UNIVERSITA' DEGLI STUDI DI PISA

FACOLTA' DI SCIENZE MAT. FIS. E NAT.

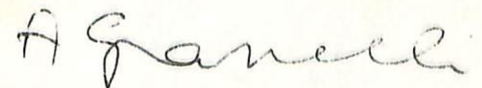
Corso di Laurea in Fisica

TESI DI LAUREA:

"UN PROGRAMMA ASSEMBLATORE PER UN CALCOLATORE DIDATTICO"

Relatore:

Chiar.mo Prof.A.GRASSELLI



Candidato:

Attilio Ripoli

Anno Accademico 1969-70

TESINE

"L'EQUAZIONE DI CLAUSIUS - CLAPEYRON, SUA DEDUZIONE E SIGNIFICATO"

L. BOSMAN

"NOTE SUL METODO MONTECARLO"

R. CASALI

"ISTOGRAMMI E POTERE RISOLUTIVO DEGLI STRUMENTI"

A. BIGI

I N D I C E

INTRODUZIONE. . . . . .pag. I-V

Il linguaggio SIMBOLCANE. . . . . " 1

Tabelle dei simboli. . . . . " 24

Struttura del programma . . . . " 47

Software e liste. . . . . " 59

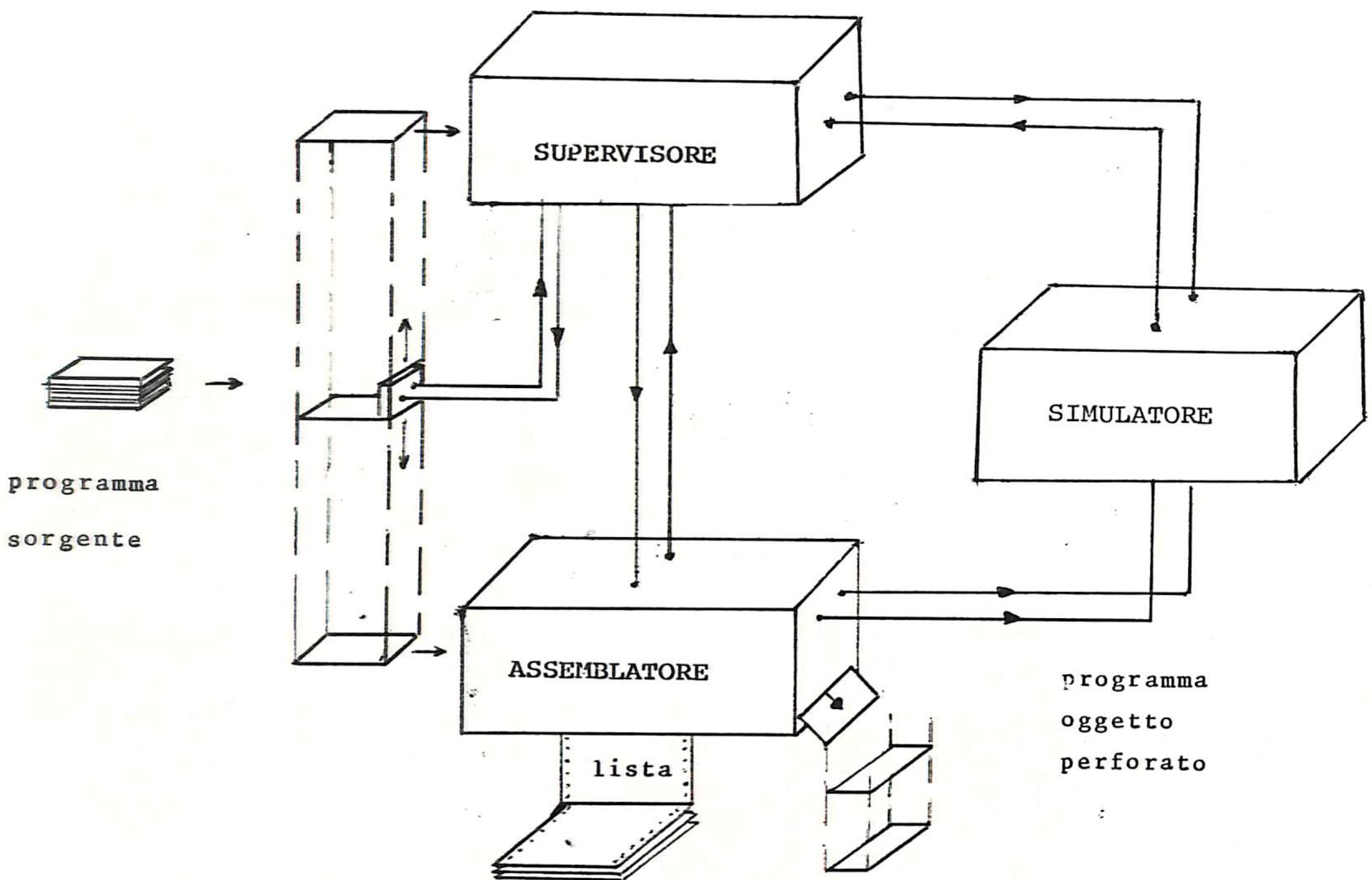
Segnalazione degli errori . . . . " 71

Appendice 1 : istruzioni del CANE . . . . " 83

Appendice 2 : codici BCD . . . . " 87

Appendice 3 : struttura del pacco di schede
per un programma in SIMBOLCANE . . . . " 88

BIBLIOGRAFIA


programma
sorgente
SUPERVISORE
ASSEMBLATORE
lista
SIMULATORE
programma
oggetto
perforato

INTRODUZIONE

Scopo di questo lavoro è di fornire un programma di assemblaggio per un linguaggio simbolico, che chiameremo SIMBOLCANE, per il Calcolatore Automatico Numerico Educativo (CANE).Il CANE è un calcolatore didattico che, sebbene non sia stato costruito, è funzionante perchè simulato su un calcolatore IBM 7090, e serve alle esercitazioni degli studenti del primo biennio del corso di laurea in Scienza dell'Informazione.La struttura di questo calcolatore e il programma che lo simula, detto SIMULCANE, sono stati oggetto di una precedente tesi di Laurea (si vedano 1 e 2 in bibliografia).

Questo programma di assemblaggio simbolico, o più semplicemente assemblatore, consta di un programma principale, scritto in MAP, per oltre 5000 istruzioni, e di tre sottoprogrammi scritti in FORTRAN, con qualche centinaio di istruzioni. Esso provvede alla trascrizione automatica di un programma scritto nel linguaggio simbolico, (o linguaggio sorgente), nel corrispondente programma in linguaggio macchina, (o linguaggio oggetto); forni-

sce alcune pseudo-operazioni, mediante le quali viene facilitata, come vedremo, la scrittura dei programmi in SIMBOLCANE. Oltre a questo, il programma assemblatore segnala alcuni degli errori formali che si possono trovare nel programma scritto in linguaggio sorgente, specificandone posizione e tipo, e può, a richiesta, fornire perforato su schede il programma scritto nel linguaggio oggetto.

Per ogni programma assemblato, cioè sottoposto al processo di trasposizione da simbolico a macchina, viene stampata una lista in cui compaiono, faccia a faccia, il linguaggio sorgente e quello oggetto, le eventuali segnalazioni di errori, e, alla fine, una tabella in cui sono riportati i simboli usati nel programma sorgente e il valore che loro è stato attribuito durante l'assemblaggio, nonchè, se è stato richiesto il pacco di schede col programma oggetto, una stampa di questo pacco.

La necessità di un tale programma assemblatore si può evidenziare notando che il linguaggio di un calcolatore, cioè il già citato linguaggio macchina, è fatto di configurazioni binarie:

numeri per indicare le operazioni da svolgere, numeri per indicare gli operandi, e così via; invece il linguaggio usato dal programmatore è sempre una combinazione di numeri e simboli che, nella sua forma più semplice, rispecchia il linguaggio del calcolatore, mentre nelle forme più evolute si avvicina a un linguaggio di tipo matematico, o commerciale, o linguistico, cioè, come si dice, orientato al problema, mentre nel caso più semplice si parlerà di linguaggi orientati verso la macchina. Tanto per fare un esempio, l'istruzione "somma il numero 1" può essere scritta dal programmatore come (ci riferiremo sempre a linguaggi del tipo più semplice, qual'è il SIMBOLCANE)

ADDA 1

mentre per il calcolatore essa dovrà assumere la forma binaria

0011100000000000001

che possiamo scrivere in forma ottale come

160001

16 è il codice di somma diretta per il CANE, 001 indica l'operando, e lo 0 centrale significa, come vedremo, che non è considerato

alcun registro indice.

La corrispondenza tra i due linguaggi non è sempre così semplice, tanto che si parla di "barriera di linguaggio";per superare questa barriera bisogna provvedere a quella trascrizione automatica di cui si diceva prima, in modo che scrivere su una scheda

ADDA 1

equivalga a porre nella memoria del CANE il numero

160001

E' evidente che, per l'utente del calcolatore, un linguaggio di tipo simbolico è molto più facile a scriversi che non un linguaggio esclusivamente numerico: torneremo comunque su questo punto, e sull'utilità di avere perforato su schede il programma oggetto, così come torneremo sul fatto che la segnalazione degli errori rappresenta un valido aiuto per i programmatori, specie per i meno esperti. A questo proposito, giova ricordare ancora una volta che il CANE è stato simulato soprattutto per gli studenti dei primi anni del corso di laurea in Scienza

dell'Informazione; e di questa impostazione risente in parte, come vedremo, anche l'assemblatore.

La perforazione del programma oggetto non è compito diretto dell'assemblatore, e va considerata come un servizio reso all'utente del CANE, così come il caricatore dei programmi perforati, che è accluso a questo lavoro.

Per finire, vogliamo ricordare il formato delle parole di memoria del calcolatore IBM 7090 e del CANE.

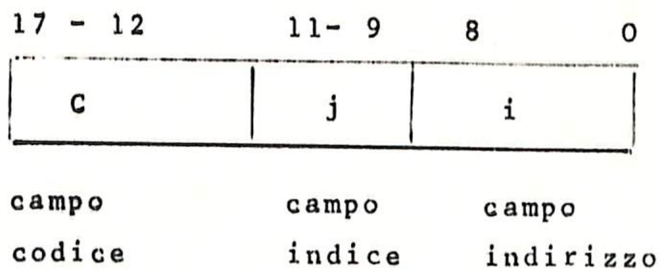


fig.2.Formato
parola del CANE

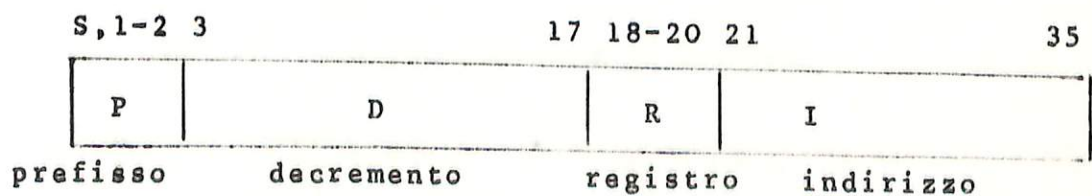


fig.2a. Formato parola del 7090

CAPITOLO I

IL LINGUAGGIO SIMBOLCANE

Il linguaggio in cui vengono scritti i programmi sorgente è detto SIMBOLCANE: si tratta di un linguaggio simbolico semplice, che è molto simile al linguaggio macchina.

I numeri da 00 a 64 che specificano le operazioni del CANE sono sostituiti da altrettanti simboli, che sono l'abbreviazione dei nomi o delle locuzioni italiane che indicano l'operazione. Ad esempio

ADDA

vuol dire "aggiungi direttamente al registro accumulatore A" mentre

SLT

significa "salta". Anche le pseudo operazioni sono indicate con nomi che ricordano immediatamente la loro funzione. Per l'elenco completo delle istruzioni del CANE, si veda l'Appendice 1.

Le varie posizioni di memoria sono individuate dal loro indirizzo, che varia da 000 a 512; esse possono anche essere individuate con un simbolo, che viene in quel caso detto etichetta.

Alle etichette ci si può poi riferire allorché si calcolano gli operandi di una istruzione. Ad esempio, supponiamo che nella cella 125 ci sia l'istruzione:

(cella 125)   ETIC   ADDA   1

ETIC individua la cella 125. Quindi scrivere

TRA   ETIC

equivale a scrivere

TRA   125

Alla etichetta è assegnato il valore che ha il contatore di istruzioni, che contiene l'indirizzo della cella cui ci si riferisce, la prima volta che l'etichetta è trovata. Nel nostro esempio, ad ETIC è attribuito il valore 125. Una eventuale ricomparsa dello stesso simbolo come etichetta non fa variare il valore attribuitogli in precedenza, bensì provoca la comparsa di un messaggio di errore. Così, se alla cella 171 è attribuito ancora il nome ETIC

(cella 171)   ETIC   MEA   147

a ETIC resta pur sempre il valore 125. L'etichetta non è mai obbligatoria.

I simboli, siano essi usati come etichette, o come operandi nel calcolo degli indirizzi, devono soddisfare certe regole: essi debbono essere formati da una stringa di caratteri alfanumerici, al massimo sei, il primo dei quali deve comunque essere alfabetico. Tra gli altri caratteri, non sono mai permessi, come caratteri simbolici, il+, il-, la virgola, le parentesi e lo asterisco: la loro comparsa potrebbe portare a una impossibilità di comprensione di un indirizzo. Infatti, se essi fossero leciti, come sarebbe possibile sapere se il simbolo A+*,7 significa il simbolo A+*,7, ovvero il simbolo A+* e la specifica del registro 7, ovvero ancora la somma di A e del valore attuale di *, con la specifica del registro 7, o infine la somma dei simboli A e *,7?

Il carattere * ha un valore particolare: esso assume il valore del contatore di istruzioni. In

```
(cella 160)        AX        TRB     *
```

AX e * hanno il valore 160. Esso è molto utile quando ci si voglia riferire a istruzioni vicine a quella che stiamo scrivendo: infatti

```
                                    SLT     * + 2
```

vuol dire "salta all'istruzione che ha indirizzo quello di questa istruzione + 2".

Un altro carattere che non deve essere usato nei simboli è la barra: esso ha una funzione particolare. Infatti le schede su cui è perforata una barra in colonna 1 sono considerate come schede di commento, e come tali stampate, ma non assemblate.

Gli indirizzi sono in genere formati da simboli e numeri, uniti dai segni di addizione (+) e sottrazione (-). Non importa che i simboli siano già stati definiti o no (virtuali).

Le istruzioni sono perforate su schede, seguendo certe regole.

Una istruzione è formata da tre parti: un codice operativo, un indirizzo e una eventuale specifica di registro indice. Queste tre parti possono essere, in un linguaggio assemblatore, distinte in vari modi, ad esempio separandole con un certo carattere, oppure fissando la porzione di scheda in cui devono essere perforate, ovvero combinando i due modi detti; noi abbiamo adottato questo ultimo metodo.

Ogni scheda è divisa in tre campi, che si estendono in colon-

ne 1 / 6, 7 / 12 e 13 / 72, (Le colonne 73/80 non vengono mai considerate), i primi due fissi, il terzo variabile.

