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

Franco Preparata

INTRODUZIONE AGLI ASSEMBLATORI

MONOGRAFIE DIDATTICHE
D - 73 - 4
MAGGIO 1973

INTRODUZIONE AGLI ASSEMBLATORI

1. Generalità

L'esperienza di realizzare algoritmi mediante programmi scritti in linguaggio di macchina (o linguaggio base) ha certamente il risultato positivo di conferire al programmatore una consapevolezza della interfaccia più dettagliata attraverso cui comunicare col calcolatore. Al tempo stesso però essa consente l'individuazione di alcuni sostanziali difetti di questa tecnica di codifica dei programmi, che si possono così schematizzare:

1.1. L'impiego di codici numerici di istruzione è particolarmente innaturale, in quanto obbliga il programmatore ad una gravosa memorizzazione di una corrispondenza del tutto arbitraria tra codici ed operazioni. A ciò si è ovviato molto presto nella storia dei calcolatori sostituendo ai codici numerici i cosiddetti codici simbolici o mnemonici (per esempio, per il calcolatore didattico CANE è molto più naturale ricordare il significato di MEA0004 anzichè del suo equivalente 10 0004 in linguaggio base).

1.2. E' esperienza comune la difficoltà nell'assegnare indirizzi assoluti (cioè indirizzi di precise locazioni di memoria) ai dati nel corso della codifica del programma, quando non se ne conosce ancora la lunghezza in termini di numero di istruzioni. Infatti è desiderabile, per sfruttare la modalità consueta di trasferimento del controllo (da un indirizzo all'indirizzo consecutivo, ~~eccetto il caso~~ di istruzioni di salto) poter costruire un programma come la concatenazione di una sequenza di istruzioni e di una sequenza di dati. Pertanto è pratica consueta designare gli indirizzi degli operandi con simboli, che saranno poi convertiti in indirizzi di locazioni di macchina una volta che la stesura delle istruzioni sia completa.

1.3. L'assegnazione di indirizzi assoluti alle istruzioni e agli operandi conferisce una notevole rigidità al programma. Infatti, essa rende gravosa sia l'inserzione che la soppressione di istruzioni, in

Per sopprimere istruzioni basta mettere NOP, senza rilocare.

quanto entrambe queste operazioni comportano la rilocazione di porzioni di programma (con possibile conseguente modifica dei campi di indirizzo di alcune istruzioni). Inoltre, la "codifica in assoluto" di programmi ne ostacola l'inserimento come sottoprogrammi "aperti" all'interno di programmi di utente.

Ad esempio, un semplicissimo programma che esegue l'addizione di due numeri A e B e ne memorizza la somma è codificato usando i codici mnemonici del C.A.N.E. in figura 1(a). In figura 1(b) si presenta una codifica in cui si designano simbolicamente non solo i co-

```
002   TRA 006
003   ADA 007
004   MEA 010
005   SLT 011
006     (A)
007     (B)
010   (A+B)
011
```

```
      TRA OP1
      TDA OP2
      MEA SOMMA
      SLT
OP1
OP2
SOMMA
          (b)
```

Figura 1

dici di istruzione, ma anche gli indirizzi degli operandi. Ovviamente la stesura data in figura 1(b) non ci vincola sulla definizione degli indirizzi OP1, OP2 e SOMMA, come anche lascia libera la scelta dell'indirizzo della prima istruzione TRA OP1. Queste scelte cioè potranno essere compiute una volta che l'intero programma (i cui segmenti possono anche essere costruiti da più programmatori operanti indipendentemente) sarà stato scritto in questa forma, molto più naturale per un programmatore dell'astruso linguaggio di macchina.

La conversione da un programma scritto in questa forma nel suo corrispettivo in linguaggio macchina appare come un'operazione di natura essenzialmente meccanica. Infatti ci si rese conto ben presto (durante gli anni 50) della fattibilità di questa conversione automatica posto che si imponesse una certa formalizzazione delle regole per la stesura simbolica dei programmi. La conversione automatica viene compiuta da un programma, noto come assemblatore, che viene a far parte del software di servizio del sistema. Il compito di questo programma di servizio è quindi la traduzione di un programma sorgente (in linguag-

gio simbolico) in un programma oggetto (in linguaggio di macchina).

Pertanto, in queste note ci proponiamo di esaminare dapprima il problema della formulazione di un linguaggio preciso per mezzo del quale si possa esprimere un programma sorgente da fornire all'assemblatore perchè lo traduca nel corrispondente programma oggetto. In seguito dall'esame delle funzioni dell'assemblatore trarremo motivazione adeguata per il progetto sufficientemente dettagliato di un assemblatore tipico.

2. Il linguaggio di assemblatore

Ogni problema di comunicazione implica la preventiva scelta convenzionale del linguaggio da usare. Senza volere stabilire analogie con la comunicazione tra esseri umani, il cosiddetto "linguaggio di macchina" è il mezzo per comunicare con il calcolatore al livello più dettagliato; similmente, per comunicare al livello dell'assemblatore - cioè del calcolatore (hardware) arricchito dell'assemblatore (software) - dovremo sviluppare un opportuno linguaggio. Questo linguaggio, chiamato di assemblatore, viene comunemente classificato come un linguaggio di livello intermedio, cioè a metà strada tra i linguaggi a basso livello (di macchina) e quelli ad alto livello (per compilatore). E' ora nostra intenzione fornire una descrizione dei linguaggi di questo tipo.

E' chiaro che il linguaggio di assemblatore non può possedere la varietà e la flessibilità di un linguaggio naturale (come, ad esempio, l'italiano); al contrario, è altamente formalizzato, anzi, si tratta di un esempio di una classe di strutture chiamate tecnicamente linguaggi formali. In analogia con i linguaggi naturali, i linguaggi formali constano di un insieme di espressioni (chiamate convenzionalmente dichiarazioni) che sono costruite secondo un insieme di precise regole, che costituisce la grammatica. E' allora opportuno compiere una breve digressione sulla struttura dei linguaggi formali, traendo la neces-

saria motivazione dai linguaggi naturali.

