisione se sommare o no il moltiplicando.

La somma vera e propria è realizzata con le 12 istruzioni seguenti (segnate dalle labels SU1 e SU2), che sommano prima gli L bytes meno significativi, poi gli L bytes più significativi.

Lo spostamento del moltiplicando verso sinistra è realizzato da ROLS, anche in questo caso spostando prima gli L bytes meno significativi, poi i più significativi, cioè quelli contenuti nella locazione ADDR5. Infine (con SU5) il programma rimette il moltiplicando nelle locazioni originarie, dopo che è stato spostato durante il blocco ROLS.

Il tempo di esecuzione è dato dall'espressione:

$$T = ((17 \times 2n + [22 \times 2n] + 18) \times 8 + 7)n$$
 con n=numero di bytes da moltiplicare (FIG. 9)

Il termine racchiuso in un rettangolo tratteggiato può essere eseguito o no, a seconda dei valori dei singoli bit del moltiplicatore.

Per confronto, con n=2 il tempo di esecuzione varia tra 1.4 e 2.8 mS, contro gli 1.1 - 1.7 del prog. 4.

Questo programma, sebbene realizzato per numeri interi, può ~~realizzar~~ eseguire anche il prodotto di numeri con virgola fissa, usando lo stesso stratagemma usato nelle operazioni manuali. In questo caso si rivela utile il fatto che il risultato è diviso in due parti: infatti se moltiplicando e moltiplicatore sono formati dallo stesso numero di cifre intere e frazionarie, la parte intera del prodotto si troverà ad ADDRH mentre la parte frazionaria ad ADDRL.

```

004E A5 0C      LDA CONT
0050 10 BD      BPL MOLT
0052 A4 0A      LDY L
0054 B1 00      LDA (ADDR5),Y
0056 91 00      STA (ADDR4),Y
0058 00         DEY
0059 10 F9      BPL SU5
005B E6 0A      INC L
005D 00         BRK
;Vincenzo Villa      V D      2/3/1983
                        END

ERRORS=0000
PROG. 5 - FINE
    
```

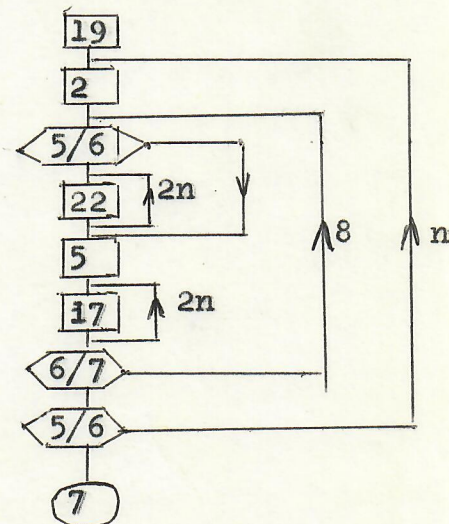


FIG. 9 Cicli macchina

DIVISIONE

Dato che il microprocessore 6502 non esegue direttamente, ^{il rapporto} è necessario realizzare un apposito programma; il primo algoritmo a cui si può pensare

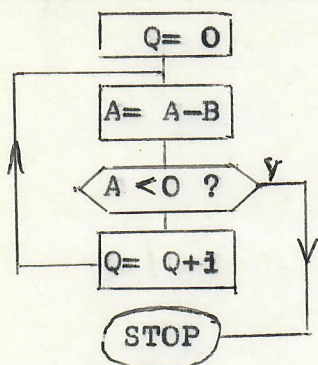


FIG. 10

è quello basato su una serie di sottrazioni successive (fig. 10, $A : B = Q$) ma, facendo un discorso analogo a quello già fatto per il prodotto, si nota che si tratta di un diagramma molto lento.

Un algoritmo migliore è raffigurato in fig. 11, che rispecchia quello della divisione manuale binaria. In pratica si separa una cifra del

dividendo con uno spostamento di un bit sulla sinistra, trasferendolo in una locazione consecutiva; si vede quindi se questa parte separata è maggiore o minore del divisore: nel primo caso si pone il primo bit del quoziente a 1 e si sottrae dalla parte separata il divisore; si separa quindi un altro bit e si prosegue. Un esempio è riportato in fig. 12, con riferimento al rapporto di due nibbles. Il programma (progr. 6) è riportato a fianco.

Il tempo di esecuzione è compreso fra 0,37 e 0,27 ^{ms} come si può capire dalla fig. 13

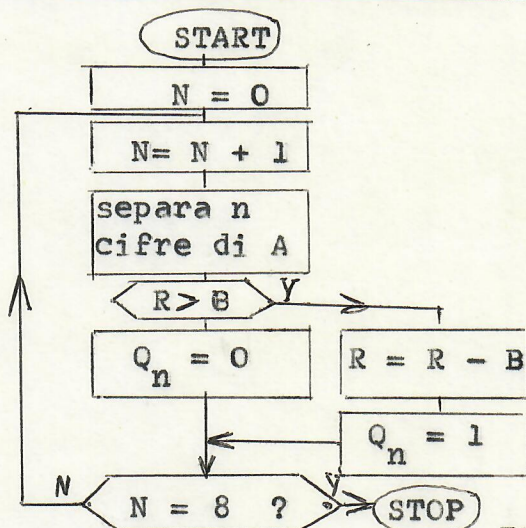


FIG. 11

```

1110 : 0011 = 0100
 1
11-
11=
001
 10 ← resto
  
```

FIG. 12

che, come al solito si riferisce allo schema blocchi.

La variabilità nel tempo di esecuzione è dovuta alla possibilità di sottrarre o no dalla parte separata il divisore.

Da notare che, finito il programma (che

naturalmente dà un quoziente intero), un eventuale resto sarà contenuto nella locazione in cui è stato fatto scorrere il dividendo. Osservando il progr. 6 si deduce che il divisore è da porsi nella locazione N.0, il dividendo nella locazione 2, il quoziente si troverà nella locazione 3. Da notare infine che la locazione in cui sarà fatto scorrere il dividendo deve essere

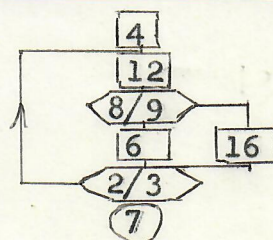


FIG. 13

azzerata prima di richiamare il programma.

Per porre un singolo bit del quoziente a 0 o a 1 si possono usare due metodi: a) si può fare un OR tra il quoziente e l'accumulatore (con l'istruzione ORA)/ l'accumulatore contiene un solo bit a 1 e quindi dopo l'OR questo bit sarà forzato a 1. E' quindi sufficiente far scorrere l'accumulatore verso destra per porre i bits del quoziente a 1. Questo metodo richiede però che il quoziente sia preventi-

vamente azzerato (fig. I4a).
b) si può spostare il quoziente verso sinistra, facendo entrare nella prima posizione (proveniente dal carry) un bit 1 o 0 (fig. I4b).

si tratta di due metodi praticamente equivalenti: ad es. il primo é usato nel progr. 6, mentre il secondo nel progr. 9.

BYTE	CARRY	BYTE	MASCHERA
xxxx	1	0000	1000
xxx1	0	1000 ^{ORA}	0100
xx10	1	1000	0010
x101	0	1010 ^{ORA}	0001
1010		1010	

FIG.14 - Forzare i bit 1 e 3 a 1

PAGE 0001

ADDR OBJECT SOURCE

```

; Divisione tra due numeri di 8 bit
; Divisore $0      Dividendo $2
; $1=$0           Resto $1      Quoziente $3
0000 D8           CLD
0001 A2 80        LDX #80
0003 18           SU          CLC
0004 26 02        ROL 2
0006 26 01        ROL 1
0008 B0 06        BCS SOT
000A A5 01        LDA 1
000C 05 00        CMP 0
000E 90 09        BCC SALT
0010 E5 00        SOT        SBC 0
0012 85 01        STA 1
0014 8A           TXA
0015 05 03        ORA 3
0017 85 03        STA 3
0019 8A           TXA
001A 4A           LSR A
001B F0 04        BEQ END
001D AA           TAX
001E 4C 03 08     JMP SU
0021 00           END        BRK
; Vincenzo Villa   V D      7/2/1983
                        END

```

ERRORS=0000

Il programma appena visto naturalmente non tiene conto di eventuali parti frazionarie del quoziente: questo difetto é superato con il programma stesso, che non fa altro che ripetere lo stesso procedimento prima sul dividendo (ottenendo la parte intera) poi sul resto.

Il programmatore può scegliere quanti bytes dopo la virgola desidera attraverso una variabile posta nella locazione n° 4; il programma comunque si ferma automaticamente quando il resto é 0.

Il tempo di esecuzione dipende principalmente dal numero di bytes dopo la virgola ed é di circa 0,32 mS per ognuno di essi.

I problemi legati alla realizzazione di divisioni tra numeri più grandi di 255, sono risolti dal programma 8 che, basandosi su un indirizzamento indiretto indicizzato, permette di realizzare il rapporto tra due numeri di lunghezza qualsiasi definita dal programmatore.

Lo schema a blocchi seguito é quello già visto per la divisione di due bytes, naturalmente adattato a numeri più grossi. Anche in questo caso come già fatto a proposito della moltiplicazione in multipla precisione, le locazioni in cui é fatto scorrere il dividendo non devono essere necessariamente ad esso adiacente e sono automaticamente azzerate dal programma.

```

PAGE 0001
ADDR OBJECT SOURCE
;Divisione 8 bit con M cifre decimali
;Divisore $0 Dividendo $2
; $1=##0 Resto $1
;Quoziente = $5,$6,... (M byte)
; $4 = Numero byte del quoziente (=M)
0000 D8 CLD
0001 A0 00 LDY ##0
0003 A5 04 INIZIO LDA $4
0005 F0 12 BEQ FINE
0007 C6 04 DEC $4
0009 A9 00 LDA ##0
000B 85 03 STA 3
000D 28 1A 08 JSR DIV
0010 A5 03 LDA 3
0012 99 05 00 STA 5,Y
0015 C8 INY
0016 4C 03 08 JMP INIZIO
0019 00 FINE BRK
;
001A A2 00 DIV LDX ##80
001C 18 SU CLC
001D 26 02 ROL 2
001F 26 01 ROL 1
0021 B0 06 BCS SOT
0023 A5 01 LDA 1
0025 C5 00 CMP 0
0027 90 09 BCC SALT
0029 E5 00 SOT SBC 0
002B 85 01 STA 1
002D 8A TXA
002E 05 03 ORA 3
0030 85 03 STA 3
0032 8A SALT TXA
0033 4A LSR A
0034 F0 04 BEQ END
0036 AA TAX
0037 4C 1C 08 JMP SU
003A 60 END RTS
;Vincenzo Villa V D 11/2/1983
END
ERRORS=0000

```


ADDR OBJECT SOURCE

Divisione M byte

```

;#0 e #1=Indirizzo del dividendo
;#2 e #3=Indirizzo del divisore
;#4 e #5=Indirizzo del quoziente
;#6 e #7=Indirizzo del resto
;#8=Numero byte
;#9=##00 (contatore)

```

```

ADDRA=#0
ADDRB=#2
ADDRQ=#4
ADDRR=#6
L=#8
CONT=#9

```

```

0000 A9 00      DIVISO LDA ##0
0002 85 09      STA CONT
0004 C6 08      INI    DEC L
0006 A4 08      LDY L
0008 18         CLC
0009 B1 00      SU1    LDA (ADDRA),Y
000B 2A         ROL A
000C 91 00      STA (ADDRA),Y
000E 88         DEY
000F 10 F8      BPL SU1
0011 A4 08      LDY L
0013 B1 06      SU2    LDA (ADDRR),Y
0015 2A         ROL A
0016 91 06      STA (ADDRR),Y
0018 88         DEY
0019 10 F8      BPL SU2
001B B0 0F      BCS DIVER
001D A0 FF      LDY ##FF
001F C8         SU3    INY
0020 B1 06      LDA (ADDRR),Y
0022 D1 02      CMP (ADDRB),Y
0024 D0 06      BNE DIVER
0026 C4 08      CPY L
0028 D0 F5      BNE SU3
002A F0 02      BEQ UGUALI
002C 90 0C      DIVER  BCC SALTO
002E A4 08      UGUALI LDY L
0030 B1 06      SU4    LDA (ADDRR),Y
0032 F1 02      SBC (ADDRB),Y
0034 91 06      STA (ADDRR),Y
0036 88         DEY
0037 10 F7      BPL SU4
0039 38         SEC
003A A4 08      SALTO  LDY L
003C B1 04      SU5    LDA (ADDRQ),Y
003E 2A         ROL A
003F 91 04      STA (ADDRQ),Y
0041 88         DEY
0042 10 F8      BPL SU5
0044 E6 08      INC L
0046 E6 09      INC CONT
0048 A5 09      LDA CONT
004A 4A         LSR A
004B 4A         LSR A
004C 4A         LSR A
004D C5 08      CMP L
004F D0 B3      BNE INI
0051 00         BRK   (D RTS)

```

Vincenzo Villa V D 21/3/1983

.END

ERRORS=0000

Dando una scorsa veloce, al programma si nota che il loop SUI continuato poi da SU2 realizza lo spostamento a sinistra e la divisione del dividendo; il loop segnato da SU3 confronta la parte separata del dividendo con il divisore e decide se si deve sottrarre il divisore (attraverso il loop SU 4) e il valore del bit del quoziente inserito con il loop SU5; infine si controlla se il programma ha finito di separare tutte le cifre del dividendo.

Il tempo di esecuzione è espresso dalla formula $T = (37,5n + 37) \cdot 8n$; ad es. , con $n = 2$ il tempo di esecuzione è di circa 2mS.

Il programma occupa 84 BYTES.

Questo programma, sebbene realizzato per numeri interi, può funzionare anche su numeri composti da un certo numero fisso di decimali usando le stesse tecniche usate nella divisione decimale. Ad es. per realizzare il rapporto con un decimale 15:4, si può realizzare il rapporto fra interi $150: 4 = 37...$ tenendo conto che l'ultima cifra è in realtà la prima decimale (quindi il risultato reale è 3,7).

Può essere infine utile un suggerimento per realizzare in modo molto veloce e compatto la divisione o il prodotto di un numero per 2, 4, 16 .. (cioè per la base e i suoi multipli): in pratica si usa lo stesso metodo che, nella Base 10 si usa per moltiplicare o dividere per 10, 100, 1000 ... In pratica dovendo moltiplicare per 2 o per un suo multiplo si aggiungono uno o più zeri alla destra del numero, spostandolo verso sinistra con l'istruzione ASL. Analogamente per dividere si "sposta la virgola" di una o più posizioni verso sinistra con l'istruzione LSR. In questo modo è possibile realizzare un prodotto o una divisione in 6 μ S per bytes.

RADICE QUADRATA

Un altro programma che può spesso essere usato è quello che calcola la radice quadrata di un numero con un certo numero di cifre dopo la virgola. Si tratta di una funzione piuttosto lunga e complessa da realizzare con il linguaggio Assembler e usando l'algoritmo diretto. Un metodo più semplice da realizzare avendo a disposizione un programma che esegue la divisione è il seguente: si divide il radicando per un numero qualsiasi positivo diverso da 0; quindi si fa la media aritmetica tra il divisore e il quoziente; si divide allora il radicando per questa media e si prosegue ricalcolando la media e continuando a dividere fino a che divisore e quoziente sono uguali.

L'estrazione di una radice può quindi essere ridotta ad una serie di divisioni e somme, usando le routines già viste (progr. 2 e 9).

16:1 =16	$\frac{16 + 1}{2} = 8,5$
16:8,5=1,88235	M=5,19118
16:5,2=3,08215	M=4,13666
16:4,1=3,86785	M=4,00225
16:4,0022575=3,9977437	M=4,0000006
16:4,00...=3,9999994	M=4
16:4=4	STOP

FIG. 16

Questo algoritmo può sembrare lungo come tempo di esecuzione ma in realtà non sono richieste in generale più di 10 o 15 divisioni per un numero di 3 bytes, come emerge chiaramente dalla fig.16.

Il programma all'inizio definisce il numero di bytes su cui i programmi di divisione e di somma andranno ad agire. Questa variabile L

è ottenuta dalla somma del numero di bytes del radicando e dal doppio dei bytes frazionari che il programmatore desidera, analogamente a quan

to fatto nell'algoritmo manuale. Inoltre il programma vero e proprio è preceduto dalla definizione delle variabili che contengono gli indirizzi del dividendo, del divisore e del resto, che occupano una di seguito all'altra i primi bytes a partire dalla locazione I4 fino a dove è necessario, in funzione della variabile L.

Il programma vero e proprio inizia con l'azzeramento del dividendo e del resto; quindi il radicando viene trasferito alla locazione ADDRA, operazione che è necessario ripetere ogni volta dato che il dividendo è distrutto dalla routines DIVISO. Da notare che il radicando è posto come dividendo non partendo dalla prima cifra a destra ma lasciando il doppio di bytes frazionari desiderati nella radice liberi.

A questo punto è realizzata la divisione richiamando la routine DIVISO (da notare che il valore del divisore alla prima divisione è completamente casuale); sono quindi confrontati il divisore e il quoziente: se sono uguali il programma è bloccato altrimenti prosegue con il calcolo della loro media aritmetica (con la loro somma e la divisione per 2 realizzata con lo scorrimento di un bit verso destra); il programma riprende quindi dall'inizio.

Un altro modo per bloccare il programma può essere la creazione di un indice che determini il numero massimo di cicli da eseguire: si tratta di un metodo che limita la precisione ma che può riverarsi utile nel ca-

PAGE 0001

ADDR OBJECT SOURCE

Estrazione radice quadrata

; \$A e \$B=Indirizzo del radicando
; \$4 e \$5=Indirizzo della radice
; (Reg X=Numero massimo di cicli)
; \$C=Numero cifre intere Del radicando
; \$D=Numero byte frazionari della radice

ADDRA=\$0
ADDRB=\$2
ADDRQ=\$4
ADDRR=\$6
L=\$8
CONT=\$9
RAD=\$A
RADICE=\$4
INT=\$C
FRA=\$D
DIVISO=\$087C

0000 A5 0D	LDA FRA
0002 0A	ASL A
0003 65 0C	ADC INT
0005 85 08	STA L
0007 A9 00	LDA #\$0
0009 85 01	STA ADDRA+1
000B A9 0E	LDA #\$E
000D 85 00	STA ADDRA
000F 18	CLC
0010 A5 00	LDA ADDRA
0012 65 08	ADC L
0014 85 02	STA ADDRQ
0016 A9 00	LDA #\$0
0018 65 01	ADC ADDRA+1
001A 85 03	STA ADDRQ+1
001C 18	CLC
001D A5 02	LDA ADDRQ
001F 65 08	ADC L
0021 85 06	STA ADDRQ
0023 A9 00	LDA #\$0
0025 65 02	ADC ADDRQ
0027 85 07	STA ADDRQ+1

so che il programma debba essere eseguito in un tempo ben determinato.

Il tempo di esecuzione dipende in primo luogo dal numero di bytes su cui il programma lavora e dal numero di volte che è necessario ripetere il loop. Quest'ultimo valore non è determinabile a priori ma dipende dal numero che casualmente si trova nel divisore al primo ciclo: se ad es. all'inizio vi è già la radice, il programma si blocca subito; se invece questo numero è piuttosto lontano dal valore della radice è necessario ripetere il ciclo diverse volte.

A titolo di esempio, avendo un radicando di 5 bytes e desiderando un byte dopo la virgola (L=6) e ponendo che occorrono 20 loop (generalmente più che sufficienti), il tempo di esecuzione è; di meno di 300 mS.

Il programma occupa in totale 123 bytes oltre a quelli richiesti dalla routine DIVISO.

```

0029 A4 00      LDY L
002B 00        DEY
002C A9 00      LDA #0
002E 91 00      STA (ADDRA),Y
0030 00        DEY
0031 10 FB      BPL UP1
0033 A4 00      LDY L
0035 00        DEY
0036 A9 00      LDA #0
0038 91 06      STA (ADDRR),Y
003A 00        DEY
003B 10 FB      BPL UP2
003D A4 0C      LDY INT
003F 00        DEY
0040 B1 0A      LDA (RAD),Y
0042 91 00      STA (ADDRA),Y
0044 00        DEY
0045 10 F9      BPL UP3
0047 20 7C 00   JSR DIVISO
004A A4 08      LDY L
004C 00        DEY
004D B1 02      LDA (ADDRB),Y
004F D1 04      CMP (RADICE),Y
0051 D0 05      BNE CONTI
0053 00        DEY
0054 10 F7      BPL UP4
0056 30 23      BMI END
; Le seguenti 2 istruzioni possono essere
; omesse (limitano la precisione)
0058 CA        CONTI
0059 30 20      BMI END
005B 18        CLC
005C A4 00      LDY L
005E 00        DEY
005F B1 02      LDA (ADDRB),Y
0061 71 04      ADC (RADICE),Y
0063 91 02      STA (ADDRB),Y
0065 00        DEY
0066 10 F7      BPL UP5
0068 A0 00      LDY #0
006A 18        CLC
006B B1 02      LDA (ADDRB),Y
006D 6A        ROR A
006E 91 02      STA (ADDRB),Y
0070 C8        INY
0071 00        PHP
0072 C4 08      CPY L
0074 F0 BD      BEQ INIZIO
0076 28        PLP
0077 90 F2      BCC UP6
0079 B0 F0      BCS UP6
007B 00        BRK
;
; Vincenzo Villa   V D
;
; 24/3/1983
; END
ERRORS=0000

```


ARITMETICA IN VIRGOLA MOBILE

Tutti i programmi visti fino ad ora lavorano solo su numeri interi (o comunque su numeri con un numero fisso di decimali e che sono ricondotti con alcuni stratagemmi a numeri interi); si tratta di una rappresentazione efficiente in moltissimi casi ma che si rivela dispendiosa nel momento in cui si ha a che fare con numeri molto grossi e/o tendenti a 0. Ad es., facendo un ragionamento per il sistema decimale, per rappresentare numeri come 20.000.000 o 0,0000002 si è obbligati a dedicare almeno 8 cifre alla parte intera e 7 alla parte decimale, scrivendo un gran numero di zeri che servono solo ad indicare il peso delle cifre significative. Questo problema può essere superato con una notazione in cui il valore del numero è dato dal prodotto di un mantissa e di una base elevata ad un esponente: ad es. i 2 numeri precedenti possono essere rappresentati rispettivamente come 2×10^7 e 2×10^{-7} .

Lo stesso discorso può essere rapportato ai numeri binari, con una rappresentazione tipica mostrata in figura 17: da essa appare come l'esponente è rappresentato con il suo segno nel byte più a sinistra mentre la mantissa è costituita dai 3 bytes consecutivi (il primo

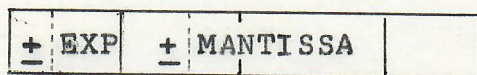


FIG. 17

bit è il segno). E' così possibile rappresentare numeri compresi tra $+ 2,85 \times 10^{45}$ e $- 2,9 \times 10^{-39}$. La mantissa anche quando è negativa è espressa nella sua forma naturale, mentre l'esponente negativo è espresso col suo complemento a 2.

In una rappresentazione di questo tipo si finirà inevitabilmente per "perdere" alcune cifre meno significative oltre i 23 bit, fatto che ne rende sconsigliato l'uso per calcoli di tipo contabile o che comunque richiedono una precisione su tutte le cifre.

NORMALIZZAZIONE

Una caratteristica dei numeri in virgola mobile è di essere normalizzati, di avere cioè il bit più significativo della mantissa (escluso il segno) uguale ad 1. Ad es. il numero 1 000100 EXP 0 100 (equivalente a $-9 \times 2^4 = -144$) è normalizzato spostando verso sinistra la mantissa di 3 bit (lasciando però fisso il segno) e diminuendo di 3 l'esponente, cioè

eguagliandolo a I 1001000 EXP 0 001 (equivalente a $-72 \times 2^1 = -144$). Alla destra del numero possono essere inseriti tanto degli zeri come in questo caso (se non esistono o non sono conosciute altre cifre meno significative) quanto i bit, contenuti in altre locazioni di memoria considerate la continuazione del numero stesso (ad es. il numero su

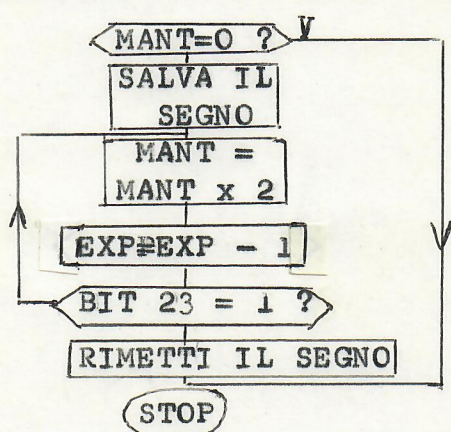


FIG. 18

rappresentato avrebbe potuto essere in realtà I 0001001 (,110...) EXP 0 100, quindi normalizzato sarebbe stato I 1001110 EXP 0 001). Lo schema a blocchi che realizza tale funzione è rappresentato in fig. 18, in cui MANT è la mantissa del numero e EXP l'esponente. Da notare che all'inizio è necessario controllare se la mantissa è 0 perché, in questo caso, è inutile cercare un I dove non ne esistono.

Da notare infine che la mantissa del numero è formata dai 3 bytes di mantissa vera e propria ed altri 3 bytes che potremmo considerare la parte decimale.

Il programma che realizza tale funzione è **PROGR.II**.

Per salvare il segno per poi rimetterlo al primo bit sono forzati a 0

PAGE 0001

ADDR OBJECT SOURCE

; Normalizzazione
; #6 e #7=Indirizzo del numero (alto)
; #4 e #5=Indirizzo del numero (basso)
; Routine ERROR da cambiare se necessario

```

      ADDRNL=#6
      ADDRNL=#4
      SIG=#B
      ERROR=#750
0000 A0 01      LDY #1
0002 B1 06      LDA (ADDRNL),Y
0004 C8         INY
0005 11 06      ORA (ADDRNL),Y
0007 C8         INY
0008 11 06      ORA (ADDRNL),Y
000A F0 2F      BEQ ZERO
000C A0 01      LDY #1
000E B1 06      LDA (ADDRNL),Y
0010 29 00      AND #80
0012 05 0B      STA SIG
0014 18         CLC
0015 B1 06      LDA (ADDRNL),Y
0017 29 40      AND #40
0019 D0 28      BNE END1
001B A0 03      LDY #3
001D B1 04      LDA (ADDRNL),Y
  
```

```

001F 2A         ROL A
0020 91 04      STA (ADDRNL),Y
0022 88         DEY
0023 D0 F8      BNE LOOP1
0025 A0 03      LDY #3
0027 B1 06      LDA (ADDRNL),Y
0029 2A         ROL A
002A 91 06      STA (ADDRNL),Y
002C 88         DEY
002D D0 F8      BNE LOOP2
002F 38         SEC
0030 B1 06      LDA (ADDRNL),Y
0032 E9 01      SBC #1
0034 91 06      STA (ADDRNL),Y
0036 50 DD      BVC SU
0038 20 50 07   JSR ERROR
003B A0 00      LDY #0
003D A9 00      LDA #0
003F 91 06      STA (ADDRNL),Y
0041 F0 08      BEQ END2
0043 A0 01      LDY #1
0045 B1 06      LDA (ADDRNL),Y
0047 05 0B      ORA SIG
0049 91 06      STA (ADDRNL),Y
004B 00         BRK (0 RTS)
; Vincenzo Villa   V D   7/3/1983
      END
      ERRORS=0000
  
```

PROG. 11

i bits 6-0 del primo byte (naturalmente solo nell'accumulatore); alla fine del programma questo bit è rimesso al suo posto con un OR.

Il salto alla routine ERROR serve per segnalare un eventuale OVERFLW dell'esponente, nel caso che questo diventa minore di -128.

Il tempo di esecuzione è dato da $T = 26 + 92n$, con n =numero di spostamenti richiesti dalla mantissa.

SOMMA

La somma F.P. è un'operazione piuttosto lunga da realizzare soprattutto perché è necessario rendere uguali gli esponenti dei due numeri (ad es. non si può realizzare direttamente la somma $2 \times 10^3 + 2,23 \times 10^5$, ma bisogna sommare $0,02 \times 10^5$ e $2,23 \times 10^5$). Un'altra difficoltà nasce nel momento in cui bisogna sommare mantisse negative: infatti è necessario

realizzare il complemento a 2.

Lo schema a blocchi completo è raffigurato in fig. 19. Come si nota si tratta di blocchi piuttosto brevi tranne quelli in cui si "prepara" A o B e quello del complemento a 2, che meritano una spiegazione a parte.

La preparazione (prog.12) consiste nell'eguagliare gli esponenti dei due numeri, incrementando il minore: questo si ottiene spostando verso destra i tre Bytes ^{della mantissa} (tranne il segno) tante volte quanto è la differenza tra i due esponenti.

Lo schema a blocchi è in fig. 20; durante lo sviluppo di questo programma alcuni bit andranno persi,

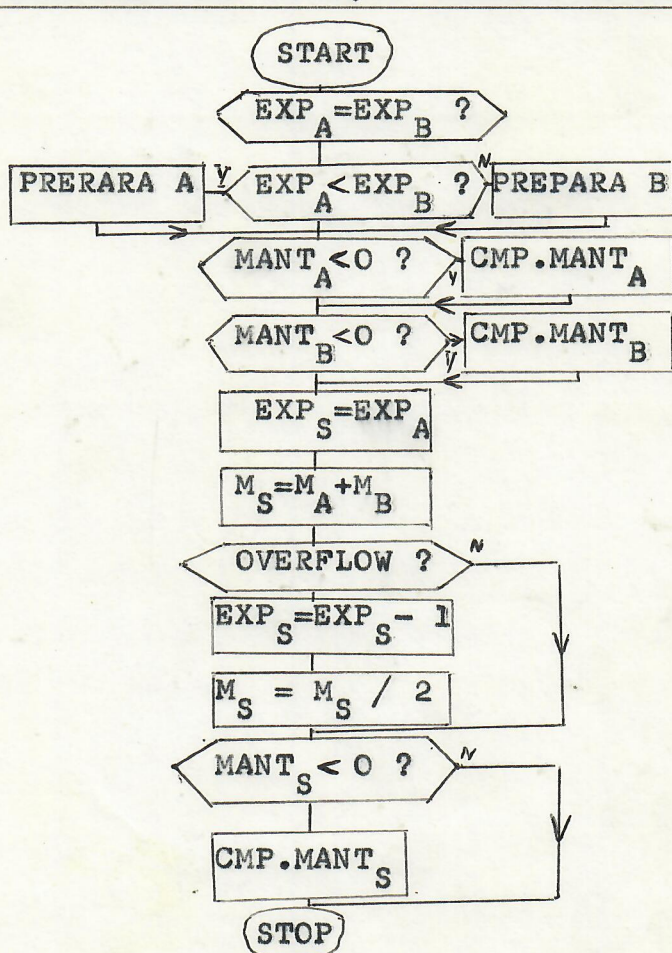


FIG. 19

specialmente se $EXP_A \ll EXP_B$.

Il ragionamento fatto per A vale per ogni numero dato che l'indirizzo è indicizzato: prima di richiamare la routine l'indirizzo dell'esponente da decrementare va posto nel registro X, quello con cui confrontarlo

ADDR OBJECT SOURCE

; Preparazione per somme di numeri F.P.
 ; #0 = Esponente definitivo
 ; Registro X = Indirizzo del numero

```
0000 D0      PREP   CLD
0001 B5 01    LDA   1,X
0003 29 00    AND   #00
0005 85 01    STA   1
0007 A9 7F    LDA   #7F
0009 35 01    AND   1,X
000B 95 01    STA   1,X
000D 18      CLC
000E B5 00    SU     LDA   0,X
0010 C5 00    CMP   0
0012 F0 00    BEQ   END
0014 F6 00    INC   0,X
0016 18      CLC
0017 76 01    ROR   1,X
0019 76 02    ROR   2,X
001B 76 03    ROR   3,X
001D 18      CLC
001E 90 EE    BCC   SU
0020 B5 01    END    LDA   1,X
0022 05 01    ORA   1
0024 95 01    STA   1,X
0026 00      BRK
```

; Vincenzo Villa V D 7/3/1983

.END

ERRORS=0000

PROG. 12

nella locazione 0.