Il secondo e il terzo campo contengono l'istruzione vera e propria: il simbolo che sta a rappresentare l'operazione è perforato a partire da colonna 8, la colonna 7 restando sempre bianca per motivi estetici, in quanto separa i primi due campi, e termina al massimo a colonna undici, così che anche la colonna 12 resta bianca, per motivi analoghi a quelli della col.7; indirizzo e specifica di registro sono perforati a partire da colonna 13. Questo terzo campo, a differenza degli altri due, non è fisso: infatti indirizzo e registro sono separati da una virgola, la cui posizione può variare di volta in volta, ed uno spazio bianco indica che tutto ciò che si trova dopo deve essere considerato un commento. Se è necessario specificare solo un registro, il terzo campo può cominciare con una virgola, seguita dalla specifica del registro: in questo caso si prende come indirizzo il valore zero. Quindi scrivere

```
SUB        ,7
```

equivale a scrivere

SUB 0,7

Una istruzione non può estendersi al di là della colonna 72, né proseguire su una scheda successiva.

Può essere spesso conveniente considerare la parte indirizzo come un operando, cioè poter ad esempio scrivere, per caricare il numero 2 nel registro A:

TRA = $ 2

Quindi il carattere $ distingue il numero due, operando, dal l'indirizzo della cella 2. Riconosciuto il due come operando, il programma assemblatore lo memorizza in una cella, il cui indirizzo sostituisce poi l'operando stesso nella istruzione. Perciò l'effetto dell'istruzione precedente è il medesimo che se si fosse scritto:

TRA COST

dove in altra parte del programma compariva la pseudoistruzione:

COST DEC 2

Questo programma considera tre tipi di operandi: numeri interi decimali, numeri interi ottali e caratteri alfanumerici.

i) decimali. Si identificano scrivendo =D in colonne 13/14, e non possono superare il valore di 262144, pari a 10^6 ottale.

ii) ottali: si identificano scrivendo =0, sempre in colonne 13/14, e devono consistere di un massimo di sei cifre ottali, esclusi cioè 8 e 9

iii) alfanumerici: si identificano scrivendo =A, ancora in colonne 13/14; i tre caratteri successivi sono considerati dati alfanumerici, e come tali riportati in memoria secondo il codice BCD dell'IBM 7090.

Tanto per esemplificare, una istruzione

(cella 16)	TRA	20

può dar luogo alla

(cella 16)	TRA	147

e in questo caso il contenuto della cella 147 sarà:

(cella 147)	000024

essendo $24_8=20$; mentre una istruzione

TRA =ABED

provoca la comparsa, ad esempio, di

TRA 417

La cella 417 conterrà:

(cella 417)	202322

poichè in BCD il numero ottale 20 equivale a B, 23 corrispondente ad E e 22 a D.

Nella lista del programma assemblato sono riportate tutte le parole create dall'assemblatore, dopo l'ultima istruzione del programma. La specifica di due dati eguali non porta alla creazione di due parole contenenti lo stesso dato; si usa due volte lo stesso indirizzo,

```
          TRA    =064
    ...   ...    ...
          TRB    =D52
    ...   ...    ...
          TRA    =A00U
```

saranno assemblate, ad esempio, come se fossero

```
          TRA    145
    ...   ...    ..
          TRB    145
    ...   ...    ...
          TRA    145
```

e il contenuto della cella 145 sarà

(cella 145)   000064

Nel caso in cui la memoria del CANE non fosse sufficiente alla creazione di tutte le parole richieste, verrebbe stampato un messaggio di errore. E' da notare, comunque, che queste parole sono create a partire dalla cella il cui indirizzo si trova nel contatore di locazione alla fine del programma: questo può portare come conseguenza che alcune istruzioni precedentemente assemblate vengano cancellate. Sarà cura di chi scrive i programmi in SIMBOLCANE evitare casi tipo:

```
                    ...      ....    .....
(cella 132)                  MEA     B
                    ...      ...     ....
                             ORIG    127
(cella 127)                  TRA     =D3
(  "   130)                  MEA     B,1
(  "   131)                  MEA     C
                             FINE    VIA
```

la istruzione numero 132 viene cancellata dalla prima stringa alfanumerica assemblata.

PSEUDO-OPERAZIONI

Il programma di assemblaggio simbolico del CANE comprende nove pseudo-operazioni, ovvero ORIG (origine), BCS (blocco che comincia col simbolo), DEC (decimali), OTT (ottali), FINE, ALFA (alfanumerici),PASSA (trasferimento a subroutine, con eventuali parametri), SALVA (salvataggio dei registri indici specificati,e automatico per il registro 7), TORNA (rientro dalle subroutines). Esse vengono eseguite durante il primo passo (ORIG, BCS, ALFA, FINE, OTT, DEC, SALVA), o durante il secondo (TORNA), o in parte nel primo e in parte nel secondo (PASSA).

ORIG

La pseudo-operazione ORIG ridefinisce il valore del contatore di locazione, detto anche contatore di istruzioni, attribuendo il nuovo valore alla etichetta, se presente. Tale valore si ricava dal contenuto del campo indirizzo: in esso non possono apparire simboli non definiti, o specifiche di registri. Il valore che si ottiene calcolando il contenuto del campo indirizzo viene riportato a modulo 511. Il formato della pseudo-operazione ORIG è

etichetta	operazione	indirizzo
simbolo o bianco	ORIG	espressione non contenente simboli non definiti

BCS

La pseudo-operazione BCS incrementa il valore del contatore di locazione di quanto specificato nella parte indirizzo: in esso, debbono comparire espressioni formate solo da numeri e/o simboli già definiti.

Come risultato dell'incremento del contatore di locazione si ha che un blocco di parole di memoria è riservato da questa pseudo operazione, ed è identificato dall'etichetta eventualmente presen te. Si noti che, mentre col BCS il blocco è identificato mediante la prima parola, un uso opportuno della ORIG permette di identificare una serie di parole di memoria mediante l'ultima di esse, realizzando così l'analogo di quello che fa il BES (block ending with symbol) nel MAP. Questo modo di operare è utile sopratutto quando i registri indici funzionano sottraendo il loro contenuto

dagli indirizzi che si devono indiciare.

Il formato della pseudo BCS è

etichetta	operazione	indirizzo
simbolo o bianco	BCS	espressione

OTT (DEC)

Vi sono due pseudo-operazioni che introducono in memoria dati numerici, espressi nel programma in forma ottale (o decimale). In ambedue, il campo indirizzo può essere diviso in più sottocampi, fino a sessanta, separati tra loro da virgole: ognuno di essi può contenere un numero ottale (o decimale), di non più di sei cifre, oltre al segno. Un campo indirizzo bianco viene assemblato come se contenesse uno zero, così come un segno seguito da una virgola; anche due virgole consecutive vengono interpretate come se tra loro ci fosse uno zero. Una eventuale etichetta viene assegnata come riferimento alla prima delle parole assemblate, quella che si trova cioè nel primo sottocampo a partire da sinistra; gli altri sottocampi sono poi assemblati in sequenza. Ad esempio :

(cella 145) OTTALE OTT 17,,-3,+,1

produce la sequenza

(cella 145) 000017

(cella 146) 000000

(cella 147) 777775

(cella 150) 000000

(cella 151) 000001

mentre all'etichetta OTTALE viene assegnato il valore 145

Il formato della pseudo-operazione è:

etichetta	operazione	indirizzo
simbolo o bianco	OTT (DEC)	fino a 60 sotto campi, separati da virgole.

ALFA

La pseudo-operazione ALFA introduce in memoria dati espressi nel programma in forma alfanumerica, secondo il codice BCD del- l'IBM 7090. Con questo codice, tutti i caratteri, alfabetici, nu merici o di altro tipo, sono rappresentati nella memoria del cal-

colatore da numeri ottali che vanno da 00 a 77. In questo modo, ogni parola del CANE può contenere da uno a tre caratteri alfanumerici. Il numero di parole da assemblare, o numero di conteggio, è specificato nelle colonne 13/14: tale numero può variare da un minimo di 1, a un massimo di 19, poichè sono disponibili solo 57 colonne, quelle che vanno dalla 16 alla 72 compresa. Il numero di conteggio deve comunque essere formato da due cifre, cioè non si può scrivere 7, ma si deve scrivere 07. A colonna 15 deve sempre esserci una virgola. Qualunque dato si estenda al di là di quelli specificati dal numero di conteggio è considerato come commento. Una eventuale etichetta è assegnata come riferimento alla prima informazione alfanumerica memorizzata; le altre sono messe in sequenza. Ad esempio, una istruzione del tipo

(cella 32)   DATA ALFA   04, INFORMAZIONI ALFAN.

dà luogo alla sequenza

(cella 32)   314526

(cella 33)   465144

(cella 34)   217131

(cella 35)   464531

che corrisponde ai caratteri INFORMAZIONI, mentre ALFAN è considerato come commento.

Il formato di questa pseudo-operazione è

etichetta	operazione	indirizzo
simbolo o bianco	ALFA	due sottocampi, separati da una virgola 1) numero di conteggio 2) dati alfanumerici

FINE

La pseudo-operazione FINE segnala la fine del programma sorgente; essa deve sempre essere presente, e deve essere l'ultima scheda del programma sorgente. Questa pseudo-operazione fa passare il programma di assemblaggio alla sua seconda parte, e specifica nella sua parte indirizzo il punto di partenza del programma sorgente, cioè la cella di memoria da cui si deve partire per eseguire il programma. E' poi compito del programma di assemblaggio trasmettere questa informazione al simulatore. L'indirizzo può essere numerico, simbolico, o formato da una espressione: non devono comparirvi comunque simboli virtuali. Una eventuale etichetta nel campo della pseudo FINE verrà riferita alla prima delle stringhe alfanumeriche eventual

mente create del programma assemblatore: in questo modo esse possono essere identificate durante la scrittura del programma sorgente. Ad esempio, per un programma in cui compaia, come prima stringa, la

```
        (cella 23)              TRA         =D67
```

e la cui fine sia indicata da

```
        (cella 435)  ALT        FINE        142
```

la etichetta ALT ha il valore 435, cioè quello della posizione in cui viene messa la prima stringa alfanumerica creata dal programma

```
        (cella 435)  ALT        000567
```

Quindi scrivere

```
                                TRA         ALT
```

equivale a scrivere

```
                                TRA         =D567
```

Questo programma poi sarebbe eseguito a partire dalla posizione 142. Il formato di questa pseudo-operazione è

etichetta	operazione	indirizzo
simbolo o bianco	FINE	espressione non virtuale

PASSA

La pseudo-operazione PASSA, assieme alle associate SALVA e TORNA, serve a produrre una sequenza standard di istruzioni del CANE per il passaggio ai sottoprogrammi e per il ritorno al programma principale. La pseudo-operazione PASSA è sempre del tipo

PASSA SOTP(A1,A2,...An)

dove SOTP è il nome del sottoprogramma richiamato, mentre A1,..An sono gli argomenti che si vogliono passare al sottoprogramma. L'assemblaggio di questa pseudo-operazione genera la seguente lista di istruzioni:

```
SUB     SOTP,7
SLT     *+n+1
NOP     A1
...     .....
NOP     An
```

n essendo il numero di argomenti che si vogliono passare al sottoprogramma SOTP. Gli argomenti devono essere separati da virgole, e racchiusi tra parentesi. Supponendo ad esempio di voler passare i parametri A e B al sottoprogramma CONTO, basta scrivere la

pseudo-istruzione

```
(cella 67)   X     PASSA    CONTO(A,B)
```

A partire dalla posizione di questa pseudo-istruzione l'assemblatore genera la seguente lista di istruzioni:

```
(cella 67)   X     SUB      CONTO,7
(cella 70)         SLT      73
(cella 71)         NOP      A
(cella 72)         NOP      B
```

Il registro indice 7 è sempre quello usato per il salto al sottoprogramma, e per questo motivo tale registro indice verrà poi automaticamente salvato da un'altra pseudo-operazione, la SALVA. Il numero di argomenti che si possono passare a un sottoprogramma è limitato solo dal fatto che si ha a disposizione una sola scheda, e quindi un numero di colonne minore di 60. Gli argomenti possono essere stati definiti ovunque nel programma, prima o dopo la comparsa della pseudo-operazione PASSA in cui vengono richiamati; essi devono comunque essere sempre individuati da una etichetta. Non è permessa una chiamata di questo tipo:

(cella 45)   A   BCS   6

...   .

PASSA   (A,A+1,A+2)

Il formato della PASSA è:

etichetta	operazione	indirizzo
simbolo o bianco	PASSA	due sottocampi 1°)nome del sottoprogramma 2°)argomenti

SALVA

Questa pseudo-operazione è intesa al salvataggio dei registri indici specificati, oltre che del registro indice 7, salvato automaticamente. Essa sarà in genere la prima istruzione di un sottoprogramma; e sarà del tipo

(cella 61)   A   SALVA   R1,R2,..,Rn

dove R1,R2,R3,...Rn, con n $\leq$ 6, sono numeri tra 1 e 6 che indicano i registri da salvare. Se n $\geq$ 6, solo i primi sei registri specificati sono salvati.

Comunque, una istruzione di salvataggio genera la seguente lista

di istruzioni:

```
(cella 61)      A    SLT     *+n+3
(cella 62)           TRX     *+2n+2,R1
(cella 63)           TRX     *+2n,R2
(cella 64)           TRX     *+2n-2,R3
....            .... ...     ......
(cella 62+n)         TRX     * + 2,7
(cella 63+n)         SLT     ,7
(cella 64+n)         MEX     * ,7
(cella 65+n)         MEX     *, Rn
....            .... ....    .....
(cella 63+2n)        MEX     *,R2
(cella 64+2n)        MEX     *,R1
```

Le istruzioni sono in sequenza. E' da notare come i registri vengano salvati in ordine inverso a quello specificato dalla istruzione. Ad esempio:

```
(cella 234)     A    SALVA   1,5
```

genera la seguente lista di istruzioni

```
(cella 234)      A      SLT      241
(cella 235)             TRX      243,1
(cella 236)             TRX      242,5
(cella 237)             TRX      241,7
(cella 240)             SLT      ,7
(cella 241)             MEX      241,7
(cella 242)             MEX      242,5
(cella 243)             MEX      243,1
```

Il registro indice 7, come abbiamo detto, è salvato automaticamente per primo. Una istruzione tipo

```
SALVA    1,7,3
```

diviene

```
SLT      * + 7
TRX      * + 6,1
TRX      * + 4,3
TRX      * + 2,7
SLT      ,7
MEX      *,7
```

```
MEX      *,3

MEX      *,1
```

E' da notare ancora che, mentre una istruzione come

```
SALVA    4,5,6,1,2,3
```

salva tutti e sette i registri indici,

```
SALVA    7,4,5,2,3,1,6
```

non salva il registro indice 6.

Il formato per la SALVA è:

etichetta	operazione	indirizzo
simbolo o bianco	SALVA	Fino a 6 registri indici, separati da virgole.

TORNA

Questa pseudo-operazione genera una sola istruzione,che permette, grazie alla associata SALVA, il ritorno al programma di chiamata da un programma richiamato. Ad esempio:

```
TORNA    SOTP
```

genera

```
SLT      SOTP+1
```

dove SOTP è l'etichetta associata ad una pseudo-operazione SALVA.

