UNIVERSITA' DEGLI STUDI DI PISA
ISTITUTO DI SCIENZE DELL'INFORMAZIONE

A. Grasselli - G. Pacini

IL CALCOLATORE DIDATTICO "CANE"

MANUALE DI RIFERIMENTO

Note per il Corso di
Teoria ed Applicazioni delle Macchine Calcolatrici

anno accademico 1974-1975

Introduzione

Il calcolatore CANE (Calcolatore Automatico Numerico Educativo), descritto in questa nota, è un calcolatore ideale, che è stato definito per insegnare i concetti elementari della programmazione in codice macchina e della architettura dei calcolatori; in particolare, il CANE viene impiegato nel corso di "Teoria ed applicazioni delle macchine calcolatrici" di Scienze dell'Informazione. Un simulatore del CANE (programma SIMULCANE) esiste per l'IBM 370/155: le sue modalità di impiego sono descritte in questa nota.

Perchè insegnare la programmazione in codice macchina? Non vi è dubbio alcuno che, prima o poi, gli studenti del corso di laurea in Scienze dell'Informazione debbano acquisire una certa esperienza in codice macchina. E' tutto sommato preferibile che questa esperienza venga il più presto possibile, perchè in caso contrario (se, cioè, il primo contatto con il calcolatore avviene al livello di un linguaggio simbolico) rimane una larga ed insoddisfacente zona d'ombra, che rende oltretutto difficile, se non addirittura impossibile, un discorso anche solo introduttivo sull'architettura delle macchine.

L'insegnamento introduttivo della programmazione in codice macchine dovrebbe rispettare le seguenti condizioni:

a) il calcolatore scelto dovrebbe essere "abbastanza reale";

b) si dovrebbe cercare di non terrorizzare il principiante con dettagli inessenziali al livello di una prima esposizione.

La prima condizione consiglierebbe l'adozione di un calcolatore reale, se non fosse per le considerazioni che seguono. In tutti i casi, è a nostro parere sconsigliabile impiegare un calcolatore ideale molto semplice (dieci istruzioni, per esempio), perchè la semplicità della struttura scelta si riflette invariabilmente in una artificiosa macchinosità dei programmi: con poche istruzioni nel codice operativo, anche il più semplice dei programmi (somma di n numeri!) richiede parecchie istruzioni, memorizzazione di costanti, ecc. Il rischio è quello di presentare allo studente una visione distorta della programmazione in codice macchina.

D'altra parte, in un calcolatore reale, le idiosincrasie delle istruzioni di entrata-uscita scoraggiano di solito anche l'iniziato; in più, le manipolazioni alla consolle presentano una certa difficoltà; infine, di solito la struttura ed il codice operativo, frutto di compromessi di ogni tipo, violano talvolta la condizione b) di cui sopra.

Si è quindi ritenuto preferibile definire un calcolatore come il CANE, con un codice operativo ricco di possibilità ma di facile comprensione. Anche la struttura del CANE è tuttavia frutto di compromessi: la memoria è troppo piccola, mancano le istruzioni in virgola mobile, e le unità periferiche sono ridotte al lettore di schede ed alla stampante. A proposito delle dimensioni della memoria, noteremo che esse sono dettate da un lato dalla lunghezza della parola (a sua volta scelta in 16 bit perchè è allora possibile scrivere una istruzione mediante 6 cifre ottali), e dall'altro dal desiderio di codificare ognuno dei tre campi dell'istruzione mediante una o più cifre ottali: si noti che al livello di esercizi di programmazione elementare le dimensioni della memoria possono senz'altro essere giudicate adeguate.

La programmazione in codice macchina, a macchina "nuda", comporta però, almeno nel caso del CANE, la seguente difficoltà: i programmi dovrebbero essere redatti e introdotti in memoria in binario. E' discutibile se una o due esperienze di questo tipo possano risultare positive, ma è sicuro che continuare a imporre la programmazione in binario porterebbe, in brevissimo tempo, ad una completa diserzione dalla programmazione in CANE.

Tenendo presente questa difficoltà si è pensato bene di corredare il CANE di un pacchetto-software (un minisistema operativo) che permetta di accedere alla macchina in modo meno precario. Sostanzialmente, il minisistema dà la possibilità di presentare programmi redatti in ottale (quindi ancora in linguaggio macchina) piuttosto che in binario, e facilita la lettura e stampa di dati numerici.

L'introduzione di questo minisistema operativo ha due scopi principali: primo, lasciare l'utente a contatto con la macchina sollevandolo però dalla gravosa schiavitù del binario; secondo, fare in modo che il mezzo di calcolo venga concepito, fin dall'inizio, come costituito dalla macchina intesa in senso propriamente fisico (hardware) più un insieme di programmi aventi lo scopo di facilitare l'uso della macchina stessa (software).

Il software del CANE è di estrema semplicità, molto lontano dalla complessità dei sistemi software reali. Questo fatto ha comunque il vantaggio che il pacchetto software può essere illustrato interamente anche al principiante. In questo modo, l'utente del CANE ha completa conoscenza e padronanza di tutti i mezzi di calcolo a sua disposizione.

PARTE PRIMA

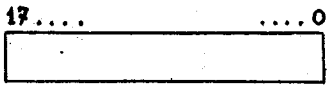
1.1 - La struttura del CANE.

L'architettura del CANE è illustrata dallo schema di fig. 1.

La memoria consta di 512 celle ciascuna costituita da 18 bit. Il blocco delle celle di memoria sarà indicato simbolicamente con

$$Mw[511:0] < 17:0 > ,$$

cioè, 512 celle numerate da 0 a 511, ciascuna costituita da 18 bit, numerati da 0 a 17 come illustrato dal seguente schema:



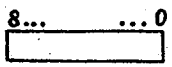
L'unità di elaborazione possiede due registri operativi (accumulatori) rA ed rB, ciascuno costituito da 18 bit :

$$rA < 17:0 > \quad e \quad rB < 17:0 > \ ;$$

rB funge da estensione di rA nelle operazioni di moltiplicazione e divisione. L'unità di elaborazione possiede inoltre 7 registri indice, indicati simbolicamente con:

$$rX[1:7] < 8:0 > ,$$

ciascuno costituito da 9 bit, numerati da 0 ad 8 come illustrato nel seguente schema



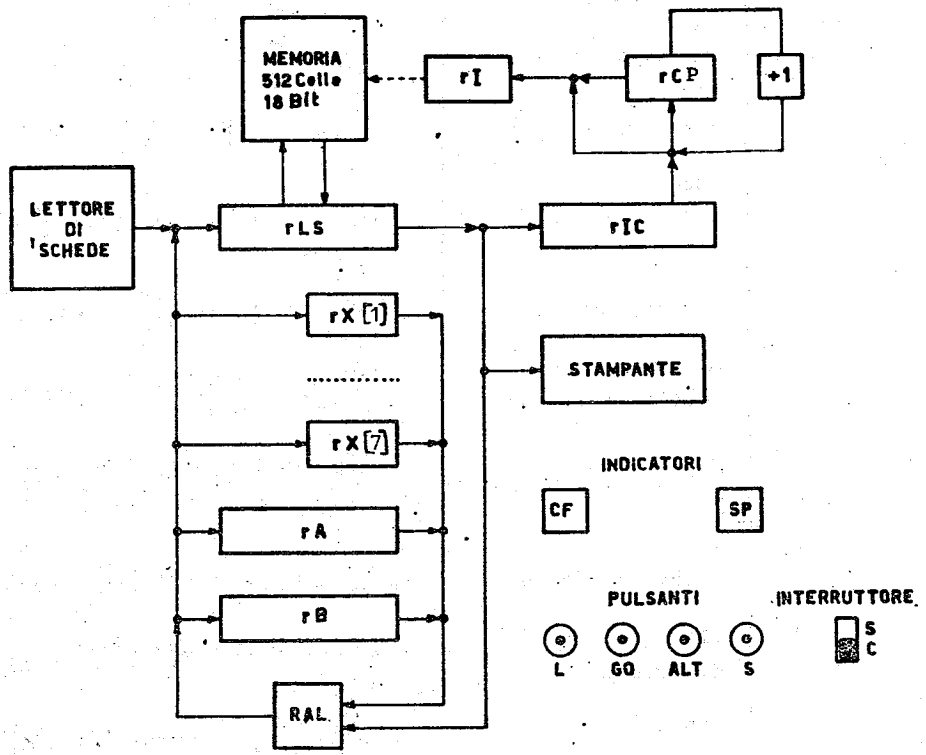
La macchina possiede infine un indicatore di confronto costiuito da 2 bit

$$CF <1:2>$$

e un indicatore di supero capacità dei registri SP costituito da un unico bit.