Un linguaggio (formale o naturale) è l'insieme delle "espressioni" linguistiche (cioè, proposizioni) corrette: questa correttezza viene verificata a più livelli, cioè lessicale (o delle parole), sintattico (o delle regole che governano la costruzione delle proposizioni) e semantico (o delle proposizioni che hanno significato). Per esempio, con riferimento all'italiano, notiamo che delle tre espressioni "strada poichè il", "il bambino mangia la mela", "la mela mangia il bambino", la prima consta di parole corrette ma non è una proposizione, le ultime due sono proposizioni grammaticalmente corrette, di cui però solo la prima è semanticamente corretta. Nel seguito ci occuperemo solo della correttezza grammaticale, trascurando se non marginalmente, l'aspetto semantico. Vediamo allora come possiamo analizzare la proposizione "il bambino mangia la mela".
Essa consta di una frase nominale, "il bambino", seguita da una frase verbale, "mangia la mela". A sua volta "il bambino" consta di un articolo, "il", seguito da un nome, "bambino". Infine "mangia la mela" consta di un verbo, "mangia", seguito da una frase nominale, "la mela", che può essere a sua volta analizzata come sopra. Questa analisi è efficacemente riassunta dall'illustrazione in figura 2 che viene interpretata dal basso all'alto nel processo di analisi (che abbiamo testè compiuto) e dall'alto al basso nel processo inverso di sintesi o generazione.

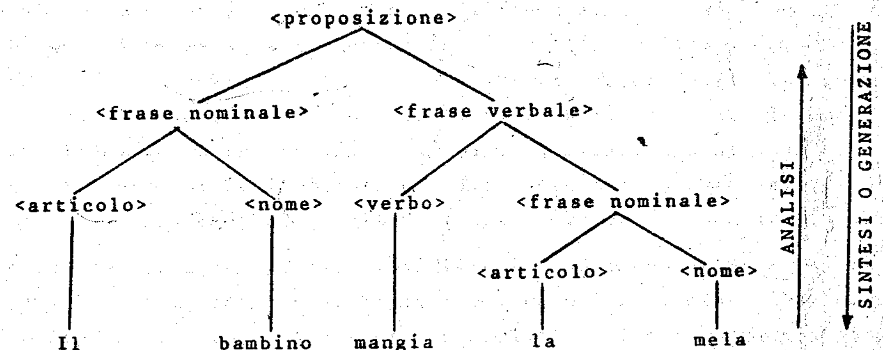

Figura 2

In figura 2 notiamo due tipi di termini, i termini grammaticali (racchiusi tra parentesi angolate<>) e le parole del lessico. L'insieme di queste ultime viene denotato tecnicamente come l'alfabeto terminale (in quanto i suoi elementi appaiono come terminali nel processo di generazione), mentre i termini grammaticali costituiscono ovviamente l'alfabeto non-terminale. Le regole che consentono di sostituire ad un termine nonterminale una concatenazione di altri termini non-terminali o terminali (eventualmente un sol termine) si chiamano regole di riscrittura o produzioni. Una maniera standard di indicare una produzione è illustrata di seguito:

<frase verbale>:=<verbo><frase nominale> (1)

E' anche concepibile che <frase verbale> possa essere riscritta semplicemente come <verbo> (si pensi ad un verbo intransitivo) per cui avremo anche la produzione

<frase verbale>:=<verbo> (2)

E' conveniente allora, per economia di notazione, combinare più produzioni che abbiano lo stesso termine sinistro in un'unica produzione il cui termine destro contiene le diverse forme possibili per il termine destro scritte di seguito e separate da una barra verticale "|". Ad esempio, (1) e (2) sono combinate come segue:

<frase verbale>:=<verbo><frase nominale>|<verbo>

Questa notazione è conosciuta come la forma normale di Backus. Ciò conclude questa breve esposizione dei rudimenti dei linguaggi formali.

Siamo ora in grado di applicare le nozioni introdotte allo studio dei linguaggi di assemblatore. Le considerazioni che seguono si riferiscono ai linguaggi di assemblatore in generale, anche se per concretezza ci riferiremo di frequente al linguaggio di assemblatore sviluppato per il calcolatore C.A.N.E.

L'alfabeto terminale o lessico consta comunemente di simboli e di numeri. I simboli sono sequenze alfanumeriche di 5 o 6 simboli al massimo, che comunemente hanno una lettera alfabetica come primo carattere; esempio: START, FINE, OPER1, LET2, ecc.(Inoltre alcuni caratteri (segni) usati comunemente per interpunzione o per scopi particolari, non sono ammessi nei simboli). I numeri sono sequenze di cifre decimali.

L'alfabeto non terminale preferiamo invece costruirlo passo passo secondo le necessità che emergeranno. Se ci riferiamo all'esempio di Figura 1, è evidente la necessità di due tipi fondamentali di dichiarazioni: dichiarazioni di istruzioni (es.: TDA OP2 in figura 1(b) e dichiarazioni di dati (es.: OP1 .... in figura 1(b)). Quindi avremo la produzione:

<proposizione>:=<dichiar.istruz.>|<dichiar.dati> (3)

Esamineremo ora separatamente la struttura della dichiarazione di istruzione e della dichiarazione di dati.

2.1. La dichiarazione di istruzione

Il formato specifico di questa dichiarazione dipenderà sia dal calcolatore per cui si intende sviluppare l'assemblatore, sia dalle scelte particolari compiute in sede di sviluppo dell'assemblatore medesimo. Con questa premessa cautelativa, si può tuttavia dire che il seguente formato (adottato per il C.A.N.E.) è sufficientemente generale

<dichiar.istruz.>:=<etichetta><codice><indirizzo>,<indice>/<commento> (4)

Si noti che i diversi termini grammaticali sono da intendersi separati da almeno uno spazio, qualora specificamente non si faccia uso di speciali caratteri terminali di interpunzione, quali "," e "/", che hanno un preciso valore sintattico; ad esempio, "/" precede il termine <commento>. I termini grammaticali introdotti in (4) sono ora spiegati formalmente (è conveniente introdurre il simbolo Λ, che denota il cosiddetto termine vuoto, cioè l'assenza di ogni termine):

<etichetta>:=<simbolo>|Λ (5)
<codice>:=<simbolo> (6)
<indirizzo>:=<costante>|<espressione> (7)

<indice>:=<numero>|Λ (8)

<commento>:= qualunque sequenza di caratteri

Nella produzione (6) è ovvio che i simboli ammessi sono solo i codici mnemonici che denotano operazioni del calcolatore: si tratta di una restrizione di tipo chiaramente semantico. Nella produzione (8), similmente, poichè vi è in genere un numero piuttosto piccolo di registri indice, <numero> consta di una o due cifre (nel C.A.N.E., una cifra da 1 a 7).