Il complemento a due di un numero si ottiene complementando ogni singolo bit (con l'istruzione ORA) FF). IL segno deve però rimanere invariato, e per questo è salvato prima di iniziare il programma vero e proprio. Ai bit complementati deve essere poi sommato 1.

Il programma che realizza una tale funzione è prog.13.

Da notare che, per ritornare al numero originario è sufficiente richiamare lo stesso programma.

PAGE 0001

ADDR OBJECT SOURCE

; Complemento a 2 e viceversa
 ; #2 e #3 = Indirizzo di N

```
0000 A0 01    LDY   #1
0002 B1 02    LDA   (2),Y
0004 10 29    BPL   END
0006 A0 03    LDY   #3
0008 A9 FF    SU     LDA   #FF
000A 51 02    EOR   (2),Y
000C 91 02    STA   (2),Y
000E 88      DEY
000F D0 F7    BNE   SU
0011 A0 03    LDY   #3
0013 38      SEC
0014 A9 00    SU2    LDA   #0
0016 71 02    ADC   (2),Y
0018 91 02    STA   (2),Y
001A 88      DEY
001B D0 F7    BNE   SU2
001D 50 0A    BVC   GIU
001F A6 02    LDX   2
0021 76 01    ROR   1,X
0023 76 02    ROR   2,X
0025 76 03    ROR   3,X
0027 F6 00    INC   0,X
0029 A9 00    GIU    LDA   #00
002B 15 01    ORA   1,X
002D 95 01    STA   1,X
002F 00      BRK
```

; Vincenzo Villa V D 9/3/1983

.END

ERRORS=0000

PROG. 13

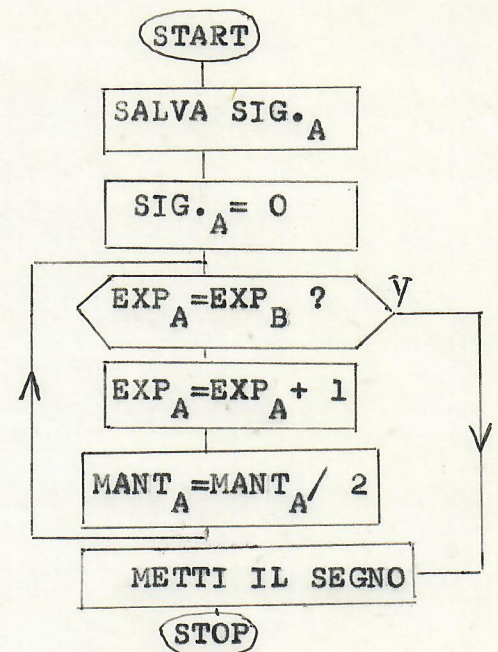


FIG.20 - Schema della "preparazione" (Prog.12)

MOLTIPLICAZIONE F.P.

PAGE 0001

ADDR OBJECT SOURCE

; MOLTIPLICAZIONE FLOATING POINT

; #0 e #1=Indirizzo di A

; #2 e #3=Indirizzo di B

; #6 e #7=Indirizzo del prodotto

```

;
;      ADDRA=#0
;      ADDR B=#2
;      ADDR P=#6
;      SIG=#12
;      ADDR RL=#4
;      ADDR S=#8
;      L=#A
;      MOLT=#8CE
;      NORM=#800
;

```

; Routine di normalizzazione da 800 a 84B

```

;      *=$84C
084C A9 00      LDA #0
084E A0 03      LDY #3
0850 91 06      AZZ STA (ADDRP),Y
0852 88        DEY
0853 10 FB      BPL AZZ
0855 A9 0C      LDA #C
0857 85 04      STA ADDRRL
0859 A9 0F      LDA #F
085B 85 08      STA ADDR S
085D A9 00      LDA #0
085F 85 05      STA ADDRRL+1
0861 85 09      STA ADDR S+1
0863 A9 03      LDA #3
0865 85 0A      STA L
0867 E6 00      INC ADDRA
0869 D0 02      BNE GIU0
086B E6 01      INC ADDRA+1
086D E6 02      GIU0 INC ADDR B
086F D0 02      BNE GIU1
0871 E6 03      INC ADDR B+1
0873 E6 06      GIU1 INC ADDR P
0875 D0 02      BNE GIU2
0877 E6 07      INC ADDR P+1
0879 A0 00      GIU2 LDY #0
087B A9 80      LDA #80
087D 31 00      AND (ADDRA),Y
087F 85 12      STA SIG
0881 A9 80      LDA #80
0883 31 02      AND (ADDR B),Y
0885 45 12      EOR SIG
0887 85 12      STA SIG
0889 B1 00      LDA (ADDRA),Y
088B 29 7F      AND #7F
088D 91 00      STA (ADDRA),Y
088F B1 02      LDA (ADDR B),Y
0891 29 7F      AND #7F
0893 91 02      STA (ADDR B),Y
0895 20 CE 08    JSR MOLT
0898 38        SEC
0899 A5 00      LDA ADDRA
089B E9 01      SBC #1
089D 85 00      STA ADDRA
089F B0 02      BCS GIU3
08A1 C6 01      DEC ADDRA+1

```

PROG.14

La moltiplicazione, presenta, rispetto alla somma, uno schema più semplice, specialmente avendo già a disposizione il programma 5.

In pratica si tratta di moltiplicare tra loro le mantisse, per ottenere la mantissa del prodotto, e di sommare gli esponenti: ad es. $2 \times 10^2 \times 3 \times 10^3 = 6 \times 10^5$.

Per il segno della mantissa è sufficiente, considerando che -x- e +x + danno + e che -x+ e +x- danno -, realizzare un EXOR tra i due bits del segno.

Il programma richiama le routines MOLT, per realizzare il prodotto delle mantisse, e la routine NORM per normalizzare il risultato.

Il tempo di esecuzione (che praticamente coincide con quello della moltiplicazione) varia da un massimo di 2,15 mS ad un minimo di 1,95

Il programma occupa 128 bytes oltre a quelli delle due routines richiamate.


```

08A3 38      GIU3  SEC
08A4 A5 02      LDA ADDR8
08A6 E9 01      SBC #1
08A8 85 02      STA ADDR8
08AA B0 02      BCS GIU4
08AC C6 03      DEC ADDR8+1
08AE A5 06      GIU4  LDA ADDR8
08B0 38      SEC
08B1 E9 01      SBC #1
08B3 85 06      STA ADDR8
08B5 B0 02      BCS GIU5
08B7 C6 07      DEC ADDR8+1
08B9 A0 00      GIU5  LDY #0
08BB 18      CLC
08BC B1 00      LDA (ADDR8),Y
08BE 71 02      ADC (ADDR8),Y
08C0 91 06      STA (ADDR8),Y
08C2 20 00 08      JSR NORM
08C5 A0 01      LDY #1
08C7 A5 12      LDA SIG
08C9 11 06      ORA (ADDR8),Y
08CB 91 06      STA (ADDR8),Y
08CD 00      BRK (o RTS)
;Segue la routine di moltiplicazione
;Vincenzo Villa  V D      13/4/1983
      END

ERRORS=0000

```

PROG. 14 - Fine

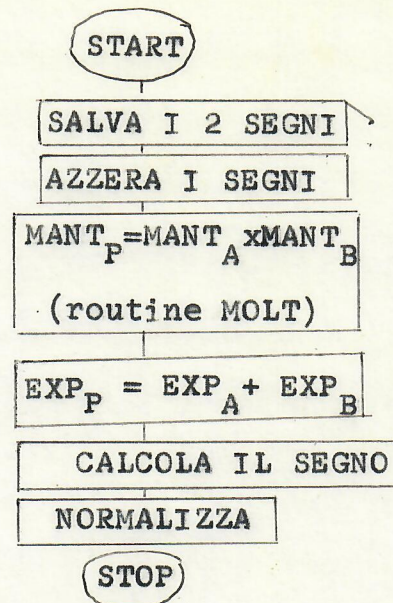


FIG.21 - Schma a blocchi

ARITMETICA DECIMALE (B C D)

Tutti i programmi esaminati fino ad ora lavorano essenzialmente su numeri binari. Generalmente però tutte le funzioni di input e di output numerici verso l'uomo sono realizzate in codice decimale: si deve quindi procedere ad una conversione tra i due codici ogni volta che si instaura questo rapporto (vedi a proposito i programmi di conversione esaminati più avanti).

Un metodo che evita la conversione, usato quando questa comunicazione è molto frequente o non si vuole perdere alcun decimale (cosa che avviene nelle conversioni), è l'uso di un codice che permette al microprocessore di calcolare direttamente in decimale.

Per far questo si usa il codice BCD (Binary Coded Decimal) cioè un codice decimale codificato in binario (TAB.2): in pratica ogni cifra decimale è rappresentata da una serie di quattro bits, tale da far sì che un byte contenga due cifre decimali; le combinazioni

1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
0	0000

TAB. 2

non usate (da IOIO a IIII) non hanno senso in BCD.

Oate queste particolarità le regole solite dell'aritmetica binaria non hanno più valore. IL 6502 é uno dei microprocessori attualmente disponibile che può meglio lavorare con numeri BCD: esiste infatti uno speciale istruzione (SED) che prepara l'unità centrale a realizzare somme e sottrazioni direttamente in decimale, senza la necessità di codifiche o decodifiche; per tornare a lavorare in binario il microprocessore deve ricevere l'istruzione CLD.

Questo permette di realizzare i programmi di somma o sottrazione con lo stesso algoritmo già visto per i numeri binari: é infatti sufficiente far precedere al programma l'istruzione SED.

Da notare che, se é vero che il tempo impiegato per eseguire la somma, ad es. di tre bytes, il tempo impiegato é uguale, sia per la somma BCD che per quella binaria, con tre bytes in BCD si possono rappresentare numeri fino a 9999999; mentre in binario si arriva fino a 16777215, cioè a numeri

più di 16 volte superiori; questo rapporto aumenta ulteriormente con l'aumento dei bytes utilizzati: con quattro bytes é 43, con 5 bytes 110. Questo fatto di essere meno compatto é lo svantaggio principale di questo sistema di numerazione, che si rivela abbastanza grave nel momento in cui si hanno grandi quantità di numeri; questo fatto si riflette negativamente anche durante la realizzazione di programmi matematici in termini di tempo.

PAGE 0001

ADDR OBJECT SOURCE

```

; Somma multipla precisione (BCD)
; #1=Indirizzo di A
; #3=Indirizzo di B
; #3=Indirizzo della somma
; #0=Numero di cifre diviso 2 meno 1
        ADDRA=#1
        ADDR B=#3
        ADDR R=#5
        PUTDIS=#F7D3
        *=$750
        .BYT 'CARRY ON'
        *=$800
        SED
        LDY 0
        CLC
        LDA (ADDR A),Y
        ADC (ADDR B),Y
        STA (ADDR R),Y
        DEY
        BPL SU
        LDX #0
        BCC END
        LDA $750,X
        JSR PUTDIS
        INX
        CPX #8
        BNE RISV
        END
        BRK
; Vincenzo Villa V D 16/2/1983
        END

```

ERRORS=0000

PROG. 15

MOLTIPLICAZIONE BCD

La moltiplicazione BCD viene realizzata con lo stesso algoritmo della moltiplicazione manuale, con uno schema già presentatato

a proposito della moltiplicazione binaria; dato che una cifra non é più solo 0 o 1 come in binario, é però necessario realizzare il prodotto di ogni singola cifra decimale con il moltiplicando opportunamente spostato.

Da notare poi che il moltiplicando, per essere moltiplicato per 10 deve essere spostato di 4 bit sulla sinistra cioè di una cifra decimale.

Il prodotto tra una singola cifra del moltiplicatore e il moltiplicando é realizzato dalla routine SMP attraverso una serie di somme successive; dato che al massimo si debbono realizzare 9 somme, questo metodo si rivela più veloce di quello usato nella moltiplicazione binaria.

Infine é necessario separare all'interno di un byte le due cifre decimali: per quella a destra é sufficiente forzare a 0 i bit 4 - 7; per quella a sinistra é invece necessario spostare 4 volte verso destra il byte.

PAGE 0001

ADDR OBJECT SOURCE

; Moltiplicazione BCD 8 cifre X 8 cifre
; Prodotto di 16 cifre
; \$0 e \$1 = Indirizzo di A
; \$2 e \$3 = Indirizzo di B (prec. da 4 0)
; \$4 e \$5 = Indirizzo del prodotto

ADDRA=\$0
ADDRB=\$2
ADDRR=\$4
CONT=\$6

0000	F8		SED
0001	A0	03	LDY #3
0003	04	06	STY CONT
0005	A4	06	LDY CONT
0007	B1	00	LDA (ADDRA), Y
0009	29	0F	AND #\$F
000B	AA		TAX
000C	20	2D 08	JSR SMB
000F	20	42 08	JSR RORR
0012	A4	06	LDY CONT
0014	B1	00	LDA (ADDRA), Y
0016	4A		LSR A
0017	4A		LSR A
0018	4A		LSR A
0019	4A		LSR A
001A	AA		TAX
001B	20	2D 08	JSR SMB
001E	A4	06	LDY CONT
0020	C0	00	CPY #0
0022	F0	08	BEG END
0024	C6	06	DEC CONT
0026	20	42 08	JSR RORR
0029	4C	05 08	JMP INI
002C	00		END
002D	E0	00	BRK
002F	F0	10	CPX #0
0031	A0	07	BEG END2
0033	18		LDY #7
0034	B1	02	CLC
0036	71	04	LDA (ADDRB), Y
0038	91	04	ADC (ADDRR), Y
003A	88		STA (ADDRR), Y
003B	10	F7	DEY
003D	CA		BPL SU
003E	4C	2D 08	DEX
0041	60		JMP SMB
0042	A2	04	END2
0044	A0	07	RTS
0046	18		LDX #4
0047	B1	02	LDY #7
0049	2A		CLC
004A	91	02	LDA (ADDRB), Y
004C	88		ROL A
004D	10	F8	STA (ADDRB), Y
004F	CA		DEY
0050	D0	F2	BPL SU2
0052	60		DEX
			BNE SU3
			RTS

; Vincenzo Villa

V D

11/3/1983

END

ERRORS=0000

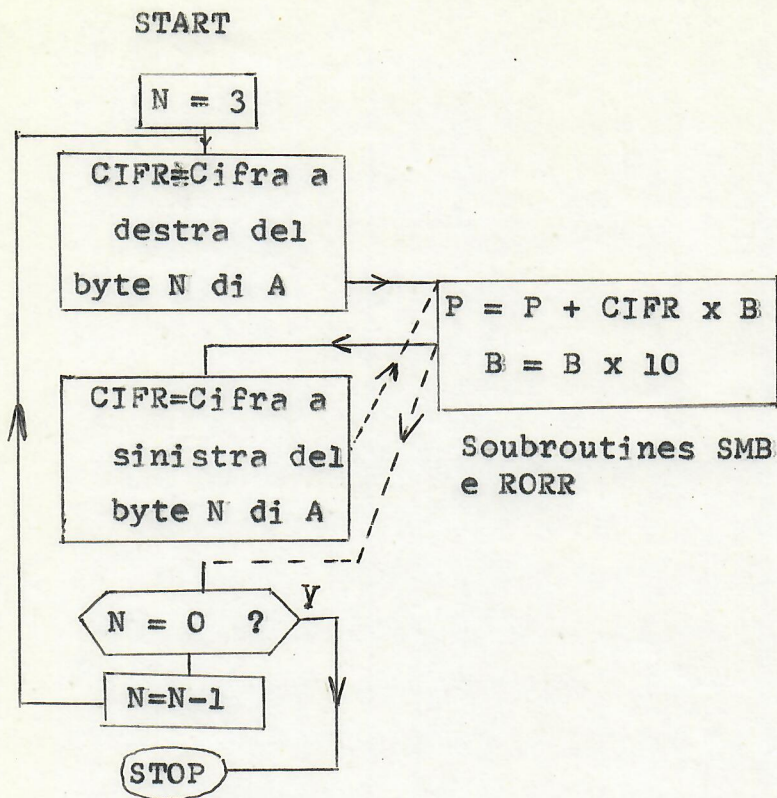


FIG. 22

Il programma funziona su numeri di 8 cifre decimali (cioè la capacità di una normale calcolatrice tascabile); il risultato è però formato fino ad un massimo di 16 cifre.

Eventualmente questo numero di cifre può essere variato agendo sul contatore CONT, secondo la formula $CONT = \frac{n}{2} - 1$ in cui n è il numero, necessariamente pari di cifre.

Come già detto per i numeri binari, questo programma funziona solo su numeri interi o numeri in virgola mobile.

CONVERSIONI TRA CODICI

Quando si prepara un programma complesso ci si trova spesso a dover risolvere problemi che richiedono particolari codici: ad es. può essere richiesta una elaborazione di dati ricevuti da una tastiera decimale con un codice binario e inviare dei dati numerici ad una stampante che usa un proprio codice.

Per risolvere questi problemi é quindi necessario avere a disposizione programmi che realizzano la conversione tra questi codici.

Sia il codice BCD che binario sono già stati descritti nella struttura e negli usi fondamentali;

un altro codice molto usato nell'elaborazione di dati, soprattutto quando si tratta di comunicare o memorizzare dati alfanumerici, é il codice ASCII (American Standard Code for Information Interchange) riportato nella tabella 3; in pratica in una serie di 7 bit é rappresentato il codice di tutte le lettere maiuscole e minuscole dei numeri, della punteggiatura e di alcuni simboli speciali; esistono inoltre 30 caratteri che indicano la fine o l'inizio di un testo, la separazione di testi diversi, il salto di una riga ed altre istruzioni che permettono di trasmettere un testo.

ASCII CHARACTER SET

HEX	DEC	ASCII	HEX	DEC	ASCII	HEX	DEC	ASCII	HEX	DEC	ASCII
00	0	NUL	20	32	SP	40	64	@	60	96	
01	1	SOH	21	33	!	41	65	A	61	97	a
02	2	STX	22	34	"	42	66	B	62	98	b
03	3	ETX	23	35	#	43	67	C	63	99	c
04	4	EOT	24	36	\$	44	68	D	64	100	d
05	5	ENQ	25	37	%	45	69	E	65	101	e
06	6	ACK	26	38	&	46	70	F	66	102	f
07	7	BEL	27	39	'	47	71	G	67	103	g
08	8	BS	28	40	(	48	72	H	68	104	h
09	9	HT	29	41	)	49	73	I	69	105	i
0A	10	LF	2A	42	*	4A	74	J	6A	106	j
0B	11	VT	2B	43	+	4B	75	K	6B	107	k
0C	12	FF	2C	44	,	4C	76	L	6C	108	l
0D	13	CR	2D	45	-	4D	77	M	6D	109	m
0E	14	SO	2E	46	.	4E	78	N	6E	110	n
0F	15	SI	2F	47	/	4F	79	O	6F	111	o
10	16	DLE	30	48	0	50	80	P	70	112	p
11	17	DC1	31	49	1	51	81	Q	71	113	q
12	18	DC2	32	50	2	52	82	R	72	114	r
13	19	DC3	33	51	3	53	83	S	73	115	s
14	20	DC4	34	52	4	54	84	T	74	116	t
15	21	NAK	35	53	5	55	85	U	75	117	u
16	22	SYN	36	54	6	56	86	V	76	118	v
17	23	ETB	37	55	7	57	87	W	77	119	w
18	24	CAN	38	56	8	58	88	X	78	120	x
19	25	EM	39	57	9	59	89	Y	79	121	y
1A	26	SUB	3A	58	:	5A	90	Z	7A	122	z
1B	27	ESC	3B	59	;	5B	91	[	7B	123	{
1C	28	FS	3C	60	<	5C	92	\	7C	124	
1D	29	GS	3D	61	=	5D	93	]	7D	125	}
1E	30	RS	3E	62	>	5E	94	^	7E	126	~
1F	31	VS	3F	63	?	5F	95	_	7F	127	DEL

NUL	- Null	DLE	- Data Link Escape
SOH	- Start of Heading	DC	- Device Control
STX	- Start of Text	NAK	- Negative Acknowledge
ETX	- End of Text	SYN	- Synchronous Idle
EOT	- End of Transmission	ETB	- End of Transmission Block
ENQ	- Enquiry	CAN	- Cancel
ACK	- Acknowledge	EM	- End of Medium
BEL	- Bell	SUB	- Substitute
BS	- Backspace	FSC	- Escape
HT	- Horizontal Tabulation	FS	- File Separator
LF	- Line Feed	GS	- Group Separator
VT	- Vertical Tabulation	RS	- Record Separator
FF	- Form Feed	US	- Unit Separator
CR	- Carriage Return	SP	- Space (Blank)
SO	- Shift Out	DEL	- Delete
SI	- Shift In		

TAB. 3

B C D ==> H E X

Un primo programma che andiamo ad esaminare é la conversione da decimale (BCD) a binario; dato che la conversione da binario a esadecimale é immediata (TAB I) é spesso usata la sigla HEX per indicare il codice binario. Prima di parlare di questa conversione é bene, riferendosi al sistema decimale illustrare come si realizza la conversione da una base qualsiasi a quella con cui le operazioni di conversione sono eseguite; in pratica (fig. 23) si moltiplica la prima cifra sulla destra per 1 (la base elevata a 0), la seconda per la base, la terza per la base alla seconda ecc.

$$1373_8 = 2 \times 8^0 + 7 \times 8^1 + 3 \times 8^2 + 8^3 = 762_{10}$$

FIG.23

Per convertire da decimale a binario (la base con cui il microprocessore lavora) si segue lo stesso procedimen

to dato che il calcolatore converte da una base qualsiasi alla sua base ed in questa realizza tutte le operazioni.

La conversione viene realizzata attraverso il prodotto delle singole cifre per un contatore che contiene uno di seguito all'altro i numeri 1, 10, 100 e 1000. Il prodotto é realizzato con la routine di moltiplicazioni già vi sta. Nel programma semplice, i numeri sono messi direttamente nella memoria altrimenti si sarebbe reso necessario moltiplicare ad ogni cifra il contatore stesso per 10, rallentando l'esecuzione; nel caso che la velocità non ha un peso determinante (ad es. nei programmi di input da tastiera), il

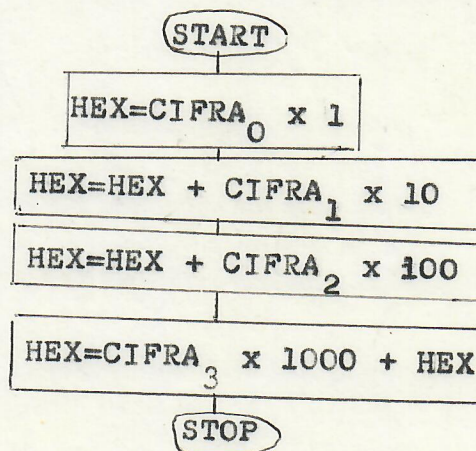


FIG.24

contatore é calcolato dal programma stesso, rendendolo così più compatto e flessibile, dato che il numero decimale non é più legato a una lunghezza fissa, in questo caso 4 cifre.

Da notare che lo stesso programma può servire per convertire un numero al binario da qualsiasi base: basta infatti caricare nel contatore le potenze della base dalla quale si vuole convertire.

Un programma che realizza una tale funzione é il programma I6 riportato nella pag. seguente.

PAGE 0001

ADDR OBJECT SOURCE

```

; Conversione BCD ==> HEX (2 byte)
; Decimali $0 $1 $2 e $3
; Esadecimale in $4 e $5 (prima = #00)
0000 D0          CLD
0001 A9 00        LDA #0
0003 85 06        STA 6
0005 A9 01        LDA #1
0007 85 07        STA 7
0009 20 32 08     JSR IN
000C A5 02        LDA 2
000E 85 03        STA 3
0010 A9 0A        LDA #A
0012 85 07        STA 7
0014 20 32 08     JSR IN
0017 A5 01        LDA 1
0019 85 03        STA 3
001B A9 64        LDA #100
001D 85 07        STA 7
001F 20 32 08     JSR IN
0022 A9 E8        LDA #E8
0024 85 07        STA 7
0026 A9 03        LDA #3
0028 85 06        STA 6
002A A5 00        LDA 0
002C 85 03        STA 3
002E 20 32 08     JSR IN
0031 00          BRK
0032 A0 01        LDY #1
0034 98          TYA
0035 25 03        AND 3
0037 F0 0D        BEQ ROL1
0039 18          CLC
003A A5 07        LDA 7
003C 65 05        ADC 5
003E 85 05        STA 5
0040 A5 06        LDA 6
0042 65 04        ADC 4
0044 85 04        STA 4
0046 18          CLC
0047 26 07        ROL 7
0049 26 06        ROL 6
004B 98          TYA
004C C9 08        CMP #8
004E F0 05        BEQ END
0050 0A          ASL A
0051 A8          TAY
0052 4C 34 08     JMP IN1
0055 60          RTS
; Vincenzo Villa      V D      28/2/1983
                          .END

```

ERRORS=0000

PROG. 16

re) il numero binario spingendolo verso sinistra con l'istruzione ROL: in pratica si moltiplica il numero binario per 2; nel contempo si moltiplicano i bit uscenti per 2 però in modo decimale sommandoli a se stessi dopo aver fatto eseguire l'istruzione SED.

Un'interpretazione più rigorosa di questo processo è la seguente: il pri-

HEX ==> BCD

Per convertire un numero dalla base decimale ad una base diversa, è sufficiente realizzare una serie di divisioni del numero per questa base: il resto delle divisioni costituirà le cifre del numero nella base decimale, a partire dalla meno significativa. La serie di divisioni finisce quando il quoziente diventa 0.

Lo stesso ragionamento può essere fatto per convertire in un sistema binario un numero da binario a BCD. Il programma I7 è la realizzazione dell'algoritmo che esegue questa funzione. Il suo funzionamento (che impiega circa 4 mS) trasforma un numero binario contenuto nei bytes 2 e 3 (quindi numeri fino a 65535) in 5 cifre BCD, poste in 5 locazioni consecutive pronte per essere inviate ad una periferica.

Il programma occupa, compresa la routine divisione, 93 bytes.

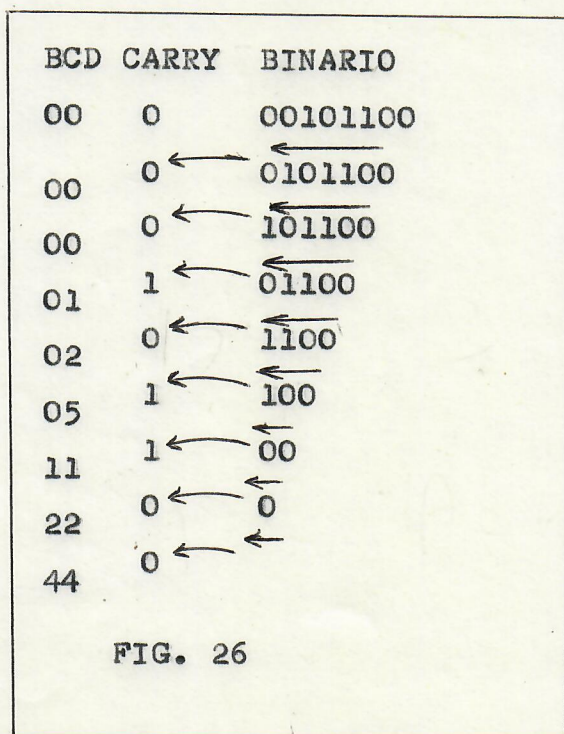
Esiste anche un altro algoritmo che pur avendo caratteristiche simili ha un tempo di esecuzione minore: esso consiste nel far "uscire

mo bit ad uscire é il bit più significativo ed è quello che viene sommato a se stesso 7 volte venendo quindi moltiplicato (in decimale) per 2^7 .

Il secondo bit é invece sommato a se stesso per 6 volte cioè é moltiplicato per 2^6 .

L'ultimo bit é semplicemente sommato ai bit precedenti cioè é moltiplicato per 2^0 ; In pratica viene quindi realizzato lo stesso algoritmo usato manualmente per convertire da binario a decimale facendo però in modo che i prodotti avvengano parallelamente (fig.26).

A differenza del primo programma i numeri BCD sono però contenuti non in bytes adiacenti (uno per cifra) ma sono fusi in un numero minore di bytes: questo rende necessario il dover dividere le singole cifre prima di inviarle ad una stampante o a un'altra periferica.



PAGE 0001

ADDR OBJECT SOURCE

; Conversione HEX=>BCD 2 Byte
; #1=00 #2 e #3=numero esadecimale
; #6,7,8,9,A =bcd

```
0000 D8          CLD
0001 A9 0A       LDA #10
0003 85 00       STA 0
0005 20 1F 00    JSR DIV
0008 85 0A       STA #A
000A 20 1F 00    JSR DIV
000D 85 09       STA 9
000F 20 1F 00    JSR DIV
0012 85 08       STA 8
0014 20 1F 00    JSR DIV
0017 85 07       STA 7
0019 20 1F 00    JSR DIV
001C 85 06       STA 6
001E 00         BRK
```

```
;
001F A2 00       DIV   LDX #0
0021 A0 80       LDY  ##80
0023 18         SU    CLC
0024 26 03       ROL  3
0026 26 02       ROL  2
0028 26 01       ROL  1
002A A5 01       LDA  1
002C C5 00       CMP  0
002E 90 09       BCC  SALT
0030 E5 00       SBC  0
0032 85 01       STA  1
0034 98         TYA
0035 15 04       ORA  4,X
0037 95 04       STA  4,X
0039 98         TYA
003A 4A         SALT  LSR  A
003B F0 04       BEQ  DUE
003D A8         TAY
003E 4C 23 08    JMP  SU
0041 E0 01       DUE   CPX  #1
0043 F0 06       BEQ  END
0045 E8         INX
0046 A0 80       LDY  ##80
```

```
;
0048 4C 23 08    JMP  SU
004B A5 05       END   LDA  5
004D 85 03       STA  3
004F A5 04       LDA  4
0051 85 02       STA  2
0053 A5 01       LDA  1
0055 A2 00       LDX  #0
0057 86 01       STX  1
0059 86 04       STX  4
005B 86 05       STX  5
005D 60         RTS
```

;Vincenzo Villa V D 23/2/1983

END

ERRORS=0000

HEX ==> ASCII E VICEVERSA

Per la rappresentazione di testi alfanumerici è generalmente usato il codice ASCII.

Le periferiche richiedono per la decodifica dei numeri che essi siano rappresentati con questo codice. Così ad es. per stampare una serie di numeri non è sufficiente inviare direttamente il codice binario o BCD ma è necessaria la loro conversione in ASCII.

Osservando la tabella 3 si nota che i numeri sono codificati da \$30 (=0) a \$39 (=9) e le lettere da \$41 (=A) a \$5A (=Z).

I numeri esadecimali sono rappresentati dalle 10 cifre decimali più 6 lettere (A=F); se si tratta di un numero minore \$9 è quindi sufficiente sommare \$30 per ottenere il carattere ASCII corrispondente (es. \$1 => \$31); se si tratta invece di una lettera è necessario aggiungere \$40 (Es. \$B=>\$42).

Lo schema a blocchi è in fig.27.

Nel caso inverso è sufficiente sottrarre \$30 o \$40 rispettivamente. Da notare che è anche necessario controllare prima della conversione che si tratti effettivamente di un numero ^{es2-}decimale: ad es. \$4E = N non corrisponde ad alcuna cifra esadecimale. Lo schema a blocchi è in fig. 28.

