18ª edizione: 1990

Copyright © by Franco Angeli Editore, Milano, Italy
È vietata la riproduzione, anche parziale o ad uso interno o didattico, con qualsiasi mezzo effettuata, non autorizzata.

I lettori che desiderano essere regolarmente informati sulle novità pubblicate dalla nostra Casa Editrice possono scrivere, mandando il loro indirizzo, alla "Franco Angeli, Viale Monza 106, 20127 Milano", ordinando poi i volumi direttamente alla loro Libreria.

INDICE

0. Riconoscimento Pag. 11

1. Introduzione " 13
1.1. Cenni sui linguaggi di programmazione " 13
1.1.1. Algol " 14
1.1.2. APL " 14
1.1.3. Basic " 14
1.1.4. Cobol " 15
1.1.5. Fortran " 15
1.1.6. Pascal " 15
1.1.7. PL/1 " 16
1.2. Scelta di un linguaggio di programmazione " 16
1.3. Lo sviluppo del Cobol " 17
1.3.1. Le origini " 17
1.3.2. Lo standard " 18
1.4. Presentazione del volume " 20

2. Considerazioni generali " 21
2.1. Preliminari " 21
2.1.1. Set grafico " 21
2.1.2. Parole " 21
2.1.3. Costanti " 26
2.1.4. Punteggiatura " 28
2.1.5. Notazioni " 28
2.1.6. Foglio di programmazione " 31
2.2. Struttura di un programma " 33
2.2.1. IDENTIFICATION DIVISION " 34
2.2.2. ENVIRONMENT DIVISION " 34
2.2.3. DATA DIVISION " 34
2.2.4. PROCEDURE DIVISION " 34
2.2.5. Struttura delle divisioni " 35

3. Relazioni fra programma ed elaboratore " 36
3.1. IDENTIFICATION DIVISION " 36
3.2. ENVIRONMENT DIVISION " 37
3.2.1. CONFIGURATION SECTION " 39
SOURCE-COMPUTER – OBJECT-COMPUTER – SPECIAL-NAMES

3.2.2. INPUT-OUTPUT SECTION pag. 42
FILE-CONTROL – I-O-CONTROL
3.2.3. Tavola di riepilogo " 43
3.3. Un esempio " 43

4. Definizione dei dati " 47
4.1. FILE SECTION " 48
4.1.1. Descrizione dei flussi " 48
4.1.2. Descrizione dei record " 50
4.1.3. Descrizione dei campi " 55
Campi di gruppo – Campi elementari – Categorie – FILLER
4.1.4. L'esempio PROG1 " 59
4.2. WORKING-STORAGE SECTION " 62
4.2.1. Campi elementari " 69
4.2.2. Campi di gruppo " 69
4.2.3. Clausola VALUE " 70
4.2.4. L'esempio PROG1 " 71

5. Trattamento dei dati " 73
5.1. Struttura della PROCEDURE DIVISION " 73
5.2. Istruzioni di entrata e uscita " 75
5.2.1. I verbi ACCEPT e DISPLAY " 75
5.2.2. Input ed output di dati organizzati in flussi " 77
Il verbo OPEN – Il verbo CLOSE – Il verbo READ – Il verbo WRITE
5.3. Istruzione di trasferimento " 82
5.4. Istruzioni aritmetiche " 85
5.4.1. Addizione " 85
5.4.2. Sottrazione " 87
5.4.3. Moltiplicazione " 88
5.4.4. Divisione " 90
5.5. Realizzazione di strutture alternative e di cicli " 91
5.5.1. I test di relazione " 93
5.5.2. I test di classe " 94
5.5.3. L'istruzione di salto incondizionato " 94
5.6. Richiamo di un sottoprogramma interno " 96
5.7. L'istruzione di arresto " 98
5.8. L'esempio PROG1 " 99
5.9. Un esempio " 99

6. Precisazioni sulla descrizione dei dati " 117
6.1. PICTURE numeriche " 117
6.1.1. Il simbolo "S" " 118
6.1.2. Il simbolo "V" " 118
6.1.3. Esempi ed osservazioni " 119
6.2. PICTURE di edizione " 120
6.2.1. I simboli "." e "," " 121
6.2.2. Il simbolo "Z" " 122
6.2.3. Il simbolo "0" " 123

6.2.4. Il simbolo "$" pag. 123
6.2.5. I simboli "+" e "–" " 125
6.2.6. I simboli "CR" e "DB" " 126
6.2.7. Il simbolo "*" " 127
6.2.8. Il simbolo "B" " 128
6.2.9. Il simbolo "/" " 128
6.3. Riepilogo sulle categorie " 129
6.3.1. Categoria numerica " 129
6.3.2. Categoria alfabetica " 129
6.3.3. Categoria alfanumerica " 129
6.3.4. Categoria numerica editata " 130
6.3.5. Categoria alfanumerica editata " 130
6.4. Clausole speciali di descrizione dati " 130
6.4.1. Strutture di un'area di lavoro " 131
6.4.2. Posizionamento di un dato in un campo di categoria alfabetica o alfanumerica " 133
6.4.3. Limitazioni nella memorizzazione di un dato in un campo " 135
6.4.4. Il segno nei campi di categoria numerica " 136
6.4.5. Edizione dei campi con valore zero " 137
6.4.6. Assegnazione di nomi " 137
6.4.7. Assegnazione di nomi-condizione " 139
6.4.8. Il formato dei dati in memoria " 141
6.5. Esempio " 141

7. Divisione procedura " 155
7.1. Istruzioni di entrata e di uscita " 155
7.1.1. Il verbo OPEN " 155
7.1.2. Il verbo CLOSE " 156
7.1.3. Il verbo READ " 157
7.1.4. Il verbo WRITE " 159
7.2. L'istruzione di trasferimento " 160
7.3. Istruzioni aritmetiche " 162
7.3.1. Arrotondamento " 162
7.3.2. Overflow del risultato " 163
7.3.3. Il resto della divisione " 164
7.3.4. I formati completati con le clausole " 165
Addizione – Sottrazione – Moltiplicazione – Divisione – Osservazioni
7.3.5. Il verbo COMPUTE " 168
7.4. Particolarità sui nomi " 170
7.4.1. Clausola CORRESPONDING per istruzioni aritmetiche " 173
7.4.2. Clausola CORRESPONDING per il verbo MOVE " 174
7.5. Frasi condizionali " 175
7.5.1. Condizioni semplici " 175
Test di relazione – Test di classe – Test di segno – Test di condizione
7.5.2. Condizioni composte " 178
7.5.3. Il verbo IF " 181
7.5.4. IF nidificate " 183

7.6. Il verbo GO TO pag. 185
7.7. Il verbo ALTER " 187
7.8. Sottoprogrammi interni " 188
7.8.1. Il verbo PERFORM " 188
7.8.2. PERFORM ... TIMES " 191
7.8.3. PERFORM ... UNTIL... " 192
7.8.4. PERFORM nidificate " 193
7.9. Trattamento delle stringhe di caratteri " 195
7.9.1. Il verbo INSPECT " 195
Formato 1 – Formato 2 – Formato 3
7.9.2. Il verbo STRING " 201
7.9.3. Il verbo UNSTRING " 204
7.10 Il verbo ENTER " 207
7.11. Esempio " 207

8. Le tabelle " 228
8.1. Tabelle gestite dal programmatore " 229
8.1.1. Definizione " 229
8.1.2. Il "particolare" elemento di tabella " 232
8.1.3. Caricamento delle tabelle " 233
Caricamento con VALUE – Caricamento da flusso esterno
8.1.4. Modalità di utilizzo di una tabella " 245
Accesso diretto – Ricerca tabellare
8.2. Tabelle gestite dal sistema " 251
8.2.1. Definizione delle tabelle " 251
8.2.2. Riferimenti a campi di tabella " 252
8.2.3. Indirizzamento relativo " 253
8.2.4. USAGE IS INDEX " 253
8.2.5. Caricamento delle tabelle " 254
8.2.6. Il verbo SET " 254
8.2.7. Il verbo SEARCH " 255
8.2.8. Esempi " 258
8.3. Tabelle multidimensionali " 259
8.3.1. Subscripting " 259
8.3.2. Indexing " 261
8.4. PERFORM... VARYING... " 262
8.5. Un esempio " 267

9. Organizzazione sequenziale " 278
9.1. Concetti generali " 278
9.2. Riferimenti in ENVIRONMENT DIVISION " 279
9.3. Riferimenti in DATA DIVISION " 280
9.4. Riferimenti in PROCEDURE DIVISION " 281
9.4.1. Il verbo OPEN " 281
9.4.2. Il verbo CLOSE " 283
9.4.3. Il verbo READ " 283
9.4.4. Il verbo WRITE " 284
9.4.5. Il verbo REWRITE " 285

9.4.6. DECLARATIVES pag. 286
Generalità – Il verbo USE
9.5. Un esempio " 288
9.6. Riepilogo " 297

10. Organizzazione "relative" " 302
10.1. Concetti generali " 302
10.2. Riferimenti in ENVIRONMENT DIVISION " 303
10.3. Riferimenti in DATA DIVISION " 304
10.4. Riferimenti in PROCEDURE DIVISION " 304
10.4.1. Il verbo OPEN " 305
10.4.2. Il verbo CLOSE " 305
10.4.3. Il verbo READ " 306
10.4.4. Il verbo WRITE " 309
10.4.5. Il verbo REWRITE " 310
10.4.6. Il verbo START " 310
10.4.7. Il verbo DELETE " 311
10.4.8. Il verbo USE " 312
10.5. Un esempio " 313
10.6. Riepilogo " 326
10.6.1. Accesso sequenziale " 326
10.6.2. Accesso random " 329
10.6.3. Accesso dynamic " 331

11. Organizzazione "indexed" " 334
11.1. Concetti generali " 335
11.2. Riferimenti in ENVIRONMENT DIVISION " 335
11.3. Riferimenti in DATA DIVISION " 336
11.4. Riferimenti in PROCEDURE DIVISION " 336
11.4.1. Il verbo OPEN " 336
11.4.2. Il verbo CLOSE " 338
11.4.3. Il verbo READ " 338
11.4.4. Il verbo WRITE " 341
11.4.5. Il verbo REWRITE " 342
11.4.6. Il verbo START " 343
11.4.7. Il verbo DELETE " 344
11.4.8. Il verbo USE " 344
11.5. Un esempio " 345
11.6. Un altro esempio " 352
11.7. Riepilogo " 361
11.7.1. Accesso sequenziale " 361
11.7.2. Accesso random " 364
11.7.3. Accesso dynamic " 365

Appendici
A. Cenni sul modulo Sort-Merge " 371
B. Cenni sul modulo Report Writer " 375
C. Cenni sul modulo Segmentation " 378
D. Cenni sul modulo Library " 380

E. Cenni sul modulo Debug pag. 382
F. Cenni sul modulo Inter-Program Communication " 386
G. Cenni sul modulo Communication " 391

Indice analitico " 393

0. RICONOSCIMENTO

Gli autori ed i detentori dei diritti del materiale utilizzato per la preparazione di qualsiasi pubblicazione riguardante il linguaggio Cobol richiedono sia premessa la seguente dichiarazione:

"Any organization interested in reproducing the COBOL report and specifications in whole or in part, using ideas from this report as the basis for an instruction manual or for any other purpose is free to do so. However, all such organizations are requested to reproduce the following acknowledgment paragraphs in their entirety as part of the preface to any such publication. Any organization using a short passage from this document, such as in a book review, is requested to mention "COBOL" in acknowledgment of the source, but need not quote the acknowledgment.

COBOL is an industry language and is not the property of any company or group of companies, or of any organization or group of organizations.
No warranty, expressed or implied, is made by any contributor or by the COBOL Committee as to the accuracy and functioning of the programming system and language. Moreover, no responsibility is assumed by any contributor, or by the committee, in connection therewith. The authors and copyright holders of the copyrighted material used herein.

FLOW-MATIC (Trademark of Sperry Raud Corporation), Programming for the Univac (R) I e II, Data Automation Systems copyrighted 1958, 1959, by Sperry Raud Corporation; IBM Commercial Translator Form No. 28-8013, copyrighted 1959 by IBM; FACT, DSI 27A5260–2760, copyrighted 1960 by Minneapolis-Honeywell have specifically authorized the use of this material in whole or in part, in the COBOL specifications. Such authorization extends to the reproduction and use of COBOL specifications in programming manuals or similar pubblications."

1. INTRODUZIONE

1.1. Cenni sui linguaggi di programmazione

La codifica di un programma è un momento del lavoro di programmazione. Nel programmare una fase, terminata la stesura dello schema logico, è necessario convertire le operazioni logiche, in esso rappresentate mediante simboli grafici, in istruzioni comprensibili ed eseguibili dell'elaboratore.

Il linguaggio comprensibile dall'elaboratore, cioè il linguaggio macchina, è un insieme di codici binari, che non hanno alcun rapporto mnemonico con la funzione che comandano.

Praticamente da sempre, al fine di superare le difficoltà della programmazione in linguaggio macchina, le Case costruttrici di elaboratori hanno fornito agli utenti dei "linguaggi simbolici orientati alla macchina", ovvero dei linguaggi che permettono di scrivere i programmi utilizzando dei simboli mnemonici, che richiamano il tipo di funzione da eseguire. Un linguaggio simbolico di questo tipo è però strettamente connesso alle possibilità logiche dell'elaboratore (al fine di sfruttarne appieno le risorse), pertanto al programmatore è richiesta una approfondita conoscenza delle possibilità della macchina che deve essere utilizzata.

Si è sentita allora la necessità di creare dei linguaggi indipendenti dalle caratteristiche della macchina e orientati alla logica umana, per consentire una più facile scrittura dei programmi.

Per risolvere questo problema sono stati messi a punto numerosi linguaggi, non solo "vicini" alla logica umana, ma addirittura specializzati alla risoluzione di problemi specifici: una caratteristica essenziale dei più diffusi di questi linguaggi è che possono essere utilizzati, tutt'al più con modifiche marginali, su qualunque classe di elaboratori.

Esaminiamo ora brevemente alcuni linguaggi di questo tipo.

1.1.1. *Algol*

Le prime formulazioni di questo linguaggio risalgono agli inizi degli anni sessanta; nelle versioni successive si sono avuti revisioni e perfezionamenti marginali: il fatto che non vi siano stati sostanziali cambiamenti (la versione definitiva è praticamente quella del '62) è indice della bontà di questo linguaggio.

L'Algol è stato implementato in modo eccellente nella versione Algol W ('65) dalla Ibm e dalla Burroughs ed inoltre per l'Algol 68 è stato preparato un ottimo compilatore (JCL 1907) dalla UK Royal Radar, che giudica questo linguaggio molto potente.

L'Algol esprime in forma concisa un'ampia classe di problemi numerici connessi all'automazione, ha introdotto l'idea di "statement" composti, di dichiarazioni locali e condizionali, e la parentizzazione delle espressioni.

L'Algol è stato criticato perchè, anche se le prime versioni avevano una buona formulazione generale, non erano stati curati i dettagli delle specifiche delle singole istruzioni.

In particolare va rilevato che:

a. ha un'alta velocità di compilazione;
b. la diagnostica è molto buona;
c. la velocità di esecuzione è elevata;
d. ha un ampio set di applicazione (superiore al Fortran);
e. è sostanzialmente semplice.

1.1.2. *APL*

L'Application Purpose Language è stato formulato dall'Ibm negli anni sessanta ed è stato messo a disposizione degli utenti dopo alcuni anni di uso esclusivamente interno.

L'APL utilizza notazioni di tipo commerciale, ma è essenzialmente rivolto a problemi scientifici: ha avuto un discreto successo ed ha un crescente, anche se limitato, numero di utenti.

1.1.3. *Basic*

Il Basic, nato negli Usa agli inizi degli anni sessanta, è stato formalizzato e messo a punto presso il Dartmounth College di Hanover (Ohio), con la collaborazione della General Electric, nel contesto di

un progetto per la creazione di un sistema per l'elaborazione dati in time-sharing per uso scientifico e didattico.

Il Basic è un linguaggio di programmazione facile, bastano poche ore per apprenderlo e pochi giorni per padroneggiarlo completamente,è soprattutto orientato ai non addetti ai lavori, infatti consente di automatizzare problemi anche ai non esperti di programmazione.

1.1.4. *Cobol*

E' il linguaggio più largamente usato per problemi gestionali: la sua larga diffusione deriva dal fatto che è stato il primo linguaggio evoluto, disponibile per maneggiare dati in termini di record piuttosto che di stringhe di caratteri.

E' opinione diffusa che sia un linguaggio semplice da usare ed è considerato verboso.

Le ultime versioni del Cobol prevedono la possibilità di gestire archivi organizzati in modi diversi, trasmissione dati e tecniche automatiche di stampa.

Il Cobol è particolarmente usato anche perchè le specifiche del linguaggio sono regolamentate da "standard" piuttosto rigidi, che consentono di preparare programmi adattabili facilmente ad ogni tipo di elaboratore.

1.1.5. *Fortran*

Il Fortran è, come il Cobol, un linguaggio standardizzato, ma si è adeguato poco all'evoluzione tecnologica.

E' sostanzialmente un linguaggio per applicazioni scientifiche, valido dal punto di vista generale, ma non specializzato.

Una grave deficienza di questo linguaggio è dovuta al fatto che non ha mai subito revisioni ma solo implementazioni, e questo l'ha portato a prevedere una programmazione con strumenti raffinati su una struttura antiquata.

1.1.6. *Pascal*

E' un linguaggio evoluto poco noto che permette di sfruttare le capacità logiche della macchina.

Il Pascal è il tentativo di raccogliere e consolidare i vantaggi del Fortran (efficienza), dell'Algol (chiarezza) e del Cobol (flessibilità della descrizione dati).

1.1.7. *PL/1*

E' stato sviluppato dall'Ibm come principale linguaggio evoluto e punta ad essere un linguaggio di largo uso per problemi diversificati.

Il PL/1 è di frequente criticato per essere troppo ampio, e quindi poco maneggevole, e perchè contiene troppe eccezioni.

E' un linguaggio poco stabile, come dimostra l'ampio tempo di sviluppo, nonostante l'Ibm lo spinga molto presentandolo come la soluzione del problema della ricerca di un linguaggio evoluto atto a risolvere problemi di qualsiasi tipo.

Per il PL/1 ci sono buone prospettive di implementazione della velocità di traduzione e di esecuzione in quanto l'ambiente è lo stesso che per il Fortran ed il Cobol.

1.2. Scelta di un linguaggio di programmazione

I linguaggi di programmazione sono una delle principali aree di ricerca: attuali studi hanno messo in risalto che i linguaggi di tipo naturale sono meno complessi da realizzare di quanto si pensasse.

Nonostante questo vengono realizzati una serie di linguaggi di livello intermedio fra quello naturale e quello formale della specifica del problema, ed inoltre, oltre al gran numero dei linguaggi, si ha il proliferare dei "dialetti" (il Fortran ne ha più di cento).

La scelta del linguaggio di programmazione è un momento delicato in quanto richiede una dettagliata conoscenza dell'applicazione proposta e dei linguaggi del software disponibili, la loro efficienza di implementazione, le caratteristiche dell'elaboratore da usare, e la conoscenza delle capacità dello staff di programmazione e di manutenzione dell'applicazione.

Normalmente l'alternativa si pone fra i seguenti tipi di linguaggi:

a. assemblatori;
b. procedurali;
c. applicativi.

Una particolare configurazione di elaborazione ha normalmente a disposizione un linguaggio assemblatore insieme ad un linguaggio procedurale o "business oriented" (Cobol) o "formula oriented" (Fortran) o entrambi; per quanto riguarda i linguaggi orientati alle applicazioni si può affermare sono scarsamente disponibili (fatta eccezione per problemi di simulazione e per il controllo delle macchine utensili).

La programmazione in linguaggi assemblatori, pur essendo lenta e costosa, è capace di un'alta precisione che dipende però dall'esperienza del programmatore, quindi, proprio per questo, la tendenza attuale è rivolta ad una riduzione del loro impiego.

La maggior parte dei programmi di applicazioni sono scritti utilizzando un linguaggio di tipo "general purpose" come il Cobol, il Fortran ed il PL/1.

1.3. Lo sviluppo del Cobol

1.3.1. *Le origini*

L'idea di predisporre un linguaggio per applicazioni di tipo gestionale, che fosse ben accetto ad una larga fascia di utilizzatori, partì dal Pentagono: nel '59 fu organizzata una conferenza (a cui parteciparono Enti governativi degli USA, Case costruttrici di elaboratori e Grandi Utenti) che aveva il preciso scopo di valutare se era possibile realizzare un linguaggio evoluto, che avesse caratteristiche tali da poter essere accettato dalla maggior parte degli utenti che devono automatizzare problemi di natura gestionale.

I risultati della conferenza furono positivi, venne quindi nominato un gruppo di specialisti che si occupassero degli aspetti tecnici del problema; il Comitato preposto, nel settembre dello stesso anno, aveva già pronta una bozza del linguaggio, che venne pubblicato nell'aprile del '60 con il nome di COBOL (acronimo di COmmon Business Oriented Language).

Da allora le specifiche del linguaggio sono state continuamente aggiornate attraverso una serie di documenti preparati da un gruppo di comitati preposti alla cura e all'evoluzione del linguaggio.

L'ultima versione del linguaggio Cobol, ricca di elementi che consentono di sfruttare le tecniche messe a disposizione dai moderni sistemi operativi, è il Cobol ANS 74.

1.3.2. *Lo standard*

Lo standard Cobol è un insieme di norme che stabiliscono i formati delle frasi del linguaggio e le relative interpretazioni, al fine di garantire una assoluta indipendenza del programma dal tipo di elaboratore utilizzato.

La definizione delle specifiche del linguaggio Cobol, secondo lo standard, si articola in un nucleo ed in undici moduli, che consentono di esaurire le specifiche funzionali di un qualunque processo di elaborazione.

Oltre al nucleo (Nucleus), che contiene gli elementi del linguaggio necessari per l'elaborazione interna, vi sono i seguenti moduli:

a. trattamento delle tabelle (Table handling);
b. gestione dei flussi organizzati in modo sequenziale (Sequential I-O);
c. gestione dei flussi organizzati in modo casuale (Relative I-O);
d. gestione dei flussi organizzati con indice (Indexed I-O);
e. ordinamenti e fusioni (Sort-Merge);
f. stampa automatica dei tabulati (Report Writer);
g. segmentazione (Segmentation);
h. gestione delle librerie (Library);
i. messa a punto dei programmi (Debug);
l. connessioni fra programmi (Inter program communications);
m. trasmissione dati (Communication).

Ciascuna delle dodici parti dello standard (nucleo e undici moduli) è suddivisa in livelli, precisamente due o tre (livelli uno e due oppure livelli zero e uno oppure livelli zero, uno e due).

Livello zero indica assenza di specifiche relative all'argomento del modulo, livello uno indica presenza di un certo numero di specifiche base, livello due è un complemento di specifiche rispetto al livello uno.

In fig. 1.1 è riprodotta la mappa riepilogativa dello standard Cobol.

Quando il Cobol di un certo elaboratore è standard?

Affinché il Cobol utilizzato su un certo elaboratore si possa ritenere standard è sufficiente che rispetti per ogni voce della tabella di fig. 1.1 almeno il livello minimo.

Fig. 1.1

Nucleo	Moduli funzionali											
	Tabelle	I - O sequenz.	I - O casuale	I - O con indice	Ordinam. fusione	Generaz. tabulati	Segmentazione	Librerie	Messa a punto	Comunic. fra progr.	Trasm. dati
2	2	2	2	2	2	1	2	2	2	2	2
1	1	1	1	1	1	0	1	1	1	1	1
			0	0	0		0	0	0	0	0

1.4. Presentazione del volume

Il volume è strutturato in undici capitoli e in sette appendici, la conoscenza dei primi cinque capitoli consente di acquisire gli elementi essenziali per poter scrivere un semplice programma in linguaggio Cobol.

Nei capitoli 6 e 7 vengono completate le specifiche delle frasi dichiarative e delle istruzioni trattate nei capitoli precedenti e vengono presentate nuove opzioni ed istruzioni.

Il capitolo 8 è dedicato alla presentazione delle frasi Cobol necessarie per trattare le tabelle secondo le tecniche "subscripting" ed "indexing".

Nei successivi capitoli 9, 10 ed 11 sono analizzati problemi di input-output in relazione alle organizzazioni "Sequential", "Relative" ed "Indexed": ricordiamo che nei primi capitoli vengono trattati i flussi con organizzazione ed accesso sequenziali, senza neppure accennare alle possibilità di riregistrazione su flussi su disco magnetico organizzati in modo sequenziale.

Infine, nelle appendici, si provvede ad accennare agli elementi essenziali degli "altri moduli dello standard".

2. CONSIDERAZIONI GENERALI

2.1. Preliminari

2.1.1. *Set grafico*

Un programma in linguaggio Cobol è composto da unità espressive elementari, che chiamiamo in modo generico "parole", combinate secondo precise regole: la classificazione dei tipi di parole, che possono essere utilizzate per comporre le frasi del linguaggio, è dettagliata e rigida, nonostante questo, per non appesantire troppo la trattazione, noi seguiremo la classificazione standard solo a grandi linee, impegnandoci, quando se ne presenti la necessità, di fornire le precisazioni dovute.

I simboli del set grafico che vengono utilizzati per comporre gli elementi del linguaggio (verbi, variabili, costanti, ...) sono:

a. le lettere dell'alfabeto inglese;
b. le dieci cifre;
c. i simboli:

b(spazio) + - * / = $
, ; . ( ) " > <

2.1.2. *Parole*

Tra le parole, in una prima sommaria classificazione, distinguiamo:

a. le parole Cobol
b. le parole del programmatore.

Le parole Cobol sono classificabili in obbligatorie e facoltative.

Le parole Cobol obbligatorie permettono di fornire al compilatore le informazioni essenziali alla traduzione del programma in linguaggio macchina.

Le parole Cobol facoltative consentono di scrivere il programma in forma discorsiva, non rivestono alcun interesse per il compilatore (che però esegue un controllo sulla loro validità formale), ma permettono di interpretare il programma con maggiore facilità.

Le parole Cobol sono riservate (fig. 2.1), non possono cioè essere scelte dal programmatore per scopi diversi da quelli previsti dalle regole del linguaggio.

Tra le parole Cobol, oltre alla distinzione tra obbligatorie e facoltative, è possibile classificare, ad esempio, i verbi e le parole chiave. Un verbo di procedura è l'identificativo di una istruzione, una parola chiave consente di fornire certe informazioni relative all'interpretazione di quello che l'istruzione deve eseguire (ad esempio gli ordini di movimento carta associati agli ordini di stampa).

Fig. 2.1 – Elenco delle parole riservate

ACCEPT	ACCESS	ADD	ADVANCING
AFTER	ALL	ALPHABETIC	ALTER
ALTERNATE	AND	ARE	AREA
AREAS	ASCENDING	ASSIGN	AT
AUTHOR			
BEFORE	BLANK	BLOCK	BOTTOM
BY			
CALL	CANCEL	CD	CF
CH	CHARACTER	CHARACTERS	CLOCK-UNITS
CLOSE	COBOL	CODE	COLUMN
COMMA	COMMUNICATION	COMP	COMPUTATIONAL
COMPUTE	CONFIGURATION	CONTAINS	CONTROL
CONTROLS	COPY	CORR	CORRESPONDING
COUNT	CURRENCY		
DATA	DATE	DATE-COMPILED	DATE-WRITTEN
DAY	DE	DEBUG-CONTENTS	DEBUG-ITEM
DEBUG-LINE	DEBUG-SUB-1	DEBUG-SUB-2	DEBUG-SUB-3
DEBUG-NAME	DEBUGGING	DECIMAL-POINT	DECLARATIVES
DELETE	DELIMITED	DELIMITER	DEPENDING

Fig. 2.1 – segue

DEPTH	DESCENDING	DESTINATION	DETAIL
DISABLE	DISPLAY	DIVIDE	DIVISION
DOWN	DUPLICATES	DYNAMIC	
EGI	ELSE	EMI	ENABLE
END	END-OF-PAGE	ENTER	ENVIRONMENT
EQUAL	EOP	ERROR	ESI
EVERY	EXCEPTION	EXIT	EXTEND
FD	FILE	FILE-CONTROL	FILLER
FINAL	FIRST	FOOTING	FOR
FROM			
GENERATE	GIVING	GO	GREATER
GROUP			
HEADING	HIGH-VALUE	HIGH-VALUES	
I-O	I-O-CONTROL	IDENTIFICATION	IF
IN	INDEX	INDEXED	INDICATE
INITIAL	INITIATE	INPUT	INPUT-OUTPUT
INSPECT	INSTALLATION	INTO	INVALID
IS			
JUST	JUSTIFIED		
KEY			
LABEL	LAST	LEADING	LEFT
LENGTH	LESS	LIMIT	LIMITS
LINAGE	LINAGE-COUNTER	LINE	LINE-COUNTER
LINES	LINKAGE	LOOK	LOW-VALUE
LOW-VALUES			
MEMORY	MERGE	MESSAGE	MODE
MODULES	MOVE	MULTIPLE	MULTIPLY
NEGATIVE	NEXT	NO	NOT
NUMBER	NUMERIC		
OCCURS	OBJECT-COMPUTER	OF	OFF
OMITTED	ON	OPEN	OPTIONAL
OR	ORGANIZATION	OUTPUT	OVERFLOW

Fig. 2.1 – segue

PAGE	PAGE-COUNTER	PERFORM	PF
PH	PIC	PICTURE	PLUS
POINTER	POSITIVE	PRINTING	PROCEDURE
PROCEDURES	PROCEED	PROGRAM	PROGRAM-ID
QUEUE	QUOTE	QUOTES	
RANDOM	RD	READ	RECEIVE
RECORD	RECORDS	REDEFINES	REEL
REFERENCES	RELATIVE	RELEASE	REMAINDER
REMOVAL	RENAMES	REPLACING	REPORT
REPORTING	REPORTS	RERUN	RESERVE
RESET	RETURN	REVERSED	REWIND
REWRITE	RF	RH	RIGHT
ROUNDED	RUN		
SAME	SD	SEARCH	SECTION
SECURITY	SEGMENT	SEGMENT-LIMIT	SELECT
SEND	SENTENCE	SEPARATE	SEQUENTIAL
SET	SIGN	SIZE	SORT
SORT-MERGE	SOURCE	SOURCE-COMPUTER	SPACE
SPACES	SPECIAL-NAMES	STANDARD	START
STATUS	STOP	STRING	SUB-QUEUE-1
SUB-QUEUE-2	SUB-QUEUE-3	SUBTRACT	SUM
SUPPRESS	SYMBOLIC	SYNC	SYNCHRONIZED
TABE	TALLYING	TAPE	TERMINAL
TERMINATE	TEXT	THAN	THROUGH
THRU	TIME	TIMES	TO
TOP	TRAILING	TYPE	
UNIT	UNSTRING	UNTIL	UP
UPON	USAGE	USE	USING
VALUE	VALUES	VARYING	
WHEN	WITH	WORDS	WORKING-STORAGE
WRITE			
ZERO	ZEROES	ZEROS	

+ – * / ** > < =

Le parole del programmatore, pur rispettando alcune precise regole, sono arbitrariamente scelte dall'utente per individuare flussi, record, campi, aree di lavoro, ecc...

Possiamo così riassumere le regole per la costituzione dei nomi da parte del programmatore:

a. un nome deve essere composto solo da cifre, da lettere dell'alfabeto inglese ed eventualmente dal simbolo trattino "–";
b. un nome non può iniziare o terminare con il simbolo trattino e neppure contenere trattini consecutivi;
c. la lunghezza massima prevista è di 30 caratteri.

Osserviamo che solo dalla versione dello standard del 2/74 è prevista la possibilità che un nome possa iniziare con una cifra, e che non ci risulta che questa possibilità sia reale sulla maggior parte degli elaboratori.

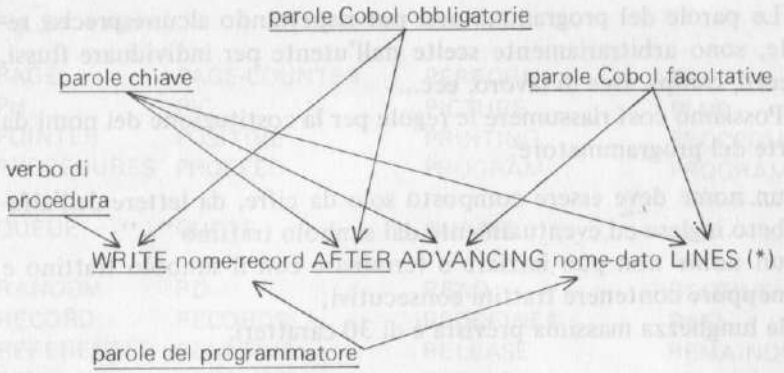
Talvolta la scrittura dei nomi è soggetta anche ad altre limitazioni, è ad esempio il caso di quei nomi che debbano essere gestiti anche da ordini del sistema operativo.

Esempi

a. Esempi di nomi validi e no:

COD-ART	
DESCRIZIONE	
IND1	
VAL-TOT-ST	
A512BETA	
VAL/ST	(non valido: /)
AREA- -REC	(non valido: - -)
-MATR1	(non valido: -)
DATA	(non valido: parola riservata)
VAL MAX	(non valido: b)
ABCDEFG...WZ12345	(non valido: 31 caratteri)

b. Esempio sui tipi di parole (evidentemente non entriamo nel merito di questa istruzione, ma limitiamoci a considerarne il significato intuitivo):



(*) STAMPA nome-record DOPO (un AVANZAMENTO di) nome-dato (LINEE).

2.1.3. *Costanti*

Il programma, oltre che dati variabili, a cui si fa riferimento tramite i nomi mnemonici assegnati ai campi atti a contenerli, deve spesso trattare valori costanti.

In un programma Cobol si possono introdurre direttamente, rispettando opportune regole ortografiche, delle costanti (parliamo in questo caso di "literal") oppure si possono utilizzare, in alcuni casi, delle parole riservate (dette "costanti figurative") che consentono di esprimere alcune costanti di largo uso.

Le costanti dirette (introdotte dal programmatore per assegnare un valore ad un'area, per fare dei test, ecc...) possono essere numeriche o alfanumeriche: la lunghezza massima prevista è di 18 e 120 caratteri rispettivamente, ma ci possono essere delle variazioni in relazione al tipo di elaboratore che si utilizza.

Le costanti numeriche possono essere senza segno, positive o negative, intere o decimali: la separazione fra la parte intera e la parte decimale del numero è ottenuta dal simbolo punto "." (notazione anglosassone), il simbolo punto deve essere utilizzato solo se è presente almeno una cifra decimale. Il segno, se presente, deve precedere la prima cifra significativa.

Le costanti alfanumeriche, che possono essere scritte utilizzando qualunque carattere del set grafico, devono essere racchiuse fra virgolette: le virgolette, in quanto delimitatori, non possono far parte di una costante alfanumerica.

Esempi:

a. Costanti alfanumeriche:

"DATA"
"INDIRIZZO"
"TOTALE = "
GIORNO (non valida)
"GIORNO (non valida)
"GIO"12X" (non valida)

b. Costanti numeriche:

1234
+123.4
-.1234
-55.678
12A34 (non valida)
123. (non valida)
25- (non valida)

Le costanti figurative disponibili sono:

ZERO ZEROS ZEROES	tre formati equivalenti per indicare uno o più zeri
SPACE SPACES	due formati equivalenti per indicare uno o più spazi
HIGH-VALUE HIGH-VALUES	due formati equivalenti per indicare uno o più valori binari di peso massimo
LOW-VALUE LOW-VALUES	due formati equivalenti per indicare uno o più valori binari di peso minimo
QUOTE QUOTES	due formati equivalenti per indicare una o più volte le virgolette: non possono essere utilizzati come deliminatori di costanti ma possono essere assegnati come valori ad un'area di memoria.
ALL costante:	ALL rappresenta, per la costante indicata (che deve essere alfanumerica), un fattore di moltiplicazione: la costante viene ripetuta tante volte in relazione alla lunghezza dell'area in oggetto. E' possibile anche utilizzare una costante figurativa: in tal caso ALL é ridondante in quanto ogni costante figurativa ha insito il fattore di ripetizione.

2.1.4. *Punteggiatura*

Il linguaggio Cobol prevede la possibilità di utilizzare i seguenti caratteri di punteggiatura:

, (virgola)
; (punto e virgola)
. (punto)

La virgola ed il punto e virgola, quando sono presenti nei formati delle istruzioni, hanno solo funzione di tipo documentativo, pertanto, per evitare di dover introdurre degli elementi ridondanti (che se sono mal utilizzati danno origine ad errori), non li prenderemo in onsiderazione nella trattazione che segue.

Il simbolo punto ha invece un interesse diverso; infatti è richiesto in posizione fissa in molte circostanze come indicatore di fine frase ed inoltre viene utilizzato come elemento determinante nel "modo" di esecuzione di alcune istruzioni: raccomandiamo di tenere presente il suo ruolo nei casi in cui se ne prevede l'utilizzo.

Ricordiamo che un erroneo inserimento di "punti" nel programma può dare origine non solo ad errori formali, ma anche ad errori logici, che sono i più difficilmente rilevabili.

2.1.5. *Notazioni*

Il linguaggio Cobol utilizza, per specificare la sintassi delle frasi che lo costituiscono, un insieme di notazioni particolarmente semplice e versatile.

Nella presentazione dei formati delle frasi del linguaggio le parole Cobol sono scritte con lettere maiuscole e quelle obbligatorie sono sottolineate.

Le parole del programmatore sono scritte in lettere minuscole, e la terminologia utilizzata richiama il tipo di elemento che il programmatore deve inserire nella frase.

Esempio:

Consideriamo il formato ridotto:

WRITE nome-record BEFORE ADVANCING intero LINES

che significa:

Prima registra il "record", poi avanza di "intero" righe.

In questo formato WRITE, BEFORE, ADVANCING e LINES sono parole Cobol, di queste solo WRITE e BEFORE sono obbligatorie; nome-record e intero sono invece parole del programmatore.

Il Cobol per indicare che una data porzione del formato è opzionale (o perchè indica funzioni che possono o meno essere richieste, o perchè tali funzioni possono essere utilizzate solo sotto determinate ipotesi), la pone tra parentesi quadrate.

Esempio:

PERFORM nome-procedura$_1$ [THRU nome-procedura$_2$]

consente di evidenziare che è possibile utilizzare sia il formato ridotto

PERFORM nome-procedura$_1$

sia quello completo:

PERFORM nome-procedura$_1$ THRU nome-procedura$_2$

In linguaggio Cobol, quando due espressioni sono scritte in parallelo, significa che l'istruzione può essere scritta utilizzando una delle due espressioni in alternativa (se sono racchiuse fra parentesi graffe), oppure (se sono racchiuse fra parentesi quadre) significa che sono opzionali, ma, se sono utilizzate sono in alternativa.

Esempio:

$$\left\{\begin{array}{l}\text{identificatore}\\ \text{intero}\\ \text{nome-dato}\end{array}\right\} \left[\begin{array}{l}\text{LINE}\\ \text{LINES}\end{array}\right]$$

in questo caso il formato precisa che deve essere usata una delle tre possibilità: identificatore, intero oppure nome dato e, in particola-

re, agli ultimi due casi è possibile associare una delle due parole opzionali LINE oppure LINES.

Evidentemente i casi che abbiamo esaminato sono particolarmente semplici, ma, anche nel caso di formati più complessi, si ha sempre una notevole facilità di interpretazione.

Esempio:

```
WRITE nome-record [FROM nome-dato1]
[ { BEFORE } ADVANCING { identificatore } ]
  { AFTER  }           { PAGE          }
                       { intero        }
                       { nome-dato     }
```

da questo formato si può dedurre che l'istruzione in oggetto può essere scritta semplicemente nella forma

```
WRITE nome-record
```

oppure con l'aggiunta della prima opzione o della seconda oppure di entrambe.

Per quanto riguarda la prima opzione non ci sono particolari considerazioni da fare in quanto è costituita da una parola Cobol obbligatoria e da un nome del programmatore.

L'esame della seconda opzione presenta una casistica più varia, infatti all'alternativa AFTER/BEFORE è associata indifferentemente una delle due possibili alternative: identificatore/PAGE oppure intero/nome-dato.

Osserviamo infine che la sintassi del Cobol offre la possibilità di indicare le iterazioni attraverso l'aggiunta di tre punti, al chiudersi della parentesi che racchiude la quantità da iterare.

Esempio:

```
MOVE nome-dato TO nome-dato1 [nome-dato2]...
```

Come si può notare dall'esempio, l'espressione che può essere ripetuta (nel nostro caso un nome-dato) viene scritta una prima

volta normalmente, in quanto deve comparire almeno una volta, una seconda volta tra parentesi quadre, e quindi vengono aggiunti i tre punti.

2.1.6. *Foglio di programmazione*

E' previsto che un programma Cobol venga scritto rispettando il tracciato di un foglio di programmazione che prevede:

colonne 1÷6 contengono il numero di sequenza della riga nell'ambito del programma;
colonna 7 può assumere i valori ƀ, –, *,/,D;
colonne 8-72 servono per scrivere tutte le proposizioni di un programma: alcune a partire da colonna 8; altre da colonna 12;
colonne 73-80 sono utilizzate, eventualmente, per l'identificazione del programma.

Le sei posizioni destinate a contenere il numero di sequenza della riga nell'ambito del programma sono di solito divise in due gruppi di tre cifre, il primo individua il numero di pagina, mentre il secondo individua il numero di riga nell'ambito della pagina: per consentire inserimenti è utile non attribuire alle righe valori successivi.

Normalmente i compilatori possono eseguire dei controlli sulla sequenzialità del pacco di schede che costituiscono il programma sorgente.

In colonna 7 viene normalmente lasciato il carattere spazio (ƀ); il carattere "–" serve per indicare che la costante alfanumerica che compare nella riga attuale è la continuazione di una iniziata nella riga precedente, i caratteri "*" e "/" indicano al compilatore che la riga attuale è di commento e quindi deve essere ignorata (in particolare nel listing di compilazione tale riga viene stampata all'inizio di pagina se in colonna 7 è indicata la "/"), "D" sta ad indicare che la riga serve per il debugging del programma (appendice E).

Le ultime 8 posizioni della riga (73÷80), che vengono utilizzate per contenere un identificativo del programma (di solito il nome del programma), sono soggette a controllo da parte del compilatore Cobol, che segnala la presenza di eventuali schede estranee nel programma sorgente; si può però sempre evitare che tale controllo sia eseguito.

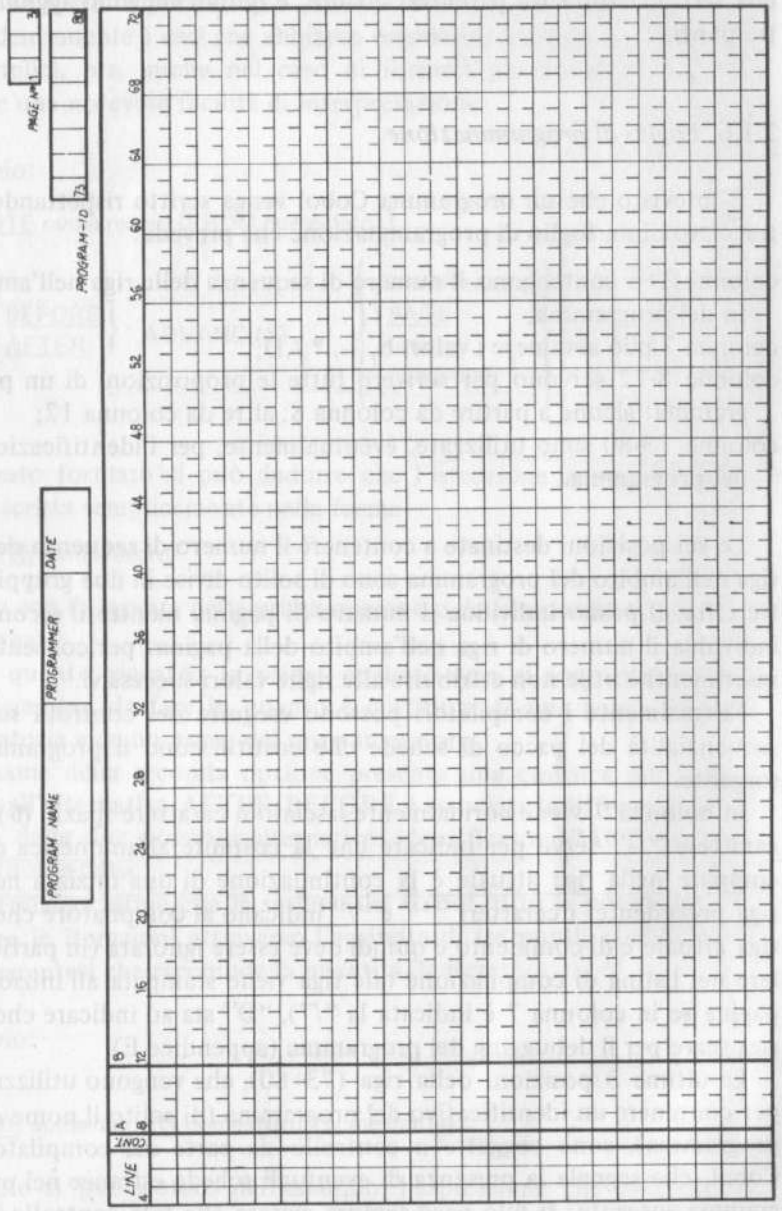


Fig. 2.2

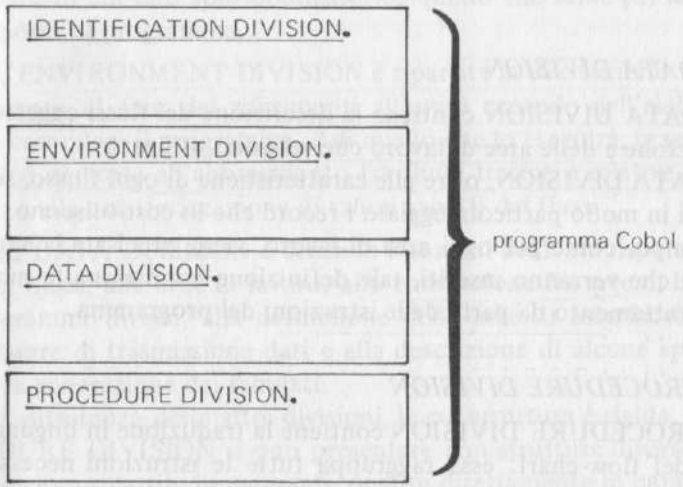
Tutte le proposizioni del programma sono inserite nella zona da colonna 8 a 72 (propriamente detta zona A) oppure limitatamente all'insieme di posizioni da colonna 12 a 72 (zona B).

In generale vengono scritti a partire da zona A i nomi-divisione, i nomi-sezione, i nomi-paragrafo, alcuni identificatori e numeri di livello, ecc...: questi elementi saranno introdotti gradualmente di seguito.

In fig. 2.2 è riportato un esempio di foglio di programmazione Cobol, osserviamo che ci possono essere delle variazioni grafiche tra moduli di diversi utenti, ma che la struttura è quella ora descritta.

2.2. Struttura di un programma

In ogni programma Cobol si identificano quattro divisioni (obbligatorie):



ognuna di queste divisioni ha alcune specifiche funzioni.

2.2.1. *IDENTIFICATION DIVISION*

Nell'IDENTIFICATION DIVISION viene attribuito il nome del programma (nome che sarà utilizzato per identificarlo usando gli ordini del sistema operativo); è inoltre possibile introdurre in questa divisione alcune informazioni documentative (facoltative) e, in generale, frasi di commento.