Al CANE sono collegate due sole unità periferiche: un lettore di schede ed una stampatrice.

La consolle del CANE possiede quattro pulsanti, denominati L (lettura e avvio), GO (avvio), ALT, S(singolo) ed un interruttore a due posizioni SC (singolo continuo).



rLS	registro di lettura e scrittura	rA	registro accumulatore
rI	" indirizzamento	rB	" moltiplicatore-quoziente
rCP	" contatore istruzioni	rX[1],...,rX[7]	registri indice
rIC	" istruzione corrente	RAL	rete aritmetico-logica
CF	indicatore (2 bit) di confronto		
SP	indicatore (1 bit) di supero		

Fig. 1 - Architettura del CANE.

I numero negativi sono rappresentati in complemento a 2.

I registri indice funzionano modulo 1000_8 : ad esempio, se nel calcolo di un indirizzo effettivo è rX[J]=770 e I=333, si ottiene

$$rX[J]+I = 770+333=323.$$

1.2. Le istruzioni

Tutte le istruzioni del CANE occupano una parola; il profilo dell'istruzione è il seguente:

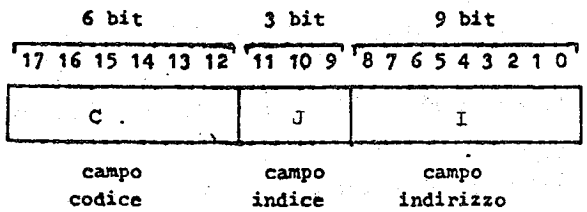


fig. 2

Il contenuto del campo codice, campo indice e campo indirizzo, e, più in generale, contenuti di celle, registri, indirizzi, ecc., verranno nel seguito indicati in ottale, senza menzione esplicita della base.

Nel seguito il contenuto di una cella di memoria (di indirizzo I) verrà indicato da Mw [I] , il contenuto di un registro indice da rX [J] , quello degli altri registri dal nome del registro. Ad esempio Mw [I+rX [4]] indicherà il contenuto della cella il cui indirizzo viene ottenuto sommando I al contenuto del registro indice rX [4]. Inoltre Mw [I] <i> indicherà l'i-esimo bit della cella di indirizzo I (vedi fig.2) e Mw [J] <i:j> il campo costituito dai bit tra i e j (compresi). Una notazione analoga varrà per i registri, ad es. rA <5:0> indicherà gli ultimi sei bit di rA (verso destra).

1.3 Istruzioni che si riferiscono alla memoria

Queste istruzioni trasferiscono il contenuto di una cella di memoria in uno dei registri (TRA,TRAN,TRB,TRX), memorizzano il contenuto di un registro (MEA,MEB,MEX) od una costante (MEZ), eseguono operazioni aritmetiche (ADA,SOA,ADB,SOB,MOL,DIV) o logiche (COP,AND,OR,ORX) per le quali uno degli operandi è in memoria.

codice ottale	mnemonico	operazione
01	TRA trasf. in rA	rA ← Mw[I+rX[J]]
02	TRAN trasf. in rA negativo	rA ← − Mw[I+rX[J]]
05	TRB trasf. in rB	rB ← Mw[I+rX[J]]
06	TRX trasf. in rXj	rX[J] ← Mw[I]<8:0>
10	MEA memorizzazione di rA	Mw[I+rX[J]] ← rA
11	MEB memorizzazione di rB	Mw[I+rX[J]] ← rB
12	MEZ memorizzazione di 0	Mw[I+rX[J]] ← 0
13	MEX memorizzazione di rXj	Mw[I]<8:0> ← rX[J] , vedi nota a)
14	ADA addizione ad rA	rA ← rA + Mw[I+rX[J]] , vedi nota b)

./.

codice ottale	mnemonico	operazione
15	SOA sottrazione da rA	rA ← rA − Mw [I+rX [J]] , vedi nota b)
20	ADB addizione ad rB	rB ← rB+Mw [I+rX [J]] , vedi nota b)
21	SOB sottrazione da rB	rB ← rB−Mw [I+rX [J]] , vedi nota b)
22	ADX addizione ad rXj	{ rX[J] ← rX J +Mw [I]<8:0> ; se rX[J]=0, rC ← rC+2 } vedi nota c)
23	SOX sottrazione da rXj	{ rX[J] ← rX[J] −Mw [I]<8:0> ; se (rX[J]=0) rC ← rC+2 } vedi nota c)
26	MOL moltiplicazione	rA,rB ← rB x Mw[I+rX[J]] , vedi nota d)
27	DIV divisione	{ rB ← rA,rB/Mw [I+rX [J]] ; rA ← resto } vedi nota e)
30	COP complemento	Mw [I+rX[J]] <i> ← ¬Mw [I+rX[J]] <i> , i=17,...,0 vedi nota f)
31	AND prodotto logico	rA <i> ← rA<i> ∧ Mw [I+rX[J]] <i> , i=17,...,0 vedi nota f)
32	OR somma logica	rA<i> ← rA<i> ∨ Mw[I+rX[J]]<i> , i=17,...,0 vedi nota f)
33	ORX somma mod 2	rA<i> ← rA<i> ⊕ Mw [I+rX[J]] <i> , i=17,...,0 vedi nota f)

Note sulle istruzioni precedenti:

a) MEX: l'esecuzione dell'istruzione lascia inviarito Mw[I]<17:9>.

b) ADA, SOA, ADB, SOB: se nella esecuzione di queste istruzioni si genera un supero di capacità dell'accumulatore (rA o rB) viene "settato" l'indicatore di supero SP: SP <— 1;

c) ADX, SOX: se, dopo l'esecuzione di queste istruzioni, il registro indice interessato contiene zero, viene eseguito uno skip, cioè viene saltata l'istruzione successiva: rCP <— rCP+ 2;

d) MOL: questa istruzione moltiplica il numero contenuto in rB per il numero in I+rX[J]; la metà più significativa del prodotto va in rA, e la metà meno significativa in rB;

e) DIV: questa istruzione divide il numero rA x 2^{17}+rB per il numero nella cella di indirizzo I+rX[J]; il quoziente va in rB, ed il resto va in rA.

Se |rA| ≥ | Mw[I+rX[J]]|, la divisione non viene eseguita, e viene settato l'indicatore di supero, SP <— 1;

f) I simboli ¬ , ∧ , ∨, indicano rispettivamente le operazioni logiche di negazione, prodotto, somma. Il simbolo ⊕ indica la somma modulo 2. Le operazioni vengono effettuate separatamente per ogni bit o coppia di bit.

1.4 Istruzioni dirette

Queste istruzioni trasferiscono in un registro, o sommano e sottraggono al contenuto di un registro il contenuto del campo indirizzo dell'istruzione stessa (TDX, ADDX,SODX), o il contenuto del campo indirizzo più il contenuto del registro indice specificato (TDA,TDAN,ADDA,SODA).

codice ottale	mnemonico	operazione
03	TDA trasf. diretto in rA	rA ← I+rX[J]
04	TDAN trasf. diretto in rA negativo	rA ← - (I+rX[J])
07	TDX trasf. diretto in rXj	rX[J] ←I
16	ADDA somma diretta ad rA	rA ← rA + I+rX[J] , vedi nota a)
17	SODA sottrazione diretta ad rA	rA ← rA - (I+rX[J]) ,vedi nota a)
24	ADDX somma diretta ad rXj	{ rX[J] ← rX[J] +I se (rX[J] =0) ,rCP ← rCP+2 } vedi nota b)
25	SODX sottrazione diretta ad rXj	{ rX[J] ← rX[J] -I se (rX[J] =0) ,rCP ← rCP+2 } vedi nota b)

Note sulle istruzioni precedenti:

a) ADDA, SODA: il caso di supero di capacità viene trattato come illustrato nel paragrafo precedente;

b) ADDX, SODX: se dopo l'operazione rX[J]=0, viene effettuato lo skip.