La produzione (7) merita un commento più dettagliato. Innanzitutto, l'operando di un'istruzione può essere specificato o come un indirizzo (specifica indiretta) ovvero come un dato (specifica diretta). I loro corrispettivi simbolici sono un simbolo o una costante, rispettivamente. Inoltre nel primo caso può accadere, ad esempio, di voler specificare una locazione che è ad una distanza <u>fissa</u> (espressa da un numero) da un indirizzo simbolico (espresso da un simbolo): da ciò la seguente definizione di <espressione>:

<espressione>:=Λ|*|<operando>|<espress.>+<operando>|<esp.>-<op.>

<operando>:=<simbolo>|<numero>

Ad esempio, BEGN, 103, START+1, LET1-K sono esempi di <espressione>. Frequentemente si introducono regole che riducono il numero delle espressioni legittime rispetto a quelle generate dalla (9). Il termine <costante> è così definito nel C.A.N.E.:

<costante>:= =D<numero>|=O<numero>|=A<simbolo> (10)

I simboli =D, =O, =A servono a denotare chiaramente il tipo di costante (D per decimale , O per ottale, A per alfabetico). Infine il simbolo * consente una lieve semplificazione nella scrittura dei programmi. Esso denota "l'indirizzo della istruzione della quale appare nel campo di indirizzo", ed ha un impiego tipico nelle istruzioni di controllo di entrata ed uscita. Ad esempio sia data la dichiarazione

CEN * ; (11)

=D literal

TRB =D 10 equivale a TDB 10

decimale

se l'istruzione corrispondente viene assegnata alla locazione, per esempio, 103, allora CEN * viene tradotta in 103 75 0 103(75 è il codice di macchina corrispondente a CEN). Si noti che lo stesso risultato può ottenersi con una dichiarazione d'istruzione in cui a <etichetta> e a <indirizzo> sia assegnato lo stesso simbolo diverso da Λ.

Naturalmente, ulteriori restrizioni possono imporsi circa le scelte di <numero> e <simbolo> nella (10).

Esaminiamo ora alcuni esempi interessanti di dichiarazioni di istruzioni nel calcolatore didattico:

```
<etichetta><codice><indirizzo>,<indice>/<commento>
  START      TRA      OPER1     ,  3
             MEA      SOMMA+2              / MEMORIZZA RISULTATO
             CEN        *
             ALT
  INDEX      TRX      =D12      ,  4
             TRA      =0721435
```

Si noti che a ciascuna dichiarazione di istruzione corrisponde, in generale, una istruzione di macchina. Eccezioni sono possibili tuttavia, se per conferire maggiore flessibilità all'assemblatore, si consente nelle dichiarazioni di istruzione l'uso di codici operativi che risultano in più di una istruzione di macchina.

2.2. Dichiarazione di dati

Un formato tipico di dichiarazione di dati è il seguente

<dichiar.dati>:=<etichetta><dato>/<commento> (12)

I termini <etichetta> e <commento> sono già stati considerati nel precedente paragrafo. Ci occupiamo quindi del termine <dato> . In generale i dati sono costanti, sia alfabetiche che numeriche. E' pertanto possibile immaginare una varietà di formati per il termine <dato>, e non possono darsi regole generali. Nell'assemblatore sviluppato per il calcolatore C.A.N.E. si è convenuto di stabilire una certa similarità tra il formato delle dichiarazioni di istruzione e delle dichia-

razioni di dati. Ciò è esplicitamente illustrato dalle seguenti produzioni:

<dato>:= DEC<dato decimale>|OTT<dato ottale>|ALFA<dato alfabetico> (13)

<dato decimale>:=<numero>,<numero>,...,<numero> (14)

<dato ottale >:=<numero>,<numero>,...,<numero> (15)

<dato alfabetico>:=<numero>,<simbolo>...<simbolo> (16)

Nelle (14) e (15) si possono esprimere numeri fino eventualmente a riempire l'intera scheda contenente la dichiarazione; nella (16) <numero> è di due cifre compreso tra 1 e 19 e indica quanti simboli, ciascuno di tre caratteri, lo seguono.

Esaminiamo ora alcuni esempi tipici. La dichiarazione

BLOC DEC 152, 28, -31

carica in tre celle consecutive, la prima delle quali ha l'etichetta BLOC, i numeri binari che rappresentano rispettivamente 152,28,-31 (che possono essere designati come BLOC,BLOC+1,BLOC+2). La dichiarazione

NUM OTT 123471

carica il numero 123471 (ottale) nella locazione etichettata con NUM. Infine la dichiarazione

DATA ALFA 05, 27 MARZO 1973

carica le celle simbolicamente designate da DATA, DATA+1, DATA+2, DATA+3, DATA+4 come segue:

```
DATA   27b
       MAR
       ZOb
       197
       3bb
```

dove "b" denota uno spazio.

Concludiano questo paragrafo con un semplice esempio. Si voglia scrivere in linguaggio di assemblatore C.A.N.E. un programma il cui scopo è di determinare il massimo di un insieme di numeri diversi da 0,

ciascuno dei quali è perforato in binario nelle prime 18 colonne di una scheda; l'insieme di queste schede viene completato con una scheda non perforata. Il diagramma di flusso è illustrato in figura 3, nella quale si sono messe in evidenza le corrispondenze tra le

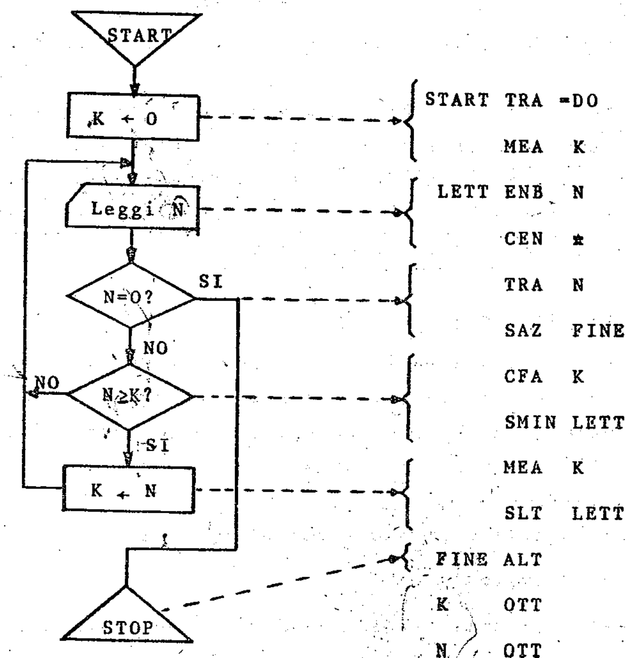

Figura 3

operazioni del diagramma e le istruzioni che le realizzano. Si noti l'efficacia dell'uso delle etichette LETT e FINE; le prime 11 dichiarazioni sono di istruzione, le due ultime sono di dati. Il programma, come scritto, è tuttavia incompleto. Infatti all'assemblatore non viene fornita nessuna indicazione sulla collocazione del programma in memoria, nè sul numero di istruzioni che lo costituiscono (nel