2.2.2. *ENVIRONMENT DIVISION*

Questa divisione è la parte del programma nella quale viene stabilito un collegamento fra il programma (ovvero l'utilizzatore) e l'elaboratore: si forniscono infatti informazioni relative all'unità centrale ed alle periferiche utilizzate.

Per sua natura l'ENVIRONMENT DIVISION, pur rispettando una ben precisa struttura richiesta dalle regole del linguaggio, si presta a contenere descrizioni aggiuntive ed eccezioni rispetto allo standard.

2.2.3. *DATA DIVISION*

La DATA DIVISION contiene la descrizione dei flussi oggetto di elaborazione e delle aree di lavoro che vengono utilizzate.

In DATA DIVISION, oltre alle caratteristiche di ogni flusso, sono descritti in modo particolareggiato i record che lo costituiscono: per ogni campo, come per ogni area di lavoro, viene precisata la natura dei dati che verranno inseriti, tale definizione condiziona le modalità di trattamento da parte delle istruzioni del programma.

2.2.4. *PROCEDURE DIVISION*

La PROCEDURE DIVISION contiene la traduzione in linguaggio Cobol del flow-chart: essa raggruppa tutte le istruzioni necessarie per descrivere le operazioni previste dal programma.

La PROCEDURE DIVISION è un insieme di istruzioni che operano sui dati descritti nella DATA DIVISION, appartenenti a flussi associati alle unità periferiche dichiarate in ENVIRONMENT DIVISION, nell'ambito di un programma il cui nome è stabilito in IDENTIFICATION DIVISION.

2.2.5. *Struttura delle divisioni*

Ogni programma Cobol è composto da quattro divisioni (il cui nome deve essere l'unico elemento di una riga, seguito da uno spazio, dalla parola DIVISION e dal punto).

Ogni divisione può essere ripartita in sezioni ed ogni sezione in paragrafi; ma non esiste una regola comune a tutte le divisioni.

Le sezioni sono identificate da un nome-sezione (scelto dal programmatore o imposto dalle regole del linguaggio) seguito da uno spazio, dalla parola SECTION e da un punto; all'assegnazione del nome ad una sezione è riservata un'intera riga (a partire dall'inizio della zona A).

I paragrafi sono identificati da nomi di paragrafo (che possono essere scelti dal programmatore oppure imposti dalle regole del Cobol), seguiti dal punto.

L'IDENTIFICATION DIVISION è direttamente ripartita in paragrafi, di cui uno solo obbligatorio: quello che serve per assegnare un nome al programma.

L'ENVIRONMENT DIVISION è ripartita in due sezioni: la prima consente di fare dei riferimenti all'unità centrale dell'elaboratore che compilerà il programma, e di quello che lo eseguirà, la seconda è invece dedicata all'abbinamento fra flussi trattati e relative periferiche, e alla caratterizzazione di taluni aspetti dei flussi.

La DATA DIVISION è divisa in 5 sezioni dedicate rispettivamente ai flussi, alle aree di lavoro, alla connessione fra aree di lavoro di programmi diversi, alla definizione delle aree di interfaccia con il software di trasmissione dati e alla descrizione di alcune specifiche per la generazione dei tabulati.

A differenza delle altre divisioni, la cui struttura è rigida, la PROCEDURE DIVISION si può presentare con strutture diverse: sezioni, sezioni ripartite in paragrafi, oppure direttamente in paragrafi.

La scelta del tipo di ripartizione della PROCEDURE DIVISION è condizionata dalle esigenze di programmazione, in ogni caso può essere conveniente strutturare questa divisione in sezioni per diversi motivi che esamineremo in seguito.

3. RELAZIONI FRA PROGRAMMA ED ELABORATORE

3.1. IDENTIFICATION DIVISION

L'IDENTIFICATION DIVISION, la prima divisione di ogni programma Cobol, ha due funzioni:

a. consente di attribuire un nome al programma;
b. permette di inserire frasi di commento, quali ad esempio:
   – il nome del programmatore,
   – le caratteristiche del programma,
   – la data di preparazione e di compilazione,
   – ecc...

Dei paragrafi che costituiscono l'IDENTIFICATION DIVISION l'unico obbligatorio è quello in cui si assegna il nome al programma; gli altri paragrafi sono invece opzionali e possono essere utilizzati a scopo documentativo.

PROGRAM-ID . nome del programma

è il paragrafo che consente di assegnare il nome al programma: la scelta del nome è subordinata sia alle regole ortografiche imposte dal Cobol, sia alle regole di formazione dei nomi del sistema operativo nel cui ambito deve essere compilato ed eseguito il programma.

Il nome del programma deve rispettare anche le limitazioni del sistema operativo in quanto il nome associato al programma nella PROGRAM-ID è quello che servirà per identificarlo in tutti i possibili trattamenti di tipo operativo.

I paragrafi facoltativi, se presenti, devono seguire il paragrafo PROGRAM-ID ed essere inseriti nell'ordine e nel formato indicati in fig. 3.1.

Fig. 3.1

```
IDENTIFICATION DIVISION.
PROGRAM-ID. nome-del-programma.
[AUTHOR. commento.]
[INSTALLATION. commento.]
[DATE-WRITTEN. commento.]
[DATE-COMPILED. commento.]
[SECURITY. commento.]
```

Le frasi di commento non hanno limitazioni di lunghezza e possono occupare più righe consecutive (limitatamente alla zona B), in tal caso, non deve essere inserita alcuna indicazione in colonna 7.

Nelle frasi di commento non può essere utilizzato il carattere punto in quanto questo carattere è l'identificativo di fine commento.

I nomi-paragrafo utilizzati nella IDENTIFICATION DIVISION non sono vincolanti per il tipo di commento che può essere inserito: tale commento può essere infatti costituito da una qualunque combinazione di caratteri del set grafico.

In fig. 3.2 è riportato un esempio di IDENTIFICATION DIVISION.

3.2. ENVIRONMENT DIVISION

Questa divisione specifica, in due distinte sezioni, un metodo standard per esprimere i collegamenti fra programma ed elaboratore:

a. CONFIGURATION SECTION;
b. INPUT-OUTPUT SECTION.

Nella CONFIGURATION SECTION vengono fornite le specifi-

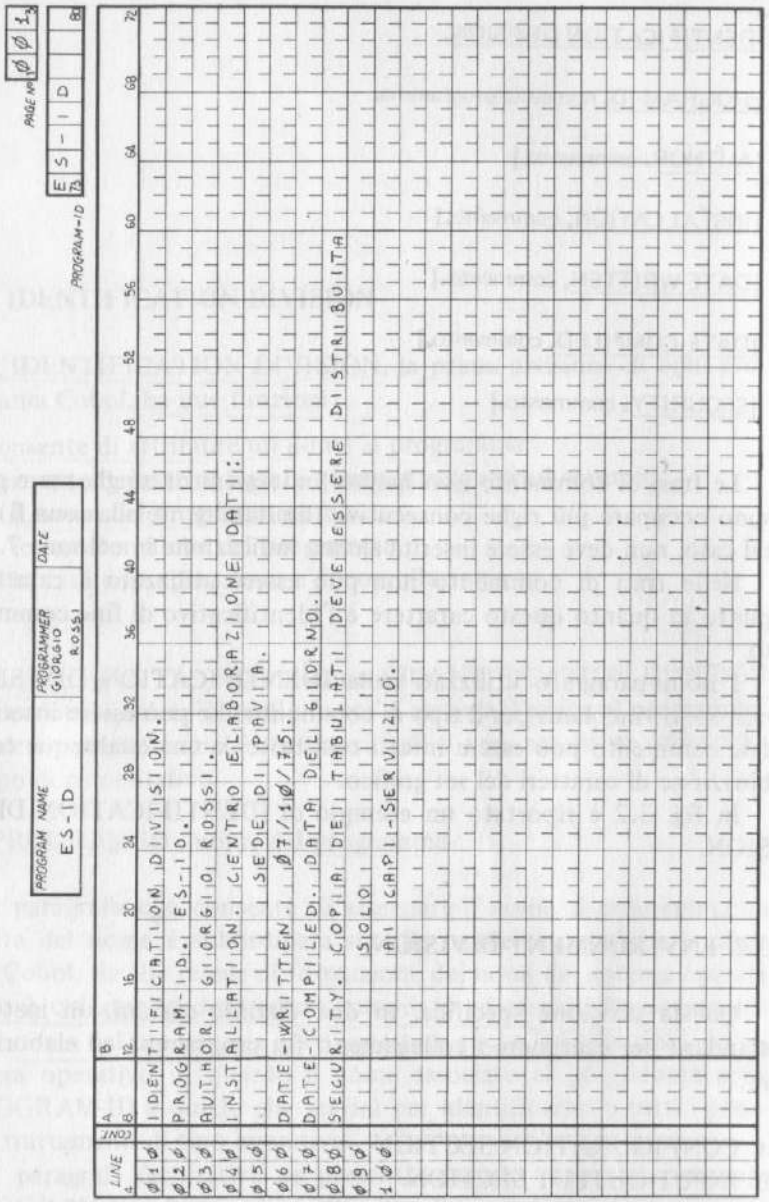


Fig. 3.2

che dell'elaboratore utilizzato in sede di compilazione ed in sede di esecuzione del programma: la presenza di questa doppia descrizione consente di indicare che le due diverse operazioni di compilazione e lancio vengono eseguite su elaboratori differenti, nell'ambito di una stessa linea di prodotti.

I paragrafi relativi sono:

a. SOURCE-COMPUTER,
b. OBJECT-COMPUTER.

La CONFIGURATION SECTION prevede inoltre il paragrafo opzionale, SPECIAL-NAMES, in cui si associano nomi mnemonici ad alcuni dispositivi di macchina (switches) e in cui si caratterizzano alcune funzionalità di stampa (canali loop).

Nella INPUT-OUTPUT SECTION viene precisata l'assegnazione dei flussi alle varie periferiche del sistema, e vengono fornite informazioni circa la gestione e le modalità di elaborazione dei flussi stessi.

Come è facilmente comprensibile, per la sua natura di divisione di collegamento fra l'utente (che vuole risolvere un problema utilizzando un linguaggio evoluto) e l'elaboratore (che deve eseguire il programma in linguaggio macchina, che gli è evidentemente personalizzato per sfruttarne nel modo migliore le caratteristiche funzionali), l'ENVIRONMENT DIVISION può essere poco standard.

3.2.1. *CONFIGURATION SECTION*

Questa sezione è dedicata alla descrizione delle specifiche hardware di elaborazione, infatti vengono indicati l'elaboratore che compila il programma e quello che lo deve eseguire, ed alcune caratteristiche di macchina come gli switches e i dispositivi di movimento carta.

3.2.1.1. *SOURCE-COMPUTER*

SOURCE-COMPUTER. sigla-elaboratore.

Questo paragrafo obbligatorio prevede l'inserimento della sigla dell'elaboratore con cui sarà compilato il programma.

3.2.1.2. *OBJECT-COMPUTER*

OBJECT-COMPUTER. sigla-elaboratore

[MEMORY SIZE intero { WORDS / CHARACTERS / MODULES }]

Questo paragrafo consente di indicare la sigla dell'elaboratore con cui sarà eseguito il "programma oggetto" prodotto dalla compilazione: è evidente che le sigle relative ai paragrafi SOURCE-COMPUTER e OBJECT COMPUTER devono identificare elaboratori appartenenti ad una stessa linea di prodotti in quanto, come è facilmente intuibile, non è possibile compilare un programma su un elaboratore ed eseguirlo su un altro, con logica e caratteristiche diverse.

La clausola opzionale MEMORY SIZE... consente di predisporre il programma oggetto affinché possa essere eseguito su un elaboratore con una "data memoria disponibile".

Questa clausola viene utilizzata quando l'elaboratore che esegue il programma ha dimensioni di memoria diverse da quello che lo compila, inoltre tale clausola ha un'importanza fondamentale in ambiente di multiprogrammazione, ove deve essere previsto di assegnare ad un programma oggetto una determinata porzione di memoria.

Le dimensioni della memoria possono essere espresse in parole di memoria (WORDS), in caratteri (CHARACTERS) oppure in moduli (MODULES).

3.2.1.3. *SPECIAL-NAMES*

In questo paragrafo opzionale, possono venire associati dei nomi mnemonici a particolari dispositivi di macchina (switches e canali loops); tali nomi, che possono essere utilizzati direttamente in PROCEDURE DIVISION, vengono associati secondo il generico formato:

identificativo-dispositivo IS nome-mnemonico

Nel paragrafo SPECIAL-NAMES possono inoltre essere usate le frasi:

a. DECIMAL-POINT IS COMMA,
b. CURRENCY SIGN IS costante.

DECIMAL-POINT IS COMMA

serve per scambiare i ruoli dei simboli virgola"," e punto ".": bisogna ricordare che il Cobol è stato messo a punto negli Usa e quindi prevede il punto, per separare la parte intera dalla parte decimale di un numero, e la virgola, come separatore delle migliaia, dei milioni, ecc...

Lo scambio è relativo, solo a problemi di stampa o più in particolare di edizione; in ogni caso le costanti dirette decimali devono essere scritte utilizzando il punto decimale.

CURRENCY SIGN IS costante

è la frase che permette di sostituire al Simbolo monetario dollaro "$" nell'ambito di alcune particolari maschere di stampa, un altro carattere da scegliersi nel seguente insieme:

E F G H I M N O Q
T U Y J W K = > <

Possiamo osservare che in questo insieme di caratteri non è compreso il carattere "L", che potrebbe essere usato come simbolo monetario, pertanto la CURRENCY SIGN... ha per noi scarso interesse.

Tutte le frasi relative al paragrafo SPECIAL-NAMES devono esse re scritte in zona B senza dover rispettare alcun ordine: il simbolo punto, indicatore della fine del paragrafo, deve essere inserito oltre che dopo il nome paragrafo, una sola volta, alla fine del gruppo di parametri.

Se non interessa citare alcuna delle voci di questo paragrafo si omette anche il nome SPECIAL-NAMES.

3.2.2. *INPUT-OUTPUT SECTION*

Nella INPUT-OUTPUT SECTION, sono inserite tutte le informazioni che permettono di collegare i flussi di elaborazione alle relative periferiche e tutte le informazioni sulla gestione dei flussi (in relazione alle forme di organizzazione e di accesso).

Questa sezione è costituita da due paragrafi:

a. FILE-CONTROL,
b. I-O-CONTROL.

3.2.2.1. *FILE-CONTROL*

Nel paragrafo FILE-CONTROL si provvede ad associare ad ogni flusso la periferica ad esso relativa, citando, per ogni flusso, il nome che servirà per identificarlo nel programma, e, per ogni periferica, la sigla che ne permetterà l'identificazione da parte del compilatore.

I nomi-flusso sono nomi del programmatore, le sigle identificative delle periferiche sono espresse mediante uno dei codici ammessi dal compilatore.

Per quanto riguarda i soli flussi in input, c'è la possibilità di utilizzare un'opzione, di tipo documentativo, per indicare che il flusso (in input) può non essere presente durante l'elaborazione.

Per ciascun flusso elaborato dal programma nel paragrafo FILE-CONTROL dovrà comparire una frase del tipo:

```
SELECT [OPTIONAL] nome-flusso
       ASSIGN TO sigla-unità-periferica.
```

Ciascuna SELECT deve terminare con il carattere punto.

Ricordiamo che l'indicazione documentativa OPTIONAL può essere riferita solo a flussi in input.

3.3.2.2. *I-O-CONTROL*

In questo paragrafo vengono inserite le informazioni relative all'allocazione delle aree record e flusso in comune per più flussi e vengono fornite informazioni circa le modalità di memorizzazione

periodica dello stato del sistema durante l'esecuzione del programma, per poter eseguire le operazioni di ripristino nel caso di interruzioni non previste (dovute ad esempio ad una improvvisa mancanza di alimentazione).

Non entriamo nei particolari della struttura delle frasi che costituiscono l'I-O-CONTROL, in quanto sono molto spesso diverse a seconda del sistema di elaborazione utilizzato, rimandiamo quindi chi voglia approfondire lo studio di questo paragrafo ai manuali Cobol relativi alle diverse linee di elaboratori.

3.2.3. *Tavola di riepilogo*

In fig. 3.3 riportiamo la struttura della ENVIRONMENT DIVISION, completa limitatamente agli aspetti affrontati finora.

Osserviamo che l'INPUT-OUTPUT SECTION è facoltativa, in quanto le specifiche in essa contenute non devono essere fornite nel caso in cui il programma non preveda il trattamento di flussi: è evidente che si tratta di un caso limite in quanto il linguaggio Cobol è stato formulato principalmente per risolvere problemi relativi al trattamento di grandi quantità di dati.

3.3. **Un esempio**

In fig. 3.4 è rappresentato l'organigramma del programma PROG1, che prevede il trattamento di un flusso in input su schede (FL-SK) e di due flussi in output (FL-DS su disco, organizzato sequenzialmente e con accesso sequenziale, e FL-ST su stampante).

Le specifiche relative al problema risolto da PROG1 sono riportate più avanti (4.1.4), per ora ci limitiamo a scrivere le prime due divisioni del programma (fig. 3.5) e a fare alcune osservazioni.

Osserviamo che:

a. L'IDENTIFICATION DIVISION è stata codificata in forma ridotta, in quanto oltre al paragrafo PROGRAM-ID è stato scritto un solo paragrafo facoltativo contenente un breve commento;

Fig. 3.3

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER. sigla-elaboratore-di-compilazione.

OBJECT-COMPUTER. sigla-elaboratore-di-esecuzione

[MEMORY SIZE intero { WORDS / CHARACTERS / MODULES }] .

[SPECIAL-NAMES.
[identificativo-dispositivo IS nome-mnemonico]...
[DECIMAL-POINT IS COMMA]
[CURRENCY SIGN IS costante].]

[INPUT-OUTPUT SECTION.

FILE-CONTROL.
SELECT [OPTIONAL] nome-flusso ASSIGN TO sigla-unità-periferica.
:

[I-O-CONTROL.
: . . . . . .].]

Fig. 3.4

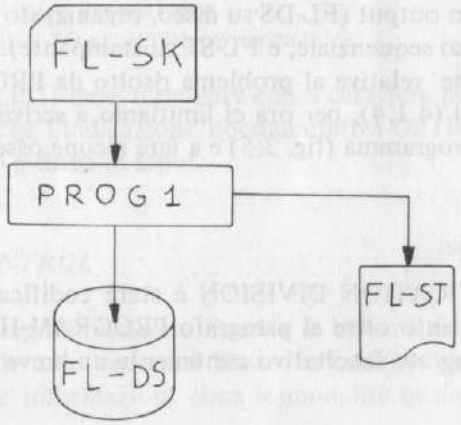
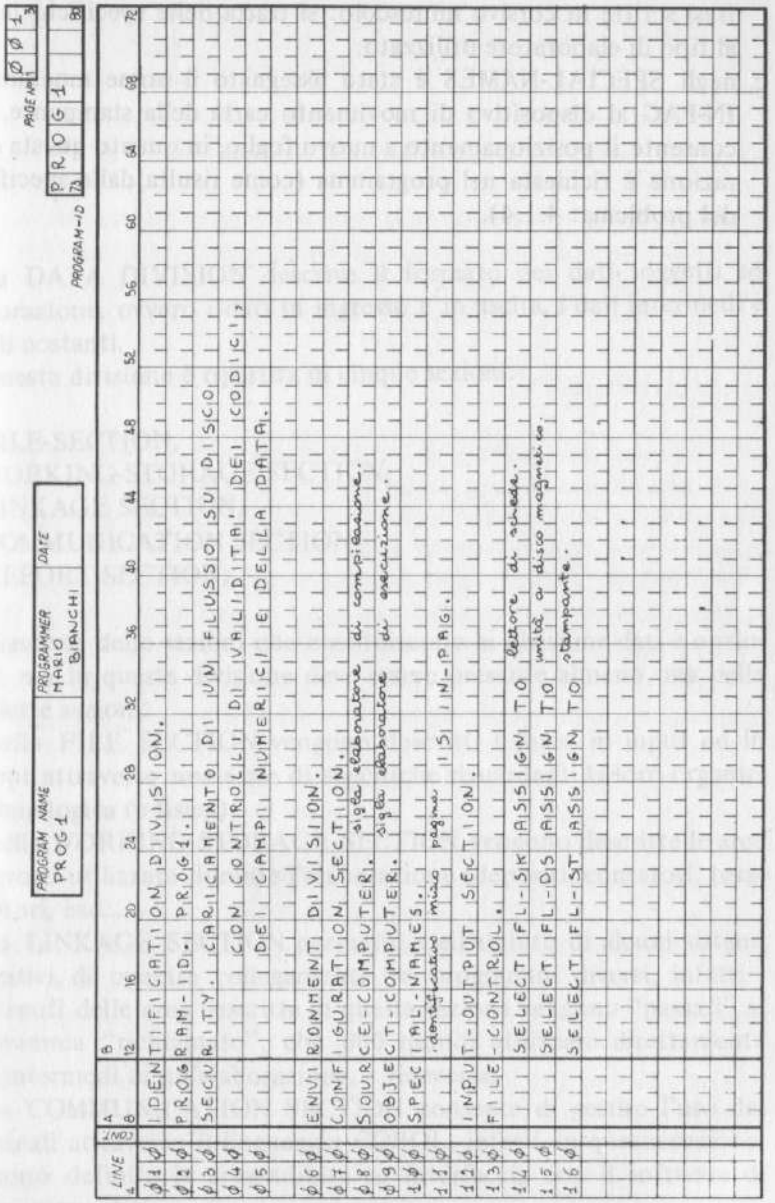


Fig. 3.5



b. in alcuni punti dell'ENVIRONMENT DIVISION compaiono delle frasi scritte in corsivo minuscolo: si tratta delle specifiche legate al tipo di elaboratore utilizzato;
c. negli SPECIAL-NAMES è stato assegnato il nome mnemonico IN-PAG al dispositivo di movimento carta della stampante, che consente il posizionamento a nuovo foglio, in quanto questa operazione è richiesta nel programma (come risulta dalle specifiche del problema: 4.1.4).

4. DEFINIZIONE DEI DATI

La DATA DIVISION descrive il formato dei dati soggetti ad elaborazione, ovvero i dati in ingresso e in uscita, i dati intermedi e quelli costanti.

Questa divisione è ripartita in cinque sezioni:

a. FILE-SECTION,
b. WORKING-STORAGE SECTION,
c. LINKAGE SECTION,
d. COMMUNICATION SECTION,
e. REPORT SECTION.

Ciascuna delle sezioni che costituiscono la divisione dati è opzionale, ma in questa divisione deve essere presente almeno una delle suddette sezioni.

Nella FILE SECTION vengono descritti i flussi in input ed in output attraverso una serie di specifiche riguardanti la loro organizzazione logica (e fisica).

Nella WORKING-STORAGE SECTION vengono descritte le aree di lavoro utilizzate durante l'elaborazione (depositi, contatori, totalizzatori, ecc...).

La LINKAGE SECTION permette, nell'ambito di alcuni sistemi operativi, di operare collegamenti fra programmi diversi, infatti i contenuti delle aree descritte in questa sezione vengono "passati" al programma "richiamato", che può quindi utilizzare direttamente dati intermedi di un'elaborazione, e viceversa.

La COMMUNICATION SECTION consente di gestire l'uso dei terminali attraverso il linguaggio COBOL, infatti, in questa sezione, vengono definite le aree adibite ad interfaccia con il software di trasmissione dati.

La REPORT SECTION serve quando si vogliono utilizzare parti-

colari procedure automatiche per la stampa dei tabulati, infatti, in questa sezione, vengono fornite le specifiche dei tabulati da stampare, precisando il formato delle intestazioni e delle chiusure di tabulato e di pagina e la struttura delle righe dettaglio.

Ci limiteremo, in questo capitolo, a fornire gli elementi essenziali della FILE SECTION e della WORKING-STORAGE SECTION (che saranno ripresi e approfonditi in 6.), alcuni particolari delle altre sezioni saranno affrontati nelle appendici dove tratteremo alcuni aspetti dei moduli dello standard che in questi primi capitoli non sono esaminati.

4.1. FILE SECTION

Un flusso è un insieme di informazioni omogenee dette record logici: nell'ambito di uno stesso flusso i record possono avere tutti la stessa struttura oppure strutture diverse, in questo secondo caso vengono identificati tramite codici record.

In FILE SECTION, per ogni flusso trattato nel programma, vengono fornite una serie di informazioni circa gli elementi organizzativi e strutturali: di ogni flusso sono date informazioni di tipo generale quali i tipi di etichette, il bloccaggio ed i tipi record che lo compongono, ogni record viene descritto in modo particolareggiato precisando la lunghezza dei campi che lo costituiscono e la natura dei dati che ogni campo dovrà contenere.

4.1.1. *Descrizione dei flussi*

La descrizione di ciascun flusso inizia con l'identificatore di livello FD (File Description) scritto all'inizio della zona A, immediatamente seguito, in zona B, dal nome flusso che è stato assegnato dal programmatore nella SELECT relativa (3.2.2.1).

Dopo questa parte iniziale occorre dichiarare se il flusso è etichettato o meno:

```
LABEL { RECORD IS  } { OMITTED  }
      { RECORDS ARE} { STANDARD }
```

Viene utilizzata la clausola OMITTED normalmente per flussi su

schede o per output su stampante e in generale nel caso di flussi privi di etichette ed in tutti quei casi di flussi etichettati ma con etichette non standard (in relazione al sistema operativo utilizzato); negli altri casi viene invece indicato STANDARD.

La clausola LABEL è obbligatoria nell'ambito di ciascuna FD.

La scelta tra la forma singolare e quella plurale è completamente indifferente.

Se il flusso in esame è su supporto magnetico, bisogna fornire delle informazioni circa la dimensione del record fisico (blocco o pagina), tale dimensione può essere espressa in numero di record logici contenuti oppure in numero di caratteri:

```
BLOCK CONTAINS [intero-1 TO] intero-2 { RECORDS   }
                                      { CHARACTERS}
```

in assenza delle parole RECORDS o CHARACTERS la dimensione è espressa automaticamente in caratteri.

La parte opzionale, tra parentesi quadre, è riferita, a scopo puramente documentativo, a blocchi con record di lunghezza variabile: essa può essere utilizzata per indicare l'occupazione minima del blocco.

La clausola BLOCK... non è obbligatoria ed inoltre viene indicata solo quando non c'è una corrispondenza uno-a-uno fra record logico e record fisico, è per questo motivo che nel formato generale della File Description (fig. 4.1) compare tra parentesi quadre.

Osserviamo che la clausola BLOCK... consente di determinare, al momento della compilazione, la dimensione dell'area blocco per quei flussi con bloccaggio maggiore di uno: in questi casi, se la clausola è omessa, il sistema operativo assegna una dimensione standard, con il rischio di sciupare spazio di memoria oppure di avere un'area insufficiente a contenere il blocco.

Di solito la FD di un flusso viene completata dalla clausola documentativa:

```
DATA { RECORD IS  } nome-record-1 [nome-record-2]...
     { RECORDS ARE}
```

questa clausola ha la funzione di elencare i vari tipi di record logici che costituiscono il flusso, tali record devono essere descritti in dettaglio dopo questa clausola.

L'ordine con cui sono elencati i nomi record nella DATA RE-

CORD... non è vincolante per l'ordine di descrizione degli stessi, del resto questa clausola è opzionale.

In fig. 4.1 sono riportati tutti gli elementi che abbiamo presentato della FD di un flusso; come si può osservare due delle tre clausole descrittive che abbiamo introdotto sono opzionali, ma, quando è possibile, è meglio utilizzarle per predisporre un programma sorgente ben documentato.

Osserviamo infine che ogni FD deve terminare con il simbolo punto.

In fig. 4.2 sono riportate le FD dei tre flussi dell'esempio PROG1, introdotto in 3.3 e precisato in 4.1.4.

Fig. 4.1

```
FD nome-flusso
   LABEL { RECORD IS  } { OMITTED  }
         { RECORDS ARE } { STANDARD }
   [BLOCK CONTAINS [intero-1 TO] intero-2 { RECORDS    }]
                                          { CHARACTERS }
   [DATA { RECORD IS  } nome-record-1 [nome-record-2]...].
         { RECORDS ARE }
```

4.1.2. *Descrizione dei record*

Ciascuna File Description è immediatamente seguita dalla descrizione dettagliata dei record logici che costituiscono il flusso in esame: tali record sono quelli (eventualmente) dichiarati nella DATA RECORD... della FD.

L'ordine con cui in una FD vengono descritti i record del flusso è arbitrario e può essere diverso da quello con cui sono stati dichiarati.

L'inizio di ogni descrizione record è identificato dal numero di livello 01 che deve essere sempre attribuito al nome del record e deve essere scritto in zona A.

Ogni record è normalmente suddiviso logicamente (ai fini dell'elaborazione) in campi, a ciascuno dei quali viene assegnato un nome (nome-dato) dal programmatore.

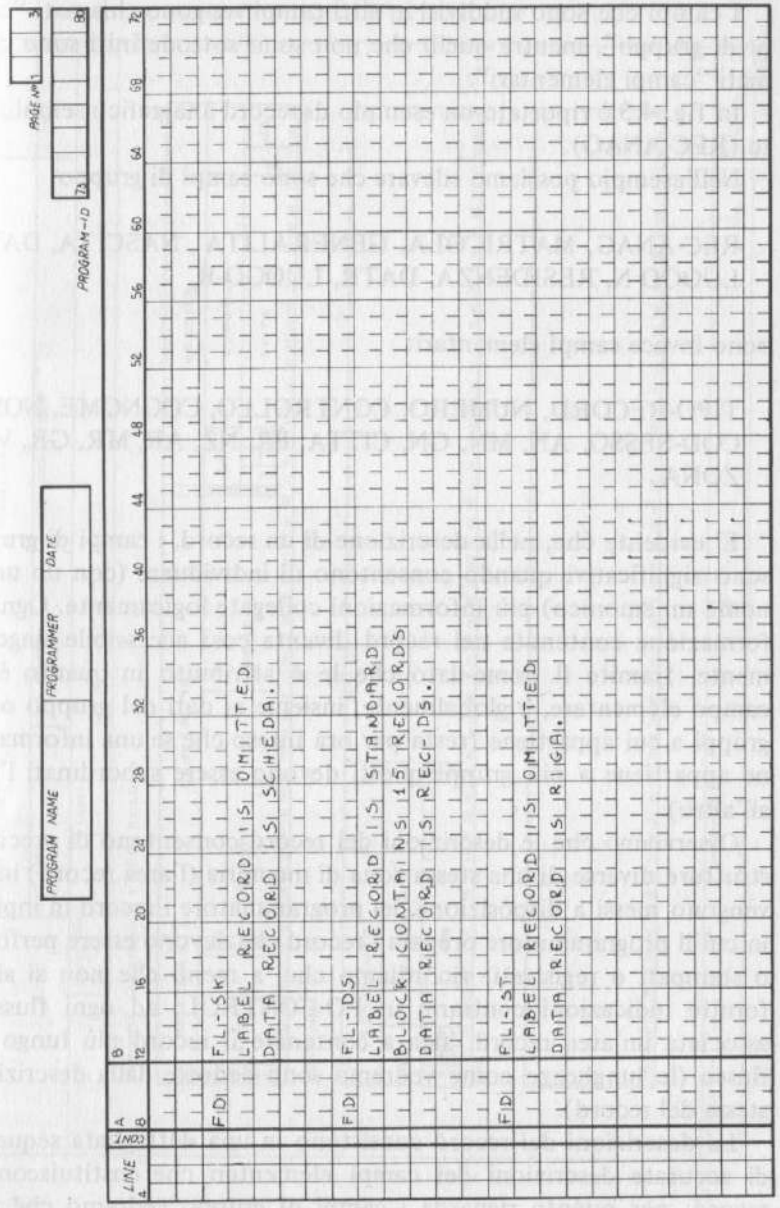


Fig. 4.2

Ogni campo può essere suddiviso a sua volta in altri campi, ecc...

I campi che sono suddivisi in altri campi vengono chiamati "campi di gruppo", mentre quelli che non sono sottodefiniti sono chiamati "campi elementari".

In fig. 4.3 è riportato un esempio di record anagrafico semplificato (REC-ANAG).

Nell'esempio possiamo rilevare che sono campi di gruppo:

REC-ANAG, MATRICOLA, GENERALITA , NASCITA, DATN, LUOGO-N, RESIDENZA, DATR, LUOGO-R,

sono invece campi elementari:

TIPO-RECORD, NUMERO, CONTROLLO, COGNOME, NOME, COD-SESSO, AN, MN, GN, CITTA, PR, NZ, AR, MR, GR, VIA, ZONA.

E' evidente che, nella descrizione di un record, i campi di gruppo sono significativi quando consentono di individuare (con un unico nome mnemonico) più informazioni collegate logicamente. Ogni informazione contenuta nel record diventa così accessibile singolarmente, tramite il nome-dato che le è attribuito in quanto è un campo elementare, e globalmente, insieme ai dati del gruppo o dei gruppi a cui appartiene (resta per ora inteso che se una informazione appartiene a più gruppi questi devono essere subordinati l'uno all'altro).

Osserviamo che le descrizioni dei record consentono di precisare strutture diverse di una stessa zona di memoria (l'area record) in cui vengono messi a disposizione del programmatore i record in input o in cui il programmatore prepara i record che devono essere perforati o stampati o registrati: ricordiamo che, a meno che non si siano fornite indicazioni contrarie in I-O-CONTROL, ad ogni flusso è associata un'area record, atta a contenere il record più lungo del flusso (le lunghezze come vedremo sono dedotte dalla descrizione stessa del record).

Le descrizioni dei record consistono in una dettagliata sequenza di accurate descrizioni dei campi elementari che costituiscono il record; per quanto riguarda i campi di gruppo vedremo che non necessita descrizione, in quanto la loro struttura è strettamente di-

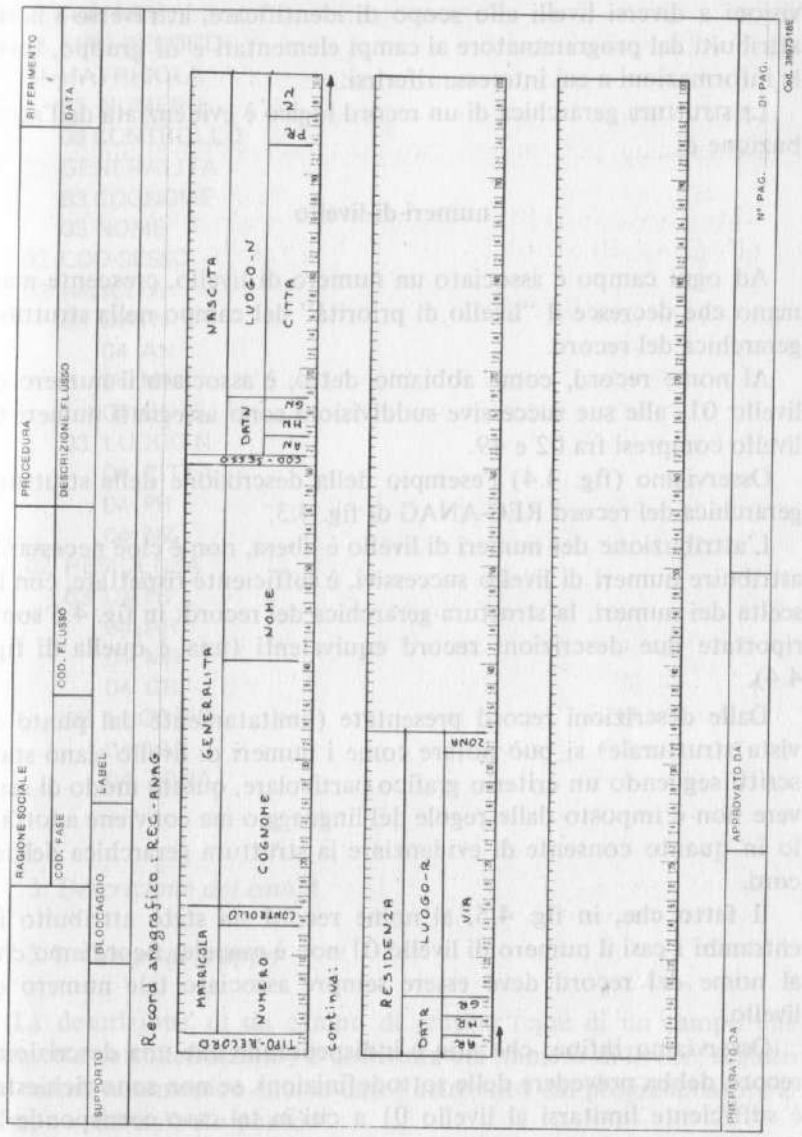


Fig. 4.3

pendente da quella dei campi elementari che li compongono.

Un record logico è molto spesso organizzato su successive suddivisioni a diversi livelli allo scopo di identificare, attraverso i nomi attribuiti dal programmatore ai campi elementari e di gruppo, tutte le informazioni a cui interessa riferirsi.

La struttura gerarchica di un record logico è evidenziata dall'attribuzione di

numeri-di-livello

Ad ogni campo è associato un numero di livello, crescente man mano che decresce il "livello di priorità" del campo nella struttura gerarchica del record.

Al nome record, come abbiamo detto, è associato il numero di livello 01, alle sue successive suddivisioni sono assegnati numeri di livello compresi fra 02 e 49.

Osserviamo (fig. 4.4) l'esempio della descrizione della struttura gerarchica del record REC-ANAG di fig. 4.3.

L'attribuzione dei numeri di livello è libera, non è cioè necessario attribuire numeri di livello successivi, è sufficiente rispettare, con la scelta dei numeri, la struttura gerarchica del record: in fig. 4.5 sono riportate due descrizioni record equivalenti (una è quella di fig. 4.4).

Dalle descrizioni record presentate (limitatamente dal punto di vista strutturale) si può notare come i numeri di livello siano stati scritti seguendo un criterio grafico particolare, questo modo di scrivere non è imposto dalle regole del linguaggio ma conviene adottarlo in quanto consente di evidenziare la struttura gerarchica del record.

Il fatto che, in fig. 4.5, al nome record sia stato attribuito in entrambi i casi il numero di livello 01 non è casuale: ricordiamo che al nome del record deve essere sempre associato tale numero di livello.

Osserviamo infine, che non è indispensabile che una descrizione record debba prevedere delle sottodefinizioni, se non sono richieste, è sufficiente limitarsi al livello 01 a cui in tal caso corrisponde la descrizione di un campo elementare.

Fig. 4.4

```
01 REC-ANAG
   02 TIPO-RECORD
   02 MATRICOLA
      03 NUMERO
      03 CONTROLLO
   02 GENERALITA
      03 COGNOME
      03 NOME
   02 COD-SESSO
   02 NASCITA
      03 DATN
         04 AN
         04 MN
         04 GN
      03 LUOGO-N
         04 CITTA
         04 PR
         04 NZ
   02 RESIDENZA
      03 DATR
         04 AR
         04 MR
         04 GR
      03 LUOGO-R
         04 VIA
         04 ZONA
```

4.1.3. *Descrizione dei campi*

4.1.3.1. *Campi di gruppo*

La descrizione di un campo di gruppo (cioè di un campo che ammette sottodefinizioni) è costituita dal numero di livello, seguito dal nome mnemonico (nome-dato) attribuito dal programmatore al campo e dal simbolo punto:

numero-di-livello nome-dato.

Un campo di gruppo è automaticamente considerato alfanumeri-

Fig. 4.5

```
01 REC-ANAG
   02 TIPO-RECORD
   02 MATRICOLA
      03 NUMERO
      03 CONTROLLO
   02 GENERALITA
      03 COGNOME
      03 NOME
   02 COD-SESSO
   02 NASCITA
      03 DATN
         04 AN
         04 MN
         04 GN
      03 LUOGO-N
         04 CITTA
         04 PR
         04 NZ
   02 RESIDENZA
      03 DATR
         04 AR
         04 MR
         04 GR
      03 LUOGO-R
         04 VIA
         04 ZONA
```

```
01 REC-ANAG
   05 TIPO-RECORD
   05 MATRICOLA
      09 NUMERO
      09 CONTROLLO
   05 GENERALITA
      06 COGNOME
      06 NOME
   05 COD-SESSO
   05 NASCITA
      12 DATN
         15 AN
         15 MN
         15 GN
      12 LUOGO-N
         23 CITTA
         23 PR
         23 NZ
   05 RESIDENZA
      10 DATR
         20 AR
         20 MR
         20 GR
      10 LUOGO-R
         11 VIA
         11 ZONA
```

co e la sua lunghezza è pari alla somma delle lunghezze dei sottocampi che lo compongono.

4.1.3.2. *Campi elementari*

Anche la descrizione di un campo elementare inizia con il numero di livello seguito dal nome con cui il campo sarà individuato per tutto il resto del programma.

La descrizione deve essere completata con la precisazione della lunghezza del campo e del tipo di dati che dovrà contenere (categoria).

La clausola descrittiva, che consente l'assegnazione della lunghezza e della categoria, è:

$$\left\{\begin{matrix}\underline{\text{PICTURE}}\\ \underline{\text{PIC}}\end{matrix}\right\} \text{IS stringa-di-caratteri}$$

La parola PICTURE/PIC sta per "immagine", la stringa di caratteri è una sequenza di caratteri, con significato speciale, che indica il formato del campo.

Il formato della descrizione di un campo elementare è quindi:

$$\text{numero-di-livello nome-dato} \left\{\begin{matrix}\underline{\text{PICTURE}}\\ \underline{\text{PIC}}\end{matrix}\right\} \text{IS stringa-di-caratteri.}$$

4.1.3.3. *Categorie*

Sono diversi i caratteri che possono essere utilizzati nella composizione di una stringa, per il momento ci limiteremo ad esaminare la funzione solo di qualcuno di essi.

Il Cobol inquadra i campi in categorie in base al tipo di informazioni contenute: le stringhe di caratteri, utilizzate nella descrizione di un campo elementare, consentono di individuare le categorie di appartenenza in modo da fissarne le modalità di trattamento nelle istruzioni che li gestiscono.

Un dato è di categoria alfanumerica se è costituito da una qualunque combinazione di caratteri del set grafico, è di categoria numerica se è costituito da cifre (ignoriamo per il momento l'eventuale presenza del segno), è di categoria alfabetica se è costituito da lettere dell'alfabeto inglese e/o dal simbolo spazio.

E' evidente che un dato di categoria numerica o alfabetica è anche di categoria alfanumerica; è importante rilevare che un numero, quando deve essere trattato come tale, e non semplicemente come sequenza di caratteri, deve essere inquadrato nella categoria numerica, e non in quella alfanumerica.

In Cobol un campo deve essere caratterizzato di categoria alfanumerica, numerica o alfabetica in relazione al tipo di dati che deve contenere e alle modalità di trattamento previste.

La lettera X contraddistingue una posizione di categoria alfanu-

merica, ovvero una posizione che può essere occupata da un carattere qualunque del set grafico.

La cifra 9 individua una posizione di categoria numerica, che potrà quindi essere occupata da una cifra.

La lettera A individua una posizione di categoria alfabetica, che potrà contenere il simbolo spazio oppure una lettera dell'alfabeto inglese.

I tre simboli:

X 9 A

vengono utilizzati per formare le stringhe di caratteri che consentono di descrivere i campi elementari: il numero dei caratteri che costituiscono la stringa determina la lunghezza del campo.

Se, ad esempio, vogliamo descrivere un campo atto a contenere una data nella forma giorno/mese (3 caratteri alfabetici)/anno, scriveremo:

```
numero-di-livello CAMPO-DATA PIC 99AAA9999.
```

Se invece vogliamo descrivere un campo atto a contenere un codice di avviamento postale scriveremo:

```
numero-di-livello CAP PIC 99999.
```

Se infine vogliamo descrivere un campo atto a contenere la descrizione di un articolo (massimo lunga 15 caratteri del set grafico) scriveremo:

```
numero-di-livello DESCR PIC XXXXXXXXXXXXXXX.
```

Invece di scrivere una sequenza di "n" caratteri uguali (per la costituzione di una stringa rappresentativa di una PICTURE) è conveniente utilizzare una scrittura sintetica del tipo:

carattere (n)

La scrittura sintetica, non solo è comoda, in quando consente di evitare di "contare" gli "n" caratteri da scrivere nella stringa, ma è talvolta indispensabile, infatti anche la stringa è soggetta alle usuali limitazioni imposte dal Cobol sulla lunghezza delle parole e pertanto non può essere più lunga di trenta caratteri.

Esempi:

9(7)	equivale a 9999999
X(5)9(6)	equivale a XXXXX999999
A(6)XX9(5)	equivale a AAAAAAXX99999

4.1.3.4. *FILLER*

Quando un campo è presente fisicamente in un record, ma non viene utilizzato in PROCEDURE DIVISION, perchè non è interessato direttamente dall'elaborazione, deve essere ugualmente indicato per rispettare una corretta definizione del record; in questo caso non è però necessario attribuire al campo un nome mnemonico, ma è sufficiente utilizzare la parola FILLER, che il Cobol mette a disposizione per circostanze di questo tipo.

Il nome Cobol FILLER consente quindi di garantire una corretta descrizione dei record, evitando il proliferare dei nomi mnemonici, a vantaggio della chiarezza del programma.

Un campo FILLER è di solito definito di categoria alfanumerica, questo tipo di definizione è coerente con la natura del campo.

4.1.4. *L'esempio PROG1*

Presentiamo ora il problema che il programma PROG1 (di cui abbiamo presentato l'organigramma in fig. 3.4) si propone di risolvere.

Si vuole caricare su disco magnetico (in un flusso FL-DS) il contenuto di record logici di 46 caratteri, perforati su schede (che costituiscono il flusso FL-SK), di tipo TSK = "C": in fig. 4.6 sono riportati i tracciati record dei due flussi.

Il caricamento deve essere effettuato per i record che superano una serie di controlli, quelli in cui vengono riscontrati errori devono essere stampati con l'indicazione del codice d'errore come dal tracciato stampa di fig. 4.7 (la stampa deve essere eseguita con un massimo di 25 righe per foglio ad interlinea doppia).