1.5 Istruzioni di confronto

Lo scopo di queste istruzioni è quello di settare l'indicatore di confronto CF in base al confronto fra il contenuto di un registro ed il contenuto di una cella di memoria. Gli indicatori di confronto vengono interrogati dalle istruzioni di salto (vedi 1.4). Nella tabella seguente, rY indica rA, oppure rB.

codice ottale	mnemonico	operazione
34	CFA confronto con rA	CF ← 10 se rY > Mw[I+rX[J]] CF ← 11 se rY = Mw[I+rX[J]] CF ← 01 se rY < Mw[I+rX[J]]
35	CFB confronto con rB	
36	CFX confronto con rXj	CF ← 10 se rX[J] > Mw[I]<8:0> CF ← 11 se rX[J] = Mw[I]<8:0> CF ← 01 se rX[J] < Mw[I]<8:0>
37	CFXD confronto con rXj diretto	CF ← 10 se rX[J] > I CF ← 11 se rX[J] = I CF ← 01 se rX[J] < I

1.6 Istruzioni di salto

Queste istruzioni rimandano all'esecuzione dell'istruzione in I+rX[J] oppure I, incondizionatamente (SLT, SUB), o condizionatamente agli indicatori (SSP, SMIN, SMAG, SUG) o ai segni dei registri rA ed rB (SAN, SAP, SAZ, SBN, SBP, SBZ).

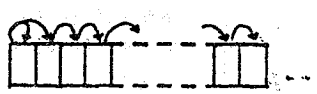
codice ottale	mnemonico	operazione
40	SLT salto	rC ← I+rX[J]
41	SUB salto a sottoprogramma	rX[J] ← rCP (rCP incrementato nella fase di fetch) rCP ← I
42	SSP salto se supero	se (SP = 1) rCP ← I+rX[J] SP ← 0
43	SMIN salto se minore	se CF = 01 / 10 / 11, rCP ← I+rX[J] vedi nota
44	SMAG salto se maggiore	
45	SUG salto se uguale	
46	SAN salto se rA negativo	se rA < 0 / > 0 / = 0, rCP ← I+rX[J]
47	SAP salto se rA positivo	
50	SAZ salto se rA zero	

codice ottale	mnemonico	operazione
51	SBN salto se rB negativo	se rB $\left\{ \begin{array}{l} <0 \\ >0 \\ =0 \end{array} \right\}$, rCP ← I+rX[J]
52	SBP salto se rB positivo	
53	SBZ salto se rB zero	

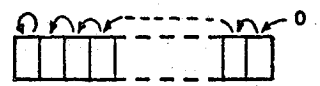
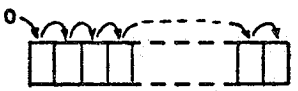
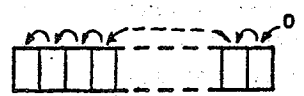
Nota: dopo l'esecuzione delle istruzioni SMIN, SMAG, SUG, lo stato dell'indicatore CF non è variato.

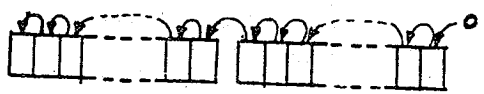
1.7 Istruzioni di spostamento

Queste istruzioni spostano di I+rX[J] posti il contenuto dei registri rA ed rB.

codice ottale	mnemonico	operazione
54	AVD rA aritmetico verso destra	 rA oppure rB ripetuto I+rX[J] volte
60	BVD rB aritmetico verso destra	

./.

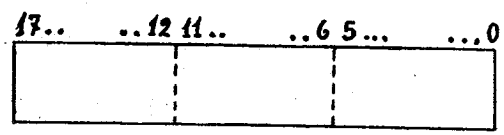
codice ottale	mnemonico	operazione
55	AVS rA aritmetico verso sinistra	 rA oppure rB ripetuto I+rX[J] volte
61	BVS rB aritmetico verso sinistra	
56	ALD rA logico verso destra	 rA ripetuto I+rX[J] volte
57	ALS rA logico verso sinistra	 rA ripetuto I+rX[J] volte
62	ABLD rA, rB logico verso destra	 rA rB ripetuto I+rX[J] volte

codice ottale	mnemonico	operazione
63	ABLS rA,rB logico verso sinistra	 rA rB ripetuto I+rX[J] volte

1.8 Istruzioni speciali

Queste istruzioni muovono caratteri fra rA e le celle di memoria (TCA,MCA), oppure trasformano rappresentazioni binarie in decimale e viceversa (CAR, NUM).

Si tenga presente che, per quanto riguarda la memorizzazione di caratteri alfanumerici, le celle o i registri del CANE sono da considerare suddivise in tre campi, come illustrato dal seguente schema:



Ciascuno dei tre campi può contenere la codifica di un carattere alfanumerico (vedi appendice I).

codice ottale	mnemonico	operazione
64	TCA trasferimento carattere	rA ← 0 se rX[1]⟨1:0⟩ = 11: rA⟨5:0⟩ ← Mw[I+rX[J]]⟨5:0⟩ 10: rA⟨5:0⟩ ← Mw[I+rX[J]]⟨11:6⟩ 01: rA⟨5:0⟩ ← Mw[I+rX[J]]⟨17:12⟩ 00: nessuna operazione
65	MCA memorizzazione carattere	se rX[1]⟨1:0⟩ = 11: Mw[I+rX[J]]⟨5:0⟩ ← rA⟨5:0⟩ 10: Mw[I+rX[J]]⟨11:6⟩ ← rA⟨5:0⟩ 01: Mw[I+rX[J]]⟨17:12⟩ ← rA⟨5:0⟩ 00: Mw[I+rX[J]] ← tre spazi
66	CAR conversione in caratteri	il valore assoluto del numero in Mw[I+rX[J]], convertito in decimale, viene scritto sotto forma di 6 cifre decimali in codice caratteri in rA (3 cifre più significative) ed rB (3 cifre meno significative).
67	NUM conversione in numero binario	le 6 cifre decimali in codice caratteri contenute in rA ed rB vengono trattate come numero positivo, che viene convertito in binario memorizzato in Mw[I+rX[J]]. Se la conversione dà luogo a supero di capacità di rA, SP ← 1; ad operazione eseguita, se vi è stato supero, Mw[I+rX[J]] contiene le 18 cifre binarie meno significative.

Si noti che la istruzione MCA modifica soltanto il gruppo di sei bit precisati da rX[1]⟨1:0⟩.

1.9 Istruzioni di entrata-uscita

L'unità di entrata del CANE è un lettore di schede a 72 colonne, e l'unità di uscita una stampante a 120 colonne. Vi sono due modalità di entrata: binaria, e alfanumerica. Nella modalità binaria, la colonna generica può essere "bianca", oppure contenere uno 0 oppure un 1: tutti gli altri caratteri sono considerati errore, e la macchina si ferma sulla istruzione di lettura interessata. Lo spazio e lo 0 hanno uguale significato. Nella modalità binaria, il contenuto della scheda (72 colonne) viene letto in 4 celle successive: le prime 18 colonne nella prima delle 4 celle, ..., le ultime 18 colonne nell'ultima delle 4 celle.

Il codice di rappresentazione dei caratteri alfanumerici all'interno del CANE è quello della Tabella 1 dell'Appendice I. Una parola del CANE ospita quindi 3 caratteri. Nella modalità di lettura alfanumerica, il contenuto della scheda (72 colonne) viene letto in 24 celle successive: le prime 3 colonne nella prima cella, ..., le ultime 3 colonne nella ventiquattresima.

Vi è una sola modalità di uscita (alfanumerica), nella quale 120 caratteri, memorizzati in un gruppo di 40 celle successive, costituiscono una riga di stampa.