E' evidente quindi che questa pseudo va adoprata insieme con la SALVA.

Il suo formato è

etichetta	operazione	indirizzo
simbolo o bianco	TORNA	simbolo

CAPITOLO II

TABELLE DEI SIMBOLI

Uno dei vantaggi che si hanno facendo uso di un programma assemblatore consiste nel fatto che si possono indicare le locazioni di memoria e i codici operativi per mezzo di nomi simbolici, invece che con i loro indirizzi numerici assoluti e coi loro codici numerici, rispettivamente. Il vantaggio che un tal modo di procedere dà all'utente si può evidenziare molto semplicemente dicendo che è certamente più facile ricordare un codice operativo mnemonico che uno numerico, specie in casi in cui le operazioni assommino a qualche centinaio, e che è infinitamente più facile riferirsi ad una istruzione, cioè ad una cella di memoria, per mezzo di un nome che attraverso il suo indirizzo, indirizzo che sarebbe necessario calcolare tenendo conto del modo in cui il programma da assemblare è stato scritto; e risalire in questo modo all'indirizzo di una cella, in un programma che può avere anche migliaia di istruzioni, è impresa estremamente difficoltosa, che facilmente porta a commettere errori. Oltre a questo, si pensi alla comodità, ad esempio, di poter indicare in uno schema a blocchi

i punti di ingresso o uscita dai blocchi con dei simboli che si ritrovano pari pari nel programma; e finalmente bisogna sottolineare il vantaggio che deriva dall'avere degli indirizzi simbolici quando si apportano correzioni ad un programma.

Molto spesso tali correzioni si fanno aggiungendo o togliendo istruzioni oltre che modificando quelle già scritte: in conseguenza di ciò, tutte le istruzioni che vengono dopo la correzione sono spostate, quindi tutti gli indirizzi che si riferivano a queste debbono essere modificati. Ad esempio, se si vuole inserire l'istruzione SOA 33 tra la 16^{a} e la 17ª istruzione del programma:

```
.................
016   TRA   41
017   ADA   10
020   ADA   24
021   MEA   31
.................
```

si debbono modificare tutti gli indirizzi $\geq$ 17

```
.................
016   TRA   42
017   SOA   33
020   ADA   10
021   ADA   25
022   MEA   32
.................
```

La presenza di nomi simbolici come indirizzi toglie questa preoccupazione al programmatore: sarà l'assemblatore ad attribuire ai simboli i nuovi valori, durante la costruzione della tabella degli indirizzi.

Per realizzare questa possibilità, è necessario e sufficiente che si stabilisca una corrispondenza, che non sempre sarà biunivoca, tra simboli e numeri: cioè tra codici operativi mnemonici e numerici, corrispondenza necessariamente biunivoca, e tra etichette ed indirizzi di memoria: in questo ultimo caso può darsi che a più etichette corrisponda un solo indirizzo.

Il modo migliore per stabilire queste corrispondenze consiste nella costruzione di tabelle che contengono e i nomi e i numeri ad essi corrispondenti, legati tra loro da qualche legge (fig.3).

OPC	002
OPD	003
OPE	004
OPF	005
OPG	011
OPH	012

3 .1 tabella dei codici operativi

ANNA	021
FRA	164
LOOP	021
SALTO	206

fig.3 3 .2 Tabella dei simboli

Per quanto riguarda le etichette, come già abbiamo detto, ogni simbolo può assumere un valore numerico qualunque, nell'ambito delle dimensioni del calcolatore in oggetto (per il CANE, non si può superare il valore 777_8 poichè la memoria del CANE è appunto composta di 777_8 voci); inoltre non c'è alcuna relazione, numerica o di altro tipo, tra i vari simboli usati in un programma, a meno che l'utente non stabilisca diversamente, facendo uso di speciali pseudo-operazioni. Ci sono tuttavia alcuni assemblatori che stabiliscono dei legami tra certi simboli, come A 0024, A 0376 e A 2214.; ma il caso non è molto frequente.

Abbiamo detto che la corrispondenza operazioni - codici operativi è biunivoca, l'altra no: oltre a questo, è da notare come nel primo caso si abbia a che fare con un numero predeterminato di simboli, cioè i codici operativi mnemonici, mentre nel secondo il numero delle etichette che compariranno nella tabella varia, in modo puramente casuale, da programma a programma e dipende in definitiva, oltre che dalla lunghezza e complessità del programma, dalla fantasia del programmatore.

Questa differenza tra i due casi porta logicamente a indivi-

duare due tipi diversi di tabelle. La prima, quella dei codici operativi, ha un numero fisso di simboli, o di entrate, (come si dice intendendo che ogni simbolo rappresenta una entrata nella tabella), ed in essa la corrispondenza simbolo-numero è biunivoca. La seconda ha invece un numero di entrate variabile, a seconda di quante etichette compaiono nel programma da assemblare, e presenta una corrispondenza univoca tra etichette e numeri, che deve esser rilevata poichè generalmente se a più etichette corrisponde uno stesso numero vuol dire che c'è stato un errore di programmazione.

Il programma assemblatore del CANE fa uso di una tabella del primo e di una del secondo tipo.

In ambo i casi, si deve risolvere il problema di come far corrispondere al simbolo il valore adatto, che può essere un codice operativo numerico o l'indirizzo di memoria cui si riferisce una etichetta; se poi il numero di entrate nella tabella è variabile, si presenta un altro problema: come organizzarla in modo da far entrare in essa solo nuovi simboli. E' sempre da tener presente che le tabelle devono esser strutturate in modo tale da permettere, in

un secondo tempo, la ricerca in esse dei simboli e la loro sostituzione coi numeri corrispondenti, nella maniera più rapida possibile, e senza possibilità di equivoci.

Il primo problema può essere risolto in una infinità di modi: se i simboli non sono lunghi, il numero può trovarsi nella stessa cella in cui è trascritto il simbolo; oppure nella cella immediatamente successiva; o ancora dopo un numero K di celle, K dipendendo dalla grandezza della tabella: in questo caso è come se si costruissero due tabelle, la prima fatta di soli simboli, la seconda di soli numeri e spostata di K posti, così che all'entrata n corrisponda in numero che sta in (n + K). Questo problema, tuttavia, non presenta aspetti particolarmente interessanti per noi, quindi ci limiteremo ad accennarvi quando descriveremo i metodi usati dai programmi assemblatori per costruire le tabelle, entrando in dettaglio solo quando esamineremo in particolare il nostro programma di assemblaggio simbolico.

Benchè si possono individuare due tipi ben distinti di tabelle, ci sono alcuni metodi di costruzione e di ricerca dei simboli che, in via teorica, sono ugualmente applicabili: nel secondo caso si

deve fare attenzione a non inserire simboli già presenti, cioè a non creare entrate doppie. In pratica, però, non sono ugualmente convenienti, ed è per questo che si parla di metodi adatti a tabelle del primo o del secondo tipo.

E' comune il fatto che le tabelle contengono i simboli in forma alfanumerica, e che i simboli siano, come si dice, 'autoaggiustati' a destra o a sinistra, cioè eventuali caratteri spazio completano i simboli stessi, a destra o a sinistra. Ad esempio, per simboli formati da sei caratteri, un nome, codice operativo mnemonico o etichetta, del tipo

ADA

viene 'autoaggiustato' a destra come:

ADA^{bbb}

dove ^{b} indica carattere spazio.

Un metodo teoricamente applicabile al primo tipo di tabelle consiste nel costruirle in maniera completamente casuale; poichè il numero di entrate è prefissato, non è possibile ritrovarsi con entrate doppie.

La ricerca dei simboli nella tabella può poi essere fatta in diversi modi: il più semplice consiste nell'esaminare uno per uno tutti i simboli della tabella, confrontandoli con quello in esame, sostituendogli, quando il confronto è positivo, il valore numerico che la tabella associa a quel simbolo. In media, se la tabella contiene N simboli, e se tutti questi appaiono con la stessa frequenza, sono necessari N / 2 confronti. In pratica, poi, non tutti i simboli appaiono con la stessa frequenza, per cui è conveniente disporli nella tabella di ordine di frequenza decrescente, e cominciare il confronto dal primo simbolo della tabella stessa.

Un secondo metodo è quello della "ricerca logaritmica". In questo caso i simboli sono considerati numeri interi, e disposti nella tabella in ordine di grandezza. La ricerca si effettua confrontando il valore numerico del simbolo in esame, con quello del punto di mezzo della tabella. Si determina così se la ricerca debba proseguire nella prima o nella seconda metà della tabella stessa (a meno che, caso fortunato, i valori non siano uguali). Ripetendo il procedimento, ci si riduce successivamente a un quarto, un ottavo della tabella, e così via.. Questo metodo richiede in media $\log_e N$

confronti, cioè un numero estremamente più basso che nel primo caso.

Un terzo metodo consiste nell'ordinare la tabella in ordine alfabetico, e nel costruire una seconda tabella che indichi gli indirizzi di partenza delle varie sezioni della prima, intendendo con sezioni i gruppi di simboli che hanno la stessa iniziale. La ricerca, una volta individuata la sezione, può proseguire in uno dei due modi detti.

Questi ultimi due metodi sono applicabili anche a tabelle aventi un numero variabile di entrate, poichè si può facilmente controllare se i simboli che vengono man mano inseriti sono nuovi o meno: in effetti, essi presentano il notevole vantaggio di disporre i simboli in ordine alfabetico, e quindi permettono di scrivere una lista dei simboli con tale ordine. Come contropartita, l'inserimento di nuovi simboli comporta un notevole dispendio di tempo: infatti una porzione delle tabelle deve essere spostata ogni volta che viene inserito un simbolo nuovo (fig.4).

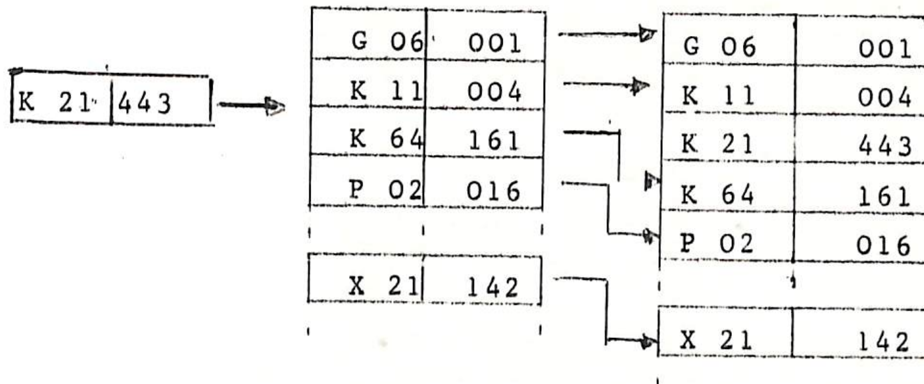


fig.4

Nel caso poi in cui si usi il terzo metodo, anche gli indirizzi che indicano i punti di inizio delle varie sezioni che sono state spostate devono essere cambiati (fig.5).

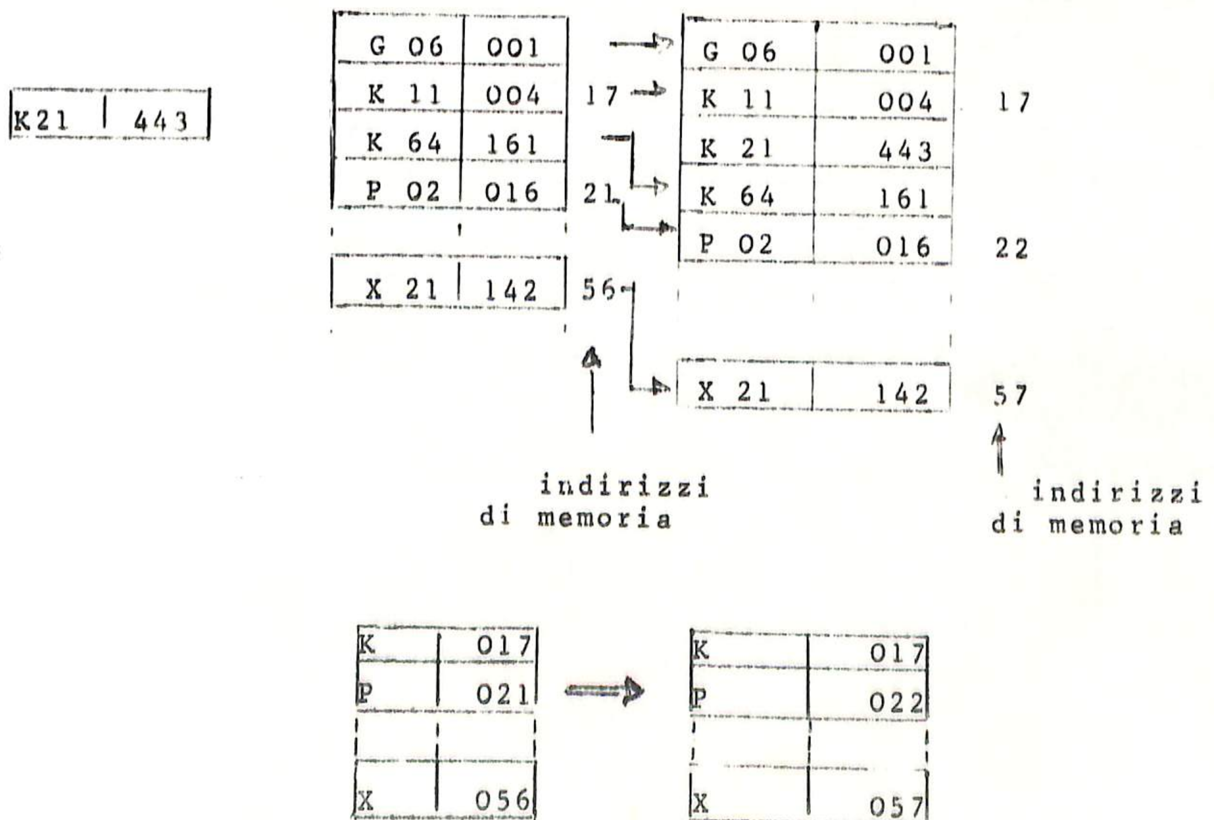


fig.5

Per esemplificare la ricerca dei simboli coi metodi ora descritti, supponiamo di aver individuato posizione e lunghezza di una sezione, quella che ha per iniziale A, col terzo metodo, e cerchiamo il simbolo A 18 (fig.6).