Fig. 4.6

	RAGIONE SOCIALE	PROCEDURA	RIFERIMENTO
	COD. FASE / COD. FLUSSO	DESCRIZIONE FLUSSO	DATA
SUPPORTO	BLOCCAGGIO / LABEL		

FLUSSO FL-SK : record SCHEDA (TSK = "C")

TSK	CH-ART		CH-CLI		DESMOV	DATA-MOV			QTA
	ART	CA	CLI	CC		ANNO	MESE	GIORNO	

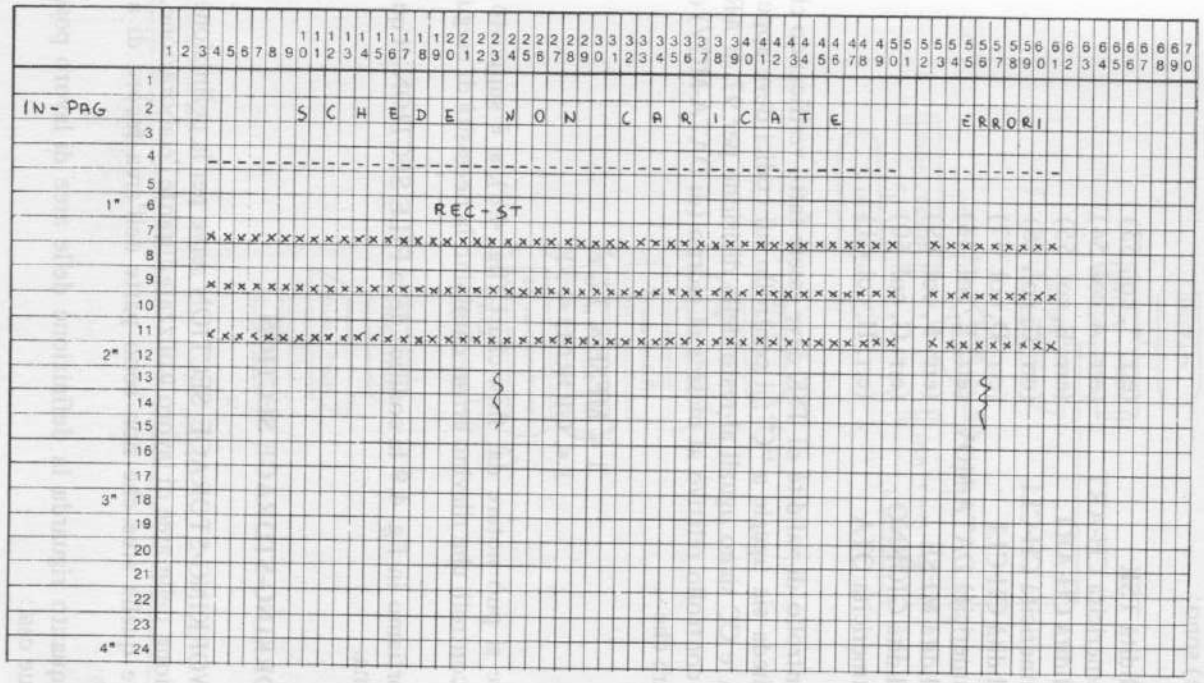
FLUSSO FL-DS : record REC-DS

REC-DS

PREPARATO DA	APPROVATO DA:	N° PAG. DI PAG.

Cod. 3897318E

Fig. 4.7



I tipi di controllo che devono essere eseguiti (e i relativi codici d'errore) sono:

validità TSK	(err: *, col. 53)
numericità CH-ART	(err: A, col. 54)
validità CH-ART	(err: B, col. 55)
numericità CH-CLI	(err: C, col. 56)
validità CH-CLI	(err: D, col. 57)
numericità DATA-MOV	(err: E, col. 58)
validità MESE	(err: F, col. 59)
validità GIORNO	(err: G, col. 60)
numericità QTA	(err: H, col. 61)

Il controllo di validità di TSK deve essere fatto verificando che il tipo scheda sia uguale a "C"; il controllo sui codici deve appurare che CA e CC siano uguali al resto della divisione per 97 di ART e CLI; il controllo relativo al mese e al giorno (di DATA-MOV) deve verificare che

$$1 \leqslant \text{MESE} \leqslant 12 \text{ e}$$
$$1 \leqslant \text{GIORNO} \leqslant 31.$$

Come si può notare dal flow-chart (fig. 4.8), si è supposto che alcuni controlli che rilevino errori eliminino la necessità di eseguirne altri.

Riportiamo in fig. 4.9 la codifica della FILE SECTION di questo problema.

4.2. WORKING-STORAGE SECTION

La WORKING-STORAGE SECTION serve per la definizione e la descrizione delle aree di lavoro utilizzate durante l'elaborazione, cioè di tutte quelle aree che non sono parte dell'area record di alcun flusso.

Per quanto riguarda la definizione delle aree di lavoro possono darsi due casi:

a. tali campi interessano in quanto campi elementari, logicamente e

Fig. 4.8a

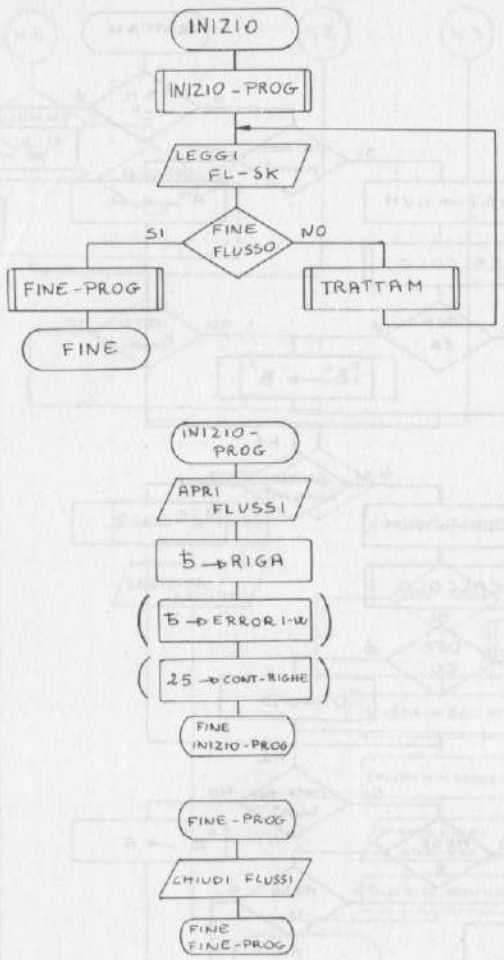


Fig. 4.8b

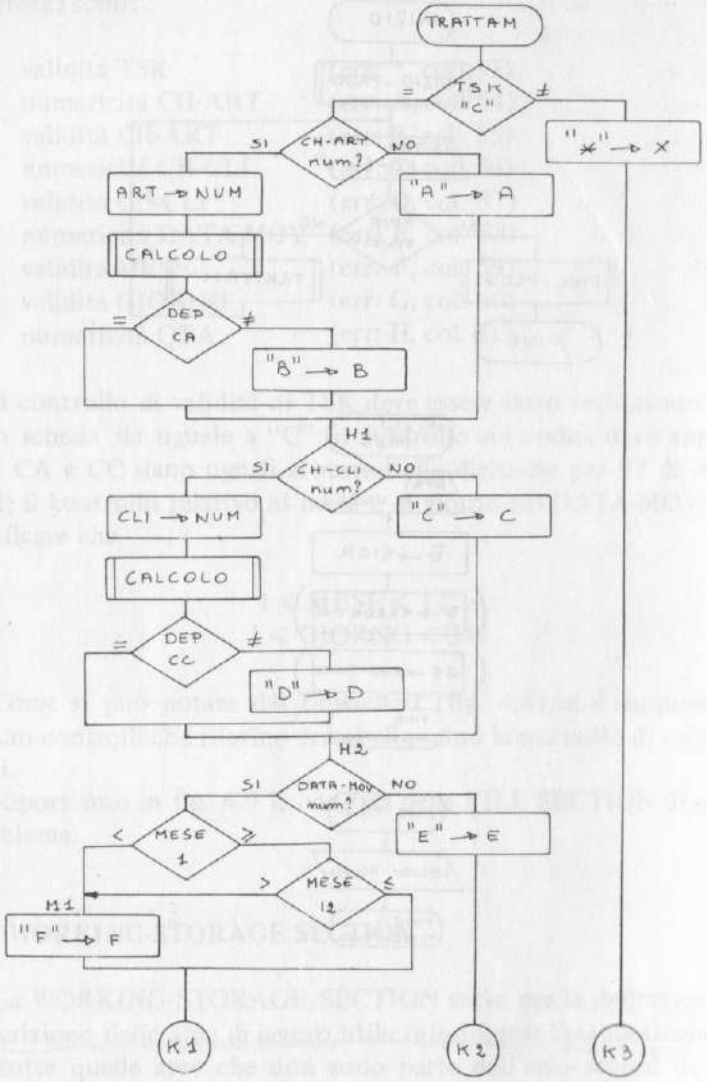


Fig. 4.8c

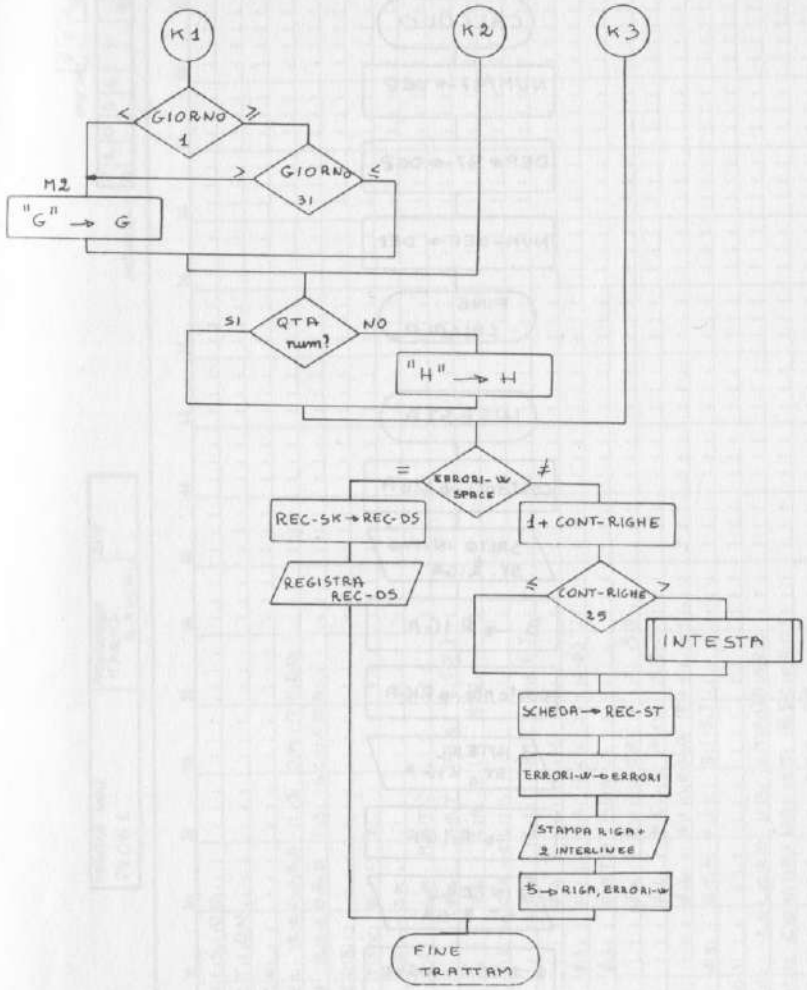
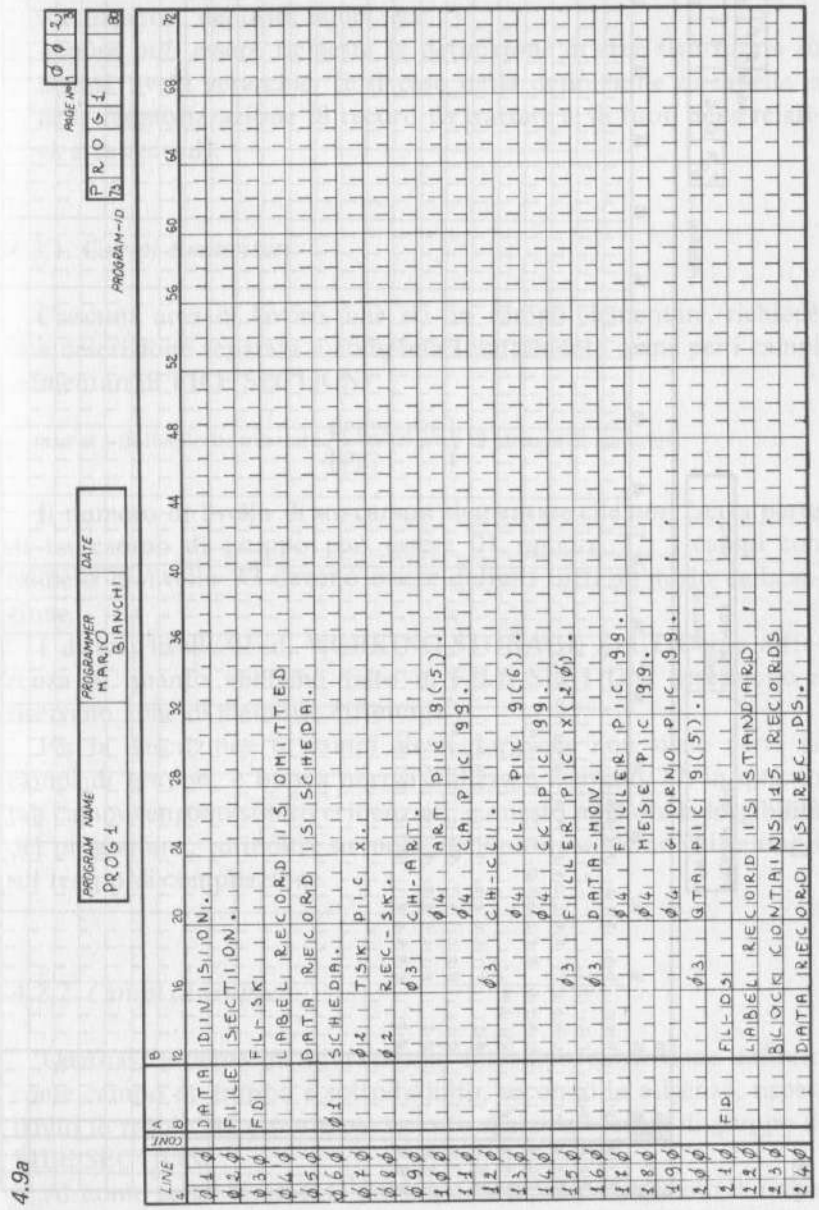


Fig. 4.8d

Fig. 4.9a



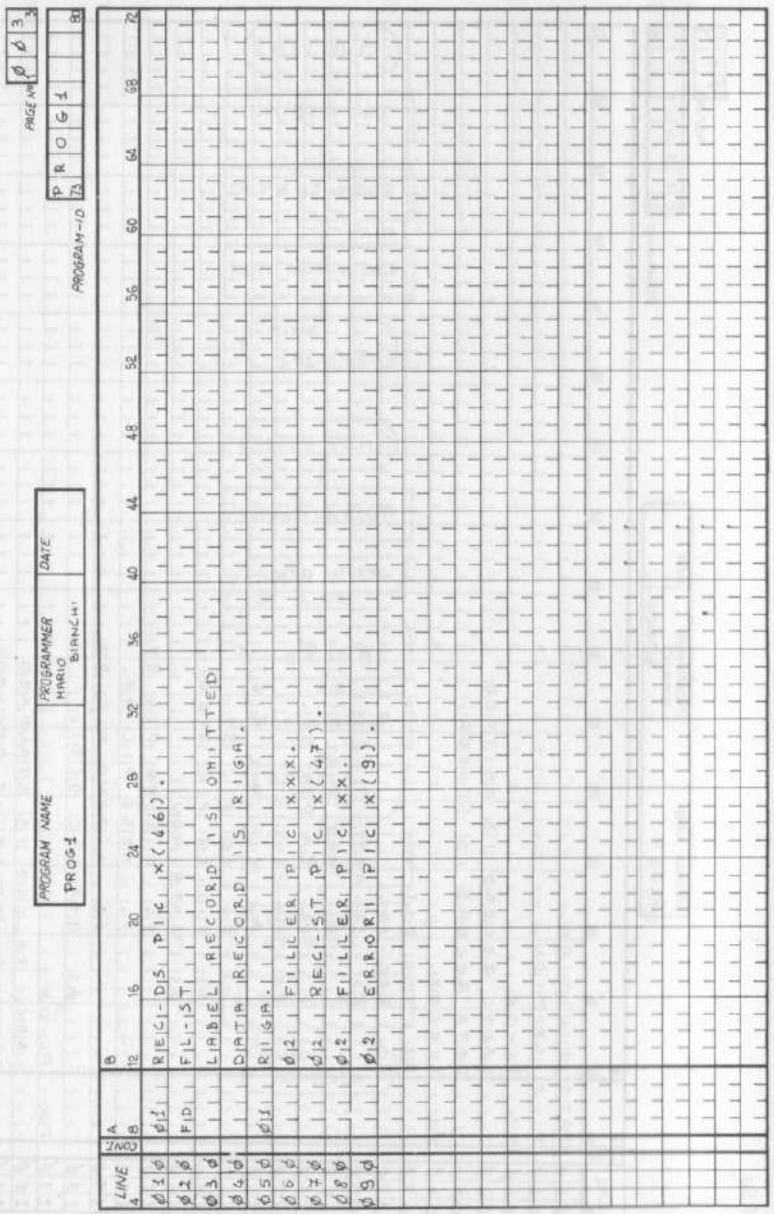


Fig. 4.9b

strutturalmente non collegati fra loro (ad esempio: contatori, totalizzatori, depositi, indici, ecc...)

b. oppure può essere richiesta la definizione di aree strutturate su diversi livelli gerarchici (è il caso della definizione di tabelle o della memorizzazione di record da trattare al di fuori della relativa area record).

4.2.1. *Campi elementari*

Ciascuna area di lavoro, che sia un campo elementare, richiede una descrizione separata e completa (esattamente come per i campi elementari di FILE SECTION):

numero-di-livello nome-dato { PICTURE / PIC } IS stringa-di-caratteri.

Il numero di livello di un campo elementare che non faccia parte di un campo di gruppo può essere 01 oppure 77: i campi con numero di livello 77 devono essere definiti tutti all'inizio della sezione.

I diversi livelli 01 di WORKING-STORAGE SECTION, a differenza di quanto abbiamo visto in FILE SECTION, descrivono e riservano zone di memoria differenti.

Per la descrizione di campi elementari, che non siano parte di campi di gruppo, è buona norma utilizzare il livello 77, in quanto tali campi vengono subito evidenziati, e questo migliora la leggibilità del programma, ed inoltre su molti elaboratori si hanno dei vantaggi sul tempo di compilazione.

4.2.2. *Campi di gruppo*

Qualsiasi insieme di informazioni interdipendenti viene definito come campo di gruppo e sottodefinito, secondo le esigenze, rispettando le regole già esposte per quanto riguarda i campi di gruppo di FILE SECTION.

Al nome gerarchicamente più importante viene associato il numero di livello 01: è importante ricordare che in WORKING-STORA-

GE SECTION diversi livelli 01 fanno riferimento a zone di memoria diverse; vedremo più avanti (6.4.1.) le modalità per definire, secondo formati diversi, una stessa area di lavoro.

4.2.3. *Clausola VALUE*

La clausola VALUE è una clausola, opzionale, per la descrizione dei campi elementari e di gruppo: ha la funzione di inizializzare, con opportuni valori, le aree di lavoro sin dal momento della compilazione del programma.

Il contenuto iniziale di un'area di lavoro non inizializzata tramite la clausola VALUE è, al momento del lancio del programma, normalmente imprevedibile.

Il formato della clausola è:

VALUE IS { costante diretta / costante figurativa }

La costante diretta o la costante figurativa rappresentano il contenuto iniziale dell'area di lavoro a cui la clausola è associata.

Se il campo a cui la VALUE si riferisce è elementare la clausola deve essere inserita dopo la stringa di caratteri che determina la lunghezza e la categoria del campo; se il campo a cui si riferisce è di gruppo la VALUE deve essere scritta dopo il nome del campo.

La costante attribuita come valore ad una data area di lavoro, che sia un campo elementare, deve essere compatibile con il tipo di PICTURE specificata per quell'area: la categoria deve essere la stessa, se vi sono delle differenze per quanto riguarda la lunghezza ci potranno essere dei troncamenti, se il campo è più corto, oppure degli inserimenti, se il campo è più lungo.

Se il campo è di categoria numerica ed è più corto del valore che la VALUE gli assegna si perdono le cifre più significative del numero, se è più lungo vengono aggiunti zeri non significativi (a sinistra).

Se il campo è di categoria alfabetica oppure alfanumerica ed è più corto del valore che gli si vuole attribuire ci sono dei troncamenti a destra, se invece è più lungo vengono aggiunti spazi a destra.

Quando la clausola VALUE è attribuita ad un campo di gruppo è sufficiente che il literal (o la costante figurativa) sia di categoria alfanumerica, ma se si vuole sfruttare l'assegnazione per attribuire

un valore ai singoli sottocampi per trattarli singolarmente con un certo valore iniziale, bisogna che vi sia compatibilità nell'ambito delle componenti elementari. Se ad esempio facciamo la seguente assegnazione:

```
01 CAMPO VALUE IS "ABC123".
   02 C1 PIC XXX.
   02 C2 PIC 999.
```

operiamo in modo corretto ed il campo C2 è trattabile, dal punto di vista aritmetico, fin dall'inizio del programma; se invece scriviamo:

```
01 CAMPO VALUE IS "ABCDEF".
   02 C1 PIC XXX.
   02 C2 PIC 999.
```

abbiamo ancora un formato corretto ma non possiamo eseguire direttamente operazioni aritmetiche su C2, i risultati sarebbero imprevedibili (non attendibili).

Nel caso in cui la clausola VALUE sia riferita ad un campo di gruppo non è più possibile attribuirla ad alcuno dei campi elementari componenti.

Osserviamo che la clausola VALUE, che permette di associare ai campi di WORKING un ben preciso contenuto iniziale, può essere associata anche a descrizioni di campi di FILE SECTION, ma solo a scopo documentativo, al fine di indicare quale sarà il contenuto di tali campi.

4.2.4. *L'esempio PROG1*

Codifichiamo ora (fig. 4.10) la WORKING-STORAGE SECTION dell'esempio PROG1 introdotto in 4.1.4.

Osserviamo che la "VALUE IS 25", attribuita al contatore righe (CONT-RIGHE), consente di eliminare l'istruzione di trasferimento: 25→CONT-RIGHE dalla codifica della PROCEDURE DIVISION; lo stesso vale per il trasferimento degli spazi in ERRORI-W.

Le due aree di lavoro DEP e NUM sono state scelte di lunghezza pari a 6 caratteri, in quanto devono essere utilizzate per il controllo di validità di entrambi i codici e la parte da analizzare del più lungo è di sei caratteri.

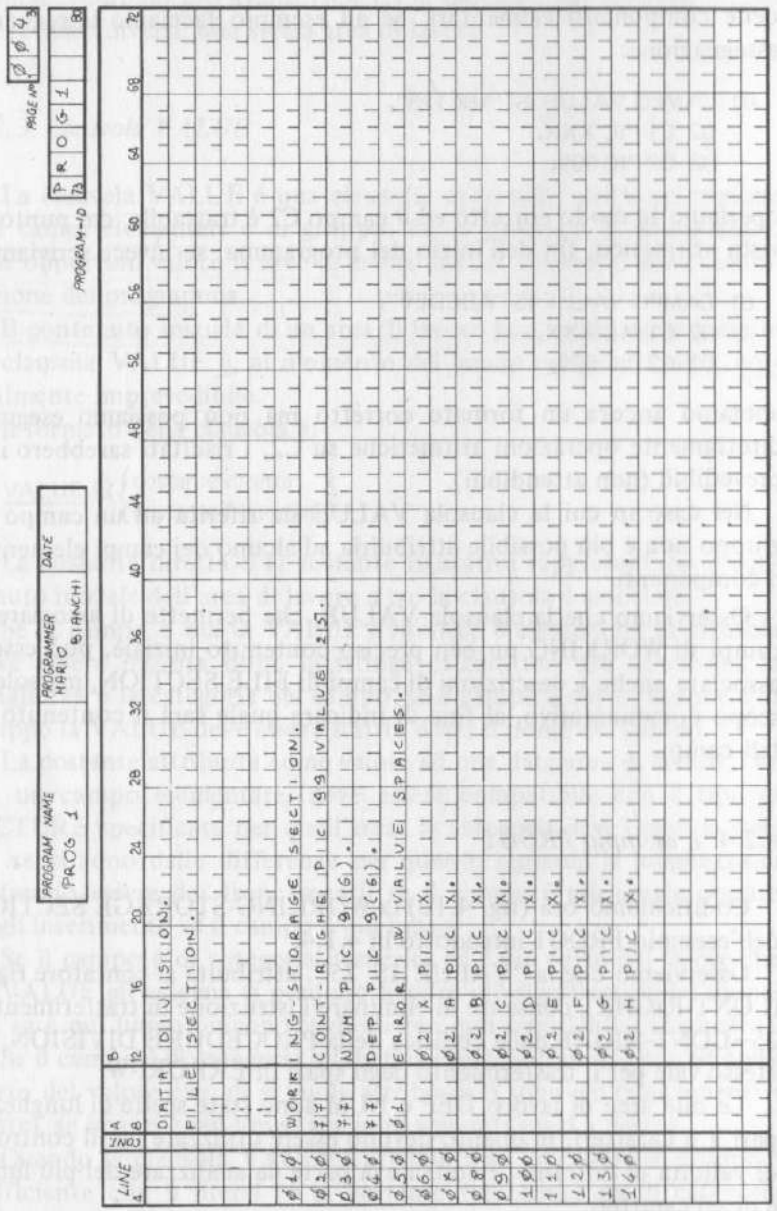


Fig. 4.10

5. TRATTAMENTO DEI DATI

In questo capitolo sono illustrate le istruzioni di PROCEDURE DIVISION che consentono di scrivere un semplice programma Cobol.

I formati che presentiamo spesso sono ridotti rispetto allo standard: verranno ripresi e approfonditi più avanti.

5.1. Struttura della PROCEDURE DIVISION

La PROCEDURE DIVISION contiene tutte le proposizioni necessarie per descrivere le elaborazioni previste dal programma.

Questa divisione può essere ripartita in sezioni a loro volta suddivise in paragrafi, ogni paragrafo può contenere un numero variabile di istruzioni.

Non è obbligatorio dividere la PROCEDURE DIVISION in sezioni, ma, se vengono specificate, devono essere individuate da un nome scelto dal programmatore (nome-sezione) seguito da uno spazio, dalla parola Cobol SECTION e da un punto:

nome-sezione SECTION.

Questa definizione della sezione deve partire da colonna 8 e deve essere la sola nell'ambito di una riga.

La fine di una sezione coincide con l'inizio di una nuova sezione o con la fine del programma; l'ultima frase di ogni sezione deve terminare con un punto.

Le sezioni che compongono la divisione procedura (o direttamente la divisione stessa) possono essere divise in paragrafi ciascuno dei quali è identificato, a partire da colonna 8, da un nome del programmatore (nome-paragrafo) seguito da un punto e da almeno uno spazio:

nome-paragrafo.

In ogni paragrafo può esserci un numero qualunque di istruzioni che possono essere scritte anche a partire dalla riga che contiene il nome-paragrafo.

La fine di un paragrafo è individuata dall'inizio di un nuovo paragrafo o di una nuova sezione oppure dalla fine del programma; l'ultima istruzione di un paragrafo deve terminare con un punto.

Questa organizzazione della PROCEDURE DIVISION non esaurisce la sua funzione dal punto di vista formale ma ha interesse in relazione alle istruzioni di salto e qualora si vogliano richiamare sottoprogrammi interni da punti diversi del programma.

Nelle figg. 5.1, 5.2 e 5.3 sono rappresentate alcune possibili suddivisioni della PROCEDURE DIVISION (osserviamo che alcuni compilatori danno delle segnalazioni d'errore quando al primo paragrafo di una sezione non è stato assegnato un nome).

Fig. 5.1

```
PROCEDURE DIVISION.
nome-sezione SECTION.
.....................
..................... .
nome-sezione SECTION.
.....................
.....................
ecc.................
```

Fig. 5.2

```
PROCEDURE DIVISION.
.....................
..................... .
nome-paragrafo.
.....................
.....................
..................... .
nome-paragrafo. ............
.....................
..................... .
ecc.................
```


Fig. 5.3

```
PROCEDURE DIVISION.
nome-sezione SECTION.
nome-paragrafo.
.....................
..................... .
nome-paragrafo. ............
.....................
..................... .
nome-sezione SECTION.
nome-paragrafo. ............
.....................
..................... .
nome-paragrafo.
.....................
..................... .
ecc.................
```

5.2. Istruzioni di entrata e uscita

5.2.1. *I verbi ACCEPT e DISPLAY*

I verbi ACCEPT e DISPLAY sono utilizzati per comandare l'ingresso e l'uscita di dati non organizzati in flussi (è il caso di modeste quantità di dati da elaborare oppure di pochi risultati in output, oppure il caso di stampa di messaggi).

Normalmente utilizziamo il verbo ACCEPT per introdurre parametri di inizializzazione (quali le costanti che variano da elaborazione ad elaborazione, la data del giorno, ecc...).

Il formato del verbo ACCEPT è:

$$\underline{\text{ACCEPT}}\ \text{nome-dato}\ \left[\underline{\text{FROM}}\ \left\{\begin{array}{l}\underline{\text{DATE}}\\ \underline{\text{DAY}}\\ \underline{\text{TIME}}\end{array}\right\}\right]$$

Il formato senza la clausola FROM consente di caricare dei valori, nella zona di memoria individuata col nome simbolico nome-dato, utilizzando l'unità periferica predisposta a tale scopo dal software (normalmente si tratta di console o di lettore di schede).

Se viene utilizzata la clausola FROM il caricamento di nome-dato è effettuato prelevando il valore dei campi di nome DATE (che contiene la data nella forma AAMMGG), oppure DAY (che contiene la data nella forma AAGGG), oppure TIME (che contiene il valore di un temporizzatore che viene incrementato ogni centesimo di secondo). Quando l'elaboratore che si utilizza prevede la possibilità di utilizzare questi campi, si possono usare direttamente, senza doverli definire in WORKING-STORAGE SECTION, in quanto le parole DATE, DAY e TIME sono riservate.

I trasferimenti di valori in nome-dato, sia vengano effettuati da campi di memoria, sia da periferica, sono soggetti alle usuali regole di trasferimento (5.3), pertanto le modalità dipendono dalla natura del campo ricevente.

Esempio:

```
WORKING-STORAGE SECTION.
01    DEP-DATA.
      02 AA PIC 99.
      02 MM PIC 99.
      02 GG PIC 99.
```

Scrivendo

```
ACCEPT DEP-DATA
```

l'area di lavoro DEP-DATA viene inizializzata con valori introdotti da lettore di schede o da console.

Scrivendo invece

```
ACCEPT DEP-DATA FROM DATE
```

il campo DEP-DATA viene inizializzato con il valore del campo di sistema di nome DATE.

Il verbo DISPLAY è di solito destinato a visualizzare messaggi che, durante l'elaborazione, servono per evidenziare particolari situazioni previste dal programmatore.

Il suo formato è

$$\underline{\text{DISPLAY}} \left\{ \begin{matrix} \text{nome-dato}_1 \\ \text{costante}_1 \end{matrix} \right\} \left[\begin{matrix} \text{nome-dato}_2 \\ \text{costante}_2 \end{matrix} \right] \ldots$$

Il numero degli elementi che si possono visualizzare (normalmente su console o su stampante) non solo è legato, dal punto di vista software, al tipo di elaboratore che si sta utilizzando, ma anche, se si vuole tener conto del formato di output, alla dimensione fisica della riga di stampa.

Esempio:

Supponiamo che le aree:

```
01 CODICE      PIC X(5).
01 DEP-LUOGO PIC X(10).
01 TOTALE      PIC 9(6).
```

contengano rispettivamente i valori:

3574N, PIACENZAbb, 586500

l'istruzione:

```
DISPLAY "ERRORE:" CODICE DEP-LUOGO TOTALE
```

produce in output la riga:

ERRORE:3574NPIACENZAbb586500

Se si vogliono inserire degli spazi per meglio evidenziare il contenuto del messaggio, bisogna introdurre nell'istruzione delle costanti dirette (uguali ad uno o più spazi):

```
DISPLAY "ERRORE:b" CODICE "b" DEP-LUOGO "b" TOTALE
```

in questo caso si ottiene:

ERRORE:b3574NbPIACENZAbbb586500

5.2.2. *Input ed output di dati organizzati in flussi*

Esaminiamo alcune istruzioni che consentono di gestire il trasferimento dei dati dalla memoria centrale alle periferiche e viceversa.

Le operazioni di lettura/registrazione dei flussi sono eseguite mediante i verbi:

a. OPEN
b. CLOSE
c. READ
d. WRITE

Per ora faremo riferimento solo a flussi con organizzazione sequenziale ed accesso sequenziale (vedi 1.)

5.2.2.1. *Il verbo OPEN*

Il verbo OPEN predispone un flusso all'elaborazione in quanto provvede alle operazioni di inizializzazione, controllando la disponibilità delle periferiche, e se, esistono, le etichette di flusso.

L'istruzione OPEN deve essere eseguita prima di qualsiasi altro trattamento del flusso: questo verbo permette di aprire in input oppure in output più flussi.

Se ci limitiamo a considerare il caso di flussi che possono essere trattati o in input oppure in output, possiamo ritenere valido il seguente formato:

$$\text{OPEN} \left\{ \begin{matrix} \text{INPUT} \\ \text{OUTPUT} \end{matrix} \right\} \text{nome-flusso}_1 \; [\text{nome-flusso}_2] \ldots$$

Esempio:

```
OPEN INPUT MOVIM ARCHIV
OPEN OUTPUT STAMPA ARCHI1
```

dove MOVIM, ARCHIV, STAMPA ed ARCHI1 sono i nomi-flusso attribuiti dal programmatore nelle SELECT relative.

5.2.2.2. *Il verbo CLOSE*

Questo verbo permette di ultimare il trattamento dell'ultimo record fisico ed eventualmente provvede al controllo e/o alla generazione delle etichette di fine flusso.

Il verbo CLOSE deve comparire come ultima istruzione operativa sul flusso prima della fine del programma, o prima di riaprire il flusso con una successiva OPEN.

La CLOSE, che chiude i flussi indipendentemente dal tipo di

OPEN utilizzata e che, per flussi su nastro magnetico, provvede anche al riavvolgimento della bobina, ha questo formato:

$$\underline{\text{CLOSE}} \; \text{nome-flusso}_1 \; [\text{nome-flusso}_2] \ldots$$

Esempio:

```
CLOSE MOVIM ARCHIV ARCHI1 STAMPA
```

5.2.2.3. *Il verbo READ*

Il verbo READ trasferisce i record logici di un flusso in un'area di memoria accessibile al programmatore attraverso i nomi-dato assegnati nella File Description (FD) di quel flusso.

Un flusso su cui si opera con questo verbo deve essere stato trattato con una OPEN INPUT.

Se il flusso in esame è su schede o su nastro (o disco) magnetico con record fisici contenenti un solo record logico, il comando READ fa eseguire la lettura di un record fisico; se invece il flusso in trattamento è su nastro (o disco) magnetico con record fisici contenenti più di un record logico, il comando READ fa eseguire saltuariamente la lettura di un record fisico e regolarmente provvede a rendere disponibile al trattamento un record logico per volta (eseguendo semplicemente un trasferimento del record da area blocco ad area record).

Il formato del verbo READ è:

$$\underline{\text{READ}} \; \text{nome-flusso AT} \; \underline{\text{END}} \; \text{istr-imperativa.} \; \ldots\ldots$$

Quando durante l'esecuzione dell'istruzione di lettura viene riscontrato l'identificativo di fine flusso vengono eseguite le istruzioni che si trovano dopo la frase AT END, tali istruzioni devono essere di tipo imperativo; se invece il record letto non è un identificativo di fine flusso, il controllo passa all'istruzione che segue il primo punto inserito nella codifica dopo la frase di lettura.

Esempio:

```
READ SCHEDE AT END GO TO FINE.
MOVE SK TO DEPOSITO ....
```

significa:

leggi il flusso SCHEDE e se è fine flusso esegui il paragrafo FINE, altrimenti trasferisci SK in DEPOSITO...

Ricordiamo che è buona norma evitare di introdurre delle strutture di flow-chart libere e pertanto è consigliabile attenersi, ad esempio, ad uno degli schemi indicati in fig. 5.4.

5.2.2.4. *Il verbo WRITE*

Il verbo WRITE consente di trasferire in output un record logico; il trasferimento interessa direttamente la periferica, a meno che il flusso non sia su nastro o disco magnetico ed abbia un bloccaggio maggiore di uno, in questo caso il record logico preparato dal programmatore in area record viene trasferito in area blocco in corrispondenza di ogni WRITE, e l'area blocco viene trasferita effettivamente in output solo quando il blocco è completo oppure al momento dell'esecuzione della CLOSE.

L'istruzione WRITE opera su flussi dichiarati in output e quindi trattati con una OPEN OUTPUT.

Se il flusso in esame è su schede, su nastro o su disco magnetico il formato è:

WRITE nome-record

Se il flusso in trattamento è su stampante il formato è più complesso, in quanto insieme all'ordine effettivo di stampa, il Cobol richiede che siano fornite nel dettaglio le eventuali specifiche per il movimento della carta:

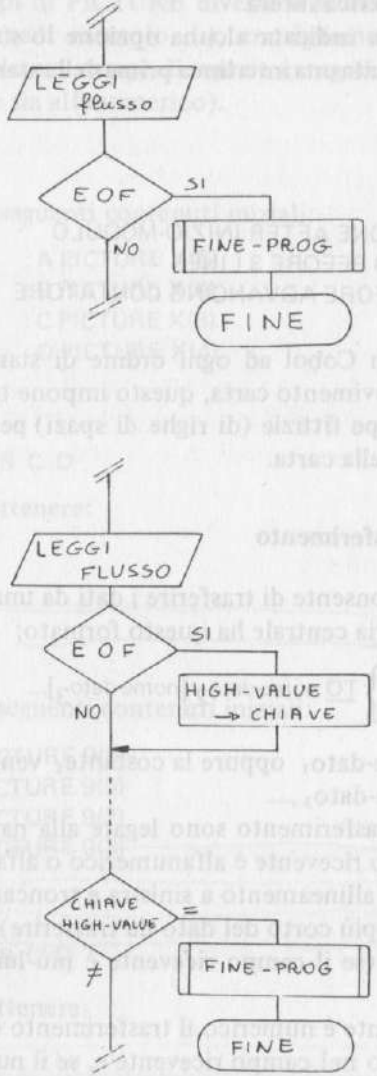
WRITE nome-record
[{ AFTER / BEFORE } ADVANCING { identificatore / nome-dato [LINE / LINES] / intero }]

le opzioni AFTER/BEFORE stanno ad indicare che la stampa della riga deve essere effettuata DOPO/PRIMA l'esecuzione del movimento carta.

L'identificatore è un nome mnemonico introdotto negli SPECIAL-NAMES per individuare un canale-loop del dispositivo di movimento carta della stampante.

La voce nome-dato oppure intero, LINE oppure LINES è relativa

Fig. 5.4



al numero di interlinee da eseguire. Nome-dato deve essere un campo di PICTURE numerica intera.

Qualora non venga indicata alcuna opzione lo standard prevede che debba essere eseguita una interlinea prima della stampa della riga*.

Esempi:

```
WRITE REC-ARK
WRITE RIGA
WRITE INTESTAZIONE AFTER INIZIO-MODULO
WRITE DETTAGLIO BEFORE 3 LINES
WRITE RIGA-W BEFORE ADVANCING CONTATORE
```

Osserviamo che in Cobol ad ogni ordine di stampa può essere associato un solo movimento carta, questo impone talvolta di dover effettuare delle stampe fittizie (di righe di spazi) per poter eseguire dei posizionamenti della carta.

5.3. Istruzione di trasferimento

L'istruzione che consente di trasferire i dati da una zona ad una o più altre della memoria centrale ha questo formato:

$$\underline{\text{MOVE}} \left\{ \begin{matrix} \text{nome-dato}_1 \\ \text{costante}_1 \end{matrix} \right\} \underline{\text{TO}}\ \text{nome-dato}_2\ [\text{nome-dato}_3]\ldots$$

il contenuto di nome-dato$_1$ oppure la costante$_1$ vengono riprodotti in nome-dato$_2$, nome-dato$_3$,...

Le modalità del trasferimento sono legate alla natura del campo ricevente: se il campo ricevente è alfanumerico o alfabetico il trasferimento avviene con allineamento a sinistra e troncando a destra (se il campo ricevente è più corto del dato da trasferire) oppure aggiungendo spazi a destra (se il campo ricevente è più lungo del dato da trasferire).

Se il campo ricevente è numerico il trasferimento è fatto allineando a destra il numero nel campo ricevente e, se il numero da trasferire è più lungo del campo ricevente vengono perdute le cifre più significative del numero, mentre se è più corto vengono inseriti degli zeri non significativi a sinistra.

Limitatamente ai campi con PICTURE esaminate finora sono

possibili tutti i tipi di trasferimento: ma poiché utilizzando il verbo MOVE per campi di PICTURE diversa si possono avere delle alterazioni dei dati (vedi esempio d.), consigliamo per ora di operare trasferimenti solo sui campi di uguale categoria (o, per lo meno, il campo ricevente sia alfanumerico).

Esempi

a. Partendo dai seguenti contenuti iniziali:

```
NAPOLI      : A PICTURE X(6)
ABC512      : B PICTURE X(6)
ROMA/+09    : C PICTURE X(8)
1234        : D PICTURE X(4)
```

l'istruzione:

```
MOVE A TO B C D
```

permette di ottenere:

```
NAPOLI    : A
NAPOLI    : B
NAPOLI55  : C
NAPO      : D
```

b. Partendo dai seguenti contenuti iniziali:

```
12000   : A PICTURE 9(5)
56789   : B PICTURE 9(5)
1234567 : C PICTURE 9(7)
128     : D PICTURE 9(3)
```

l'istruzione:

```
MOVE A TO B C D
```

permette di ottenere:

```
12000   : A
12000   : B
0012000 : C
000     : D
```

c. Partendo dai seguenti contenuti iniziali:

567123 : A PICTURE 9(6)
12+/5A : B PICTURE X(6)
TORINESE : C PICTURE X(8)

l'istruzione:

MOVE A TO B C D

permette di ottenere:

567123 : A
567123 : B
5671235b : C

d. Esaminiamo ora un caso formalmente corretto ma che porta ad errori logici in quanto non sono state considerate nel modo dovuto le regole che stabiliscono le modalità di trasferimento.
Supponiamo di avere i campi A e B con i seguenti contenuti iniziali:

12345 : A PICTURE 9(5)
FIRENZE : B PICTURE X(7)

l'istruzione:

MOVE A TO B

opera un trasferimento alfanumerico e permette di ottenere:

12345 : A
12345bb : B

se facciamo eseguire ora l'istruzione:

MOVE B TO A

che opera un trasferimento numerico otteniamo:

345bb : A
12345bb : B

osserviamo come sia diverso il valore di A dopo il duplice trasferimento

A→B→A

questa differenza dipende dalla natura dei campi soggetti ad elaborazione.

5.4. Istruzioni aritmetiche

L'analisi dei formati delle istruzioni aritmetiche ci consentirà di osservare che, in alcuni di essi, tutti i campi coinvolti nelle istruzioni sono anche operandi, in altri casi, alcuni campi che compaiono nell'operazione hanno solo il ruolo di "campi riceventi del risultato".

Tutti i campi che sono anche operandi delle istruzioni che presenteremo devono essere campi numerici, i campi che hanno la sola funzione di ricevere il risultato dell'operazione possono essere anche campi non numerici; tutte le costanti che sono citate nei vari formati devono essere numeriche.

Nella presentazione dei formati non abbiamo, in generale, citato il numero massimo di operandi previsto per l'operazione in esame in quanto tale numero è funzione dell'elaboratore che si sta utilizzando.

5.4.1. *Addizione*

Esaminiamo i possibili formati:

a. $\underline{\text{ADD}} \left\{ \begin{matrix} \text{nome-dato}_1 \\ \text{costante}_1 \end{matrix} \right\} \left[\begin{matrix} \text{nome-dato}_2 \\ \text{costante}_2 \end{matrix} \right] \ldots \underline{\text{TO}}\ \text{nome-dato}$

questa istruzione equivale a:

$\left\{ \begin{matrix} \text{nome-dato}_1 \\ \text{costante}_1 \end{matrix} \right\} + \left[\begin{matrix} \text{nome-dato}_2 \\ \text{costante}_2 \end{matrix} \right] + \ldots + \text{nome-dato} \longrightarrow \text{nome-dato}$

b. $\underline{\text{ADD}} \left\{ \begin{matrix} \text{nome-dato}_1 \\ \text{costante}_1 \end{matrix} \right\} \left\{ \begin{matrix} \text{nome-dato}_2 \\ \text{costante}_2 \end{matrix} \right\} \left[\begin{matrix} \text{nome-dato}_3 \\ \text{costante}_3 \end{matrix} \right] \ldots$
$\underline{\text{GIVING}}$ nome-dato

questa istruzione equivale a:

$\left\{ \begin{matrix} \text{nome-dato}_1 \\ \text{costante}_1 \end{matrix} \right\} + \left\{ \begin{matrix} \text{nome-dato}_2 \\ \text{costante}_2 \end{matrix} \right\} + \left[\begin{matrix} \text{nome-dato}_3 \\ \text{costante}_3 \end{matrix} \right] + \ldots \longrightarrow \text{nome-dato}$

Esempio:

Partendo dai seguenti contenuti iniziali:

```
0350  : A PIC 9(4)
130   : B PIC 9(3)
00540 : C PIC 9(5)
55000 : T1 PIC 9(5)
999   : T2 PIC 9(3)
```

l'istruzione:

```
ADD A B C 55 TO T1
```

consente di ottenere:

```
0350  : A
130   : B
00540 : C
56075 : T1
999   : T2
```

invece l'istruzione:

```
ADD A B C 55 GIVING T1
```

porta a questo risultato:

```
0350  : A
130   : B
00540 : C
01075 : T1
999   : T2
```

se infine consideriamo l'istruzione:

```
ADD A B C 55 GIVING T2
```

otteniamo:

```
0350  : A
130   : B
00540 : C
55000 : T1
075   : T2*
```


* come si può osservare, in questo caso il campo ricevente T2 è troppo corto per potere contenere il risultato e quindi viene perduta la parte più significativa: nel capitolo 7 esamineremo la clausola che permette di evitare questo inconveniente.

5.4.2. *Sottrazione*

I formati ammessi per questa operazione sono:

a. $\underline{\text{SUBTRACT}} \left\{ \begin{matrix} \text{nome-dato}_1 \\ \text{costante}_1 \end{matrix} \right\} \left[\begin{matrix} \text{nome-dato}_2 \\ \text{costante}_2 \end{matrix} \right] \ldots \underline{\text{FROM}}\ \text{nome-dato}$

questa istruzione equivale a:

$$\text{nome-dato} - \left(\left\{ \begin{matrix} \text{nome-dato}_1 \\ \text{costante}_1 \end{matrix} \right\} + \left[\begin{matrix} \text{nome-dato}_2 \\ \text{costante}_2 \end{matrix} \right] + \ldots \right) \rightarrow \text{nome-dato}$$

b. $\underline{\text{SUBTRACT}} \left\{ \begin{matrix} \text{nome-dato}_1 \\ \text{costante}_1 \end{matrix} \right\} \left[\begin{matrix} \text{nome-dato}_2 \\ \text{costante}_2 \end{matrix} \right] \ldots \underline{\text{FROM}}\ \text{nome-dato}$

$\underline{\text{GIVING}}\ \text{nome-dato}_x$

questa istruzione equivale a:

$$\text{nome-dato} - \left(\left\{ \begin{matrix} \text{nome-dato}_1 \\ \text{costante}_1 \end{matrix} \right\} + \left[\begin{matrix} \text{nome-dato}_2 \\ \text{costante}_2 \end{matrix} \right] + \ldots \right) \rightarrow \text{nome-dato}_x$$

Esempio:

Partendo dai seguenti contenuti iniziali:

```
500   : A PIC 9(3)
0560  : B PIC 9(4)
1800  : C PIC 9(4)
56705 : D PIC 9(5)
```

l'istruzione:

```
SUBTRACT A B 85 FROM C
```

consente di ottenere:

500 : A
0560 : B
0655 : C
56705 : D

mentre l'istruzione:

SUBTRACT A B 85 FROM C GIVING D

porta a questo risultato:

500 : A
0560 : B
1800 : C
00655 : D

Il problema di risultati negativi sarà automaticamente risolto quando introdurremo i campi atti a contenere numeri con segno ; per quanto riguarda problemi di "overflow" del risultato vale quanto detto per l'addizione e lo stesso è per le operazioni che presentiamo di seguito.