Le operazioni di entrata-uscita vengono iniziate dalla unità di controllo, ma completate dalla unità di entrata e di uscita. Perciò, dopo aver iniziato una operazione di entrata o di uscita, il calcolatore continua con le successive istruzioni del programma anche se l'esecuzione dell'operazione non è completata. Naturalmente, se una nuova istruzione di entrata-uscita viene incontrata nel programma, il calcolatore si ferma fino a che l'esecuzione dell'operazione precedente non è stata completata. Apposite istruzioni di controllo permettono di verificare il completamento della operazione di entrata e della operazione di uscita.

codice ottale	mnemonico	operazione
72	ENB entrata binaria	legge una scheda (72 colonne) binaria nelle 4 celle I+rX[J], I+rX[J]+1, I+rX[J]+2, I+rX[J]+3. La macchina si ferma se in una delle colonne della scheda è perforato un carattere diverso da: 0, 1, spazio.
73	ENA entrata alfanumerica	legge una scheda alfanumerica (72 colonne) nelle 24 celle I+rX[J], I+rX[J]+1, ..., I+rX[J]+23.
75	CEN controllo entrata	se il lettore di schede non ha completato la lettura, rCP ← I+rX[J]
74	USC uscita	scrive una riga di 120 caratteri sulla stampante; i 120 caratteri sono nelle celle I+rX[J], I+rX[J]+1, ..., I+rX[J]+39.
	CONTROLLORE PERIFERICO	
76	CUS controllo uscita	se la stampante non ha completato la stampa, rCP ← I+rX[J].

1.10 Istruzioni di controllo

codice ottale	mnemonico	operazione
00	ALT	rCP ← I+rX[J] la macchina si ferma
70	ESG esegui	viene eseguita l'istruzione nella cella I+rX[J]
77	NOP nessuna operazione	non viene eseguita alcuna operazione

Nota: le istruzioni TRX, TDX, MEX, ADX, SOX, ADDX, SODX, CFX, CFXD con J=0 hanno lo stesso effetto di NOP. Le istruzioni MEX, CFX, CFXD funzionano come se J=0 indicasse un registro il cui contenuto è sempre nullo.

1.11 La consolle

Nel funzionamento normale, l'interruttore SC (singolo/continuo) si trova nella posizione C (continuo). Premendo il pulsante GO (avvio), la macchina inizia l'esecuzione del programma, prendendo come prima istruzione quella contenuta nella cella specificata dal contenuto del registro contatore programma rCP. Premendo ALT, la macchina si ferma dopo aver completato l'esecuzione dell'istruzione corrente.

Premendo il pulsante L (lettura) a macchina ferma, viene letta una scheda nella modalità binaria, nelle celle 000, 001, 002 e 003, e dopo

la lettura la macchina inizia ad eseguire il programma con l'istruzione nella cella 000. Normalmente, le quattro istruzioni così caricate sono le istruzioni iniziali di un programma di caricamento (vedi paragrafo 2.1).

Il pulsante L non ha alcun effetto quando il calcolatore è in attività.

Se l'interruttore SC si trova nella posizione S (singolo), il calcolatore esegue una istruzione del programma tutte le volte che viene premuto il pulsante S(singolo).

La consolle possiede indicatori luminosi che mostrano il contenuto di tutti i registri ed indicatori della macchina.

PARTE SECONDA

2.1 Il caricatore minimo

Il CANE, come tutti i calcolatori digitali automatici, e' in grado di eseguire automaticamente procedure costituite da sequenze comunque lunghe di operazioni, purche' l'insieme delle istruzioni (programma) che definiscono la procedura sia stato predisposto nella memoria. In altri termini, il programma deve risiedere in memoria prima che l'esecuzione del programma stesso abbia inizio.

Come puo' l'utente caricare in memoria il programma in modo che lo stesso possa poi essere eseguito? In generale, la consolle dei calcolatori dispone di una pulsantiera mediante la quale e' possibile predisporre a piacimento lo stato della memoria e dei registri. Il procedimento piu' elementare per caricare ed eseguire un programma e' il seguente:

- il programma utente viene perforato su schede e il pacco di schede posto nel cassetto del lettore;
- un programma molto breve (programma di caricamento), contenente pero' un ciclo di lettura tale da caricare tutte le schede su cui e' stato perforato il programma di utente, viene immesso manualmente in memoria tramite la pulsantiera della consolle;
- il registro rCP viene predisposto sulla prima istruzione del programma di caricamento e la macchina viene attivata.

Il programma di caricamento legge le schede, cosicche' il

programma di utente entra in memoria ed e' quindi disponibile per l'esecuzione.

Il CANE non dispone della pulsantiera sulla consolle, ma dispone di un congegno equivalente: il pulsante di lettura e avvio L. Il pulsante L (vedi par. 1.11) legge il contenuto di una scheda in modalita' binaria nelle celle da 000 a 003; alla fine della lettura in rCP viene impostato l'indirizzo 000, cosicche' l'unita' di controllo inizia la esecuzione da tale istruzione. In altri termini, il pulsante L consente di caricare ed eseguire un programma di quattro istruzioni. Le quattro istruzioni nelle celle 000 ... 003 possono essere a loro volta istruzioni di lettura, cosicche' altre istruzioni perforate precedentemente su schede possono essere introdotte in memoria.

E' interessante notare come quattro istruzioni siano sufficienti per codificare un vero e proprio ciclo di caricamento, che noi diremo caricatore minimo:

```
000 TDX 1 777-(n-1)
001 ENB 1 n+4
002 ADDX 1 004
003 SLT 0 001.
```

n e' uguale al minimo multiplo di 4 maggiore o uguale al numero delle istruzioni del programma (in altri termini, n e' uguale a 4 volte il numero di schede necessarie per perforare il programma).

Il procedimento per usare il caricatore minimo e' il seguente:

- si scriva il programma in modo che sia da caricare in memoria a partire dalla cella 004 e che detta cella contenga la prima istruzione da eseguire;
- si perfori il programma in binario su schede (quattro istruzioni per scheda);
- si faccia precedere il programma da un caricatore minimo con n opportuno.

A questo punto il pulsante L permette di leggere in memoria la scheda contenente il caricatore minimo e di iniziarne la esecuzione. Il caricatore minimo esegue n/4 letture, memorizzando il programma a partire dalla cella 004. Alla fine del caricamento il registro rX[1] si azzera e di conseguenza si passa ad eseguire la prima delle istruzioni del programma.

Poiche' il calcolatore esegue lo skip sulla istruzione ADDX quando rX[1] diventa 0 e salta alla esecuzione della istruzione in 004, e' possibile che sia da eseguire, prima che il calcolatore ne abbia ultimato la lettura, una delle istruzioni contenute sull'ultima scheda, oppure, piu' in generale, una istruzione il cui corretto comportamento sia subordinato alla presenza in memoria del contenuto dell'ultima scheda. E' quindi necessario fare in modo che il programma controlli la avvenuta lettura dell'ultima scheda prima di usufruire del suo contenuto. A tale scopo si consiglia di inserire una istruzione controllo entrata (CEN) con indirizzo di salto a se stessa in una posizione opportunamente scelta.

L'utilizzo del pulsante L, associato con il caricatore minimo, permette di introdurre in memoria ed eseguire un programma costituito da un numero a piacere di istruzioni (naturalmente nei limiti della memoria del CANE).

2.2 Il software

Il caricamento ed esecuzione di programmi tramite il caricatore minimo, oltre ad altri difetti di carattere secondario (ad esempio, il programma deve esser tale che la sua prima istruzione sia nella cella 004), comporta un gravissimo inconveniente: il programma deve essere perforato in binario. Chiunque abbia provato ad utilizzare il caricatore minimo si sara' accorto che la perforazione in binario e' fonte di frequentissimi errori, tanto che gia' la perforazione di un programma di qualche decina di istruzioni e' impresa che comporta praticamente notevoli difficolta'. Estrapolando la discussione a programmi di notevole mole, si puo' intuire facilmente come, qualora l'utente fosse costretto a codificare i propri programmi in binario, i calcolatori sarebbero strumenti praticamente impossibili da utilizzare.