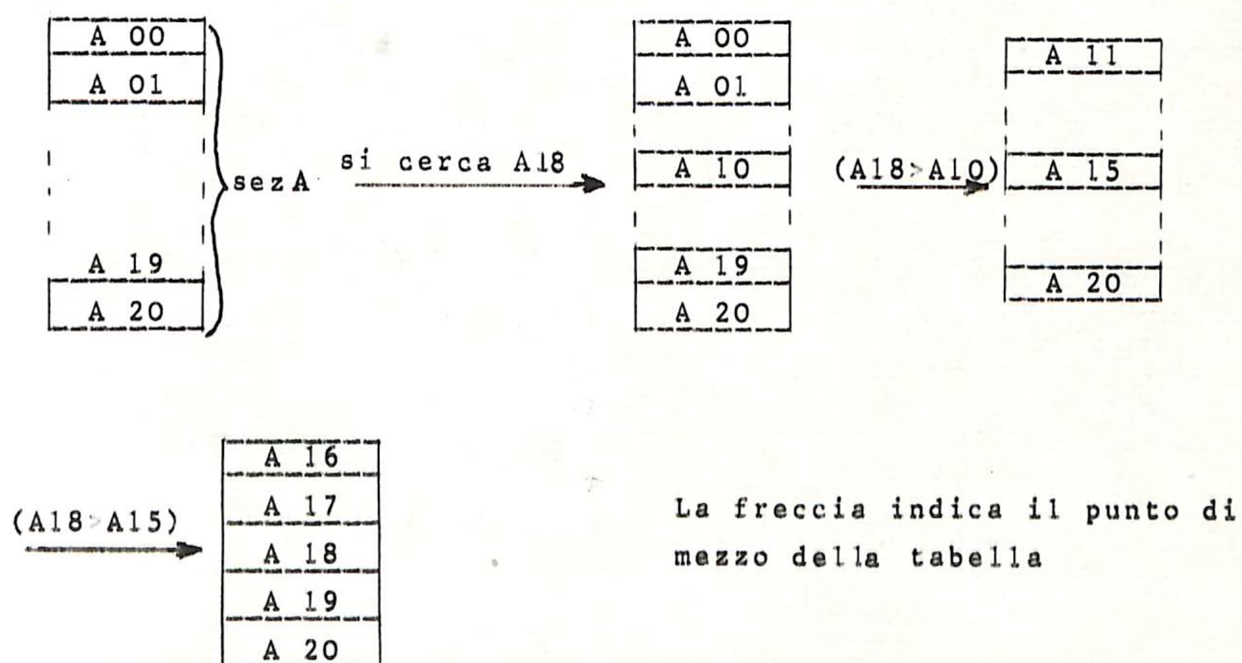


Fig. 6

Trascuriamo, per chiarezza, di associare ai simboli i loro valori numerici; come si vede, dopo solo tre confronti si è individuato il simbolo.

Il programma di assemblaggio del CANE usa un metodo diverso da quelli descritti per assegnare i 64 (77_8) codici numerici, da 00 a 77_8, alle operazioni di macchina. Ai simboli che rappresenta

no le operazioni si associano dei numeri, che si ottengono sommando i valori numerici delle singole lettere componenti i simboli, intesi come caratteri BCD (ad es., ad ALT si associa $21_8 + 43_8 + 63_8 = 147_8$).

Si ricavano così 42 numeri (la corrispondenza simbolo - numero non è biunivoca) che vanno da 54 (ADA) a 165 (che corrisponde a ben 5 simboli, NUM, SBZ, SUG, SODA, TDAN). Si costruisce allora una tabella di 115 parole (usando uno dei 3 metodi descritti prima, sarebbero state sufficienti 64 celle di memoria: ma quello che importa non è risparmiare memoria, bensì tempo) che contengono, per le operazioni per cui la corrispondenza simbolo - numero è biunivoca, il codice operativo corrispondente (nel decremento) e l'indirizzo di una cella in cui la operazione stessa è stata scritta in forma alfanumerica (nella parte indirizzo); se ad una certa operazione corrisponde il numero k, le informazioni suddette sono messe nella $(K-54)^a$ parola della tabella. Le operazioni per cui la corrispondenza simbolo - numero non sia univoca richiedono un trattamento più complesso: in questo caso, se il numero è n, nella $(n - 54)^a$ parola ci sono un segno, che indica trattar

si di corrispondenza non biunivoca, (7 nel prefisso), un numero, che indica quante operazioni danno questo valore (da I a 5 nel decremento), e l'indirizzo di una cella che contiene i codici operativi in oggetto (nella parte indirizzo). Nelle celle immediatamente precedenti questa ultima sono scritte, in alfanumerico, le operazioni stesse. Ad esempio, la I[a] parola della tabella sarà (ADB⟷55)

```
                    PZE         ADB,,16
```

La locazione ADB, che fa parte della tabella, conterrà

```
        ADB         BCI         1,ᵇ ADB
```

mentre la 51[a] (105-54) parola sarà:

```
                    SEVEN       CF51,,2
```

con

```
                    BCI         2,ᵇSOAᵇᵇᵇSBN

        CF51        OCT         5115000000
```

poichè SOA ⟶ 105, SBN ⟶ 105

La ricerca nella tabella è estremamente rapida: calcolato il valore da associare al simbolo, gli si sottrae 54, e si va a prendere la parola che sta nella tabella nella posizione indica

ta dal numero così trovato. Se il prefisso è zero si confronta il contenuto dello indirizzo col simbolo in esame, e, nel caso siano eguali, il decremento della parola contiene il codice operativo. Se il prefisso non è zero, si esaminano le celle che precedono quella il cui indirizzo si trova nella parola, tante quante ne indica il decremento, e, se una di esse è uguale al simbolo in esame, ad esso viene attribuito, in maniera opportuna, uno dei codici operativi che sono il contenuto dell'indirizzo della parola.

Rifacendoci agli esempi precedenti, al simbolo ADB viene assegnato il valore (55-54)=1; il confronto col contenuto della cella ADB è positivo, dunque ad ADB è sostituito il valore 20_8. Al simbolo SBN, invece, è assegnato il valore(105 - 54)= 51. Il decremento della 51[a] posizione della tavola, 2, è posto in un registro indice, il registro indice 2, dopo di che si confronta SBN col contenuto della cella di indirizzo (CF51-2) e, successivamente, il confronto essendo risultato negativo, col contenuto di quella precedente CF51, decrementando RX2 di 1. Il confronto è ora positivo: a SBN viene assegnato il valore 51_8, che si ottiene ruotando a

sinistra CF51 di tanti byte (1) quanti ne indica RX2, e prendendo poi il primo byte a sinistra.

Questo metodo è molto particolare, e tra l'altro non potrebbe essere usato se a più di cinque, o, con qualche modifica, sei simboli corrispondesse lo stesso numero: esso è stato realizzato per questo particolare programma.

Come criterio per valutare i vantaggi di questo metodo, si possono esaminare i seguenti dati:

a	49	numeri corrisponde un solo simbolo
a	11	" corrispondono due simboli
a	3	" " tre "
a	1	numero corrispondono cinque simboli

Ammesso che tutti i simboli si verifichino con la stessa frequenza, il numero medio di confronti necessari con questo metodo può essere calcolato come

$$\frac{1}{N} \sum \text{numeri} \cdot \text{simboli} = \frac{1}{64} \left[49+11 \cdot 2+3 \cdot 3+5 \right] = 1,3$$

Il metodo si presenta quindi estremamente rapido, poichè sfrutta in maniera del tutto originale le possibilità del calcolatore

---

(1) 1 byte = sei bits

IBM 7090, su cui, come già detto, è stato scritto questo programma.

Un principio generale è invece quello di far corrispondere ad ogni simbolo un numero, ottenuto facendo uso di una certa funzione, come vedremo parlando delle 'hash tables'.

In esse, una tavola di grandi dimensioni è divisa in K parti, ciascuna abbastanza piccola da permettere una ricerca lineare.

Per far ciò, si elabora una funzione che ad ogni simbolo associ un valore che vada da uno a K, così da selezionare una delle K sezioni: in essa poi si confrontano tutte le entrate col simbolo in esame, in modo da inserirlo nella prima entrata vuota, (se il simbolo non è già stato inserito).

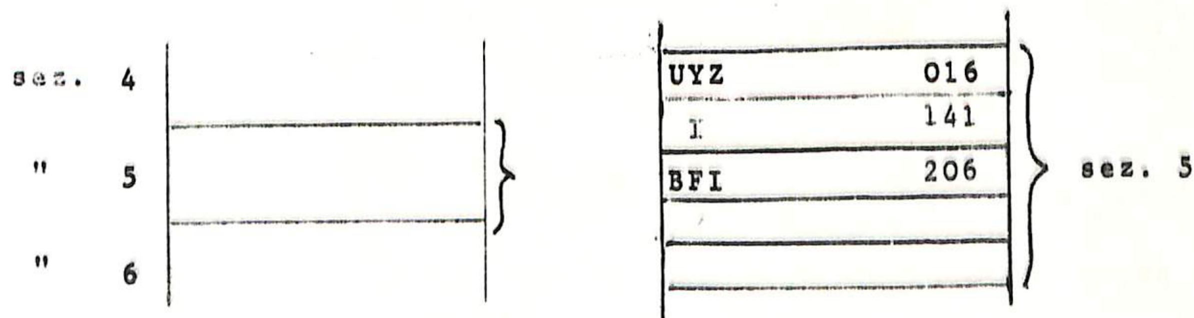
Per la ricerca dalla tabella si procede nello stesso modo; la funzione determina la sezione da guardare, e in essa si esegue una ricerca lineare. In ogni parola di questa sezione sono associati un simbolo e un numero. Ovviamente,le varie parti in cui è divisa la tabella non sono tutte uguali: ma un'appropriata scelta della funzione permette di ottenere una distribuzione abbastanza omogenea dei simboli. Per fare un esempio, la tabella può essere divisa in

10 sezioni, associando ad ogni simbolo un numero tra zero e nove.

Per far questo, si possono, ad esempio, sommare tra loro i caratteri BCD che formano il simbolo, e dividere il numero così ottenuto, se è maggiore o uguale a 10, (se è minore lo si prende subito come valore associato al simbolo) per 10, prendendo il resto della divisione come valore da associare al simbolo. In questo modo, al simbolo che etichetta una certa cella, ad es.la 441,

$$VIA = 53 + 25 + 17 = 95$$

è associato il numero 5, così come 5 è associato a I o a NYZ. Allora, per inserire VIA nella tabella, si individua la sezione 5,



e si inserisce VIA nella prima entrata libera

UYZ	016
I	141
BFI	206
VIA	441

E' comunque più appropriato, per un programma assemblatore,

modificare la costruzione della tabella in modo da ridurre ancora il tempo necessario per la costruzione e per la ricerca successiva, rendendo K molto grande, cioè elaborando una funzione che possa assumere valori in uno spazio abbastanza ampio. E' anche necessario che tali valori abbiano una distribuzione di probabilità di tipo gaussiano, in modo da evitare nella tabella dei punti, diciamo così, di condensazione, cioè in cui confluiscano troppi simboli.

Il programma di assemblaggio del CANE adotta, per la costruzione della tabella delle etichette, un metodo che può essere considerato come un ulteriore perfezionamento di quello descritto. Ad ogni simbolo si associa il valore dato dalla somma dei prodotti dei valori numerici dei caratteri componenti il simbolo per il posto da essi occupato. Ad es.

$$ABC^{bbb} \rightarrow 21_8 \times 1 + 22_8 \times 2 + 23_8 \times 3 = 156_8$$

La funzione così scelta può dare numeri che variano da 17_8 = A00000 fino a 1257 = Z(((((. (Allo stato attuale tale simbolo non è permesso, e il più alto valore che possa raggiungere la funzione è 1197 = ZZZZZZ).

Una funzione di questo tipo ha un andamento gaussiano, come si ricava costruendola per punti. (V.Fig.7)

La costruzione della tabella procede in questo modo: sia m il valore della funzione per il simbolo S, e V il valore del contatore di locazione per S. Si esamina la m<u>sima</u> entrata della tabella: se è vuota, vi si pone S e si mette V nella entrata (m+1257), nella parte indirizzo, (la tabella ha una lunghezza 2x1257+2x512=3538, essendo 512 il numero massimo di simboli che si possono trovare in un programma scritto per il CANE; esso ha infatti 512 celle di memoria, e al massimo si possono etichettarle tutte una volta). Se l'm<u>sima</u> entrata è occupata e se il suo contenuto è diverso da S (se fosse uguale ci sarebbe un errore), si esamina la parte decremento della entrata (m+1257). Se è vuota, si cerca la prima parola libera tra le ultime 1024 (il cui indirizzo si trova in una cella che chiameremo PPL), e vi si mette S, dopo di che si mette V nell'indirizzo dato da (PPL)+1, mentre il contenuto di PPL va nel decremento di (m+1257), e il contenuto di PPL è incrementato di 2. In questo modo, il decremento di (m+1257) contiene l'indirizzo della prima cella in cui si trova un simbolo avente, come valore datogli dalla fun-