5.4.3. *Moltiplicazione*

Questi sono i possibili formati:

a. $\underline{\text{MULTIPLY}} \begin{Bmatrix} \text{nome-dato} \\ \text{costante} \end{Bmatrix} \underline{\text{BY}}\ \text{nome-dato}_1\ [\text{nome-dato}_2]\ldots$

questa istruzione equivale a:

$$\begin{Bmatrix} \text{nome-dato} \\ \text{costante} \end{Bmatrix} \cdot \text{nome-dato}_1 \longrightarrow \text{nome-dato}_1$$

$$\left[\begin{Bmatrix} \text{nome-dato} \\ \text{costante} \end{Bmatrix} \cdot \text{nome-dato}_2 \longrightarrow \text{nome-dato}_2\right]$$

.....

b. $\underline{\text{MULTIPLY}} \begin{Bmatrix} \text{nome dato}_1 \\ \text{costante}_1 \end{Bmatrix} \underline{\text{BY}} \begin{Bmatrix} \text{nome-dato}_2 \\ \text{costante}_2 \end{Bmatrix}$

$\underline{\text{GIVING}}\ \text{nome-dato}_{r1}\ [\text{nome-dato}_{r2}]\ldots$

questa istruzione equivale a:

$$\begin{Bmatrix} \text{nome-dato}_1 \\ \text{costante}_1 \end{Bmatrix} \cdot \begin{Bmatrix} \text{nome-dato}_2 \\ \text{costante}_2 \end{Bmatrix} \longrightarrow \text{nome-dato}_{r1}$$

$$\left[\begin{Bmatrix} \text{nome-dato}_1 \\ \text{costante}_1 \end{Bmatrix} \cdot \begin{Bmatrix} \text{nome-dato}_2 \\ \text{costante}_2 \end{Bmatrix} \longrightarrow \text{nome-dato}_{r2}\right]$$

....

Esempio:

Partendo dai seguenti contenuti iniziali:

025 : A PIC 9(3)
04 : B PIC 9(2)
0015 : C PIC 9(4)
035 : D PIC 9(3)

l'istruzione:

MULTIPLY A BY C D

porta a questo risultato:

025 : A
04 : B
0375 : C
875 : D

mentre invece l'istruzione:

MULTIPLY A BY B GIVING C D

consente di ottenere:

025 : A
04 : B
0100 : C
100 : D

5.4.4. *Divisione*

La divisione ammette tre formati:

a. $\underline{\text{DIVIDE}} \left\{ \begin{matrix} \text{nome-dato}_1 \\ \text{costante}_1 \end{matrix} \right\} \underline{\text{INTO}}\ \text{nome-dato}_2\ [\text{nome-dato}_3]\ldots$

questa istruzione equivale a:

$$\text{nome-dato}_2 : \left\{ \begin{matrix} \text{nome-dato}_1 \\ \text{costante}_1 \end{matrix} \right\} \rightarrow \text{nome-dato}_2$$

$$\left[\text{nome-dato}_3 : \left\{ \begin{matrix} \text{nome-dato}_1 \\ \text{costante}_1 \end{matrix} \right\} \rightarrow \text{nome-dato}_3 \right]$$

...

b. $\underline{\text{DIVIDE}} \left\{ \begin{matrix} \text{nome-dato}_1 \\ \text{costante}_1 \end{matrix} \right\} \underline{\text{INTO}} \left\{ \begin{matrix} \text{nome-dato}_2 \\ \text{costante}_2 \end{matrix} \right\}$

$\underline{\text{GIVING}}\ \text{nome-dato}_{r1}\ [\text{nome-dato}_{r2}]\ldots$

questa istruzione equivale a:

$$\left\{ \begin{matrix} \text{nome-dato}_2 \\ \text{costante}_2 \end{matrix} \right\} : \left\{ \begin{matrix} \text{nome-dato}_1 \\ \text{costante}_1 \end{matrix} \right\} \rightarrow \text{nome-dato}_{r1}$$

$$\left[\left\{ \begin{matrix} \text{nome-dato}_2 \\ \text{costante}_2 \end{matrix} \right\} : \left\{ \begin{matrix} \text{nome-dato}_1 \\ \text{costante}_1 \end{matrix} \right\} \rightarrow \text{nome-dato}_{r2} \right]$$

c. $\underline{\text{DIVIDE}} \left\{ \begin{matrix} \text{nome-dato}_2 \\ \text{costante}_2 \end{matrix} \right\} \underline{\text{BY}} \left\{ \begin{matrix} \text{nome-dato}_1 \\ \text{costante}_1 \end{matrix} \right\}$

$\underline{\text{GIVING}}\ \text{nome-dato}_{r1}\ [\text{nome-dato}_{r2}]\ldots$

questo formato è perfettamente equivalente a quello presentato in b.

Esempio:

Partendo dai seguenti contenuti iniziali:

```
005  : A PIC 9(3)
0200 : B PIC 9(4)
0300 : C PIC 9(4)
0150 : D PIC 9(4)
```

l'istruzione:

```
DIVIDE A INTO B C D
```

consente di ottenere:

```
005  : A
0040 : B
0060 : C
0030 : D
```

mentre invece ciascuna delle due istruzioni equivalenti:

```
DIVIDE A INTO B GIVING C D
DIVIDE B BY A GIVING C D
```

consente di ottenere:

```
005  : A
0200 : B
0040 : C
0040 : D
```

5.5. Realizzazione di strutture alternative e di cicli

L'istruzione IF fornisce la possibilità di eseguire trattamenti diversi a seconda dell'esito di un test.

Il Cobol permette di utilizzare una vasta gamma di espressioni condizionali (tratteremo per ora solo a. e b.):

a. test di relazione,
b. test di classe,

c. test di segno,
d. test su condizioni.

Esaminiamo per ora i seguenti formati del verbo IF:

a. IF test istruzioni-1 ELSE istruzioni-2....

Se il risultato del test è positivo, vengono eseguite le istruzioni-1 e quindi il controllo passa dopo il simbolo punto; se il risultato è negativo il controllo passa alla prima istruzione del gruppo istruzioni-2.
Istruzioni-1 e/o istruzioni-2 possono anche essere di tipo condizionale, ma questo richiede ulteriori considerazioni che faremo nel capitolo 7.
Il formato ora descritto permette di codificare uno schema di flow-chart come quello riportato in fig. 5.5.

b. IF test istruzioni-1. istruzioni-2.

Se il risultato del test è positivo, il controllo passa alla prima istruzione del gruppo istruzioni-1, se invece è negativo, viene eseguita la prima istruzione scritta dopo il punto, cioè la prima istruzione del gruppo istruzioni-2.
Il formato ora descritto permette ad esempio di codificare un ramo di flow-chart come quello descritto nella fig. 5.6.

Fig. 5.5

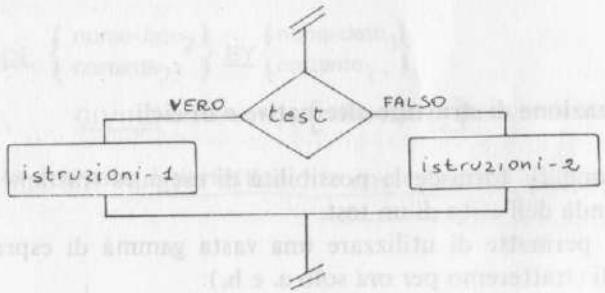
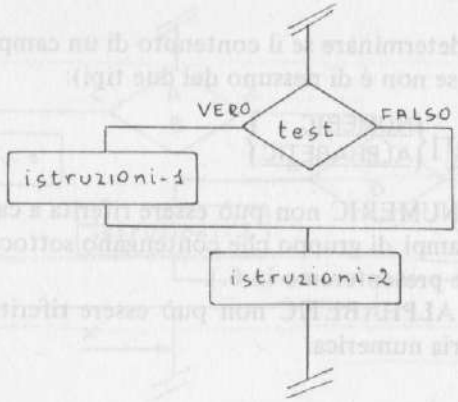


Fig. 5.6



5.5.1. *I test di relazione*

Le espressioni condizionali di relazione permettono di confrontare due dati secondo le seguenti modalità:

$$\left\{\begin{matrix}\text{nome-dato}_1\\ \text{costante}_1\end{matrix}\right\}\ \text{IS [NOT]}\ \left\{\begin{matrix}\underline{\text{GREATER}}\ \text{THAN}\\ \underline{>}\\ \underline{\text{LESS}}\ \text{THAN}\\ \underline{<}\\ \underline{\text{EQUAL}}\ \text{TO}\\ \underline{=}\end{matrix}\right\}\ \left\{\begin{matrix}\text{nome-dato}_2\\ \text{costante}_2\end{matrix}\right\}$$

Le modalità del confronto sono legate alla natura del primo operando: se il confronto è numerico non ci sono problemi relativi alla lunghezza dei due operandi in quanto il più corto viene virtualmente allungato a sinistra con l'aggiunta di zeri non significativi, se invece il confronto è di tipo alfanumerico la lunghezza del primo operando condiziona in molti casi l'operazione di confronto quindi è consigliabile, per confronti alfanumerici, lavorare sempre in campi di uguale lunghezza.

5.5.2. *I test di classe*

Permettono di determinare se il contenuto di un campo è numerico o alfabetico (o se non è di nessuno dei due tipi):

nome-dato IS [NOT] {NUMERIC / ALPHABETIC}

La condizione NUMERIC non può essere riferita a campi di PICTURE A(n) e a campi di gruppo che contengano sottocampi numerici con segno (che presenteremo in 6.).

La condizione ALPHABETIC non può essere riferita ad un nome-dato di categoria numerica.

5.5.3. *L'istruzione di salto incondizionato*

Presentiamo in questo contesto l'istruzione di salto incondizionato GO TO, in quanto deve essere intesa come un'istruzione atta a realizzare delle strutture di controllo e quindi deve essere utilizzata in modo regolamentato.

Il formato dell'istruzione è:

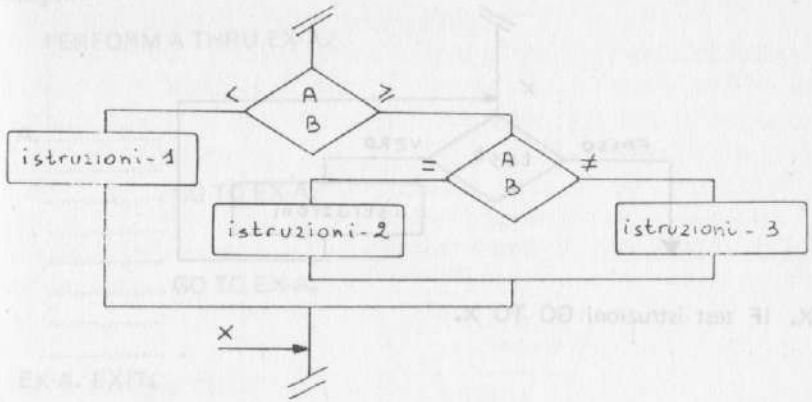
GO TO nome-procedura

nome procedura può essere un nome-paragrafo oppure un nome-sezione.

Nel contesto di una struttura alternativa (abbiamo visto come si possono codificare i due casi delle figg. 5.5 e 5.6), l'istruzione GO TO assume interesse quando all'interno della struttura si presenta la necessità di utilizzare istruzioni condizionali: in questi casi la GO TO consente di evitare di "nidificare" le IF. E' il caso del confronto a tre uscite di fig. 5.7.

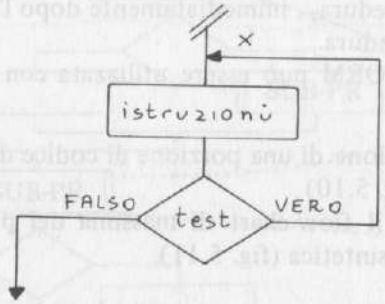
L'istruzione GO TO è indispensabile in Cobol anche per la realizzazione dei cicli come si può vedere dalle figg. 5.8 e 5.9.

Fig. 5.7



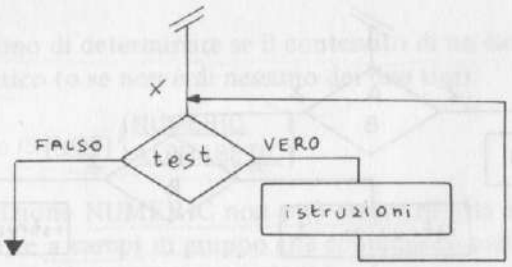

```
IF A < B istruzioni-1 GO TO X.
IF A = B istruzioni-2
ELSE istruzioni-3.
X. ...................
```

Fig. 5.8




```
X. istruzioni
IF test GO TO X.
...........
```

Fig. 5.9



X. IF test istruzioni GO TO X.

5.6. Richiamo di un sottoprogramma interno (FUNCTION)

Il Cobol dispone dell'istruzione

PERFORM nome-procedura$_1$ [{ THROUGH / THRU } nome-procedura$_2$]

che consente di passare il controllo alla prima istruzione di procedura$_1$ (paragrafo o sezione) per rientrare, dopo l'esecuzione dell'ultima istruzione di procedura$_2$, immediatamente dopo l'istruzione che ha richiamato la procedura.

L'istruzione PERFORM può essere utilizzata con un duplice significato:

a. richiamare l'esecuzione di una porzione di codice da punti diversi del programma (fig. 5.10)
b. scrivere in Cobol il flow-chart di massima del programma per averne una visione sintetica (fig. 5.11).

Talvolta la struttura di flow-chart che si vuole codificare prevede il richiamo tramite una PERFORM di una procedura (semplice o composta da più procedure in sequenza) in cui risulta difficile codificare il punto d'uscita "unico" del sottoprogramma: per questi casi il Cobol mette a disposizione il verbo EXIT (che deve essere l'unica istruzione di un paragrafo) che ha lo scopo di identificare il punto di uscita del sottoprogramma.

Esempio:

```
PERFORM A THRU EX-A.
.
.
A. ..............
..............
.............. GO TO EX-A.
..............
..............
.............. GO TO EX-A.
..............
.............. .
EX-A. EXIT.
```

Fig. 5.10

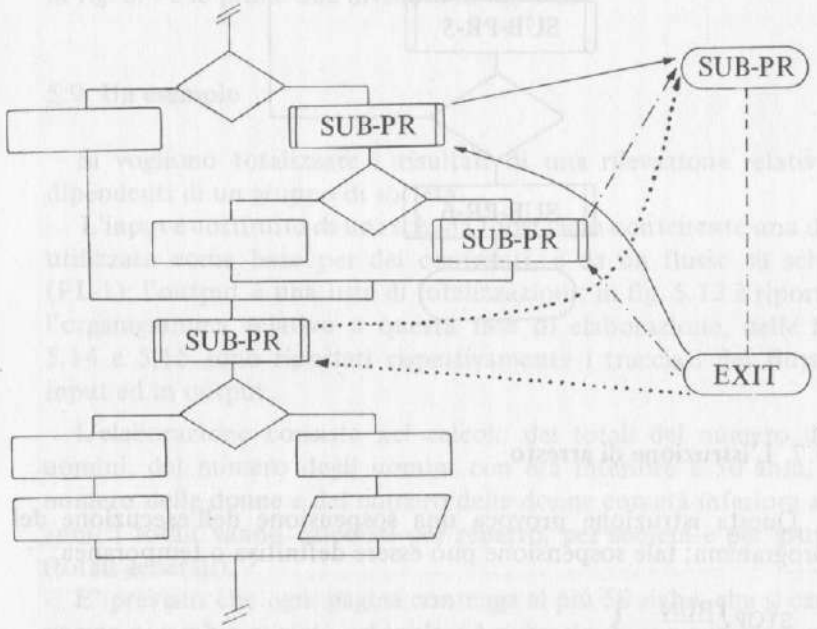
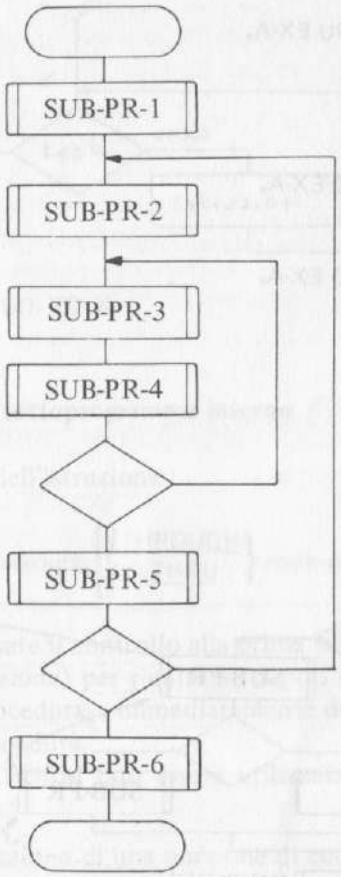


Fig. 5.11



5.7. L'istruzione di arresto

Questa istruzione provoca una sospensione dell'esecuzione del programma; tale sospensione può essere definitiva o temporanea:

STOP { RUN / costante }

L'istruzione STOP, nella forma STOP RUN, viene utilizzata come ultima istruzione operativa del programma.

Nella forma "STOP costante" questa istruzione provoca un arresto temporaneo dell'elaborazione e, secondo le modalità previste dal sistema operativo dell'elaboratore utilizzato, fornisce all'operatore come segnalazione la costante indicata. Quando l'operatore da lo "Start" ad un programma dopo l'esecuzione di uno stop temporaneo il controllo passa all'istruzione immediatamente seguente. La costante può essere una qualunque costante Cobol ma non può contenere la parola ALL.

5.8. L'esempio PROG1

Codifichiamo ora (fig. 5.12) la PROCEDURE DIVISION dell' esempio PROG1 introdotto in 4.1.4.

Ricordiamo che l'organigramma di PROG1 era stato presentato in fig. 3.4 e le prime due divisioni in fig. 3.5.

5.9. Un esempio

Si vogliono totalizzare i risultati di una rilevazione relativa ai dipendenti di un gruppo di società.

L'input è costituito da una scheda fuori ciclo contenente una data, utilizzata come base per dei confronti, e da un flusso su schede (FL-1); l'output è una lista di totalizzazioni: in fig. 5.13 è riportato l'organigramma relativo a questa fase di elaborazione, nelle figg. 5.14 e 5.15 sono riportati rispettivamente i tracciati dei flussi in input ed in output.

L'elaborazione consiste nel calcolo dei totali del numero degli uomini, del numero degli uomini con età inferiore a 50 anni, del numero delle donne e del numero delle donne con età inferiore a 40 anni; i totali vanno calcolati per reparto, per società e per gruppo (totali generali).

E' previsto che ogni pagina contenga al più 50 righe, che si cambi pagina a cambio società, ed inoltre è richiesto di non cambiare pagina per la stampa dei soli totali di società.

Fig. 5.12a

PROGRAM NAME | PROGRAMMER | DATE

PAGE Nº 005

PROGRAM-ID PROG1

LINE	CONT.	A B	B 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72
010		PROC	EDURE DIVISION.
020		PROG	RAMMA SECTION.
030		INI12	IO.
040			PERFORM INI12 IO-PROG.
050		LETTURA.	
060			READ FILE-SKI AT END PERFORM FINE-PROG STOP RUN.
070			PERFORM TRATTAM
080			GO TO LETTURA.
090			
100		INI12	IO-PROG SECT IONI.
110			OPEN INPUT FIL-SK
120			OPEN OUTPUT FIL-DS FIL-STI
130			MOVE SPACE TO RIGA.
140			
150		FINE	-PROG SECTIONI.
160			CLOSE FIL-SKI FIL-STI FIL-DS.
170			
180		TRATT	AM SECTIONI.
190			IF TSKI NOTI = "C" MOVE "*" TO K GO TO K3.
200			IF CH-ART NOTI NUMERI C MOVE "A" TO A GO TO H1.
210			MOVE ART TO NUM
220			PERFORM CALCOLO
230			IF DEP NOTI = "B" TO B.
240		H1.	IF CH-CL NOTI NUMERI C MOVE "C" TO C GO TO H2.

Fig. 5.12b

PROGRAM NAME | PROGRAMMER | DATE

PAGE Nº 006

PROGRAM-ID PROG1

LINE	CONT.	A B	B 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72
010			MOVE CLI TO NUM
020			PERFORM CALCOLO
030			IF DEP NOTI = "D" TO D.
040		H2.	IF DATA-MOV NOTI NUMERI C MOVE "E" TO E GO TO K2.
050			IF MESE < 1 GO TO M1.
060			IF MESE NOTI > 12 GO TO K1.
070		M1.	MOVE "F" TO F.
080		K1.	IF GIORNO < 1 GO TO M2.
090			IF GIORNO NOTI > 31 GO TO K2.
100		M2.	MOVE "G" TO G.
110		K2.	IF QTA NOTI NUMERIC MOVE "H" TO H.
120		K3.	IF ERROR-W = SPACE
130			MOVE RECC-DS
140			WRITE RECC-DS GO TO FIL-TRATTAM.
150			ADD 1 TO CONTI-RIGHE
160			IF CONTI-RIGHE > 25 PERFORM INTESTA.
170			MOVE SCHEDA TO RECC-STI MOVE ERRORI-W TO ERRORI
180			WRITE RIGA BEFORE ADVANCING 2 LINES
190			MOVE SPACE TO RIGA ERROR-W.
200		FINE	-TRATTAM. EXIT.
210		CALCOLO	SECTIONI.
220			DIVIDE NUM BY 97 GIVING DEP
230			MULTIPLY 97 BY DEP
240			SUBTRACT DEP FROM NUM GIVING DEP.

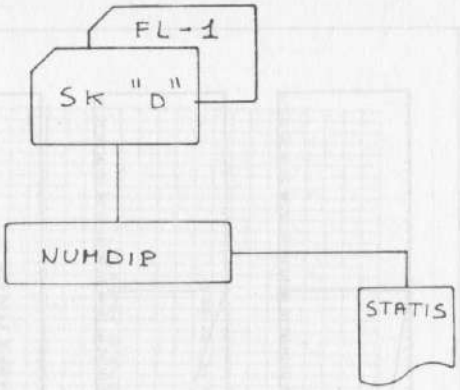
Fig. 5.12c

PROGRAM NAME | PROGRAMMER | DATE

PROGRAM-ID PROG1 PAGE NO. 00

LINE	CONT.	A	B
010		INTESTA SELECTION.	
020		MOVE "L"	
030		TO REC-1ST	
040		MOVE "1" TO ERROR1	
050		WRITE RIGA AFTER ADVANCING 1 NI-PAG	
060		MOVE SPACE TO RIGA	
070		MOVE ALL "-" TO REC-1ST ERROR1	
080		WRITE RIGA AFTER ADVANCING 2 LINEE	
090		MOVE SPACE TO RIGA	
100		WRITE RIGA AFTER ADVANCING 3 LINEE	
110		MOVE 1 TO CONT-RIGHE.	
120			
130			

Fig. 5.13



I totali generali (totali per gruppo) devono essere stampati a pagina nuova.

L'età va calcolata in base alla data di elaborazione, che non tiene conto del giorno.

In fig. 5.16 è rappresentato il flow-chart descrittivo del problema proposto, in fig. 5.17 la relativa codifica.

Si suppone che il codice SESSO sia sicuramente 1 (= uomini) o 2 (= donne).

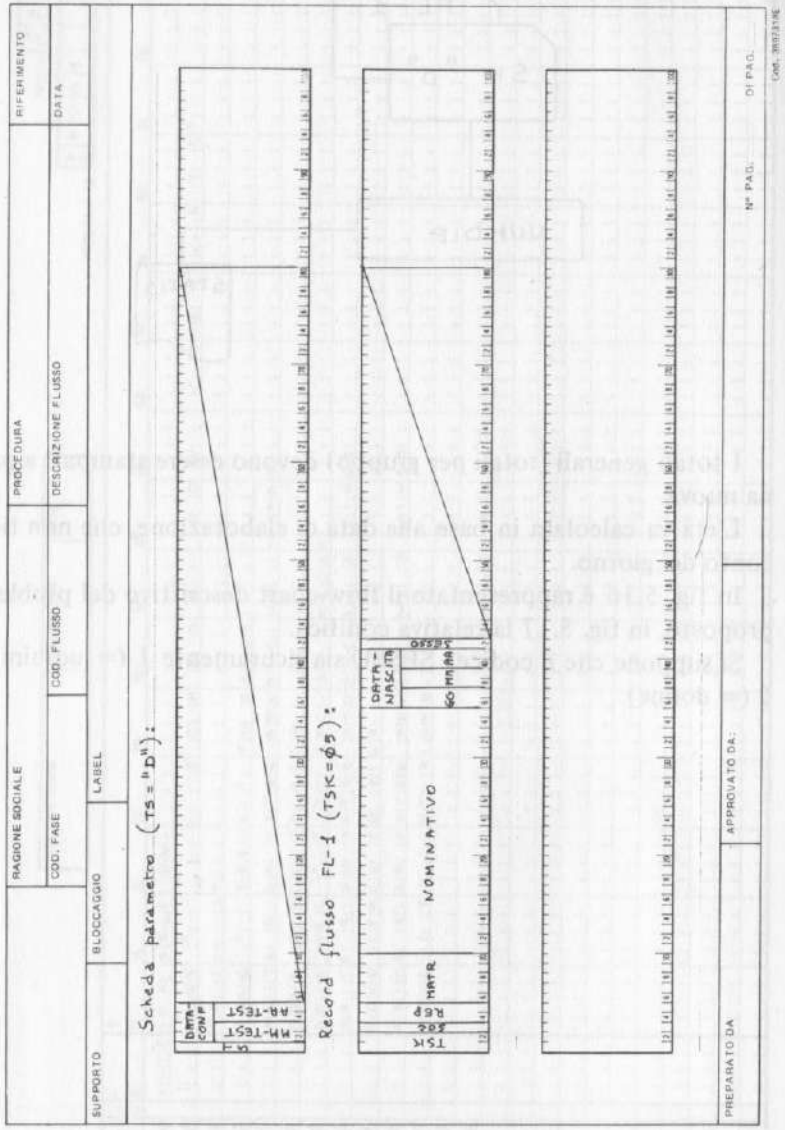


Fig. 5.14

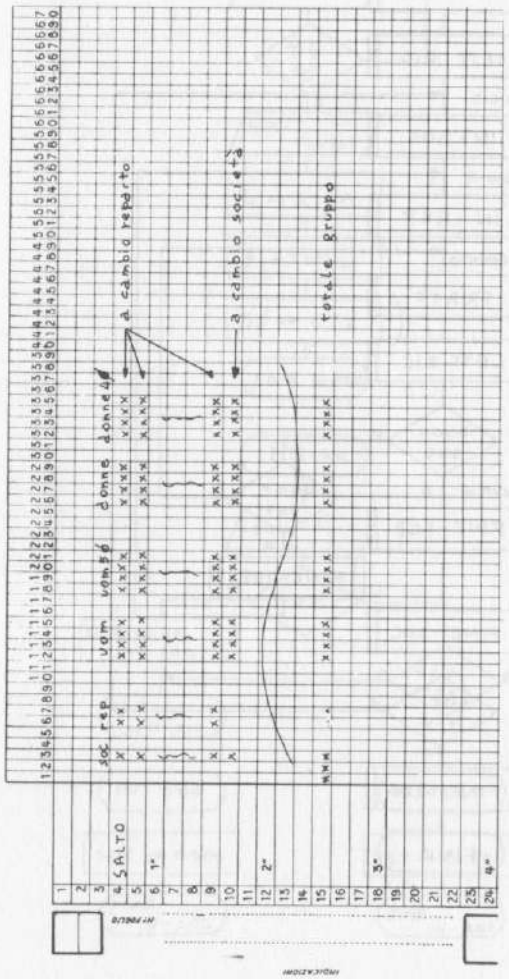


Fig. 5.15

Fig. 5.16a

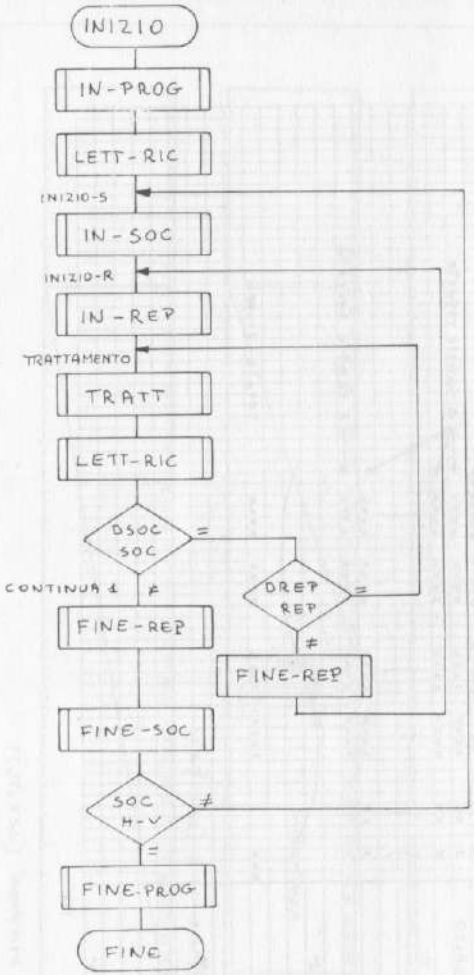


Fig. 5.16b

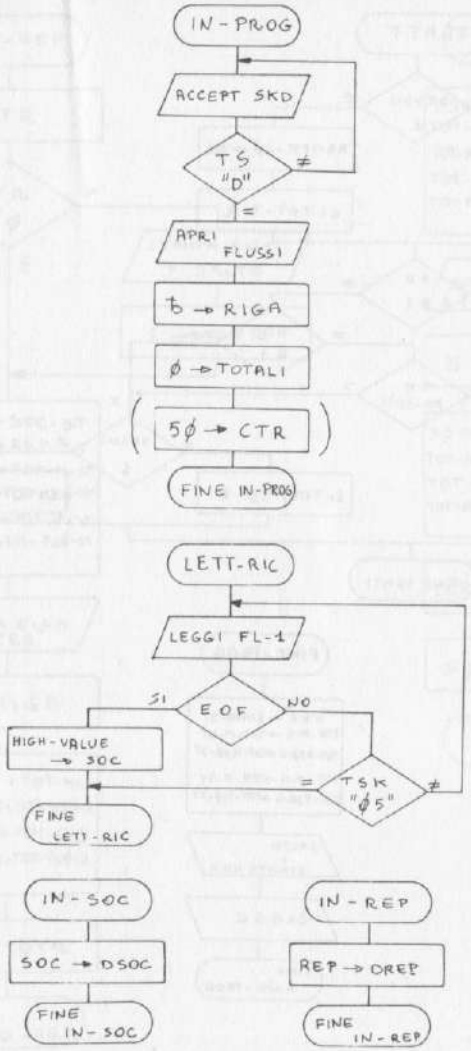


Fig. 5.16c

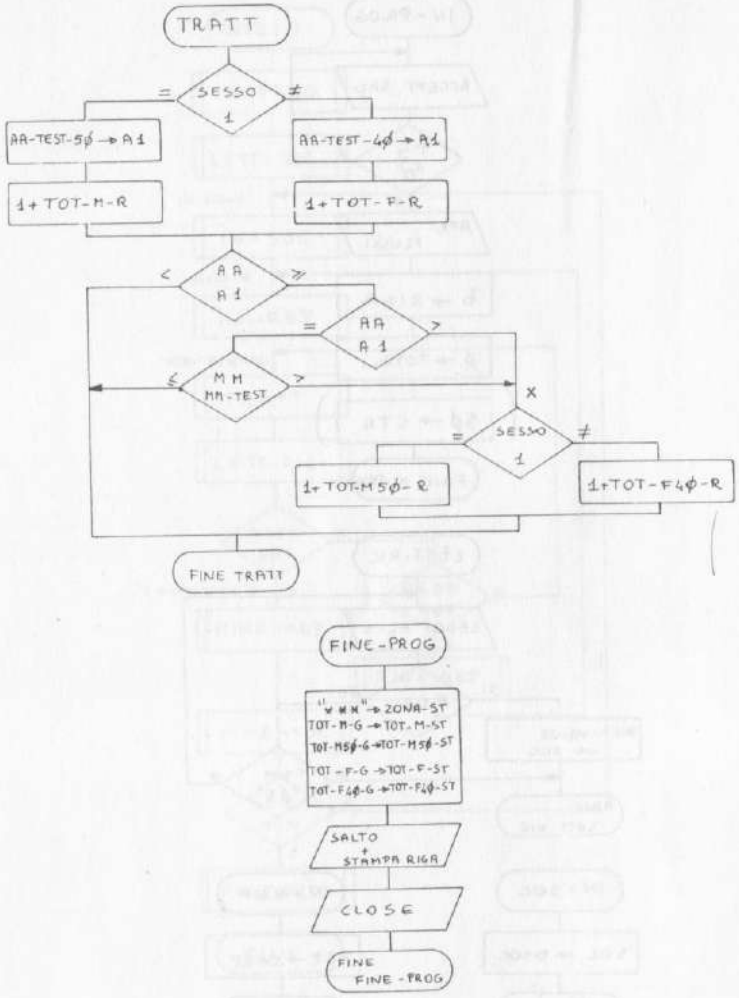


Fig. 5.16d

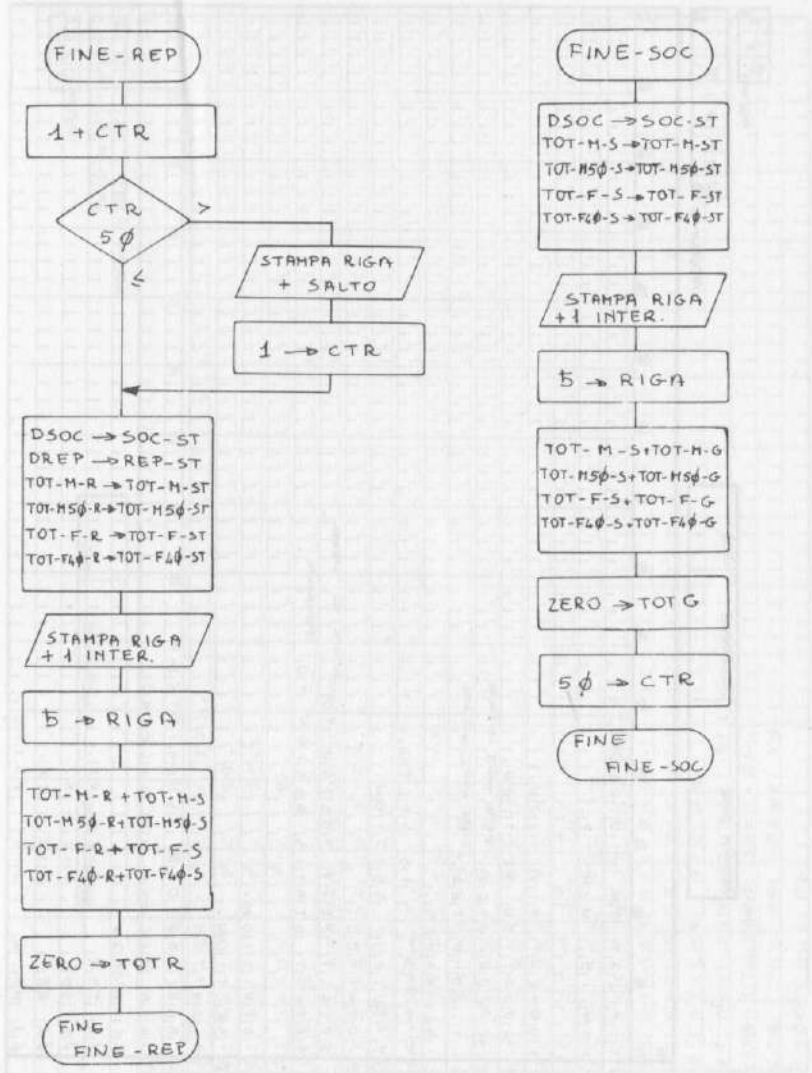


Fig. 5.17a

PAGE Nº 1 Ø Ø 1

PROGRAM NAME	PROGRAMMER	DATE

PROGRAM-ID 73 N U M D I P 80

LINE 4	CONT.	A 8	B 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72
Ø1Ø		IDEN	TIFICATIONI DIVISIONI.
Ø2Ø		PROG	RAMMI-IDI. NUMDIPI.
Ø3Ø			
Ø4Ø		ENVI	RONMENTI DIVISION.
Ø5Ø		CONF	IGURATION SECTION.
Ø6Ø		SOUR	CE-COMPUTER. sigla unità centrale.
Ø7Ø		OBJE	CT-COMPUTER. sigla unità centrale.
Ø8Ø		SPEC	IAL-NAMES.
Ø9Ø			inizio pagina IS SALTO.
1Ø0		INPU	T-OUTPUT SECTION.
11Ø		FILE	-CONTROL.
12Ø			SELECT FILE-1 ASSIGN TO lettore di schede.
13Ø			SELECT STAMPA ASSIGN TO stampante.
14Ø			
15Ø		DATA	DIVISION.
16Ø		FILE	SECTION.
17Ø		FD	FILE-1
18Ø			LABEL RECORD IS OMITTED
19Ø			DATA RECORD IS RECVA2.
2Ø0	Ø1		RILEVA2.
21Ø			Ø2 TSK PIC XX.
22Ø			Ø2 SOC PIC X.
23Ø			Ø2 REP PIC XX.
24Ø			Ø2 MATR PIC XXXX.

Fig. 5.17b

PAGE Nº 1 Ø Ø 2

PROGRAM NAME	PROGRAMMER	DATE

PROGRAM-ID 73 N U M D I P 80

LINE 4	CONT.	A 8	B 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72
Ø1Ø			Ø2 FILLER PIC X(127).
Ø2Ø			Ø2 MM PIC 99.
Ø3Ø			Ø2 AA PIC 99.
Ø4Ø			Ø2 SESSO PIC 9.
Ø5Ø		FD	SITAT IS
Ø6Ø			LABEL RECORD IS OMITTED
Ø7Ø			DATA RECORD IS RIGA.
Ø8Ø	Ø1		RIGA.
Ø9Ø			Ø2 CONA-ST.
1Ø0			Ø3 FILLER PIC XX.
11Ø			Ø3 SOC-ST PIC XX.
12Ø			Ø2 FILLER PIC XX.
13Ø			Ø2 REP-ST PIC XX.
14Ø			Ø2 FILLER PIC X(14).
15Ø			Ø2 TOT-M-ST PIC 9(14).
16Ø			Ø2 FILLER PIC XX.
17Ø			Ø2 TOT-MSØ-ST PIC 9(41).
18Ø			Ø2 FILLER PIC X(41).
19Ø			Ø2 TOT-F-ST PIC 9(14).
2Ø0			Ø2 FILLER PIC XX.
21Ø			Ø2 TOT-F4Ø-ST PIC 9(41).
22Ø	WORK		INGI-STORAGE SECTION.
23Ø	77		CTR PIC 99 VALUE ZØ.
24Ø	77		DISOC PIC X.

Fig. 5.17c

PROGRAM NAME | PROGRAMMER | DATE

PAGE Nº 003

PROGRAM-ID NUMDIP

LINE	CONT.	A	B
010		7	DIREIP PI CI XX.
020		01	TOTALI.
030			02 ITOTRI.
040			03 ITOTI-M-R PI CI 9(4).
050			03 ITOTI-M50-R PI CI 9(4).
060			03 ITOTI-F-R PI CI 9(4).
070			03 ITOTI-F40-R PI CI 9(4).
080			02 ITOTS.
090			03 ITOTI-M-S PI CI 9(4).
100			03 ITOTI-M50-S PI CI 9(4).
110			03 ITOTI-F-S PI CI 9(4).
120			03 ITOTI-F40-S PI CI 9(4).
130			02 ITOTG.
140			03 ITOTI-M-G PI CI 9(4).
150			03 ITOTI-M50-G PI CI 9(4).
160			03 ITOTI-F-G PI CI 9(4).
170			03 ITOTI-F40-G PI CI 9(4).
180		01	SKIDI.
190			02 ITS PI CI X.
200			02 IMM-TESTI PIC 99.
210			02 IAA-TESTI PIC 99.
220		01	AI PIC 99.
230			
240			

Fig. 5.17d

PROGRAM NAME | PROGRAMMER | DATE

PAGE Nº 004

PROGRAM-ID NUMDIP

LINE	CONT.	A	B
010		PRO	EDURE DIVISION.
020		PRO	-PRINCIPAL SECTION.
030			PERFORM INI-PROG
040			PERFORM LEITT-RIC.
050		INI2	IO-S.
060			PERFORM INI-SOC.
070		INI2	IO-R.
080			PERFORM INI-REP.
090		TRAT	TAMENTO.
100			PERFORM TRATT
110			PERFORM LEITT-RIC
120			IF DSOC NOT = SOC GO TO CONTINUA1.
130			IF DREP = REP GO TO TRATTAMENTO.
140			PERFORM FINE-REP
150			GO TO INI2IO-R.
160		CONT	INUA1.
170			PERFORM FINE-REP
180			PERFORM FINE-SOC
190			IF SOC NOT = HIGH-VALUE GO TO INI2IO-S.
200			PERFORM FINE-PROG
210			STOP RUN.
220			
230			
240			

Fig. 5.17e

PROGRAM NAME | PROGRAMMER | DATE

PAGE Nº 005

PROGRAM-ID NUMDIP

LINE	CONT	A	B
010		IN-P	RIOG SECTION.
020			ACCEPT ISKD FITS NOT = " D " GOTO IN-PROG.
030			OPEN INPUT FL-1
040			OPEN OUTPUT STATIS
050			MOVE SPACE TO RIGA
060			MOVE ZERO TO TOTALI.
070			
080		LETT	-RIC SECTION.
090			READ FIL-1 AT END MOVE HIGH-VALUE TO SOCI
100			GOTO USCITA-LETTI-RIC.
110			IF TSKN NOT = "05" GOTO LETTI-RIC.
120		USCI	TA-LETTI-RIC. EXIT.
130			
140		IN-S	OC SECTION.
150			MOVE SOC TO DSOC.
160			
170		IN-R	EP SECTION.
180			MOVE REP TO DREP.
190			
200		TRAT	TI SECTION.
210			IF SESSO = 1 SUBTRACT 50 FROM AA-TEST GOTO VINGA1
220			ADD 1 TO TOT-M-R
230			ELSE SUBTRACT 60 FROM AA-TEST GOTO VINGA1
240			ADD 1 TO TOT-F-R.

Fig. 5.17f

PROGRAM NAME | PROGRAMMER | DATE

PAGE Nº 006

PROGRAM-ID NUMDIR

LINE	CONT	A	B
010			IF AA < 1 GOTO USCITA-TRATTI.
020			IF AA NOT = 1 GOTO TO X.
030			IF MM NOT > MIMI-TEST GOTO TO USCITA-TRATTI.
040	X.		IF SESSO = 1 ADD 1 TO TOT-M50-R
050			ELSE ADD 1 TO TOT-F40-R.
060		USCI	TA-TRATTI. EXIT.
070			
080		FINE	-REP SECTION.
090			ADD 1 TO CTR
100			IF CTR > 50 WRITE RIGA AFTER SALTO
110			MOVE 1 TO CTR.
120			MOVE DSOC TO SOC-ST
130			MOVE DREP TO REP-ST
140			MOVE TOT-M-R TO TOT-M-ST
150			MOVE TOT-M50-R TO TOT-M50-ST
160			MOVE TOT-F-R TO TOT-F-ST
170			MOVE TOT-F40-R TO TOT-F40-ST
180			WRITE RIGA BEFORE 1 LINE
190			MOVE SPACE TO RIGA
200			ADD TOT-M-R TO TOT-M-ST
210			ADD TOT-M50-R TO TOT-M50-ST
220			ADD TOT-F-R TO TOT-F-ST
230			ADD TOT-F40-R TO TOT-F40-ST
240			MOVE ZERO TO TOTR.

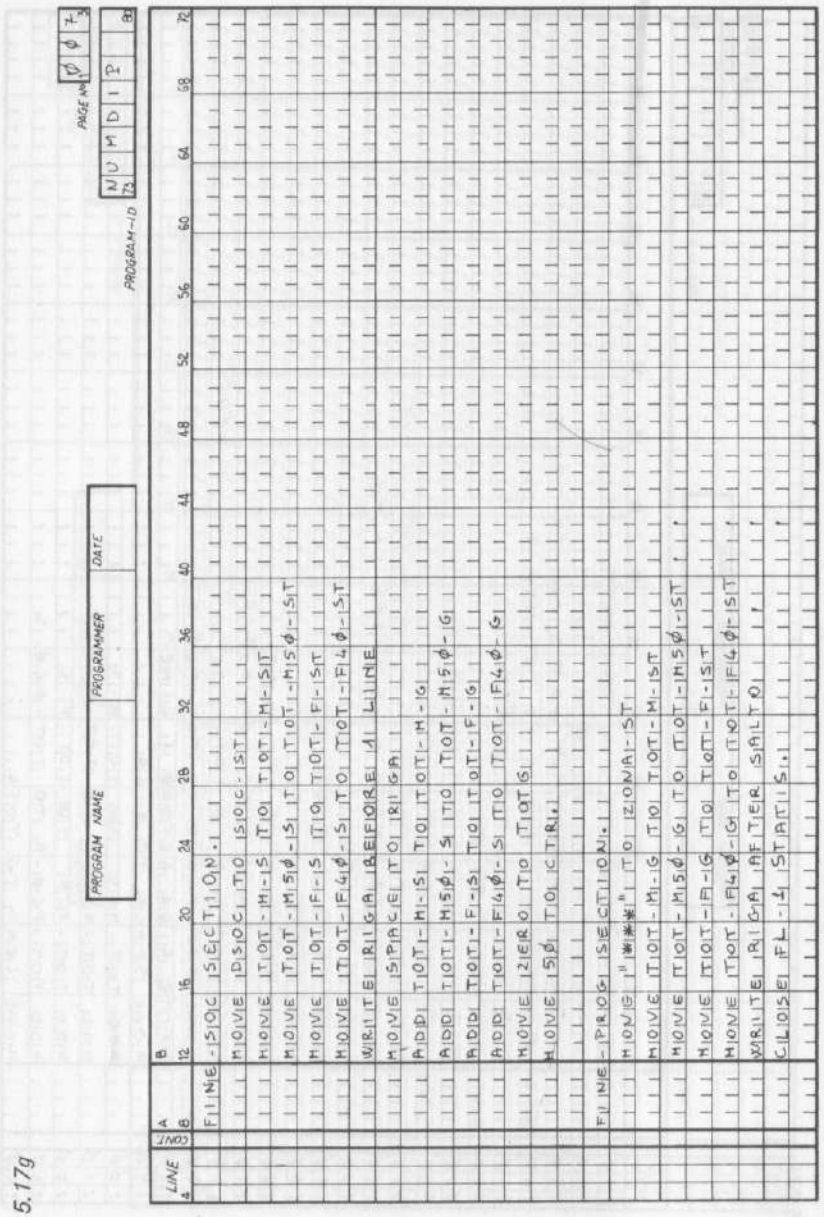


Fig. 5.17g

6. PRECISAZIONI SULLA DESCRIZIONE DEI DATI

In 4 abbiamo visto che il Cobol inquadra i campi dati in categorie (in base al tipo di informazioni contenute) e che le categorie di appartenenza sono individuate dalle PICTURE.

Abbiamo fornito alcune informazioni di base sulle categorie

a. numerica
b. alfabetica
c. alfanumerica

che, come sappiamo, sono individuate rispettivamente dai simboli

9 A X

Spingiamo oltre la trattazione completando e puntualizzando le possibilità descrittive delle PICTURE ed introduciamo alcuni utili clausole di descrizione dati.