senso che non viene specificato se vi siano altre dichiarazioni oltre la N OTT). Occorrerà quindi prevedere la possibilità di impartire informazioni di tipo organizzativo all'assemblatore. Ne scaturisce la necessità di un nuovo tipo di dichiarazione, chiamata dichiarazione di comando o pseudo-dichiarazione (per brevità, pseudo), per cui la produzione (3) viene modificata come segue:

<proposizione>:=<dichiar.istruz.>|<dichiar.dati>|<dichiar.comando> (3')

Si noti che mentre ciascuna dichiarazione sia di istruzioni sia di dati ha come risultato la definizione del contenuto di "righe" di programma di macchina, ciò non accade per la dichiarazione di comando: ciò giustifica il termine "pseudo". Il prossimo paragrafo è dedicato alla discussione di questo tipo di dichiarazione.

2.3. La dichiarazione di comando

Dalle precedenti considerazioni si rileva la necessità inderogabile di almeno due dichiarazioni di comando: una per la collocazione assoluta del programma in memoria (denotata dal simbolo ORIG, mnemonico per "origine"), l'altra per indicare la fine del programma (denotata dal simbolo FINE).

Consideriamo la prima. Un suo formato tipico è

Λ ORIG<numero>[1]

dove <numero> designa, in decimale, l'indirizzo della istruzione dichiarata nella riga successiva. Ad esempio nel C.A.N.E., le dichiarazioni

```
     ORIG  64
LETT ENB   N
```

(1) - Frequente è anche il formato Λ ORIG *+<numero>, che consente di saltare <numero> locazioni consecutive a partire dall'indirizzo corrente.

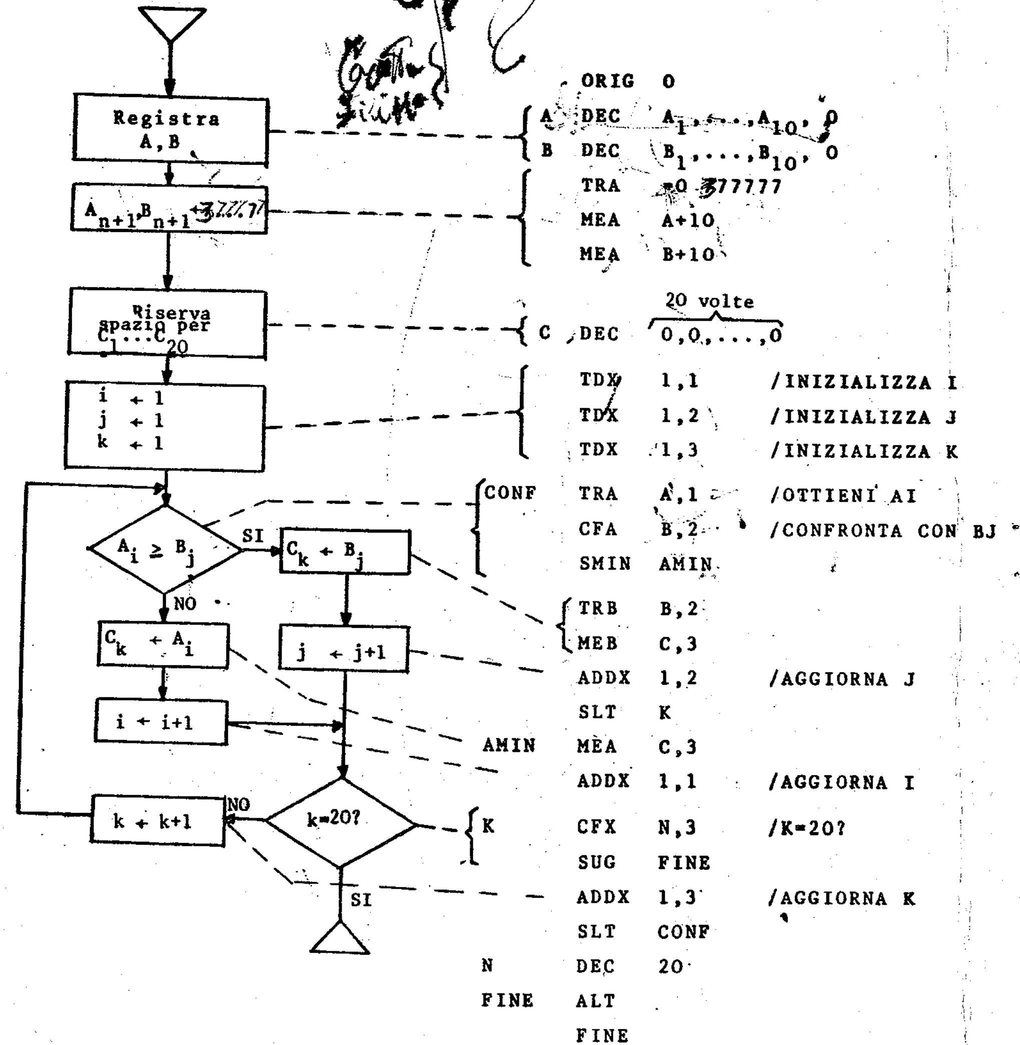

Figura 4

indicano che la istruzione corrispondente a ENB N dovrà essere caricata nella locazione 100 (in ottale). Le dichiarazioni che seguono una pseudo ORIG dànno luogo ad un segmento di programma caricato in locazioni contigue. E' allora possibile segmentare un programma e caricarne i segmenti in differenti blocchi di memoria (cioè sequenze di celle) semplicemente premettendo a ciascun segmento un'opportuna dichiarazione ORIG.

La dichiarazione di fine programma ha il formato tipico

Λ FINE Λ

che non richiede ulteriori commenti. Nel caso che il processo di traduzione sia seguito immediatamente dall'esecuzione del programma tradotto (si parla in questo caso di assemblatori load-and-go, "carica e va"), allora la pseudo FINE specificherà anche l'istruzione iniziale del programma ed avrà il formato: Λ FINE<espressione>.