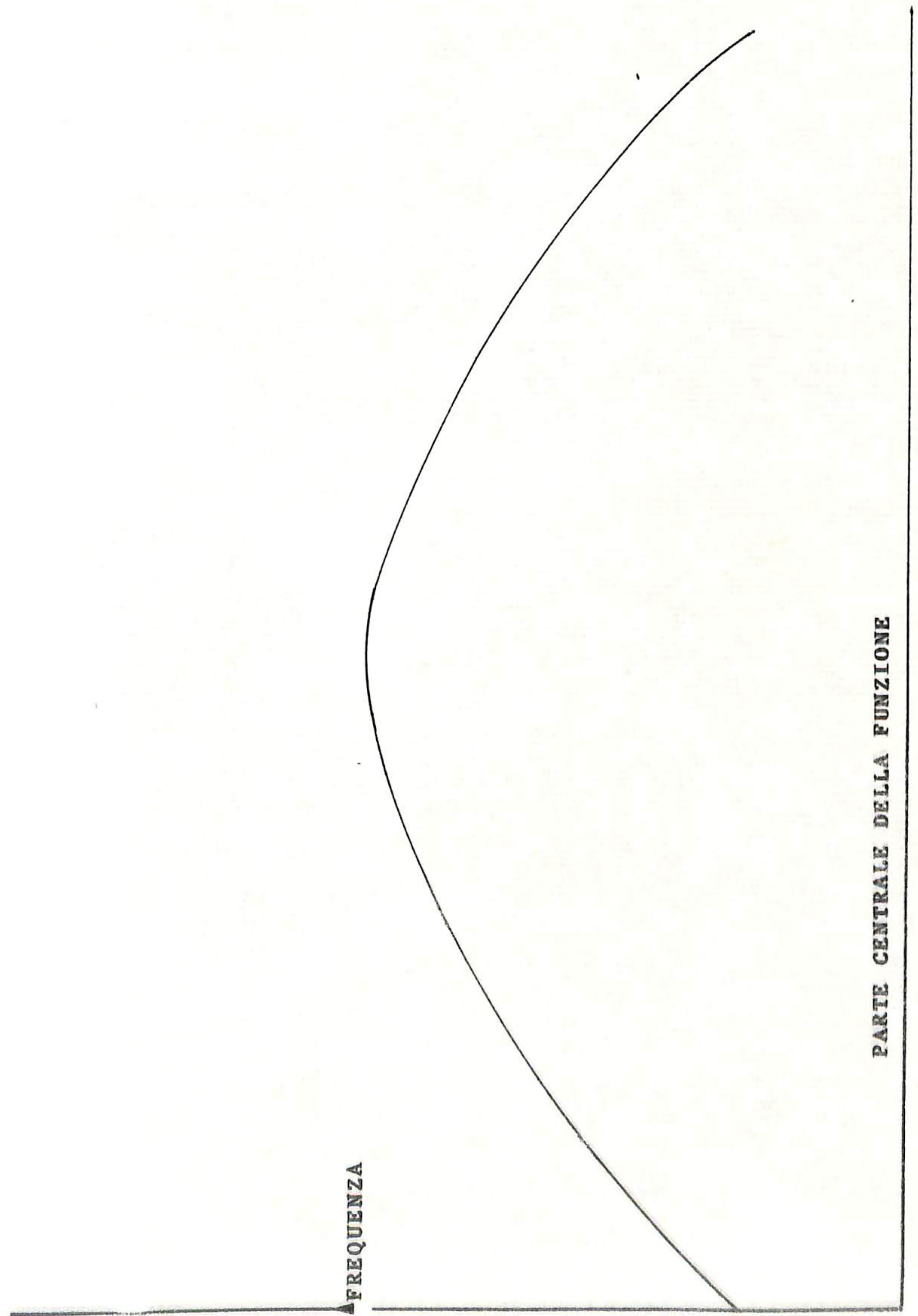
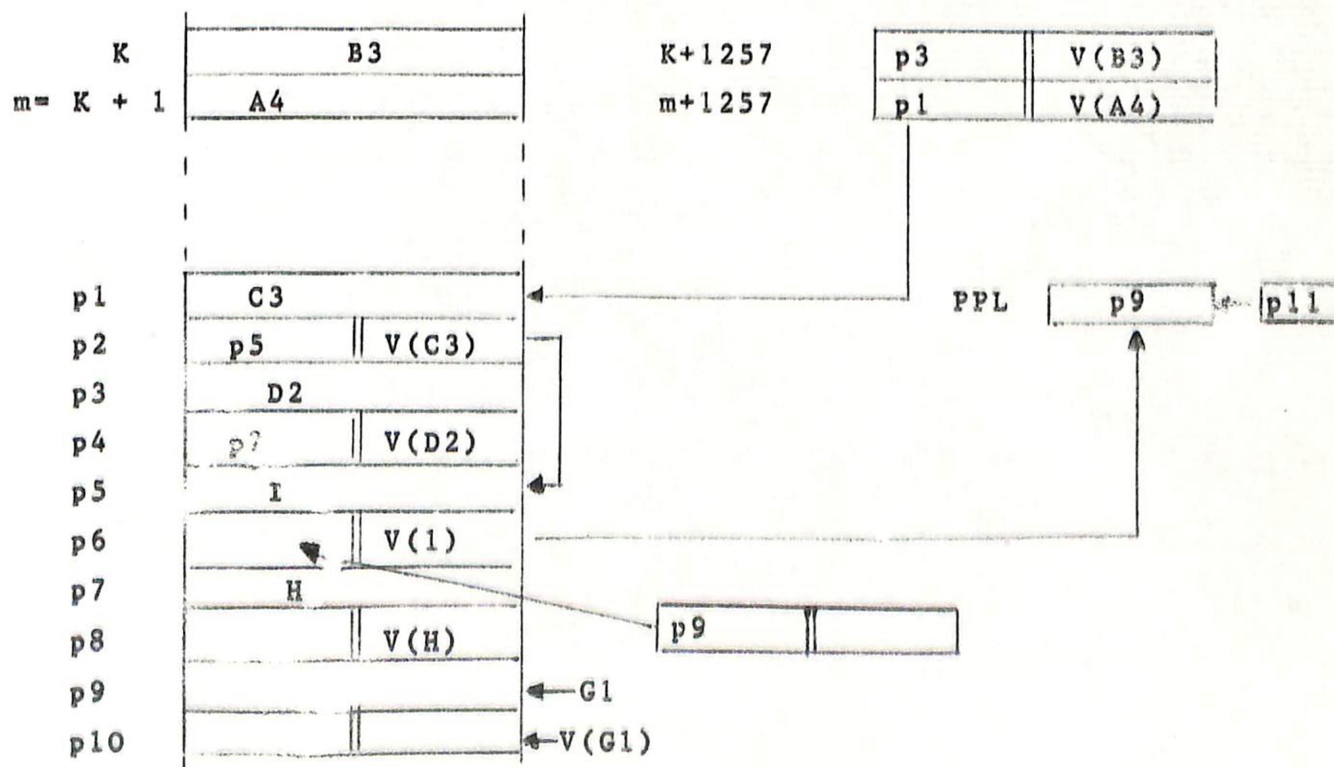


Fig. 7

zione, ancora m. Quindi, per mettere un simbolo nella tabella, sono effettuati solo i confronti con le entrate in cui sono simboli aventi lo stesso valore per la funzione. In questo modo, mentre nelle hash-tables all'aumentare del numero dei simboli aumentava molto il numero dei confronti da effettuare, ora tale numero aumenta in maniera meno sensibile. E' come se avessimo una freccia che ci indica il cammino da percorrere volta per volta. Ad esempio, si abbia da inserire il simbolo G1, per cui f(G1)=25, in una tabella che contiene già, tra gli altri, i simboli A4,G3,I, B3, D2, H per cui f(A4)=f(G3)=f(G1)=f(I)=25=m mentre f(H)=f(B3)=f(D2)=24=K



Esista poi la corrispondenza B3→V(B3), dove V(B3) è il valore

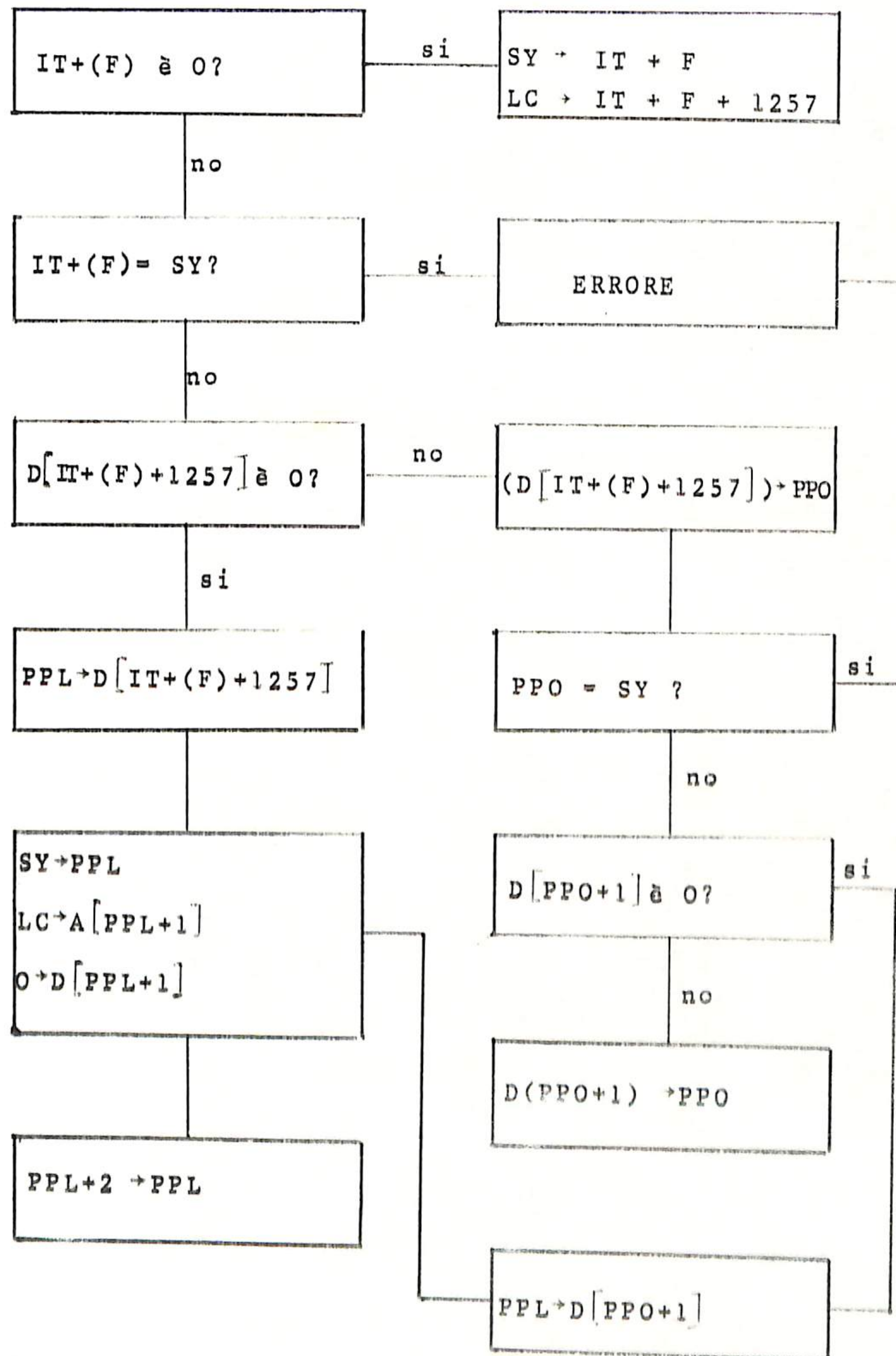
del contatore di locazione da associare a B3, e così per gli altri. f(G1) =25 = m. L'entrata m contiene A4 ≠ G1. Il decremento di (m+1257) contiene p1; p1 contiene C3, per cui f(C3) = m, ma C3 ≠ G1. Il decremento di p2 contiene p5, il cui contenuto è I ≠ G1, ma tale che f(I) = m. Il decremento di p6 è vuoto, per cui si guarda il contenuto di PPL, che è proprio p9, e quindi si mette G1 in p9, V (G1) nell'indirizzo di p 10, mentre nel decremento di p6 si mette p9 e in PPL va p11.

La ricerca nella tabella richiede ancora il calcolo della funzione, durante il quale si ricercano anche eventuali errori formali sui simboli e si aggiustano tagliandoli a sinistra cioè facendo in modo da non lasciare spazi a sinistra. Questo è importante perchè un simbolo definito come ^{bb}A in una istruzione, deve esser riconosciuto anche se è scritto come A^{bb}. Ad es

```
bbAbbb      TRA       *  +6
.........................
.........MEB......A....
.........................
```

Riportiamo, per maggiore chiarezza, gli schemi a blocchi per la costruzione della tabella, e per la ricerca dei simboli nella stessa.

Costruzione della tabella degli indirizzi.



PPL = IT + 2514

PPO = nome di una cella

IT = prima parola della tabella

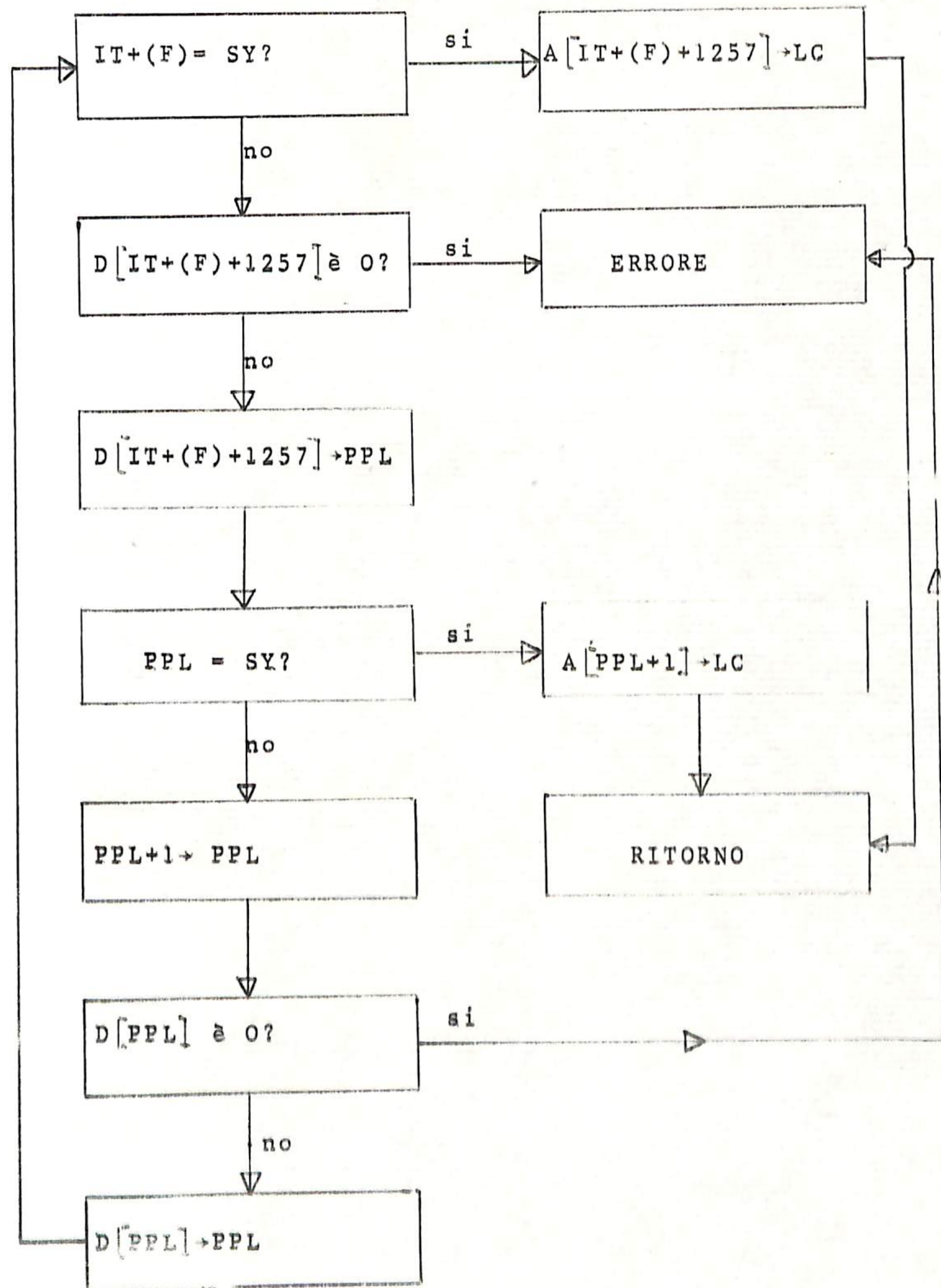
F = funzione associata al simbolo

A[X]= indirizzo di X

D[X]= supplemento di X

./.

Ricerca nella tabella dei simboli.



D[X] = decremento di X

LC = contatore di istruzione

SY = simbolo

(X) = contenuto di X

CAPITOLO III

STRUTTURA DEL PROGRAMMA

Un programma assemblatore può essere scritto in modo da funzionare con una sola passata, oppure con due o più: si parla allora di assemblatori a una, due, tre passate, e così via (benchè non sia facile trovarne a più di due). Con passata si intende il procedimento di lettura dell'intero programma sorgente: quindi lo assemblatore a due passate legge due volte il programma sorgente, anche se quasi mai la lettura avviene tutte e due le volte dalle unità di ingresso lente (lettore di schede); in genere la seconda volta si legge da nastri o dischi o addirittura, come nel nostro caso, direttamente dalla memoria veloce.