6.1. PICTURE numeriche

Esaminiamo le possibilità offerte dal Cobol per la definizione delle PICTURE numeriche relative alla descrizione di campi atti a contenere numeri con segno e/o numeri con parte decimale: in questo paragrafo non vengono presi in considerazione problemi di edizione (output su stampante) che sono esaminati in 6.2.

6.1.1. *Il simbolo "S"*

Abbiamo visto che, per la preparazione di PICTURE numeriche relative a campi atti a contenere numeri interi da trattare dal punto di vista aritmetico, il Cobol consente di utilizzare il simbolo 9 e che una descrizione del tipo

PICTURE 9(n)

permette di definire un campo numerico di lunghezza "n" (che può contenere quindi "n" cifre).

Se si vuole indicare un campo numerico lungo "n" atto a contenere un numero con segno si deve scrivere

PICTURE S9(n)

questo consente di definire in memoria un campo lungo "n" in cui non solo verrà inserito il valore assoluto del numero, ma anche il segno, secondo le modalità previste dal software dell'elaboratore che si utilizza.

Normalmente il segno è associato alla cifra meno significativa.

Una PICTURE contenente il simbolo "S" non può essere utilizzata per la descrizione di un sottocampo dell'area record di un flusso in stampa.

6.1.2. *Il simbolo "V"*

Se si vuole definire un campo numerico atto a contenere un numero con "m" cifre nella parte intera ed "n" nella parte decimale, si deve scrivere

PICTURE 9(m)V9(n)

in questo caso viene riservata un'area di memoria di lunghezza "m+n" e il compilatore predispone affinché il contenuto di quell' area sia considerato come un numero composto da parte intera di "m" cifre e parte decimale di "n" cifre; poiché il trattamento avviene come se la virgola fosse effettivamente presente il simbolo "V" viene chiamato virgola virtuale.

6.1.3. *Esempi ed osservazioni*

a. Naturalmente i simboli "S" e "V" possono essere usati insieme nella definizione di un campo e pertanto il formato generale della descrizione è:

PICTURE S9(m)V9(n)

b. Osserviamo che

Per contenere i valori	Sono necessarie almeno le PICTURE
561	9(3)
- 7	S9
-5144	S9(4)
31,6	99V9
-51,812	S99V999
-,33	SV99

c. Avere a disposizione uno strumento che consente di definire campi atti a contenere numeri con segno consente di ampliare il significato e le possibilità operative delle istruzioni aritmetiche (vedi 5.4), infatti, ad esempio, non si pongono più problemi in relazione a risultati negativi di sottrazioni.

d. Ricordiamo che la virgola virtuale viene riconosciuta ed utilizzata per gli incolonnamenti quando vengono eseguite le istruzioni aritmetiche, ed inoltre il risultato di una operazione viene trasferito nel campo ricevente rispettando l'incolonnamento in relazione alla posizione della virgola virtuale (l'incolonnamento viene fatto a destra se il campo ricevente è numerico intero).

Nel caso in cui il campo ricevente del risultato di una operazione di trasferimento oppure di una operazione aritmetica non consenta di inserire tutte le cifre decimali previste si ha un arrotondamento per difetto (vengono cioè trascurate le cifre decimali eccedenti).

Esempio:

Partendo dai seguenti contenuti iniziali:

03V16 : A PIC 99V99
0V3 : B PIC 9V9

l'istruzione:

MULTIPLY B BY A

consente di ottenere:

00V94 : A
0V3 : B

il risultato dell'operazione è 0,948 ma vengono trasferiti in A solo due decimali.

e. Se vogliamo introdurre da schede dei valori numerici con segno è sufficiente eseguire le perforazioni come se si trattasse di introdurre dei valori assoluti ed inoltre, nel caso di numeri negativi, bisogna perforare sulla colonna della cifra meno significativa un identificativo del segno meno "–".

f. Quando si deve predisporre un input con valori numerici non interi è sufficiente definire la PICTURE del campo ricevente con la virgola virtuale in posizione opportuna ed eseguire le perforazioni come se la virgola non esistesse: al momento della lettura del record fisico il dato viene trasferito in memoria ed interpretato secondo la descrizione fornita nella PICTURE.

6.2. PICTURE di edizione

Il linguaggio Cobol mette a disposizione per la preparazione dei dati da stampare, una serie di caratteri che vengono utilizzati per la scrittura delle PICTURE descrittive dei campi di output:

a. "." e ","
b. "Z"
c. "0"
d. "+" e "–"
e. "B"
f. "$"
g. "*"
h. "CR" e "DB"
i. "/"

6.2.1. *I simboli "." e ","*

In nomenclatura anglosassone il simbolo punto viene utilizzato come separatore fra la parte intera e la parte decimale di un numero, il simbolo virgola viene invece utilizzato come separatore per le migliaia, i milioni, i miliardi, ecc...

I due simboli punto e virgola possono essere utilizzati, con lo stesso significato, nella preparazione di PICTURE di edizione numeriche e ciascuno di questi simboli occupa una posizione di stampa.

Il punto decimale utilizzato nella preparazione di PICTURE di edizione viene riconosciuto per gli incolonnamenti numerici.

Non è possibile inserire il punto decimale all'estrema destra di una PICTURE.

Esempi:

a.

PICTURE interna	contenuto	PICTURE di output	output
9(3)	012	99,999	00,012
9(3)V99	01234	999.9	012.3
9(4)V9	01234	9,999.99	0,123.40
9V9	78	99.9	07.8
V9	5	9	0

b. Supponiamo di volere portare in output un numero intero di 3 cifre e di volere anche il punto decimale.

Poiché non è possibile scrivere una PICTURE del tipo 999. dovremo predisporre l'output di due campi contigui, il primo di PICTURE 999 contenente il numero da stampare ed il secondo di categoria alfanumerica (PICTURE X) contenente il carattere punto.

Se invece della nomenclatura anglosassone vogliamo stampare i numeri, utilizzando la nostra simbologia corrente ("," virgola decimale e "." separatore fra migliaia, milioni, miliardi, ecc...), è sufficiente "informare" il compilatore inserendo negli SPECIAL-NAMES la frase

DECIMAL-POINT IS COMMA.

In questo caso il significato dei simboli "." e "," è automaticamente scambiato a tutti gli effetti, cioè sia nella costruzione della PICTURE sia per quanto riguarda l'output.

Esempio: esaminiamo il seguente schema supponendo che negli SPECIAL-NAMES sia stata introdotta la

DECIMAL-POINT IS COMMA.

PICTURE interna	contenuto	PICTURE di output(*)	output
9(3)	012	99.999	00.012
9(3)V99	01234	999,9	012,3
9(4)V9	01234	9.999,99	0.123,40
9V9	78	99,9	07,8
V9	5	9	0

In tutti gli esempi che seguiranno è sottinteso che utilizzeremo la nomenclatura latina e quindi che negli SPECIAL-NAMES è stata introdotta la frase DECIMAL-POINT IS COMMA.

6.2.2. *Il simbolo "Z"*

Come abbiamo potuto osservare dagli esempi di 6.2.1. l'uso del simbolo 9 per la formazione delle PICTURE numeriche di edizione, può portare ad inconvenienti di tipo estetico in quanto vengono stampati anche gli zeri non significativi.

Il simbolo "Z" è usato nell'ambito di PICTURE numeriche di edizione per indicare che le posizioni corrispondenti del campo non devono essere visualizzate se contengono zeri o separatori non significativi: per separatori non significativi intendiamo le virgole, in nomenclatura anglosassone, e i punti in nomenclatura latina, quando sono preceduti solo da zeri non significativi.

Il simbolo "Z" consente di sopprimere dall'output solo gli zeri non significativi relativi alla parte intera del numero.

Esempio:

PICTURE interna	contenuto	PICTURE di output	output
9(5)	03595	ZZ.ZZZ	3.595
9(5)	00123	ZZ.ZZZ	123
9(3)V99	02345	ZZZ.ZZ9,9	23,4
V99	34	Z99,999	00,340
9(5)	12357	ZZZ.ZZ9,9	12.357,0
V9	0	ZZ,Z	,0
V9	0	99,Z	00,0

6.2.3. *Il simbolo "0"*

Il simbolo zero "0" è utilizzato nell'ambito di PICTURE relative a campi numerici editati per rappresentare le posizioni di un campo di output in cui deve essere inserito il carattere "0" indipendentemente dal contenuto del campo che deve essere portato in output.

Esempio:

PICTURE interna	contenuto	PICTURE di output	output
9(5)V99	0012345	999099,9000	001023,4000
9(4)	1234	9090	3040
9(5)	00012	99000	12000

6.2.4. *Il simbolo "$"*

Questo simbolo può essere utilizzato nell'ambito di PICTURE numeriche editate come simbolo monetario. Sono previste due possibilità di uso:

a. in posizione fissa all'estrema sinistra della PICTURE,

b. in modo ripetuto da sinistra a destra: in questo caso avviene la soppressione di zeri e separatori non significativi ed il simbolo monetario è stampato nella posizione immediatamente precedente la prima cifra del numero, o per lo meno il più a destra possibile.

Esempio:

PICTURE interna	contenuto	PICTURE di output	output
9(4)	1234	$ZZZ9	$1234
9(4)	0005	$ZZZ9	$ 5
9(4)	0005	$$$$$	$5
9(4)	0005	$$$ZZ	$ 5
9(4)	1234	$$.$$$	$1.234
9(4)	0123	$$.$$$	$123

Il Cobol offre la possibilità di sostituire al simbolo monetario "$" un altro simbolo da scegliersi fra i seguenti caratteri:

E F G H I M N O Q
T U Y J W K = > <

Possiamo osservare che questo insieme di caratteri non contiene il simbolo "L" e quindi ha per noi scarso interesse il fatto che il simbolo monetario possa essere utilizzato direttamente nella preparazione delle PICTURE.

In ogni caso se si vuole operare la sostituzione bisognerà indicare (negli SPECIAL-NAMES) la frase

CURRENCY SIGN IS costante

dove "costante" è un simbolo appartenente alla tabellina di valori ora presentata; "costante", se compare negli SPECIAL-NAMES la frase CURRENCY SIGN..., sostituisce a tutti gli effetti il simbolo monetario dollaro "$".

6.2.5. *I simboli "+" e "–"*

I simboli più "+" e meno "–" sono utilizzati nell'ambito di PICTURE numeriche di edizione per produrre in output il segno di un dato numerico.

Il simbolo più "+" permette di visualizzare il segno sia di numeri positivi che di numeri negativi; il simbolo "–" consente di visualizzare il segno solo nel caso di numeri negativi, se il numero è positivo la posizione corrispondente al segno viene lasciata a spazio (b).

I simboli di edizione del segno possono essere utilizzati secondo diverse modalità:

a. alla destra del numero: il simbolo prescelto deve essere indicato immediatamente dopo la posizione che viene occupata dalla cifra meno significativa.
Esempio:

PICTURE interna	contenuto	PICTURE di output	output
S9(4)	1234 pos.	ZZ.ZZZ+	1.234+
S9(4)	1234 neg.	ZZ.ZZZ+	1.234–
S9(4)V9	12345 pos.	ZZ.ZZZ,9–	1.234,5
S9(4)V9	12345 neg.	ZZ.ZZZ,9–	1.234,5–

b. alla sinistra del numero in posizione fissa: il simbolo relativo al segno deve essere scritto come primo elemento della stringa di caratteri che rappresenta la PICTURE del campo di output nella posizione prescelta; la determinazione della posizione in cui deve essere stampato il segno è effettuata dimensionando opportunamente il numero delle posizioni da assegnare all'output del dato.
Esempio:

PICTURE interna	contenuto	PICTURE di output	output
S9(4)	1234 pos.	+Z.ZZZ	+1.234
S9(4)	1234 neg.	+Z.ZZZ	– 1.234
S9(4)	1234 pos.	+ZZ.ZZZ	+ 1.234
S9(4)	1234 neg.	+ZZ.ZZZ	– 1.234
S9(4)	1234 pos.	–Z.ZZZ	1.234
S9(4)	1234 neg.	–Z.ZZZ	– 1.234
S9(4)	1234 pos.	–ZZ.ZZZ	1.234
S9(4)	1234 neg.	–ZZ.ZZZ	– 1.234

c. alla sinistra del numero in posizione ripetuta: in questo caso si ha la soppressione degli zeri e dei separatori non significativi corrispondenti alle posizioni della PICTURE che contengono il simbolo relativo al segno; il segno viene stampato immediatamente prima della prima cifra significativa oppure, se questo non è possibile per il tipo di PICTURE, la stampa del segno è fatta il più a destra possibile.

Esempio:

PICTURE interna	contenuto		PICTURE in output	output
S9(7)	0000123	pos.	++.+++.+++	+123
S9(7)	0000123	neg.	++.+++.+++	-123
S9(7)	0000123	pos.	--.---.---	123
S9(7)	0000123	neg.	--.---.---	-123
S9(7)	0001234	pos.	++.+++.+++	+1.234
S9(7)	0001234	neg.	++.+++.+++	-1.234
S9(7)	0001234	pos.	--.---.---	1.234
S9(7)	0001234	neg.	--.---.---	-1.234
S9(7)	0001234	pos.	++.+ZZ . ZZZ	+ 1.234
S9(7)	0001234	neg.	++.+ZZ . ZZZ	- 1.234
S9(7)	0001234	pos.	--.-ZZ . ZZZ	1.234
S9(7)	0001234	neg.	--.-ZZ . ZZZ	- 1.234
S9(7)	0012345	pos.	++.+ZZ . ZZZ	+12.345
S9(7)	0012345	neg.	++.+ZZ . ZZZ	-12.345
S9(7)	0012345	pos.	--.-ZZ . ZZZ	12.345
S9(7)	0012345	neg.	--.-ZZ . ZZZ	-12.345
S9(7)	1234567	pos.	++.+ZZ . ZZZ	+1.234.567
S9(7)	1234567	neg.	++.+ZZ . ZZZ	-1.234.567
S9(7)	1234567	pos.	--.-ZZ . ZZZ	1.234.567
S9(7)	1234567	neg.	--.-ZZ . ZZZ	-1.234.567

6.2.6. *I simboli "CR" e "DB"*

Questi simboli, abbreviazione di credito e debito, vengono utilizzati nell'ambito di PICTURE numeriche di edizione (immediatamente alla destra del simbolo relativo alla cifra meno significativa

del numero) e consentono di evidenziare valori numerici negativi. Se si deve stampare un numero negativo e nella PICTURE è inserito uno dei suddetti simboli anche questo viene stampato (alla destra del numero).

I simboli CR e DB possono occupare una o due posizioni di stampa ciascuno: il caso di una sola posizione è relativo ad elaboratori che prevedono nel set grafico i due simboli speciali "CR" e "DB" graficamente predisposti delle dimensioni di un solo carattere ciascuno.

Esempio:

PICTURE interna	contenuto	PICTURE di output	output
S9(3)	123 pos.	ZZZCR	123
S9(3)	123 neg.	ZZZCR	123CR
S9(3)	123 pos.	ZZZDB	123
S9(3)	123 neg.	ZZZDB	123DB

6.2.7. *Il simbolo "*"*

Il simbolo asterisco "*", ha, nell'ambito di PICTURE numeriche di edizione, la stessa funzione che ha il simbolo "Z" con questa differenza: gli zeri ed i separatori non significativi che si trovano in corrispondenza di zone del campo di output descritte dal simbolo asterisco vengono sostituiti con il simbolo asterisco.

Esempio:

PICTURE interna	contenuto	PICTURE di output	output
9(3)	055	**9	*55
9(3)	055	*.**9	***55
9(4)	4455	***.**9	**4.455

6.2.8. *Il simbolo "B"*

Questo è un simbolo di inserimento che può essere utilizzato per la preparazione di PICTURE di edizione sia numeriche che alfanumeriche: le posizioni che nella stringa di caratteri della PICTURE sono occupate da questo simbolo saranno a "blank" (spazio) in output.

La funzione del simbolo "B" è essenzialmente di carattere estetico infatti il suo inserimento consente di distanziare fra loro caratteri o gruppi di caratteri.

Esempio:

PICTURE interna	contenuto	PICTURE di output	output
X(4)	ROMA	XBXBXBX	R O M A
X(5)	AXT32	XXXBXX	AXT 32
9(4)	1234	ZBZBZBZ	1 2 3 4
9(4)	0123	ZBZBZBZ	1 2 3

6.2.9. *Il simbolo "/"*

Il simbolo barra "/" è un simbolo di inserimento utilizzato per la preparazione di PICTURE numeriche e alfanumeriche di edizione: serve per indicare la posizione del campo di output in cui deve essere inserito il simbolo "/".

Esempio:

PICTURE interna	contenuto	PICTURE di output	output
9(6)	210273	99/99/99	21/02/73
X(6)	12345A	X(5)/X	12345/A

6.3. Riepilogo sulle categorie

Poiché quanto esposto in 6.1 e 6.2 costituisce, con gli elementi forniti in 4.1.3.3, l'insieme delle regole Cobol per la definizione dei dati in termine di categorie, diamo in questo paragrafo un riepilogo degli elementi più importanti.

6.3.1. *Categoria numerica*

Sono di categoria numerica quei campi la cui PICTURE è costituita unicamente dai caratteri

9 S V

Sono da considerare di categoria numerica anche la costante figurativa ZERO ed i literal numerici (es: 57, –205 , 1.3 , –12.5 , –.9 , ecc...).

I campi (o le costanti) che sono operandi in istruzioni aritmetiche devono essere di categoria numerica.

6.3.2. *Categoria alfabetica*

I campi di categoria alfabetica sono quelli in cui la stringa di caratteri della PICTURE è costituita solo da simboli A.

Sono inoltre da considerare di categoria alfabetica la costante figurativa SPACE ed i litteral alfabetici (costanti del programmatore costituite solo da lettere dell'alfabeto inglese ed eventualmente dal carattere spazio).

6.3.3. *Categoria alfanumerica*

Sono di categoria alfanumerica quei campi la cui PICTURE è costituita da qualsiasi combinazione dei simboli

A X 9

(purché non siano tutti "A" o tutti "9"!).

Il campo la cui PICTURE è di tipo misto è trattato come se la PICTURE fosse costituita solo da simboli "X": una PICTURE mista ha funzione documentativa del tipo di dati che dovranno essere inseriti in corrispondenza delle varie posizioni descritte in modo diverso nella stringa di caratteri della PICTURE.

Sono da considerare di categoria alfanumerica i literal alfanumerici e le costanti figurative (tranne ZERO e SPACE).

6.3.4. *Categoria numerica editata*

Sono di categoria numerica editata quei campi la cui PICTURE contiene almeno uno dei caratteri

Z * $, . + − CR DB

eventualmente associati ad uno o più simboli "9".

Sono inoltre di categoria numerica editata quei campi le cui PICTURE contengono almeno uno dei caratteri di inserzione

0 B /

e non contengono i simboli "A" ed "X".

E' da considerare di categoria numerica editata un campo descritto con la clausola

BLANK WHEN ZERO

che presenteremo in 6.4.6.

6.3.5. *Categoria alfanumerica editata*

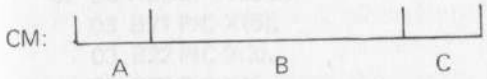
Sono di questa categoria quei campi la cui PICTURE contiene almeno un carattere di inserzione ("B" oppure "/") ed almeno uno dei caratteri "A" oppure "X".

6.4. Clausole speciali di descrizione dati

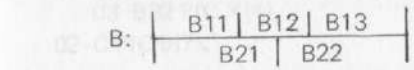
Oltre alle numerose possibilità offerte dai simboli che si possono utilizzare per la costruzione delle PICTURE, che servono a caratterizzare dal punto di vista trattamento il campo oggetto di descrizione, il Cobol prevede una serie di clausole che permettono di specializzare ulteriormente i campi definendo per essi trattamenti opportuni.

6.4.1. *Strutture di un'area di lavoro*

Supponiamo di dover considerare il campo CM



in cui alternativamente il sottocampo B deve essere analizzato in relazione alle due diverse strutture



Se CM è un nome record di un flusso, oppure un sottocampo di un record, sappiamo risolvere il problema in quanto ad ogni livello 01 in FD corrisponde una ridefinizione della stessa area (area record).

Per esempio:

```
FD FLUSSO.....
   .....DATA RECORDS ARE REC REC1.
01 REC.
   02 FILLER PIC X(5).
   02 CM.
      03 A PIC X(10).
      03 B.
         04 B11 PIC 9(3).
         04 B12 PIC 9(3).
         04 B13 PIC 9(5).
      03 C PIC X(9).
   02 FILLER PIC X(25).
01 REC1.
   02 FILLER PIC X(15).
   02 B21     PIC 9(4).
   02 B22     PIC 9(7).
   (02 FILLER PIC X(34).)
```

Se invece CM è un'area di lavoro, gli strumenti di cui disponiamo non ci consentono di risolvere il problema.

La soluzione è offerta dalla clausola REDEFINES:

numero-di-livello nome-dato-1 REDEFINES nome-dato-2.

che nel nostro caso (supponendo che CM sia un'area di lavoro con le caratteristiche del caso precedente) viene così applicata:

```
01 CM.
   02 A PIC X(10).
   02 B.
      03 B11 PIC 9(3).
      03 B12 PIC 9(3).
      03 B13 PIC 9(5).
   02 BB REDEFINES B.
      03 B21 PIC 9(4).
      03 B22 PIC 9(7).
   02 C PIC X(9).
```

Osserviamo che questo procedimento era adottabile anche nella descrizione record precedente in cui era necessaria la ridefinizione del campo B.

La clausola REDEFINES non può essere utilizzata a livello 01 nell'ambito della FILE SECTION in quanto ad ogni livello 01 di una FD corrisponde una ridefinizione dell'area record del flusso.

Oltre al fatto di consentire di strutturare in modo diverso un'area di memoria in relazione alla lunghezza dei campi, la clausola REDEFINES permette di caratterizzare in modi diversi un'area di memoria in relazione alla categoria:

```
01 A PIC X(10).
01 A1 REDEFINES A PIC 9(10).
```

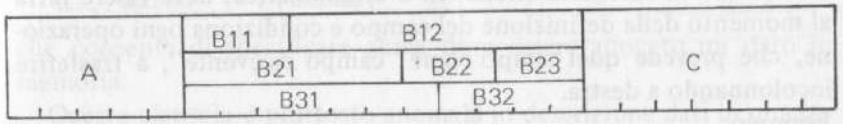
Il Cobol prevede che un'area di memoria possa essere ridefinita più volte: in questo caso bisogna fare riferimento sempre al primo nome.

Esempio

Alla definizione:

```
01 CAMPO.
   02 A PIC X(5).
   02 B.
      03 B11 PIC 99.
      03 B12 PIC X(8).
   02 BB REDEFINES B.
      03 B21 PIC X(5).
      03 B22 PIC 9(3).
      03 B23 PIC XX.
   02 BBB REDEFINES B.
      03 B31 PIC X(6).
      03 B32 PIC X(4).
   02 C PIC 9(7).
```

corrisponde la struttura:



Ricordiamo che la ridefinizione di un campo deve essere fatta immediatamente dopo la fine della definizione del campo stesso e che si deve utilizzare lo stesso numero di livello.

Quando si vuole inizializzare un'area di lavoro, la clausola VALUE deve essere associata alla prima definizione di un campo e non ad eventuali ridefinizioni.

6.4.2. *Posizionamento di un dato in un campo di categoria alfabetica o alfanumerica*

Supponiamo di avere i seguenti campi elementari:

```
..... A PIC X(4) VALUE "ROMA".
..... B PIC X(6).
```

l'istruzione:

MOVE A TO B

permette di ottenere come contenuto del campo B:

ROMAbb

infatti le modalità del trasferimento sono funzione della categoria del campo ricevente e, un campo ricevente alfanumerico (o alfabetico) viene "caricato" dall'operazione di trasferimento "da sinistra verso destra".

Il Cobol dispone di una clausola di descrizione dati che consente di eseguire i trasferimenti in campi alfabetici o alfanumerici incolonnando il dato da trasferire alla destra del campo ricevente:

$$\left\{\begin{matrix}\underline{\text{JUSTIFIED}}\\ \underline{\text{JUST}}\end{matrix}\right\}\ \text{RIGHT}$$

L'attribuzione della JUST, che può essere fatta solo ad un campo elementare di categoria alfabetica o alfanumerica, deve essere fatta al momento della definizione del campo e condiziona ogni operazione, che prevede quel campo come "campo ricevente", a trasferire incolonnando a destra.

Esempio:

Assegnati i due campi:

.... A PIC X(4) VALUE "ROMA".
.... B PIC X(6) JUST RIGHT.

l'istruzione:

MOVE A TO B

consente di ottenere in B:

bbROMA

Utilizzando la clausola JUST i dati trasportati in B vengono incolonnati da destra con riempimento di spazi oppure con troncamento a sinistra; se una stessa area la si volesse utilizzare con le modalità previste da questa clausola ed anche in modo usuale bisognerebbe provvedere ad una ridefinizione.

Esempio

Contenuto del campo emittente	Contenuto del campo ricevente: X(6) con clausola	senza clausola
ABC	bbbABC	ABCbbb
ABCD	bbABCD	ABCDbb
ABCbb	bABCbb	ABCbbb
ABCDEFG	BCDEFG	ABCDEF
ABCDEFbbb	DEFbbb	ABCDEF

6.4.3. *Limitazioni nella memorizzazione di un dato in un campo*

In Cobol è disponibile la clausola

$$\left\{\begin{matrix}\underline{\text{SYNCHRONIZED}}\\ \underline{\text{SYNC}}\end{matrix}\right\}\left[\begin{matrix}\underline{\text{LEFT}}\\ \underline{\text{RIGHT}}\end{matrix}\right]$$

che consente di specificare come deve essere allocato un dato in memoria.

Questa clausola è piuttosto anomala in descrizione dati in quanto si riferisce in modo specifico all'hardware dell'elaboratore e precisamente è significativa per elaboratori con memorie ad alto parallelismo.

Come sappiamo esistono elaboratori con memoria strutturata in modo tale da contenere in ogni parola di memoria la rappresentazione di un solo carattere del set grafico ed elaboratori con memoria ad alto parallelismo, in grado cioè di contenere in una parola di memoria la rappresentazione di più caratteri.

Nei primi la allocazione di un nome-dato in memoria impegna un "certo" numero di parole, mentre nei secondi possono essere impegnate anche delle frazioni di parole di memoria, in tal caso il sistema operativo può utilizzare le frazioni residue per la definizione di altri campi.

Se attribuiamo ad un nome dato la specifica SYNCHRONIZED vincoliamo il compilatore ad utilizzare per la definizione del campo un numero intero di parole (una o più in relazione alla lunghezza del campo e al parallelismo).

In particolare l'attribuzione alternativa delle due parole opzionali LEFT o RIGHT precisa che il dato deve essere definito allineandolo a sinistra o a destra nell'insieme di parole che devono essergli riservate; in assenza di tali specifiche l'allineamento è quello standard previsto dal compilatore.

Per elaboratori con parole atte a contenere un solo carattere la clausola non è significativa e può eventualmente essere utilizzata a scopo documentativo.

6.4.4. *Il segno nei campi di categoria numerica*

Abbiamo visto che per la definizione dei campi numerici segnati viene utilizzata la PICTURE

S9(n)

e che, con una PICTURE di questo tipo, il segno non occupa una posizione ma viene associato, secondo modalità previste dal software al campo atto a contenere il numero.

Utilizzando la clausola:

SIGN IS {LEADING / TRAILING}

il segno viene associato rispettivamente alla prima o all'ultima posizione del numero.

Completando la clausola con

SEPARATE CHARACTER

si precisa che il segno deve occupare una posizione che sarà all'estrema sinistra o all'estrema destra del campo a seconda che questo sia di tipo LEADING oppure TRAILING. La posizione riservata al segno contiene in tal caso uno dei due simboli "+" oppure "–" ed il campo continua ad essere di categoria numerica: si ha il vantaggio che, oltre poterlo trattare dal punto di vista aritmetico, lo si può portare direttamente in stampa.

Visualizzando il segno in questo modo non si può effettuare la soppressione degli zeri non significativi e l'edizione dei simboli "." e "," che si ottengono con le specifiche PICTURE di edizione.

6.4.5. *Edizione dei campi con valore zero*

La clausola

BLANK WHEN ZERO

che può essere utilizzata solo nell'ambito di un campo con PICTURE numerica o numerica editata, permette di inserire spazi in un dato elementare il cui valore è zero.

Questa clausola prevale su tutte le regole di edizione specificate nelle PICTURE ad eccezione della soppressione zeri mediante asterischi.

Quando la clausola è utilizzata per un campo di PICTURE numerica, la categoria del campo diventa numerica editata.

Esempio:

Se si porta in stampa il valore "zero" in un campo descritto con PICTURE:

9,99

si ottiene:

0,00

se invece la PICTURE è:

9,99 BLANK WHEN ZERO

non si ottiene alcun risultato in stampa in quanto le quattro posizioni di output relative al dato sono riempite da spazi.

6.4.6. *Assegnazione di nomi*

Il linguaggio Cobol consente di identificare un'area di memoria con nomi diversi senza dover procedere alla ridefinizione (ricordiamo che REDEFINES è una clausola di "struttura").

Se ad un'area già definita:

..... AREA PIC ....

si vuole assegnare un ulteriore nome (ad esempio ABCD) basterà scrivere

66 ABCD RENAMES AREA.

Se si vogliono attribuire ad una stessa area di memoria più nomi è sufficiente utilizzare più RENAMES consecutive a livello 66 facendo riferimento sempre al nome assegnato per primo.

L'aspetto della RENAMES che abbiamo ora analizzato si riferisce solo ad una parte del formato ammesso per questa clausola; il formato completo è:

```
66 nome-dato RENAMES nome-dato-1 [THRU nome-dato-2].
```

La clausola nel suo formato completo consente di assegnare un nome mnemonico (di gruppo) ad un insieme di campi consecutivi di cui il primo è nome-dato-1 e l'ultimo è nome-dato-2.

Esempio:

```
01 CAMPO.
   02 A PIC ...
   02 B PIC ...
   02 C PIC ...
   02 D PIC ...
   02 E PIC ...
66 AREA RENAMES B THRU D.
```

equivale a:

```
01 CAMPO.
   02 A PIC ...
   02 AREA.
      03 B PIC ...
      03 C PIC ...
      03 D PIC ...
   02 E PIC ...
```

La clausola RENAMES non può fare riferimento ai livelli 01, 77 e 88 (vedi 6.4.7.) di WORKING-STORAGE SECTION.

Se la clausola è utilizzata per assegnare un nome ad un gruppo di campi, questi devono fare parte di uno stesso record logico.

Tutti i livelli 66 associati ad una descrizione di un record devono essere riportati dopo la descrizione dell'ultimo campo del record.

Non è possibile associare ad un livello 66 alcuna altra clausola di descrizione dati.

6.4.7. *Assegnazione di nomi-condizione*

In Cobol è possibile associare un nome mnemonico al fatto che una condizione sia verificata oppure no:

```
88 nome-condizione { VALUE IS   } costante1 [THRU costante2]
                   { VALUES ARE }
   [costante3 [THRU costante4]]...
```

I livelli 88 possono fare riferimento solo a campi elementari e, se utilizzati, devono seguire immediatamente la descrizione del campo.

Ad uno stesso campo possono essere associati più livelli 88.

Esempio:

Sia C un campo di PICTURE 99, si vogliono individuare, per poterli trattare diversamente i seguenti casi

CASO-1 : 7 , 25÷32 , 74÷81
CASO-2 : 11 , 18÷24 , 82÷99
CASO-3 : i rimanenti valori compresi fra 0 e 99.

Basterà, per raggiungere questo scopo, definire il campo C e, subito dopo, predisporre opportuni livelli 88:

```
01 C PIC 99.
   88 CASO-1 VALUES ARE 7 25 THRU 32
      74 THRU 81.
   88 CASO-2 VALUES ARE 11 18 THRU 24
      82 THRU 99.
```

e, per individuare il caso che interessa, basterà, in PROCEDURE DIVISION scrivere:

```
IF CASO-1 ..... .
IF CASO-2 ..... .
.......................
```

Osserviamo che disporre del livello 88 è particolarmente utile anche nel momento della preparazione del flow-chart che risulta particolarmente lineare se i test vengono fatti sui "nomi-condizione": nelle figg. 6.1 e 6.2 sono riportati i flow-chart che visualizzano

Fig. 6.1

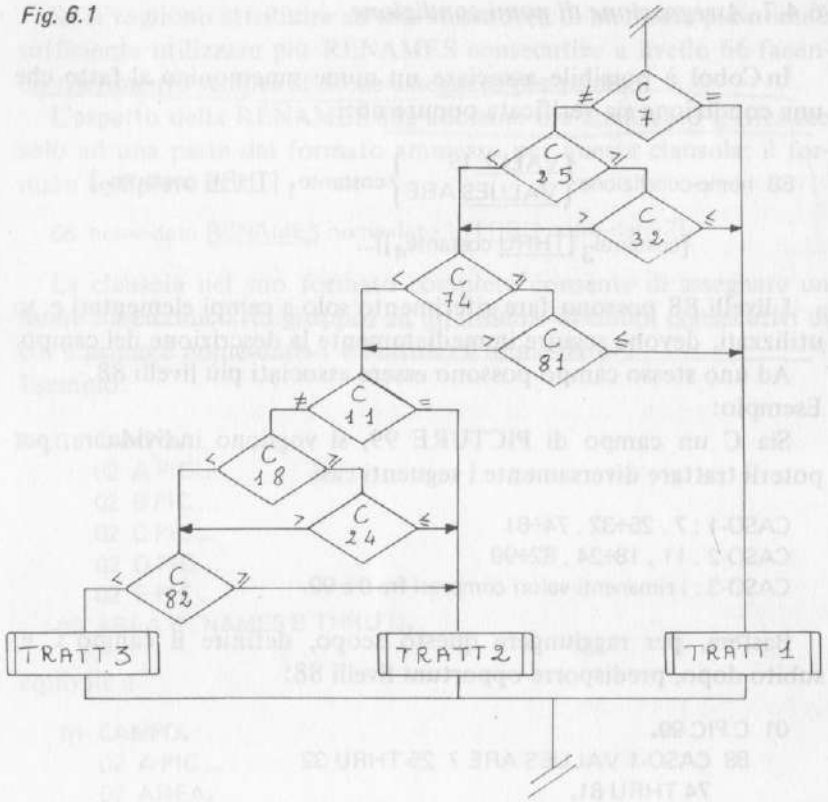
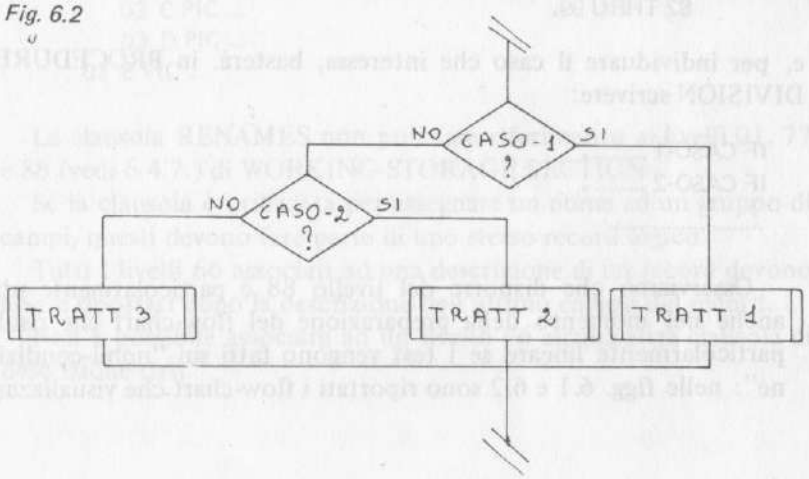


Fig. 6.2



l'esempio che abbiamo presentato nel caso in cui non si disponga oppure si disponga del livello 88.

Ricordiamo che il livello 88 non può contenere altre clausole di descrizione dati.

6.4.8. *Il formato dei dati in memoria*

Nella presentazione delle specifiche e delle clausole di descrizione dati non abbiamo in generale fatto riferimento alle rappresentazioni dei dati in memoria, ma ci siamo sempre limitati a considerare un nome-dato come un'area atta a contenere una sequenza di caratteri.

L'atteggiamento che abbiamo assunto è quello che corrisponde all'assegnazione al nome-dato di una

USAGE IS DISPLAY

che fra l'altro può essere omessa.

Il Cobol prevede che la USAGE di un campo possa essere anche diversa al fine di specificare che un dato numerico può essere trattato in modo diverso che non "a caratteri".

In relazione a diversi tipi di elaboratori può essere previsto che i dati numerici vengano trattati in modo binario oppure che vengano considerati impaccati: le indicazioni di USAGE in questi casi sono diverse e non previste dallo standard che indica un generico

$$\text{USAGE IS} \left\{ \begin{array}{l} \text{COMPUTATIONAL} \\ \text{COMP} \end{array} \right\}$$

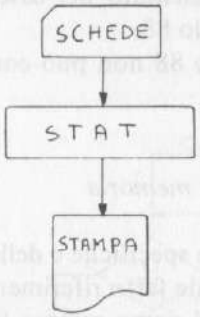
dove COMPUTATIONAL e COMP sono equivalenti.

In 8. incontreremo un altro tipo di USAGE relativo ad aree di memoria che devono essere trattati come lo stesso formato degli indici definiti in modo implicito per tabelle trattate con il metodo indexing.

6.5. Esempio

In fig. 6.3 è riportato l'organigramma del problema che si vuole risolvere; nelle figg. 6.4 e 6.5 sono rappresentati rispettivamente il

Fig. 6.3



tracciato dei record da elaborare ed il formato pagina del tabulato che si vuole ottenere in output.

Analizziamo gli elementi caratteristici relativi ai dati in input, ai risultati che si vogliono ottenere ed al tipo di elaborazione da eseguire.

a. CODART: il codice articolo è costituito da due parti, la prima (COD) è il codice vero e proprio, la seconda (CV) è un controllo di validità del codice; si ritiene che un codice è valido se CV rappresenta il resto della divisione di COD per 97; si vogliono trattare quei record validi in cui i due caratteri centrali di COD, che è lungo 6, rappresentano il numero 10, oppure il numero 25, oppure un numero compreso tra 30 e 50 (estremi inclusi).
b. DESCR: è la descrizione dell'articolo.
c. QV: rappresenta la quantità venduta nell'anno è costituita da 3 interi e 2 decimali, i decimali indicano una frazione dell'unità "confezione".
d. COST-UN: costo unitario, 4 interi.
e. PR-AS: prezzo unitario di alta stagione, 4 interi.
f. C-AS: coefficiente di vendita in alta stagione, 1 intero e 2 decimali.
g. PR-BS: prezzo unitario di bassa stagione, 4 interi.
h. C-BS: coefficiente di vendita di bassa stagione, 1 intero e 2 decimali.
i. Le aree di output del codice articolo, della descrizione, della quantità venduta e del costo unitario hanno le stesse dimensioni dei campi corrispondenti del flusso in input.

l. Il prezzo medio è calcolato dalla formula:

(PR-AS * C-AS + PR-BS * C-BS) / 2

m. Il guadagno deve essere espresso, con segno, in migliaia, il valore massimo della parte da considerare è esprimibile mediante 4 interi.
n. Ogni modulo deve essere intestato e contiene al più 50 righe dettaglio.
o. Alla fine dell'elaborazione, su nuovo modulo, deve essere stampata la dicitura "SCARTI" seguita dalla somma algebrica delle quantità di guadagno che sono state trascurate nell'arrotondamento alle migliaia.
p. Nella stampa evidenziare le virgole decimali ed i punti di separazione dei vari ordini di grandezza.

In fig. 6.6 è riportato il flow-chart relativo al problema in esame ed in fig. 6.7 la relativa codifica.

Osserviamo l'uso del livello 88 per "isolare" gli elementi (con CODART valido) da trattare e la ridefinizione del campo COD che deve essere considerato di categoria numerica per il controllo di validità del codice e nel contempo deve essere sottolivellato per poter individuare i record da trattare.

Come si può osservare dal flow-chart, sono stati omessi una serie di controlli (in pratica necessari) per evitare di appesantire l'esempio centrato sul problema della descrizione dei dati.

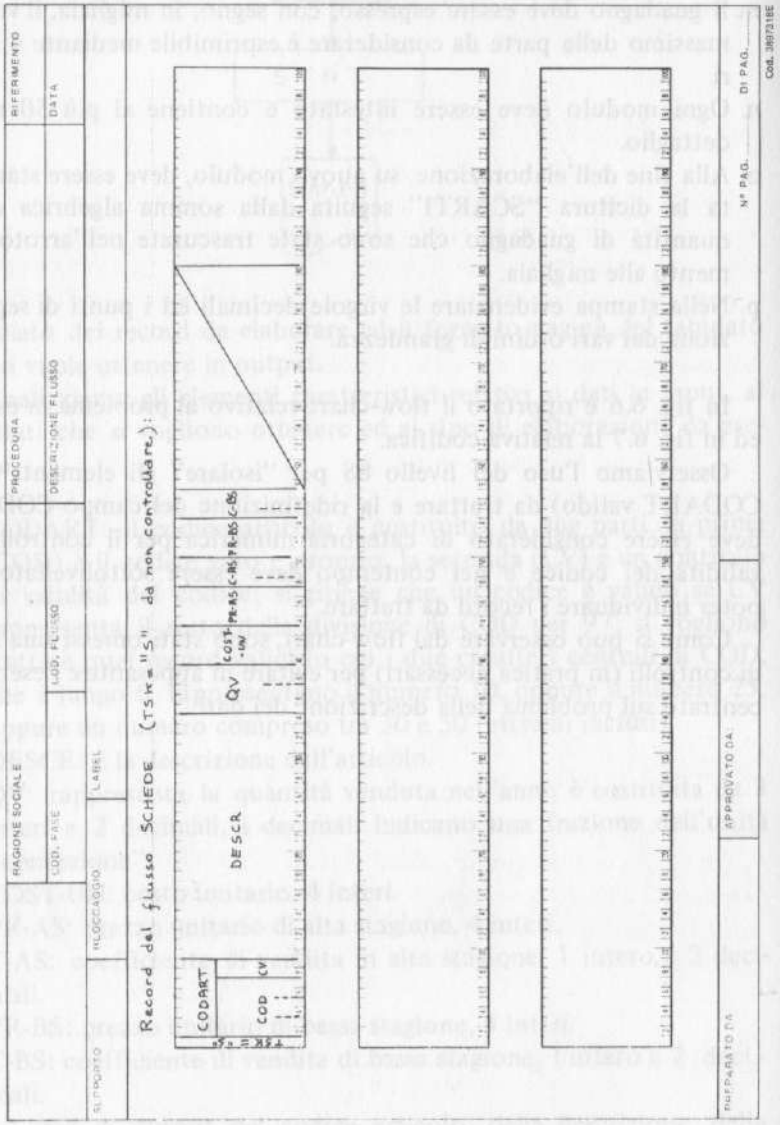

RAGIONE SOCIALE
PROCEDURA
RIFERIMENTO
COD. FASE
LOD. FLUSSO
DESCRIZIONE FLUSSO
DATA
SUPPORTO
MECCANOGRAFIO
LABEL
Record del flusso SCHEDE (TSK = "S" da non controllare)
CODART
COD CV
DESCR
QV
COST -UV
PR-AS
C-AS
PR-BS
C-BS
PREPARATO DA
APPROVATO DA
N° PAG.
DI PAG.
Cod. 3897318E