I calcolatori, in effetti, vengono messi a disposizione dell'utente gia' corredati di un insieme di programmi (software) aventi lo scopo di permettere un accesso meno difficoltoso al mezzo di calcolo. Nel senso piu' elementare cio' puo' essere inteso come possibilita' di presentare programmi redatti in un formalismo diverso dal binario. Piu' in generale, il software puo' essere visto come un "filtro"

che modifica sostanzialmente le caratteristiche della macchina. Senza entrare nei dettagli, possiamo dire che cio' che l'utente vede non e' la macchina intesa come componenti propriamente fisiche (hardware), ma la macchina piu' un certo insieme di programmi (software). Questo insieme (hardware piu' software, fig. 3) puo' presentare delle caratteristiche notevolmente diverse dalla macchina vera e propria, tanto da permettere, se il software e' opportunamente sofisticato, l'accesso al mezzo di calcolo anche a utenti che ignorano completamente (o quasi) i principi di funzionamento e l'architettura della macchina che stanno utilizzando.

I paragrafi seguenti descrivono il software del CANE. Il loro scopo e' quello di chiarire, con un esempio, quanto accennato nel presente paragrafo. Il software del CANE e' di una semplicita' estrema, tanto da rischiare di diventare deformante. Quello che segue deve percio' essere considerato nella giusta luce, cioe' come l'apertura di un discorso tutt'altro che semplice, che sara' poi ripreso massicciamente negli anni di corso successivi al primo.

2.3 Il pacchetto software del CANE

Il software del CANE e' costituito da:

1) il nucleo di controllo;
2) il caricatore ottale rilocante;
3) la routine di lettura-scrittura.

Il calcolatore viene messo a disposizione con il software gia' caricato in memoria, nella zona da 004 a 314. Le celle da 000

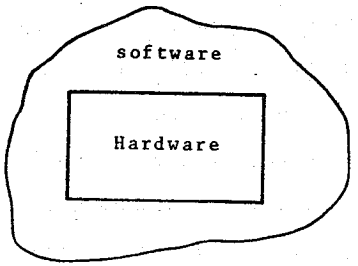


Fig. 3

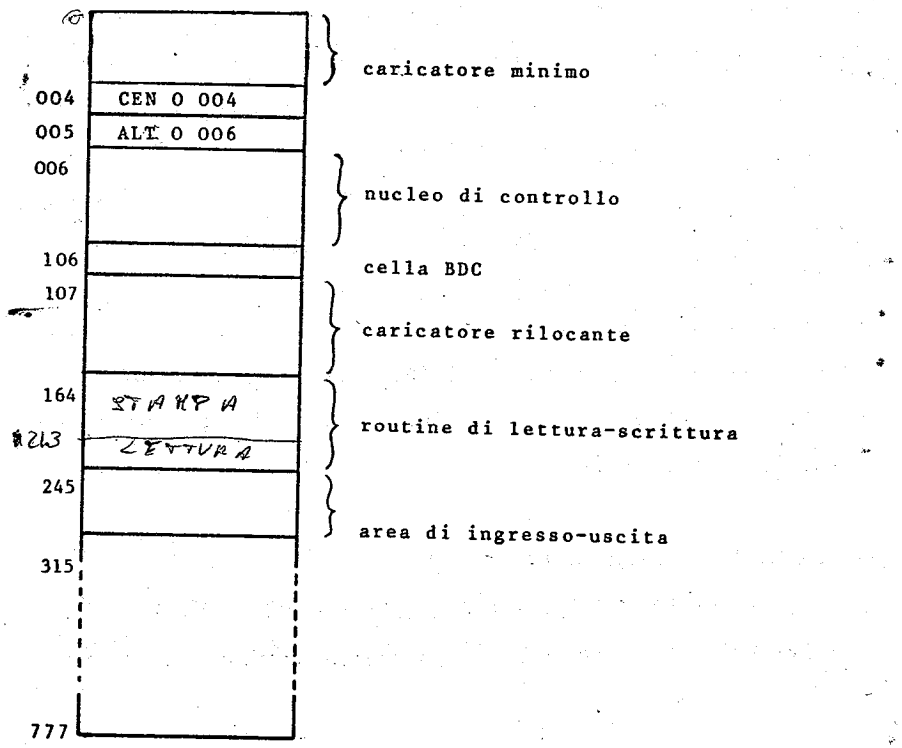


Fig. 4

a 003 sono occupate da un caricatore minimo mediante il quale il pacchetto software, perforato in binario su schede, viene inizialmente introdotto in memoria dall'operatore. La fig. 4 descrive la configurazione della memoria del CANE dopo il caricamento del pacchetto software.

Il nucleo di controllo, come pure il caricatore e la routine di lettura scrittura, saranno descritti nei prossimi paragrafi. Comunque, conviene anticipare fin da ora le funzione del caricatore e della routine di lettura scrittura. Il caricatore ottale rilocante e' progettato in modo da poter caricare in memoria programmi scritti in ottale, in una zona qualsiasi purche' di indirizzo superiore a 314 (vedi fig. 4). La routine di lettura scrittura ha lo scopo di aiutare l'utente durante l'ingresso e uscita di dati numerici, operazione non del tutto banale in quanto richiede conversioni da decimale a binario e viceversa.

Lo scopo della istruzione "ALT 0 006" nella cella 005 sara' illustrato nei paragrafi 2.4 e 2.7. Esso e' connessa con l'attivazione del nucleo di controllo. Tale attivazione e' a carico dell'operatore. Essa viene effettuata, una prima volta, dopo il caricamento iniziale del pacchetto software; successivamente, al termine dell'esecuzione di ogni programma utente (vedi par. 2.7).

2.4 Il nucleo di controllo

Il nucleo di controllo viene attivato immediatamente dopo il termine del caricamento del pacchetto software (vedi par.

2.7). Esso e' descritto dallo schema a blocchi di fig. 5.

Il nucleo di controllo accetta tre tipi di schede controllo:

a) la scheda *CAR;

b) la scheda *ESG;

c) la scheda *ALT.

I formati sono:

```
1    6 7 9
*CAR  xxx
```

```
1    6 7 9
*ESG  xxx
```

oppure

```
1    6 7 9
*ESG
```

```
1    6 7 9
*ALT
```

dove "x" indica una cifra ottale.

La scheda *CAR chiede l'intervento del caricatore rilocante per caricare un programma perforato in ottale su schede (vedi par. 2.5). La scheda *ESG chiede invece che sia dato inizio alla esecuzione di un programma gia' caricato in memoria.

Nel caso della scheda *CAR, prima dell'esecuzione del "SLT 0 107 (attivazione caricatore rilocante, vedi fig. 5), il valore xxx viene depositato nella cella indicata simbolicamente con BDC (indirizzo 106, vedi fig. 4) e lasciato cosi' a disposizione del caricatore stesso. Il valore xxx ha infatti significato di punto di partenza per il caricamento (vedi par. 2.5), cioe' base di caricamento (simbolicamente