Tra i programmi assemblatori ad una passata, e quelli a due, vi sono delle differenze di impostazione concettuali, oltre che di ordine tecnico ed esecutivo. Infatti la scelta tra i due è condizionata in buona parte da motivi esterni al programma, quali la dimensione della memoria veloce e la rapidità delle memorie ausiliarie, nastri o dischi. Nel caso in cui si abbiano memorie ausiliarie lente la scelta si orienta naturalmente su un programma che sia

in grado di funzionare leggendo le schede del programma sorgente una volta sola, e quindi nel minor tempo possibile. Un altro motivo preferenziale per la scelta di un programma assemblatore a una passata è dato dal modo di funzionare del calolatore: infatti con un programma di questo tipo il calcolatore deve funzionare nel modo "load and go", perchè al termine della lettura del programma sorgente nella memoria si trova già il programma oggetto.

Lo svantaggio maggiore di questi assemblatori consiste nel fatto che, poichè le istruzioni del programma sorgente vengono assemblate, e poste in memoria subito dopo, la zona di memoria che è lasciata libera dall'assemblatore può essere saturata da un programma sorgente molto lungo, nel qual caso è necessario trasferire sulle memorie ausiliarie parte del programma assemblato per far posto a quanto deve essere ancora assemblato.

Per questo è evidente che è un fattore determinante per la scelta di un programma assemblatore di questo tipo la previsione di aver programm sorgente di modeste dimensioni, tali da poter coabitare in memoria coi programma assemblatore; e questo vuol dire avere a che fare con un calcolatore di piccole dimensioni, poi-

chè su questi vengono in genere passati programmi brevi, e quasi sempre diversi.

A questo punto, la scelta di un programma assemblatore a una passata potrebbe sembrare la soluzione ideale per il nostro caso: infatti il CANE è un calcolatore di piccole dimensioni, ha una memoria ausiliaria, il lettore di schede, estremamente lenta, funziona in modo "load and go" ed infine ha in genere da eseguire programmi brevi. E, in effetti, sarebbe così se questo programma assemblatore fosse stato scritto sul CANE; il fatto che invece sia stato scritto per il CANE, ma sull'IBM 7090, spiega la apparente contraddizione che c'è tra quanto ora detto e il fatto che questo è un programma assemblatore in due passate. I motivi preferenziali cui abbiamo accennato svaniscono: infatti il calcolatore IBM 7090 ha caratteristiche antitetiche a quelle da noi postulate per la scelta di un programma assemblatore a una passata;sopra tutto ha una memoria veloce così grande, che ci permette di non usare memorie esterne durante l'assemblaggio.

Sempre a proposito di assemblatori in una passata vogliamo aggiungere che, poichè in questo caso, se supponiamo trattarsi di

un programma che carica in memoria e passa immediatamente dopo alla esecuzione, ogni istruzione viene letta e subito messa in memoria in forma binaria, nasce il problema dei simboli non definiti, cioè che appaiono in un indirizzo senza mai essere apparsi prima come etichette. Il problema è stato risolto in vari modi; esaminiamone uno.

Supponiamo che un simbolo non si trovi nella tabella dei simboli (se c'è non ci sono problemi); esso viene messo in una tabella ausiliaria, e gli è associato l'indirizzo dell'istruzione in cui si trova. Successivamente, ogni qual volta una etichetta è messa nella tabella dei simboli, si confronta anche con la tabella ausiliaria: se c'è, si provvede a sostituire il suo valore nella istruzione il cui indirizzo è dato dalla tabella ausiliaria, si cancella dalla tabella e si prosegue la ricerca. Infatti lo stesso simbolo può comparire più volte nella tabella ausiliaria.

Ad esempio, per un programma in cui compaiono le istruzioni

```
..................................
(cella 11)      PISA      SUB     BOH,3
(celal 12)                TRA     SIM
..................................
```

(cella 36)		MEA	SIM
....			
(cella 43)	BOH	SLT	106
(cella 44)	SIM	TRB	*
....			

al momento di mettere SIM nelle tabelle dei simboli, la situazione si presenta così:

Ora un programma di assemblaggio simbolico di questo tipo per il CANE sarebbe stato, come abbiamo accennato, un nonsenso. Infatti non avendo il CANE memoria ausiliaria, sarebbe stato necessario inserire l'assemblatore nella memoria veloce: e delle 512 parole, a disposizione dei programmi sorgente ne sarebbero rimaste ben poche (ammesso di poter scrivere un programma assemblatore con qualche pretesa in meno di 512 parole di 18 bits), mentre abbiamo a disposizione tutta la memoria del 7090.

In pratica, e lo abbiamo detto, sono delle condizioni esterne, di "hardware", che portano alla scelta di programmi assembla-

tori del tipo a una passata: e queste condizioni vengono a mancare poichè il CANE è stato simulato su una macchina come lo IBM 7090.

I programmi assemblatori in due passate sono descritti implicitamente più oltre, poichè questo programma segue gli schemi più classici, almeno nella sua concezione generale: l'assemblaggio si svolge in due passi successivi, durante i quali si leggono, ambo le volte, le istruzioni del programma sorgente. La costruzione delle tabelle dei simboli è sempre fatta durante il primo passo, il calcolo degli indirizzi durante il secondo.

Generalmente, le tabelle sono messe in una memoria esterna e richiamate, quando servono, anche in parte soltanto: anche le istruzioni assemblate vengono messe in una memoria esterna, talvolta perforate su schede in forma binaria, quindi la memoria del calcolatore non è mai satura. Naturalmente, il tempo necessario per l'assemblaggio è maggiore, poichè il trasferimento di dati dalla memoria veloce a quelle esterne e viceversa è piuttosto lento (si consideri il tempo necessario a far scorrere un nastro alla ricerca di un dato).

Questo programma assemblatore riunisce in sé i vantaggi dei due metodi: la seconda lettura non è fatta da una memoria esterna, nastro o disco, ma direttamente dalla memoria veloce, perciò in un tempo estremamente ridotto, pari a pochi microsecondi, e nello stesso tempo una buona parte della memoria non è utilizzata. In pratica, una volta deciso di scrivere questo programma sul 7090, non sussistendo preoccupazioni circa la possibilità di occupare memoria, si è cercato di elaborare un programma di assemblaggio il più veloce possibile.

Lo schema di principio del programma è, come abbiamo detto, quello classico, cioè comune a molti di questi programmi: esso può essere considerato come diviso in vari programmi interni, che talvolta funzionano in maniera diversa nella seconda passata che nella prima, e che vengono chiamati dal programma principale a seconda di quale istruzione debba essere assemblata.

Durante il primo passo si provvede alla costruzione della tabella dei simboli e alla esecuzione, talvolta parziale, delle pseudo operazioni; nel secondo si termina l'esecuzione delle pseudo-operazioni, si calcolano gli indirizzi con l'aiuto della tabella dei sim-

boli, e si assemblano le parole in una porzione di memoria del 7090 che simula quella del CANE. Infine una seconda parte del programma provvede alla stampa della lista di assemblaggio del programma sorgente, al riconoscimento e alla stampa di eventuali errori, all'eventuale caricamento nella memoria del CANE, all'eventuale perforazione su schede del programma oggetto. Gli indirizzi essendo tutti non rilocabili, non sorge il problema di individuare, tra le espressioni simboliche, quelle assolute, le rilocabili e le indeterminate.

Inoltre, poichè i campi in cui son divise le schede che formano il programma sorgente son fissi, non sorge neppure il problema di distinguere tra loro, in qualche modo, le etichette dalle operazioni e dagli indirizzi.

I° PASSO

Le schede di commento vengono solo ricopiate. Per le altre, si individuano anzitutto le pseudo-origine (ORIG), per le quali si attribuisce al contatore di locazione il valore della loro parte indirizzo, che non può quindi contenere simboli non ancora definiti;

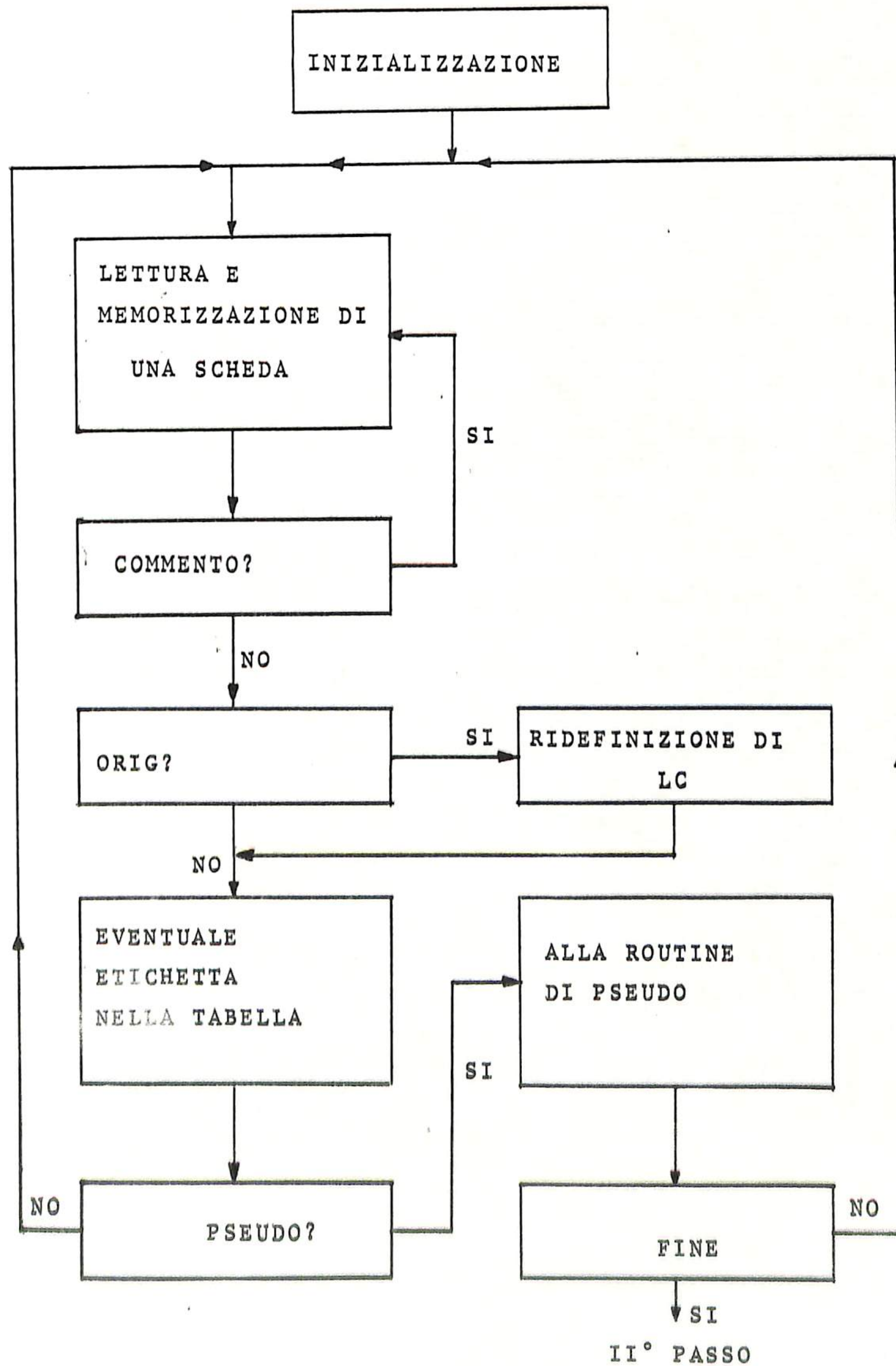
quindi si inseriscono le etichette nella tabella dei simboli; infine si va a vedere se l'istruzione specifica una operazione vera o una pseudo. Se si tratta di una operazione vera, si incrementa il valore del contatore di locazione e si passa a leggere e a memorizzare la scheda successiva, altrimenti si esegue la pseudo. La ORIG e il BCS richiedono solo una ridefinizione del valore del contatore di istruzioni (già effettuata per la ORIG), che viene fatta modulo 511, le altre portano comunque ad un certo incremento del contatore di locazione, come abbiamo visto in dettaglio. La FINE fa cominciare il secondo passo: se, come può accadere, la FINE è stata omessa, continua il primo passo finchè non si trova una scheda controllo, detta scheda asterisco, che indica lo inizio di un nuovo programma. A quel punto il programma assemblatore cede il controllo al supervisore, e gli segnala l'errore oltre a trasmettergli l'immagine della scheda * letta.

II° PASSO

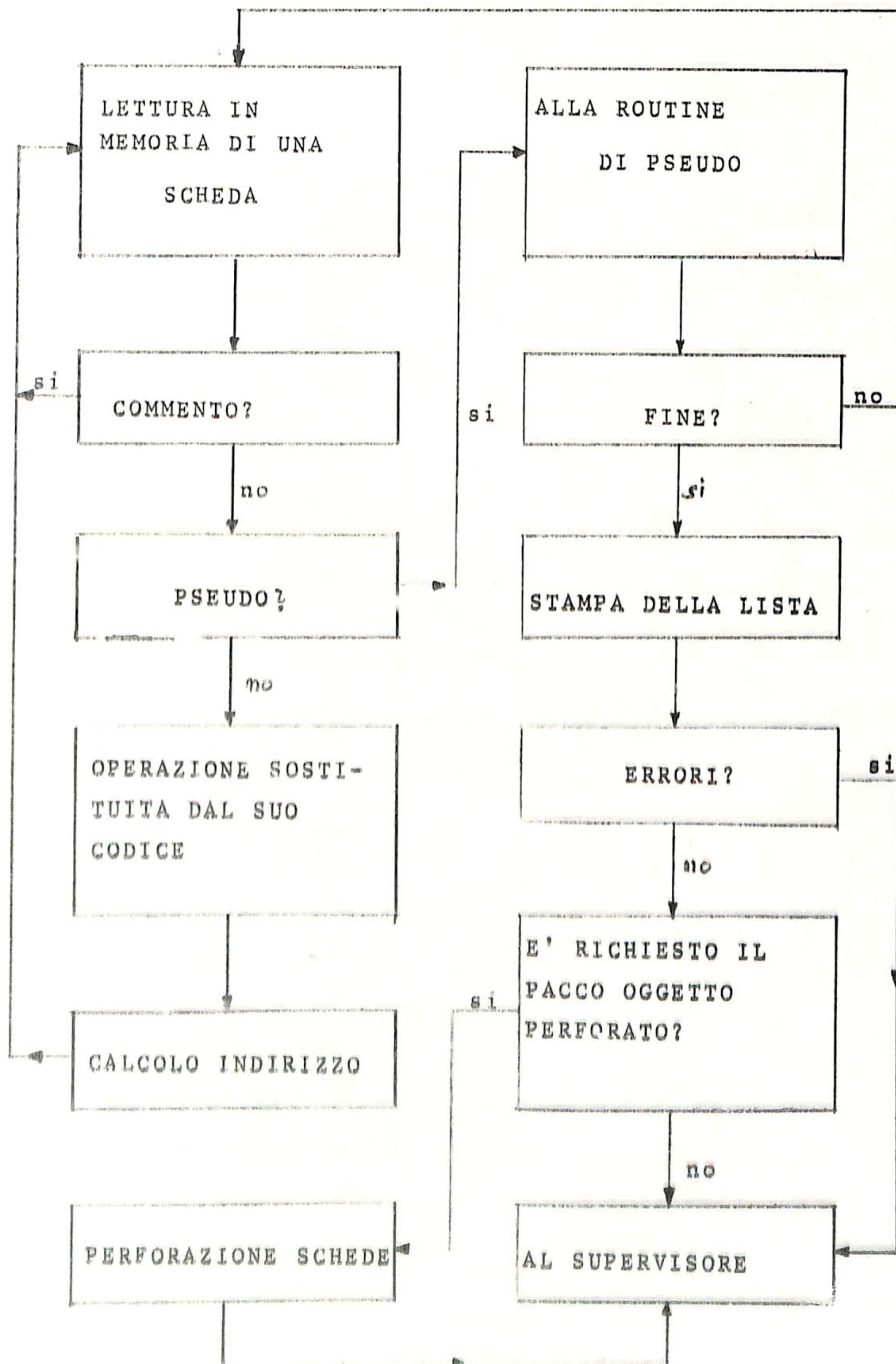
Durante la seconda passata vengono completate le pseudo-operazioni, e si sostituiscono ai nomi simbolici delle operazioni i rispettivi codici numerici. Si calcolano poi gli indirizzi. Nel campo

indirizzo, che si estende per 60 colonne, si possono trovare numeri e simboli separati dai segni di + e -, a formare una espressione aritmetica; e, dopo una virgola, può esserci anche una specifica di registro. Inoltre nell'indirizzo può anche essere specificata una "stringa alfanumerica". Comunque, non essendovi simboli rilocabili, il calcolo degli indirizzi è facilitato: ad ogni simbolo si sostituisce direttamente il valore del contatore di locazione che gli è associato nella tabella. Tutte le espressioni vengono calcolate modulo 511. Quando si incontra di nuovo la FINE, si passa alla seconda parte del programma, cioè alla trasmissione del programma oggetto al simulatore, alle stampe della lista e della tabella dei simboli, e all'altra operazione eventualmente richiesta, cioè la perforazione del programma oggetto su schede.