Fig. 6.4

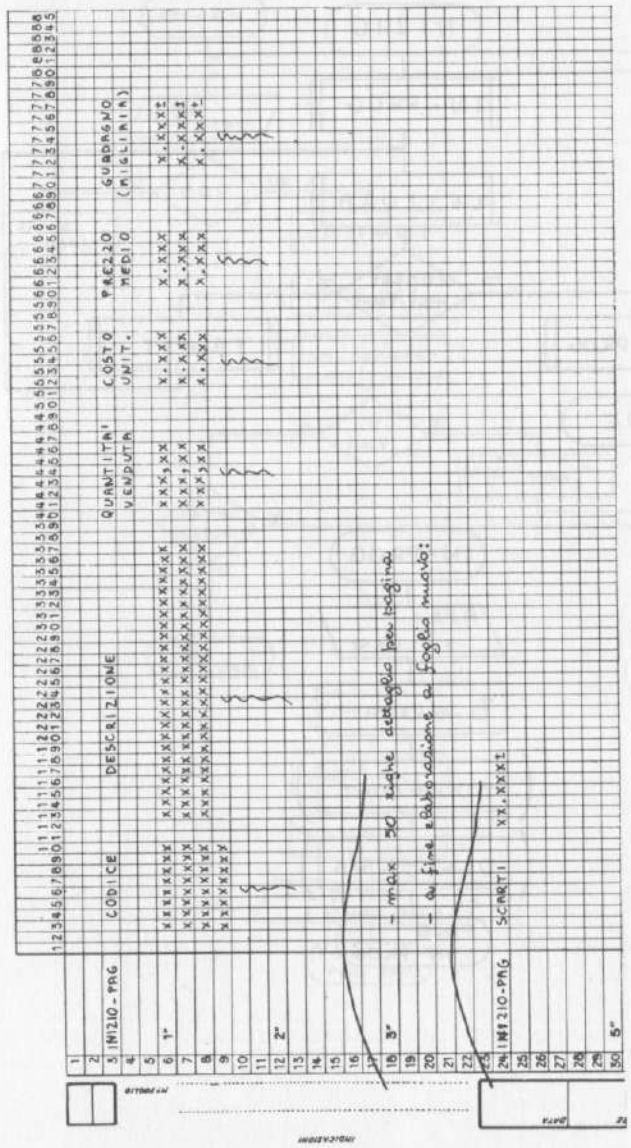

CODICE
DESCRIZIONE
QUANTITA' VENDUTA
COSTO UNIT.
PREZZO MEDIO
GUADAGNO (MIGLIAIA)
INIZIO-PAG
SCARTI
INDICAZIONI

Fig. 6.5

Fig. 6.6a

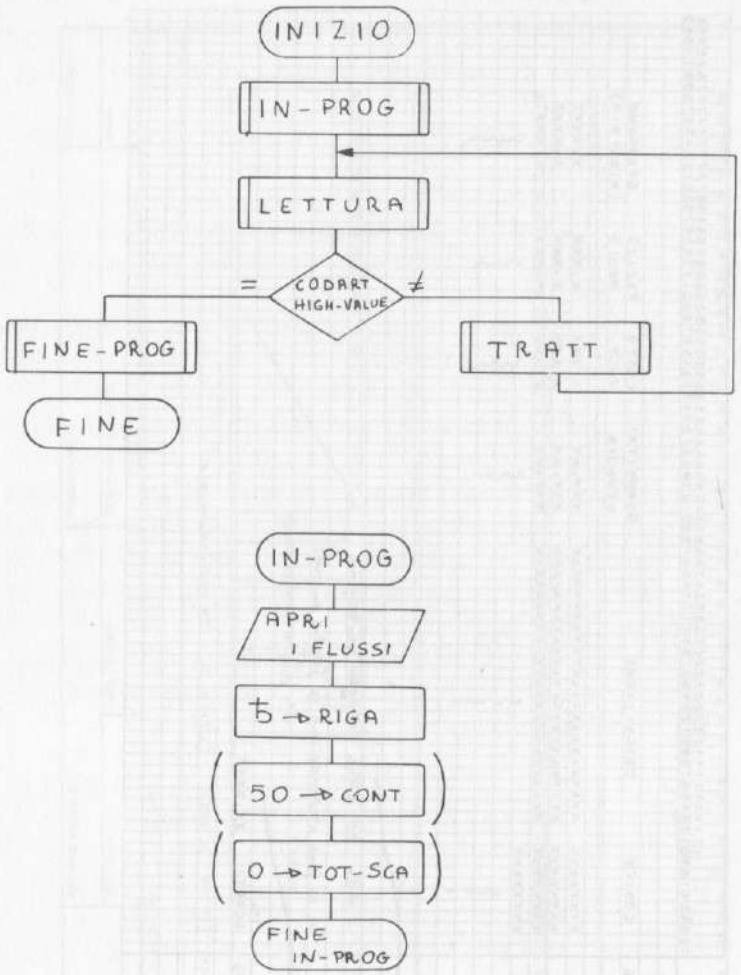


Fig. 6.6b

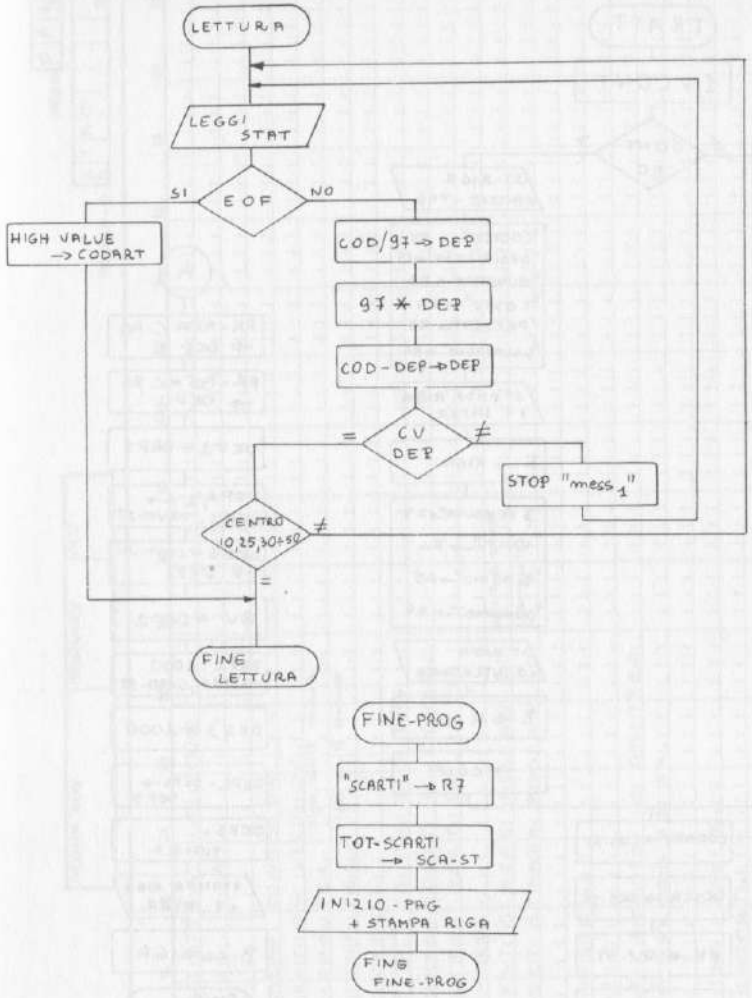


Fig. 6.6c

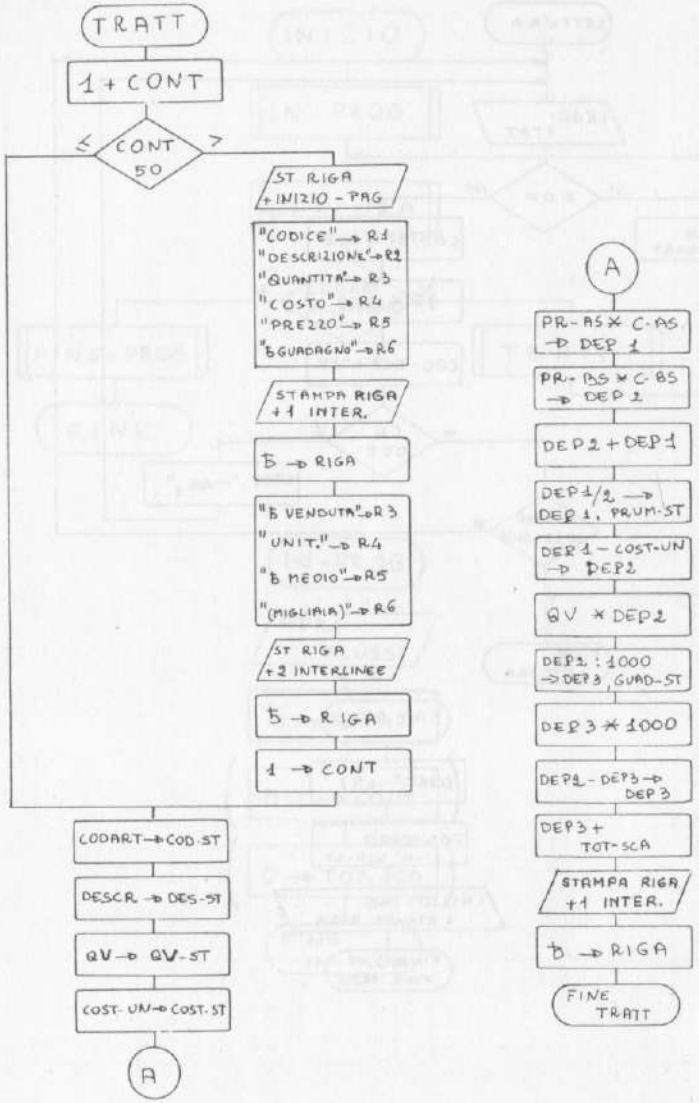


Fig. 6.7a

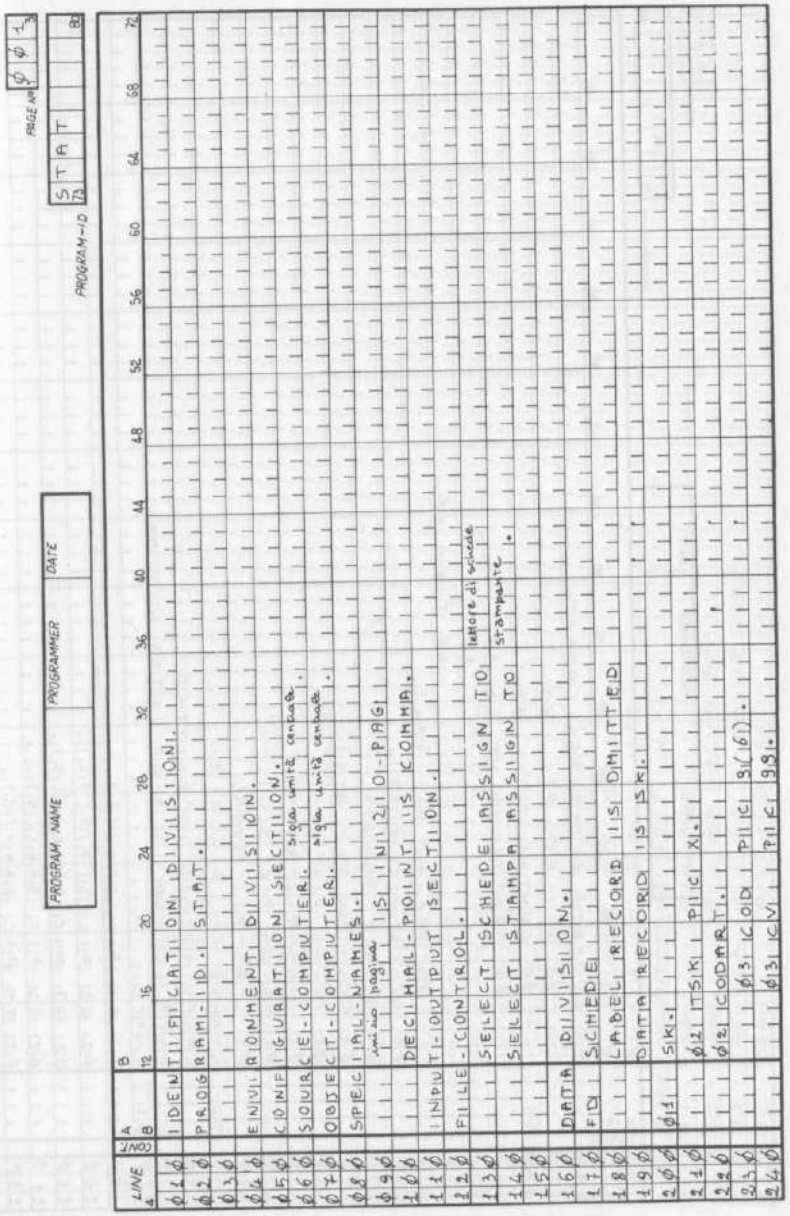


Fig. 6.7b

PROGRAM NAME	PROGRAMMER	DATE

PAGE NR 1 Ø Ø 2 3

PROGRAM-ID 73 S T A T 80

LINE 4	CONT. A B	B 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72
Ø1Ø		Ø2 CODART REDEF INES CODART.
Ø2Ø		Ø3 FILLER PIC XX.
Ø3Ø		Ø3 CENTRO PIC 99.
Ø4Ø		88 TRATTA VALUE IS 1Ø
Ø5Ø		25
Ø6Ø		3Ø TIRU 15Ø.
Ø7Ø		Ø3 FILLER PIC X(41).
Ø8Ø		Ø2 DESCR PIC X(251).
Ø9Ø		Ø2 GV PIC 99V99.
1ØØ		Ø2 COSTI-UN PIC 9(14).
11Ø		Ø2 PRI-PIS PIC 9(14).
12Ø		Ø2 CO-BIS PIC 9V99.
13Ø		Ø2 PRI-BIS PIC 9(14).
14Ø		Ø2 CO-BIS PIC 9V99.
15Ø	FD	STAMPA
16Ø		LABEL RECORD IS OMITTED
17Ø		DATA RECORD IS RIGA
18Ø	Ø1	INTES.
19Ø		Ø2 R1 PIC BBBX(16).
2ØØ		Ø2 R2 PIC B(71)X(11).
21Ø		Ø2 R3 PIC B(121)X(91).
22Ø		Ø2 R4 PIC BBBX(15).
23Ø		Ø2 R5 PIC BBBX(16).
24Ø		Ø2 R6 PIC BBBX(10).

Fig. 6.7c

PROGRAM NAME	PROGRAMMER	DATE

PAGE NR 1 Ø Ø 3 3

PROGRAM-ID 73 S T A T 80

LINE 4	CONT. A B	B 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72
Ø1Ø	Ø1	RIGA.
Ø2Ø		Ø2 FILLER PIC XX.
Ø3Ø		Ø2 CODI-STI PIC X(18).
Ø4Ø		Ø2 FILLER PIC XX.
Ø5Ø		Ø2 DESI-STI PIC X(251).
Ø6Ø		Ø2 FILLER PIC XXX.
Ø7Ø		Ø2 QV-IST PIC 22,99.
Ø8Ø		Ø2 FILLER PIC X(15).
Ø9Ø		Ø2 COSTI-STI PIC 2.2299.
1ØØ		Ø2 FILLER PIC X(14).
11Ø		Ø2 PRUMI-STI PIC 2.2299.
12Ø		Ø2 FILLER PIC X(16).
13Ø		Ø2 GUADI-STI PIC 2.2299+.
14Ø		Ø2 FILLER PIC X.
15Ø	Ø1	VILTIKAI.
16Ø		Ø2 FILLER PIC XX.
17Ø		Ø2 R7 PIC X(16).
18Ø		Ø2 FILLER PIC XXX4
19Ø		Ø2 SCAI-STI PIC 22.229+.
2ØØ	W0RK	INGI-STORAGE SECTION.
21Ø	77	DEP PIC 9(6).
22Ø	77	CONT PIC 999 VALUE 15Ø.
23Ø	77	TOTI-SCA PIC S99(5) VALUE ZERO.
24Ø	77	DEP1 PIC 9(4)V99.

Fig. 6.7d

PROGRAM NAME | PROGRAMMER | DATE

PAGE No. 004

PROGRAM-ID STAT

LINE	CONT.	A	B
010		77	DEP2 PICI 9(141) V19,9.
020		77	DEP3 PICI 9(71).
030			
040		PROC	EDURE DIVISION.
050		FLOW	SECTION.
060			PERFORM INI-PROG.
070		LEGG	1.
080			PERFORM LETTURA
090			IF CODARTI NOTI = HIGH-VALUE
100			PERFORM TRATTI
110			GOTO LEGG1.
120			PERFORM FINE-PROG
130			STOP-RUN.
140			
150		INI-PROG	SECTION.
160			OPEN INPUT SCHEDE
170			OPEN OUTPUT STAMPA
180			MOVE SPACE TO RIGA.
190			
200		LETTURA	SECTION.
210			READ STATI AT END MOVE HIGH-VALUE TO CODARTI
220			GOTO USCITA-LETTURA.
230			DIVIDE 3 INTO CODIGINGDEP
240			MULTIPLY 3 BY DEP

Fig. 6.7e

PROGRAM NAME | PROGRAMMER | DATE

PAGE No. 005

PROGRAM-ID STAT

LINE	CONT.	A	B
010			SUBTRACT DEP FROM CODIGINGDEP
020			IF CVNOTI = DEP STOP "mess." GOTO LETTURA.
030			IF NOTI TRATTA GOTO LETTURA.
040		USCITA-LETTURA.	EXIT.
050			
060		FINE-PROG	SECTION.
070			MOVE "SKARTI" TO R7
080			MOVE TOT-SCARTO SCAR-ST
090			WRITE RIGA AFTER ADVANCING PAGE
100			
110		TRATTI	SECTION.
120			ADD 4 TO CONT
130			IF CONT > 50 WRITE RIGA BEFORE ADVANCING PAGE
140			MOVE "CODICE" TO R1
150			MOVE "DESCRIZIONE" TO R2
160			MOVE "QUANTITA'" TO R3
170			MOVE "COSTO" TO R4
180			MOVE "PREZZO" TO R5
190			MOVE "GUADAGNO" TO R6
200			WRITE RIGA BEFORE 1 LINE
210			MOVE SPACE TO RIGA
220			MOVE "IVENDUTA" TO R3
230			MOVE "UNITA" TO R4
240			MOVE "MEDIO" TO R5

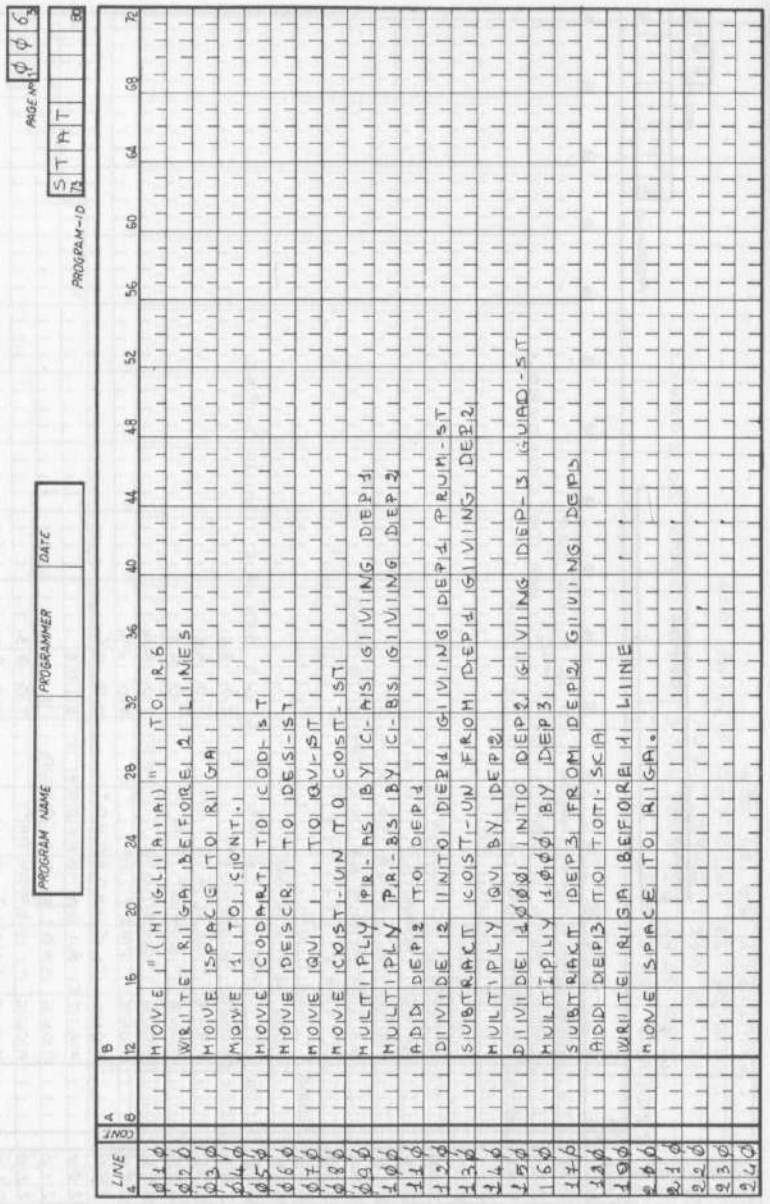


Fig. 6.7f

7. DIVISIONE PROCEDURA

In questo capitolo prenderemo in esame le istruzioni già trattate in 5. e le completeremo illustrando quelle clausole che per semplicità di trattazione avevamo omesso; analizzeremo inoltre alcune istruzioni che, pur non essendo di importanza fondamentale, sono particolarmente utili per la soluzione di diversi problemi.

Precisiamo che, per quanto concerne le istruzioni di input/output, non entreremo ancora nel merito delle varie organizzazioni e delle possibilità d'accesso, ma ci limiteremo a considerare (come in 5.) il caso di organizzazione sequenziale con accesso sequenziale in input oppure in output.

7.1. Istruzioni di entrata e di uscita

7.1.1. *Il verbo OPEN*

Avevamo presentato questo verbo con il seguente formato:

```
OPEN {INPUT  } nome-flusso [nome-flusso]...
     {OUTPUT }
```

Il Cobol offre però maggiori possibilità come si può osservare dall'analisi di questo formato:

```
                                                                  ]]
OPEN {INPUT  nome-flusso [REVERSED      ][nome-flusso [REVERSED      ]]...}
     {OUTPUT nome-flusso [WITH NO REWIND][nome-flusso [WITH NO REWIND]]...}
```

Quando è specificata la clausola REVERSED, che può essere utilizzata per soli flussi in input, al momento dell'esecuzione della OPEN il posizionamento avviene in corrispondenza dell'etichetta di fine flusso e le successive istruzioni di lettura rendono disponibili i record logici in area record in ordine inverso, cioè dall'ultimo al primo.

La clausola REVERSED, che ha significato solo per flussi su nastro o su disco magnetico, può essere usata solo per flussi monovolume.

La clausola WITH NO REWIND consente di eseguire la OPEN di un flusso su nastro magnetico senza provvedere al riavvolgimento della bobina: questa clausola è utile qualora si debbano trattare bobine multiflusso (se questo tipo di trattamento è permesso dal software dell'elaboratore che si utilizza).

7.1.2. *Il verbo CLOSE*

Il verbo CLOSE assume la seguente struttura completa:

```
                  { REEL }[ WITH NO REWIND ]
                  { UNIT }[ FOR REMOVAL    ]
CLOSE nome-flusso[                         ]
                  [ WITH { NO REWIND }     ]
                  [      { LOCK      }     ]

[ nome-flusso [ { REEL }[ WITH NO REWIND ] ] ]
              [ { UNIT }[ FOR REMOVAL    ] ] ...
              [                            ]
              [ WITH { NO REWIND }         ]
              [      { LOCK      }         ]
```

L'istruzione CLOSE, che può essere riferita solo a flussi trattati con una OPEN, può, per flussi multivolume, fare riferimento all'intero flusso oppure ad un volume del flusso (REEL per un volume su nastro magnetico, UNIT per un volume su disco magnetico).

Nel caso in cui la CLOSE sia riferita ad un volume del flusso in output, consente di registrare una etichetta di fine volume, mentre per flussi in input permette di passare al trattamento del volume successivo.

L'opzione NO REWIND indica che, per flussi su nastro magnetico, non deve essere effettuato il riavvolgimento della bobina: questa clausola può essere utilizzata sia in concomitanza di una CLOSE del flusso sia di una CLOSE di volume (REEL).

La clausola REMOVAL, che può essere riferita a volumi di un flusso su nastro o disco magnetico, consente di indicare che l'unità periferica su cui era montato il volume, diventa nuovamente disponibile per accogliere un nuovo volume e quindi è possibile effettuare nuovi accessi a quella periferica.

La clausola LOCK, ammessa solo nella CLOSE di un flusso, vincola il flusso a non essere più trattato da una OPEN e quindi inibisce qualsiasi ulteriore elaborazione di quel flusso nell'ambito del programma in esecuzione.

In fig. 7.1 è riportata una tabella in cui sono evidenziati i possibili formati della CLOSE, le categorie di flussi e le possibili attribuzioni dei formati (naturalmente bisogna tenere presente le regole ora esposte in relazione alle singole opzioni).

Fig. 7.1

Tipi di close	Categorie di flussi		
	Non organizzati a volume	Sequenziali monovolume	Sequenziali multivolume
CLOSE	si	si	si
CLOSE WITH LOCK	si	si	si
CLOSE NO REWIND	no	si	si
CLOSE REEL/UNIT	no	no	si
CLOSE REEL/UNIT FOR REMOVAL	no	no	si
CLOSE REEL/UNIT WITH NO REWIND	no	no	si

7.1.3. *Il verbo READ*

Esaminiamo questo formato dell'istruzione di lettura:

READ nome-flusso [INTO nome-dato] [AT END istruz.]

La clausola INTO consente di rendere disponibile al programmatore il record logico trattato dalla READ non solo in area record ma anche nella zona di memoria individuata dal nome mnemonico nome-dato. Nome-dato non può essere uno dei nomi attribuiti dal programmatore all'area record del flusso e neppure il nome che identifica una porzione dell'area record.

La clausola INTO non può essere utilizzata se il flusso è costituito da record di lunghezza variabile.

Esempio:

```
FD    ARCH LABEL RECORD IS STANDARD
      BLOCK CONTAINS 15 RECORDS
      DATA RECORD IS REC.
01    REC PIC X(95).
WORKING-STORAGE SECTION.
.........
01    DEP-REC.
      02 .....
      02 .....
      02 .....
.........
PROCEDURE DIVISION.
.........
      READ ARCH INTO DEP-REC AT END ....
.........
```

Ogni record del flusso ARCH è così disponibile, dopo ogni lettura, sia nell'area record REC sia nell'area di lavoro DEP-REC.

Se un flusso in input è stato dichiarato OPTIONAL nella relativa SELECT ed è assente durante l'elaborazione, al momento dell'esecuzione del primo ordine di lettura si ha l'uscita per fine-flusso.

E' a carico del programmatore predisporre il programma in modo che non si ritorni, successivamente, su un ordine di lettura per il flusso in cui si è riscontrata la fine, infatti, in tal caso si avrebbero degli errori bloccanti.

La clausola AT END può essere omessa qualora si provveda da programma alla gestione del fine flusso: più avanti analizzeremo le procedure da utilizzare in questo caso. Ricordiamo che normalmente la frase AT END viene utilizzata, la si omette solo nel caso in cui

l'etichetta di fine flusso non sia presente oppure non sia standard per il sistema operativo che si sta utilizzando e quindi non può essere riconosciuta automaticamente.

7.1.4. *Il verbo WRITE*

Il formato generale di questo verbo è:

```
WRITE nome-record [FROM nome-dato]
[ {BEFORE} ADVANCING {nome-dato} [LINE]  ]
[ {AFTER}            {intero     } [LINES] ]
[                    {identificatore}      ]
[                    {PAGE}                ]
```

Esaminiamo gli elementi nuovi che questo formato presenta rispetto a quello che già conosciamo.

Può essere comodo, in relazione a determinati tipi di elaborazione, preparare in parallelo più record da registrare; in questi casi si "duplica" l'area record definendo opportunamente delle aree di lavoro.

La clausola FROM consente di indicare che un record da mandare in output deve essere prelevato dalla zona indicata: si evita quindi di provvedere al trasferimento delle informazioni, che costituiscono il record logico, dal deposito in cui sono state memorizzate in area record. In ogni caso prima di essere mandato in output il contenuto di nome-dato viene trasferito in area record.

Il nome-dato non può essere un nome che identifica l'area record del flusso oppure una porzione di essa.

La clausola FROM può essere riferita solo a flussi con record di lunghezza fissa.

Per quanto riguarda il controllo dei movimenti carta nei flussi di stampa, oltre alle clausole già note, il Cobol prevede la possibilità di fare eseguire un salto pagina prima o dopo la stampa della riga:

```
{AFTER  } ADVANCING PAGE
{BEFORE }
```

PAGE è una parola riservata associata all'inizio della pagina del foglio della stampante.

7.2. L'istruzione di trasferimento

Quando in 5. abbiamo illustrato la funzione del verbo MOVE, abbiamo consigliato di effettuare i trasferimenti tra campi di uguale natura. Molto spesso infatti, se non si è sicuri della natura dei dati soggetti a trasferimento, i contenuti dei campi riceventi sono diversi rispetto al previsto e, in sede di elaborazione, si hanno risultati imprevedibili.

Ricordando che i campi possono essere di categoria alfabetica, alfanumerica, numerica, numerica editata ed alfanumerica editata, e che i campi di gruppo sono considerati di tipo alfanumerico occorre attenersi alle seguenti regole:

a. non eseguire MOVE da un campo numerico editato o alfanumerico editato in un campo numerico o numerico editato,
b. non eseguire MOVE da un campo numerico o numerico editato in un campo alfabetico,
c. non eseguire MOVE da un campo numerico non intero in un campo alfanumerico o alfanumerico editato.

Inoltre, considerando le sole MOVE di tipo numerico o numerico editato bisogna tenere presente che:

a. l'allineamento è regolato dalla posizione della virgola (o del punto decimale),
b. i troncamenti del dato proveniente dal campo emittente possono aver luogo sia sulla parte intera del numero (le cifre più significative!) sia sulla parte decimale (le cifre meno significative!),
c. l'inserimento di zeri non significativi può avvenire sia alla sinistra del numero sia alla destra nella parte decimale,
d. il segno del numero è trattato automaticamente.

Nella tabella di fig. 7.2 sono rappresentate le MOVE permesse in funzione delle natura dei campi emittente e ricevente; ricordiamo che quando parliamo di "MOVE permessa" intendiamo "MOVE logica", infatti a priori ogni tipo di MOVE è permessa, eventualmente i risultati sono imprevedibili.

Fig. 7.2

DA \ A	ALFABETICO	ALFANUMERICO	ALFANUMERICO EDITATO	NUMERICO INTERO	NUMERICO DECIMALE	NUMERICO EDITATO
ALFABETICO	si	si	si	no	no	no
ALFANUMERICO	si	si	si	si	si	si
ALFANUMERICO EDITATO	no	si	si	no	no	no
NUMERICO INTERO	no	si	si	si	si	si
NUMERICO DECIMALE	no	no	no	si	si	si
NUMERICO EDITATO	no	si	si	no	no	no

7.3. Istruzioni aritmetiche

I formati dei verbi aritmetici che già conosciamo (ADD, SUBTRACT, MULTIPLY e DIVIDE) ed il formato del verbo COMPUTE (che ora presenteremo e che permette di esprimere un'espressione aritmetica mediante le notazioni classiche utilizzate dai linguaggi di programmazione per rappresentare formule) hanno, relativamente ai nomi-dato che intervengono, delle caratteristiche in comune:

a. i campi che contengono operandi sono dei campi elementari, non necessariamente dello stesso formato, nel senso che si possono trattare in una stessa operazione numeri segnati o no, con o senza virgola virtuale.
b. i campi atti a contenere i risultati, purché non siano operandi, possono essere di tipo aritmetico oppure con segno, con o senza virgola virtuale oppure possono essere campi numerici con caratteri di edizione, oppure campi alfanumerici.

7.3.1. *Arrotondamento*

Quando vengono eseguite delle operazioni e il risultato deve essere trasferito nel campo ricevente, se il numero delle cifre decimali del risultato è superiore a quanto previsto dalla PICTURE del campo destinato ad accoglierlo si ha un troncamento dei decimali in eccesso con una conseguente perdita della precisione.

Al fine di evitare di ottenere solo arrotondamenti per difetto può essere utilizzata la clausola

ROUNDED

Questa clausola consente di effettuare un arrotondamento

a. per difetto se la cifra decimale più significativa della parte trascurata è minore di cinque,
b. per eccesso se la cifra decimale più significativa della parte trascurata è maggiore o eguale a cinque.

Esempio:

Risultato nell'unità aritmetica - logica	Contenuto del campo ricevente 999V99 con ROUNDED	senza ROUNDED
234V567	234V57	234V56
234V5649	234V56	234V56
234V564	234V56	234V56
234V5641	234V56	234V56
234V599	234V60	234V59

7.3.2. *Overflow del risultato*

Durante l'esecuzione di una operazione si può incorrere in due tipi di errori:

a. divisione per zero,
b. eccedenza della parte intera del risultato rispetto alle dimensioni del campo ricevente.

Quando si ordina l'esecuzione di una divisione per zero, possono darsi tre casi in relazione al tipo di elaboratore che si usa:

a. si ha uno stop del programma,
b. al campo ricevente viene assegnato un risultato equivalente al valore numerico massimo previsto dall'elaboratore per il campo ricevente,
c. il risultato è imprevedibile.

Quando invece il risultato di una operazione eccede la lunghezza del campo ricevente, per quanto riguarda la sua parte intera, si ha un troncamento della parte più significativa del numero con una conseguente sequenza di errori nelle elaborazioni seguenti.

Per evitare questi inconvenienti il Cobol mette a disposizione la clausola

ON SIZE ERROR..... . .....

Se questa clausola è presente e si è in caso di divisione per zero

oppure di overflow del risultato, non si hanno trasferimenti nel campo ricevente ed il controllo passa all'istruzione codificata dopo la parola ERROR, se invece non si riscontrano irregolarità, l'operazione viene eseguita correttamente ed il controllo passa alla prima istruzione scritta dopo il punto.

Esempio:

Il risultato di una operazione in unità aritmetico-logica sia:

7540ˇ 36

il campo ricevente (999V99) contenga inizialmente:

123ˇ 34

Se non è prevista la clausola ON SIZE ERROR alla fine dell'operazione nel campo ricevente si avrà:

540ˇ 36

con la perdita della cifra più significativa, se invece la clausola è presente nella forma:

ON SIZE ERROR GO TO ERRORE.....

nel campo ricevente si avrà ancora:

123ˇ 45

ed il controllo passa alla procedura di nome ERRORE.

7.3.3. *Il resto della divisione*

Utilizzando i formati delle operazioni elementari presentati finora, se si presenta la necessità di memorizzare in un'area R il resto della divisione fra A e B, bisogna procedere come in fig. 7.3.

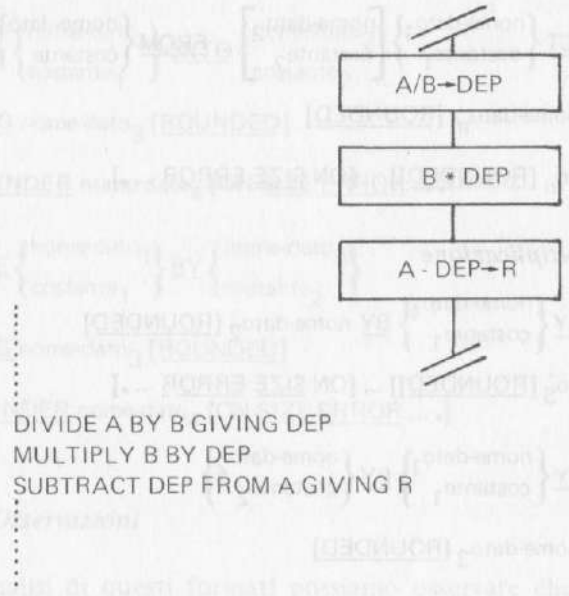
Il formato più generale della divisione ammette anche la possibilità di conservare il resto in un'area opportunamente predisposta utilizzando la clausola

REMAINDER nome-dato

Nel caso in cui in una divisione compaiano sia la clausola relativa al resto sia la ON SIZE ERROR e si ha errore sul quoziente non vengono alterati i contenuti di partenza dei campi quoziente e resto,

mentre se l'errore è sul resto il valore del quoziente viene fornito ugualmente; in ogni caso il controllo passa all'istruzione specificata dopo la parola ERROR.

Fig. 7.3



7.3.4. *I formati completati con le clausole*

7.3.4.1. *Addizione*

a. ADD $\left\{\begin{matrix}\text{nome-dato}_1\\ \text{costante}_1\end{matrix}\right\}\left[\begin{matrix}\text{nome-dato}_2\\ \text{costante}_2\end{matrix}\right]$ ... TO nome-dato$_m$ [ROUNDED]

[nome-dato$_n$ [ROUNDED]] ... [ON SIZE ERROR ... .]

b. ADD $\left\{\begin{matrix}\text{nome-dato}_1\\ \text{costante}_1\end{matrix}\right\}\left\{\begin{matrix}\text{nome-dato}_2\\ \text{costante}_2\end{matrix}\right\}\left[\begin{matrix}\text{nome-dato}_3\\ \text{costante}_3\end{matrix}\right]$ ...

GIVING nome-dato$_m$ [ROUNDED]

[nome-dato$_n$ [ROUNDED]] ... [ON SIZE ERROR ... .]

7.3.4.2. *Sottrazione*

a. SUBTRACT {nome-dato$_1$ / costante$_1$} [nome-dato$_2$ / costante$_2$] ... FROM nome-dato$_m$ [ROUNDED]

[nome-dato$_n$ [ROUNDED]] ... [ON SIZE ERROR ... .]

b. SUBTRACT {nome-dato$_1$ / costante$_1$} [nome-dato$_2$ / costante$_2$] ... FROM {nome-dato / costante}

GIVING nome-dato$_m$ [ROUNDED]

[nome-dato$_n$ [ROUNDED]] ... [ON SIZE ERROR ... .]

7.3.4.3. *Moltiplicazione*

a. MULTIPLY {nome-dato$_1$ / costante$_1$} BY nome-dato$_2$ [ROUNDED]

[nome-dato$_3$ [ROUNDED]] ... [ON SIZE ERROR ... .]

b. MULTIPLY {nome-dato$_1$ / costante$_1$} BY {nome-dato$_2$ / costante$_2$}

GIVING nome-dato$_3$ [ROUNDED]

[nome-dato$_4$ [ROUNDED]] ... [ON SIZE ERROR ... .]

7.3.4.4. *Divisione*

a. DIVIDE {nome-dato$_1$ / costante$_1$} INTO nome-dato$_2$ [ROUNDED]

[nome-dato$_3$ [ROUNDED]] ... [ON SIZE ERROR ... .]

b. DIVIDE {nome-dato$_1$ / costante$_1$} INTO {nome-dato$_2$ / costante$_2$}

GIVING nome-dato$_3$ [ROUNDED]

[nome-dato$_4$ [ROUNDED]] ... [ON SIZE ERROR ... .]

c. DIVIDE {nome-dato$_1$ / costante$_1$} BY {nome-dato$_2$ / costante$_2$}

GIVING nome-dato$_3$ [ROUNDED]

[nome-dato$_4$ [ROUNDED]] ... [ON SIZE ERROR ... .]

d. DIVIDE {nome-dato$_1$ / costante$_1$} INTO {nome-dato$_2$ / costante$_2$}

GIVING nome-dato$_3$ [ROUNDED]

REMAINDER nome-dato$_4$ [ON SIZE ERROR ... .]

e. DIVIDE {nome-dato$_1$ / costante$_1$} BY {nome-dato$_2$ / costante$_2$}

GIVING nome-dato$_3$ [ROUNDED]

REMAINDER nome-dato$_4$ [ON SIZE ERROR ... .]

7.3.4.5. *Osservazioni*

Dall'analisi di questi formati possiamo osservare che, rispetto a quelli che già conoscevamo, vi sono alcune differenze non solo relative alle clausole che abbiamo introdotto, infatti è anche ammesso trasferire il risultato di una addizione o di una sottrazione anche in più campi riceventi: osserviamo che nel caso in cui non sia presente il GIVING i campi riceventi sono anche operandi (ma di operazioni diverse).

Esempi:

ADD A B C TO X Y equivale a:

A+B+C+X→X e
A+B+C+Y→Y

SUBTRACT A B C FROM X Y equivale a

X - (A+B+C) → X e
Y - (A+B+C) → Y

Osserviamo inoltre che per la divisione abbiamo scritto cinque formati diversi, infatti se è presente la clausola REMAINDER il verbo DIVIDE deve esprimere una sola divisione con un solo campo ricevente del risultato.

7.3.5. *Il verbo COMPUTE*

Il verbo COMPUTE consente di esprimere una o più operazioni aritmetiche utilizzando simboli analoghi a quelli usati, per la scrittura delle operazioni, in matematica.

I simboli che possono essere utilizzati per comporre le espressioni aritmetiche sono:

+ per l'addizione
– per la sottrazione
* per la moltiplicazione
/ per la divisione
** per l'elevazione a potenza
(
)

le parentesi servono per modificare l'ordine di esecuzione delle operazioni, infatti se non sono utilizzate le parentesi l'ordine di esecuzione è il seguente

**
* e / nell'ordine in cui sono scritte
+ e – nell'ordine in cui sono scritte

I simboli "+" e "–" possono essere utilizzati anche associati ad un solo operando, infatti è possibile scriverli preposti al nome di un dato oppure ad un'espressione scritta fra parentesi.

Esempio:

+ OTA
- (VAL +TOT)
- (A * VALORE)

Il formato generale del verbo COMPUTE è:

COMPUTE nome-dato$_1$ [ROUNDED][nome-dato$_2$ [ROUNDED]] ...
= espressione-aritmetica [ON SIZE ERROR ... ■]

come si vede utilizzando questo verbo sono ammessi più campi riceventi del risultato (nome-dato$_1$, nome-dato$_2$,...) e le clausole ROUNDED ed ON SIZE ERROR.

Osserviamo che disporre del verbo COMPUTE è utile non solo perchè si riescono ad esprimere in modo sintetico formule complesse, ma anche perchè si possono codificare espressioni che contengono il simbolo di radice (infatti $\sqrt[n]{f} = f^{1/n}$).

Esempi:

a. Per calcolare:
$x = \sqrt[3]{(a+b)/(a-b)}$
possiamo scrivere:

COMPUTE X = ((A + B) / (A - B)) ** (1 / 3)

osserviamo che senza la COMPUTE non saremmo riusciti ad esprimere la radice cubica.

b. Per calcolare:
$e = [a + (b - c)^2 / b] \cdot d$
possiamo scrivere:

COMPUTE E = (A + (B - C) ** 2 / B) * D

che è equivalente alla sequenza di operazioni:

SUBTRACT C FROM B GIVING DEP
MULTIPLY DEP BY DEP
DIVIDE B INTO DEP
ADD A TO DEP
MULTIPLY DEP BY D GIVING E

Osserviamo che per codificare l'espressione data utilizzando i formati delle quattro operazioni elementari abbiamo dovuto introdurre un'area "DEP" da utilizzare come area di lavoro.

Osserviamo che, utilizzando la COMPUTE, è particolarmente vantaggioso disporre di nomi mnemonici brevi per poter meglio leggere la formula che deve essere calcolata, del resto dei nomi brevi non rispondono di solito all'esigenza di essere anche nomi mnemonici: per risolvere questo problema si può procedere assegnando, tramite il livello 66, due nomi alle aree che devono essere trattate dalle istruzioni usuali e da una COMPUTE ed inserire prima della formula delle frasi di commento che richiamino il significato dei "nomi-brevi".

7.4. Particolarità sui nomi

Finora, anche se non è stato detto in modo esplicito, ci siamo attenuti alla regola che i nomi che il programmatore assegna ad aree di memoria (aree record o aree di lavoro) devono essere fra loro diversi. Questa limitazione è superflua in quanto il linguaggio Cobol prevede semplicemente che zone di memoria distinte possano essere individuate in modo rigoroso attraverso i loro nomi o, eventualmente, specificando anche il nome del gruppo di cui fanno parte.

Per individuare due aree di memoria che hanno nomi uguali basterà quindi fare riferimento ad un nome gerarchicamente più importante che chiamiamo "qualificatore", che consenta una identificazione univoca dell'area che interessa.

Se un nome ammette più qualificatori questi possono essere citati indifferentemente.

La qualificazione di un nome ha la seguente struttura:

$$\text{nome}\left\{\begin{matrix}\underline{\text{OF}}\\ \underline{\text{IN}}\end{matrix}\right\}\text{qualificatore}_1\left[\left\{\begin{matrix}\underline{\text{OF}}\\ \underline{\text{IN}}\end{matrix}\right\}\text{qualificatore}_2\right]\ldots$$

Esempio di qualificazioni:

Data la seguente struttura di aree record:


```
FILE SECTION.
FD SCHEDE LABEL RECORD IS OMITTED
   DATA RECORD IS SK.
01 SK.
   02 CSK PIC ....
   02 DES PIC ....
   02 QTA PIC ....
   02 PRZ PIC .....
   02 DATASK.
      03 AA PIC ....
      03 MM PIC ....
      03 GG PIC ....
FD STAMPA LABEL RECORD IS OMITTED
   DATA RECORD IS RIGA.
01 RIGA.
   02 FILLER PIC ....
   02 DES    PIC ....
   02 FILLER PIC ....
   02 QTA    PIC ....
   02 FILLER PIC ....
   02 PRZ    PIC ....
   02 FILLER PIC ....
   02 DATARIGA.
      03 AA PIC ....
      03 MM PIC ....
      03 GG PIC ....
```

possiamo, ad esempio, fare le seguenti operazioni, utilizzando i qualificatori:

```
1) MOVE DES IN SK TO DES IN RIGA
2) MOVE AA OF DATASK TO AA OF DATARIGA
3) MOVE AA OF SK TO AA OF RIGA
4) MOVE AA OF DATASK TO AA OF RIGA
```

osserviamo che i casi 2), 3), 4) rispecchiano qualificazioni diverse degli stessi campi, tutte ugualmente corrette.

E' evidente che non esistono strumenti per poter qualificare aree di lavoro a livello 01 oppure a livello 77 che quindi non possono avere nomi uguali ad altri utilizzati nel programma.

Ricordiamo inoltre che non è possibile assegnare nomi uguali a campi di una stessa gerarchia che siano allo stesso livello.

Esempi:

La seguente definizione è corretta:

```
01 A.
   02 B.
      03 C PIC ....
      03 D PIC ....
   02 E.
      03 C PIC ....
      03 X PIC ....
```

mentre questa definizione non è permessa:

```
01 A.
   02 B PIC ....
   02 C PIC ....
   02 B PIC ....
   02 E PIC ....
```

Il Cobol offre anche la possibilità di utilizzare nomi di paragrafo uguali in sezioni diverse, in tal caso il qualificatore del nome paragrafo è il nome della sezione.

Quando si fa riferimento ad un paragrafo dall'interno della sezione a cui appartiene, non è necessario procedere alla qualificazione.

Esempio:

```
PROCEDURE DIVISION.
PRIMA SECTION.
A. ..........
   ..........
   .......... PERFORM B IN SECONDA
B. ..........
   ..........
   .......... GO TO A.
SECONDA SECTION.
X. ..........
   ..........
A. ..........
   .......... PERFORM B
B. ..........
   ..........
   .......... GO TO A IN PRIMA.
```


7.4.1. *Clausola CORRESPONDING per istruzioni aritmetiche*

La clausola CORRESPONDING consente di eseguire le operazioni di somma e sottrazione tra campi di ugual nome appartenenti a due diversi gruppi gerarchici.

Il formato dei verbi ADD e SUBTRACT quando è utilizzata questa clausola è:

a. ADD $\left\{\begin{matrix}\text{CORRESPONDING}\\ \text{CORR}\end{matrix}\right\}$ nome-dato$_1$ TO nome-dato$_2$ [ROUNDED]

[ON SIZE ERROR ... .]

b. SUBTRACT $\left\{\begin{matrix}\text{CORRESPONDING}\\ \text{CORR}\end{matrix}\right\}$ nome-dato$_1$ FROM nome-dato$_2$ [ROUNDED]

[ON SIZE ERROR ... .]

vengono considerati corrispondenti tutti i campi elementari numerici descritti con lo stesso nome sia in nome-dato$_1$ che in nome-dato$_2$, i campi di nome-dato$_2$ interessati dall'operazione sono anche campi riceventi del risultato.

Se è specificata, l'opzione ROUNDED interessa tutti i campi riceventi.

Se è usata la clausola ON SIZE ERROR, e si ha overflow su almeno uno dei campi riceventi, il controllo passa all'istruzione scritta dopo la parola ERROR.

I campi FILLER non sono interessati dalle operazioni come pure i campi elementari che contengono le clausole REDEFINES, RENAMES, OCCURS o USAGE IS INDEX o che sono subordinati a campi che contengono le clausole REDEFINES, OCCURS o USAGE IS INDEX (di queste ultime due clausole si parlerà in 8.).

Esempio:

Siano definiti i seguenti campi:

```
01 X1.
   02 A PIC 9(5).
   02 B PIC 9(5).
   02 C PIC X(12).
   02 D PIC 9(8).
   02 E PIC 99.
   02 F PIC 9(8).
```

```
01 Y1.
   02 E PIC 9(5).
   02 K PIC 9(5).
   02 A PIC XX.
   02 B PIC 9(7).
   02 C PIC X(12).
   02 D PIC X(5).
```

l'istruzione

```
ADD CORR X1 TO Y1
```

provoca la seguente serie di operazioni

```
B OF X1 + B OF Y1 → B OF Y1
D OF X1 + D OF Y1 → D OF Y1
E OF X1 + E OF Y1 → E OF Y1
```

Osserviamo che i campi F e K non sono interessati dall'operazione, in quanto compaiono solo in uno dei due gruppi gerarchici, mentre A e C che compaiono in entrambi i gruppi, non sono considerati in quanto i due campi C non sono numerici, come pure uno dei due campi A.

7.4.2. *Clausola CORRESPONDING per il verbo MOVE*

La clausola CORRESPONDING applicata al verbo MOVE offre delle possibilità analoghe a quelle esaminate in 7.4.1.

Il formato della MOVE CORRESPONDING è:

$$\underline{\text{MOVE}} \left\{ \begin{matrix} \underline{\text{CORRESPONDING}} \\ \underline{\text{CORR}} \end{matrix} \right\} \text{nome-dato}_1 \ \underline{\text{TO}} \ \text{nome-dato}_2$$

Questo formato è riassuntivo di un certo numero di MOVE e precisamente tante quanti sono i sottocampi di uguale nome in nome-dato$_1$ e nome-dato$_2$: ogni operazione di trasferimento è eseguita secondo le regole che conosciamo, che sono funzione della categoria del campo ricevente.

Esempio:

Siano assegnati i seguenti campi:

```
01 RECA.
   02 CODMAG   PIC 999.
   02 CODART   PIC 9(5).
   02 DESART   PIC X(20).
   02 QTA      PIC S9(5).
   02 SCORTAMIN PIC 9(5).
   02 PRUN     PIC 9(6).
   02 DATRECA  PIC 9(6).
01 RECB.
   02 CODART   PIC 9(5).
   02 CODMAG   PIC 999.
   02 QTA      PIC S9(5).
   02 CODPRIOR PIC 99.
   02 NOMVEND  PIC X(20).
   02 PRUN     PIC 9(6).
   02 SCONTO   PIC 99V99.
```

l'istruzione:

```
MOVE CORR RECA TO RECB
```

provoca la seguente serie di trasferimenti:

```
CODMAG OF RECA → CODMAG OF RECB
CODART OF RECA → CODART OF RECB
QTA    OF RECA → QTA OF RECB
PRUN   OF RECA → PRUN OF RECB
```

7.5. Frasi condizionali

7.5.1. *Condizioni semplici*

I tipi di condizione che si possono testare in Cobol sono:

a. di relazione,
b. di classe,
c. di segno,
d. relative al livello 88.