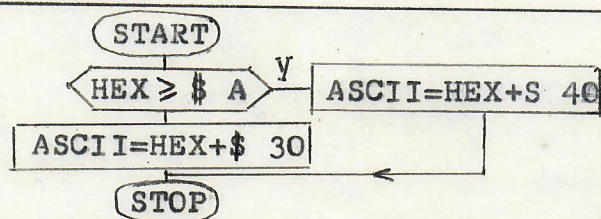


FIG. 27

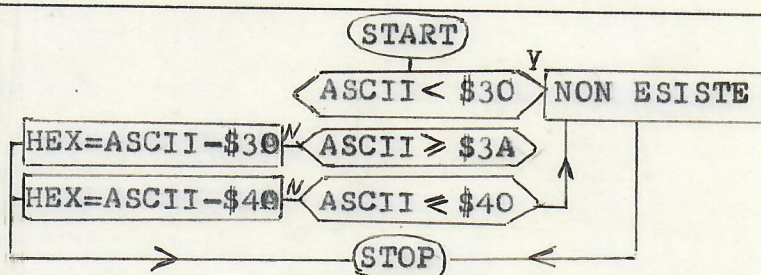


FIG. 28

I programmi che realizzano queste conversioni sono rispettivamente i programmi I8 e I9.

Il monitor dell'AIM 65/40 ha due routines che realizzano la stessa funzione, rispettivamente H2ASCII e A2HEX.

Questo tipo di programma si rivela indispensabile tutte le volte che, dall'assembler è necessario inviare dati alla stampante o al display o è necessario leggere dei dati provenienti dalla tastiera.

Un altro metodo per realizzare conversioni di qualunque tipo è quello


```

PAGE 0001
ADDR OBJECT SOURCE
;
; CONVERSIONE HEX ==> ASCII
;
; ASCII=#1
; HEX=#0
;
0000 18 CLC
0001 A5 00 LDA HEX
0003 18 CLC
0004 69 30 ADC ##30
0006 09 3A CMP ##3A
0008 90 03 BCC RIT
000A 18 CLC
000B 69 07 ADC ##7
000D 85 01 RIT STA ASCII
000F 00 BRK
;Vincenzo Villa V D 27/5/1983
END
ERRORS=0000

```

```

PAGE 0001
ADDR OBJECT SOURCE
;
; CONVERSIONE ASCII ==> HEX
;
; ASCII=#0
; HEX=#1
;
0000 A5 00 LDA ASCII
0002 09 30 CMP ##30
0004 90 0C BCC RTSC
0006 09 47 CMP ##47
0008 B0 08 BCS RTSC
000A 09 3A CMP ##3A
000C 90 06 BCC NUM
000E 09 41 CMP ##41
0010 B0 09 BCS ALFA
0012 18 RTSC CLC
0013 00 BRK
0014 38 NUM SEC
0015 E9 30 SBC ##30
0017 85 01 STA HEX
0019 38 SEC
001A 00 BRK
001B 38 ALFA SEC
001C E9 37 SBC ##37
001E 85 01 STA HEX
0020 38 SEC
0021 00 BRK
;Vincenzo Villa V D 27/5/1983
END
ERRORS=0000

```

PROG.18

PROG. 19

di creare in memoria una tabella con i valori finali della conversione alle locazioni indirizzate dal codice di partenza. Ad es., riferendosi alla conversione HEX ==> ASCII, si può creare una tabella come in fig. 29: volendo ad es. sapere a cosa corrisponde il numero 8 é sufficiente leggere direttamente il contenuto della locazione 8 per trovare il codice ASCII corrispondente (=38). Si tratta di un tipo di conversione velocissima (basta infatti una sola lettura della memoria) ma che richiede una occupazione di memoria non indifferente, nel caso che si abbia a che fare con molti valori, e la necessità di calcolarla precedentemente.

Questo tipo di conversione è usata quando non esiste una semplice legge di conversione tra i due codici: tipiche applicazioni sono quelle che ne fanno il display e la stampante dell'AIM 65/40 per generare i caratteri da stampare dato il codice ASCII.

TABELLA	30	31	32	33	34	35	36	37	38	39	41	42	43	44	45	46
INDIRIZZO	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

FIG. 29

I N P U T E O U T P U T

Tutti i programmi visti fino ad ora lavorano esclusivamente su dati già contenuti nella memoria del sistema e pongono i propri risultati sempre e solo all'interno della memoria; quindi sia per porre i dati su cui il programma lavora sia per leggere ciò che il programma ha realmente fatto è necessario usare un programma apposito che ci mostri il contenuto della memoria ed un altro che ci permette di modificarla. Ad es. per tutti i programmi visti fino ad ora, era necessario inserire i dati con i tasti M e /, usati da Monitor.

In un programma normale è però necessario inserire direttamente dalla tastiera i dati chiesti dal programma e leggere su carta o su un video i risultati di tali programmi. In pratica si tratta di creare programmi atti a leggere lo stato di una periferica (ad es. rilevare un tasto schiacciato e interpretarlo) oppure a trasmettere un segnale che possa rendere evidente un messaggio (ad es. una serie di led accesi oppure un carattere stampato).

La difficoltà di un tale tipo di programmi consiste non tanto nel creare e riconoscere un codice adatto allo scopo, quanto piuttosto nel sincronizzare le operazioni del computer e della periferica che, di solito, presentano un tempo di funzionamento molto diverso: ad es. una stampante richiede per imprimere materialmente un carattere o per posizionare il cursore non meno di qualche millisecondo; il computer richiede invece per trasmettere un carattere o un ordine non più di qualche milionesimo di secondo. In generale, il computer, non può quindi limitarsi ad inviare i caratteri uno di seguito all'altro o a leggere continuamente gli stati di una periferica ma deve controllare che essa abbia finito di eseguire l'ordine precedente o che sia disponibile ad inviare nuovi dati.

In questa sede saranno visti i programmi che permettono di colloquiare in modo efficiente con le periferiche, in particolare è affrontato l'uso della stampante, del display e della tastiera con l'uso delle routines del I/O ROM.

OUTPUT SU STAMPANTE

In questa sezione sono presi in considerazione programmi che permettono di usare la stampante sia in modo normale e semigrafico (per stampare testi alfanumerici, formule e schemi) che in modo grafico (per la stampa di disegni e grafici in alta risoluzione).

Prima di passare ai programmi è meglio descrivere le routines principali del firmware del sistema che permettono di gestire la stampante:

OPNPTR (che si trova all'indirizzo \$F8C8) abilita la stampante a ricevere dati, collegando la porta di uscita del computer a quella di ingresso della stampante; questa routine va chiamata prima di qualsiasi programma che usa la stampante ed in effetti l'istruzione JSR OPNPTR è la prima istruzione di qualunque programma di stampa.

CLOPTR (\$F8B3) ha la funzione contraria di disabilitare la stampante dal ricevere dati ed in pratica è chiamata alla fine di un programma per "spegnere" la stampante.

PUTPTR (\$F8F1) è la routine che materialmente prende il carattere ASCII contenuto nell'accumulatore e lo invia al buffer della stampante, pronto per essere stampato.

Da notare che la prima istruzione della routine PUTPTR richiama la routine WAITP (\$F935) che ha essenzialmente il compito di controllare se la stampante è pronta a ricevere nuovi bytes; in caso contrario questa routine blocca l'esecuzione del programma; l'esecuzione è ripresa

quando la stampante segnale di essere di nuovo disponibile, permettendo così di sincronizzare il computer alla stampante.

Un semplice esempio è il programma 20 il cui schema è riportato nella fig.30. Esso permette di inviare alla stampante un testo formato da NUM caratteri ASCII posto in memoria a partire dalla locazione FROM. Il programma legge con l'indirizzamento indiretto indicizzato un Byte alla volta e lo invia con la routine PUTPTR.

Non è però sufficiente inviare alla stampante

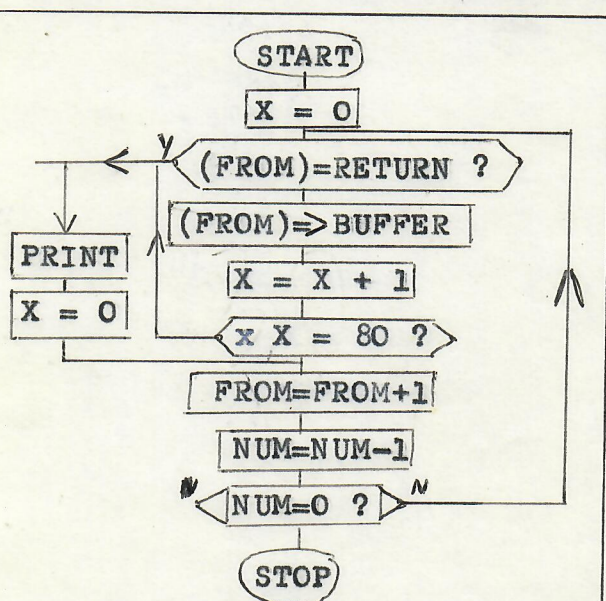


FIG. 30

pante i caratteri uno di seguito all'altro, infatti la stampante raccoglie questi caratteri in un buffer di 80 bytes senza stamparli: per far questo é necessario inviare un byte \$A che segna la stampante di stampare il buffer, di posizionare il cursore sulla prima colonna e di cancellare il buffer.

Questo carattere fa parte di una serie di 32 bytes che non corrispondono ad alcun carattere ASCII ma che servono per indicare alla stampante una serie di ordini da eseguire, come cancellare il buffer, spostare a destra o a sinistra il cursore, cancellare un singolo carattere ecc. (tabella 4). Il byte \$A é inviato dal programma

PAGE 0001

ADDR OBJECT SOURCE

; GESTIONE DELLA STAMPANTE

; FROM=Locazione iniziale da stampare

; NUM=Numero locazioni da stampare

```

OPNPTR=$F8C8
CLOPTR=$F8B3
PUTPTR=$F8F1
WAITP=$F935
FROM=$0
NUM=$2

0000 20 C8 F8      JSR OPNPTR
0003 A0 00        LDY #0
0005 A2 00        LDX #0
0007 B1 00        SU   LDA (FROM),Y
0009 C9 03        CMP #3
000B F0 1B        BEQ END
000D C9 0D        CMP #$0D =RETURN
000F F0 20        BEQ PRINT
0011 20 F1 F8     JSR PUTPTR
0014 E8          INX
0015 E0 51        CPX #81
0017 F0 18        BEQ PRINT
0019 C8          INIZIO INY
001A F0 22        BEQ INCFRM
001C C6 03        RIT  DEC NUM+1
001E D0 E7        BNE SU
0020 C6 02        DEC NUM
0022 A5 02        LDA NUM
0024 C9 FF        CMP #$FF
0026 D0 DF        BNE SU
0028 A9 0A        END  LDA #$A
002A 20 F1 F8     JSR PUTPTR
002D 20 B3 F8     JSR CLOPTR
0030 00          BRK
0031 A9 0A        PRINT LDA #$A =LINE FEED
0033 20 F1 F8     JSR PUTPTR
0036 20 35 F9     JSR WAITP
0039 A2 00        LDX #$0
003B 4C 19 08     JMP INIZIO
003E E6 01        INCFRM INC FROM+1
0040 4C 1C 08     JMP RIT

```

;Vincenzo Villa V D 11/4/1983

END

ERRORS=0000

ASCII Code	Control Character	Description
00	CTRL @	*
01	CTRL A	Clear Buffer
02	CTRL B	Clear to End of Buffer
03	CTRL C	Clear Buffer
04	CTRL D	Clear to End of Buffer
05	CTRL E	Clear Buffer
06	CTRL F	Clear to End of Buffer
07	CTRL G	*
08	CTRL H	Backspace (Left Arrow)
09	CTRL I	Forwardspace (Right Arrow)
0A	CTRL J	Line Feed (Down Arrow)
0B	CTRL K	Line Feed (Down Arrow)
0C	CTRL L	Form Feed
0D	CTRL M	Carriage Return (Home On Line)
0E	CTRL N	Carriage Return (Home On Line)
0F	CTRL O	Carriage Return (Home On Line)
10	CTRL P	Pass Through Next Character
11	CTRL Q	*
12	CTRL R	*
13	CTRL S	Toggle Insert Character Mode
14	CTRL T	Delete One Character
15	CTRL U	*
16	CTRL V	*
17	CTRL W	Cursor On
18	CTRL X	Cursor Off
19	CTRL Y	Cold Reset
1A	CTRL Z	Warm Reset
1B	CTRL [	Escape Command (ESC)
1C	CTRL \	Print Without Clear
1D	CTRL]	Print Display Image (Window)
1E	CTRL ^	Paper Feed
1F	CTRL _	*

NOTE:

(*) Characters with no indicated function are acknowledged, but do not otherwise affect the printer operation.

quando nel testo è incontrato il carattere \$D, coincidente col RETURN che indica la fine di una riga, oppure quando sono stati inviati 80 caratteri di seguito, per evitare la cancellazione dei primi caratteri inviati. Un programma di questo tipo è in grado di stampare dalle due alle quattro righe al secondo di 40 caratteri ciascuna; il tempo di esecuzione non è però legato al tempo impiegato per inviare i caratteri, perchè in questo caso si arriverebbe sull'ordine delle decine di migliaia di caratteri al secondo, quanto piuttosto dalla "lentezza" della stampante che, secondo le specifiche del costruttore impiega dai 0,25 ai 0,5 secondi per stampare una riga.

Questo programma può essere usato per stampare qualunque dei 96 caratteri ASCII standard, i 32 caratteri di controllo quali frecce ed esponenti (se preceduti dal carattere di controllo \$IO ed infine 128 caratteri semigrafici che vanno dalle lettere greche a simboli matematici a lettere con la dieresi a caratteri semigrafici veri e propri.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		@	Q	W	7	7	7	7	7	7	7	7	7	7	7	7
1		0	1	2	3	4	5	6	7	8	9	:	:	:	:	:
2		!	"	#	\$	%	&	'	(	)	*	+	,	-	.	/
3		@	A	B	C	D	E	F	G	H	I	J	K	L	M	N
4		P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^
5		p	q	r	s	t	u	v	w	x	y	z	{		}	~
6		P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^
7		p	q	r	s	t	u	v	w	x	y	z	{		}	~
8		0	1	2	3	4	5	6	7	8	9	:	:	:	:	:
9		!	"	#	\$	%	&	'	(	)	*	+	,	-	.	/
A		@	A	B	C	D	E	F	G	H	I	J	K	L	M	N
B		P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^
C		p	q	r	s	t	u	v	w	x	y	z	{		}	~
D		0	1	2	3	4	5	6	7	8	9	:	:	:	:	:
E		!	"	#	\$	%	&	'	(	)	*	+	,	-	.	/
F		@	A	B	C	D	E	F	G	H	I	J	K	L	M	N

FIG. 31

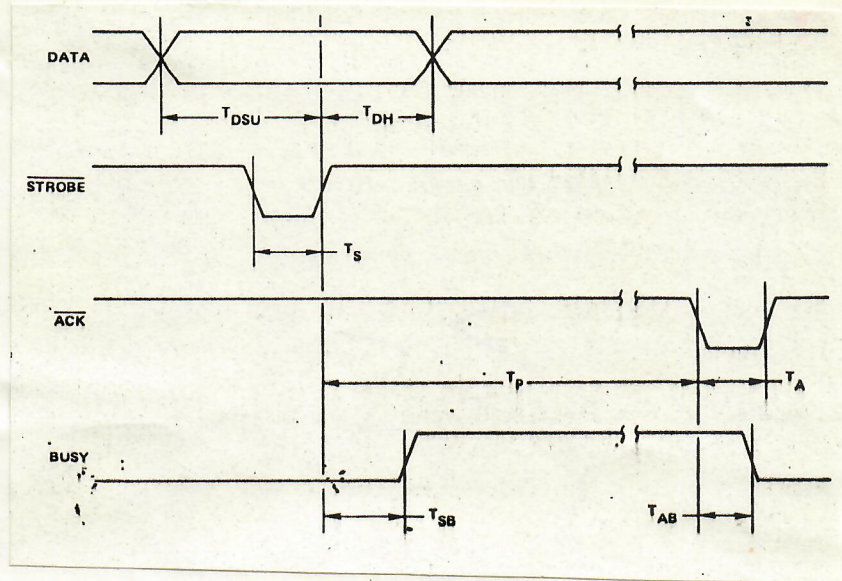
La lista di tutti i caratteri normali e semigrafici è riportata in fig. 31; le cifre esadecimali riportate sulla prima riga e sulla prima colonna del listato, rappresentano rispettivamente la prima e la seconda cifra esadecimale usata per rappresentare ogni singolo carattere.

Il programma appena visto usa le routines del monitor; data l'importanza delle funzioni di I/O è bene esaminare approfonditamente il

loro funzionamento in particolare per quanto riguarda PUTPTR.

In fig. 32 sono mostrate le forme d'onda su alcuni terminali della stampante: a) sulla prima riga (DATA) è riportato il bus di otto linee di ingresso della stampante su cui sono trasmesse tutti i dati che la CPU invia alla stampante (siano essi caratteri o ordini). Questo bus è connesso alla porta A di un integrato VIA 6522 (che sarà descritto nei dettagli successivamente): in pratica compito del programma è quello di porre su

questa porta i bytes da trasmettere alla stampante come in una normale locazione di memoria. b) A questo punto bisogna segnalare alla stampante che il dato disponibile sul bus è valido e non si tratta di valori casuali: questo è fatto attraverso la linea STROBE che è generalmente a uno logico ma che viene forzata a zero e poi di nuovo a uno quando il dato è disponibile sul bus. Questa linea è connessa al bit 2 di un'altra porta dello stesso VIA: bisogna quindi prima forzarlo a zero (AND \$FD) e poi a uno (ORA 2). Da notare che il dato deve rimanere sul bus un tempo relativamente lungo come indicato nella stessa fig. 32: questo è reso possibile dal fatto che le uscite dell'porta sono bufferate cioè, il dato rimane invariato fino a che ne è scritto uno nuovo.



Parameter	Symbol	Min	Typ	Max	Units
Data Set-Up	T_{DSU}	0	-	-	uS
Data Hold	T_{DH}	25	-	-	uS
Strobe Pulse Width	T_S	50	-	-	nS
Processing Time (non-printing)	T_P	0.13	0.15	12	mS
Processing Time (printing)	T_P	0.25	-	0.5	S
Acknowledge Width	T_A	-	5	-	uS
Strobe-to-Busy	T_{SB}	-	14	25	nS
Acknowledge-to-Busy	T_{AB}	-	20	40	nS

NOTE
tr, tf = 10 to 30 ns.

FIG. 32

c) A questo punto la stampante inizia l'elaborazione del dato presente sulla porta A. da notare che il tempo di esecuzione non è costante ma varia fra 0,13 e 500 mS; la fine di questo processo è segnalata dalla linea ACK che generalmente è a 1 ed è posta a 0 dalla stampante quando ha finito il suo lavoro. Il programma deve allora continuamente controllare lo stato di questa linea attraverso un loop (da ATT e per tre istruzioni nel progr. ventuno) da cui esce solo quando il flag che segnala una transizione su questa linea è posto a 1. Il programma completo che svolge tale funzione è il progr. 21 che in pratica svolge le stesse funzioni di PUTPTR.

Il progr. 20 permette di stampare una grande varietà di caratteri; a volte si rende però necessario realizzare stampati raffiguranti disegni,

schemi o diagrammi con una precisione raggiungibile solo con molta approssimazione usando tali caratteri.

Un carattere normale é rappresentato entro un reticolo di otto per sette "....."(dot) in cui alcuni sono stampati ed altri no.(fig. 33)

In caso di stampati ad alta risoluzione é necessario poter individuare singolarmente ogni singolo dot che forma il reticolo e decidere se dovrà essere stampato o no: é così possibile realizzare figure con una risoluzione pari alla

distanza minima tra due puntini. Questo é possibile inviando alla stampante il comando ESC G seguito immediatamente da una sequenza di bytes indicanti quali dots devono essere stampati; i bytes (di cui é ignorato l'ultimo bit) sono così interpretati: il primo bit rappresenta il primo dot, cioè, se esso é zero il primo puntino non é stampato, se é 1 sì;

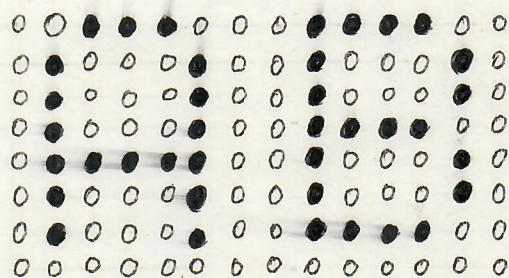


FIG. 33-Lettere A e B

il secondo bit rappresenta il secondo dot ed é interpretato allo stesso modo; il primo bit del secondo byte rappresenta l'ottavo dot e così via fino al quarantesimo byte (infatti quaranta per sette dà 280; il numero dei dots di una riga); il quarantesimo byte rappresenta i primi sette dots della seconda riga.

Per far stampare correttamente tutti i bits é necessario inviare 40 bytes e, mentre questi sono stampati, inviare i successivi 40.

Questa operazione deve avvenire in modo perfettamente sincronizzato con l'operazione di stampa altrimenti si rischia o di sovrapporre serie di bytes se l'invio é troppo veloce, o di non fare arrivare in tempo i nuovi bytes da stampare e quindi far uscire automaticamente la stampante

PAGE 0001

ADDR OBJECT SOURCE

```
; STAMPANTE (senza routine I/O ROM)
DATA=$0
PA=$FFB1
PB=$FFB0
IFR=$FFBD
PCR=$FFBC
```

```
0000 A9 C1 ROUT LDA #$C1
0002 8D BD FF STA IFR
0005 A5 00 LDA DATA
0007 8D B1 FF STA PA
000A AD B0 FF LDA PB
000D 29 FD AND #$FD
000F 8D B0 FF STA PB
0012 09 02 ORA #$2
0014 8D B0 FF STA PB
0017 AD BD FF ATT LDA IFR
001A 29 02 AND #2
001C F0 F9 BEQ ATT
001E AD B1 FF LDA PA
0021 00 BRK
```

;Vincenzo Villa V D 18/4/1983

END

ERRORS=0000

PROG. 21

dai modo grafico nel caso che l'invio è troppo lento.

Per evitare grossi problemi di sincronizzazione, è quindi necessario avere già pronti in memoria tutti i bytes da inviare in modo che il programma non deve fare altro che leggere e inviarli.

PAGE 0001

ADDR OBJECT SOURCE

STAMPANDE IN MODO GRAFICO

; Stampa i byte dalla locazione START a
; STOP in modo grafico

OPNPTR=#F8C8
CLOPTR=#F8B3
PUTPTR=#F8F1
WAITP=#F935
START=#0
STOP=#2
PA=#FFB1
PB=#FFB0
IFR=#FFBD

```

0800 A0 00          LDY #0
0802 58            CLI
0803 20 C8 F8      JSR OPNPTR
0806 A9 1B          LDA #1B
0808 20 F1 F8      JSR PUTPTR
080B 20 35 F9      JSR WAITP
080E A2 47          LDX #47
0810 78            SEI
0811 8E B1 FF      INIZIO STX PA
0814 A9 C1          LDA #C1
0816 8D BD FF      STA IFR
0819 AD B0 FF      LDA PB
081C 29 FD          AND #FD
081E 8D B0 FF      STA PB
0821 09 02          ORA #2
0823 8D B0 FF      STA PB
0826 AD BD FF      ATT  LDA IFR
0829 4A            LSR A
082A 90 FA          BCC ATT
082C B1 00          LDA (START),Y
082E AA            TAX
082F E6 00          INC START
0831 D0 02          BNE OK
0833 E6 01          INC START+1
0835 A5 00          OK  LDA START
0837 C5 02          CMP STOP
0839 D0 D6          BNE INIZIO
083B A5 01          LDA START+1
083D C5 03          CMP STOP+1
083F D0 D0          BNE INIZIO
0841 20 B3 F8      JSR CLOPTR
0844 00            BRK

```

; Vincenzo Villa V D

20/4/1983

.END

ERRORS=0000

Da notare che quando la stampante funziona in modo grafico non è richiesto alcun ordine per stampare il buffer ma questo viene fatto automaticamente ogni 40 bytes, cioè ad ogni riga.

La lunghezza della rappresentazione non è limitata da alcun parametro tranne che dalla disponibilità fisica della memoria: bisogna infatti tenere conto che, per es. per rappresentare una figura 280 X 280 Punti (circa 9,5 x 9,5 cm) è necessario definire 78.400 Bit, che occupano circa 11 Kbytes di memoria, cioè circa un terzo della memoria totale dell'AIM 65/40 nella versione standard.

Inoltre vi è la difficoltà pratica di definire tutti questi bit uno a uno, con un lavoro molto lungo e noioso.

A titolo di esempio, in fig. 34 è riportato uno stampato realizzato in modo grafico: si tratta di una sezione qualsiasi di memoria (circa 13 Kbytes) semplicemente letta e inviata alla stampante in modo grafico. La parte in alto dello stampato è costituito da una serie continua di bit tutti a 1: si tratta di dati letti da una memoria

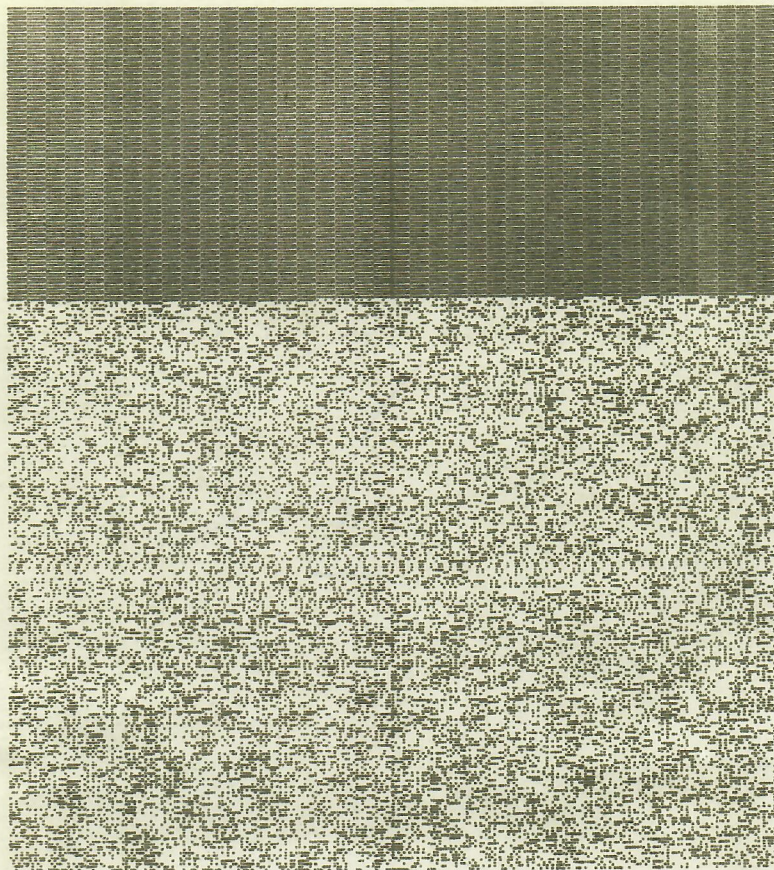


FIG. 34

non esistente e quindi a livello logico 1. La parte inferiore è invece letta dalla ROM del sistema in cui sono contenuti valori "casuali".

Questo listato è stato realizzato con il progr. 22; vediamo di descriverlo nella sua struttura essenziale : dopo aver abilitato la stampante (OPNPTR) viene inviata la sequenza ESC G (cioè SIB e S47) che la portano in modo grafico; viene quindi inviato di seguito il primo byte (dall'indirizzo START) controllando prima che la stampante sia pronta a riceverlo; START viene quindi incrementato per inviare il byte successivo.

Il programma finisce quando START viene a coincidere con STOP e la stampante, non ricevendo più bytes esce automaticamente dal modo grafico.

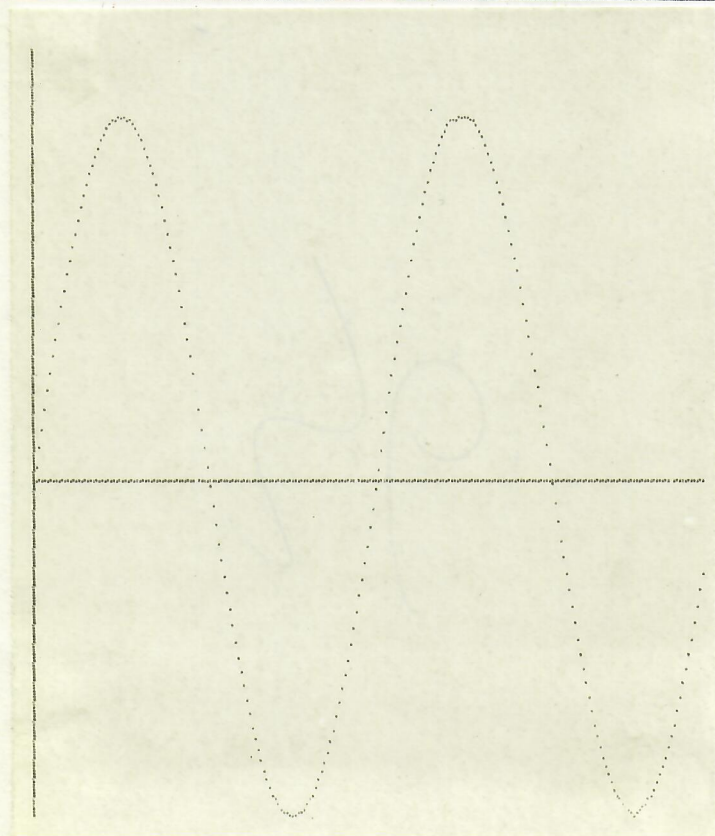


FIG.35 - $y = \sin (x)$

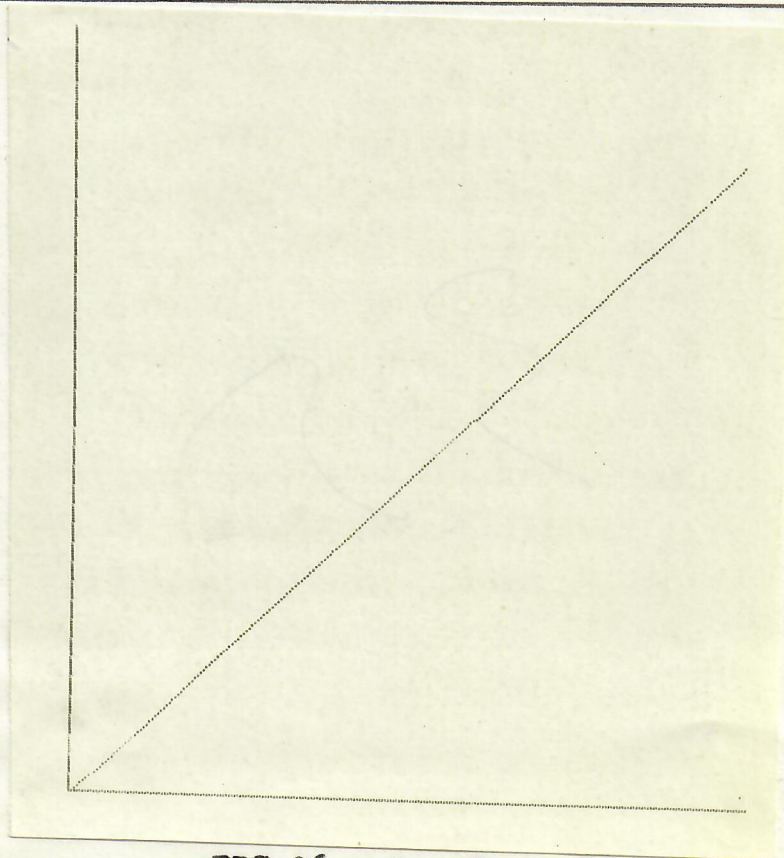


FIG.36 - $y = x$

L'invio vero e proprio è realizzato dal programma 2I perché il richiamo delle routines PUTPTR e WAIT potrebbe rallentare eccessivamente il programma facendo perdere il necessario sincronismo.