I° PASSO



II° PASSO



CAPITOLO IV

SOFTWARE E LISTE

Ci sono alcuni programmi, o sottoprogrammi, che vengono usati spesso dal programmatore: ad esempio, quelli che si chiamano "sottoprogrammi di libreria", che vengono, a richiesta, messi a disposizione dell'utente e servono a calcolare, tra l'altro, le radici quadrate, le funzioni trigonometriche, i logaritmi, e così via. Il sottoprogramma che calcola, dato N, il valore di N !, e che abbiamo riportato come esempio di lista di assemblaggio (vedi tabulato), può essere considerato uno di questi. Allo stato attuale, per usufruire di questi programmi è necessario, tutte le volte, far assemblare il pacco di schede su cui sono perforati in simbolico: e questo significa impiegare un certo tempo. Sarebbe più semplice, o meglio più rapido, caricare il programma già in forma assemblata. In effetti tutti i calcolatori sono dotati di un software che permette la perforazione su schede dell'immagine di un programma così come è stato caricato in memoria, ovvero in forma numerica (binaria o ottale), in modo da permettere, successivamente, il caricamento diretto dello stesso. Tutto questo è possibile

farlo anche con il CANE.

Qualora si richieda la perforazione su schede del programma oggetto, una scheda controllo deve essere messa nel programma sorgente, subito dopo la scheda asterisco. Tale scheda porta, in colonne 1/6, la perforazione * PACCO. Si noti che essa deve essere la seconda scheda di un programma, in quanto viene riconosciuta solo in quella posizione: se è letta dopo, viene considerata come errore.

Il programma oggetto è allora perforato su schede, se il programma sorgente non contiene errori; a questo punto, però, il calcolatore non funziona più in modo "load and go", cioè in modo che i programmi privi di errori vengono eseguiti subito dopo esser stati assemblati. Questo perchè si pensa che la richiesta del pacco di schede che rappresentano il programma oggetto viene fatta, in genere, per programmi o sottoprogrammi il cui funzionamento è già stato accertato, e che si presume di dover utilizzare nuovamente nel futuro, ma di cui non interessa l'esecuzione. E' allora inutile eseguire i programmi, e perciò, perforato il pacco di schede, si passa al programma successivo.

Nel caso in cui si voglia anche l'esecuzione del programma, nelle colonne 9/II della scheda * PACCO deve essere perforata la specifica VAI.

Per quanto riguarda il formato del programma perforato, ogni scheda può contenere un massimo di undici parole, scritte in ottale a partire da colonna 7. Le prime tre colonne di ogni scheda contengono l'indirizzo a cui si deve caricare la prima parola: le altre della scheda sono caricate in sequenza. Le colonne 5/6 contengono un numero minore di 12 (al massimo 13 ottale), che indica quante parole sono state perforate in quella scheda: questa è una facilitazione evidente per chi voglia scrivere un caricatore, poichè permette di eliminare il controllo, altrimenti necessario, sui caratteri perforati, per decidere quando smettere di caricare. I caratteri spazio, da colonna 7 a 72, valgono zero. La colonna 4 è lasciata bianca per motivi estetici. Le colonne 72/80 contengono il nome del programma, ed un numero progressivo da I a 99. Se le schede sono in numero maggiore, la numerazione ricomincia da zero.

Per caricare in memoria le schede si può far uso di un opportuno caricatore: quello da noi indicato (vedi tabulato) ha due possi-

bilità di ritorno. Se l'ultima scheda del pacco perforato ha in colonna 5/6 un numero maggiore di 12, si torna alla prima istruzione seguente la chiamata del caricatore (tipo subroutine); se in colonne 5/6 c'è zero o bianco si torna allo indirizzo specificato in colonne I/3 della stessa scheda. Questa scheda deve essere perforata dal programmatore.

Il caricatore stampa il programma, dopo averlo caricato.

Si ricordi che, gli indirizzi non essendo rilocabili, non è possibile spostare la posizione in memoria di un programma oggetto; quindi si deve fare attenzione a non sovrapporre programmi.

In ogni caso, terminato l'assemblaggio, viene stampata la lista di assemblaggio, ovvero la lista delle istruzioni del programma sorgente con a fronte le stesse già assemblate, in forma ottale, e con indicate le posizioni di memoria occupate. Per le pseudo-operazioni la forma della parola assemblata varia caso per caso; per le istruzioni vere e proprie sono indicati, intervallati da spazi bianchi, per una migliore comprensione, la posizione in memoria, il codice operativo, il registro indice, l'indirizzo:

```
017  24  7  001          ADDX     ,7
```

Vengono listate 27 istruzioni per pagina, saltando un rigo tra l'una e l'altra: in cima alla pagina sono riportati il nome del programma (col.6/11 della I^{a} scheda *), e il numero della pagina, preceduto dalla parola PAGINA. Sotto al nome del programma compare la scritta: LISTA DI ASSEMBLAGGIO.

Nel caso in cui il programma oggetto presenti degli errori, essi vengono specificati di seguito alla istruzione cui si riferiscono, in ordine inverso a quello in cui sono stati trovati durante la operazione di assemblaggio; ogni messaggio di errore è preceduto da un numero d'ordine. Per il modo col quale vengono ricercati gli errori, non è possibile segnalare, per una singola istruzione, oltre tre errori: è d'altronde oltremodo difficile fare più di tre errori con una sola scheda. Dopo la lista di assemblaggio, viene anche stampata la tavola dei simboli e delle loro posizioni: cioè una lista dei simboli definiti nel programma sorgente con, a fronte, la posizione della istruzione cui essi erano stati messi come etichetta, cioè il valore che essi assumono durante la esecuzione del programma stesso. Questa lista inizia con la scritta TAVOLA DEI SIMBOLI E DEI LORO INDIRIZZI. Un simbolo definito più volte comparirebbe

nella tavola con un solo valore, il primo attribuitogli, e una so

la volta.

Per ultima viene stampata l'immagine delle schede perforate, secondo il formato descritto in precedenza.

Tutte queste stampe sono riportate qui di seguito.

UNOA

LISTA DI ASSEMBLAGGIO

```
              702    FAT     ORIG 450
      702   400706           SALVA1
      703   061707
      704   067706
      705   407000
      706   137706
      707   131707
      710 24 7 001           ADDX 1,7
      711 13 7 712           MEX  *+1,7
      712 06 1 000           TRX  ,1
      713 13 1 730           MEX  N.,1
      714 24 7 001           ADDX 1,7
      715 13 7 716           MEX  *+1,7
      716 06 7 000           TRX  ,7
      717 13 7 724           MEX  NFAT.,7
      720 13 7 726           MEX  NFAT.+2,7
      721 01 0 733           TRA  I.
      722 16 0 001           ADDA 1
      723 10 0 733           MEA  I.
      724 05 0 000   NFAT.   TRB
      725 26 0 733           MOL  I.
      726 11 0 000           MEB
      727 01 0 733           TRA  I.
      730 34 0 000   N.      GFA
      731 43 0 722           SMIN *-7
      732   400703           TORNAFAT
      733   000000   I.      OTT
              002            FINE NFAT+1
```

Tabulato 2

```
     UNOA                                                            PAGINA  1
LISTA DI ASSEMBLAGGIO
                  /
                  /
                  /               ESEMPIO DI PROGRAMMA SCRITTO NEL LINGUAGGIO DELL'ASSEMBLA
                  /               TORE.  IL SOTTOPROGRAMMA FAT  CALCOLA IL VALORE FATTORIA-
                  /               LE DI N. E LO PONE IN NFAT.
                  /
                  /
   000   000006   N      DEC  6
   001   000001   NFAT   OTT  1
   002   417702          PASSAFAT(N.NFAT)
   003   400006
   004   770000
   005   770001
                  /
                  /
                  /
                  /               IL SOTTOPROGRAMMA FAT PUO' ESSERE PERFORATO SU SCHEDE E
                  /               RICARICATO. IN QUESTO CASO. ESSO PARTE SEMPRE DALLA POSI-
                  /               ZIONE CHE GLI E' STATA ATTRIBUITA DURANTE L'ASSEMBLAGGIO.
                  /               QUINDI PUO' ESSERE RICHIAMATO INDICANDOLO CON LA PSEUDO
                  /               ORIG. IN QUESTO MODO
                  /               SOTP  ORIG 450
                  /                   PASSASOTP(N.NFAT)
```

TABULATO 1 - Esempio di programma assemblato

UNOA PAGINA 3

TAVOLA DEI SIMBOLI E DEI LORO INDIRIZZI

SIMBOLO	INDIRIZZO
N	0
I.	733
N.	730
FAT	702
NFAT	1
NFAT.	724

Tabulato 3

```
  0  6      6      14177024000067700007700 01       0       0       0       0       0UNOA   1
702 13400706 61707 677064070001377061317072470011377 12 610001317302470 01UNOA   2
715 13137716 67000137724137726 10733160001100733 500002607331100 00 10733UNOA   3
730  434000043 0722400703       0       0       0       0       0       0       0       0UNOA   4
```

Tabulato 4 - Immagine del pacco perforato

PAC

PAGINA 1

LISTA DI ASSEMBLAGGIO

```
            500            ORIG 320
500 13 7 544               MEX  VIA+1,7
501 07 1 000               TDX  0,1
502 07 2 020               TDX  16,2
503 65 2 606      CIC      MCA  BUFFER+23,2
504 25 2 001               SODX 1,2
505 40 0 503               SLT  CIC
506 76 0 506               CUS  *
507 73 0 557      LETT     ENA  BUFFER
510 75 0 510               CEN  *
511 74 0 557               USC  BUFFER
512 07 2 001               TDX  1,2
513 07 1 002      LOOP     TDX  2,1
514 01 2 557               TRA  BUFFER,2
515 62 0 003               ABLD 3
516 56 0 003               ALD  3
517 62 0 003               ABLD 3
520 56 0 003               ALD  3.
521 62 0 003               ABLD 3
522 25 1 001               SODX 1,1
523 40 0 554               SLT  RET
524 37 2 000               CFXD 0,2
525 44 0 545               SMAG MEM
526 11 0 627               MEB  TR
527 22 2 627               ADX  TR,2
530 37 2 013               CFXD 11,2
531 22 2 627               ADX  TR,2
532 13 2 550               MEX  CF,2
```

Tabulato 5 - Esempio di caricatore

LISTA DI ASSEMBLAGGIO

```
533 62 0 011          ABLD 9
534 11 0 627          MEB  TR
535 06 7 627          TRX  TR,7
536 44 0 543          SMAG VIA
537 37 2 000          CFXD 0,2
540 13 7 544          MEX  VIA+1,7
541 07 2 003          TDX  3,2
542 44 0 513          SMAG LOOP
543 76 0 543   VIA    CUS  *
544 40 0 000          SLT  0
545 11 7 000   MEM    MEB  ,7
546 24 7 001          ADDX 1,7
547 77 0 000          NOP
550 37 2 000   CF     CFXD ,2
551 24 2 003          ADDX 3,2
552 43 0 513          SMIN LOOP
553 40 0 506          SLT  LETT-1
554 25 2 001   RET    SODX 1,2
555 77 0 000          NOP
556 40 0 514          SLT  LOOP+1
557       050  BUFFER BCS  40
627       001  TR     BCS  1
          500         FINE 320
```

Tabulato 6

CAPITOLO V°

SEGNALAZIONE DEGLI ERRORI.

Scrivendo un programma per un calcolatore, in uno qualunque dei tanti linguaggi oggi esistenti, si possono commettere una notevole varietà di errori: nella impostazione logica del programma, o nella sua trascrizione nel linguaggio scelto.

Uno dei compiti, ed uno dei più importanti, di un programma di assemblaggio simbolico, è la ricerca e la segnalazione degli errori del secondo tipo , cioè degli errori formali. Il SIMBOLCANE riconosce e segnala ventuno errori (lo Assembler dell'IBM 7090, o MAP,per esempio, ne segnala 128)

Il riconoscimento degli errori è naturalmente necessario se si desidera una compilazione corretta: infatti, il mancato riconoscimento porterebbe ad un programma certamente anomalo,e, in taluni casi addirittura alla fermata dell'assemblatore per situazioni non previste.

Ogni errore viene dapprima identificato con una lettera, che etichetta l'istruzione errata: solo alla fine dell'assemblaggio

essi vengono decodificati, e quindi sono stampati dei messaggi di errore immediatamente comprensibili. Certi errori, inoltre, provocano l'immediato ritorno al supervisore: il programma in assemblaggio viene scaricato. Questo accade se viene letta una scheda *, il che significa per come è organizzato il pacco di schede per un programma in SIMBOLCANE, (v. Appendice III), che il programma sorgente in esame non aveva la scheda FINE: in questo caso la immagine della scheda viene trasmessa al supervisore, e il programma in assemblaggio scaricato. Questo avviene anche se si trova la scheda *FINE, che segnala la fine del pacco di programmi che debbono essere eseguiti dal CANE, e provoca la scrittura, da parte del programma supervisore, della lista dei programmi eseguiti e la fine della esecuzione stessa. Si ricordi che il supervisore, prima di chiamare l'Assemblatore, legge la scheda * (o *FINE, nel qual caso si ferma). L'assemblatore legge finchè non trova una pseudo. FINE: quindi, solo se questa manca, può leggere una scheda * o * FINE (v. appendice III).