Abbiamo già accennato a tutti questi tipi di test, e precisamente in 5 abbiamo trattato i casi a, b, c ed in 6 il caso d.

Per quanto riguarda il caso di test legati al nome-condizione assegnato a livello 88 non abbiamo niente da aggiungere, mentre per gli altri casi si tratta di fare un'analisi completa alla luce del fatto che ora sono note tutte le PICTURE disponibili in Cobol relative alle varie categorie di dati.

7.5.1.1. *Test di relazione*

Come sappiamo questi test consentono di confrontare due operandi secondo questo formato:

$$\left\{\begin{matrix}\text{nome-dato}_1\\ \text{costante}_1\end{matrix}\right\} \text{ IS } [\underline{\text{NOT}}] \left\{\begin{matrix}\underline{\text{GREATER}}\ \text{THAN}\\ \underline{>}\\ \underline{\text{LESS}}\ \text{THAN}\\ \underline{<}\\ \underline{\text{EQUAL}}\ \text{TO}\\ \underline{=}\end{matrix}\right\} \left\{\begin{matrix}\text{nome-dato}_2\\ \text{costante}_2\end{matrix}\right\}$$

Se i due termini del confronto sono di categoria numerica la lunghezza, in numero di caratteri, degli operandi non è significativa in quanto il confronto è basato sul valore algebrico degli operandi.

Lo zero è considerato un'unico valore indipendentemente dal segno: una affermazione di questo tipo non deve stupire in quanto se ci si riferisce a campi segnati ogni elemento contenuto nel campo segnato deve avere un segno (analizziamo ad esempio i risultati delle operazioni di fig. 7.4).

Esaminiamo ora il caso in cui uno dei due termini del confronto oppure entrambi non siano di categoria numerica.

a. un operando numerico non intero non può essere confrontato con un operando non numerico.
b. se un operando è non numerico oppure una costante alfanumerica e l'altro operando è numerico, quest'ultimo viene trattato come se il suo valore fosse inserito in un campo alfanumerico della stessa lunghezza, e quindi il confronto è di tipo alfanumerico, ovvero i due operandi sono confrontati carattere per carattere, da sinistra a destra per lunghezza pari a quella del termine di confronto lungo, infatti il più corto viene virtualmente allungato a

Fig. 7.4

VALORI INTERESSATI DALL'OPERAZIONE	VALORE ASSOLUTO DEL RISULTATO	SEGNO
100 − 80	20	+
100 − 95	5	+
100 − 99	1	+
100 − 100	0	+
− 100 + 80	20	−
− 100 + 95	5	−
− 100 + 99	1	−
− 100 + 100	0	−

destra con spazi (notiamo che questa normalizzazione non viene eseguita su qualche elaboratore).

7.5.1.2. *Test di classe*

Abbiamo visto che un test di classe viene eseguito secondo queste modalità:

$$\text{nome-dato IS } [\underline{\text{NOT}}] \left\{\begin{matrix}\underline{\text{NUMERIC}}\\ \underline{\text{ALPHABETIC}}\end{matrix}\right\}$$

Questo tipo di test consente di determinare se il contenuto di nome-dato è numerico oppure alfabetico.

La condizione NUMERIC può riferirsi a campi di tipo numerico intero o decimale o campi numerici con segno, può inoltre riferirsi a campi alfanumerici sia elementari che di gruppo purché non formati da campi numerici segnati; questa condizione non è ammessa per campi di categoria alfabetica.

La condizione ALPHABETIC non può essere riferita ad un nome dato di categoria numerica.

I test di classe vengono utilizzati soprattutto per il controllo dei dati provenienti da sistemi di "data entry" ed in particolar modo il controllo è relativo alla numericità dei dati che esigono un trattamento con istruzioni aritmetiche.

7.5.1.3. *Test di segno*

Utilizzando il test di segno, secondo il seguente formato:

$$\text{nome-dato IS } [\underline{\text{NOT}}] \left\{ \begin{array}{l} \underline{\text{NEGATIVE}} \\ \underline{\text{POSITIVE}} \\ \underline{\text{ZERO}} \end{array} \right\}$$

è possibile determinare se un campo di categoria numerica o numerica segnata contiene un valore $>$, $\geqslant$, $<$, $\leqslant$, $=$ oppure $\neq$ da zero; osserviamo che in ogni caso, utilizzando in modo appropriato i test di relazione si possono ottenere gli stessi risultati.

7.5.1.4. *Test di condizione*

Abbiamo già trattato, a proposito del livello 88, questo tipo di test. Ricordiamo che a livello 88 è possibile assegnare un nome mnemonico che chiamiamo nome-condizione al fatto che una certa area di lavoro contenga una classe di valori: il Cobol consente di eseguire un test per sapere se l'area di lavoro suddetta contiene oppure no uno dei valori espressi da un nome-condizione. Questo è il formato del test di condizione:

[NOT] nome-condizione

7.5.2. *Condizioni composte*

La combinazione di più condizioni semplici, effettuata tramite gli operatori logici OR, AND e NOT, dà luogo a condizioni composte.

In fig. 7.5 è riportata una tabella che riassume il significato degli operatori logici (A e B sono due condizioni).

Il formato generale di una condizione composta è

$$\text{condizione}_1 \left\{ \begin{array}{l} \underline{\text{AND}} \\ \underline{\text{OR}} \end{array} \right\} \text{condizione}_2 \left[\left\{ \begin{array}{l} \underline{\text{AND}} \\ \underline{\text{OR}} \end{array} \right\} \text{condizione}_3 \right] \ldots$$

Osserviamo che nel formato generale non compare la parola NOT, questo perchè NOT è sempre previsto, come elemento opzionale, come negazione di una condizione espressa, non solo per le condizioni semplici presentate in 7.5.1. ma anche come negazione di condizioni composte; l'operatore NOT non può precedere una condizione che inizi con NOT.

Fig. 7.5

Condizione		Condizione e valore		
A	B	A AND B	A OR B	NOT A
vera	vera	vera	vera	falsa
vera	falsa	falsa	vera	falsa
falsa	vera	falsa	vera	vera
falsa	falsa	falsa	falsa	vera

Le condizioni composte vengono esaminate secondo il seguente ordine:

a. calcolo di espressioni aritmetiche
b. operatori di relazione
c. NOT
d. AND
e. OR

per modificare questo ordine (ricordiamo che simboli dello stesso livello di priorità di trattamento vengono esaminati da sinistra a destra) si possono utilizzare le parentesi rotonde.

Esempi:

In questi esempi la condizione composta è stata ambientata in una IF per meglio coglierne il significato.

a. IF A NOT = B AND C OR D GO TO X. GO TO Y.

è equivalente a:

```
IF A NOT = B AND A NOT = C OR A NOT = D
   GO TO X. GO TO Y.
```

e rappresenta la codifica del ramo di flow chart di fig. 7.6.

b. IF A NUMERIC AND A NOT = Y
   OR L NEGATIVE GO TO X. GO TO Y.

Fig. 7.6

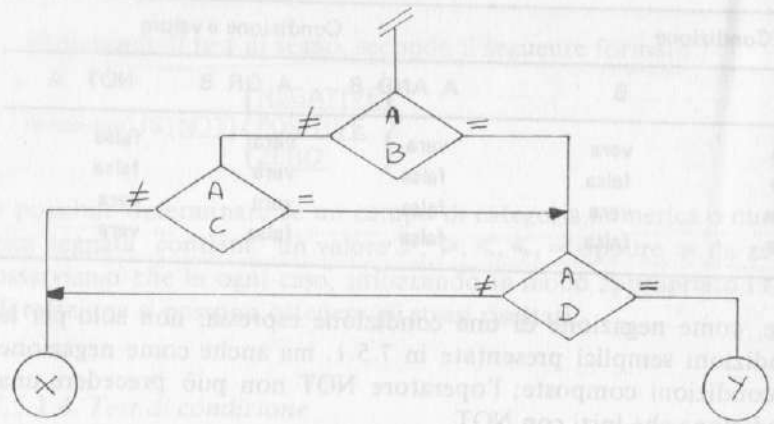
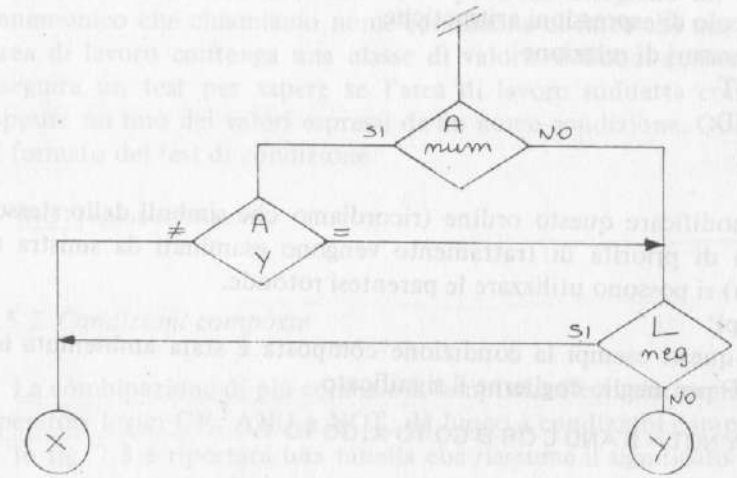


Fig. 7.7

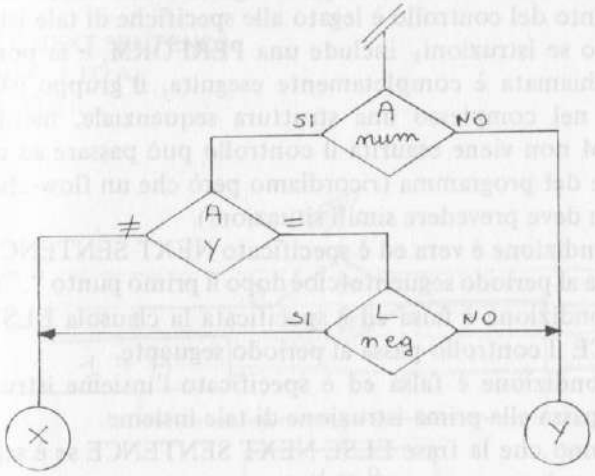


è la traduzione del ramo di flow-chart di fig. 7.7.

c. IF A NUMERIC AND (A NOT = Y OR
L NEGATIVE) GO TO X. GO TO Y.

è la codifica del ramo di flow-chart di fig. 7.8.

Fig. 7.8



Osserviamo che le condizioni composte "consentono" di predisporre flow-chart particolarmente complessi e quindi se ne sconsiglia l'uso: inoltre anche la codifica delle condizioni composte è abbastanza poco chiara dal punto di vista della leggibilità.

7.5.3. *Il verbo IF*

Come sappiamo il verbo IF consente di valutare una condizione e, a seconda che sia vera o falsa il controllo passa a due diversi punti.

Abbiamo per ora esaminato dei formati ridotti della IF , quello completo è:

$$\text{IF condizione} \left\{ \begin{array}{l} \text{istruzioni}_1 \\ \underline{\text{NEXT SENTENCE}} \end{array} \right\} \left\{ \begin{array}{l} \underline{\text{ELSE}}\ \text{istruzioni}_2 \\ \underline{\text{ELSE}}\ \underline{\text{NEXT SENTENCE}} \end{array} \right\}$$

Se la condizione è vera ed è precisato l'insieme di istruzioni chiamato istruzioni$_1$, il controllo passa alla prima istruzione di tale insieme; se istruzioni$_1$ è una struttura sequenziale al termine della esecuzione di tale struttura il controllo passa al periodo seguente, cioè alla prima istruzione scritta dopo il primo punto; se istruzioni$_1$

contiene una istruzione di salto o di richiamo di sottoprogramma il trasferimento del controllo è legato alle specifiche di tale istruzione: ad esempio se istruzioni$_1$ include una PERFORM, e la porzione di codice richiamata è completamente eseguita, il gruppo istruzioni$_1$ mantiene nel complesso una struttura sequenziale, mentre se la PERFORM non viene esaurita il controllo può passare ad un punto qualunque del programma (ricordiamo però che un flow-chart "corretto" non deve prevedere simili situazioni).

Se la condizione è vera ed è specificato NEXT SENTENCE il controllo passa al periodo seguente (cioè dopo il primo punto ".").

Se la condizione è falsa ed è specificata la clausola ELSE NEXT SENTENCE il controllo passa al periodo seguente.

Se la condizione è falsa ed è specificato l'insieme istruzioni$_2$ il controllo passa alla prima istruzione di tale insieme.

Osserviamo che la frase ELSE NEXT SENTENCE se è seguita dal punto "." può essere omessa per intero.

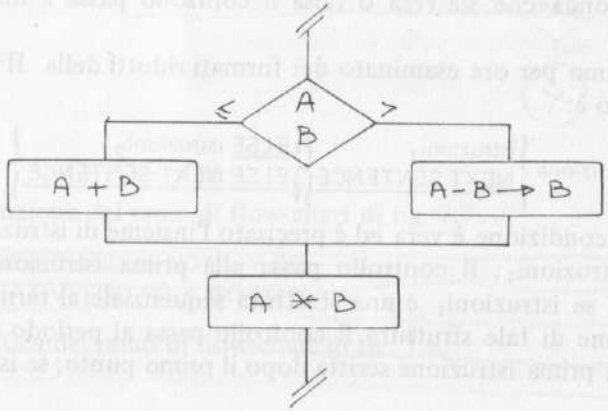
Istruzioni$_1$ e istruzioni$_2$ possono essere anche di tipo condizionale; negli esempi che seguono non considereremo il caso delle IF nidificate che sono trattate in 7.5.4.

Esempi

a. Il ramo di flow-chart di fig. 7.9 può essere codificato così:

```
IF A > B SUBTRACT B FROM A GIVING B
ELSE ADD A TO B.
MULTIPLY A BY B ....
```

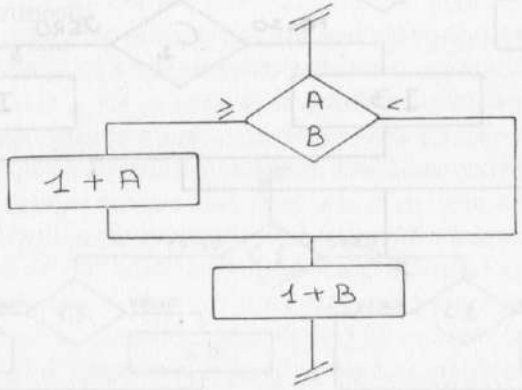
Fig. 7.9



b. Il ramo di flow-chart di fig. 7.10 può essere codificato così:

```
IF A < B NEXT SENTENCE
ELSE ADD 1 TO A.
ADD 1 TO B .....
```

Fig. 7.10



7.5.4. *IF nidificate*

Esaminiamo qualche caso in cui istruzioni$_1$ e istruzioni$_2$ del formato generale delle IF contengano una istruzione condizionale.

Una struttura di questo tipo potrebbe essere quella rappresentata in fig. 7.11 che ha questo formato di codifica:

```
IF condizione1
IF condizione2 istruzioni2
ELSE istruzioni3
ELSE istruzioni1.
.......
```

In fig. 7.12 è rappresentata un'altra struttura di IF nidificate la cui codifica è:

```
IF condizione1
IF condizione2 istruzioni1
ELSE NEXT SENTENCE
ELSE GO TO X.
istruzioni2.
X. istruzioni3 .......
```

Fig. 7.11

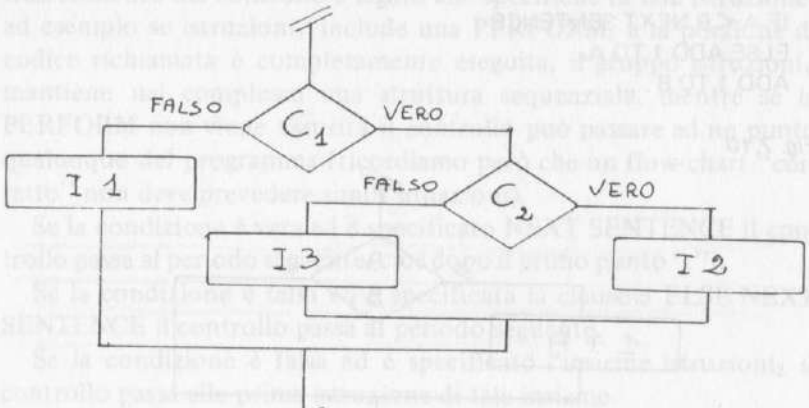
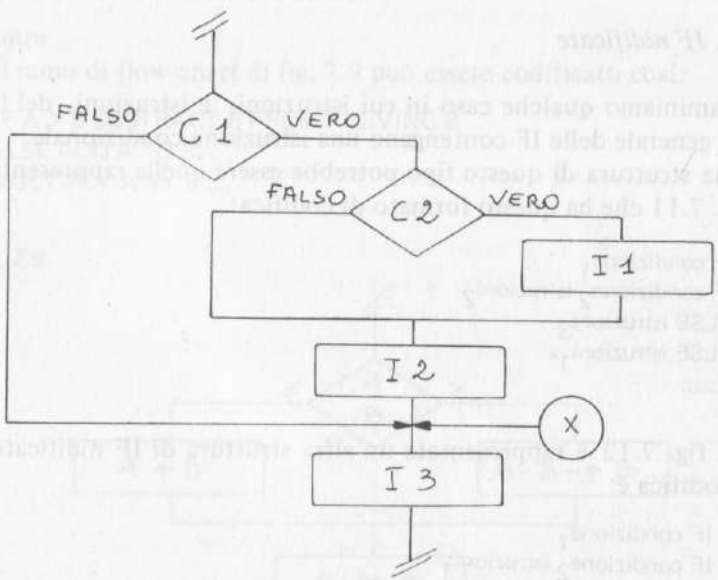


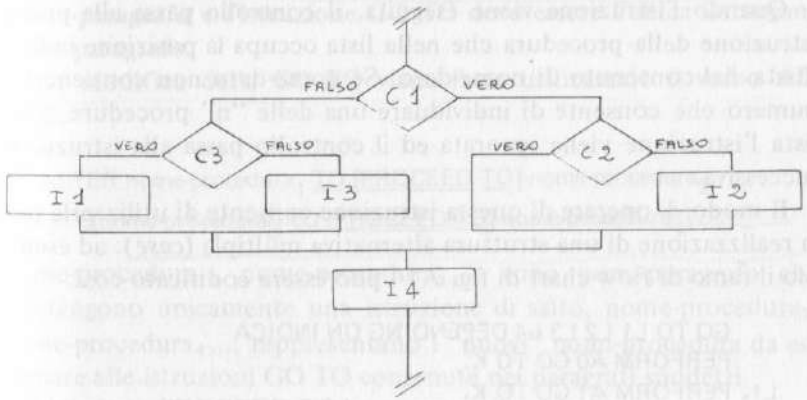
Fig. 7.12



In fig. 7.13 è proposta una terza struttura codificabile con IF nidificate come segue:

IF condizione$_1$
IF condizione$_2$ NEXT SENTENCE
ELSE istruzioni$_2$
ELSE IF condizione$_3$ istruzioni$_1$
ELSE istruzioni$_3$.
istruzioni$_4$ .....

Fig. 7.13



7.6. Il verbo GO TO → SCONSIGLIATO

Abbiamo introdotto ed utilizzato il verbo GO TO per poter realizzare, in taluni casi, delle strutture di controllo, sostanzialmente cicli e strutture alternative nidificate (in 7.5.4. abbiamo visto che esiste la possibilità di codificare delle IF nidificate, ma è abbastanza evidente che le codifiche proposte sono abbastanza poco leggibili): continuiamo a raccomandare di utilizzare il verbo GO TO sempre secondo tali direttive in quanto un uso indiscriminato dell'istruzione di salto incondizionato porta alla costruzione di programmi scarsamente leggibili, quindi difficilmente percorribili e pertanto poco agevolmente testabili e manutenibili.

Occupiamoci ora di quel formato del verbo GO TO che consente di realizzare, con opportuni correttivi, una struttura alternativa multipla (*case*):

GO TO nome-procedura$_1$ [nome-procedura$_2$] ....

.... DEPENDING ON nome-dato

I nomi procedura citati possono essere indifferentemente nomi di paragrafo e di sezione, il numero massimo previsto è funzione del tipo di elaboratore che si utilizza.

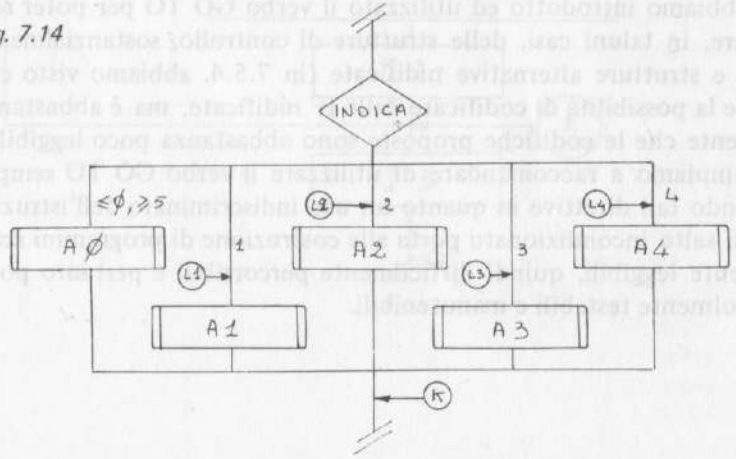
Nome-dato è l'identificativo di un campo numerico intero.

Quando l'istruzione viene eseguita, il controllo passa alla prima istruzione della procedura che nella lista occupa la posizione individuata dal contenuto di nome-dato. Se nome-dato non contiene un numero che consente di individuare una delle "n" procedure della lista l'istruzione viene ignorata ed il controllo passa alla istruzione successiva.

Il modo di operare di questa istruzione consente di utilizzarla per la realizzazione di una struttura alternativa multipla (*case*): ad esempio il ramo di flow-chart di fig. 7.14 può essere codificato così:

```
    GO TO L1 L2 L3 L4 DEPENDING ON INDICA
    PERFORM A0 GO TO K.
L1. PERFORM A1 GO TO K.
L2. PERFORM A2 GO TO K.
L3. PERFORM A3 GO TO K.
L4. PERFORM A4.
K. ............
```

Fig. 7.14



7.7. Il verbo ALTER

Il formato del verbo GO TO, che abbiamo presentato in 5. ed utilizzato negli esempi per esprimere una istruzione di salto incondizionato, è incompleto; il formato completo è infatti:

GO TO [nome-procedura]

Questo formato si giustifica in quanto il linguaggio Cobol dispone di una istruzione che consente di "inserire" un nome-procedura in un'istruzione di salto oppure di sostituire con un nome nuovo un nome preesistente: in questi casi "nome-procedura" deve essere un nome-paragrafo e l'istruzione GO TO deve essere l'unica istruzione di un paragrafo.

La modifica della GO TO viene fatta utilizzando il verbo ALTER:

ALTER nome-procedura$_1$ TO [PROCEED TO] nome-procedura$_2$
[nome-procedura$_3$ TO [PROCEED TO] nome-procedura$_4$] ....

nome-procedura$_1$, nome-procedura$_3$,... sono nomi-paragrafo che contengono unicamente una istruzione di salto, nome-procedura$_2$, nome-procedura$_4$,... rappresentano i "nuovi" nomi-procedura da assegnare alle istruzioni GO TO contenute nei paragrafi suddetti.

L'opzione PROCEED TO ha carattere documentativo.

Esempio:

Proviamo a seguire l'esecuzione del seguente sottoprogramma:

```
ALTER-ESEMPIO SECTION.
A.  istruzioni-1
    ..........
B.  GO TO Y.
Y.  istruzioni-2
    ..........
    ALTER B TO X
    ..........
    GO TO A.
X.  istruzioni-3
    ..........
    IF condizione GO TO FINE.
    ALTER B TO Y
    ..... GO TO A.
FINE. istruzioni-4
    ..... ALTER B TO Y.
```

Quando inizia l'esecuzione viene eseguito il paragrafo A, quindi la GO TO del paragrafo B passa il controllo ad Y; nel paragrafo Y viene predisposta in B una GO TO X, pertanto, esaurite le istruzioni del paragrafo Y, dopo l'esecuzione di A il controllo da B viene trasferito in X. Durante l'esecuzione di X, se non è verificata la condizione che porta all'esecuzione del paragrafo FINE, viene predisposto B in modo che la GO TO in esso contenuta trasferisca il controllo ad Y e quindi vengono nuovamente eseguite le istruzioni del paragrafo A, ecc...

Osserviamo che quando è verificata la condizione espressa in X, dopo le operazioni di fine, nel sottoprogramma in B viene predisposta una GO TO Y, così, ad un nuovo rientro nel sottoprogramma si troveranno le stesse condizioni iniziali.

Osserviamo che l'utilizzo del verbo ALTER è molto semplice, ciò nonostante ne sconsigliamo l'uso in quanto (come si può notare dall'esempio) le porzioni di codice Cobol che ne fanno uso non sono più immediatamente interpretabili.

7.8. Sottoprogrammi interni

7.8.1. *Il verbo PERFORM*

In 5. abbiamo introdotto un formato del verbo PERFORM:

PERFORM nome-procedura$_1$ { THROUGH / THRU } nome-procedura$_2$

Utilizzando questo formato, durante l'esecuzione del programma, vengono richiamate tutte le istruzioni che si trovano fra la prima della prima procedura e l'ultima della seconda, ed il controllo viene restituito all'istruzione scritta dopo la PERFORM al termine dell'esecuzione dell'ultima istruzione della procedura il cui nome è nome-procedura$_2$.

Esempio:

Se si vuole richiamare il sottoprogramma interno SOTT-1 costituito dall'insieme di paragrafi S1, S2, S3, S4 codificati così:


```
SOTT-1 SECTION.
S1. ..................
    ..................
S2. ..................
    ..................
S3. ..................
    ..................
S4. ..................
    ..................
```

è possibile scrivere:

```
PERFORM SOTT-1
```

oppure:

```
PERFORM S1 THRU S4
```

solo se la fine logica del sottoprogramma coincide con la fine del paragrafo S4 (che è anche la fine della sezione SOTT-1).

Come abbiamo visto in 5. se la fine logica di un sottoprogramma non coincide con l'ultima istruzione codificata, oppure se la fine logica si può raggiungere da punti diversi del sottoprogramma, bisogna, per esaurire la procedura richiamata e passare il controllo dell'esecuzione all'istruzione codificata sotto il punto di richiamo, fare in modo di eseguire un'ultima istruzione fisica di tipo non operativo. Per questi casi si può utilizzare il verbo EXIT che è un'istruzione di tipo non operativo e, per il suo significato in inglese, è di tipo documentativo nel testo del programma.

Il verbo EXIT, che deve essere l'unica istruzione di un paragrafo, viene codificato nell'ultimo paragrafo di un gruppo richiamato da una PERFORM ed è il punto d'arrivo, tramite GO TO, di tutte le fini logiche del sottoprogramma: una GO TO al paragrafo contenente il verbo EXIT consente quindi di "forzare" l'esecuzione dell'ultimo paragrafo richiamato dalla PERFORM.

Esempio:

Supponendo che il sottoprogramma SOTT-1 dell'esempio precedente possa terminare logicamente in S2 oppure in S4 lo codificheremo così:

```
SOTT-1 SECTION.
S1. ..........
    ..........
S2. ..........
    .......... GO TO S-USCITA.
S3. ..........
    ..........
S4. ..........
    ..........
S-USCITA. EXIT.
```

In tal caso lo richiameremo o con:

```
PERFORM SOTT-1
```

(in questo caso la sezione SOTT-1 comprende anche il nuovo paragrafo S-USCITA), oppure:

```
PERFORM S1 THRU S-USCITA
```

Riassumiamo ora per punti le particolarità relative a questo formato della PERFORM:

a. Quando viene incontrata una PERFORM il controllo è trasferito alla prima istruzione della sezione o del paragrafo specificato in nome-procedura$_1$.

b. Un implicito rientro alla istruzione immediatamente successiva alla PERFORM viene eseguito:
   - dopo l'esecuzione dell'ultima istruzione di nome-procedura$_1$ se nome-procedura$_1$ è un nome-paragrafo ed è stata omessa la parte "THRU...",
   - dopo l'esecuzione dell'ultima istruzione dell'ultimo paragrafo di nome-procedura$_1$, se nome-procedura$_1$ è un nome-sezione ed è stata omessa la parte "THRU...",
   - dopo l'esecuzione dell'ultima istruzione di nome-procedura$_2$, se nome-procedura$_2$ è un nome-paragrafo,
   - dopo l'esecuzione dell'ultima istruzione dell'ultimo paragrafo di nome-procedura$_2$, se nome-procedura$_2$ è un nome-sezione.

c. Non occorre che nome-procedura$_1$ e nome-procedura$_2$ siano entrambi nomi di sezione o di paragrafo.

d. Raccomandiamo di strutturare nome-procedura$_2$ sempre come paragrafo contenente solo il verbo EXIT, questo può essere utile non solo per "forzare", nei punti in cui si presenta la fine-logica, il passaggio del controllo alla fine fisica, ma è anche utile per evidenziare, a scopo documentativo, la fine del sottoprogramma.

7.8.2. *PERFORM... TIMES*

In Cobol è disponibile anche il seguente formato del verbo PERFORM:

$$\underline{\text{PERFORM}}\ \text{nome-procedura}_1 \left[\left\{ \begin{matrix} \underline{\text{THROUGH}} \\ \underline{\text{THRU}} \end{matrix} \right\} \text{nome-procedura}_2 \right] \left\{ \begin{matrix} \text{nome-dato} \\ \text{intero} \end{matrix} \right\} \underline{\text{TIMES}}$$

Questo formato consente di eseguire l'insieme di istruzioni individuato dai nomi nome-procedura$_1$ e nome-procedura$_2$ tante volte quanto è il valore dell'intero oppure per un numero di volte pari al contenuto nel campo nome-dato.

Le modalità dell'esecuzione sono le stesse presentate per la PERFORM in 7.8.1.

Se il "numero di volte" è un valore minore o uguale a zero la PERFORM viene ignorata.

Durante l'esecuzione di una PERFORM... TIMES il contenuto di nome-dato non può essere modificato.

Esempio:

Supponiamo che un sottoprogramma di stampa di un modulo sia codificato così:

```
MODULO SECTION.
A. ..........
   ..........
   ⋮
EX-MODULO. EXIT.
```

Il modulo debba essere stampato tante volte pari al contenuto del campo NUM; per il richiamo del sottoprogramma scriveremo:

```
PERFORM MODULO THRU EX-MODULO NUM TIMES
```

7.8.3. *PERFORM... UNTIL...*

Quando il numero di volte (TIMES) in cui si deve eseguire un sottoprogramma non è definibile a priori, si può usare l'opzione UNTIL che consente di introdurre una condizione a controllo dell'esecuzione iterata del sottoprogramma:

PERFORM nome-procedura$_1$ [{ THROUGH / THRU } nome-procedura$_2$]

UNTIL condizione.

Se la condizione specificata è già verificata all'inizio, il sottoprogramma non viene eseguito ed il controllo passa subito all'istruzione seguente.

Se invece la condizione non è verificata il sottoprogramma viene eseguito e l'esecuzione è iterata fino a quando la condizione non risulta verificata: il test sulla condizione viene ripetuto prima di ogni esecuzione del sottoprogramma.

Le modalità di ogni singola esecuzione del sottoprogramma sono le stesse presentate in 7.8.1.

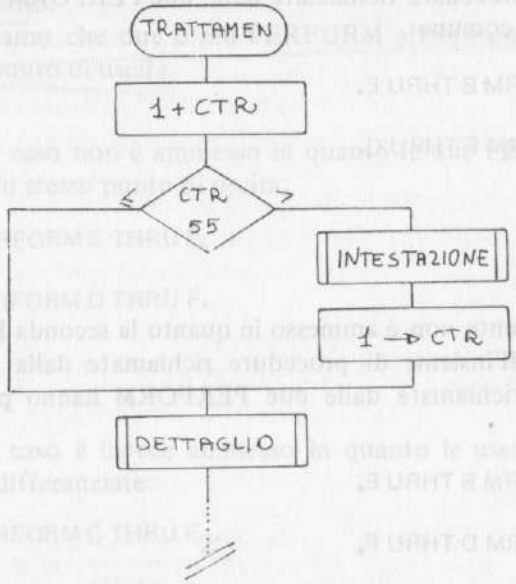
Esempio:

Questa potrebbe essere la codifica del ramo di flow-chart di fig. 7.15.

```
TRATTAMENTO SECTION.
    ADD 1 TO CTR
    PERFORM INTESTAZIONE THRU UNO-IN-CTR
    UNTIL CTR NOT > 55.
    PERFORM DETTAGLIO THRU FINE-DETTAGLIO
    .....
```

Osserviamo che nel sottoprogramma relativo all'intestazione deve essere inclusa l'istruzione che pone il contatore-righe (CTR) a 1.

Fig. 7.15



7.8.4. *PERFORM nidificate*

Se una sequenza di istruzioni richiamate da una PERFORM include un'altra PERFORM le procedure associate a quest'ultima devono essere totalmente interne o totalmente esterne alle procedure richiamate dalla prima PERFORM.

Esempi:

a. Questo caso è ammesso in quanto la sequenza richiamata dalla seconda PERFORM è totalmente interna alla sequenza richiamata dalla prima:

```
A. PERFORM B THRU F.
B.
C. PERFORM D THRU E.
D.
E.
F.
```

b. Anche il caso che presentiamo ora è ammesso in quanto la sequenza di procedure richiamate dalle due PERFORM non hanno paragrafi in comune:

```
A. PERFORM B THRU E.
B.
C. PERFORM F THRU G.
D.
E.
F.
G.
```

c. Il caso seguente non è ammesso in quanto la seconda PERFORM è interna all'insieme di procedure richiamate dalla prima e le procedure richiamate dalle due PERFORM hanno paragrafi in comune:

```
A. PERFORM B THRU E.
B.
C. PERFORM D THRU F.
D.
E.
F.
```

d. Questo caso è invece ammesso in quanto le due PERFORM, che richiamano sequenze di procedure con paragrafi in comune, sono totalmente indipendenti:

```
A. PERFORM D THRU F.
B.
C. PERFORM E THRU G.
D.
E.
F.
G.
```

Evidentemente, con le dovute cautele, si può estendere il discorso delle PERFORM nidificate a più livelli, bisogna in sostanza tenere conto di questa regola: una PERFORM attiva, che rimanda ad una procedura, tramite una PERFORM in essa contenuta, deve essere

esaurita solo dopo che viene disattivata la PERFORM richiamata a livello interno.

Ricordiamo che due o più PERFORM attive non possono avere lo stesso punto di uscita.

Esempi:

a. Questo caso non è ammesso in quanto le due PERFORM attive hanno lo stesso punto di uscita:

```
A. PERFORM C THRU F.
B.
C. PERFORM D THRU F.
D.
E.
F.
```

b. Questo caso è invece ammesso in quanto le uscite sono chiaramente differenziate:

```
A. PERFORM C THRU F2.
B.
C. PERFORM D THRU F1.
D.
E.
F.
F1. EXIT.
F2. EXIT.
```

7.9. Trattamento delle stringhe di caratteri

7.9.1. *Il verbo INSPECT*

Attraverso il verbo INSPECT si possono contare (fig 7.16) o sostituire (fig. 7.17) oppure contare e sostituire (fig. 7.18) uno o più caratteri in un campo.

Esaminiamo i singoli formati senza spingerci a fondo nei dettagli.

Fig. 7.16

FORMATO 1 (per contare):

INSPECT nome-dato$_1$ TALLYING

nome-dato$_2$ FOR {vedi dettaglio} [vedi dettaglio] ...

[nome-dato$_5$ FOR {vedi dettaglio} [vedi dettaglio] ...]...

dettaglio:

{ALL / LEADING / CHARACTERS} {nome-dato$_3$ / costante$_1$} [{BEFORE / AFTER} INITIAL {nome-dato$_4$ / costante$_2$}]

7.9.1.1. *Formato 1*

a. Nome-dato$_1$ può essere un campo elementare o di gruppo di qualunque categoria.
b. Nome-dato$_3$ deve essere un campo elementare alfabetico o numerico o alfanumerico.
c. Le costanti devono essere non numeriche e possono essere costanti-figurative (eccetto ALL).
d. In generale (Cobol livello 1) è richiesto che le costanti e nome-dato$_3$, nome-dato$_4$ ecc.... siano costituite da un solo carattere.
e. Nome-dato$_2$ deve essere un campo elementare numerico.
f. L'analisi di nome-dato$_1$ comincia dal carattere più a sinistra e procede verso destra carattere per carattere.
g. Il contenuto del contatore nome-dato$_2$ non è inizializzato dall'avvio della INSPECT.
h. Se è indicato ALL il contatore nome-dato$_2$ viene incrementato di uno ogni volta che viene individuato il carattere espresso da costante$_1$ o contenuto in nome-dato$_3$.
i. Se è indicato LEADING il contatore nome-dato$_2$ viene incrementato di uno in corrispondenza di ognuno dei caratteri, specificati da costante$_1$, o dal contenuto di nome-dato$_3$, contigui a partire dall'inizio di nome-dato$_1$.

l. Se è indicato CHARACTERS il contatore viene incrementato di uno in corrispondenza di ogni carattere incontrato.
m. Se manca la frase AFTER/BEFORE il procedimento interessa tutto il campo nome-dato$_1$.
n. Se è indicato BEFORE l'analisi di nome-dato$_1$ comincia dal carattere più a sinistra e termina in corrispondenza del primo carattere (escluso) eguale alla costante$_2$ o al contenuto di nome-dato$_3$
o. Se è indicato AFTER l'analisi inizia a partire dal carattere immediatamente seguente il primo carattere trovato uguale a costante$_2$ o al contenuto di nome-dato$_3$.

Esempio:

Siano dati i campi VALORE e CONT rispettivamente contenenti:

```
VALORE : ABCDEAAABBCXEE
CONT   : 0
```

l'istruzione:

```
INSPECT VALORE TALLYING CONT FOR ALL "A".
```

lascia inalterato il contenuto di VALORE e assegna a CONT il contenuto 4:

```
VALORE : ABCDEAAABBBCXEE
CONT   : 4
```

Partendo dagli stessi contenuti iniziali l'istruzione

```
INSPECT VALORE TALLYING CONT FOR LEADING "A"
        AFTER INITIAL "E".
```

lascia invariato il contenuto di VALORE e assegna a CONT il valore 3:

```
VALORE : ABCDE|AAABBBCXEE
CONT   : 3
```

Partendo dagli stessi contenuti iniziali l'istruzione:

```
INSPECT VALORE TALLYING CONT FOR CHARACTERS
        BEFORE INITIAL "X".
```

lascia invariato il contenuto del campo VALORE e pone 12 in CONT:

```
VALORE : ABCDEAAABBBC XEE
CONT   : 12
```

7.9.1.2. *Formato 2*

a. Nome-dato$_1$ può essere un campo elementare o di gruppo di qualunque categoria.
b. Le costanti devono essere non numeriche e possono essere figurative (eccetto ALL).
c. In generale (Cobol livello 1) è richiesto che le costanti e nome-dato$_2$, nome-dato$_3$, nome-dato$_4$ ... siano costituiti da un solo carattere.
d. Il trattamento di nome-dato$_1$ comincia dal carattere più a sinistra e procede verso destra carattere per carattere.
e. Se è usata la frase CHARACTERS ogni carattere trattato è sostituito con costante$_1$ oppure con il contenuto di nome-dato$_2$.

Fig. 7.17

FORMATO 2 (per sostituire):

INSPECT nome-dato$_1$ REPLACING

{ CHARACTERS BY { nome-dato$_2$ / costante$_1$ } [vedi dettaglio$_1$]

{ { ALL / LEADING / FIRST } { vedi dettaglio$_2$ } [vedi dettaglio$_2$] ... { { ALL / LEADING / FIRST } { vedi dettaglio$_2$ } [vedi dettaglio$_2$] ... } ... }

dettaglio$_1$: { BEFORE / AFTER } INITIAL { nome-dato$_3$ / costante$_2$ }

dettaglio$_2$: { nome-dato$_4$ / costante$_3$ } BY { nome-dato$_5$ / costante$_4$ } [vedi dettaglio$_1$]

f. Se è specificato ALL, ogni qual volta viene incontrato, il carattere costante$_3$ oppure il contenuto di nome-dato$_4$, viene sostituito con costante$_4$, oppure con il contenuto di nome-dato$_5$.
g. Se è specificato LEADING il primo gruppo di caratteri uguali, contigui, uguali a costante$_3$ oppure al contenuto di nome-dato$_4$ viene sostituito con un gruppo di caratteri tutti uguali a costante$_4$ oppure al contenuto di nome-dato$_5$.
h. Se è specificato FIRST la sostituzione con costante$_4$ oppure con il contenuto di nome-dato$_5$ riguarda solo il primo carattere trovato uguale a costante$_3$ oppure al contenuto di nome-dato$_4$.
i. Le opzioni AFTER o BEFORE stanno ad indicare che l'analisi del campo nome-dato$_1$ deve essere eseguita solo a partire dal carattere seguente quello specificato, oppure fino a quello precedente quello specificato.

Esempio:

A partire dal seguente CAMPO con contenuto iniziale:

```
CAMPO : AABBBCDDABBCCCD
```

con l'istruzione:

```
INSPECT CAMPO REPLACING CHARACTERS BY "X"
        BEFORE INITIAL "C".
```

otteniamo:

```
CAMPO : XXXXXCDDABBCCCD
```

invece, dallo stesso contenuto iniziale, con l'istruzione

```
INSPECT CAMPO REPLACING ALL "B" BY "Y"
```

otteniamo:

```
CAMPO : AAYYYCDDAYYCCCD
```

Se infine partiamo dagli stessi valori iniziali, l'istruzione:

```
INSPECT CAMPO REPLACING FIRST "D" BY "Z"
        AFTER INITIAL "B".
```

consente di ottenere:

```
CAMPO : AABBBCZDABBCCCD
```

7.9.1.3. *Formato 3*

Per quanto riguarda il formato 3, che come si può osservare è la combinazione dei due precedenti, le considerazioni da fare sul significato delle parole Cobol, delle costanti e dei campi che intervengono sono deducibili da quanto esposto in 7.9.1.1. e 7.9.1.2.

Fig. 7.18

FORMATO 3 (per contare e sostituire):

```
INSPECT nome-dato1 TALLYING
nome-dato2 FOR { vedi          } [ vedi          ] ...
               { dettaglio1    }   dettaglio1
[ nome-dato8 FOR { vedi          } [ vedi          ] ... ] ...
                 { dettaglio1    }   dettaglio1
REPLACING
{ CHARACTERS BY { nome-dato6 } [ vedi          ]
                { costante4  }   dettaglio2
{ { ALL     } { vedi          } [ vedi          ] ... { { ALL     } { vedi          } [ vedi          ] ... }
  { LEADING } { dettaglio3    }   dettaglio3          { { LEADING } { dettaglio3    }   dettaglio3          }
  { FIRST   }                                         { { FIRST   }                                         }
}

[dettaglio1] : { { ALL     } { nome-dato3 } } [ vedi          ]
               { { LEADING } { costante1  } }   dettaglio2
               { { CHARACTERS             }

[dettaglio2] : { BEFORE } INITIAL { nome-dato4 }
               { AFTER  }         { costante2 }

[dettaglio3] : { nome-dato5 } BY { nome-dato6 } [ vedi          ]
               { costante3  }    { costante4  }   dettaglio2
```


Esempi:

a. Dati i campi:

```
AREA : LAXXXXLLL
CT1  : 0
CT2  : 0
```

l'istruzione:

```
INSPECT AREA TALLYING CT1 FOR ALL "L"
BEFORE INITIAL "X"
CT2 FOR LEADING "X" AFTER INITIAL "X"
REPLACING FIRST "X" BY "Y"
AFTER INITIAL "X".
```

permette di ottenere:

```
AREA : LAXYXXLLL
CT1  : 1
CT2  : 3
```

b. Dati i campi:

```
AREA  : RXXBQWYbbbb
COUNT : 0
```

l'istruzione:

```
INSPECT AREA TALLYING COUNT LEADING SPACE
AFTER INITIAL SPACE
REPLACING ALL "X" BY "B"
SPACE BY "*" AFTER INITIAL SPACE.
```

consente di ottenere:

```
AREA  : RBBBQWYb***
COUNT : 3
```

7.9.2. *Il verbo STRING*

L'istruzione STRING consente di fondere i contenuti parziali o completi di due o più campi in un singolo campo; questo è il suo formato:

```
STRING {nome-dato1} [{nome-dato2} ... DELIMITED BY {nome-dato3}]
       {costante1}  [{costante2}                   {costante3}
                                                   {SIZE      }
  [{nome-dato4} [{nome-dato5} ... DELIMITED BY {nome-dato6}]
   {costante4}  [{costante5}                   {costante6}
                                               {SIZE      }]
  INTO nome-dato7 [WITH POINTER nome-dato8]
  [ON OVERFLOW ..... .]
```

a. In questo formato nome-dato$_1$, nome-dato$_2$, nome-dato$_4$ e nome-dato$_5$ sono i campi emittenti, nome-dato$_7$ è il campo ricevente.

b. I contenuti dei campi emittenti vengono trasferiti, con trasferimento di tipo alfanumerico, senza riempimento con spazi della parte eccedente a destra, e quindi raggruppati nel campo ricevente secondo diverse modalità.

c. Se è indicato DELIMITED BY SIZE ogni singolo carattere dei campi emittenti corrispondenti è trasferito nel campo ricevente; i campi emittenti sono trattati nell'ordine in cui sono elencati; il trasferimento riguarda tutti i campi elencati e termina o per campi emittenti esauriti (senza riempimento della parte residua del campo ricevente con spazi) oppure per esaurimento dei campi emittenti.

d. Se è indicato DELIMITED BY nome-dato$_x$ oppure costante$_x$ il trasferimento da ogni singolo campo emittente cessa non appena si incontrano il carattere o il gruppo di caratteri specificati da costante$_x$ o contenuti in nome-dato$_x$; i caratteri che segnalano il fine trasferimento non sono trattati.

e. Quando è specificata la clausola WITH POINTER, nome-dato$_8$ dovrà essere definito come un campo elementare numerico in grado di contenere un valore pari alla dimensione più uno, del campo ricevente nome-dato$_7$; nome-dato$_8$ punta costantemente alla posizione di nome-dato$_7$ che dovrà essere caricata: dovrà quindi essere inizializzato con un valore maggiore o uguale a uno; nome-dato$_8$ ad ogni trasferimento è incrementato automaticamente.

f. Se è presente l'opzione ON OVERFLOW... nel caso in cui il

campo ricevente non sia sufficientemente grande per contenere i valori previsti, il trasferimento cessa ed il controllo passa alla prima istruzione scritta dopo la parola OVERFLOW; se l'opzione è presente, ed il trasferimento avviene, regolarmente il controllo passa alla prima istruzione scritta dopo il primo punto; in assenza del test di overflow, se lo spazio disponibile su nome-dato$_7$ non è sufficiente per contenere i dati che è previsto trasferire, il controllo, esauriti i trasferimenti possibili, passa alla prossima istruzione.