Naturalmente se il programmatore deve creare ogni volta il disegno digitalizzandolo manualmente ogni volta, si troverebbe a dover affrontare un lavoro enorme: una soluzione più logica è quella di creare un programma in grado di compiere questo lavoro. Un tipico esempio è il Prog. 23 che è in grado di realizzare, date le ordinate, un grafico ad alta risoluzione $y = f(x)$. Il programma funziona secondo le seguenti direttive: in memoria a partire dalla locazione 6 sono contenuti uno di seguito all'altro i valori delle y della funzione (ad es. nella locazione 6 vi è $f(x_0)$, in 7 $f(x_1)$ in 8 $f(x_2)$ ecc.). i valori contenute in queste locazioni devono essere necessariamente numeri interi positivi tra 0 e 255; quindi sia i valori di x che di y devono essere moltiplicati per una costante che permetta di far coincidere il valore massimo della funzione a $\$FF$, quello minimo a 0, siano essi positivi che negativi.

Il programma a questo punto legge y_0 e, partire dalla locazione START (= $\$4000$), inserirà $y_0 - 1$ bits uguale 0, un bit uguale a 1 nella posizione y_0 ed altri $280 - y_0$ bits a 0: questi 280 bits verranno successivamente stampati graficamente in modo che, su tutta una riga, vi sia un solo punto.

Quindi START è incrementato di 40 ed è fatto lo stesso procedimento per y_1 fino a quando la locazione NUM che contiene il numero delle y da stampare è ridotta a 0 dal decremento realizzato ogni volta che è elaborata una nuova y . Il programma a questo punto richiama la routine appena vista che stampa graficamente il contenuto della memoria a partire da START.

Un miglioramento possibile è quello che prevede la stampa dei due assi cartesiani ottenuto forzando a 1 i primi 280 bits (per l'asse y) ed il primo bit di ogni riga per l'asse x .

Il grafico viene stampato verticalmente e per essere interpretato correttamente deve essere girato di 90° in modo che l'asse x che esce verticalmente sia posto orizzontalmente. Gli assi, se necessario, possono essere posti anche in altre posizioni. Esempi di grafici realizzati con tale programma sono riportati nelle figure 35,36,37 e 38.

ADDR OBJECT SOURCE

; GRAFICO AD ALTA RISOLUZIONE

; NUM=Numero di righe del grafico

; Y sono da porre nelle locazioni da \$6

; (Max 250 righe)

;

START=\$0
 STOP=\$2
 COL=\$4
 NUM=\$5
 GRAFIC=\$900

```

0800 A9 00      LDA #0
0802 85 00      STA START
0804 85 02      STA STOP
0806 A9 40      LDA #$40
0808 85 01      STA START+1
080A 85 03      STA STOP+1
080C A0 00      LDY #0
080E A9 FF      LDA #$FF
0810 91 02      SU    STA (STOP),Y
0812 C8         INY
0813 C0 41      CPY #$41
0815 D0 F9      BNE SU
0817 A2 00      LDX #0
0819 F0 35      BEQ SOMMA
081B E4 05      PRINT CPX NUM
081D F0 41      BEQ END
081F A0 00      LDY #0
0821 B5 06      LDA 6,X
0823 E8         INX
0824 85 04      STA COL
0826 A5 04      INI  LDA COL
0828 78         SEI
0829 E9 07      SBC #$7
082B 90 09      BCC PUNTO
082D 85 04      STA COL
082F A9 00      LDA #0
0831 91 02      STA (STOP),Y
0833 C8         INY
0834 D0 F0      BNE INI
0836 A9 80      PUNTO LDA #$80
0838 C6 04      SU1  DEC COL
083A 30 05      BMI ZERO
083C 4A         LSR A
083D D0 F9      BNE SU1
083F A9 00      ZEROS LDA #0
0841 91 02      ZERO STA (STOP),Y
0843 C8         INY
0844 C0 41      CPY #$41
0846 D0 F7      BNE ZEROS
0848 A0 00      LDY #0
084A A9 80      LDA #$80
084C 11 02      ORA (STOP),Y
084E 91 02      STA (STOP),Y
0850 18      SOMMA CLC
0851 A9 28      LDA #40
0853 65 02      ADC STOP
0855 85 02      STA STOP
0857 A9 00      LDA #0
0859 65 03      ADC STOP+1
085B 85 03      STA STOP+1
085D 4C 1B 08   JMP PRINT
0860 20 00 09   END  JSR GRAFIC
; Vincenzo Villa V D 27/4/1983
END

```

ERRORS=0000

PROG. 23

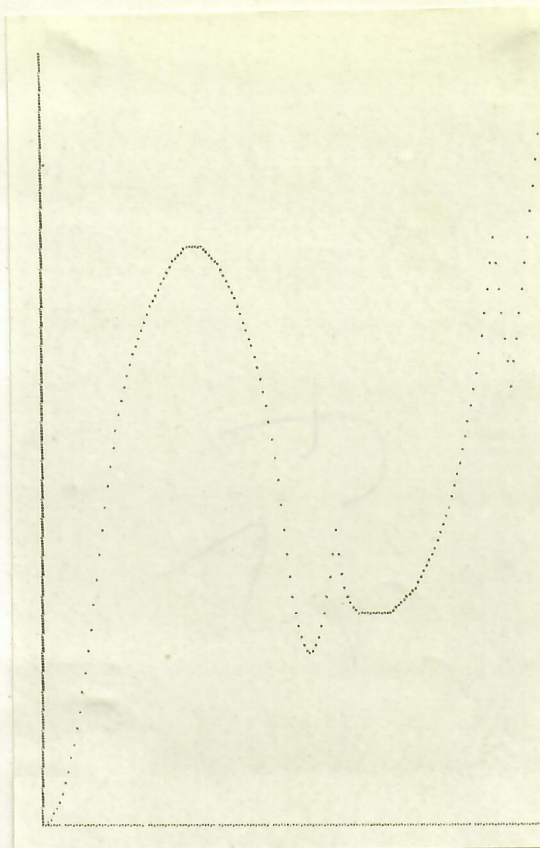


FIG.37

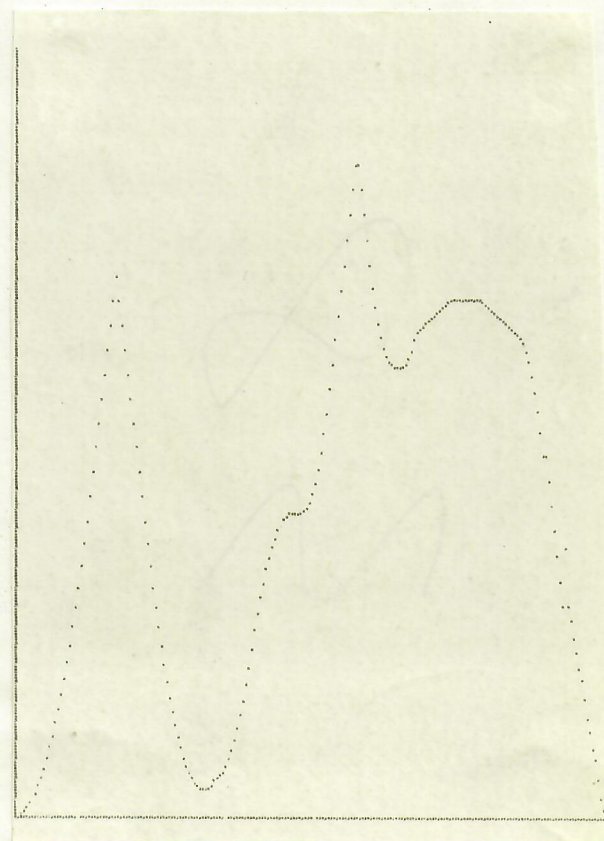


FIG. 38 - Grafici semi-
 casuali realizzati con
 l'interprete BASIC ed
 il PROG. 23

OUTPUT SU DISPLAY

Quanto detto per la stampante può essere trasferito con i dovuti cambiamenti anche al display.

In effetti per scrivere una serie di caratteri è necessario inviare il rispettivo codice ASCII sempre attraverso la porta A dello stesso VIA, segnalarlo con un comando STROBE su un'altra linea e attendere il segnale di risposta su una linea ACK. Questo lavoro è svolto dalla routine PUTDIS dell'I/O ROM.

A differenza di quanto succede per la stampante, quando si deve visualizzare una serie di caratteri sul display non è necessario né abilitare prima la trasmissione né disabilitarla dopo; non è neppure necessario inviare un byte che trasmette l'ordine di stampare il buffer in quanto questo è fatto automaticamente alla ricezione di ogni carattere.

Un programma che usa la routine PUTDIS è già stato visto nel progr. 2 in cui, nel caso di un eventuale resto oltre le cifre sommate, era visualizzata la parola "OVERFLOW"; per ottenerla è stato necessario scrivere in otto bytes i codici ASCII di ciascuna lettera attraverso la direttiva .BYT ; quindi è caricata nell'accumulatore la prima lettera ed è chiamata la routine PUTDIS; quindi sono caricate le altre lettere.

INPUT DA TASTIERA

Un programma per lavorare correttamente deve poter leggere dei dati inseriti dall'operatore attraverso una tastiera alfa-numerica.

Usando l'AIM 65/40 è possibile leggere lo stato della tastiera usando la routine READ (SF22C) che legge lo stato della tastiera attraverso una scansione delle 16 linee di I/O di un apposito VIA ed aspetta fino a quando viene premuto un tasto; a questo punto la routine pone nell'accumulatore il codice ASCII corrispondente e ritorna il controllo al programma principale. Si tratta di una routine piuttosto semplice da usarsi, che non richiede né particolari abilitazioni, né programmi esterni di temporizzazione.

Un programma che gestisce correttamente la tastiera del display è il progr. 24 che legge un codice ASCII dalla tastiera, lo visualizza sul

display e lo memorizza a partire da una particolare locazione di memoria; per assegnare la locazione da cui ^{il} programma memorizza il testo ^{usato} é la routine FROM che, dopo aver inviato al display i caratteri FROM=0000 legge un indirizzo esadecimale di due bytes.

Il programma prima di visualizzare esegue una serie di controllo sul carattere letto: se si tratta del tasto RETURN (= SD)"va a capo" cancellando il display e memorizzando la fine di una riga; se si tratta del tasto DELETE (=S7F) viene cancellata l'ultima lettera editata, permettendo la correzione degli errori; infine il tasto ESCAPE indica che il testo é finito e il programma é bloccato.

Questo programma, con poche modifiche, permette anche di simulare una macchina da scrivere normale, con l'aggiunta prestazioni non standard, quale la cancellazione delle lettere errate e la memorizzazione dei testi.

PAGE 0001

ADDR OBJECT SOURCE

INPUT DA TASTIERA

START=#0
FROM=\$AE39
READ=\$F22C
PUTDIS=\$F7D3

0000	20	39	AE		JSR FROM
0003	86	00			STX START
0005	84	01			STY START+1
0007	A9	01			LDA #1
0009	20	D3	F7		JSR PUTDIS
000C	A0	00		SU	LDY #0
000E	20	2C	F2	QUI	JSR READ
0011	C9	0D			CMP #D =RETURN
0013	F0	15			BEQ RTN
0015	C9	7F			CMP #7F =DELETE
0017	F0	1B			BEQ DEL
0019	C9	1B			CMP #1B =ESC
001B	F0	20			BEQ ESC
001D	20	D3	F7		JSR PUTDIS
0020	91	00		PRINT	STA (START),Y
0022	C8				INY
0023	D0	E9			BNE QUI
0025	E6	01			INC START+1
0027	4C	0C	08		JMP SU
002A	AA			RTN	TAX
002B	A9	01			LDA #1
002D	20	D3	F7		JSR PUTDIS
0030	8A				TXA
0031	4C	20	08		JMP PRINT
0034	A9	08		DEL	LDA #8
0036	20	D3	F7		JSR PUTDIS
0039	88				DEY
003A	4C	0E	08		JMP QUI
003D	00			ESC	BRK

Vincenzo Villa V D 25/5/1983
END

ERRORS=0000

L' INTEGRATO 6522 VIA

L'integrato 6522 Versatile Interface Adapter é un circuito abbastanza complesso che ha il suo uso principale nel controllo dell'Input/Output: in effetti, é attraverso questo chip o un altro analogo (ad. es. il 6520 PIO) che avvengono le comunicazioni tra il 6502 ed il mondo esterno. Nella scheda AIM 65/40 esistono ben 3 di questi integrati di cui due servono per comunicare con la stampante, il display e la tastiera come si é già accennato (sono il System VIA e il Keyboard VIA) e l'altro é a disposizione dell'utente (lo User VIA); a quest'ultimo integrato é dedicata la massima attenzione dato che attraverso esso avvengono tutte le comunicazioni da o verso i convertitori Digitali/Analogici e Analogici/Digitali e gli altri dispositivi controllati dal computer oltre che permettere la temporizzazione di determinati fenomeni.

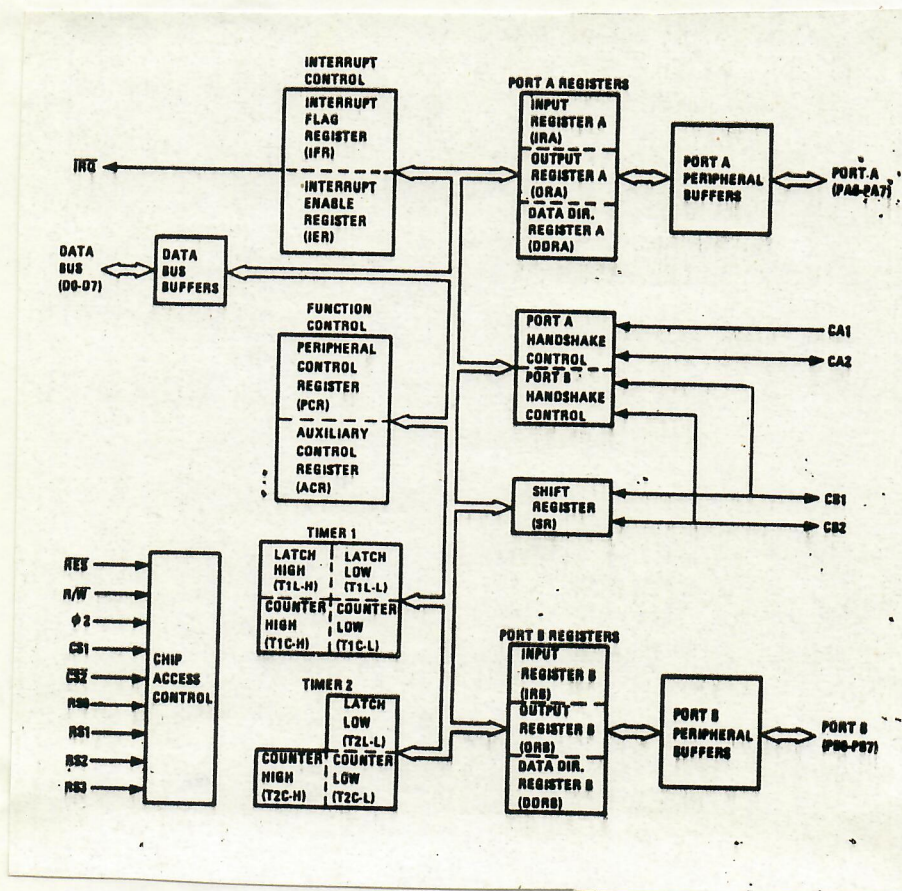


Fig. 39

In fig. 39 é riportato lo schema completo di un VIA: a destra vi sono le linee dirette verso l'esterno (due porte bidirezionali di 8 bit ciascuna e 4 linee di controllo), a sinistra le linee collegate al computer (in particolare si nota il data bus e parte del control bus, i collegamenti all'address bus sono sottintesi). In funzionamento corretto dell'integrato é garantito da 4 registri che servono per indicare la direzione in cui i dati si muovono (in input o in output), per controllare il funzionamento delle 4

linee di handshake, dei due timer e dello shift register.

Inoltre un particolare registro abilita o disabilita le varie sezioni dell'integrato a generare delle interrupts ed un altro le segnala.

Prima di passare alla descrizione di ogni singolo blocco é necessario ricordare che ogni registro é gestito come una normale locazione di memoria, dato che nel 6502 non esistono particolari istruzioni di I/O, e come tali possono, con poche eccezioni, essere letti, scritti, fatti scorrere, incrementati e decrementati. In TAB.5 é riportato l'elenco

completo di tutti i registri dello USER VIA con i relativi indirizzi e la label che gli individua. Il cuore del sistema può essere individuato nelle due porte parallele di I/O (porta A e porta B, abbreviate PA e PB): esse sono praticamente uguali (le differenze si riducono al fatto che PB é in grado di erogare correnti

AIM 65/40 SYSTEM I/O ADDRESSES				
Address	Label	Bytes	Reset	Parameter
FF80	PRIRTY	1	00	IRQ Priority Mask
FFA0	UORB	1	FF	Port B Data Register
FFA1	UORA	1	FF	Port A Data Register
FFA2	UDRB	1	FF	Port B Data Direction Register
FFA3	UDRA	1	00	Port A Data Direction Register
FFA4	UT1CL	1	-	Timer 1 Latch/Counter Low
FFA5	UT1CH	1	-	Timer 1 Latch/Counter High
FFA6	UT1LL	1	-	Timer 1 Latch Low
FFA7	UT1LH	1	-	Timer 1 Latch High
FFA8	UT2CL	1	-	Timer 2 Latch/Counter Low
FFA9	UT2CH	1	-	Timer 2 Counter High
FFAA	USR	1	FF	Shift Register (SR)
FFAB	UACR	1	00	Auxiliary Control Register (ACR)
FFAC	UPCR	1	00	Peripheral Control Register (PCR)
FFAD	UIFR	1	00	Interrupt Flag Register (IFR)
FFAE	UIER	1	80	Interrupt Enable Register (IER)
FFAF	UORAX	1	FF	Port A Data Register (w/o Handshake)

TAB. 4 - Indirizzi dei registri dello USER VIA

più intense e quindi é più adatta all'output) e possono fungere sia come linee di ingresso che di uscita. La direzione dei dati é decisa, linea per linea, dal contenuto dei due registri DRA (per la porta A) e DRB (per PB): mettendo uno 0 in un dato bit, la linea corrispondente é in input, mettendo un 1, la linea é in output. All'interno di una singola porta é quindi possibile dedicare alcune linee all'uscita e altre all'ingresso, anche se generalmente é opportuno dedicare tutta una porta al transito nella stessa direzione.

Per leggere il dato presente su una porta é sufficiente metterla in input (caricando \$00 nella locazione \$FFA2 o \$FFA3) e leggere la locazione indicante PA o PB. Per trasmettere é necessario caricare \$FF in DRA o DRB per mettere la porta in output e scrivere il dato in PA o PB. E' inutile dire che non ha senso leggere una locazione in output o scrivere quando é in input.

Se la periferica é più complessa, non é sufficiente trasferire semplicemente il dato: nel caso di input il processore deve sapere se il dato é disponibile sulla porta (ad esempio se un tasto di una tastiera

é premuto); Nel caso di output, il processore deve sapere se la periferica é pronta a ricevere un nuovo dato (ad esempio se una stampante ha finito di stampare il carattere precedente).

Normalmente le periferiche hanno una linea per indicare questa disponibilità: una transizione su di essa indica la disponibilità di un dato o la fine della precedente elaborazione. La sezione di I/O deve a questo punto poter riconoscere la transizione e comunicarla al microprocessore. Usando il 6522 VIA questa linea di stato deve essere collegata alle linee CA1 o CB1 (rispettivamente per PA e PB); é poi necessario determinare se la transizione avverrà dal livello alto a quello basso o viceversa. Per far questo esistono due flags all'interno del PCR (Peripheral Control Register, il registro che controlla le quattro linee di handshake), precisamente i bit 0 per CA1 e 4 per CB1, che, se posti a 1, riconoscono una transizione da 0 a 1 (rising edge), se posti a 0, riconoscono una transizione da 1 a 0 (falling edge).

Per verificare se la transizione é avvenuta viene controllato il bit I (CA1) o 4 (CB1) dell' IFR (Interrupt Flag Register): se il bit é a 1 la transizione é avvenuta.

Le porte A e B possono avere sia nella fase di input che di output le uscite bufferate, cioè capaci di memorizzare lo stato presente in un certo momento attraverso dei latches. Durante l'output i latches sono sempre abilitati; nel caso di input, se necessario, possono essere abilitati i latches ponendo a 1 i bits 0 (per PA) o 1 (per PB) dell'ACR (AUXILIARY ControlRegister): il dato memorizzato é quello presente sulla porta alla transizione utile su CA1 o CB1.

Le periferiche possono inoltre richiedere l'indicazione dell'avvenuto trasferimento del dato dalla memoria alla porta oppure l'indicazione che la porta é disponibile per un nuovo invio.

Per esempio un dispositivo come il convertitore D/A richiede un impulso di START con cui si indica che la conversione può cominciare.

Un segnale di output può essere anche richiesto per accendere o spegnere un dispositivo, per abilitare un display, o indicare un modo di funzionamento.

Questi segnali sono forniti dal VIA attraverso le due linee CA2 e CB2, secondo le indicazioni fornite dai 3 + 3 bits del PCR; di seguito sono riportate le tre configurazioni di tali bits riferendoci a CB2 (CA2 usa

i bits 3,2, e 1).

Il bit 7 é sempre posto a 1 dato che sia CBI che CB2 sono usati in output; usando queste due linee come input si avrebbero due linee praticamente uguali a CAI e CBI.

Bit 7 6 5

1 0 0 CB2 va alto nel momento in cui si scrive in PB e rimane tale fino ad una transizione su CBI(fig. 40a).

1 0 0 1 CB2 va basso scrivendo in PB e ritorna alto dopo un colpo di clock (fig. 40b)

1 1 X CB2 assume il livello del bit 5 indipendentemente dal funzionamento della porta. Questa configurazione é usata per accendere o spegnere la periferica o per generare impulsi molto lunghi.

Un importante registro del VIA é l'IER (Interrupt Enable Register) che ha il compito di abilitare le varie sezioni del VIA a generare un interrupt verso la CPU: per adesso basti dire che un interrupt é un segnale asincrono al microprocessore, generato da un circuito esterno per richiedere l'intervento del processore; esso blocca il normale svolgersi del programma e fa partire un nuovo programma che, dopo aver individuato la richiesta, la soddisfa. Per una trattazione specifica si può vedere più avanti.

L'IER, a differenza degli altri registri, non é usato come una normale locazione di memoria ma la scrittura di un certo bit é fatta attraverso il bit 7: se esso é a 1, pone a 1 tutti i bit a 1, se é a 0, pone a 0 tutti i bit a 1. Un esempio chiarisce bene questo modo di procedere: volendo porre a 1 i bit 2,3 e 5 non bisogna scrivere 00101100 ma 1 0101100, volendo azzerare gli stessi bit é necessario scrivere 0 0101100; in ambedue i casi gli altri 4 bit rimangono invariati.

Un'altra caratteristica del VIA é nell'avere due timer/contatori program_mabili a 16 bit (T1 e T2) usati per generare ond_e quadre a una determi_nata frequenza, per contare impulsi o determinare intervalli di tempo. Il timer 2 (T2) é più semplice e occupa due locazioni c_nsecutive; esso é determinato nel suo funzionamento dal bit 5 dell'ACR: se é uno T2 conta gli impulsi sul bit 6 di PB, se é 0 decrementa il suo contenuto ad ogni colpo di clock fino a 0. In ambedue i casi pone a 1 il bit 5 dell'IFR e, se abilitato, richiede un interrupt alla CPU.

Il timer 1 è controllato dai bit 6 e 7 dell'ACR: se il bit 6 è a 0, il timer genera un singolo intervallo di tempo (come T2); se è a 1, genera una serie continua di intervalli con una frequenza determinata dal numero in esso contenuto; il bit 7, se posto a 1 genera sulla linea 7 di PB un'onda quadra di frequenza prefissata.

Il timer 1 occupa 4 bytes di memoria: i primi due sono il contatore vero e proprio, gli altri due (TI-L) contengono il valore che è trasferito nel timer quando questi giunge a 0.

Anche TI quando arriva a 0 può generare un interrupt, ponendo a 1 il bit 6 dell'IFR.

Il via possiede inoltre uno shift register capace di convertire i dati da seriale a parallelo o viceversa.

ESEMPI DI INPUT E OUTPUT

Vediamo ora un semplice esempio dell'uso della porta parallela usata per inviare dei dati ad un convertitore D/A: esso è in grado di generare un'onda a denti di sega con una risoluzione molto buona.

Lo schema è molto semplice, bastando caricare nel PCR il numero IOIXXXX per determinare su CB2 una transizione ogni volta che P B è scritta: questa transizione dà al convertitore l'ordine di iniziare la conversione che impiega un tempo molto breve. PB viene semplicemente incrementata di 1 all'interno di un loop senza uscita: quando PB arriva a 255 un ulteriore incremento la porta a 0.

Il loop impiega un tempo di 9 μ S, generando un'onda dalla frequenza di circa 440 Hz (sono infatti necessari 256 incrementi per tornare al punto di partenza); per bloccare il

PAGE 0001

ADDR OBJECT SOURCE

Generatore di onde a denti di sega
(Con convertitore D/A)
Frequenza massima = 440 Hz
(Si può rallentare inserendo una serie di istruzioni NOP (2 microsecondi))