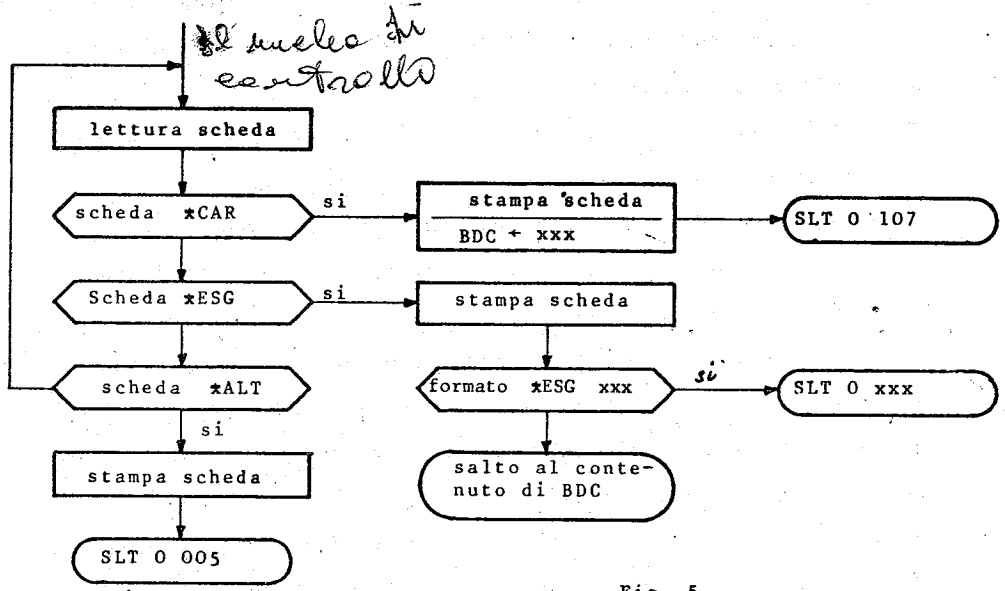
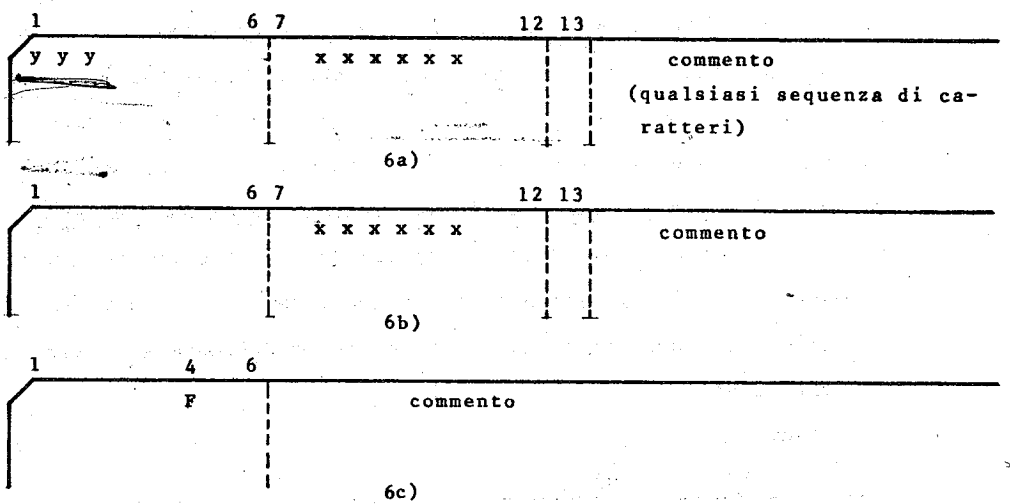


Fig. 5



"x" e "y" indicano cifre ottali

colonna 13: * oppure spazio bianco

Fig. 6

BDC). Nel caso della scheda *ESG, il nucleo di controllo esegue un salto sulla istruzione in xxx o sulla istruzione il cui indirizzo e' in BDC. In altri termini, se il campo xxx di una *ESG e' lasciato in bianco, il nucleo di controllo assume che il programma inizi dalla base di caricamento specificata nella scheda *CAR precedentemente incontrata.

La scheda *ALT segnala al nucleo di controllo che la elaborazione di un programma utente e' giunta al suo termine. Come illustrato dalla figura 5, quando il nucleo di controllo incontra una scheda *ALT viene eseguita una istruzione "SLT 0 005"; poiche' la cella 005 contiene una "ALT 0 006" (vedi fig. 4), la macchina si arresta e viene impostato nel registro rCP l'indirizzo 006 (vedi par. 1.10), corrispondente esattamente all'ingresso nel nucleo di controllo. A questo punto sara' compito dell'operatore riattivare la macchina per dare inizio all'elaborazione di un nuovo pacco programma di utente (vedi par. 2.7).

Nel caso che la scheda letta non sia ne' una *CAR, ne' una *ESG ne' una *ALT, il nucleo trascura la scheda e passa ad una nuova lettura.

L'area di memoria utilizzata per le letture e' l'area di ingresso-uscita indicata in fig. 4.

2.5 Il caricatore rilocante

Il caricatore rilocante e' sostanzialmente un "traduttore": esso accetta programmi redatti in ottale rilocabile, genera il corrispondente codice binario e lo

carica in memoria. Il tipo di traduzione e', evidentemente, di estrema semplicita', in quanto la codifica ottale e' molto vicina a quella binaria. L'ottale, infatti, puo' essere riguardato come una forma contratta del binario.

Conviene, prima di procedere, spendere qualche parola per tentare di chiarire il significato del termine "rilocabile". Un programma redatto come una sequenza di istruzioni (e costanti del programma) scritte in ottale e' chiaramente vincolato ad una ben precisa posizione in memoria. Per convincersi basta pensare al campo indirizzo delle istruzioni di salto, o, piu' in generale, ai campi indirizzo delle istruzioni che fanno riferimento a celle di memoria. Lasciando al lettore la verifica della seguente asserzione, possiamo dire che:

> un programma in ottale (o in binario) scritto per essere caricato ed eseguito a partire da una cella xxx, puo' essere caricato ed eseguito a partire da una cella yyy solo se a tutti i campi indirizzo che fanno riferimento alla memoria viene aggiunta, all'atto del caricamento, la quantita': yyy - xxx.

La "traslazione" di programmi in memoria, che, come abbiamo detto, implica la modifica dei campi indirizzo di alcune delle istruzioni, va di solito sotto il nome di rilocazione. Un programma (in ottale o binario) si dice rilocabile se contiene indicazioni che precisano quali sono le istruzioni il cui campo indirizzo deve essere modificato all'atto della rilocazione. Il caricatore del CANE e' detto

rilocante in quanto accetta una codifica ottale rilocabile.

Per usare correttamente il caricatore rilocante, l'utente puo' procedere come segue:

- scrivere il programma come se fosse da caricare e poi eseguire a partire dalla cella 000 di memoria;
- segnalare esplicitamente, con un asterisco (vedi fig. 6), le istruzioni il cui campo indirizzo e' da modificare all'atto del caricamento.

I formati accettati dal caricatore rilocante sono illustrati in fig. 6. Il formato 6a indica che la istruzione (o la costante di programma) codificata da xxxxxx e' da caricare nella cella di indirizzo: BDC + yyy, cioe' in una cella spostata di yyy posizioni rispetto alla base di caricamento definita nella scheda *CAR che precede il programma ottale. Il formato 6b indica che il binario corrispondente a xxxxxx e' da caricare nella cella immediatamente seguente quella in cui la precedente istruzione (o costante) e' stata caricata. Il formato 6b indica, cioe', caricamento sequenziale; nel caso che la prima scheda dopo la *CAR abbia formato 6b, si assume yyy uguale a 000. In entrambi i casi (formati 6a e 6b), se la colonna 13 contiene un asterisco, la configurazione xxxxxx viene incrementata del valore contenuto in BDC. L'asterisco in colonna 13, cioe', segnala al caricatore quali sono le istruzioni la cui parte indirizzo e' da modificare all'atto del caricamento.

Esempio:

```
1    6 7    12 13
|023  |100127 |*
```

Il binario ottenuto da 100127+BDC verra' caricato nella cella di indirizzo 023+BDC.

Esempio:

1 6	7 12	13
023	100127	

Il binario corrispondente a 100127 sara' caricato nella cella 023+BDC.

La scheda 6c indica la fine del caricamento. Quando il caricatore incontra una scheda di tipo 6c, esso riattiva il nucleo di controllo eseguendo un salto alla cella di indirizzo 006, corrispondente esattamente all'ingresso nel nucleo di controllo (vedi fig. 4).

Quanto detto dovrebbe essere sufficiente per utilizzare il caricatore rilocante. Eventuali dubbi potranno essere risolti dall'esame dello schema a blocchi di fig. 7.

Per la lettura delle schede, il caricatore utilizza l'area di ingresso-uscita indicato in fig. 4.

2.6 Routine di lettura-scrittura

La routine di lettura-scrittura consente di leggere da schede e inviare alla stampatrice dati numerici. Essa si occupa della conversione da decimale a binario e viceversa.