Anche nel caso in cui un programma sia composto di oltre 512 schede esso viene scaricato: questo perchè, pur essendo la memoria del CANE circolare, si è dovuto serbare un'area di memoria ben definita nel 7090 per costruirvi l'immagine della lista di assemblaggio, ed essa consente un massimo di 512 schede; sarebbe possibile rendere circolare anche questa porzione di memoria,cioè inserire la 513ª scheda al posto della prima, e così via, ma il risultato sarebbe una lista di assemblaggio "monca" cioè priva di alcune parti, e non facilmente comprensibile.

Durante il primo passo dell'assemblaggio si costruisce la tabella dei simboli e si eseguono, talvolta parzialmente, alcune pseudo-operazioni: dunque si cercano errori sulle etichette e su queste pseudo operazioni. Nelle etichette possono esservi caratteri non permessi, e il trovarne uno fa sì che la lettera che indica quell'errore venga attaccata alla istruzione, mentre la etichetta stessa viene scartata, così che non è possibile rilevare due errori formali nella stessa etichetta.Ad esempio, per

AB+C-1

una volta letto il +, non può essere letto anche il -(la lettura va da sinistra a destra) . Anche durante l'esecuzione di una pseudo-operazione, il primo eventuale errore provoca la sospensione di tale esecuzione, impedendo quindi il rilevamento di altri eventuali errori. Ad esempio;

OTT 9,I,6, 871

la presenza del primo 9 termina l'assemblaggio della pseudo e impedisce di rilevare l'errore che c'è in 871.

Durante il secondo passo si riconoscono i codici operativi, ed ecco una seconda possibilità di errore, e si calcolano gli indirizzi, terza possibilità di errore: infatti il primo errore trovato in un indirizzo fa terminare il calcolo dell'indirizzo stesso, che viene preso uguale a zero, così come viene messo uno zero nel campo del registro indice.

Concludendo, si possono segnalare, per ogni istruzione, un massimo di tre errori.

E' da notare che questo programma assemblatore scrive i messaggi d'errore subito dopo l'istruzione cui si riferiscono, e non, come succede in genere, alla fine della lista di assem-

blaggio.

Questo è dovuto al fatto che il dizionario degli errori, cioè una tabella in cui sono riportati le lettere che identificano i possibili errori, e i messaggi di errore, è tenuto in memoria durante l'assemblaggio, mentre in genere tale dizionario è chiamato in memoria dai programmi assemblatori e compilatori (ad esempio, FORTRAN IV e MAP) solo dopo che la lista di assemblaggio è stata trasferita su nastro, per ragioni di spazio.

E' da notare infine che non sono stati introdotti livelli di gravità: tutti gli errori impediscono l'esecuzione del programma (manca cioè il livello 1 del MAP): questo è dovuto a ragioni didattiche, cioè al fatto che non si è voluto che fossero segnalati anche eventuali errori triviali, cioè che potrebbero anche non aver conseguenze deleterie sull'assemblaggio o l'esecuzione, dal punto di vista formale. Ad esempio,

NOP 125,3

la NOP non richiede nè indirizzo nè registro, ciononostante la presenza dei due non è dannosa, poichè essi non sono considerati in fase di esecuzione, dato che NOP fa passare all'istruzione suc

cessiva e basta.

Per gli stessi motivi, non sono segnalati errori triviali che portano ad una errata esecuzione del programma, come, ad esempio, la istruzione

TDX 6

che richiederebbe la specifica obbligatoria del registro indice. (E' da tener presente che gli utenti del CANE, e anche di questo assemblatore, sono soprattutto gli studenti dei primi anni del corso di laurea in Scienza della Informazione, e che non si è voluto fornire loro un assemblatore in grado di segnalare tutti gli errori, per non spingerli a scriver programmi senza fare attenzione).

In pratica, oltre 100 degli errori segnalati, ad esempio, dal MAP, si riferiscono a pseudo-operazioni che non sono state definite da questo programma, o ad anomale condizioni di funzionamento della macchina o del programma stesso (in questo caso l'assemblaggio termina). Considerando solo le pseudo operazioni comuni ad ambo i programmi, e tenendo conto che il CANE non ha nè indiriz-

zamento indiretto, nè decremento, nè prefisso,il MAP segnala solo 25 errori, cioè più o meno gli stessi di questo assemblatore.

Riportiamo di seguito il dizionario degli errori: i messaggi si spiegano da soli. Il numero ordinale sotto "PASSO" dice se l'errore è rilevato durante il I°, il II°, o tutti e due i passi dell'assemblatore.

Ad esempio, errori nel campo indirizzo, codifica P, possono essere rilevati solo durante il II° passo, cioè quando si calcolano gli indirizzi: mentre uno qualunque degli errori formali sui simboli può essere trovato durante il primo passo, in una etichetta, o durante il secondo, in un indirizzo.

Alcuni messaggi segnalano nello stesso modo errori diversi: ad esempio la codifica R di "carattere non permesso" può riferirsi sia alla presenza di una parentesi in una etichetta o in un indirizzo (esclusa la PASSA), sia a caratteri non ammessi nelle literals =O e =D

Quindi:

```
        A)B     TRA     A
                MEA     (0)
                ADA     =D1A2
                SOA     =018
```

provocano il ripetersi di uno stesso messaggio d'errore:

```
        A)B     TRA     A
** 1   CARATTERE NON PERMESSO
                MEA     (0)
** 2   CARATTERE NON PERMESSO
                ADA     =D1A2
** 3   CARATTERE NON PERMESSO
                SOA     =018
** 4   CARATTERE NON PERMESSO
```

In questo caso sotto la scritta "TIPO" compaiono più specificazioni (per R, sono SIMBOLO e LITERALS (=O e =D), per indicare i vari tipi di errore segnalati nello stesso modo.

Si noti infine che i messaggi d'errore sono numerati in maniera progressiva a partire da 1.

Dizionario degli errori

PASSO	CODIFICA	TIPO	MESSAGGIO
I°-II°	A	SIMBOLO	CARATTERE * NON PERMESSO NEI SIMBOLI
I°	B	ALFA	MANCA LA VIRGOLA IN COLONNA 15
I°	C	SIMBOLO	SIMBOLO GIA' DEFINITO
II°	D	"	" NON "
I°	E	ALFA	NELLE COLONNE 13/14 DEVE COMPARIRE UN NUMERO DA 01 a 19
II°	G	OP	CODICE OPERATIVO IGNOTO
I°-II°	H	SIMBOLO	I SIMBOLI DEVONO AVERE IL PRIMO CARATTERE ALFABETICO
I°	I	OTT	CARATTERE NUMERICO O NUMERO NON OTTALE O TROPPO GRANDE
I°	L	ORIG BCS FINE	REGISTRO NON AMMESSO
II°	N	LITERALS	SPECIFICA ERRATA O OLTRE TRE CARATTERI ALFANUMERICI
I°-II°	O	SIMBOLO	CARATTERE-NON PERMESSO NEI SIMBOLI
II°	P	INDIRIZZO	ERRORE NEL CAMPO INDIRIZZO O REGISTRO

PASSO	CODIFICA	TIPO	MESSAGGIO
I°-II°	Q	SIMBOLO	CARATTERE +NON PERMESSO NEI SIMBOLI
I°-II°	R	SIMBOLO LITERALS(ØoD)	CARATTERE NON PERMESSO
I°-II°	S	SIMBOLO	" " " NEI SIMBOLI
I°-II°	T	BCS LITERALS	FUORI MEMORIA
I°-II°	U		CARATTERE STRANO NELLA SCHEDA
I°	V	PASSA	ERRORE NELLA CHIAMATA
I°	X	SALVA	ERRORE NEL SALVATAGGIO

```
UNOA                                                            PAGINA 1

LISTA DI ASSEMBLAGGIO

        000    475146    A        ALFA 15,PROVA DI ERRORE
        001    652160
        002    243160
        003    255151
        004    465125
        005    606060
        006    606060
        007    606060
        010    606060
        011    606060
        012    606060
        013    606060
        014    606060
        015    606060
        016    606060
                         /          DIAMO ALCUNI ESEMPI DI MESSAGGI DI ERRORE
        017 77 0 000     A        NOP
 1  SIMBOLO GIA' DEFINITO
        020 10 0 000              MEA   8
 2  SIMBOLO NON DEFINITO
        021    000000    A        OTT   142,87
** 3  CARATTERE NON NUMERICO, O NUMERO NON OTTALE O TROPPO GRANDE
** 4  SIMBOLO GIA' DEFINITO
        022 65 0 027              MCA   =AXJV
        023 77 0 000              NOP   *2
** 5  ERRORE NEL CAMPO INDIRIZZO
        024    000000    A        ALFA 02 ERRORE
** 6  MANCA LA VIRGOLA IN COLONNA 15
```

LISTA DI ASSEMBLAGGIO

```
**  7  SIMBOLO GIA' DEFINITO

                     025 00 0 000    2A      MVA  (A+B)

**  8  CARATTERE NON PERMESSO

**  9  CODICE OPERATIVO IGNOTO

** 10  I SIMBOLI DEVONO AVERE IL PRIMO CARATTERE ALFABETICO

                     026 65 0 000    &       MCA

** 11  CARATTERE STRANO NELLA SCHEDA

                     027   674165   *LORIG

                               000           FINE A,1

** 12  REGISTRO NON AMMESSO
```

APPENDICE I

Elenco delle istruzioni del CANE

00	ALT		
01	TRA		in rA
02	TRAN		in rA negativo
03	TDA		diretto in rA
04	TDAN	trasferimento	diretto in rA negativo
05	TRB		in rB
06	TRX		in rXj
07	TDX		diretto in rXj
10	MEA		di rA
11	MEB		di rB
12	MEZ	memorizzazione	di zero
13	MEX		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	CFA	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		a sottoprogramma
42	SSP		se supero
43	SMIN		se minore
44	SMAG		se maggiore
45	SUG		se uguale
46	SAN	salto	se rA negativo
47	SAP		se rA positivo
50	SAZ		se rA zero
51	SBN		se rB negativo
52	SBP		se rB positivo
53	SBZ		se rB zero
54	AVD		aritmetico di rA verso destra
55	AVS		aritmetico di rA verso sinistra
56	ALD		logico di rA verso destra
57	ALS		logico di rA verso sinistra
60	BVD	spostamento	aritmetico di rB verso destra
61	BVS		aritmetico di rB verso sinistra
62	ABLD		logico di rA ed rB verso destra
63	ABLS		logico di rA ed rB verso sinistra

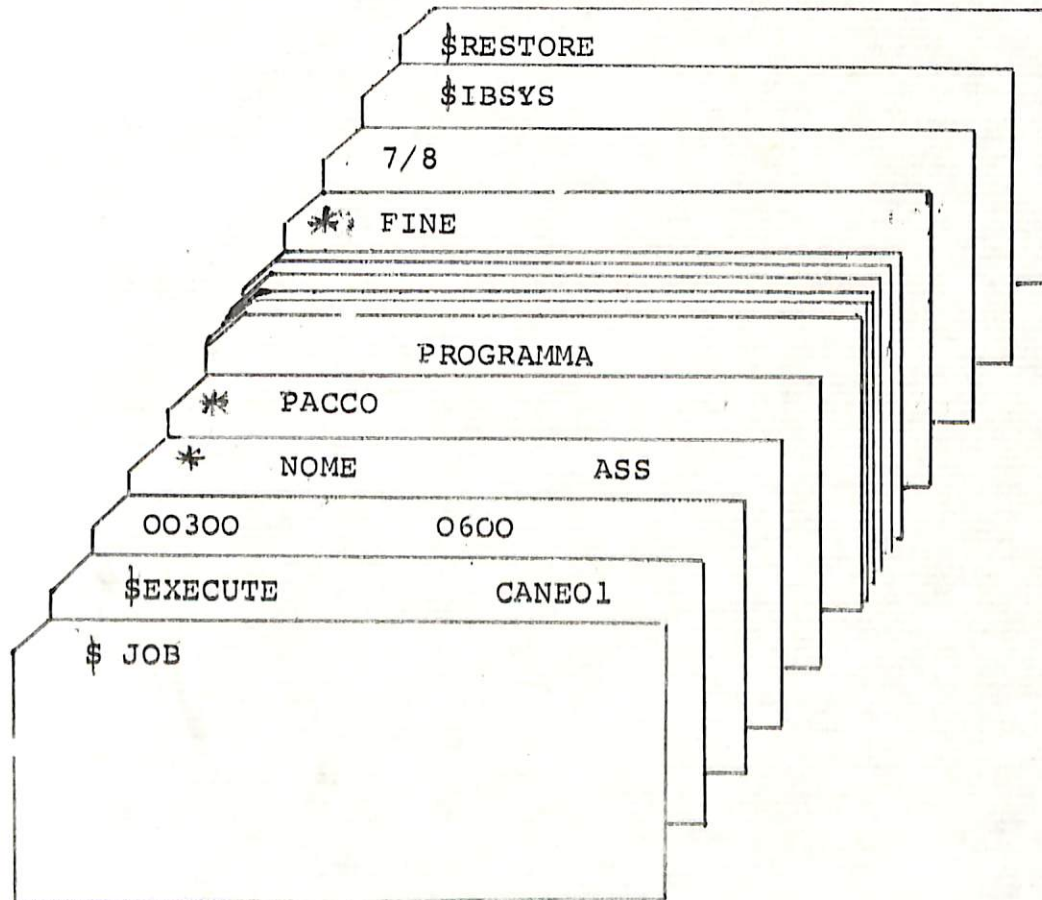
64	TCA	trasferimento carattere
65	MCA	memorizzazione carattere
66	CAR	conversione in caratteri
67	NUM	conversione in numero binario
70	ESG	esegui
71	GULP	
72	ENB	entrata binaria
73	ENA	entrata alfanumerica
74	USC	uscita
75	CEN	controllo entrata
76	CUS	controllo uscita
77	NOP	nessuna operazione

APPENDICE II

Codice caratteri BCD

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	,	001100
V	110101	=	001011
W	110110		
X	110111		
Y	111000		
Z	111001		

APPENDICE III



Struttura del pacco di schede per un programma in SIMBOLCANE.

L'Assemblatore legge dalla scheda dopo la scheda * fino alla pseudo FINE, quindi il controllo torna al supervisore.

<u>BIBLIOGRAFIA</u>

1) .A Grasselli - G. Pacini. "CANE,SIMULCANE e SYSTEMCANE"

Nota Tecnica C70/1.I.E.I.

2) G.Pacini. "ARCHITETTURA E SIMULAZIONE DI UN CALCOLATORE DIDATTICO".

Tesi di laurea in Fisica all'Università di Pisa

3) P.M.Sherman. "PROGRAMMING AND CODING DIGITAL COMPUTER".

J. Wiley and Sons, Inc.

4) I.B.M. "IMB 7090/94 IBSYS Operating SYSTEM,VERSION 13.MACRO ASSEMBLY PROGRAM(MAP)LANGUAGE

5) "IBM 7090 PRINCIPLES OF OPERATIONS".

IBM Systems Reference Library

6) "SISTEMA OPERATIVO IBM 7090/94,VERSIONE 13.LINGUAGGIO SIMBOLICO FORTRAN IV".

IMB Documentazione sui Sistemi.