PB=\$FFA0
PBDIR=\$FFA2
CR=\$FFAC

```
0000 A9 A0          LDA #$A0 (=0x10100)
0001
0002 8D AC FF        STA CR
0003 A9 FF          LDA #$FF
0004 8D A2 FF        STA PBDIR
0005 EE A0 FF        INC PB
0006 NOP
0007 4C 0A 08        JMP QUI
0008
0009 4C 0A 08        JMP QUI
0010
0011 4C 0A 08        JMP QUI
0012
0013 4C 0A 08        JMP QUI
0014
0015 4C 0A 08        JMP QUI
0016
0017 4C 0A 08        JMP QUI
0018
0019 4C 0A 08        JMP QUI
0020
0021 4C 0A 08        JMP QUI
0022
0023 4C 0A 08        JMP QUI
0024
0025 4C 0A 08        JMP QUI
0026
0027 4C 0A 08        JMP QUI
0028
0029 4C 0A 08        JMP QUI
0030
0031 4C 0A 08        JMP QUI
0032
0033 4C 0A 08        JMP QUI
0034
0035 4C 0A 08        JMP QUI
0036
0037 4C 0A 08        JMP QUI
0038
0039 4C 0A 08        JMP QUI
0040
0041 4C 0A 08        JMP QUI
0042
0043 4C 0A 08        JMP QUI
0044
0045 4C 0A 08        JMP QUI
0046
0047 4C 0A 08        JMP QUI
0048
0049 4C 0A 08        JMP QUI
0050
0051 4C 0A 08        JMP QUI
0052
0053 4C 0A 08        JMP QUI
0054
0055 4C 0A 08        JMP QUI
0056
0057 4C 0A 08        JMP QUI
0058
0059 4C 0A 08        JMP QUI
0060
0061 4C 0A 08        JMP QUI
0062
0063 4C 0A 08        JMP QUI
0064
0065 4C 0A 08        JMP QUI
0066
0067 4C 0A 08        JMP QUI
0068
0069 4C 0A 08        JMP QUI
0070
0071 4C 0A 08        JMP QUI
0072
0073 4C 0A 08        JMP QUI
0074
0075 4C 0A 08        JMP QUI
0076
0077 4C 0A 08        JMP QUI
0078
0079 4C 0A 08        JMP QUI
0080
0081 4C 0A 08        JMP QUI
0082
0083 4C 0A 08        JMP QUI
0084
0085 4C 0A 08        JMP QUI
0086
0087 4C 0A 08        JMP QUI
0088
0089 4C 0A 08        JMP QUI
0090
0091 4C 0A 08        JMP QUI
0092
0093 4C 0A 08        JMP QUI
0094
0095 4C 0A 08        JMP QUI
0096
0097 4C 0A 08        JMP QUI
0098
0099 4C 0A 08        JMP QUI
0100
0101 4C 0A 08        JMP QUI
0102
0103 4C 0A 08        JMP QUI
0104
0105 4C 0A 08        JMP QUI
0106
0107 4C 0A 08        JMP QUI
0108
0109 4C 0A 08        JMP QUI
0110
0111 4C 0A 08        JMP QUI
0112
0113 4C 0A 08        JMP QUI
0114
0115 4C 0A 08        JMP QUI
0116
0117 4C 0A 08        JMP QUI
0118
0119 4C 0A 08        JMP QUI
0120
0121 4C 0A 08        JMP QUI
0122
0123 4C 0A 08        JMP QUI
0124
0125 4C 0A 08        JMP QUI
0126
0127 4C 0A 08        JMP QUI
0128
0129 4C 0A 08        JMP QUI
0130
0131 4C 0A 08        JMP QUI
0132
0133 4C 0A 08        JMP QUI
0134
0135 4C 0A 08        JMP QUI
0136
0137 4C 0A 08        JMP QUI
0138
0139 4C 0A 08        JMP QUI
0140
0141 4C 0A 08        JMP QUI
0142
0143 4C 0A 08        JMP QUI
0144
0145 4C 0A 08        JMP QUI
0146
0147 4C 0A 08        JMP QUI
0148
0149 4C 0A 08        JMP QUI
0150
0151 4C 0A 08        JMP QUI
0152
0153 4C 0A 08        JMP QUI
0154
0155 4C 0A 08        JMP QUI
0156
0157 4C 0A 08        JMP QUI
0158
0159 4C 0A 08        JMP QUI
0160
0161 4C 0A 08        JMP QUI
0162
0163 4C 0A 08        JMP QUI
0164
0165 4C 0A 08        JMP QUI
0166
0167 4C 0A 08        JMP QUI
0168
0169 4C 0A 08        JMP QUI
0170
0171 4C 0A 08        JMP QUI
0172
0173 4C 0A 08        JMP QUI
0174
0175 4C 0A 08        JMP QUI
0176
0177 4C 0A 08        JMP QUI
0178
0179 4C 0A 08        JMP QUI
0180
0181 4C 0A 08        JMP QUI
0182
0183 4C 0A 08        JMP QUI
0184
0185 4C 0A 08        JMP QUI
0186
0187 4C 0A 08        JMP QUI
0188
0189 4C 0A 08        JMP QUI
0190
0191 4C 0A 08        JMP QUI
0192
0193 4C 0A 08        JMP QUI
0194
0195 4C 0A 08        JMP QUI
0196
0197 4C 0A 08        JMP QUI
0198
0199 4C 0A 08        JMP QUI
0200
0201 4C 0A 08        JMP QUI
0202
0203 4C 0A 08        JMP QUI
0204
0205 4C 0A 08        JMP QUI
0206
0207 4C 0A 08        JMP QUI
0208
0209 4C 0A 08        JMP QUI
0210
0211 4C 0A 08        JMP QUI
0212
0213 4C 0A 08        JMP QUI
0214
0215 4C 0A 08        JMP QUI
0216
0217 4C 0A 08        JMP QUI
0218
0219 4C 0A 08        JMP QUI
0220
0221 4C 0A 08        JMP QUI
0222
0223 4C 0A 08        JMP QUI
0224
0225 4C 0A 08        JMP QUI
0226
0227 4C 0A 08        JMP QUI
0228
0229 4C 0A 08        JMP QUI
0230
0231 4C 0A 08        JMP QUI
0232
0233 4C 0A 08        JMP QUI
0234
0235 4C 0A 08        JMP QUI
0236
0237 4C 0A 08        JMP QUI
0238
0239 4C 0A 08        JMP QUI
0240
0241 4C 0A 08        JMP QUI
0242
0243 4C 0A 08        JMP QUI
0244
0245 4C 0A 08        JMP QUI
0246
0247 4C 0A 08        JMP QUI
0248
0249 4C 0A 08        JMP QUI
0250
0251 4C 0A 08        JMP QUI
0252
0253 4C 0A 08        JMP QUI
0254
0255 4C 0A 08        JMP QUI
0256
0257 4C 0A 08        JMP QUI
0258
0259 4C 0A 08        JMP QUI
0260
0261 4C 0A 08        JMP QUI
0262
0263 4C 0A 08        JMP QUI
0264
0265 4C 0A 08        JMP QUI
0266
0267 4C 0A 08        JMP QUI
0268
0269 4C 0A 08        JMP QUI
0270
0271 4C 0A 08        JMP QUI
0272
0273 4C 0A 08        JMP QUI
0274
0275 4C 0A 08        JMP QUI
0276
0277 4C 0A 08        JMP QUI
0278
0279 4C 0A 08        JMP QUI
0280
0281 4C 0A 08        JMP QUI
0282
0283 4C 0A 08        JMP QUI
0284
0285 4C 0A 08        JMP QUI
0286
0287 4C 0A 08        JMP QUI
0288
0289 4C 0A 08        JMP QUI
0290
0291 4C 0A 08        JMP QUI
0292
0293 4C 0A 08        JMP QUI
0294
0295 4C 0A 08        JMP QUI
0296
0297 4C 0A 08        JMP QUI
0298
0299 4C 0A 08        JMP QUI
0300
0301 4C 0A 08        JMP QUI
0302
0303 4C 0A 08        JMP QUI
0304
0305 4C 0A 08        JMP QUI
0306
0307 4C 0A 08        JMP QUI
0308
0309 4C 0A 08        JMP QUI
0310
0311 4C 0A 08        JMP QUI
0312
0313 4C 0A 08        JMP QUI
0314
0315 4C 0A 08        JMP QUI
0316
0317 4C 0A 08        JMP QUI
0318
0319 4C 0A 08        JMP QUI
0320
0321 4C 0A 08        JMP QUI
0322
0323 4C 0A 08        JMP QUI
0324
0325 4C 0A 08        JMP QUI
0326
0327 4C 0A 08        JMP QUI
0328
0329 4C 0A 08        JMP QUI
0330
0331 4C 0A 08        JMP QUI
0332
0333 4C 0A 08        JMP QUI
0334
0335 4C 0A 08        JMP QUI
0336
0337 4C 0A 08        JMP QUI
0338
0339 4C 0A 08        JMP QUI
0340
0341 4C 0A 08        JMP QUI
0342
0343 4C 0A 08        JMP QUI
0344
0345 4C 0A 08        JMP QUI
0346
0347 4C 0A 08        JMP QUI
0348
0349 4C 0A 08        JMP QUI
0350
0351 4C 0A 08        JMP QUI
0352
0353 4C 0A 08        JMP QUI
0354
0355 4C 0A 08        JMP QUI
0356
0357 4C 0A 08        JMP QUI
0358
0359 4C 0A 08        JMP QUI
0360
0361 4C 0A 08        JMP QUI
0362
0363 4C 0A 08        JMP QUI
0364
0365 4C 0A 08        JMP QUI
0366
0367 4C 0A 08        JMP QUI
0368
0369 4C 0A 08        JMP QUI
0370
0371 4C 0A 08        JMP QUI
0372
0373 4C 0A 08        JMP QUI
0374
0375 4C 0A 08        JMP QUI
0376
0377 4C 0A 08        JMP QUI
0378
0379 4C 0A 08        JMP QUI
0380
0381 4C 0A 08        JMP QUI
0382
0383 4C 0A 08        JMP QUI
0384
0385 4C 0A 08        JMP QUI
0386
0387 4C 0A 08        JMP QUI
0388
0389 4C 0A 08        JMP QUI
0390
0391 4C 0A 08        JMP QUI
0392
0393 4C 0A 08        JMP QUI
0394
0395 4C 0A 08        JMP QUI
0396
0397 4C 0A 08        JMP QUI
0398
0399 4C 0A 08        JMP QUI
0400
0401 4C 0A 08        JMP QUI
0402
0403 4C 0A 08        JMP QUI
0404
0405 4C 0A 08        JMP QUI
0406
0407 4C 0A 08        JMP QUI
0408
0409 4C 0A 08        JMP QUI
0410
0411 4C 0A 08        JMP QUI
0412
0413 4C 0A 08        JMP QUI
0414
0415 4C 0A 08        JMP QUI
0416
0417 4C 0A 08        JMP QUI
0418
0419 4C 0A 08        JMP QUI
0420
0421 4C 0A 08        JMP QUI
0422
0423 4C 0A 08        JMP QUI
0424
0425 4C 0A 08        JMP QUI
0426
0427 4C 0A 08        JMP QUI
0428
0429 4C 0A 08        JMP QUI
0430
0431 4C 0A 08        JMP QUI
0432
0433 4C 0A 08        JMP QUI
0434
0435 4C 0A 08        JMP QUI
0436
0437 4C 0A 08        JMP QUI
0438
0439 4C 0A 08        JMP QUI
0440
0441 4C 0A 08        JMP QUI
0442
0443 4C 0A 08        JMP QUI
0444
0445 4C 0A 08        JMP QUI
0446
0447 4C 0A 08        JMP QUI
0448
0449 4C 0A 08        JMP QUI
0450
0451 4C 0A 08        JMP QUI
0452
0453 4C 0A 08        JMP QUI
0454
0455 4C 0A 08        JMP QUI
0456
0457 4C 0A 08        JMP QUI
0458
0459 4C 0A 08        JMP QUI
0460
0461 4C 0A 08        JMP QUI
0462
0463 4C 0A 08        JMP QUI
0464
0465 4C 0A 08        JMP QUI
0466
0467 4C 0A 08        JMP QUI
0468
0469 4C 0A 08        JMP QUI
0470
0471 4C 0A 08        JMP QUI
0472
0473 4C 0A 08        JMP QUI
0474
0475 4C 0A 08        JMP QUI
0476
0477 4C 0A 08        JMP QUI
0478
0479 4C 0A 08        JMP QUI
0480
0481 4C 0A 08        JMP QUI
0482
0483 4C 0A 08        JMP QUI
0484
0485 4C 0A 08        JMP QUI
0486
0487 4C 0A 08        JMP QUI
0488
0489 4C 0A 08        JMP QUI
0490
0491 4C 0A 08        JMP QUI
0492
0493 4C 0A 08        JMP QUI
0494
0495 4C 0A 08        JMP QUI
0496
0497 4C 0A 08        JMP QUI
0498
0499 4C 0A 08        JMP QUI
0500
0501 4C 0A 08        JMP QUI
0502
0503 4C 0A 08        JMP QUI
0504
0505 4C 0A 08        JMP QUI
0506
0507 4C 0A 08        JMP QUI
0508
0509 4C 0A 08        JMP QUI
0510
0511 4C 0A 08        JMP QUI
0512
0513 4C 0A 08        JMP QUI
0514
0515 4C 0A 08        JMP QUI
0516
0517 4C 0A 08        JMP QUI
0518
0519 4C 0A 08        JMP QUI
0520
0521 4C 0A 08        JMP QUI
0522
0523 4C 0A 08        JMP QUI
0524
0525 4C 0A 08        JMP QUI
0526
0527 4C 0A 08        JMP QUI
0528
0529 4C 0A 08        JMP QUI
0530
0531 4C 0A 08        JMP QUI
0532
0533 4C 0A 08        JMP QUI
0534
0535 4C 0A 08        JMP QUI
0536
0537 4C 0A 08        JMP QUI
0538
0539 4C 0A 08        JMP QUI
0540
0541 4C 0A 08        JMP QUI
0542
0543 4C 0A 08        JMP QUI
0544
0545 4C 0A 08        JMP QUI
0546
0547 4C 0A 08        JMP QUI
0548
0549 4C 0A 08        JMP QUI
0550
0551 4C 0A 08        JMP QUI
0552
0553 4C 0A 08        JMP QUI
0554
0555 4C 0A 08        JMP QUI
0556
0557 4C 0A 08        JMP QUI
0558
0559 4C 0A 08        JMP QUI
0560
0561 4C 0A 08        JMP QUI
0562
0563 4C 0A 08        JMP QUI
0564
0565 4C 0A 08        JMP QUI
0566
0567 4C 0A 08        JMP QUI
0568
0569 4C 0A 08        JMP QUI
0570
0571 4C 0A 08        JMP QUI
0572
0573 4C 0A 08        JMP QUI
0574
0575 4C 0A 08        JMP QUI
0576
0577 4C 0A 08        JMP QUI
0578
0579 4C 0A 08        JMP QUI
0580
0581 4C 0A 08        JMP QUI
0582
0583 4C 0A 08        JMP QUI
0584
0585 4C 0A 08        JMP QUI
0586
0587 4C 0A 08        JMP QUI
0588
0589 4C 0A 08        JMP QUI
0590
0591 4C 0A 08        JMP QUI
0592
0593 4C 0A 08        JMP QUI
0594
0595 4C 0A 08        JMP QUI
0596
0597 4C 0A 08        JMP QUI
0598
0599 4C 0A 08        JMP QUI
0600
0601 4C 0A 08        JMP QUI
0602
0603 4C 0A 08        JMP QUI
0604
0605 4C 0A 08        JMP QUI
0606
0607 4C 0A 08        JMP QUI
0608
0609 4C 0A 08        JMP QUI
0610
0611 4C 0A 08        JMP QUI
0612
0613 4C 0A 08        JMP QUI
0614
0615 4C 0A 08        JMP QUI
0616
0617 4C 0A 08        JMP QUI
0618
0619 4C 0A 08        JMP QUI
0620
0621 4C 0A 08        JMP QUI
0622
0623 4C 0A 08        JMP QUI
0624
0625 4C 0A 08        JMP QUI
0626
0627 4C 0A 08        JMP QUI
0628
0629 4C 0A 08        JMP QUI
0630
0631 4C 0A 08        JMP QUI
0632
0633 4C 0A 08        JMP QUI
0634
0635 4C 0A 08        JMP QUI
0636
0637 4C 0A 08        JMP QUI
0638
0639 4C 0A 08        JMP QUI
0640
0641 4C 0A 08        JMP QUI
0642
0643 4C 0A 08        JMP QUI
0644
0645 4C 0A 08        JMP QUI
0646
0647 4C 0A 08        JMP QUI
0648
0649 4C 0A 08        JMP QUI
0650
0651 4C 0A 08        JMP QUI
0652
0653 4C 0A 08        JMP QUI
0654
0655 4C 0A 08        JMP QUI
0656
0657 4C 0A 08        JMP QUI
0658
0659 4C 0A 08        JMP QUI
0660
0661 4C 0A 08        JMP QUI
0662
0663 4C 0A 08        JMP QUI
0664
0665 4C 0A 08        JMP QUI
0666
0667 4C 0A 08        JMP QUI
0668
0669 4C 0A 08        JMP QUI
0670
0671 4C 0A 08        JMP QUI
0672
0673 4C 0A 08        JMP QUI
0674
0675 4C 0A 08        JMP QUI
0676
0677 4C 0A 08        JMP QUI
0678
0679 4C 0A 08        JMP QUI
0680
0681 4C 0A 08        JMP QUI
0682
0683 4C 0A 08        JMP QUI
0684
0685 4C 0A 08        JMP QUI
0686
0687 4C 0A 08        JMP QUI
0688
0689 4C 0A 08        JMP QUI
0690
0691 4C 0A 08        JMP QUI
0692
0693 4C 0A 08        JMP QUI
0694
0695 4C 0A 08        JMP QUI
0696
0697 4C 0A 08        JMP QUI
0698
0699 4C 0A 08        JMP QUI
0700
0701 4C 0A 08        JMP QUI
0702
0703 4C 0A 08        JMP QUI
0704
0705 4C 0A 08        JMP QUI
0706
0707 4C 0A 08        JMP QUI
0708
0709 4C 0A 08        JMP QUI
0710
0711 4C 0A 08        JMP QUI
0712
0713 4C 0A 08        JMP QUI
0714
0715 4C 0A 08        JMP QUI
0716
0717 4C 0A 08        JMP QUI
0718
0719 4C 0A 08        JMP QUI
0720
0721 4C 0A 08        JMP QUI
0722
0723 4C 0A 08        JMP QUI
0724
0725 4C 0A 08        JMP QUI
0726
0727 4C 0A 08        JMP QUI
0728
0729 4C 0A 08        JMP QUI
0730
0731 4C 0A 08        JMP QUI
0732
0733 4C 0A 08        JMP QUI
0734
0735 4C 0A 08        JMP QUI
0736
0737 4C 0A 08        JMP QUI
0738
0739 4C 0A 08        JMP QUI
0740
0741 4C 0A 08        JMP QUI
0742
0743 4C 0A 08        JMP QUI
0744
0745 4C 0A 08        JMP QUI
0746
0747 4C 0A 08        JMP QUI
0748
0749 4C 0A 08        JMP QUI
0750
0751 4C 0A 08        JMP QUI
0752
0753 4C 0A 08        JMP QUI
0754
0755 4C 0A 08        JMP QUI
0756
0757 4C 0A 08        JMP QUI
0758
0759 4C 0A 08        JMP QUI
0760
0761 4C 0A 08        JMP QUI
0762
0763 4C 0A 08        JMP QUI
0764
0765 4C 0A 08        JMP QUI
0766
0767 4C 0A 08        JMP QUI
0768
0769 4C 0A 08        JMP QUI
0770
0771 4C 0A 08        JMP QUI
0772
0773 4C 0A 08        JMP QUI
0774
0775 4C 0A 08        JMP QUI
0776
0777 4C 0A 08        JMP QUI
0778
0779 4C 0A 08        JMP QUI
0780
0781 4C 0A 08        JMP QUI
0782
0783 4C 0A 08        JMP QUI
0784
0785 4C 0A 08        JMP QUI
0786
0787 4C 0A 08        JMP QUI
0788
0789 4C 0A 08        JMP QUI
0790
0791 4C 0A 08        JMP QUI
0792
0793 4C 0A 08        JMP QUI
0794
0795 4C 0A 08        JMP QUI
0796
0797 4C 0A 08        JMP QUI
0798
0799 4C 0A 08        JMP QUI
0800
0801 4C 0A 08        JMP QUI
0802
0803 4C 0A 08        JMP QUI
0804
0805 4C 0A 08        JMP QUI
0806
0807 4C 0A 08        JMP QUI
0808
0809 4C 0A 08        JMP QUI
0810
0811 4C 0A 08        JMP QUI
0812
0813 4C 0A 08        JMP QUI
0814
0815 4C 0A 08        JMP QUI
0816
0817 4C 0A 08        JMP QUI
0818
0819 4C 0A 08        JMP QUI
0820
0821 4C 0A 08        JMP QUI
0822
0823 4C 0A 08        JMP QUI
0824
0825 4C 0A 08        JMP QUI
0826
0827 4C 0A 08        JMP QUI
0828
0829 4C 0A 08        JMP QUI
0830
0831 4C 0A 08        JMP QUI
0832
0833 4C 0A 08        JMP QUI
0834
0835 4C 0A 08        JMP QUI
0836
0837 4C 0A 08        JMP QUI
0838
0839 4C 0A 08        JMP QUI
0840
0841 4C 0A 08        JMP QUI
0842
0843 4C 0A 08        JMP QUI
0844
0845 4C 0A 08        JMP QUI
0846
0847 4C 0A 08        JMP QUI
0848
0849 4C 0A 08        JMP QUI
0850
0851 4C 0A 08        JMP QUI
0852
0853 4C 0A 08        JMP QUI
0854
0855 4C 0A 08        JMP QUI
0856
0857 4C 0A 08        JMP QUI
0858
0859 4C 0A 08        JMP QUI
0860
0861 4C 0A 08        JMP QUI
0862
0863 4C 0A 08        JMP QUI
0864
0865 4C 0A 08        JMP QUI
0866
0867 4C 0A 08        JMP QUI
0868
0869 4C 0A 08        JMP QUI
0870
0871 4C 0A 08        JMP QUI
0872
0873 4C 0A 08        JMP QUI
0874
0875 4C 0A 08        JMP QUI
0876
0877 4C 0A 08        JMP QUI
0878
0879 4C 0A 08        JMP QUI
0880
0881 4C 0A 08        JMP QUI
0882
0883 4C 0A 08        JMP QUI
0884
0885 4C 0A 08        JMP QUI
0886
0887 4C 0A 08        JMP QUI
0888
0889 4C 0A 08        JMP QUI
0890
0891 4C 0A 08        JMP QUI
0892
0893 4C 0A 08        JMP QUI
0894
0895 4C 0A 08        JMP QUI
0896
0897 4C 0A 08        JMP QUI
0898
0899 4C 0A 08        JMP QUI
0900
0901 4C 0A 08        JMP QUI
0902
0903 4C 0A 08        JMP QUI
0904
0905 4C 0A 08        JMP QUI
0906
0907 4C 0A 08        JMP QUI
0908
0909 4C 0A 08        JMP QUI
0910
0911 4C 0A 08        JMP QUI
0912
0913 4C 0A 08        JMP QUI
0914
0915 4C 0A 08        JMP QUI
0916
0917 4C 0A 08        JMP QUI
0918
0919 4C 0A 08        JMP QUI
0920
0921 4C 0A 08        JMP QUI
0922
0923 4C 0A 08        JMP QUI
0924
0925 4C 0A 08        JMP QUI
0926
0927 4C 0A 08        JMP QUI
0928
0929 4C 0A 08        JMP QUI
0930
0931 4C 0A 08        JMP QUI
0932
0933 4C 0A 08        JMP QUI
0934
0935 4C 0A 08        JMP QUI
0936
0937 4C 0A 08        JMP QUI
0938
0939 4C 0A 08        JMP QUI
0940
0941 4C 0A 08        JMP QUI
0942
0943 4C 0A 08        JMP QUI
0944
0945 4C 0A 08        JMP QUI
0946
0947 4C 0A 08        JMP QUI
0948
0949 4C 0A 08        JMP QUI
0950
0951 4C 0A 08        JMP QUI
0952
0953 4C 0A 08        JMP QUI
0954
0955 4C 0A 08        JMP QUI
0956
0957 4C 0A 08        JMP QUI
0958
0959 4C 0A 08        JMP QUI
0960
0961 4C 0A 08        JMP QUI
0962
0963 4C 0A 08        JMP QUI
0964
0965 4C 0A 08        JMP QUI
0966
0967 4C 0A 08        JMP QUI
0968
0969 4C 0A 08        JMP QUI
0970
0971 4C 0A 08        JMP QUI
0972
0973 4C 0A 08        JMP QUI
0974
0975 4C 0A 08        JMP QUI
0976
0977 4C 0A 08        JMP QUI
0978
0979 4C 0A 08        JMP QUI
0980
0981 4C 0A 08        JMP QUI
0982
0983 4C 0A 08        JMP QUI
0984
0985 4C 0A 08        JMP QUI
0986
0987 4C 0A 08        JMP QUI
0988
0989 4C 0A 08        JMP QUI
0990
0991 4C 0A 08        JMP QUI
0992
0993 4C 0A 08        JMP QUI
0994
0995 4C 0A 08        JMP QUI
0996
0997 4C 0A 08        JMP QUI
0998
0999 4C 0A 08        JMP QUI
1000
1001 4C 0A 08        JMP QUI
1002
1003 4C 0A 08        JMP QUI
1004
1005 4C 0A 08        JMP QUI
1006
1007 4C 0A 08        JMP QUI
1008
1009 4C 0A 08        JMP QUI
1010
1011 4C 0A 08        JMP QUI
1012
1013 4C 0A 08        JMP QUI
1014
1015 4C 0A 08        JMP QUI
1016
1017 4C 0A 08        JMP QUI
1018
1019 4C 0A 08        JMP QUI
1020
1021 4C 0A 08        JMP QUI
1022
1023 4C 0A 08        JMP QUI
1024
1025 4C 0A 08        JMP QUI
1026
1027 4C 0A 08        JMP QUI
1028
1029 4C 0A 08        JMP QUI
1030
1031 4C 0A 08        JMP QUI
1032
1033 4C 0A 08        JMP QUI
1034
1035 4C 0A 08        JMP QUI
1036
1037 4C 0A 08        JMP QUI
1038
1039 4C 0A 08        JMP QUI
1040
1041 4C 0
```


rare una forma d'onda qualsiasi contenuta in memoria: così per generare una sinusoide è sufficiente calcolare il valore del seno all'interno di un periodo (con incrementi a piacere ma costanti) e porli in memoria facendo in modo che varino fra un minimo di 0 ed un massimo di 255.

Il programma legge uno di seguito all'altro questi valori e li trasferisce ad un convertitore D/A. Dopo aver letto N valori si ritorna al primo.

La frequenza massima di lavoro dipende dal numero di valori all'interno di un periodo ed è di $f_{MAX} = \frac{1}{15N} 10^6$. Se ad es. sono desiderati 20 intervalli, si arriva a generare onde dalla frequenza di più di 3300 Hz, sufficienti, dopo essere state opportunamente filtrate, ad imitare con buona approssimazione la voce umana.

Un programma che realizza la funzione inversa è il progr. 27: esso è in grado di leggere da un convertitore A/D una serie di valori e caricarli in memoria. L'acquisizione dei dati

da questo convertitore è leggermente più complessa dell'output, dati i tempi di conversione nettamente più lunghi (ad es. il convertitore AD 7570 della Analog Devices impiega circa $40 \mu S$): non è più possibile dare semplicemente il segnale di START quando si mette il dato sulla porta ma bisogna attendere che il convertitore abbia finito la conversione prima di leggere un nuovo dato. Questa attesa può essere fatta sia dal punto di vista software (ad es. con una serie

PAGE 0001

ADDR OBJECT SOURCE

; Generatore di onde periodiche
; (Con convertitore D/A)
; (Si può rallentare inserendo una serie
; di istruzioni NOP (2 microsecondi))
; Il periodo dipende da LUNG (N. byte)
PB=\$FFA0
PBDIR=\$FFA2
CR=\$FFAC
LUNG=\$AA (0 altro valore)

(256)
0000 A9 A0 LDA #\$A0 (=X10100
0001
0002 8D AC FF STA CR
0005 A9 FF LDA #\$FF
0007 8D A2 FF STA PBDIR
000A A2 00 SU LDX #\$0
000C B5 00 QUI LDA 0,X
000E 8D A0 FF STA PB
; NOP NOP
0011 E8 INX
0012 E0 AA CPX #LUNG
0014 D0 F6 BNE QUI
0016 4C 0A 00 JMP SU
; Vincenzo Villa V D 16/3/1983
END

ERRORS=0000

PROG. 26

di NO OPERATION) che rilevando il segnale BUSY generato dal convertitore alla fine del suo lavoro: é stata scelta questa seconda strada perché più precisa, dato che non rischia di leggere prima che la conversione sia finita né dopo che é finita da tempo, e perché indipendente dal tipo di convertitore usato.

```
PAGE 0001
ADDR OBJECT    SOURCE
;      INPUT DA A-D CONVERTER
; Il terminale STRT va collegato a CA2
; Il terminale BUSY va collegato a CA1
;
;      PA=$FFA1
;      PAD=$FFA3
;      PCR=$FFAC
;      IFR=$FFAD
;      DATA=$0
;
0000 A9 00          LDA #$0
0002 8D A3 FF      STA PAD
0005 A5 00          LDA DATA
0007 8D A1 FF      STA PA
000A A9 0D          LDA #$D
000C 8D AC FF      STA PCR
000F A9 0F          LDA #$F
0011 8D AC FF      STA PCR
0014 AD AD FF      LDA IFR
0017 29 02          AND #$2
0019 F0 F9          BEQ LOOP
001B 00            BRK
; Vincenzo Villa V D      30/5/1983
;      .END

ERRORS=0000
```


I N T E R R U P T

Parlando del progr. N. 2I é emerso come, attendendo che la stampante segnali sulla linea ACK , la disponibilità a ricevere nuovi dati, il programma si blocca per qualche millesecundo facendo il lavoro molto banale di controllare lo stato di una linea: in questo tempo la CPU potrebbe benissimo dedicarsi allo svolgimento di altri lavori più impegnativi. Un altro difetto di un tale sistema, si ha nel caso in cui il microprocessore non sa se e quando avverrà la trasmissione: in questo secondo caso il programmatore si trova obbligato a far controllare abbastanza frequentemente le periferiche, per ricevere spesso risposte negative, rallentando inoltre eccessivamente l'esecuzione del programma. una soluzione a questi problemi é fornito dall'interruzione del programma (interrupt). Con questa modalità, é possibile che determinati eventi generati da unità esterne interrompano l'esecuzione normale del programma, che potremmo definire principale, e forzino la CPU ad eseguire una particolare serie di istruzioni prima di poter tornare all'esecuzione del programma principale.

Riferendoci sempre al caso della stampante, si potrebbe, ad es., inviare un primo carattere alla porta di I/O e, subito dopo intraprendere un programma qualsiasi(diciamo la risoluzione di un'espressione algebrica); dopo un certo tempo la stampante comunica che ha finito di stampare il carattere precedente ed é pronta a riceverne uno nuovo, segnalandolo su una particolare linea che genera un'interruzione nel programma principale ed invita la CPU a trasmettere un nuovo carattere; il processore invia allora un altro byte e, subito dopo, riprende la risoluzione della espressione esattamente dove l'aveva lasciata.

Questo lavoro può dare l'impressione che la CPU svolga contemporaneamente due funzioni e, agli occhi dell'utilizzatore, appare effettivamente così anche se realmente i due lavori sono svolti alternativamente.

Vediamo di spiegare come reagisce il 6502 ricevendo una richiesta di interrupt. Bisogna innanzitutto dire che questo microprocessore possiede due linee di interrupt: la IRQ é una linea mascherabile che può cioè essere ignorata dal processore nel caso che un particolare bit del flags register sia posto a 1, ed una che é sempre eseguita (NMI).

Dato che quest'ultima non é direttamente accessibile da parte dell'utiliz=

zatore dell'AIM 65/40, lo ignoriamo, dicendo semplicemente che é collegata al tasto ATT della tastiera ed é usata per bloccare un programma "entrato in loop").

Ricevendo un segnale sulla linea IRQ ed essendo tale linea abilitata, la CPU carica nel program counter l'indirizzo contenuto nelle locazioni FFFE e FFFF e da questo indirizzo comincia l'esecuzione di un programma. Nel caso dell'AIM 65/40 queste due locazioni sono in ROM e come tali sono molto difficilmente modificabili da parte di un utilizzatore: il programma che però parte da questo indirizzo ha come prima istruzione il salto indiretto all'indirizzo contenuto in due locazioni RAM accessibili al programmatore. Il programma standard eseguito ogni volta che é richiesta un interrupt consiste nel controllo di alcune periferiche per poter riconoscere quali di queste ha chiesto l'interruzione ed eseguire il programma appropriato: infatti, al momento della richiesta dell'interruzione il microprocessore non sa cosa l'ha richiesta e deve eseguire un polling di tutti quei dispositivi che la possano richiedere finché trova che l'ha richiesta. Il programma standard controlla i due VIA del sistema (tastiera stampante e display) e, se non ha trovato chi ha richiesto l'interruzione, richiama, attraverso un altro salto indiretto, un programma che segnala sul display la frase IRQ ERROR. Generalmente il programmatore mette in queste due ultime locazioni l'indirizzo iniziale del proprio programma delle interrupt, facendo sì che, dopo aver verificato che l'interrupt non é stata richiesta da nessuno dei due VIA del sistema, può eseguire una ulteriore ricerca e, se trovato il dispositivo responsabile, fa partire un programma che accontenta la richiesta.

Come accennato all'inizio, dopo che é stata soddisfatta la richiesta della periferica, é necessario che la CPU riprenda il programma esattamente dove era stato interrotto: questo é possibile perché alla richiesta di interrupt il processore salva automaticamente il program counter e il flags register (che sono modificati da qualunque istruzione), per potersi poi ricordare dove si era fermato e poter riprendere dallo stesso punto. Un microprocessore possiede però anche altri registri che possono essere modificati durante lo svolgimento del programma di gestione degli interrupts: per questo spesso tali programmi cominciano con il salvataggio dei registri X e Y e dell'accumulatore e finiscono con il loro ricaricamento.

PAGE 0001

ADDR OBJECT SOURCE

O R O L O G I O

; Ore = Byte 0

; Minuti = Byte 1

; Secondi = Byte 2

;

CO=#3

SECO=#2

PRIMI=#1

ORE=#0

```

0900 A9 C0      LDA #$C0
0902 8D AB FF   STA $FFAB (ACR)
0905 8D AE FF   STA $FFAE (IER)
0908 A9 50      LDA #$50
090A 8D A4 FF   STA $FFA4 (T1-L)
090D A9 C3      LDA #$C3
090F 8D A5 FF   STA $FFA5 (T1-H)
0912 58         CLI
0913 A9 00      LDA #$0
0915 8D 2D 02   STA $22D
0918 A9 08      LDA #$08
091A 8D 2E 02   STA $22E
091D 00         BRK

```

```

0800 2D AD FF   BIT $FFAD (IFR)
0803 30 03      BMI CONT
0805 4C A7 F5   JMP $F5A7 ERROR
0808 70 04      BVS TI
080A 4C A7 F5   JMP $F5A7 ERROR
080D 18         CLC
080E F8         SED
080F A5 03      LDA CO
0811 69 01      ADC #1
0813 85 03      STA CO
0815 C9 20      CMP #$20
0817 D0 31      BNE RIT
0819 18         CLC
081A A9 00      LDA #0
081C 85 03      STA CO
081E A5 02      LDA SECO
0820 69 01      ADC #1
0822 85 02      STA SECO
0824 C9 60      CMP #$60
0826 D0 22      BNE RIT
0828 18         CLC
0829 A9 00      LDA #0
082B 85 02      STA SECO
082D A5 01      LDA PRIMI
082F 69 01      ADC #1
0831 85 01      STA PRIMI
0833 C9 60      CMP #$60
0835 D0 13      BNE RIT
0837 18         CLC
0838 A9 00      LDA #0
083A 85 01      STA PRIMI
083C A5 00      LDA ORE
083E 69 01      ADC #1
0840 85 00      STA ORE
0842 C9 24      CMP #$24
0844 D0 04      BNE RIT
0846 A9 00      LDA #0
0848 85 00      STA ORE
084A AD A4 FF   RIT
084D 40         RTI

```

; Vincenzo Villa

V D

30/3/1983

END

ERRORS=0000

PROG. 28

Per il ritorno allo svolgimento del programma normale è inoltre necessario fare eseguire l'istruzione RTI che riimmagazzina automaticamente il vecchio programma counter e il flags register.

Alla fine della gestione dell'interrupt i registri del microprocessore si trovano quindi nello stesso stato di prima della richiesta ed il programma principale prosegue senza cambiamenti, con un ritardo sul tempo di esecuzione normale.

OROLOGIO

Vediamo ora un programma gestito tramite interrupt.

Il progr. 28 è in grado di imitare le prestazioni fornite da un normale orologio digitale senza praticamente influire sullo svolgimento dei programmi normali: questo è possibile grazie al timer TI gestito tramite interrupt.

Nel timer è posto il numero \$C350 equivalente al decimale 50.000 in modo da provocare ogni 50 mS un interrupt alla CPU, richiamando il programma posto alla locazione \$800 attraverso l'indirizzo messo in \$22D e \$22E. Questo programma

non fa altro che incrementare un contatore che, giunto a 20, incrementa i secondi (infatti $50 \times 10^{-3} = 1$) e quando questo giungono a 60 i minuti e quindi le ore.