La routine di lettura-scrittura ha due punti di ingresso che indicheremo con L (lettura, indirizzo 213) e S (stampa, indirizzo 164). In lettura i numeri devono essere perforati secondo il formato descritto in fig. 8. Chiamando il punto di

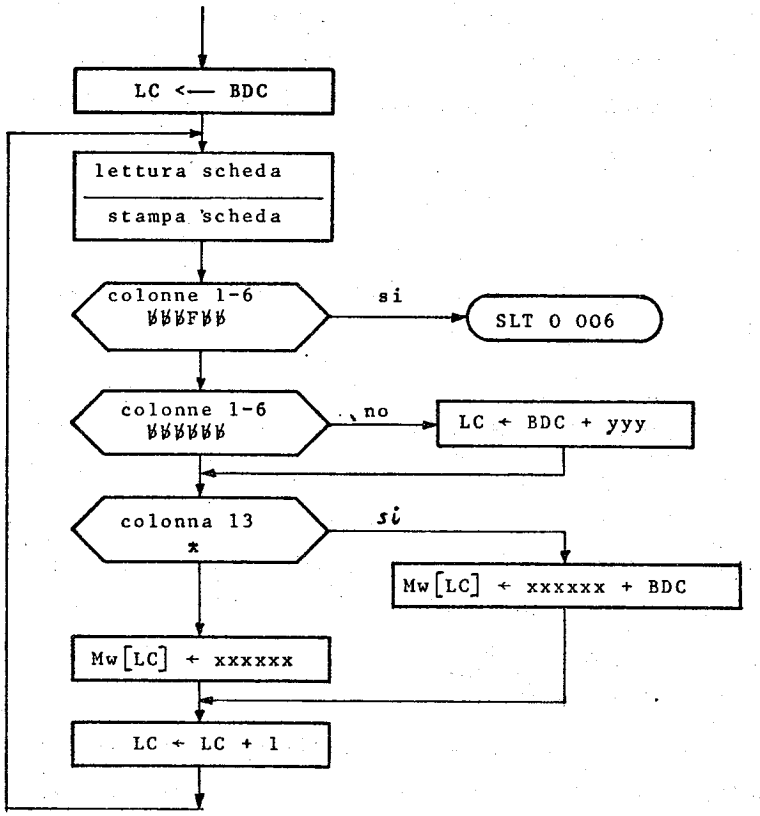


Fig. 7

ingresso L, la routine legge il numero di 6 cifre decimali e segno in uno dei campi descritti in fig. 8; una successiva chiamata di L provoca la lettura del numero nel campo successivo della stessa scheda, oppure, se la scheda e' esaurita, del numero contenuto nel primo campo della scheda successiva. Il salto alla scheda successiva viene effettuato anche nei seguenti due casi:

a) viene chiamato il punto S (stampa);

b) i primi tre caratteri di un campo sono bianchi.

Il numero letto, convertito in binario, viene lasciato nel registro rA.

Chiamando il punto di ingresso S, la routine stampa il contenuto del registro rA nel formato: ± xxxxxx, dove "x" indica una cifra decimale.

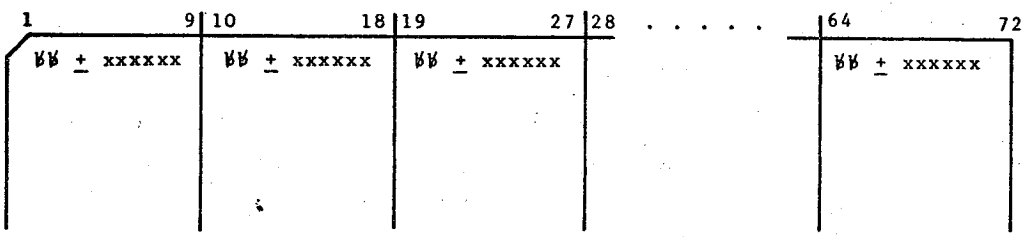
La routine di lettura-scrittura puo' essere attivata quante volte si vuole inserendo nel programma le seguenti istruzioni:

SUB 7 213 punto L (lettura)

SUB 7 164 punto S (scrittura).

La routine si comporta a tutti gli effetti come un sottoprogramma (si noti che essa viene attivata con la istruzione SUB, vedi par. 1.6). Dopo la lettura o la stampa, la routine rimanda il controllo alla istruzione successiva a quella che ha provocato la sua attivazione. Cioe', l'esecuzione riprende regolarmente dalla istruzione successiva alla "SUB 7 213" oppure "SUB 7 164".

Per la letture e le stampe viene ancora utilizzata l'area



xxxxxx cifre decimali

Fig. 8

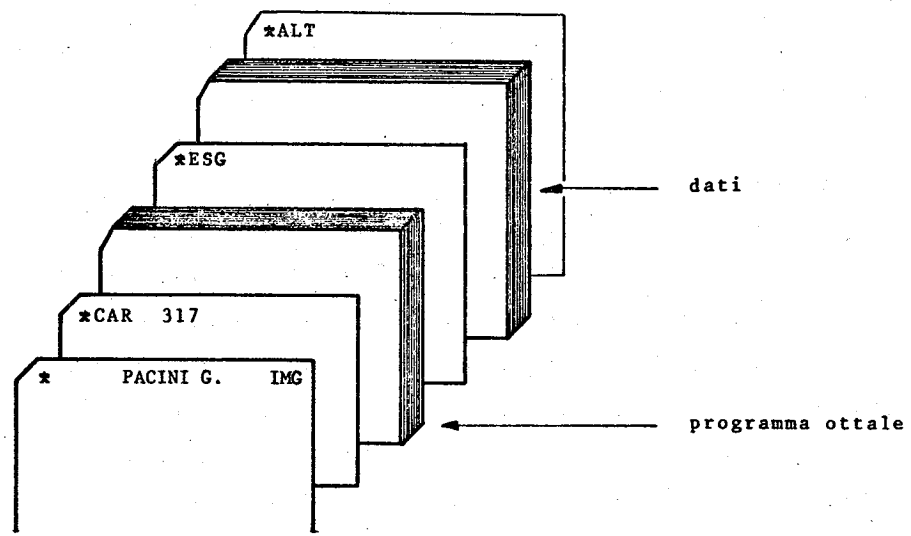


Fig. 9

di ingresso-uscita in fig. 4.

2.7 Utilizzo del software del CANE

Come abbiamo detto il CANE viene messo a disposizione dell'utente con il software gia' pronto in memoria. Il pacchetto software (perforato in binario su schede) viene caricato inizialmente con un caricatore minimo dove la quantita' n e' stata opportunamente predisposta.

Cosa succede di preciso dopo? Chiariamo ora definitivamente la ragione della istruzione "ALT 0 006" nella cella 005; in questo modo, infatti, dopo il caricamento in memoria del pacchetto software, la macchina si arresta con il registro rCP posizionato sulla cella 006 (vedi par. 1.10), che corrisponde esattamente al punto di ingresso nel nucleo di controllo (fig. 4). A questo punto l'operatore prende il primo dei pacchi programma utente, lo pone nel cassetto del lettore di schede e attiva la macchina mediante il pulsante GO (vedi par. 1.11), dando cosi' inizio alla esecuzione del nucleo di controllo.

Quello che succede dopo dipende dalla composizione del pacco programma utente. Comunque, poiche' per ovvie ragioni si vuole evitare la ricarica del pacchetto software in corrispondenza ad ogni utente, l'utente deve attenersi alle seguenti regole:

1) l'ultima istruzione eseguita da un programma utente deve essere non una istruzione di ALT, bensi' un "SLT 0 006";
2) l'ultima scheda di un pacco programma utente deve essere

una scheda controllo *ALT;
3) I programmi di utente non devono modificare la zona di memoria compresa tra 000 e 314.

Se le regole 1, 2, e 3 sono state rispettate, il programma utente termina riattivando il nucleo di controllo (regola 1); il nucleo di controllo incontra la scheda *ALT e la macchina si arresta con il rCP posizionato sulla istruzione nella cella 006 (vedi par. 2.4). In questo modo, alla fine della elaborazione di ogni programma di utente, il calcolatore viene rimesso nella condizione iniziale; per dare avvio all'elaborazione di un altro programma utente, l'operatore dovra' semplicemente porre il pacco di schede nel cassetto del lettore e riattivare la macchina con il pulsante GO.