Concludiamo questo paragrafo con un semplice programma che fonde due successioni ordinate di 10 numeri ciascuna in una successione ordinata di 20 numeri. Si suppone che le tre successioni $(A_1,\dots,A_{10})$ $(B_1,\dots,B_{10})$ e $(C_1,\dots,C_{20})$ siano ordinate in ordine crescente. Si ricordi che il massimo numero positivo è nel C.A.N.E. (in ottale) 377777. Il diagramma di flusso del programma è dato in figura 4 (a), ed il corrispondente programma per assemblatore è dato in figura 4 (b).

3. Struttura di un programma assemblatore

Come si è più volte menzionato, la funzione dell'assemblatore è la traduzione di un programma scritto in un linguaggio a livello intermedio (il cosiddetto programma sorgente) in un programma espresso in linguaggio di macchina (il cosiddetto programma oggetto). Questa traduzione avviene elaborando in sequenza le dichiarazioni del programma sorgente.

In questo processo vi sono due categorie di simboli che devono essere tradotti: i codici mnemonici e le etichette. Mentre la corrispondenza tra i primi e i codici di macchina è nota a priori, lo stes-

so non può certo dirsi per le etichette ed i corrispondenti indirizzi, in quanto le etichette vengono inventate dal programmatore. Non vi sarebbe alcun problema se nella scansione riga per riga del programma sorgente ciascuno dei simboli usati come etichetta comparisse prima come etichetta e poi come indirizzo (o, come suol dirsi, riferimento di memoria). Infatti, in questi casi, si è in grado di assegnare una locazione al simbolo etichetta e di usare poi questo assegnamento quando si incontri lo stesso simbolo usato, ad esempio, come indirizzo di un'istruzione. Ad esempio, nel caso del programma sorgente

N OTT 27

TRA N

supponendo di assegnare ad N la locazione 102 (cioè il contenuto di 102 è 000027), la dichiarazione TRA N risulta nell'istruzione 01 0 102. Tuttavia, per offrire maggiore flessibilità al programmatore, si consente anche la prassi alternativa, per cui un simbolo viene introdotto inizialmente come indirizzo, cioè come riferimento ad un'etichetta che verrà definita oltre nel programma. Poichè il processo di traduzione preserva l'ordine di "righe" corrispondenti nel programma sorgente e nel programma oggetto, l'assegnazione di un indirizzo al simbolo in questione non è nota sino a che non lo si sarà incontrato come etichetta. Ad esempio, nel caso del programma sorgente

TRA N

N OTT 27

la dichiarazione TRA N non potrà essere tradotta, in quanto la sua parte indirizzo risulterà definita solo dopo l'elaborazione della dichiarazione N OTT 27 (il che avverrà in seguito).

E' allora evidente che, perchè si possa effettuare la conversione occorre dapprima stabilire la corrispondenza tra etichette e indirizzi di macchina. Quindi, il processo di traduzione consiste di due fasi: l'assegnazione di indirizzi alle etichette, e la conversione vera e propria nel programma di macchina. Poichè entrambe le fasi richiedono la scansione di tutte le dichiarazioni del programma sorgente, ciascuna di esse viene denotata come una passata e l'assemblatore consta di due passate. Ovviamente, se si introduce il vincolo che la dichiarazione di un'etichetta debba precedere il suo uso, è possibile avere un assemblatore ad una passata.

3.1. La fase di assegnazione (Prima passata)

Come si è accennato, lo scopo della prima passata è l'assegnazione di indirizzi alle etichette.* Come vedremo è opportuno effettuare contemporaneamente altre utili operazioni, che però possiamo per ora ignorare se facciamo l'ipotesi che il programma sorgente sia privo di errori.

Con questa premessa, il compito del programma che esegue la prima passata è solamente l'assegnazione di indirizzi ad etichette. Poichè, come si è detto, la traduzione da programma sorgente a programma oggetto preserva l'ordinamento tra elementi corrispondenti, alle "righe" del programma oggetto verranno assegnati indirizzi consecutivi. E' allora ovvio che l'assemblatore dovrà essere dotato di un contatore, chiamato contatore di locazione (CL), per l'assegnazione di indirizzo. Uno schema semplificato del diagramma che compie la prima passata è allora illustrato in figura 5, e viene ora rapidamente descritto. Le dichiarazioni vengono acquisite una dopo l'altra. Per ogni dichiarazione si esamina se essa sia una pseudo, cioè se il secondo termine sia ORIG o FINE: nel primo caso occorre reimpostare il contatore di locazione, nel secondo caso si ha la terminazione dell'algoritmo. La circostanza normale è però quella in cui si ha una dichiarazione di istruzione o di dati. Per ciascuna di esse occorre verificare la presenza di un simbolo nel campo etichetta e, in caso

* Cioè la costruzione di una tabella che a ogni simbolo usato come etichetta associa un indirizzo numerico. Tale tabella sarà detta ...

** ... zione, per vedere se è una dichiarazione di dato; perchè una dichiar. di dato può originare il contenuto di più celle di memoria.