Da notare che le prime 5 istruzioni del programma controllano che è stato effettivamente TI a generare l'interrupt (in caso contrario è segnalato un errore) ed anche la penultima che serve per cancellare il flag dell'IFR che segnala che l'interrupt è stata richiesta da TI, per evitare che, appena ripreso il programma principale, la CPU sia subito interrotta dal fatto che quel flag è ancora a 1.

Il tempo impiegato da questo programma è dell'ordine del millisecondo per ogni secondo, occupando meno dello 0,1 % del tempo totale del microprocessore.

Lo stesso programma può essere usato anche per misurare il tempo trascorso dal momento in cui TI è stato abilitato ad interrompere il microprocessore a quando sono lette le locazioni contenenti il tempo misurato.

=====

CONTROLLO DELLA
VELOCITA' DI
UN MOTORE C.C.

=====

L'applicazione forse più interessante dei microprocessori è attualmente il loro uso nel controllo automatico di processi esterni, cosa che sta diventando sempre più conveniente perchè, alle caratteristiche di velocità, affidabilità e precisione, si vanno aggiungendo costi sempre più bassi che attualmente sono scesi, per grandi quantità, a poche dollari il pezzo.

In effetti si vanno sempre più diffondendo macchine che, attraverso un microcomputer (che a volte può essere contenuto in un singolo chip), tengono sotto controllo una serie di fenomeni fisici assai diversi tra di loro: tipici esempi si possono trovare nel computer di bordo installato su alcune automobili per tenere controllate le funzioni più importanti e segnalare gli eventuali guasti, le cosiddette macchine a controllo numerico che, una volta programmate, sono in grado di eseguire una serie di operazioni meccaniche con grande precisione e velocità, i piloti automatici di aerei e navi, la realizzazione di elettrodomestici quali lavabiancheria e forni a micro-onde controllati digitalmente ecc.. Come si vede si tratta di elaborare dati riferiti a fenomeni fisici assai diversi tra di loro, che vanno da velocità a pressioni, a temperature, da spostamenti a intensità luminose.

In questa sezione è esaminato un programma o, meglio, una serie di programmi che permettono al microprocessore di rilevare la velocità di un motore in corrente continua e di mantenerla in un intorno di quella determinata attraverso la tastiera; quella che in questo caso è una velocità scelta direttamente dall'uomo, potrebbe essere benissimo calcolata da un altro programma del computer in funzione dei dati provenienti da sensori. Ad es., se il motore è collegato ad una pompa, la sua velocità dipenderà dalla pressione rilevata; oppure se si sta controllando un ventilatore, la sua velocità dipende dalla temperatura effettiva misurata.

Il controllo della velocità vero e proprio, è sempre realizzato totalmente in linguaggio macchina, attraverso un'opportuna gestione degli interrupts provenienti dallo USER VIA, rendendolo così molto veloce (bisogna infatti notare che sono realizzati circa 400 controlli al secondo) e contemporaneamente lasciando libero il computer di eseguire altri programmi: infatti, mentre la velocità è mantenuta stabile, è pos

sibile programmare con qualsiasi linguaggio ed usare tutte le utility del sistema, senza alcun sintomo sensibile, né per quanta riguarda la velocità di esecuzione, che diminuisce nei casi limite di meno del 4%, né in termini della correttezza di esecuzione, nel caso che non si usino le locazioni che contengono il programma di controllo (meno si un centinaio di bytes su circa 32000 bytes di memoria disponibile) né i sostegni hardware usati per il controllo del motore (che si riducono alle linee CBI e CB2 e al timer 2).

Per quanto riguarda la comunicazione con l'uomo, si è preferito usare due strade: A) Una prima, anch'essa realizzata completamente in linguaggio macchina, permette di inserire la velocità in termini assoluti come giri al minuto e di verificare il funzionamento attraverso la lettura, espressa numericamente, in modo relativo sul display o, sulla stampante, attraverso un grafico ad alta risoluzione. B) Una seconda strada seguita, usando il linguaggio BASIC, permette, grazie alle caratteristiche di gran lunga migliori, dal punto di vista dell'I/O e dell'elaborazione numerica di questo linguaggio, di realizzare un programma più flessibile unitamente alla semplicità e alla brevità di programmazione. Bisogna notare che questa seconda strada non è percorribile quando si ha a che fare con un sistema a controllo numerico di tipo industriale, dato che ben raramente è disponibile un interprete BASIC.

All'interno di questo lavoro si sono divisi i programmi in tre grossi blocchi: inserimento della velocità da stabilizzare e di altri dati necessari in seguito; rilevazione e controllo della velocità; visualizzazione degli effetti del programma.

Il blocco più importante è certamente il secondo in cui il computer si trova a dover lavorare in tempo reale attraverso gli interrupts, controllando un processo abbastanza veloce anche se non velocissimo. Gli altri due blocchi sono invece orientati all'uomo e, come tale, non fanno parte del controllo vero e proprio ma servono per poter guidare e verificare l'azione della macchina.

RILEVAZIONE DELLA VELOCITA'

Esaminiamo come primo punto i modi attraverso cui il computer può leggere la velocità reale del motore. Esistono fondamentalmente due tecniche:

- Un primo metodo consiste nell'uso di una dinamo tachimetrica cioè un trasduttore di velocità formato da un rotore che gira in un campo magnetico generato da magneti permanenti. La tensione prelevata fra le spazzole della macchina è esprimibile con la relazione:

$$V = (Np/s) \cdot \Phi \cdot n$$

dove:

N = numero di conduttori attivi

p = numero di poli statorici

s = numero di spazzole

Φ = flusso prodotto da ogni polo statorico

n = velocità del rotore (giri al secondo)

La relazione sopra riportata indica pertanto la proporzionalità che esiste fra la tensione alle spazzole e la velocità del rotore trascinato dall'organo del quale si vuole conoscere la velocità. A questo punto è sufficiente leggere la tensione ai capi del rotore, eventualmente amplificata, attraverso un convertitore A/D ed un programma del tipo di progr. 27 per rilevare, attraverso un'opportuna costante, la velocità. La risoluzione ottenibile con una tale tipo di controllo è data essenzialmente dalle caratteristiche del convertitore ed in particolare dal numero di bits con cui lavora: se, ad es. si ha a disposizione un convertitore a 8 bits (come l'AD 7570) e volendo misurare le velocità comprese fra 0 e 3000 giri al minuto, si ha un errore massimo teorico di ± 12 giri al minuto, a cui si devono aggiungere gli errori accidentali dovuti al convertitore, alla non perfetta linearità del trasduttore e ad errori nella misurazione della costante che lega velocità e tensione. Si tratta comunque di un metodo più che sufficiente nella maggior parte dei casi, caratterizzato dal fatto che il computer che decide quando la misurazione deve essere fatta, dato che la tensione è sempre presente alle spazzole.

- Un secondo metodo consiste nel rilevare attraverso un sensore la fine di un giro dell'albero motore che si vuole controllare ed attraverso questo risalire alla velocità. Per segnalare la fine di un giro é possibile usare un sensore meccanico (ad es. una spazzola strisciante che in un determinato punto dà un segnale elettrico chiudendo un interruttore), magnetico (in cui vi é solidale all'albero, un piccolo magnete che, passando davanti ad una bobina genera una tensione che sarà amplificata e squadrata) od infine ottico, (in cui vi é un disco collegato all'albero motore con un foro laterale; tra le due facce del disco sono posti una sorgente luminosa ed un sensore ottico che, in corrispondenza del foro, per un breve istante dà un segnale elettrico).

In campo industriale il metodo più usato é il secondo dato che non vi é necessità di contatto fisico tra rilevatore ed albero ed é possibile proteggerli entrambi da spruzzi di lubrificanti, polvere e da tutto quello che potrebbe danneggiarli ad es. con un rivestimento plastico.

Nell'applicazione pratica che ho fatto, ho preferito usare un sistema ottico data la sua più facile reperibilità e la compatibilità diretta con le porte TTL nel sensore usato.

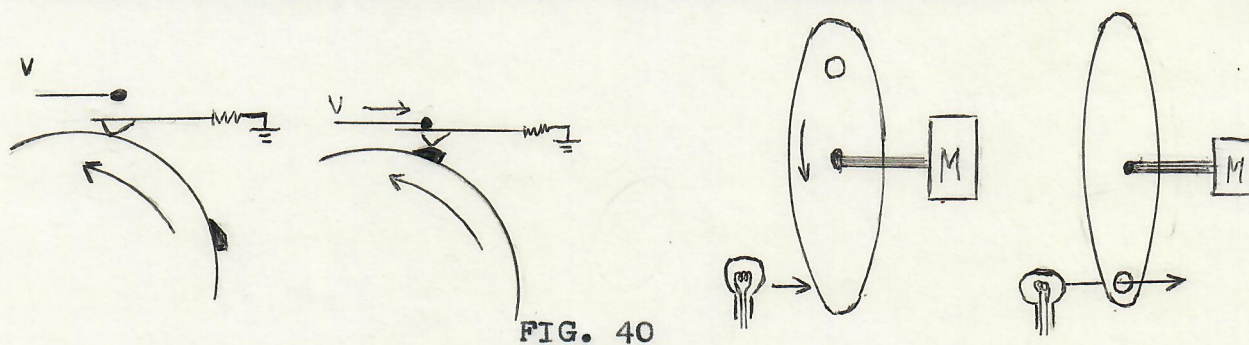


FIG. 40

Questo tipo di controllo può a sua volta essere fatto in due modi diversi:

- a) Si può fissare un certo tempo e contare quanti giri sono fatti in questo intervallo per ottenere direttamente la velocità. Si tratta di un metodo piuttosto impreciso perché, fissando ad es. un intervallo di un secondo; tempo piuttosto lungo, é ottenibile un errore massimo di $\pm$ un giro al secondo cioè di $\pm$ 60 giri al minuto; quando poi il controllo deve essere effettuato su tempi più brevi come avviene in realtà (dell'ordine di almeno 1/10, 1/100 di secondo) l'errore cresce ulteriormente, diventando troppo grosso: é ad es; impensabile leggere una velocità di 3000

giri al minuto quando l'errore é di 600 (nel caso che si rilevi ogni decimo di secondo) o addirittura di 6000 giri al minuto)(rilevando ogni centesimo di secondo) ottenendo un errore superiore alla velocità stessa da misurare.

Una soluzione parziale potrebbe essere quella di mettere non I ma più fori di riferimento all'interno di uno stesso giro, ad es. inserendo 10 fori o 10 magneti, in questo modo la precisione della misurazione aumenta proporzionalmente con l'aumento dei riferimenti. In effetti questo metodo si rivela adatto solo nei casi in cui vi é la possibilità di fornire molti impulsi per giro dell'albero motore e la velocità é piuttosto elevata (alcune decine di migliaia di giri al minuto). Il vantaggio principale di questo metodo sta nel fatto che la velocità é ottenuta direttamente ed in termini assoluti, tali da non necessitare né di una conversione, né della costruzione sperimentale di una curva che lega la velocità alla grandezza misurata.

b) Un altro metodo consiste nel misurare il tempo intercorso e, facendo l'inverso, trovare la velocità reale in giri al secondo.

La difficoltà principale si ha nel momento di misurare con sufficiente precisione questo tempo che é dell'ordine dei millesimi di secondo: per far questo é usato un timer del computer gestito tramite interrupt e ciò permette di ottenere un errore massimo teorico di $1 \mu s$; a ciò deve poi essere aggiunto un certo errore generato dal tempo di reazione del programma. L'errore totale si mantiene comunque inferiore a 5 giri al minuto per una velocità di 3000 giri al minuto; scendendo a velocità più basse l'errore, sempre uguale in termini assoluti, influisce percentualmente di meno, dato che il tempo trascorso tende ad aumentare. Un'altra difficoltà si può trovare nel calcolare ogni volta l'inverso del tempo attraverso un programma piuttosto lungo e che non sempre fa in tempo a dare i risultati abbastanza velocemente; questo problema é aggirato, come illustrato in seguito, semplicemente calcolando l'inverso, una volta per tutte e realizzando il controllo non sulla velocità ma direttamente sul tempo.

Un difetto di un tale tipo di rilevazione sta nel fatto che il momento di lettura é deciso dal motore e dipende dalla sua velocità: questo fatto

che non presenta problemi alle velocità medie ed alte diventa inefficiente alle velocità più basse dell'ordine di pochi giri al secondo. Anche in questo caso è possibile, per migliorare le prestazioni, inserire più fori all'interno del disco a distanze rigidamente costanti: in questo modo la rilevazione è più frequente proporzionalmente al numero di fori.

C O N T R O L L O D E L L A V E L O C I T A'

Quando il computer ha rilevato la velocità, deve poter controllare se essa è quella prescritta e, se no, deve poterla modificare. Anche in questo caso ci si può muovere secondo due direttive: una prima è il controllo ON-OFF, la seconda prevede l'invio di una tensione proporzionale alla velocità.

Una via di mezzo è il cosiddetto controllo ON-OFF con ritorno a 0.

a) Nel controllo ON-OFF la velocità rilevata con una delle tecniche appena viste è confrontata con quella da stabilizzare: se è minore, al motore è applicata una tensione equivalente a quella di funzionamento (9 V in questo caso); se superiore il motore è "spento". Si tratta di un metodo abbastanza semplice da realizzare e che richiede una rete di potenza altrettanto semplice ma che riesce, nel caso in cui il controllo è molto più frequente della variazione del fenomeno, a mantenerlo abbastanza stabile in un intorno dei valori decisi.

b) Un secondo metodo consiste nell'inviare una tensione costante calcolata tenendo conto della velocità attuale e di quella da stabilizzare: se la velocità rilevata è minore di quella prefissata questa tensione va aumentata, se è uguale va lasciata invariata, altrimenti bisogna diminuirla. Per poter avere la tensione per alimentare il motore è usato un convertitore digitale / analogico accoppiato con uno stadio di potenza: si tratta di un metodo abbastanza costoso in termini economici per la necessità di un convertitore; esso permette comunque di ottenere controlli abbastanza sofisticati.

Un problema che si può creare usando questo tipo di controllo è il pericolo di innescare oscillazioni persistenti intorno al

punto di lavoro: è possibile risolverlo realizzando controlli più frequenti della variazione dell'errore; (tipicamente almeno 5 volte di più). Nel caso del controllo della velocità del motore, è possibile arrivare facilmente ad una frequenza di controllo di diverse migliaia di volte al secondo, più che sufficienti per eliminare qualsiasi instabilità

- c) Il controllo ON-OFF CRZ è costituito da un'onda quadra di frequenza costante con ciclo utile variabile in funzione della velocità da stabilizzare; essa, prima di arrivare al motore viene integrata (attraverso un condensatore), fornendo una tensione costante proporzionale al ciclo utile.

DESCRIZIONE DELL'HARDWARE

Prima di descrivere il programma che realizza il controllo è utile accennare brevemente su come è rilevata e controllata la velocità dal punto di vista hardware.

La rilevazione della velocità è realizzata con un sistema ottico attraverso un disco solidale all'albero motore con 8 fori dal diametro di circa 0,5 cm. a distanza rigidamente costante. La sorgente luminosa è costituita da un diodo ad infrarossi il TIL 38 della TEXAS INSTRUMENTS (le specifiche sono in fig. 42) alimentato a 5 V attraverso una resistenza da 150Ω . La corrente assorbita è quella consigliata dal

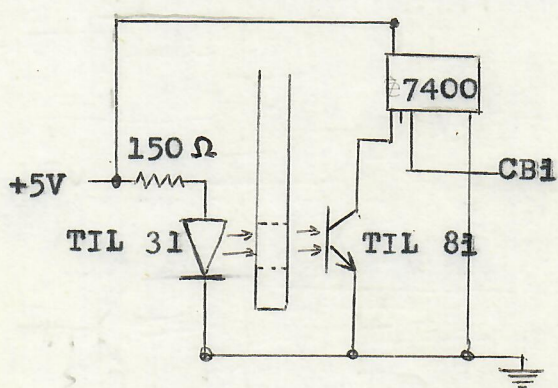


FIG. 41

costruttore di 25 mA. Per la rilevazione del foro è usato il fototransistor TIL 81 sempre nella TI, compatibile come frequenza luminosa e intensità col il fotodiodo; esso è messo esattamente di fronte al fotodiodo ad una distanza di circa 0,7 cm, sufficiente per far passare il disco ma non eccessiva per permettere una sufficiente illuminazione.

L'emettitore è collegato direttamente a massa mentre il collettore è collegato direttamente all'ingresso di una porta TTL che polarizza il transistor attraverso la resistenza interna collegata a +5V, secondo

do gli schemi consigliati dal costruttore.

Il transistor funziona o all'interdizione nel caso che la luce del diodo non gli arriva (portando quindi ad I logico l'ingresso della porta TTL) o in posizione molto prossima alla saturazione nel caso in cui la luce del fotodiode lo colpisce attraverso un foro del disco: dato che V_{ce} non supera i 0,8 V assorbendo una corrente di sink di 1,6 mA il transistor é capace di portare a livello 0 la porta TTL. L'uscita della porta é collegata direttamente alla linea CBI del

TYPES TIL31, TIL33, TIL34 P-N GALLIUM ARSENIDE INFRARED-EMITTING DIODES

TYPICAL CHARACTERISTICS

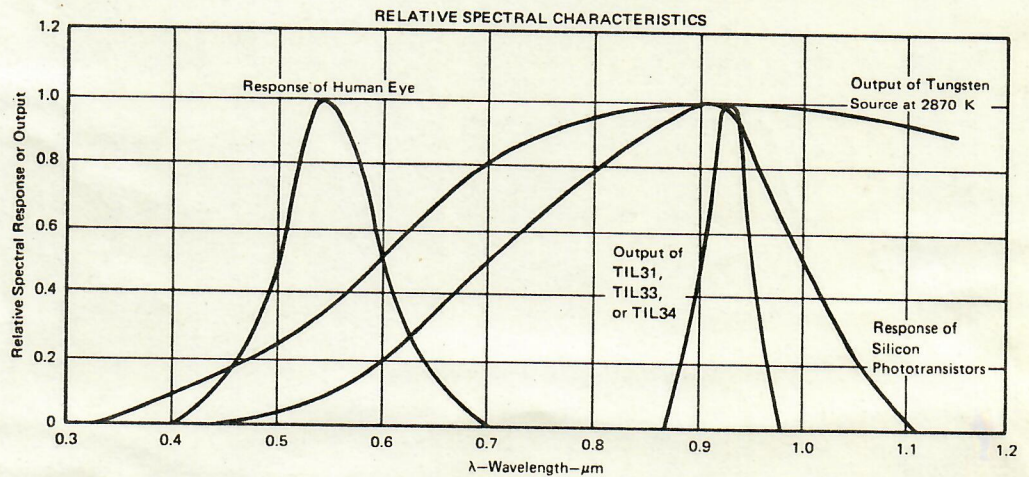


FIGURE 1

FIG. 42

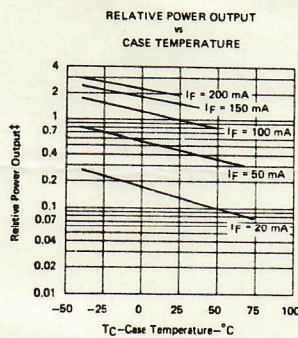


FIGURE 2

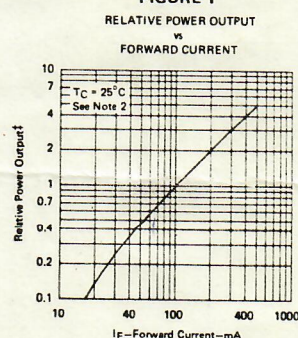


FIGURE 3

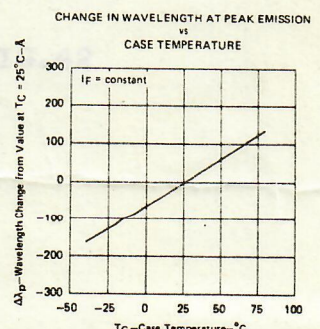


FIGURE 4

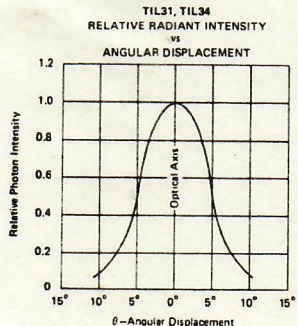


FIGURE 5

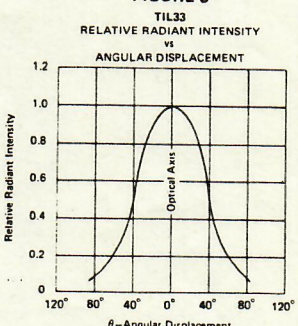


FIGURE 6

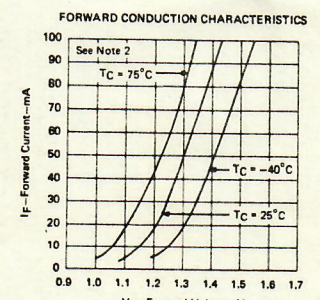


FIGURE 7

NOTE 2: This parameter must be measured using pulse techniques. $t_w = 0.04$ ms, duty cycle $\leq 10\%$.
‡ Normalized to output at $I_F = 10$ mA, $T_C = 25^\circ\text{C}$.

computer ed é attraverso quest'unica linea che é letta la velocità. Per il cpntrollo del motore con metodo ON-OFF é usato un transistor di potenza il cui emettitore é collegato a massa ed il collettore ad una delle spazzole del motore; l'atra spazzola é collegata alla tensione di alimentazione a +9 V.

La base é collegata alla linea CB2 del computer attraverso una resistenza di 4 K ed un buffer in grado di erogare fino ad 8 mA.

Inoltre vi é una resistenza da 1 K verso massa.

Anche questo transistor funziona solo in interdizione (nel caso che

TYPE TIL81 N-P-N PLANAR SILICON PHOTOTRANSISTOR

TYPICAL CHARACTERISTICS

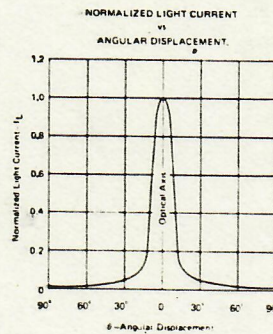


FIGURE 2

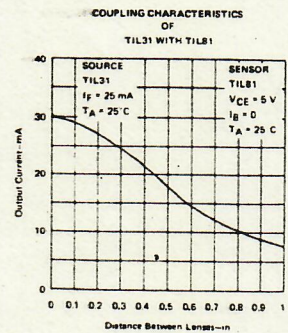


FIGURE 3

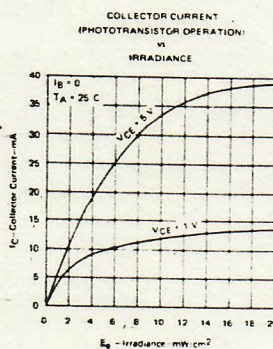


FIGURE 4

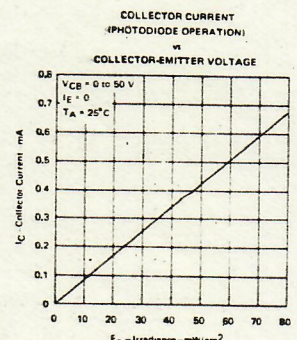


FIGURE 5

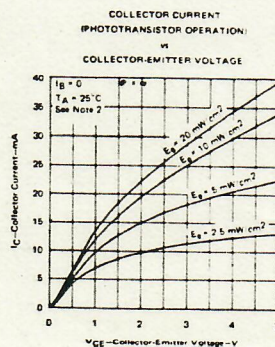


FIGURE 6

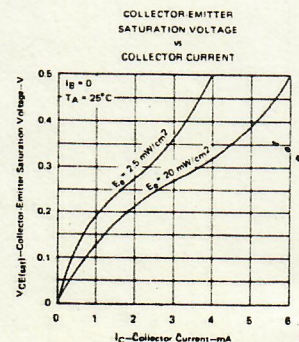


FIGURE 7

NOTE 2: Irradiance (E_0) is the radiant power per unit area incident upon a surface. For these measurements the source is an unfiltered tungsten linear-filament lamp operating at a color temperature of 2870 K.

CB2 si trovi a livello logico 0) o in saturazione se CB2 si trova a livello logico 1: nel primo caso il transistor si comporta come un interruttore aperto, V_{ce} é circa uguale a 9 V e la corrente di collettore é praticamente 0 (quindi il motore non é alimentato; se il CB2 é ~~all~~ 1, la tensione V_{ce} é di 0,3 V, permettendo al motore di funzionare.

Per scaricare a massa i picchi ad alta frequenza causati dallo spegnimento del motore é stato inoltre aggiunto un condensatore da 2000 pF.

P R O G R A M M A D I C O N T R O L L O

Passiamo ora alla descrizione del programma che controlla la velocità del motore; esso é realizzato completamente in assembler ed é diviso in tre blocchi:

- in una prima parte é presentato il listato completo che permette di leggere da tastiera la velocità da stabilizzare, di convertirla da decimale a binario, di chiedere il numero di fori sul disco ed il tipo o i tipi di output desiderati (numerico o grafico); esso occupa le locazioni da \$8AE fino a \$9B6 oltre a 174 bytes usati per definire le frasi per chiedere queste informazioni.
- una seconda parte calcola l'inverso della velocità, prepara al funzionamento corretto i timers e le due porte di handshake CBI e CB2, abilita gli interrupt e fa partire il motore.
- la terza realizza infine il controllo vero e proprio tramite interrupt e l'output verso l'uomo.

Dato lo sviluppo del programma in 9 "pagine" é impossibile descriverlo in tutti i dettagli e si é preferito tralasciare una descrizione approfondita delle parti che sono già state viste precedentemente e concentrare l'attenzione al controllo cioè alla terza parte.

P A R T E P R I M A (Richiesta, della velocità da stabilizzare)

In questa parte del programma si é cercato di chiedere all'utilizzatore tutte quelle informazioni richieste dal programma per funzionare; é infatti molto scomodo dover definire direttamente in memoria le variabili usate, magari direttamente in esadecimale.

Si é cercato pertanto di usare una forma chiara e semplice: le domande sono infatti poste in forma diretta scrivendole sul display e, se richiesto, sulla stampante. Le risposte sono date attraverso la tastiera scrivendo semplicemente un numero in decimale o, se é richiesta una risposta affermativa o negativa, le lettere Y o N, iniziali di yes e no.

Per poter stampare le frasi che qualificano il programma e chiedono dati é necessario definirle prima in memoria attraverso la direttiva .BYT: ad es. nelle locazioni da \$800 a \$818 sono scritti i 27 codici ASCII componenti la frase "LIST SU STAMPANTE (Y/N)?"; la stessa cosa é fatta anche per le altre 6 frasi scritte dal programma, per un totale di 174 caratteri.

Vediamo ora, a grandi linee come si sviluppa il programma, tenendo conto sia dei due esempi di funzionamento che del listato (PAGE)1,2 e3)
La prima frase stampata chiede se é desiderata la stampa su carta delle

```
LIST SU STAMPANTE (Y/N) ? Y
  CONTROLLO DI VELOCITA' ON-OFF
  LETTURA CON FOTOTRANSISTOR

VELOCITA'=3000
NUMERO FORI =8
OUTPUT NUMERICO (Y/N) ?Y
GRAFICO AD ALTA RISOLUZIONE (Y/N) ?Y

CMDB      0B B9 08 FF 00 00 00 86 .....
```

```
LIST SU STAMPANTE (Y/N) ? Y
  CONTROLLO DI VELOCITA' ON-OFF
  LETTURA CON FOTOTRANSISTOR

VELOCITA'=255
NUMERO FORI =1
OUTPUT NUMERICO (Y/N) ?N
GRAFICO AD ALTA RISOLUZIONE (Y/N) ?N

CMDB      00 FF 01 00 00 00 00 86 .....
```

successive risposte: la risposta affermativa é data attraverso la tastiera premendo il tasto Y, quella negativa con uno qualsiasi degli

altri tasti. Nei due esempi é stata data risposta affermativa altrimenti

PAGE 0001

ADDR OBJECT SOURCE

CONTROLLO ON - OFF

```
0800 4C 49          .BYT 'LIST SU STAM
PANTE (Y/N) ?'
0819 20 20          .BYT 'CONTROL
LO DI VELOCITA' ON-OFF', $A
083B 0A
083C 20 20          .BYT 'LETTURA
CON FOTOTRANSISTOR', $A, $A
085B 0A
085C 0A
085D 56 45          .BYT 'VELOCITA'='
0867 4E 55          .BYT 'NUMERO FORI
='
0874 4F 55          .BYT 'OUTPUT NUMER
ICO (Y/N) ?'
088B 47 52          .BYT 'GRAFICO AD A
LTA RISOLUZIONE (Y/N) ?'
```

```
READ=$F22C
OPNPTR=$F8C8
CLOPTR=$F8B3
PUTPTR=$F8F1
PUTDIS=$F7D3
A2DEC=$B044
```

```
HEX=$A
BCD=$C
DEC1=$11
DEC2=$13
NUM=$C
OUT=$D
```

```
08AE A2 06          INPUT LDX ##6
08B0 A9 00          LDA #0
08B2 95 0A          SUB STA $A,X
08B4 CA            DEX
08B5 10 FB          BPL SUB
08B7 20 C8 F8       JSR OPNPTR
08BA A0 00          LDY ##0
08BC B9 00 08       SU LDA $800,Y
08BF 20 A9 09       JSR OUTPUT
08C2 C8            INY
08C3 C0 1B          CPY #27
08C5 D0 F5          BNE SU
08C7 20 2C F2       JSR READ
08CA 20 A9 09       JSR OUTPUT
08CD C9 59          CMP ##59
08CF F0 03          BEQ YES
08D1 20 B3 F8       JSR CLOPTR
08D4 A9 0A          YES LDA ##A
08D6 20 A9 09       JSR OUTPUT
08D9 B9 00 08       SU2 LDA $800,Y
08DC 20 A9 09       JSR OUTPUT
08DF C8            INY
08E0 C0 67          CPY #103
08E2 D0 F5          BNE SU2
```

sulla stampante non sarebbe stato scritto nulla anche se il programma si sarebbe svolto esattamente nello stesso modo, ponendo le domande solo sul display. Ad una risposta negativa il programma non fa altro che disabilitare la stampante (JSR CLOPTR).

La stampa é fatta inviando uno di seguito all'altro i primi 27 caratteri ASCII contenuti nella memoria a partire da \$800 sia al display che alla stampante con la routine OUTPUT (PAGE 13, in basso) che invia il carattere prima alla stampante e poi al display per mezzo delle routines dell'I/O ROM PUTPTR e PUTDIS; nel caso che il carattere da inviare é SA (stampa !), esso viene convertito prima di inviarlo al display in SI (che cancella il suo buffer e lo prepara a ricevere nuovi dati).

Questa differenza di comandi dati alla fine di una frase é dovuta alla differenza tra gli effetti dei control Character sulla stampante e sul display; per maggiori dettagli si rimanda alle appropriate sezioni nella prima par-

te di questo lavoro.