Naturalmente, per errore o per particolare cattiveria, l'utente puo' violare una delle regole del gioco. Per esempio, un programma puo' terminare in modo anomalo, o per fine dati o con un ALT; oppure, piu' disastrosamente, puo' alterare la zona di memoria da 000 a 314. In tutti questi casi, la questione viene risolta in modo radicale: il pacchetto software viene ricaricato completamente e il procedimento ricomincia.

Evidentemente l'alterazione della zona di memoria contenente il software viene, nella maggior parte dei casi, rivelata da un anomalo comportamento del sistema durante il caricamento del programma successivo. In certi casi possono sorgere dubbi sul fatto che un comportamento strano dipenda da una precedente "caduta" del sistema oppure da errori nel pacco

programma in elaborazione. In caso di dubbio, l'operatore risolvera' la questione ricaricando l'intero pacchetto software.

2.8 La scheda asterisco

esempi di pacco-programma di utente

Ogni pacco-programma utente deve iniziare con una scheda, che ha il seguente formato (scheda asterisco):

- colonne 1-6: un asterisco seguito da 5 spazi bianchi;
- colonne 7-68: nome e cognome utente, nome del programma e annotazioni a piacere;
- colonne 70-72: IMG oppure tre spazi bianchi.

Se le colonne 70-72 contengono la scritta IMG, alla fine della esecuzione del programma viene stampata una immagine completa della memoria del CANE, come essa e' al momento in cui viene eseguita la istruzione di ALT. Il lettore e' pregato di non chiedersi come cio' avviene; comunque, si puo' immaginare facilmente come il software del CANE potrebbe essere arricchito con un programma capace di produrre la suddetta immagine.

La composizione tipica di un pacco programma per il CANE e':

esempio 1:

1) scheda asterisco
2) scheda *CAR xxx
3) programma ottale
4) scheda *ESG

5) dati per il programma di cui al punto 3
6) scheda *ALT.

Nel caso illustrato in fig. 9, gli eventi si succedono come segue:

- il nucleo di controllo legge la scheda asterisco e la trascura;
- il nucleo di controllo legge la *CAR e attiva il caricatore con una base di caricamento (BDC) uguale a 317;
- alla fine del caricamento, segnalato dalla scheda "fine caricamento" (vedi fig. 6, formato 6c) il nucleo di controllo legge la scheda *ESG: la esecuzione del programma utente ha inizio a partire dalla istruzione in 317.

A questo punto nel cassetto del lettore di schede sono rimasti solo i dati e la *ALT. I dati saranno letti ed elaborati durante la esecuzione del programma di cui al punto 3; la scheda *ALT provochera' infine l'arresto della macchina (vedi par. 2.4 e 2.7).

Altre composizioni piu' complesse sono evidentemente possibili.

Esempio 2:

1) scheda asterisco
2) scheda *CAR xxx
3) programma ottale
4) scheda *ESG
5) dati

6) scheda *ESG

7) dati

8) scheda *ALT.

Il programma di cui al punto 3 viene eseguito due volte, in corrispondenza rispettivamente ai dati di cui al punto 5 e 7.

Esempio 3:

1) scheda asterisco

2) scheda *CAR xxx

3) programma in ottale

4) scheda *CAR yyy

5) programma in ottale

6) scheda *ESG xxx

7) dati

8) scheda *ALT.

In questo esempio si e' supposto che il programma da mandare in esecuzione sia composto dalle due sezioni 3 e 5.

Esempio 4:

1) scheda asterisco
2) scheda * CAR xxx
3) programma in ottale
4) scheda *ESG
5) dati
6) scheda *CAR xxx
7) programma in ottale
8) scheda *ESG
9) dati
10) scheda *ALT.

In questo ultimo esempio l'utente fa in realta' eseguire due programmi distinti (ma non necessariamente indipendenti. Perche'?).

Perché dopo l'ultima scheda dei dati al punto 5, si mette la scheda 7, così che fa saltare il controllo, al nucleo di controllo facendo così attivare il nucleo di controllo e legge così un altro pezzo programma

APPENDICE I

Carattere	Codice	Carattere	Codice
A	010001	0	000000
B	010010	1	000001
C	010011	2	000010
D	010100	3	000011
E	010101	4	000100
F	010110	5	000101
G	010111	6	000110
H	011000	7	000111
I	011001	8	001000
J	100001	9	001001
K	100010		
L	100011	b(spazio)	110000
M	100100	.	011011
N	100101	(	111100
O	100110	+	010000
P	100111	$	101011
Q	101000	*	101100
R	101001	)	011100
S	110010	-	100000
T	110011	/	110001
U	110100	,	111011
V	110101	=	001011
W	110110		
X	110111		
Y	111000		
Z	111001		

Tabella I - Codice caratteri

2^n	n	2^{-n}
1	0	1,0
2	1	0,5
4	2	0,25
8	3	0,125
16	4	0,0625
32	5	0,03125
64	6	0,015625
128	7	0,0078125
256	8	0,00390625
512	9	0,001953125
1024	10	0,0009765625
2048	11	0,00048828125
4096	12	0,000244140625
8192	13	0,0001220703125
16384	14	0,00006103515625
32768	15	0,000030517578125
65536	16	0,0000152587890625
131072	17	0,00000762939453125
262144	18	0,000003814697265625
524288	19	0,0000019073486328125
1048576	20	0,00000095367431640625

Tabella II - Potenze di 2.

n	8^n	16^n
0	1	1
1	8	16
2	64	256
3	512	4096
4	4096	65536
5	32768	1048576
6	262144	16777216

Tabella III - Potenze di 8 e di 16

APPENDICE II

Elenco delle istruzioni

00	ALT	
01	TRA	trasferimento: in rA
02	TRAN	trasferimento: in rA negativo
03	TDA	trasferimento: diretto in rA
04	TDAN	trasferimento: diretto in rA negativo
05	TRB	trasferimento: in rB
06	TRX	trasferimento: in rXj
07	TDX	trasferimento: diretto in rXj
10	MEA	memorizzazione: di rA
11	MEB	memorizzazione: di rB
12	MEZ	memorizzazione: di zero
13	MEX	memorizzazione: di rXj
14	ADA	addizione ad rA
15	SOA	sottrazione da rA
16	ADDA	addizione diretta ad rA
17	SODA	sottrazione diretta da rA
20	ADB	addizione ad rB
21	SOB	sottrazione da rB
22	ADX	addizione ad rXj
23	SOX	sottrazione da rXj
24	ADDX	addizione diretta ad rXj
25	SODX	sottrazione diretta da rXj
26	MOL	moltiplicazione

27	DIV	divisione
30	COP	complemento
31	AND	prodotto logico
32	OR	somma logica
33	ORX	somma modulo 2
34	CRA	confronto: con rA
35	CFB	confronto: con rB
36	CFX	confronto: con rXj
37	CFXD	confronto: diretto con rXj
40	SLT	salto incondizionato
41	SUB	salto: a sottoprogramma
42	SSP	salto: se supero
43	SMIN	salto: se minore
44	SMAG	salto: se maggiore
45	SUG	salto: se uguale
46	SAN	salto: se rA negativo
47	SAP	salto: se rA positivo
50	SAZ	salto: se rA zero
51	SBN	salto: se rB negativo
52	SBP	salto: se rB positivo
53	SBZ	salto: se rB zero
54	AVD	spostamento: aritmetico di rA verso destra
55	AVS	spostamento: aritmetico di rA verso sinistra
56	ALD	spostamento: logico di rA verso destra
57	ALS	spostamento: logico di rA verso sinistra
60	BVD	spostamento: aritmetico di rB verso destra
61	BVS	spostamento: aritmetico di rB verso sinistra
62	ABLD	spostamento: logico di rA ed rB verso destra
63	ABLS	spostamento: logico di rA ed rB verso sinistra

64	TCA	trasferimento carattere
65	MCA	memorizzazione carattere
66	CAR	conversione in caratteri
67	NUM	conversione in numero binario
70	ESG	esegui
71	GULP	
72	ENB	entrata binaria
73	ENA	entrata alfanumerica
74	USC	uscita
75	CEN	controllo entrata
76	CUS	controllo uscita
77	NOP	nessuna operazione