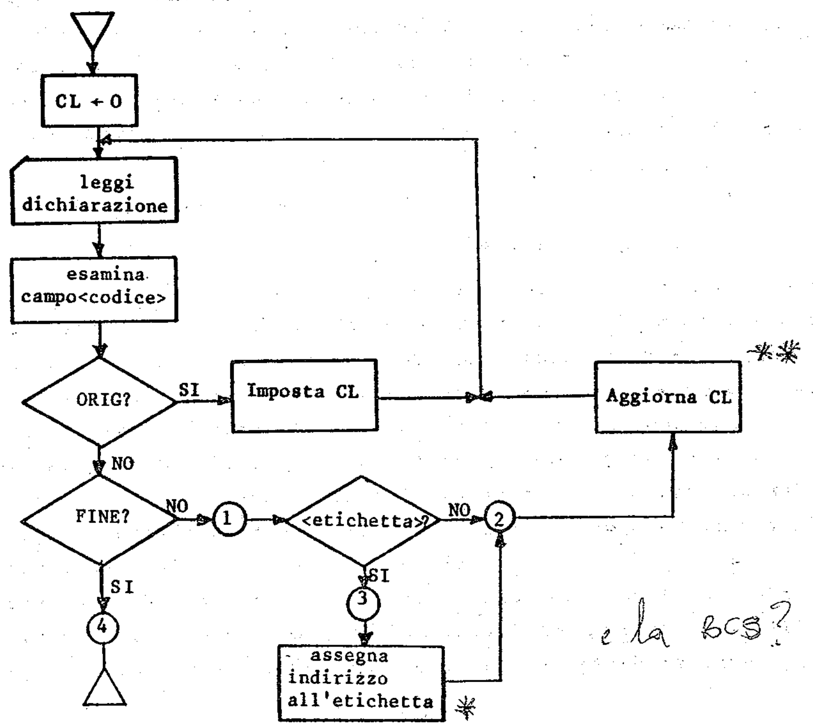

Figura 5
Schema semplificato del programma per la prima passata

affermativo, assegnare a questo simbolo un indirizzo assoluto. Naturalmente, dopo la elaborazione di una generica dichiarazione occorre incrementare CL del numero di celle di memoria richieste per la sua conversione in linguaggio di macchina (normalmente una).

L'ipotesi che il programma sorgente sia privo di errori è, tuttavia, poco realistica. E' quindi conveniente compiere una verifica di

* inserisci etichetta nella tabella etichette/indirizzi associando all'etichetta l'attuale valore di CL

* dichiarato = introdotto nel campo etichetta
(Un simbolo non può apparire più volte come etichetta
2. Verifica di ciascun simbolo usato nel campo indirizzo

correttezza strettamente lessicale e sintattica che consenta di eliminare almeno i più ovvii errori del programmatore (o di chi perfora le schede). E' innanzitutto indispensabile che i simboli usati - etichette o codici - siano simboli validi secondo le regole del lessico. E' poi necessario che ciascun simbolo introdotto * dal programmatore sia dichiarato esattamente una volta. Ne segue che il programma dovrà compiere i seguenti controlli:

1. Verifica di validità dei simboli (correttezza lessicale)
2. Verifica che ciascun simbolo usato (come indirizzo), e non ancora dichiarato, sia dichiarato in seguito
3. Verifica che nessuna etichetta sia dichiarata più di una volta.

La violazione di ciascuna di queste verifiche risulta in un errore. E' allora conveniente che il programma stampi una lista degli errori rilevati prima di terminare la prima passata. Esaminiamo ora separatamente ciascuna delle verifiche elencate sopra.

E' opportuno inserire la verifica di validità dei simboli al punto indicato con (1) nel diagramma di flusso di figura 5. Chiaramente il sottoprogramma preposto a questo fine consisterà di due componenti essenziali, - verifica e registrazione d'errore - , come è illustrato in figura 6.

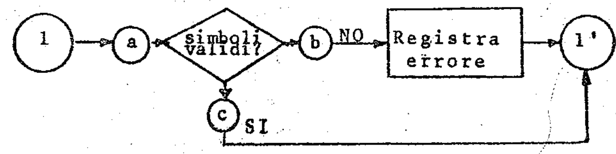

Figura 6
Sottoprogramma per la verifica della validità dei simboli

Tuttavia ciascuno dei componenti è esso stesso un programma di complessità non banale, come ora ci renderemo conto. Infatti, supponiamo che la dichiarazione in via di elaborazione contenga un simbolo

ci si deve accertare che il primo carattere non sia una cifra

$X_1X_2X_3X_4X_5X_6$. Allora dovremo innanzitutto accertarci che il carattere X_1 non sia un numero e che nessuno dei 6 caratteri sia un segno speciale *,+,/,-,(,), ecc. Quindi il controllo di validità (compreso, in figura 6, tra i connettori indicati con a,b,c) verrà espanso come è illustrato dal diagramma in figura 7, che non richiede

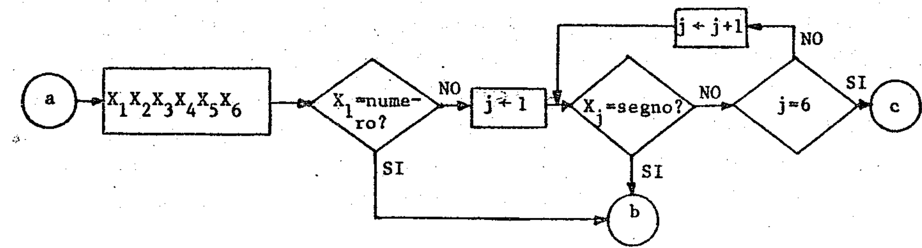

Figura 7

de ulteriori spiegazioni. Si noti che ciascuno dei due saggi "X_1=numero?" e "X_j=segno?" in effetti denota una "ricerca tabellare" di una certa complessità. Ad esempio, il saggio "X_j=segno?" richiede il confronto del carattere X_j con ciascuno dei segni speciali, contenuti in una tabella, registrata in celle consecutive di memoria.

Similmente il blocco operativo REGISTRA ERRORE in figura 6 indica la compilazione di una tabella di errori in una zona di memoria, chiamata convenzionalmente "buffer degli errori". Ciascun errore dà luogo alla composizione di una riga di stampa, contenente le informazioni sull'errore (cioè tipo di errore, dichiarazione in cui appare, ecc.) da comunicare all'utente alla conclusione del primo passo.

La verifica che ciascun simbolo sia dichiarato esattamente una volta può essere compiuta nel seguente modo. Per ogni simbolo usato

come indirizzo si controlla se sia già stato dichiarato; in caso contrario, lo si registra in un apposito buffer (una zona di memoria riservata a questo scopo) e vi si appone una marca. Per ogni simbolo dichiarato come etichetta si verifica se non sia già stato dichiarato (nel qual caso si ha un errore). In caso contrario, si verifica se sia già stato usato come indirizzo, nel qual caso esso appare "marcato" nel buffer delle etichette; si rimuove allora la marca. Nessuna azione di controllo è dovuta quando il simbolo dichiarato non sia stato usato precedentemente. Il sottoprogramma schematico del controllo della marcatura, descritto dal diagramma in figura 8(a), è convenientemente inserito al punto (2) nel diagramma di figura 5. Il sottoprogramma della doppia assegnazione è descritto dal diagramma in figura 8(b) ed è convenientemente inserito al punto (3) del