Riprendendo il programma vi é la stampa delle due frasi di "presen-
tazione" (CONTROLLO ON-OFF e LETTURA CON FOTOTRANSISTOR) sempre fatte

PAGE 0002

ADDR OBJECT SOURCE

```
00E4 20 20 F2 IN JSR READ
00E7 C9 0D CMP #D
00E9 F0 1C BEQ END
00EB 20 A9 09 JSR OUTPUT
00EE 20 44 B0 JSR A2DEC
00F1 A6 10 LDX BCD+4
00F3 85 10 STA BCD+4
00F5 A5 0F LDA BCD+3
00F7 86 0F STX BCD+3
00F9 A6 0E LDX BCD+2
00FB 85 0E STA BCD+2
00FD A5 0D LDA BCD+1
00FF 86 0D STX BCD+1
0901 85 0C STA BCD
0903 B0 DF BCS IN
0905 90 DD BCC IN
0907 A9 0A LDA #A
0909 20 A9 09 JSR OUTPUT
```

```
;
;
090C A9 00 CONVER LDA #0
090E 85 11 STA DEC1
0910 A9 01 LDA #1
0912 85 12 STA DEC1+1
0914 A2 05 LDX #5
0916 CA STAR DEX
0917 B5 0C MOL LDA BCD, X
0919 F0 11 BEQ ENDMOL
091B D6 0C DEC BCD, X
091D 18 CLC
091E A5 12 LDA DEC1+1
0920 65 0B ADC HEX+1
0922 85 0B STA HEX+1
0924 A5 11 LDA DEC1
0926 65 0A ADC HEX
0928 85 0A STA HEX
092A 90 EB BCC MOL
092C 06 12 ENDMOL ASL DEC1+1
092E 26 11 ROL DEC1
0930 A5 12 LDA DEC1+1
0932 0A ASL A
0933 85 14 STA DEC2+1
0935 A5 11 LDA DEC1
0937 2A ROL A
0938 85 13 STA DEC2
093A 06 14 ASL DEC2+1
093C 26 13 ROL DEC2
093E 18 CLC
093F A5 14 LDA DEC2+1
0941 65 12 ADC DEC1+1
0943 85 12 STA DEC1+1
0945 A5 13 LDA DEC2
0947 65 11 ADC DEC1
0949 85 11 STA DEC1
094B E0 00 CPX #0
094D D0 C7 BNE STAR
```

attraverso la stessa tecnica e della richiesta della velocità da stabilizzare espressa in giri al minuto: essa deve essere introdotta battendo le cifre decimali (fino a 65535) e completando con un RETURN.

La lettura delle singole cifre é realizzata con la routine I/O ROM READ (già usata nel programma 24 a cui si rimanda per maggiori dettagli); dopo essere state convertite da ASCII (progr. 19) sono poste nelle 5 locazioni consecutive a SC (dalla label IN per 16 istruzioni). A questo punto é necessario realizzare la conversione da BCD a binario usando il programma I6: a differenza di quello, il contatore per cui sono moltiplicate le cifre BCD (DECI) é però calcolato da programma moltiplicandolo per 10 ogni volta attraverso due scorrimenti a sinistra (x 4) una somma (x 5) ed un altro scor-

rimento a sinistra (x IO). Il risultato sarà contenuto nelle due locazioni HEX (= SA) e le cifre BCD sono azzerate.

Nei due esempi é possibile leggere nell'ultima riga, nei due primi bytes la traduzione in esadecimale della velocità desiderata: infatti 3000 = \$BB8 e 255 = \$FF.

Un altro dato importante per interpretare la velocità rilevata é il numero di fori presenti sul disco: questo é letto da tastiera in risposta alla domanda "NUMERO FORI = " ed é posto dopo essere stato convertito in BCD nella locazione NUM (= SC) come é visibile nel terzo byte riportato nell'esempio.

A questo punto sono poste le domande riguardo il tipo di output desiderato (numerico, su display o grafico su stampante): é possibile sceglierne uno solo, nessuno o entrambi. Al momento della scelta bisogna tener presente che, volendo un output numerico diventerà praticamente impossibile utilizzare il display su cui appariranno ogni secondo circa le velocità rilevate; se invece é richiesto un grafico, la stampante sarà occupata per 15-20 secondi.

La segnalazione di queste due opzioni é fatta con i bytes OUT(= \$D) in cui i 4 bits più significativi, se posti a 1, indicano che é richiesto l'output numerico ed i 4 meno significativi la richiesta del grafico.

Si tratta di un programma molto lungo rispetto al lavoro svolto (circa 130 istruzioni, oltre alla definizione dei caratteri ASCII delle frasi) e mostra chiaramente come sia stato utile avere già risolto i problemi di I/O e di conversione, cosa che ha semplificato il lavoro di programmazione che é rimasto comunque lungo.

PAGE 0003

ADDR	OBJECT	SOURCE	
094F	B9 00 08	SU3	LDA \$800, Y
0952	20 A9 09		JSR OUTPUT
0955	C8		INY
0956	C0 74		CPY #116
0958	D0 F5		BNE SU3
095A	20 2C F2		JSR READ
095D	20 A9 09		JSR OUTPUT
0960	20 44 B0		JSR A2DEC
0963	85 0C		STA NUM
0965	A9 0A		LDA #A
0967	20 A9 09		JSR OUTPUT
096A	B9 00 08	SU4	LDA \$800, Y
096D	20 A9 09		JSR OUTPUT
0970	C8		INY
0971	C0 8B		CPY #139
0973	D0 F5		BNE SU4
0975	20 2C F2		JSR READ
0978	20 A9 09		JSR OUTPUT
097B	C9 59		CMP #59
097D	D0 04		BNE NO
097F	A9 F0		LDA #F0
0981	85 0D		STA OUT
0983	A9 0A	NO	LDA #A
0985	20 A9 09		JSR OUTPUT
0988	B9 00 08	SU5	LDA \$800, Y
098B	20 A9 09		JSR OUTPUT
098E	C8		INY
098F	C0 AE		CPY #174
0991	D0 F5		BNE SU5
0993	20 2C F2		JSR READ
0996	20 A9 09		JSR OUTPUT
0999	C9 59		CMP #59
099B	D0 06		BNE NO2
099D	A9 0F		LDA #F
099F	05 0D		ORA OUT
09A1	85 0D		STA OUT
09A3	A9 0A	NO2	LDA #A
09A5	20 A9 09		JSR OUTPUT
09A8	00		BRK .. CONTINUA
09A9	20 F1 F8	OUTPUT	JSR PUTPTR
09AC	C9 0A		CMP #A
09AE	F0 04		BEQ DIS
09B0	20 D3 F7	DISRIT	JSR PUTDIS
09B3	60		RTS
09B4	A9 01	DIS	LDA #1
09B6	D0 F8		BNE DISRIT
			.END

ERRORS=0000

P A R T E S E C O N D A (Preparazione al funzionamento)

PAGE 0004

ADDR	OBJECT	SOURCE
		DIVIDE=#3
		RESTO=#0
		DIVISO=#6
		QUOZ=#9
		CONT=#B
		L=#C
		*=\$9B6
09B6	A9 00	DIVISI LDA #0
09B8	85 00	STA RESTO
09BA	85 01	STA RESTO+1
09BC	85 02	STA RESTO+2
09BE	85 0B	STA CONT
09C0	C6 0C	INI DEC L
09C2	A6 0C	LDX L
09C4	18	CLC
09C5	B5 03	SU1 LDA DIVIDE,X
09C7	2A	ROL A
09C8	95 03	STA DIVIDE,X
09CA	CA	DEX
09CB	10 F8	BPL SU1
09CD	A6 0C	LDX L
09CF	B5 00	SU2 LDA RESTO,X
09D1	2A	ROL A
09D2	95 00	STA RESTO,X
09D4	CA	DEX
09D5	10 F8	BPL SU2
09D7	B0 0F	BCS DIVER
09D9	A2 FF	LDX #FF
09DB	E8	SU3 INX
09DC	B5 00	LDA RESTO,X
09DE	D5 06	CMP DIVISO,X
09E0	D0 06	BNE DIVER
09E2	E4 0C	CPX L
09E4	D0 F5	BNE SU3
09E6	F0 02	BEQ UGUALI
09E8	90 0C	DIVER BCC SALTO
09EA	A6 0C	UGUALI LDX L
09EC	B5 00	SU4 LDA RESTO,X
09EE	F5 06	SBC DIVISO,X
09F0	95 00	STA RESTO,X
09F2	CA	DEX
09F3	10 F7	BPL SU4
09F5	38	SEC
09F6	A4 0C	SALTO LDY L
09F8	B1 09	SU5 LDA (QUOZ),Y
09FA	2A	ROL A
09FB	91 09	STA (QUOZ),Y
09FD	88	DEY
09FE	10 F8	BPL SU5
0A00	E6 0C	INC L
0A02	E6 0B	INC CONT
0A04	A5 0B	LDA CONT
0A06	4A	LSR A
0A07	4A	LSR A
0A08	4A	LSR A
0A09	C5 0C	CMP L
0A0B	D0 B3	BNE INI
0A0D	60	RTS

Il programma tra \$9B6 e \$AD7 elabora i dati letti per mezzo del programma precedente; inoltre predispone i timers e le due linee di handshake al funzionamento richiesto ed abilita gli interrupts.

PAGE 4 raccoglie una routine di divisione già vista precedentemente e che nelle sue linee essenziali rispecchia il progr. 9: essa é richiamata in seguito. Il rapporto é realizzato tra gli L bytes della locazione DIVIDE e quelli della locazione DIVISO, ponendo il quoziente all'indirizzo contenuto nei due bytes QUOZ.

E' inutile descrivere di nuovo l'algoritmo già presentato nella prima sezione di questo lavoro.

La seconda parte del programma comincia a \$AOE (PAGE 5).

Per poter controllare la velocità senza dover ogni volta calcolare l'inverso del tempo rilevato é, come già detto, più conveniente eseguire una volta per tutte questo rapporto per poi confrontare due tempi anziché due velocità. La prima operazione da fare é il calcolo del tempo che dovrà passare fra gli

PAGE 0005

ADDR OBJECT SOURCE

```

HEX=$A
NUM=$C
DISP=$E
NUM2=$F
TEMPO=$10
T22=$12
BCDOUT=$14
PRINT=$19
GIRI=$1B

T1C=$FFA4
T1L=$FFA6
T2=$FFA8
ACR=$FFAB
PCR=$FFAC
IER=$FFAE
READ=$F22C
PUTDIS=$F7D3

```

```

0A0E A9 00      INIZIO LDA #$0
0A10 A2 00      LDX #$0
0A12 95 00      UP1  STA 0,X
0A14 E8         INX
0A15 E0 0A      CPX #$A
0A17 D0 F9      BNE UP1
0A19 A5 0C      LDA NUM
0A1B 85 0F      STA NUM2

```

```

0A1D 18         UP2  CLC
0A1E A5 0A      LDA HEX
0A20 65 08      ADC DIVISO+2
0A22 85 08      STA DIVISO+2
0A24 A5 0B      LDA HEX+1
0A26 65 07      ADC DIVISO+1
0A28 85 07      STA DIVISO+1
0A2A A9 00      LDA #$0
0A2C 65 06      ADC DIVISO
0A2E 85 06      STA DIVISO
0A30 C6 0C      DEC NUM
0A32 D0 E9      BNE UP2

```

```

0A34 46 06      LSR DIVISO
0A36 66 07      ROR DIVISO+1
0A38 66 08      ROR DIVISO+2
0A3A 46 06      LSR DIVISO
0A3C 66 07      ROR DIVISO+1
0A3E 66 08      ROR DIVISO+2

```

```

0A40 A9 03      LDA #$3
0A42 85 0C      STA L
0A44 A9 0F      LDA #$F
0A46 85 09      STA QUOZ
0A48 A9 00      LDA #$0
0A4A 85 0A      STA QUOZ+1

```

; \$E4E1C0 = 15000000

```

0A4C A9 00      LDA #$C0
0A4E 85 05      STA DIVIDE+2
0A50 A9 E1      LDA #$E1
0A52 85 04      STA DIVIDE+1
0A54 A9 E4      LDA #$E4
0A56 85 03      STA DIVIDE
0A58 20 B6 09   JSR DIVISI

```

impulsi generati da due fori successivi, alla velocità da stabilizzare; esso è espresso dalla relazione:

$$T = \frac{1}{\frac{G_m}{60} \times N} = \frac{60}{G_m N}$$

dove:

Gm = giri al minuto

N = numero di fori

T = tempo espresso in secondi

Dato che a noi non interessa

il tempo espresso in μ S perché

questo è il tempo-base del timer

occorre moltiplicare il tutto

per 1.000.000 ottenendo:

$$T' = \frac{60000000}{G_m N} = \frac{15000000}{G_m N} \times 4 = \frac{15000000}{G_m N} \times 4 =$$

La separazione del rapporto è stata fatta per poter contenere il dividendo in tre bytes (15000000 = \$E4E1C0).

La prima operazione compiuta è il prodotto della velocità per il numero dei fori realizzata con un algoritmo a somme successive, dato che NUM è al massimo 9; la somma è memorizzata direttamente nelle locazioni del divisore.

Viene poi realizzata la divisione per 4 attraverso due spostamenti a destra dei tre bytes del divisore. Quindi, dopo aver definito il dividendo (\$E4E1C0),

PAGE 0006

ADDR OBJECT SOURCE

```

0A5B 38 SEC
0A5C A9 FF LDA #FF
0A5E E5 11 SBC TEMPO+1
0A60 85 11 STA TEMPO+1
0A62 A9 FF LDA #FF
0A64 E5 10 SBC TEMPO
0A66 85 10 STA TEMPO
0A68 A5 0F LDA TEMPO-1
0A6A D0 5B BNE ERROR
;
0A6C A9 D8 LDA #D8
0A6E 8D 2D 02 STA $22D
0A71 A9 0A LDA #A
0A73 8D 2E 02 STA $22E
0A76 A9 7F LDA #$7F
0A78 8D AE FF STA IER
;
0A7B A9 40 LDA #$40
0A7D 8D AB FF STA ACR
0A80 A9 FF LDA #FF
0A82 8D A6 FF STA T1L
0A85 8D A7 FF STA T1L+1
0A88 8D A4 FF STA T1C
0A8B 8D A5 FF STA T1C+1
0A8E 8D A8 FF STA T2
0A91 8D A9 FF STA T2+1
;
0A94 A9 F0 LDA #$F0
0A96 8D AC FF STA PCR
;
0A99 A9 00 LDA #0
0A9B 85 1B STA GIRI
0A9D 85 1C STA GIRI+1
0A9F 85 1D STA GIRI+2
0AA1 85 12 STA T22
0AA3 85 13 STA T22+1
0AA5 85 1A STA PRINT+1
0AA7 85 0E STA DISP
0AA9 A9 1E LDA #$1E
0AAB 85 19 STA PRINT
;
0AAD A9 F0 LDA #$F0
0AAF 8D AE FF STA IER
0AB2 58 CLI
0AB3 00 BRK
;
0AB4 4D 4F ERRORF .BYT 'MOTORE TROPPO
0 LENTO'
0AC7 A2 00 ERROR LDX #0
0AC9 BD B4 0A UP3 LDA ERRORF,X
0ACC 20 D3 F7 JSR PUTDIS
0ACF E8 INX
0AD0 E0 13 CPX #19
0AD2 D0 F5 BNE UP3
0AD4 20 2C F2 JSR READ
0AD7 00 BRK
END

```

ERRORS=0000

il numero di bytes su cui lavorare e l'indirizzo del quoziente (\$F) é chiamata la routine DIVISI.

A questo punto non si é però finito perché T2, che sarà usato per il controllo, conta alla rovescia, partendo dal numero caricato (in questo caso 65535 = \$FFFF) fino a 0. Quindi leggendo ad esempio 60000 significa non che sono trascorsi 60 mS, ma solamente 5,535. Anche in questo caso per snellire il programma di controllo vero e proprio conviene calcolare una sola volta 65535 - T' (inizio PAGE 6).

Da notare che, nel caso che il tempo calcolato sia superiore a 65 mS, il computer segnala che si tratta di una velocità troppo bassa da leggere con questo sistema (in basso, nella stessa pagina, con la routine ERROR). Poiché il controllo é realizzato tramite interrupt é necessario definire la routine di controllo il cui inizio é fissato a \$AD8, cioè alla fine di questo programma, caricando questo indirizzo nelle locazioni \$22D e \$22E.

L'abilitazione delle interrupts é data in seguito per permettere al programma di completare il

il suo lavoro senza essere interrotto.

Ora sono definiti i modi di funzionamento dei due timers: T1 é usato in modo da generare una continua serie di interrupts ogni $65535 \mu s$ per essere usato durante l'output (infatti ogni punto del grafico é ottenuto facendo una lettura della velocità ogni 65 ms e la visualizzazione della velocità avviene a distanza di 15 interrupts, cioè ogni secondo); T2 funziona in modo da generare un singolo intervallo ed é il mezzo con cui sono rilevati i tempi. Tutti e due i timers sono inizializzati a \$FFFF.

E' fatto poi partire il motore, ponendo a 1 la linea (CB2), e fatto riconoscere a CBI un cambiamento della linea collegata alla porta TTL da 0 a 1 (rising edge).

E' infine necessario azzerare le variabili usate dal programma, in particolare i tre bytes GIRI usati per contare il numero dei giri fatti dal motore.

A questo punto sono abilitati gli interrupts con l'istruzione CLI.

P A R T E T E R Z A (Controllo e output)

Questa parte del programma é gestita per mezzo degli interrupts; prima di passare al programma é meglio illustrare i tre blocchi del VIA che possono richiedere un'interruzione. Per una trattazione generale di come sono gestiti gli interrupts sia dall'AIM 65/40 che dall'6522 VIA si rimanda a quanto già detto.

CBI é collegata direttamente all'uscita della porta TTL che segnala il passaggio di un foro.

Questo é il segnale usato dal computer per rilevare la velocità: il programma richiamato da una transizione su questa linea richiama un programma che legge da T2 il tempo trascorso dall'ultimo impulso, lo interpreta e, di conseguenza, decide il livello di CB2; inoltre lo stesso programma ri-inizializza T2 a \$FFFF ed incrementa di 1 il numero di giri fatti.

T2 genera un interrupt quando arriva a 0: ciò significa che, per una

ragione qualsiasi, dall'ultimo impulso sono passati più di 65 mS. Questo fatto vuol dire che il motore si è fermato (e quindi non è più in grado di generare altri interrupts) o che comunque è sceso sotto la velocità minima misurabile che, nel caso in cui si hanno 8 fori, è di circa 115 circa al minuto.

L'unico effetto di questo interrupt è di porre ad I CB2 nella speranza di far ripartire il motore e segnalare che al prossimo interrupt proveniente da CB2 la velocità segnalata è priva di valore.

TI genera un interrupt ogni 65 mS e questa interruzione è usata per l'output sia numerico sul display che grafico. In pratica ad ogni interruzione generata da TI è calcolata la velocità (facendo l'inverso del complemento di T2,) fornendo un dato per il grafico ad alta risoluzione e, ogni 15 interruzioni, cioè ogni secondo circa, convertendo questa velocità in decimale per poterla visualizzare. Naturalmente questi passaggi sono saltati nel caso che, alle domande iniziali riguardanti l'output è stato risposto di no.

Passiamo ora ad analizzare i particolari di questo programma che, come già detto, parte da \$AD8, dopo che la gestione è stata passata attraverso il programma standard dell'AIM 6540.

Mentre questa parte è richiamata, bisogna ricordare che il computer funziona praticamente allo stesso modo ed è quindi possibile far girare in contemporanea altri programmi.

La prima istruzione eseguita è quella di salvare l'accumulatore nello stack per poterlo poi ripescarlo alla fine della gestione. Poi il programma ricerca quale parte del VIA ha chiesto l'interruzione per poter eseguire la procedura appropriata. Il controllo è effettuato tramite l'IFR: se il bit più significativo è a 0, lo USER VIA non ha causato interrupts e quindi il programma salta ad una routine che segnala un errore per mezzo della scritta "IRQ ERROR". Vengono quindi controllati i bits 6,5 e 4 corrispondenti rispettivamente a TI, T2 e CB2 e, trovato il flag a I il controllo è passato al rispettivo programma di gestione. Esaminiamoli ora singolarmente:

CBIIRQ è la parte richiamata quando è rilevata una transizione da 0 a 1 sulla linea CBI, cioè dal completamento di 1/8 di giro dell'al-

PAGE 0007

ADDR OBJECT SOURCE

IRQERR=\$F5A7
IFR=\$FFAD
PB=\$FFA0
VELOCI=\$EE
GRAFIC=\$1000

```

0AD0 40
0AD9 AD AD FF
0ADC 0A
0ADD 90 09
0ADF 0A
0AE0 B0 68
0AE2 0A
0AE3 B0 57
0AE5 0A
0AE6 B0 03
0AE8 4C A7 F5 ERR
0AEB A5 12
0AED F0 3F
0AEF AD A8 FF
0AF2 85 12
0AF4 AD A9 FF
0AF7 85 13
0AF9 A9 FF
0AFB 8D A8 FF
0AFE 8D A9 FF
0B01 E6 10
0B03 D0 06
0B05 E6 10
0B07 D0 02
0B09 E6 18
0B0B A5 13
0B0D C5 10
0B0F F0 10
0B11 90 07
0B13 A9 00
0B15 8D AC FF
0B18 D0 0F
0B1A A9 F0
0B1C 8D AC FF
0B1F D0 08
0B21 A5 12
0B23 C5 11
0B25 90 F3
0B27 B0 EA
0B29 AD A0 FF
0B2C 68
0B2D 40
0B2E A9 01
0B30 85 12
0B32 A9 FF
0B34 8D A8 FF
0B37 8D A9 FF
0B3A D0 DE
0AD0 40 PHA
0AD9 AD AD FF LDA IFR
0ADC 0A ASL A
0ADD 90 09 BCC ERR
0ADF 0A ASL A
0AE0 B0 68 BCS T1IRQ
0AE2 0A ASL A
0AE3 B0 57 BCS T2IRQ
0AE5 0A ASL A
0AE6 B0 03 BCS CB1IRQ
0AE8 4C A7 F5 ERR JMP IRQERR
0AEB A5 12 CB1IRQ LDA T22
0AED F0 3F BEQ OVERTI
0AEF AD A8 FF LDA T2
0AF2 85 12 STA T22
0AF4 AD A9 FF LDA T2+1
0AF7 85 13 STA T22+1
0AF9 A9 FF LDA #$FF
0AFB 8D A8 FF STA T2
0AFE 8D A9 FF STA T2+1
0B01 E6 10 INC GIRI+2
0B03 D0 06 BNE DOWN
0B05 E6 10 INC GIRI+1
0B07 D0 02 BNE DOWN
0B09 E6 18 INC GIRI
0B0B A5 13 DOWN LDA T22+1
0B0D C5 10 CMP TEMPO
0B0F F0 10 BEQ LESS
0B11 90 07 BCC ON
0B13 A9 00 OFF LDA #$D0
0B15 8D AC FF STA PCR
0B18 D0 0F BNE RITCB1
0B1A A9 F0 ON LDA #$F0
0B1C 8D AC FF STA PCR
0B1F D0 08 BNE RITCB1
0B21 A5 12 LESS LDA T22
0B23 C5 11 CMP TEMPO+1
0B25 90 F3 BCC ON
0B27 B0 EA BCS OFF
0B29 AD A0 FF RITCB1 LDA PB
0B2C 68 PLA
0B2D 40 RTI
0B2E A9 01 OVERTI LDA #1
0B30 85 12 STA T22
0B32 A9 FF LDA #$FF
0B34 8D A8 FF STA T2
0B37 8D A9 FF STA T2+1
0B3A D0 DE BNE ON

```

bero motore.

Tralasciando per il momento le prime due istruzioni che rilevano un overtime in T2 e come tali sono descritte con T2IRQ, passiamo alle altre.

Le quattro istruzioni seguenti sono usate per leggere il contenuto di T2 e memorizzarlo perché è su questo tempo che sarà calcolata la velocità. Il timer è quindi ricaricato con \$FFFF per poter riprendere il suo conto alla rovescia.

Il tempo contenuto in T2 è quello trascorso dall'ultima interruzione su CBI, espresso in μ S, e nel caso che la velocità fosse stabilizzata, corrisponderebbe al numero contenuto nei due bytes TEMPO (\$I0 e \$I1).

Il programma a questo punto incrementa i tre bytes GIRI che semplicemente contano il numero di giri fatti dal motore o, meglio, gli ottavi di giro: tenendo una velocità fissa di 3000 giri al minuto, è possibile contare per circa 11 ore fino a 16.777.215 giri.

Questo contatore non è usato dal programma ma, in un controllo industriale potrebbe servire ad es. per controllare lo spostamento di un braccio mecca-

PAGE 0008

ADDR	OBJECT	SOURCE	
0B3C	A9 00	T2IRQ	LDA #0
0B3E	85 12		STA T22
0B40	A9 F0		LDA #F0
0B42	8D AC FF		STA PCR
0B45	AD A8 FF		LDA T2
0B48	68		PLA
0B49	40		RTI
0B4A	A5 0D	T1IRQ	LDA OUT
0B4C	D0 03		BNE CONT
0B4E	4C 15 0C		JMP T1RIT
0B51	8A	CONT	TXA
0B52	48		PHA
0B53	98		TYA
0B54	48		PHA
0B55	A5 0D		LDA OUT
0B57	0A		ASL A
0B58	90 0B		BCC NODISP
0B5A	A5 0E		LDA DISP
0B5C	C9 10		CMP #16
0B5E	D0 03		BNE INCDIS
0B60	20 AF 0B		JSR OUTDIS
0B63	E6 0E	INCDIS	INC DISP
0B65	A5 0D	NODISP	LDA OUT
0B67	4A		LSR A
0B68	90 03		BCC CONT2
0B6A	4C 15 0C		JMP T1RIT
0B6D	20 8E 0B	CONT2	JSR VEL
0B70	A0 00		LDY #0
0B72	A5 EE		LDA #EE
0B74	F0 06		BEQ OK
0B76	A9 FF		LDA #FF
0B78	91 19		STA (PRINT),Y
0B7A	D0 04		BNE FINITO
0B7C	A5 EF	OK	LDA #EF
0B7E	91 19		STA (PRINT),Y
0B80	E6 19	FINITO	INC PRINT
0B82	A5 19		LDA PRINT
0B84	C9 EE		CMP #EE
0B86	D0 03		BNE CONT3
0B88	4C 1A 0C		JMP STAMPA
0B8B	4C 11 0C	CONT3	JMP REST
0B8E	38	VEL	SEC
0B8F	A9 FF		LDA #FF
0B91	85 05		STA DIVIDE+2
0B93	85 04		STA DIVIDE+1
0B95	E5 12		SBC T22
0B97	85 08		STA DIVISO+2
0B99	A9 FF		LDA #FF
0B9B	E5 13		SBC T22+1
0B9D	85 07		STA DIVISO+1
0B9F	A9 02		LDA #2
0BA1	85 0C		STA L
0BA3	A9 00		LDA #00
0BA5	85 0A		STA QUOZ+1
0BA7	A9 EE		LDA #EE
0BA9	85 09		STA QUOZ
0BAB	20 B6 09		JSR DIVISI
0BAE	60		RTS

nico collegato al motore.

Il nucleo di tutto il programma è costituito dalle 15 istruzioni seguenti (dalla label DOWN a RITCBI) che, data la velocità da stabilizzare e quella misurata decidono il livello di CB2.

Prima il confronto è realizzato sui due bytes più significativi e solo nel caso che sono uguali il confronto è fatto anche sugli altri due.

Il controllo è realizzato ponendo a 1 (con la routine ON) o a 0 (con la routine OFF) la linea CB2, a seconda che sia maggiore la velocità da controllare o quella rilevata.

Il programma si conclude con l'azzeramento del flag dell'IFR attraverso la lettura di PB, con il ricaricamento dell'accumulatore (PLA) e l'istruzione (RTI).

Il tempo di reazione del programma (cioè il tempo che intercorre dal momento in cui la velocità è rilevata a quando la linea CB2 assume il livello desiderato) è di meno di 150 μ S. Di questo tempo 86 μ S sono richiesti dalla routine dell'I/O ROM per controllare se la richiesta di interruzione proviene da uno dei due VIA del si-

stema: é possibile eliminarli ponendo l'indirizzo di inizio del programma di gestione a \$22B anziché a \$22D controllando subito se l'inerrupt proviene dallo USER VIA e passando solo in un secondo momento ed in caso di risposta negativa al controllo della stampante, del display e della tastiera. E' possibile diminuirlo ulteriormente evitando di contare il numero dei giri o facendolo dopo che é stato deciso il livello di CB2.

Il tempo di reazione non é comunque troppo alto perché il motore, per variare la sua velocità non richiede un tempo che é almeno uno o due ordini di grandezza superiore.

T2IRQ é la parte di programma richiamata quando T2 arriva a 0 cioè supera il tempo massimo che può misurare; in sostanza "accende" il motore che é sceso sotto limiti massimi misurabili ed inoltre segnala, ponendo a 0 la locazione T22, che la lettura realizzata dalla successiva interruzione causata da CBI non é valida: per questo, quando all'interno di CBIIRQ é rilevato questo, é eseguito un salto alla routine OVERTI che ri-inizializza T22 e pone a 1 CB2.

TIIRQ é la routine che, se richiesto, realizza l'output: infatti, nel caso che il byte OUT é 0 (cioé non é richiesto output) il programma ritorna immediatamente, dopo aver azzerato il flag con la lettura di TI.

Da notare che é necessario salvare anche i registri X e Y per poi ricaricarli, dato che sono modificati durante lo svolgimento di questa routine.

Si tratta in sostanza di due programmi, uno per il grafico, e l'altro per la visualizzazione numerica che, pur avendo alcune parti in comune si possono considerare indipendenti.

Esaminiamo prima come sono calcolati i valori per realizzare il grafico.

Ogni volta che questo programma é richiamato (ed é verificato che si vuole il grafico ad alta risoluzione) é prima calcolato l'inverso del complemento dell'ultimo tempo registrato da T2; questo valore, generalmente di un byte (in caso contrario é automaticamente forzato a \$FF cioè il numero massimo rappresentabile) é posto in memoria in

ADDR	OBJECT	SOURCE	
0BAF	A9 00	OUTDIS	LDA #00
0BB1	85 0E		STA DISP
0BB3	20 8E 0B		JSR VEL
0BB6	A0 05		LDY #5
0BB8	A9 00		LDA #0
0BBA	85 06		STA DIVISO
0BBC	85 07		STA DIVISO+1
0BBE	85 08		STA DIVISO+2
0BC0	85 03		STA DIVIDE
0BC2	85 04		STA DIVIDE+1
0BC4	A6 0F		LDX NUM2
0BC6	18	SU1	CLC
0BC7	A5 EF		LDA VELOCI+1
0BC9	65 08		ADC DIVISO+2
0BCB	85 08		STA DIVISO+2
0BCD	A5 EE		LDA VELOCI
0BCF	65 07		ADC DIVISO+1
0BD1	85 07		STA DIVISO+1
0BD3	CA		DEX
0BD4	D0 F0		BNE SU1
0BD6	A9 0A	CONVE	LDA #A
0BD8	85 05		STA DIVIDE+2
0BDA	20 06 09	SU2	JSR DIVISI
0BDD	18		CLC
0BDE	A5 02		LDA REST0+2
0BE0	69 30		ADC #30
0BE2	99 14 00		STA BCDOUT, Y
0BE5	A5 EF		LDA VELOCI+1
0BE7	85 08		STA DIVISO+2
0BE9	A5 EE		LDA VELOCI
0BED	85 07		STA DIVISO+1
0BED	A8		TAY
0BEE	D0 EA		BNE SU2
0BF0	A9 01		LDA #1
0BF2	20 D3 F7		JSR PUTDIS
0BF5	A5 14		LDA BCDOUT
0BF7	20 D3 F7		JSR PUTDIS
0BFA	A5 15		LDA BCDOUT+1
0BFC	20 D3 F7		JSR PUTDIS
0BFF	A5 16		LDA BCDOUT+2
0C01	20 D3 F7		JSR PUTDIS
0C04	A5 18		LDA BCDOUT+4
0C06	20 D3 F7		JSR PUTDIS
0C09	A5 18		LDA BCDOUT+4
0C0B	20 D3 F7		JSR PUTDIS
0C0E	4C 65 0B		JMP NODISP
0C11	68	REST	PLA
0C12	A8		TAY
0C13	68		PLA
0C14	AA		TAX
0C15	AD A4 FF	T1RIT	LDA T1
0C18	68		PLA
0C19	40		RTI
0C1A	58	STAMPA	CLI
0C1B	A9 CE		LDA #206
0C1D	85 05		STA NUM
0C1F	A9 D0		LDA #D0
0C21	8D AC FF		STA PCR
0C24	20 00 10		JSR GRAFIC
0C27	4C 11 0C		JMP REST

END

ERRORS=0000

una serie di bytes a partire da \$IE fino a \$ED per essere stampati dopo circa 15 secondi in modo grafico attraverso la routine GRAFIC già vista (PROGR. 22 e 23).

L'inversione è realizzata dalla routine VEL che richiama a sua volta DIVISI.

Il grafico è in grado di rappresentare velocità fino a 30000 giri al minuto avendo un solo foro: nel caso in cui si hanno 8 fori la velocità di qualche migliaia di giri è rappresentata nella fascia tra il 15° e il 50° punto del grafico poiché la risoluzione è di 64 giri al minuto.

L'output numerico è realizzato moltiplicando ogni 15 interrupts di TI (cioè ogni $15 \times 65535 \times 10^{-6} = 0,98$ s) il numero calcolato dalla routine VEL per il numero di fori (NUM2).

Dopo essere stato convertito da binario addecimale (CONVE) e poi in ASCII questo numero non è però mostrato in termini reali (cioè come giri al minuto) ma con un numero ad esso proporzionale.

La routine si conclude con il ricaricamento dell'accumulatore

e dei registri X e Y, la cancellazione del flag e l'istruzione RTI.

Concludendo queste descrizioni illustro ora i difetti ed i pregi di questo programma:

Difetti: a) il difetto principale risiede nel tipo di tensione impulsiva con cui il motore è controllato, difetto tipico dei controlli ON-OFF: ciò provoca una serie continua di accelerazioni e decelerazioni con conseguente instabilità. Questo problema può essere risolto aumentando l'inerzia meccanica del motore o usando un integratore (anche semplicemente un condensatore abbastanza grosso). Questo, però, può aumentare anche notevolmente il tempo di reazione perché, ad es. il condensatore deve scaricarsi prima che la tensione di comando sia sufficientemente bassa. Praticamente si cerca di realizzare un compromesso. b) La lunghezza e la complessità del programma rendono difficili successivi interventi di miglioramento da parte di chi non conosce a fondo tutti gli aspetti del programma. Purtroppo, lavorando in assembler questo problema è inevitabile a causa delle caratteristiche di questo linguaggio. Una soluzione sta nell'uso di linguaggi ad alto livello come è riportato in seguito. c) Il ritmo di controllo è funzione della velocità e ciò presenta qualche inconveniente quando si scende sotto 12200/300 giri al minuto; eventualmente si può intervenire aumentando il numero di fori. d) Un difetto particolare di questo programma è il fatto che la velocità non è rilevabile dall'utilizzatore direttamente; è però sufficiente moltiplicarla per una costante.

Pregi: a) possibilità di rilevare un vasto range di velocità (con 8 fori si va da 115 a 50000 e più giri al minuto) con un errore molto piccolo anche se in funzione della velocità. b) Possibilità di realizzare il controllo in contemporanea con altri programmi; c) Rilevazione della velocità senza una precedente analisi sperimentale, permettendo di realizzare il programma senza riferirsi ad un particolare sensore d) Utilizzazione di sole 2 linee, una per la lettura ed una per il controllo. e) Possibilità di funzionare direttamente su un sistema industriale caratterizzato dal non avere linguaggi evoluti: è questa la ragione per cui è stato realizzato completamente in linguaggio-macchina f) Tempo di reazione software dell'ordine di alcune decine di microsecondi.


```

5 PRINT:PRINT:PRINT
10 PRINT"CONTROLLO ON-OFF"
20 PRINT"LETTURA CON FOTOTRANSISTOR":PR
INT
27 PRINT"Versione BASIC":PRINT:PRINT
30 PRINT"Per fissare la velocita' R
UN 40"
32 PRINT"Grafico ad alta risoluzione R
UN 200"
33 PRINT"Letture velocita' R
UN 400"
34 PRINT"Ciclo utile R
UN 500"
35 PRINT
36 END
40 INPUT"VELOCITA' (GIRI AL MINUTO)":A
50 INPUT "NUMERO FORI":B
55 PRINT:PRINT:PRINT
60 TIME=1000000/((A/60)*B)
70 HIG=INT (TIME/256)
80 POKE 128,255-HIG
90 LOW=TIME-HIG*256
100 POKE 129,255-LOW
110 POKE 65440,255
120 POKE 65454,144
130 END
140 REM
150 REM
200 PRINT:PRINT:PRINT
205 PRINT"GRAFICO AD ALTA RISOLUZIONE"
210 PRINT"Usa la routine GRAFIC in asse
mblar":PRINT:PRINT
220 DIM OUT(255)
230 MAX=0
240 FOR Y=0 TO 255
260 TIME=65535-(PEEK(130)*256+PEEK(131)
)
270 OUT(Y)=1/TIME
280 IF OUT(Y)>MAX THEN MAX=OUT(Y)
290 NEXT Y
300 K=255/MAX
310 FOR X=0 TO 255
320 POKE 13568+X,INT(OUT(X)*K)
330 NEXT X
340 X=USR(A)
344 PRINT:PRINT
345 PRINT:PRINT
350 END
400 PRINT:PRINT:PRINT
405 PRINT"LETTURA DELLA VELOCITA'":PRIN
T
410 INPUT"NUMERO VALORI":NUM
420 INPUT"NUMERO FORI":B
421 PRINT
425 PRINT"VELOCITA'"
430 FOR I=1 TO NUM
435 PTR=0
440 FORMEAN=0 TO 10
450 PTR=PTR+(1000000/(65535-(PEEK(130)*
256+PEEK(131))))*B)
460 NEXT
470 PRINT INT(PTR/12)
480 NEXT
481 PRINT:PRINT
490 END
500 FOR I=0 TO 99
510 A=PEEK(65440)
520 IFA=255 THEN DUTY=DUTY+1
530 NEXT
535 PRINT:PRINT
540 PRINT"IL CICLO UTILE E' IL ";DUTY;"
X"

```

VERSIONE BASIC + ASSEMBLER

Osservando il programma di controllo appena visto, si vede come la maggior parte dello spazio in termini di lunghezza di programmazione è dato dalle parti che realizzano l'input/output (ad es. per richiedere le informazioni necessarie al funzionamento sono richieste 130 istruzioni) e la risoluzione di espressioni numeriche (ad es. è stato necessario definire una routine di divisione che occupa un centinaio di bytes e che richiede per essere richiamata almeno una decina di istruzioni). Del resto queste operazioni non richiedono necessariamente tempi di esecuzioni molto veloci; è infatti inutile avere programmi velocissimi che devono realizzare dei loop attendendo la risposta da tastiera. A questo va aggiunta una difficoltà di programmazione non indifferente.

Una soluzione a questi problemi si può trovare realizzando alcune parti del programma con un linguaggio evoluto, ad es. in BASIC. Ciò aggiunge una certa lentezza

za esecutiva (naturalmente se ci si riferisce alle velocità del linguaggio macchina) ma ha una brevità e semplicità di programmazione molto maggiore.

Naturalmente questo ripiego può essere usato solo nel momento in cui l'interlocutore é l'uomo o nei casi in cui il tempo di esecuzione non é determinante: infatti la lentezza del BASIC diventa eccessiva nel caso di controlli che devono essere effettuati in tempi di qualche centinaio di microsecondi, come ad es. il controllo vero e proprio della velocità. Inoltre, lavorando in BASIC, non é possibile una gestione tramite interrupt e si sarebbe quindi costretti a dedicare continuamente la macchina solo al controllo di una velocità.

Da queste considerazioni é nata l'idea di dividere il controllo in due parti: la prima é realizzata in assembler ed é gestita per mezzo di interrupts svolgendo l'operazione di controllo senza influenzare il funzionamento della macchina che in modo trascurabile. In assembler é pure realizzata la parte che realizza il grafico ad alta risoluzione, dopo essere stata richiamata da BASIC.

L'uso del linguaggio-macchina si é reso necessario per assicurare la sincronizzazione con la stampante.

In BASIC sono invece realizzate le parti che dal punto di vista della velocità non hanno particolari esigenze e che nel contempo si rivelano abbastanza lunghe o complesse.

Il listato BASIC completo é riportato nella pag. precedente: la prima cosa che si nota é la sua brevità rispetto alla versione assembler. A questo deve essere però aggiunto il programma in linguaggio-macchina che realizza il controllo e il grafico ad alta risoluzione.

Dandogli una scorsa veloce si vede la stampa di alcune frasi di presentazione, la richiesta della velocità da stabilizzare e del numero di fori, il calcolo del tempo tra due fori (linea 60), l'abilitazione degli interrupts da CBI e T2, e la scelta del modo di funzionamento di T2 (linea I10 e I20) A questo punto ad ogni ottavo di giro é richiamata la routine in assembler che legge la velocità e la stabilizza. Dato che essa occupa meno del 5% del tempo della CPU (é infatti richiamata "solo" 400-500 volte al

CONTROLLO ON-OFF
LETTURA CON FOTOTRANSISTOR

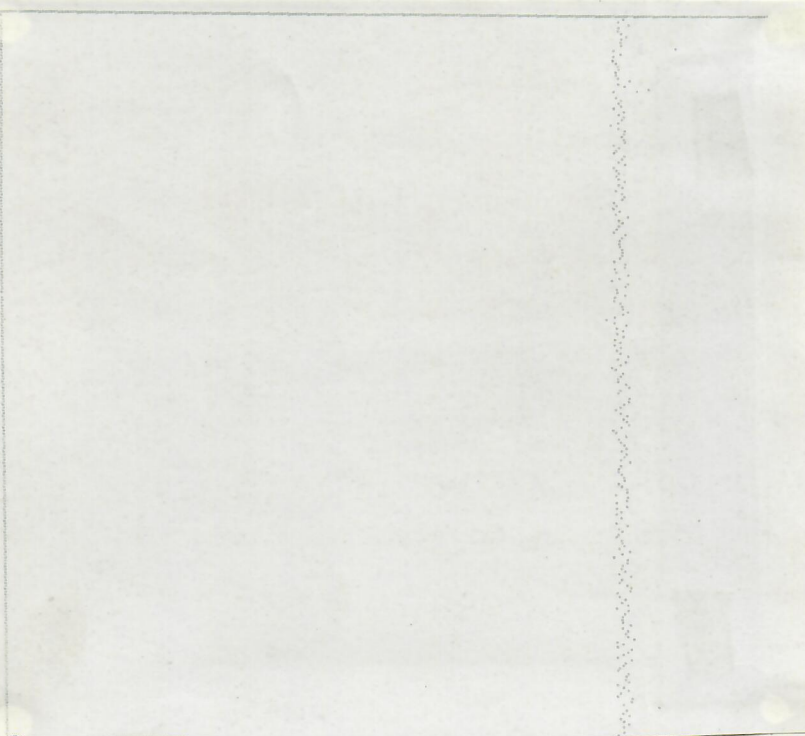
Versione BASIC

Per fissare la velocità	RUN 40
Grafico ad alta risoluzione	RUN 200
Lettura velocità	RUN 400
Ciclo utile	RUN 500

RUN40
VELOCITA' (GIRI AL MINUTO)? 2000
NUMERO FORI? 8

RUN200

GRAFICO AD ALTA RISOLUZIONE
Usa la routine GRAFIC in assembler



RUN400

LETTURA DELLA VELOCITA'

NUMERO VALORI? 5
NUMERO FORI? 8

VELOCITA'
2024
1982
1956
2022
2015

RUN500

IL CICLO UTILE E' IL 44 %

secondo) l'interprete BASIC può continuare a funzionare senza sensibili variazioni. Tutte le parti del programma funzionano in contemporanea con il controllo del motore ed in modo completamente asincrono rispetto ad esso. La parte seguente del programma è divisa in tre sezioni che possono essere richiamate in qualsiasi momento ed indipendentemente: la prima (dalla linea 200) realizza il grafico ad alta risoluzione della velocità: essa raccoglie per circa 10 secondi i valori della velocità leggendoli direttamente dalle locazioni in cui sono posti i due bytes di T2 dal programma in linguaggio-macchina e li mette in un vettore di 255 sezioni. Prima di porre questi valori nelle locazioni da cui saranno letti dal programma in linguaggio-macchina GRAFIC, essi sono moltiplicati per una costante K definita dal programma stesso volta per volta in funzione della velocità registrata. Essa, se da una parte impedisce che il grafico sia realizzato avendo ordinate fisse ed assolute (come nel caso del grafico assembler),

permette di ottenere^{la} massima risoluzione possibile per quella velocità: sevcosì, ad es. , la velocità massima rilevata é 4000, la costante é data da $4000/255 = 15,69...$ in modo che la differenza minima rappresentabile di due velocità é di circa 16 giri al minuto.

Se la velocità massima é di 500, la variazione rilevabile é di due giri al minuto.

Per la stampa del grafico é chiamata la già vista routine in linguaggio-macchina per mezzo dell'istruzione $X=USR(A)$.

Una seconda parte (dalla linea 400) permette di leggere una serie di velocità e di stamparle in termini di giri al minuto. Il numero di valori da stampare é deciso volta per volta dall'utente.

Una terza parte del programma (dalla linea 500) permette di ottenere il ciclo utile dell'onda quadra che alimenta il motore.

Ciò é fatto con 100 letture della locazione del VIA che comanda il motore ed incrementando di 1 una variabile se si trova che il motore é acceso. Questa variabile rappresenta con sufficiente approssimazione il ciclo utile reale.

INTERPRETAZIONE DEI DATI

Dopo aver visto come é realizzato il programma che controlla la velocità, é necessario verificare se esso funziona ed in che misura. Un primo controllo é effettuato per mezzo di un'osservazione diretta del motore, sentendo " ad orecchio" come il motore varia la sua velocità sotto il controllo del ^{computer} motore e come questa sia insensibile entro certi limiti alle variazioni della tensione di alimentazione. Naturalmente si tratta di una verifica molto approssimativa ma che permette una verifica immediata.

Per un controllo più preciso sono usati i due programmi del grafico ad alta risoluzione e della stampa della velocità nella versione BASIC. Sono stati riportati anche tre grafici realizzati in assembler, per poter confrontare i due diversi tipi di grafici e poterne dedurre vantaggi e svantaggi.

Cominciamo la nostra analisi proprio dal confronto di questi grafici di cui sono riportati tre esempi che si riferiscono alle stesse velocità di mille, milleseicento e duemilacinquecento giri al minuto. Fin da un primo esame emergono queste differenze.

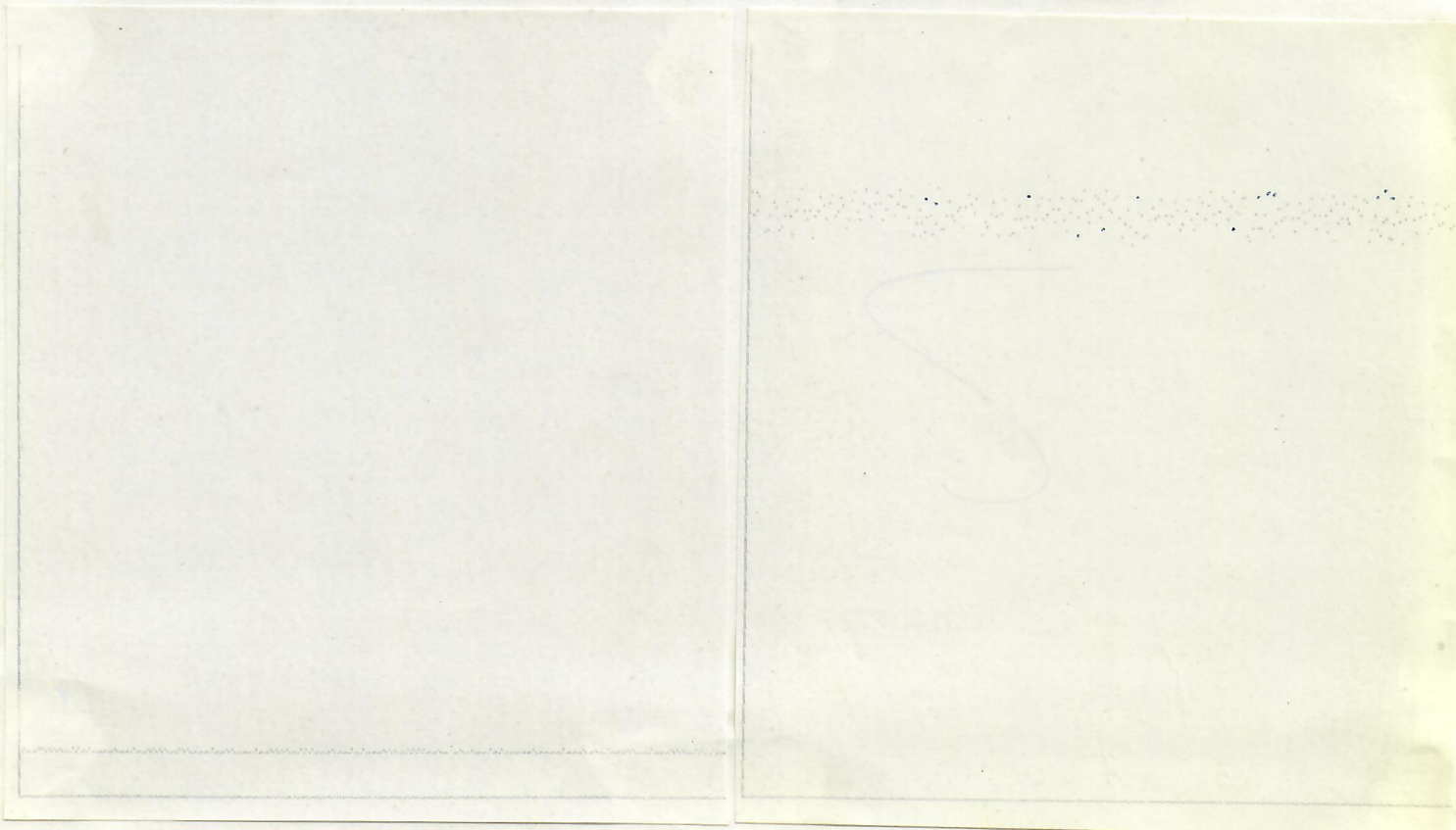
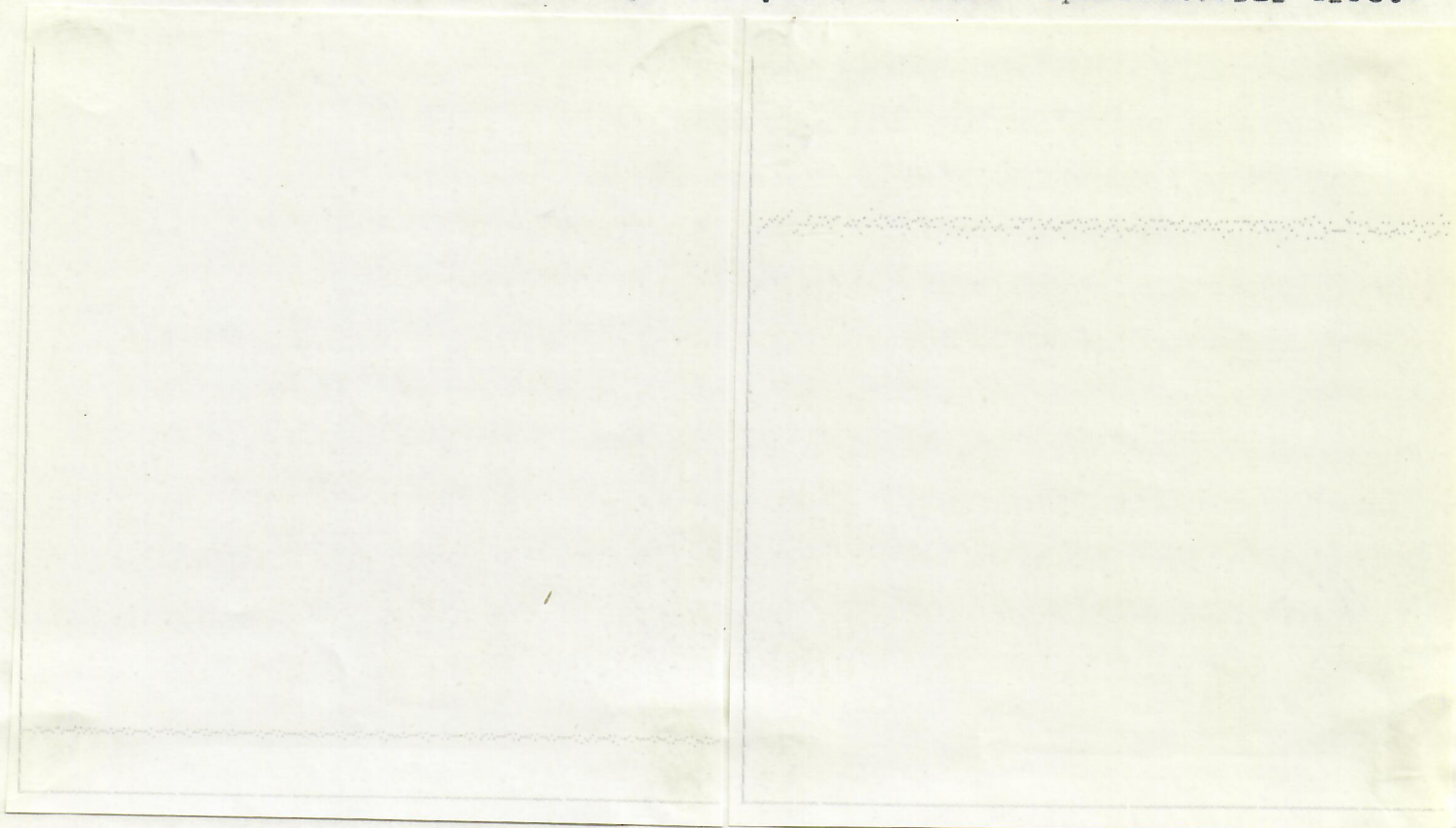
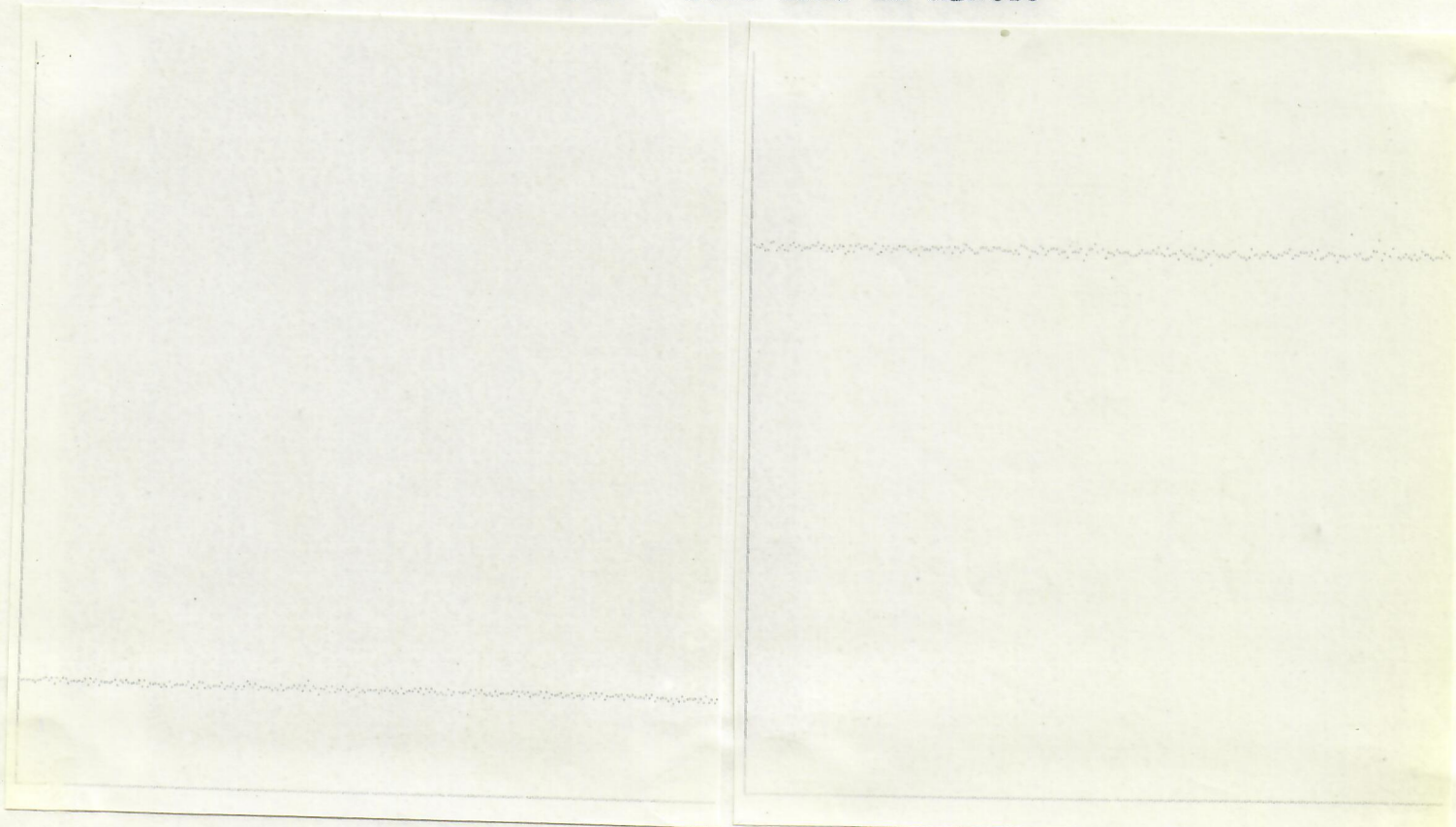


GRAFICO DELLA VELOCITA' : ASSEMBLER E BASIC - VELOCITA' 1000 GIRI

In particolare si nota come il grafico realizzato in assembler abbia l'ordinata fissa, in modo che due grafici siano confrontabili diret=



VELOCITA' 1600 GIRI AL MINUTO



VELOCITA' 2500 GIRI AL MINUTO

tamente senza ulteriori informazioni. L'incremento di un dot corrisponde alla variazione di 64 giri al minuto.

Questo fatto però impedisce di apprezzare le variazioni della velocità inferiori a 64 giri: per questo i punti che indicano sul grafico la velocità sono molto vicini tra di loro, generalmente sopra o sotto una linea mediana.

Il grafico BASIC (in BASIC é però realizzata la parte che raccoglie i dati; infatti la routine che realizza la stampa é sempre GRAFIC) permette invece di usare tutto lo spazio disponibile, amplificando la linea e facendo emergere le variazioni più piccole realizzabili ad una determinata velocità senza superare il limite massimo di 255: Ciò rende impossibile il confronto di due grafici senza sapere a che velocità sono stati realizzati dato che la linea appare sempre circa la stessa altezza, sia che la velocità é di 1000 o 2500 giri al minuto. Il grafico BASIC permette inoltre di valutare ad una prima occhiata l'errore relativo sulla velocità media valutando la larghezza della banda in cui i punti sono contenuti; ciò non

VEL.	ERR. MAX	RELATIVO
3000	1-2%	
2500	2%	
1600	3-4%	
1000	6%	
500	15%	
250	30%	

TAB. 5

é possibile con il grafico assembler, da cui emerge l'errore assoluto.

In seguito useremo sempre il grafico realizzato dal programma BASIC.

Un primo dato che emerge é il legame tra la velocità da controllare e la precisione con cui é stabilizzata: per verificare questo sono riportati i grafici che si riferiscono alle velocità di 3000, 500 e 250 giri al minuto; altre verifiche si possono

fare con i grafici già riportati. Da questi dati emerge come dato inconfutabile che l'errore relativo é tanto maggiore quanto minore é la velocità da controllare: da una lettura approssimativa di questi grafici può essere tratta la tabella 5 che rispecchia abbastanza bene questa tendenza.

Per un controllo più preciso può essere usato il programma che realizza la stampa delle varie velocità in termini numerici.

250 GIRI

500 GIRI

3000 GIRI

Di seguito sono riportati alcuni esempi. Dal loro esame approfondito si trae la tabella 6 in cui sono riportati alcuni valori che permettono di rilevare meglio l'andamento del fenomeno.

Nella prima colonna è riportata la velocità che dovrebbe essere imposta dal programma ; nella seconda la velocità media rilevata; nella terza lo scarto quadratico medio

$$\sigma = \sqrt{\frac{\sum_{i=0}^N (V_i - \bar{V})^2}{N}}$$

Dove: indica

N = numero di valori

V_i = Velocità

$\bar{V}$ = Velocità media.

RUN40
VELOCITA' (GIRI AL MINUTO)? 300
NUMERO FORI? 8

RUN400

LETTURA DELLA VELOCITA'

NUMERO VALORI? 5
NUMERO FORI? 8

VELOCITA'

312
349
386
393
388

RUN500

IL CICLO UTILE E' IL 32 %

RUN40
VELOCITA' (GIRI AL MINUTO)? 500
NUMERO FORI? 8

LETTURA DELLA VELOCITA'

NUMERO VALORI? 5
NUMERO FORI? 8

VELOCITA'

589
585
497
495
511

RUN500

IL CICLO UTILE E' IL 33 %

RUN40
VELOCITA' (GIRI AL MINUTO)? 3000
NUMERO FORI? 8

RUN400

LETTURA DELLA VELOCITA'

NUMERO VALORI? 7
NUMERO FORI? 8

VELOCITA'

3007
3021
2977
2966
3023
2993
2989

Essa indica l'intervallo entro cui si trovano il 68% dei valori rilevati in pratica è proporzionale all'errore assoluto; nella quarta colonna è riportato l'errore massimo relativo ottenuto dalla formula:

$$\Delta_{MAX} = \frac{V_{MAX} - V_{min}}{\bar{V}} 100$$

dove:

V_{MAX} = velocità massima

V_{min} = velocità minima

$\bar{V}$ = velocità media

Esso rispecchia abbastanza bene la precedente tabella 5.

Infine è riportato l'errore medio

relative espresso dalla formula:

$$\Delta_m = \frac{\sum_{i=0}^N |V_m - V_i|}{N \bar{V}} 100$$

V	V registrata	σ	$\Delta_{MAX} \%$	$\Delta \%$
300	331	31	26	10
500	519	35	16	4.5
1000	1006	33	6.6	3.1
2000	2000	27	3.1	1.2
3000	2997	20	1.4	0.6

TAB. 6

Da questa tabella emerge non solo che l'errore relativo diminuisce con l'aumento della velocità ma anche che l'errore assoluto segue questa strada, anche se con andamento più irregolare.

Una spiegazione a questo fenomeno è data dal fatto che, quando la velocità è più alta, i controlli sono più frequenti ed è quindi possibile un controllo più fine. Quando la velocità arriva a qualche giro al secondo, i controlli sono effettuati ogni qualche decina di millisecondo ed in questo tempo il motore cambia la sua velocità in modo abbastanza sensibile: per queste velocità il sistema di controllo si rivela inefficiente.

Un altro dato che emerge è il fatto che, alle basse velocità, la velocità rilevata è piuttosto più alta di quella impostata: questo può essere spiegato con il fatto che l'accelerazione è maggiore della decelerazione e pertanto, quando il motore aumenta la sua velocità, lo fa abbastanza velocemente, superando quella impostata, anche abbastanza, prima di essere spento.

Il fenomeno contrario sembra avvenire alle alte velocità, dove lo spunto del motore è minore; vi è però un forte ridimensionamento, dato che i controlli sono molto più frequenti.

Un ultimo dato interessante è la velocità massima che il motore può raggiungere. Esso è ottenuto imponendo una velocità da controllare sicuramente superiore ad essa (nell'esempio 50000): il computer è in questo caso obbligato a mantenere sempre alta la linea CB2. Nel caso che la tensione di alimentazione è di 9 V, la velocità massima rilevata è di circa 3650 giri al minuto.