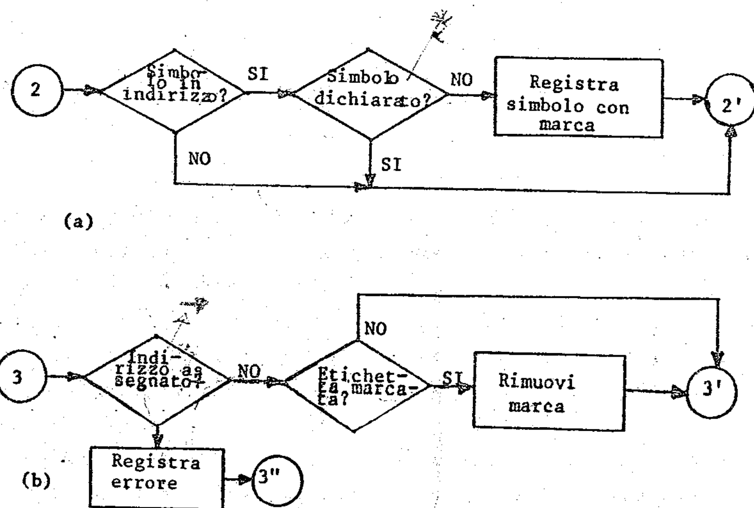

Figura 8

Sottoprogrammi di controllo di marcatura (a) e di doppia assegnazione (b)

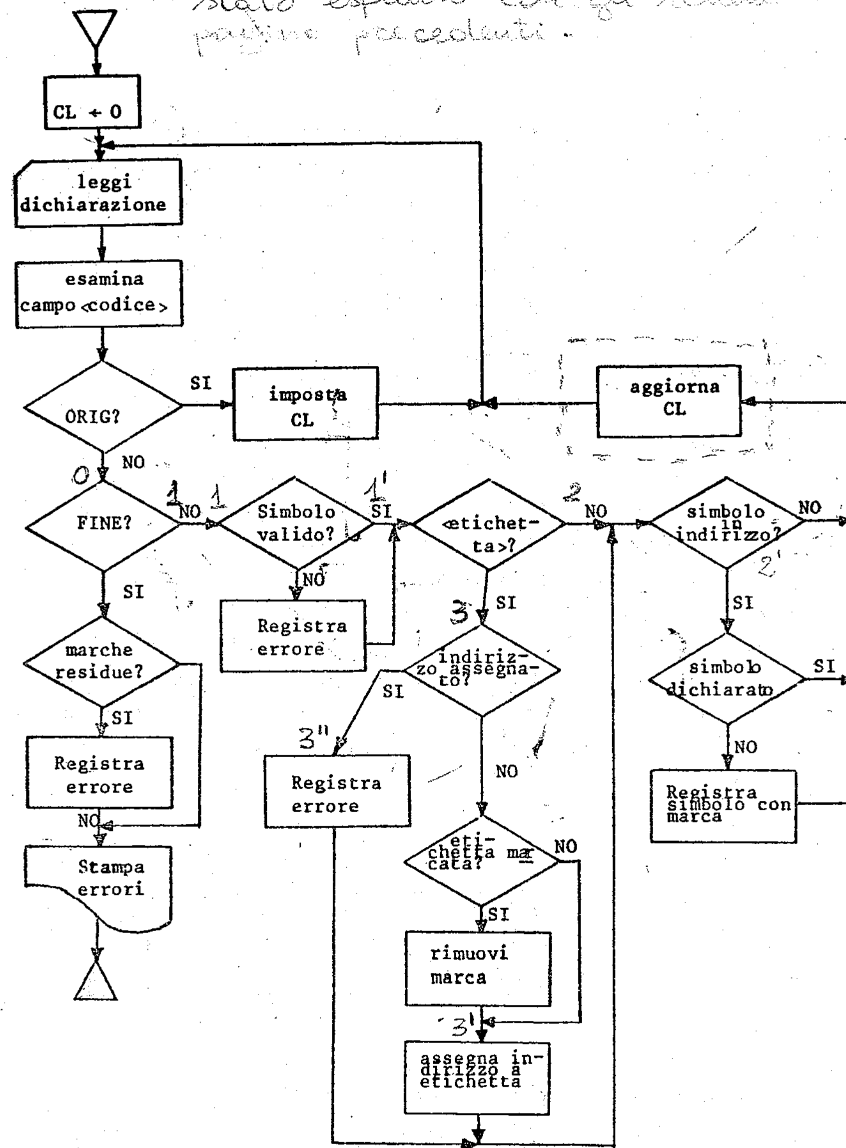

Figura 9

Schema completo del programma per la prima passata

diagramma generale della prima passata (figura 5). Non è necessario alcun commento speciale sul sottoprogramma di stampa degli errori, inserito al punto (4) del diagramma in figura 5; la stampa vera e propria è preceduta dal controllo di eventuali marche non rimosse nella tabella delle etichette (che stanno ad indicare omissione di dichiarazione). Per convenienza del lettore, il diagramma completo del primo passo dell'assemblatore è riportato in figura 9.

3.1. La fase di traduzione (seconda passata)

Al termine della prima passata, dopo l'eventuale correzione degli errori rilevabili, il programma assemblatore dispone di due tabelle di corrispondenza: quella dei (codici mnemonici)-(codici operativi) e quella dei (simboli)-(indirizzi); oltre a ciò, esso dispone naturalmente di una tabella dei codici delle dichiarazioni dei dati e delle pseudo. Il processo di traduzione è concettualmente molto semplice ed è illustrato dal diagramma di flusso in figura 10. La parte organizzativa di questa fase coincide con quella della prima fase: la differenza essenziale risiede nel fatto che laddove la prima passata stabiliva la corrispondenza etichetta-indirizzo, la seconda esegue la traduzione in linguaggio di macchina. E' solo quest'ultima operazione che perciò merita di essere commentata in dettaglio (in figura 10 i blocchi operativi che la costituiscono sono indicati a tratto forte).

Innanzitutto occorre esaminare se la dichiarazione corrente sia di istruzione o di dati: ciò vien fatto esaminando il codice. Nel primo caso (dichiarazione di istruzione) viene tradotto dapprima il codice, facendo riferimento alla tabella dei codici, e viene quindi inserito il campo indice. Infine, facendo uso della tabella simboli-indirizzi, viene valutata l'espressione nel campo indirizzo della dichiarazione: il numero che ne risulta costituisce il valore da introdurre nel campo indirizzo dell'istruzione testé tradotta. Il processo di traduzione è completo e l'istruzione di macchina viene inserita in un'apposita zona di memoria (il buffer del programma

* tabella etichette/indirizzi

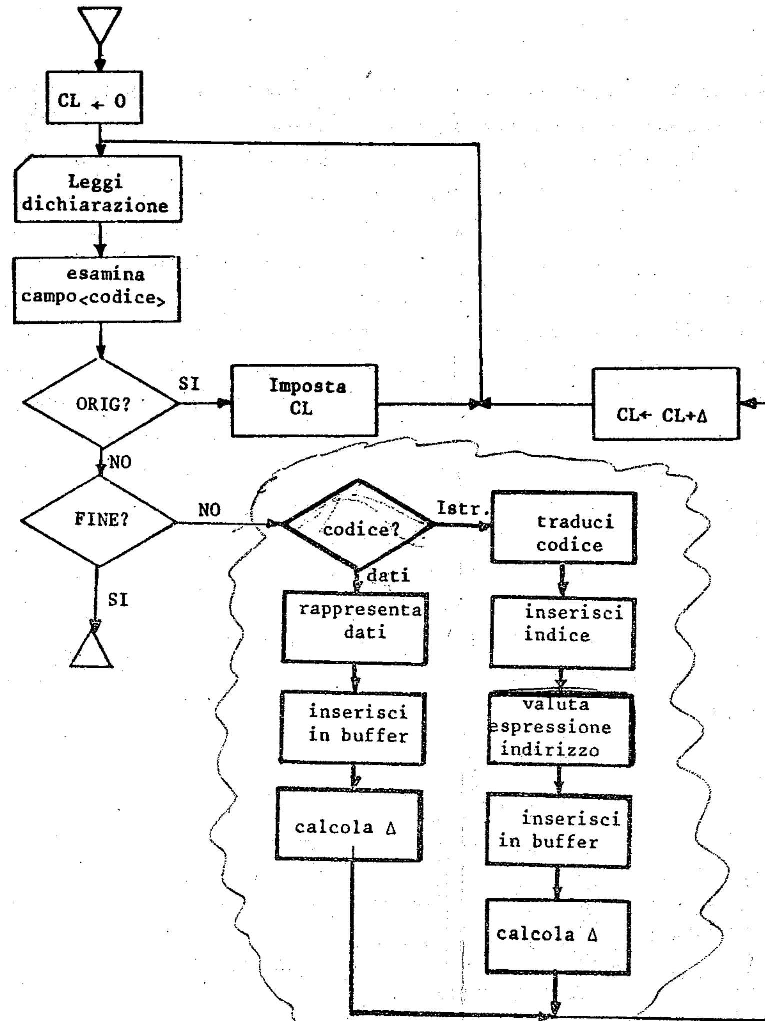

Figura 10

Schema del programma per la passata di traduzione

oggetto). Viene poi calcolato il valore Δ (normalmente $\Delta=1$) di cui occorrerà incrementare il contatore di locazione CL.

Nel caso della dichiarazione di dati occorre rappresentare i dati nel formato del calcolatore (tipica è una conversione da decimale a binario) e registrare i dati in locazioni consecutive del buffer del programma oggetto. Al termine di ciò occorre calcolare Δ, che in questo caso è un intero qualunque.

Ciò conclude la descrizione di un semplicissino programma assemblatore. Riassumendo, si nota che un assemblatore tipico consta di svariate componenti: il programma vero e proprio, che possiamo chiamare "controllo", ed alcune zone di memoria usate per contenere tabelle (le tabelle dei codici operativi, delle etichette, dei codici pseudo) ovvero come zone di lavoro (i cosiddetti buffer degli errori, del programma oggetto oltrechè il buffer di entrata in cui viene temporaneamente immagazzinata la dichiarazione corrente). Queste componenti sono illustrate negli schemi a blocchi nelle figure 11(a) e (b), che illustrano il flusso delle informazioni durante la prima e la seconda passata, rispettivamente, e - si confida - non richiedono ulteriori commenti.

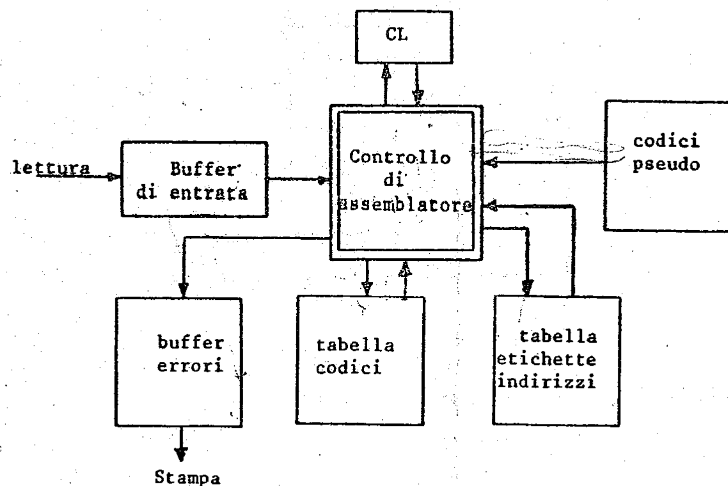

Figura 11 (a)

Flusso delle informazioni durante la prima passata del processo di assemblaggio.

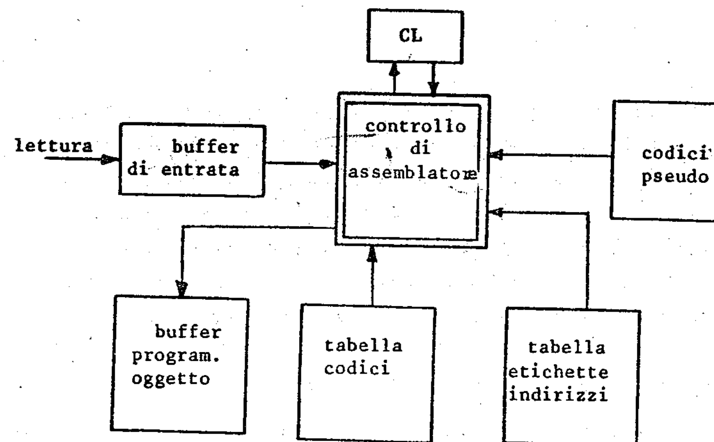

Figura 11 (b)

Flusso delle informazioni durante la seconda passata del processo di assemblaggio.

4. Ringraziamento

La qualità di queste note è stata considerevolmente migliorata dall'attento esame e dai suggerimenti dei Dr. Barsi, Galbiati, Lodi, Montangero, Pacini e Spadafora ai quali va il mio ringraziamento per la collaborazione.