Esempi:

a. A partire dai campi A, B, C, D, T con i seguenti contenuti iniziali:

```
A : ROMAbb
B : DEbFRANCESCHI
C : 1530
D : 00151
T : ABCDEFGHILMNOPQRSTUVZABCDEFGH
```

l'istruzione:

```
STRING A B DELIMITED BY SPACE
       C DELIMITED BY ZERO
       D DELIMITED BY SIZE
       INTO T.
```

consente di ottenere:

```
T : ROMADE15300151QRSTUVZABCDEFGH
```

b. A partire dai campi A, B, C, P con i seguenti contenuti iniziali:

```
A : 1234
B : 5678910
C : ABCDEFGHILMNOPQRSTUVZ
P : 5
```

l'istruzione

```
STRING A DELIMITED BY SIZE
       B DELIMITED BY "9"
       INTO C WITH POINTER P.
```

consente di ottenere:

C : ABCD12345678OPQRSTUVZ
P : 13

7.9.3. *Il verbo UNSTRING*

L'istruzione UNSTRING consente di individuare e separare, in distinti campi riceventi, informazioni logiche contigue raggruppate in un unico campo emittente; il suo formato è:

UNSTRING nome-dato$_1$

[DELIMITED BY [ALL] {nome-dato$_2$ / costante$_1$}

[OR [ALL] {nome-dato$_3$ / costante$_2$}] ...] INTO nome-dato$_4$

[DELIMITER IN nome-dato$_5$] [COUNT IN nome-dato$_6$]

[nome-dato$_7$ [DELIMITER IN nome-dato$_8$] [COUNT IN nome-dato$_9$]]...

[WITH POINTER nome-dato$_{10}$] [TALLYING IN nome-dato$_{11}$]

[ON OVERFLOW ... .]

a. In questo formato nome-dato$_1$ è il campo emittente, nome-dato$_4$, nome-dato$_7$, ecc... sono i campi riceventi.
b. Le frasi DELIMITER IN e COUNT IN possono essere utilizzate solo se si utilizza la DELIMITED BY.
c. Al momento dell'avvio dell'esecuzione dell'istruzione il campo ricevente corrente è nome-dato$_4$, quindi, al verificarsi di determinate condizioni, diventa nome-dato$_7$, ecc...
d. Se è stata specificata la clausola POINTER il campo emittente viene esaminato dalla posizione individuata da nome-dato$_{10}$ (campo numerico intero che può essere inizializzato dal programmatore); in assenza della clausola POINTER nome-dato$_1$ viene esaminato a partire dal primo carattere di sinistra.
e. Se è specificata la clausola DELIMITED BY l'esame di nome-dato$_1$ procede da sinistra verso destra fino all'incontro di uno dei

delimitatori nome-dato$_2$, costante$_1$, nome-dato$_3$, ecc. ALL, sta ad indicare che una o più ricorrenze contigue del delimitatore sono trattate come se comparissero una sola volta; in assenza di questa clausola il numero dei caratteri esaminati in nome-dato$_1$ è pari alla dimensione del campo ricevente corrente; se durante tale analisi viene riscontrata la fine di nome-dato$_1$, l'esame si arresta con l'ultimo carattere preso in considerazione.
f. I caratteri di nome-dato$_1$, esaminati in relazione ad un certo campo ricevente, vengono considerati come un dato di tipo alfanumerico e trasferiti nel campo ricevente, che può essere di categoria numerica, alfabetica o alfanumerica, secondo le regole della MOVE.
g. Se in corrispondenza del campo ricevente corrente è specificata la clausola DELIMITER IN gli eventuali delimitatori, considerati come un campo elementare alfanumerico, sono trasferiti secondo le regole della MOVE nel campo indicato, che deve essere di categoria alfanumerica.
h. Se è specificata la clausola COUNT IN viene portato nel campo indicato, che deve essere di categoria numerica, il numero dei caratteri trattati per il campo ricevente corrente (il numero dei delimitatori è escluso dalle considerazioni).
i. Dopo il trasferimento (dei caratteri analizzati) in nome-dato$_4$, il campo nome-dato$_7$ diventa campo ricevente corrente e per esso si ripetono le stesse considerazioni, con questa differenza: l'analisi di nome-dato$_1$ inizierà dal primo carattere alla destra del delimitatore che aveva arrestato il trasferimento in nome-dato$_4$ o, se non era stata specificata la clausola DELIMITED BY, dal carattere seguente l'ultimo trasferito; tale procedimento continua fino all'esaurimento dei caratteri di nome-dato$_1$ oppure fino all'esaurimento dei campi riceventi.
l. Durante l'esecuzione dell'istruzione il contenuto di nome-dato$_{10}$ è automaticamente incrementato di uno in corrispondenza di ciascun carattere esaminato in nome-dato$_1$; al termine della esecuzione dell'istruzione tale campo risulta incrementato del numero di caratteri esaminati.
m. Se è stata specificata l'opzione TALLYING IN alla fine dell'esecuzione nome-dato$_{11}$ risulta incrementato di un valore uguale al numero dei campi riceventi trattati.
n. Si ha rilevazione di overflow (se è stata specificata la ON OVER-

FLOW) se il valore iniziale di nome-dato$_{10}$ non consente di individuare un carattere della stringa (< 1 oppure > numero di caratteri), oppure se sono esauriti tutti i campi riceventi prima dell' analisi completa di nome-dato$_1$.

Esempio:

Siano dati i seguenti campi (con relativi contenuti):

```
GRUPPO : AAA*BBbbbC*Db123*E
GR1    : ABCABC
GR2    : ABCABC
GR3    : ABCABC
GR4    : ABCABC
GR5    : 12345
NUM1   : XYZ
NUM4   : LK3
CONTA1 : 1
CONTA5 : 9
CTC    : 2
CTR    : 4
```

l'istruzione:

```
UNSTRING GRUPPO
   DELIMITED BY ALL SPACE OR "*"
   INTO GR1 DELIMITER IN NUM1 COUNT IN CONTA1
   GR2 GR3 GR4 DELIMITER IN NUM4
   GR5 COUNT IN CONTA5
   WITH POINTER CTC TALLYING CTR.
```

permette di ottenere:

```
GRUPPO : AAA*BBbbbC*Db123*E
GR1    : AAbbbb
GR2    : BBbbbb
GR3    : Cbbbbb
GR4    : Dbbbbb
GR5    : 00123
NUM1   : *bb
NUM4   : bbb
CONTA1 : 2
CONTA5 : 3
CTC    : 16
CTR    : 9
```


7.10. *Il verbo ENTER*

In Cobol è possibile inserire, nel corpo del programma sorgente, in PROCEDURE DIVISION, gruppi di istruzioni in un altro linguaggio (di solito del linguaggio assemblatore dell'elaboratore che si sta utilizzando); a questo scopo si utilizza il verbo ENTER che ha questo formato:

```
ENTER nome-linguaggio
```

La possibilità di utilizzare, tramite una ENTER, un determinato linguaggio deve essere prevista dal software del sistema: la parte di codifica che segue la ENTER sarà scritta secondo le regole del linguaggio richiamato e in essa si potrà fare riferimento ad aree definite in DATA DIVISION e/o ad aree definite al momento, se il linguaggio richiamato lo consente.

Dopo l'ultima istruzione scritta nel linguaggio specificato è prescritta la frase

```
ENTER COBOL
```

che indica al compilatore il ritorno alla codifica in linguaggio Cobol.

In fig. 7.19 è riportato in forma schematica un esempio di utilizzo del verbo ENTER.

7.11. *Esempio*

Il fig. 7.20 è rappresentato l'organigramma del problema che vogliamo ora risolvere.

Mediante i valori espressi dai record del flusso MOV, movimenti, vogliamo aggiornare l'archivio prodotti ARC (generando l'archivio aggiornato ARC-A) e stampare la situazione per prodotti.

In fig. 7.21 sono rappresentati il tracciato della scheda parametro e quello del record archivio (ARC e ARC-A) e quello del record del flusso MOV; in fig. 7.22 invece è riportato il tracciato stampa.

Analizziamo dettagliatamente le specifiche del problema.

a. Flussi in input:

- ARC: i record sono ordinati su ZONA, MAGAZZINO ed ARTICOLO

Fig. 7.19

```
PROCEDURE DIVISION.
.
.
```

Fig. 7.22

- MOV: l'ordinamento è sulle stesse chiavi del flusso ARC, il campo QTA può contenere sia valori positivi che negativi

b. Flussi in output:

- ARC-A: ha le stesse caratteristiche del flusso ARC
- TABUL: è un flusso in stampa riepilogativo della situazione per zona in un certo periodo (fornito da scheda parametro). La stampa consiste in:

  1. righe di intestazione in cui si riporta il codice zona, il periodo considerato, il numero di foglio nell'ambito della zona, il numero di foglio nell'ambito di tutto il tabulato,
  2. righe dettaglio per articolo, organizzate come da tracciato,
  3. righe di totalizzazione per magazzino e per zona.

Ogni pagina contiene un massimo di 50 righe dettaglio, le righe di totalizzazione rientrano comunque nella pagina a cui si riferiscono, il cambio pagina viene inoltre effettuato al variare del codice di zona.

c. Elaborazione:

1. Viene richiesta l'introduzione di una scheda parametro di tipo "DD" per fornire le date del periodo desiderato (riga intestazione): verificare che sia presente.
2. Leggere in modo parallelo i due flussi ARC e MOV e, a parità di chiavi aggiornare la giacenza.
3. Non è prevista la presenza di movimenti senza il corrispondente record d'archivio.
4. Vanno portati in output (ARC-A) anche i record d'archivio non movimentati.
5. Ogni elemento d'archivio, anche non movimentato dà luogo alla stampa della situazione: evidenziare con un asterisco nell' apposita colonna il caso di giacenza inferiore alla scorta minima.
6. A fine elaborazione fornire, su foglio nuovo, la valorizzazione totale (cioè la somma dei totali per zona).

Nelle fig. 7.23 e 7.24 sono riportati rispettivamente il flowchart e la codifica del problema proposto.

Fig. 7.23a

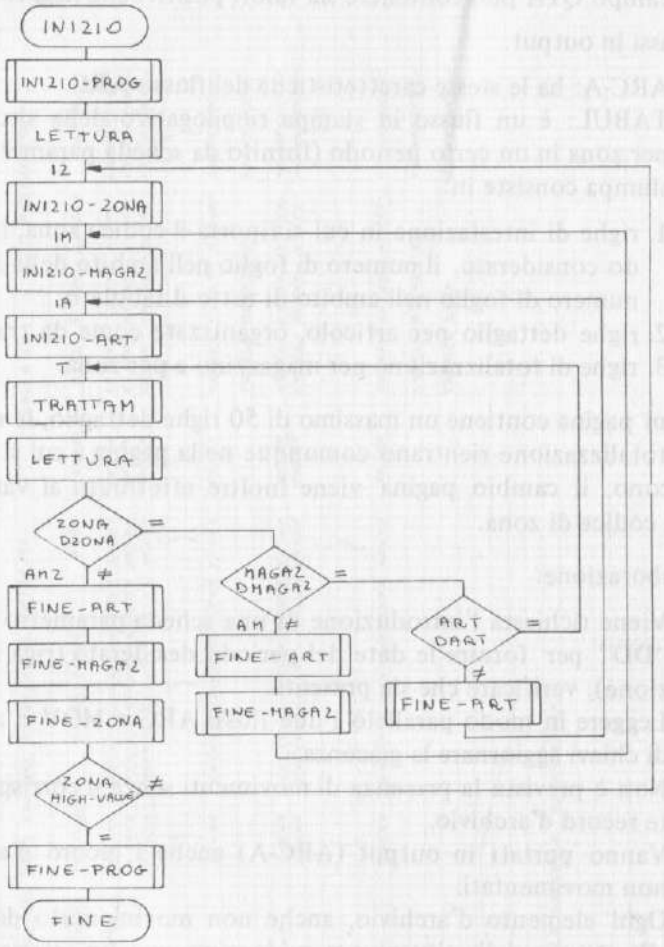


Fig. 7.23b

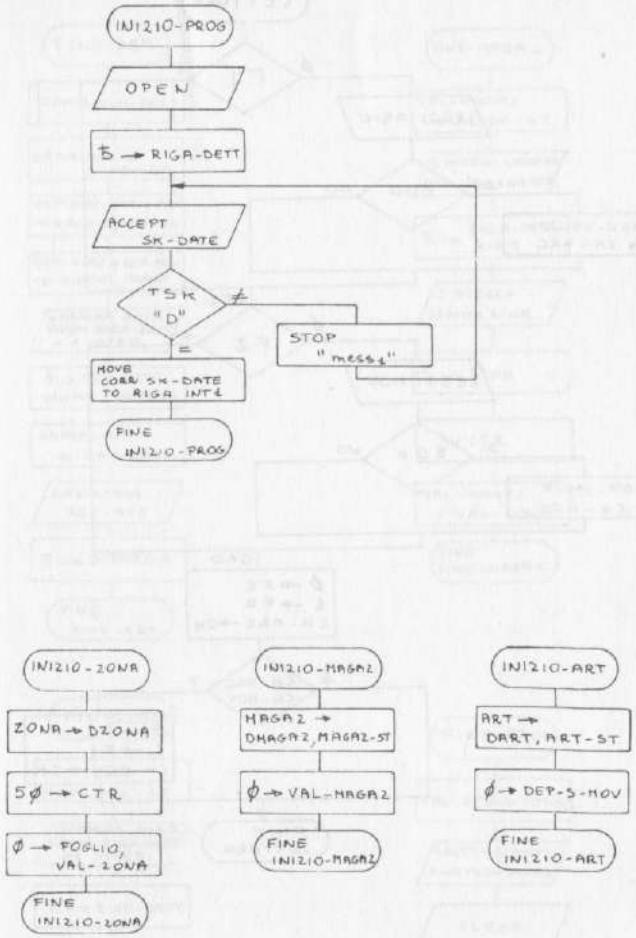


Fig. 7.23c

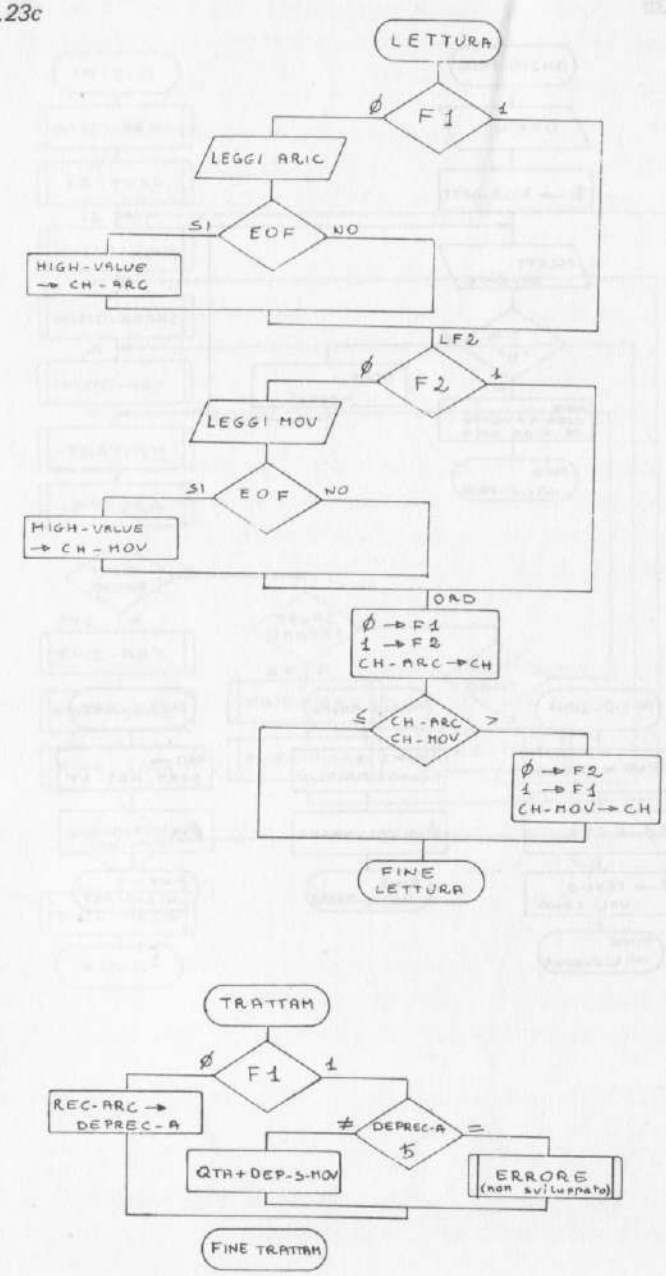


Fig. 7.23d

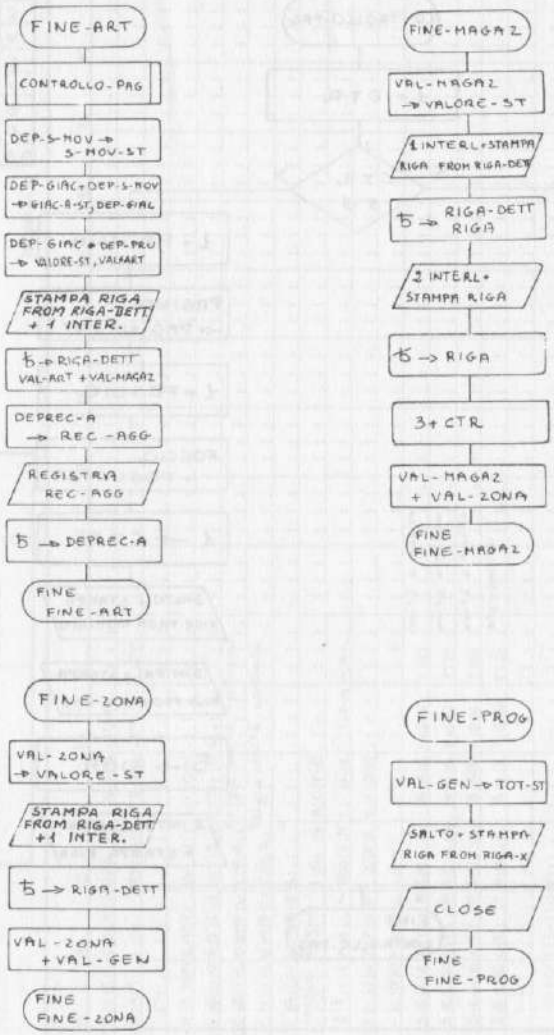


Fig. 7.23e

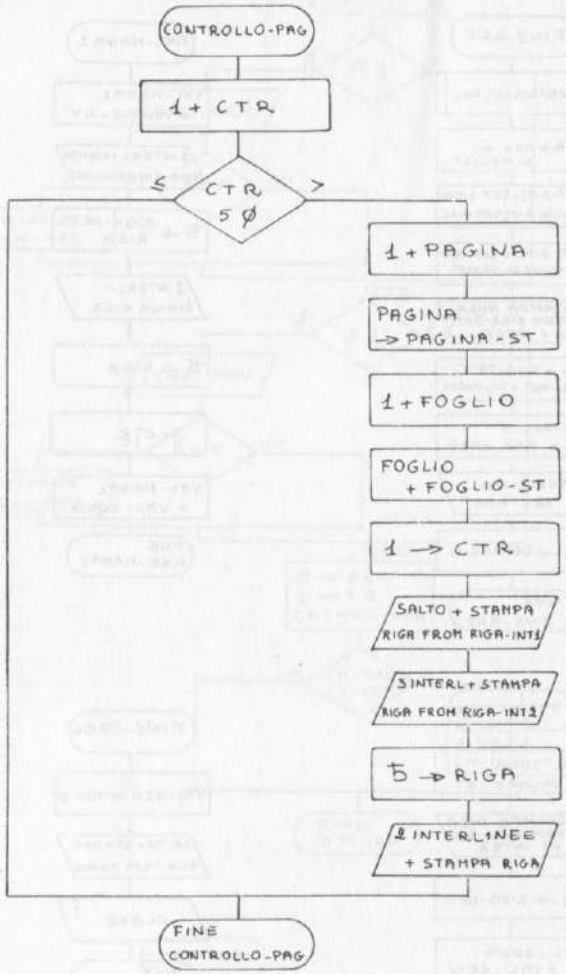


Fig. 7.24a

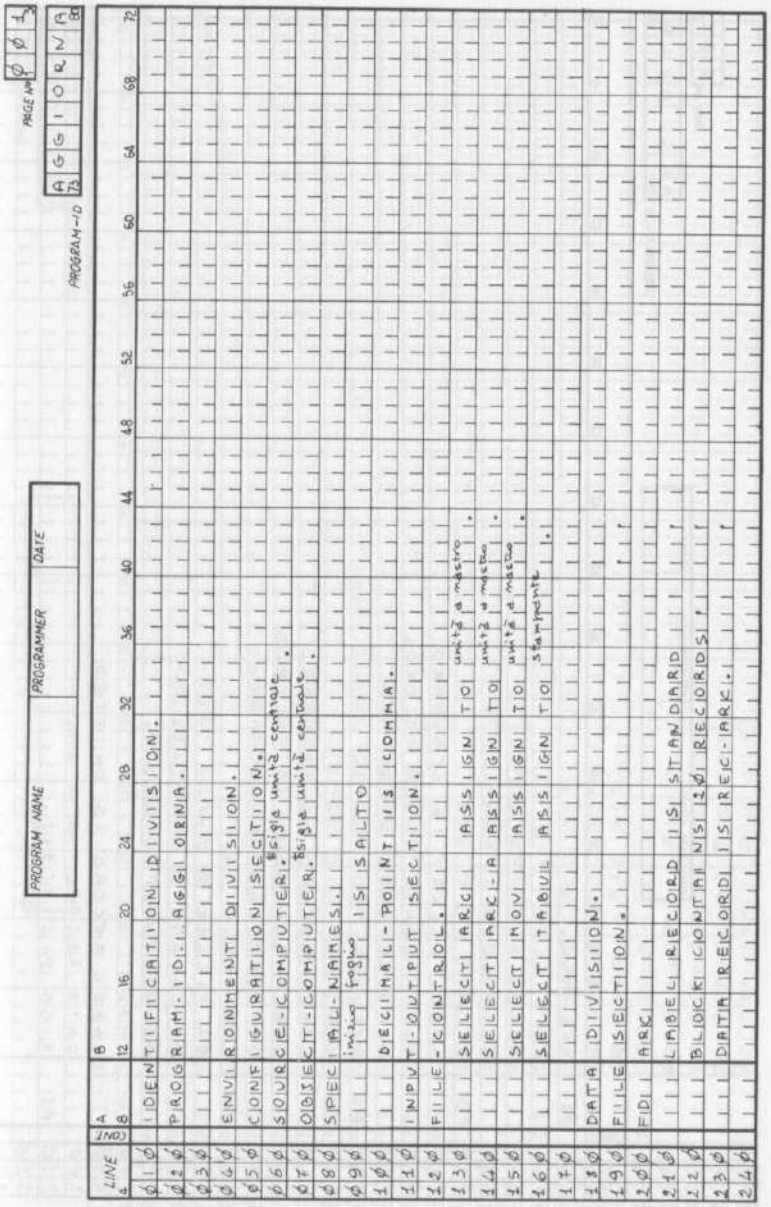


Fig. 7.24b

PAGE No. 002

PROGRAM NAME	PROGRAMMER	DATE

PROGRAM-ID AGGIORNA

LINE	CONT.	A	B
010		01	RECI-ARCI.
020			02 FILLER PIC X.
030			02 CHI-ARCI PIC X(11).
040			02 FILLER PIC X(315).
050		FD	ARCI-A
060			LABEL RECORD IS STANDARD
070			BLOCK CONTAINS 20 RECORDS
080			DATA RECORD IS RECI-AGGI.
090		01	RECI-AGGI PIC X(417).
100		FD	MOV
110			LABEL RECORD IS STANDARD
120			BLOCK CONTAINS 30 RECORDS
130			DATA RECORD IS RECI-MOV.
140		01	RECI-MOV.
150			02 FILLER PIC X.
160			02 CHI-MOV PIC X(11).
170			02 KTA PIC S9(15).
180		FD	TABULI
190			LABEL RECORD IS OMITTED
200			DATA RECORD IS RIGA.
210		01	RIGA PIC X(91).
220			
230			
240			

Fig. 7.24c

PAGE No. 003

PROGRAM NAME	PROGRAMMER	DATE

PROGRAM-ID AGGIORNA

LINE	CONT.	A	B
010		WORK	ING-STORAGE SECTION.
020		77	F1 PIC S VALUE ZERO.
030		77	F2 PIC S VALUE ZERO.
040		77	CTR PIC S9.
050		77	PAGINA PIC S99 VALUE
060		77	FOGLIO PIC S9.
070		77	VALI-MAGA PIC S9(18).
080		77	VALI-ZONA PIC S9(18).
090		77	VALI-GEN PIC S9(18) VALUE ZERO.
100		77	VALI-ART PIC S9(18).
110		77	DEPI-SMOV PIC S9(15).
120		01	CHI.
130			02 ZONA PIC XXX.
140			02 MAGA PIC XXX.
150			02 ART PIC X(15).
160		01	DCHI.
170			02 DZONA PIC XXX.
180			02 DMAGA PIC XXX.
190			02 DART PIC X(5).
200		01	DEPRECI-A.
210			02 FILLER PIC X.
220			02 DEPI-ZONA PIC XXX.
230			02 DEPI-MAGA PIC XXX.
240			02 DEPI-ART PIC X(15).

Fig. 7.24d

PAGE NO. 004

PROGRAM NAME	PROGRAMMER	DATE

PROGRAM-ID AGGIORNA

LINE	CONT.	A	B
010			02 DEPI-DESCR PICX(120).
020			02 DEPI-PRIV PICS9(15).
030			02 DEPI-GIAC-MIN PICS9(15).
040			02 DEPI-GIAC PICS9(15).
050		01	RIGA-INIT 1.
060			02 FILLER PICX(188) VALUE " ISTIZIONE ZONA ".
070			02 ZONA-STI PICXXX.
080			02 FILLER PICX(15) VALUE " DALL ".
090			02 G1 PIC Z9.
100			02 FILLER PICX VALUE " / ".
110			02 M1 PIC 99.
120			02 FILLER PICX VALUE " / ".
130			02 A1 PIC 99.
140			02 FILLER PICX(4) VALUE " AL ".
150			02 G2 PIC Z9.
160			02 FILLER PICX VALUE " / ".
170			02 M2 PIC 99.
180			02 FILLER PICX VALUE " / ".
190			02 A2 PIC 99.
200			02 FILLER PICX(44) X(17) VALUE " FOGLIO ".
210			02 FOGLIO-IST PIC Z9.
220			02 FILLER PIC X(44)X(17) VALUE " PAGINA ".
230			02 PAGINA-IST PIC Z29.
240			

Fig. 7.24e

PAGE NO. 005

PROGRAM NAME	PROGRAMMER	DATE

PROGRAM-ID AGGIORNA

LINE	CONT.	A	B
010		02	RIGA-INIT 2 PIC X(89) VALUE " MAGAZZINO ARTICOLO DESCRIZIONE ".
020	-		" IONE PRIV IGIAC-MIN GIAC ISI-MOV IGIAC-A VALORE ".
030		02	RIGA-DEITT.
040		02	FILLER PIC X(15).
050		02	MAGAZ-IST PIC XXX.
060		02	FILLER PIC X(6).
070		02	ART-IST PIC X(15).
080		02	FILLER PIC XXX.
090		02	DESCR-IST PIC X(20).
100		02	FILLER PIC X.
110		02	PRIV-IST PIC Z(15).
120		02	FILLER PIC XXX.
130		02	GIAC-MIN-IST PIC Z(15).
140		02	FILLER PIC X(4).
150		02	GIAC-IST PIC Z(15)-.
160		02	FILLER PIC X.
170		02	S-MOV-IST PIC Z(15)-.
180		02	FILLER PIC X.
190		02	GIAC-A-IST PIC Z(15)-.
200		02	FILLER PIC X.
210		02	VALORE-IST PIC Z.ZZZ.ZZ9-.
220			
230			
240			

Fig. 7.24f

PAGE NO. 006

PROGRAM NAME	PROGRAMMER	DATE

PROGRAM-ID AGGIORNA

LINE	CONT.	A	B
010		01	RIIGA-X1.
020			02 IFILLER PIC X(131) VALUE "TOTALE GENERALE".
030			02 TOT-STI PIC ZZZ.ZZZ.ZZ9-.
040		01	SKI-DATE.
050			02 TSKI PIC X.
060			02 G1 PIC 99.
070			02 M1 PIC 99.
080			02 A1 PIC 99.
090			02 G2 PIC 99.
100			02 M2 PIC 99.
110			02 A2 PIC 99.
120			
130		PROCEDURE DIVISION.	
140		FLOW-MASSIMA SECTION.	
150		INIZIO.	
160		PERFORM INIZIO-PROG	
170		PERFORM LETTURA.	
180		12.	PERFORM INIZIO-ZONA.
190		IM.	PERFORM INIZIO-MAGAZ.
200		IA.	PERFORM INIZIO-ART.
210		T.	PERFORM TRATTAM
220			PERFORM LETTURA
230			IF ZONA NOT = DZONA GOTO AM2.
240			IF MAGAZ NOT = DMAGAZ GOTO AM.

Fig. 7.24g

PAGE NO. 007

PROGRAM NAME	PROGRAMMER	DATE

PROGRAM-ID AGGIORNA

LINE	CONT.	A	B
010			IF ART = DART GOTO T.
020			PERFORM FINE-ART GOTO A.
030		AM.	PERFORM FINE-ART
040			PERFORM FINE-MAGAZ GOTO M.
050		AM2.	PERFORM FINE-ART
060			PERFORM FINE-MAGAZ
070			PERFORM FINE-ZONA
080			IF ZONA NOT = HIGH-VALUE GOTO 12.
090			PERFORM FINE-PROG
100			STOP-RUN.
110			
120		INIZIO-PROG SECTION.	
130			OPEN INPUT ARCHIVIO
140			OPEN OUTPUT ARCHIVIO-ATTIVO
150			MOVE SPACE TO RIGA-DETTA.
160		ASI.	ACCEPT SKI-DATE
170			IF TSKI NOT = "DI" STOP "ERROREE" GOTO ASI.
180			MOVE CORR SKI-DATE TO RIGA-INTES.
190		EXIT.	
200			
210		LETTURA SEGCTION.	
220			IF F1 = 1 GOTO LF2.
230			READ ARC AT END MOVE HIGH-VALUE TO CHI-ARC.
240		LF2.	IF F2 = 1 GOTO ORD.

Fig. 7.24h

PAGE NO. 008

PROGRAM NAME	PROGRAMMER	DATE

PROGRAM-ID AGGIORNA

LINE	CONT.	A	B
010			READ MOV AT END MOVE HIGH-VALUES TO CH-MOV.
020		ORD.	MOVE ZERO TO F2
030			MOVE 1 TO F2
040			MOVE CH-ART TO CH
050			IF CH-ART > CH-MOV
060			MOVE ZERO TO F2
070			MOVE 1 TO F2
080			MOVE CH-MOV TO CH.
090		EX-LE	EXIT. EXIT.
100			
110		INIZ	IO-ZONA SELECTION.
120			MOVE ZONA TO DZONA
130			MOVE 500 TO CTR
140			MOVE ZERO TO FLG LIO VAL-ZONA.
150		EX-IN	NIZIO-ZONA. EXIT.
160			
170		INIZ	IO-MAGAZ SELECTION.
180			MOVE MAGAZ TO DMAGAZ MAGAZ-ST
190			MOVE ZERO VAL-MAGAZ.
200		EX-IN	NIZIO-MAGAZ. EXIT.
210			
220		INIZ	IO-ART SELECTION.
230			MOVE ART TO DART ART-ST
240			MOVE ZERO TO DEP-ST-MOV.

Fig. 7.24i

PAGE NO. 009

PROGRAM NAME	PROGRAMMER	DATE

PROGRAM-ID AGGIORNA

LINE	CONT.	A	B
010		EX-IN	NIZIO-ART. EXIT.
020			
030		FINE	-PROG SELECTION.
040			MOVE VAL-GEN TO TOT-ST
050			WRITE RIGA FROM RIGA-XAFTER ADVANCING SALTO
060			CLOSE ARC ARC-A MOVE TABULI
070		EX-FI	NE-PROG. EXIT.
080			
090		FINE	-ZONA SELECTION.
100			MOVE VAL-ZONA TO VALORE-ST
110			WRITE RIGA FROM RIGA-DETT BEFORE 1
120			MOVE SPACE TO RIGA DETTT
130		ADD	VALI-ZONA TO VAL-GEN.
140		EX-FI	NE-ZONA. EXIT.
150			
160		FINE	-MAGAZ SELECTION.
170			MOVE VAL-MAGAZ TO VALORE-ST
180			WRITE RIGA FROM RIGA-DETT AFTER 1
190			MOVE SPACE RIGA-DETTT RIGA
200			WRITE RIGA AFTER 2
210			MOVE SPACE TO RIGA
220		ADD	3 TO CTR
230		ADD	VAL-MAGAZ TO VAL-ZONA.
240		EX-FI	NE-MAGAZ. EXIT.

Fig. 7.24l

PROGRAM NAME | PROGRAMMER | DATE

PAGE Nº 010

PROGRAM-ID AGGIORNA

LINE	CONT.	A / B
010		FINE-PART SECTION.
020		PERFORM COINTROLLO-PAG
030		MOVE DEP-SI-MOVI TO SI-MOVI-STI
040		ADD DEPI-GIAC DEP-SI-MOVI GIVING GIACIA-ST DEPI-GIAC
050		MULTIPLY DEP-GIAC BY DEP-PRU GIVING VALORE-ST VALI-ART
060		WRITE RIGA FROM RIGA-DETTI BEFORE 1
070		MOVE SPACE TO RIGA-DETTI ADD VALI-ART TO VALI-MAGA2
080		MOVE DEP RECI-A TO RECI-MGG
090		WRITE RECI-AGG IMOVE SPACE TO DEP RECI-A.
100		EXIT INE-ART. EXIT.
110		
120		COINTROLLO-PAG SECTION.
130		ADD 1 TO CTR
140		IF CTR > SØ ADD 1 TO PAGINA
150		MOVE PAGINA TO PAGINA-ST
160		ADD 1 TO FOGLIO
170		MOVE FOGLIO TO FOGLIO-ST
180		MOVE 1 TO CTR
190		WRITE RIGA FROM RIGA-INT 1 AFTER ADVANCING SALTO
200		WRITE RIGA FROM RIGA-INT 2 AFTER 3 LINES
210		MOVE SPACE TO RIGA
220		WRITE RIGA AFTER 2 LINES.
230		EXIT COINTROLLO-PAG. EXIT.
240		

Fig. 7.24m

PROGRAM NAME | PROGRAMMER | DATE

PAGE Nº 011

PROGRAM-ID AGGIORNA

LINE	CONT.	A / B
010		TRATTAMI SECTION.
020		IF 1 = ZERO MOVE RECI-ARCITO DEPRECI-A GO TO EXIT-TRATTAMI
030		IF DEP-RECI-A NOT = SPACE ADD 1 TO DEP-SI-MOV
040		ELSE PERFORM ERRORE.
050		EXIT-TRATTAMI. EXIT.
060		
070		ERRORE SECTION.
080		STOP "ERRORE 1".
090		EXIT-ERRORE. EXIT.
100	*	ILLISOTTOPROGRAMMA INONTE' STATO SVILUPPATO PERCHE' E' PREVISTO
110	*	CHE ORIGINI MONTEMENTO ABBIA CORRISPONDENTE ANAGRAFICA.

8. LE TABELLE

L'uso delle tabelle nel campo del trattamento automatico dei dati non si differenzia sostanzialmente da quello della pratica gestionale non automatizzata; i motivi che inducono all'uso delle tabelle sono principalmente:

a. memorizzare una volta per tutte alcune informazioni di uso corrente a cui poter fare rapidamente riferimento al momento dell' utilizzo;
b. ridurre il volume delle informazioni circolanti sui documenti (su cui si riporta solo un codice che serve per reperire le altre informazioni in tabella): osserviamo che la riduzione delle informazioni registrate sui record consente di diminuire la possibilità di errore che sono tanto più probabili quanto più una informazione viene ripetuta.

Prima che un programma possa fare riferimento alle informazioni di tabella esse dovranno essere caricate in memoria in una zona opportunamente riservata e definita.

Nello standard Cobol è riservato un modulo (Table Handling) per fissare le modalità di definizione e di trattamento dei dati organizzati in tabelle: nelle pagine che seguono ne riportiamo le caratteristiche principali e alcuni esempi di utilizzo.

Esistono in Cobol due diversi modi per la gestione delle tabelle:

a. Subscripting: in cui la definizione e la gestione dell'indice di tabella è a completo carico del programmatore.
b. Indexing: in cui la definizione e la gestione dell'indice e del suo modo di puntare ad un elemento di tabella sono a carico del software.

8.1. Tabelle gestite dal programmatore

8.1.1. *Definizione*

Una tabella è un insieme di elementi omogenei e di uguale struttura: il concetto di ripetitività dell'elemento di tabella è realizzato in Cobol tramite la clausola OCCURS che consente di indicare che il generico elemento di tabella è ripetuto un "certo numero di volte" e che pertanto permette di "strutturare a tabella" un'area di memoria.

La definizione delle tabelle prevista dal Cobol è particolarmente significativa perché:

a. da un punto di vista documentativo sono evidenziati l'omogeneità e la struttura dei campi, in quanto si fornisce la struttura dettagliata di un generico elemento.
b. dal punto di vista pratico, in quanto la stesura della codifica risulta particolarmente semplice e veloce,
c. dal punto di vista logico, in quanto è molto semplice, tramite un indice associato al nome del generico elemento, individuare un ben preciso elemento di tabella anche quando la sua identificazione dipende dall'esecuzione del programma.

La OCCURS è una clausola di descrizione dati che permette di indicare quante volte un elemento (campo elementare o di gruppo) è ripetuto: si elimina così la necessità di introdurre una stessa descrizione per ogni campo ricorrente, si impone però la necessità di individuare un singolo elemento tramite un indice.

Il formato di codifica della OCCURS per la descrizione della ripetitività di un elemento non sottolivellato è:

```
numero-di-livello nome-dato OCCURS
        intero TIMES {PICTURE} IS ....
                     {PIC}
```

E' evidente che se l'elemento di tabella non è un campo di gruppo deve essere precisata sia la sua lunghezza, sia la sua categoria.

Se vogliamo indicare che un generico elemento di tabella, ripetuto un numero "intero" di volte, è sottolivellato basterà far seguire la

descrizione dei suoi campi componenti alla seguente frase dichiarativa di ripetitività:

numero-di-livello { nome-dato / FILLER } OCCURS intero TIMES .

Utilizzando questo formato, indicheremo nome-dato quando interessa fare riferimento all'intero elemento di tabella, per cui è necessario attribuirgli un nome, indicheremo invece FILLER quando interessano solo i sottocampi dell'elemento.

La clausola OCCURS non può essere specificata a livello 01, 66, 77 e 88.

In presenza della OCCURS non è possibile associare al nome del generico elemento, o ad una sua sottodefinizione, le clausole di descrizione dati VALUE e REDEFINES.

Il nome e le clausole descrittive (PICTURE, JUSTIFIED RIGHT, ...), ad eccezione della OCCURS stessa, si intendono associate a ciascun elemento di tabella ed ai suoi eventuali sottocampi.

E' importante sottolineare che la clausola OCCURS si limita semplicemente a riservare e descrivere la zona di memoria richiesta dalla tabella, il caricamento delle informazioni in tabella dovrà essere fatto esplicitamente nel programma.

Esempi:

a.

```
01 TABELLA.
   02 ELEM OCCURS 50 TIMES PIC X(25).
```

viene così definita in memoria un'area (di 1250 caratteri) di nome TABELLA che è strutturata in modo da contenere 50 volte il generico elemento (di nome ELEM, e di categoria alfanumerica e di lunghezza 25).

b.

```
01 TAB-PREZZI.
   02 ELEM-ART OCCURS 120 TIMES.
      03 COD-ART PIC X(7).
      03 PRU      PIC 9(6).
```


questa definizione consente di riservare un'area di memoria di 1560 caratteri (TAB-PREZZI) strutturata in modo da contenere il generico elemento (ELEM-ART) 120 volte, ELEM-ART è sottodefinito in due campi: COD-ART codice del generico elemento ed il relativo prezzo unitario PRU.

c.

```
01 TAB.
   02 FILLER OCCURS 15 TIMES.
      03 COD PIC X.
      03 SPED PIC X(30).
```

questa definizione consente di riservare un'area di memoria di 465 caratteri strutturata in modo da contenere 15 volte un generico elemento (di cui non viene precisato il nome in quanto non dovrà essere mai preso in considerazione l'intero elemento, ma ciascuno dei suoi sottocampi).

La clausola OCCURS può essere speficicata sia in WORKING-STORAGE SECTION sia in FILE SECTION.

Nel primo caso consente di dare struttura tabellare ad un'area di lavoro (è il caso dei tre esempi precedenti), nel secondo caso viene definita con struttura tabellare un'area record o, come è più frequente, un sottocampo di un'area record.

Esempio:

```
01 REC-DIP.
   02 MATR        PIC X(5).
   02 ANAG        PIC X(70).
   02 CATEG       PIC X.
   02 LORDO       PIC 9(8).
   02 TRAT-PREV   PIC 9(7).
   02 TRAT-ALIQUO OCCURS 8 TIMES PIC 9(7).
   02 NETTO       PIC 9(8).
```

In questo esempio si definisce una struttura di un'area record: oltre alla matricola del dipendente, informazioni anagrafiche e la categoria sono riportati il lordo progressivo annuo, le trattenute previdenziali, una tabella di trattenute (per aliquote) ed il netto.

8.1.2. *Il "particolare" elemento di tabella*

Abbiamo detto che per particolarizzare un generico elemento di tabella o un suo sottocampo bisogna far riferimento ad un indice (o subscript).

Gli indici sono valori interi positivi e possono essere espressi sia tramite costanti numeriche, sia tramite nomi-dato di categoria numerica; l'indice, comunque espresso, deve essere riportato fra parentesi rotonde dopo il nome-dato del campo a cui si intende riferirsi: fra il nome-dato della definizione e la prima parentesi deve essere inserito uno spazio.

Consideriamo ad esempio la seguente tabella:

```
01 TABELLA.
   02 EL-PROV OCCURS 150 TIMES.
      03 CAP        PIC 9(5).
      03 NOME-PR PIC X(15).
```

costituita da un massimo di 150 elementi ciascuno composto da un campo CAP e da un campo NOME-PR.

Volendo fare riferimento al quarto elemento o all'ottavo codice di avviamento postale o al quarantesimo nome provincia, scriveremo:

```
EL-PROV (4)
CAP (8)
NOME-PR (40)
```

oppure

```
EL-PROV (IND)
CAP (IND)
NOME-PR (IND)
```

dove IND è un nome-dato riferito ad un campo che nei vari casi contiene rispettivamente i valori: 4, 8, 40; IND è un campo di categoria numerica definito in FILE SECTION oppure in WORKING-STORAGE SECTION.

E' compito del programmatore assicurarsi, al fine di evitare errori durante l'elaborazione, che il valore dell'indice non ecceda il numero specificato nella OCCURS oppure che l'indice non assuma valore zero.

Quando il programmatore conosce a priori (a livello stesura del programma) la posizione dell'elemento a cui intende riferirsi utilizza come indici delle costanti numeriche; quando la posizione dell'elemento dipende dall'esecuzione del programma sarà necessario usare come indice un nome-dato ovvero un'area che può essere modificata dal programma.

Nel caso in cui il nome del generico elemento di una tabella, oppure di un suo sottocampo, sia già presente nel programma, è necessario provvedere alla qualificazione: in questo caso l'indice segue l'ultimo qualificatore.

Esempio:

```
EL OF REC-ARK (3)
COD OF TAB1 (VAL)
```

8.1.3. *Caricamento delle tabelle*

Prima che il programma possa accedere alle informazioni di tabella, occorre che il programma provveda a caricarvi tali informazioni.

Il caricamento può avvenire in due modi:

a. per tabelle di piccole dimensioni e di contenuto costante si può inizializzare, con i valori richiesti, un'area di memoria a cui si da struttura tabellare: il riferimento è a tabelle definite in WORKING-STORAGE SECTION ed inizializzate tramite la clausola VALUE;
b. per tabelle di notevoli dimensioni o di contenuto variabile si provvede, di solito, al caricamento tramite input da un flusso esterno.

8.1.3.1. *Caricamento con VALUE*

Questo tipo di preparazione delle tabelle consiste nella definizione di un'area di memoria con particolari contenuti (attribuiti mediante la clausola VALUE) e nella ridefinizione di tale area con struttura

tabellare: in questo modo i valori degli elementi di tabella sono assegnati già nel programma sorgente; questo procedimento è conveniente solo se il numero delle costanti da caricare non è molto elevato e soprattutto se i valori degli elementi di tabella non sono soggetti a frequenti variazioni (ricordiamo che apportare una variazione al programma sorgente implica che deve essere ricompilato).

Procedendo in questo modo non si è in contrasto con la regola che dice che in presenza della OCCURS non possono essere utilizzate le clausole VALUE e REDEFINES in quanto prima tramite la VALUE si carica con costanti un'area di memoria, quindi utilizzando la REDEFINES le si assegna un nuovo nome e solo in seguito tale campo è definito con struttura tabellare tramite la OCCURS.

Esempi:

a.

```
01 TAB.
   02 TAB-MESI PIC X(36)
      VALUE "GENFEBMAR .... NOVDIC".
   02 MESE REDEFINES TAB-MESI
      PIC X(3) OCCURS 12 TIMES.
```

b.

```
01 VALORI.
   02 FILLER PIC 9(4)V99 VALUE 5.3.
   02 FILLER PIC 9(4)V99 VALUE .07.
   02 FILLER PIC 9(4)V99 VALUE 124.1.
   02 FILLER PIC 9(4)V99 VALUE 7.
   02 FILLER PIC 9(4)V99 VALUE 3.14.
01 TAB-VAL REDEFINES VALORI.
   02 VAL PIC 9(4)V99 OCCURS 5 TIMES.
```

Osserviamo che nei due esempi sono state seguite tecniche diverse, infatti in a. la valorizzazione è stata fatta con un'unica VALUE, questo perché non interessava evidenziare il valore del singolo elemento, mentre in b., in quanto interessa evidenziare i valori delle singole costanti si è provveduto a definizioni separate.

Osserviamo inoltre che talvolta si è costretti ad effettuare definizioni separate in quanto come sappiamo (vedi 2) il Cobol prevede che al più una costante numerica possa essere lunga 18 cifre ed una alfanumerica 120.

Esempio:

Supponiamo che debba essere definita un'area di memoria atta a contenere una tabella di dodici elementi, ciascuno dei quali è costituito dal nome di un mese e da un'area numerica lunga dieci e contenente il valore zero.

In questo caso l'area da valorizzare è costituita da (9+ 10) * 12 =) 228 caratteri e pertanto è indispensabile inizializzarla con almeno due VALUE:

```
01 VAL-TAB.
   02 FILLER PIC X(114) VALUE "GENNAIObbb0000000000 ...
-  "..... GIUGNObbbb0000000000".
   02 FILLER PIC X(114) VALUE "LUGLIObbbbb0000000000 ...
-  "..... DICEMBREb0000000000".
01 TAB-MESI REDEFINES VAL-TAB.
   02 EL-MESE OCCURS 12 TIMES.
      03 DES PIC X(9).
      03 TOT PIC 9(10).
```

8.1.3.2. *Caricamento da flusso esterno*

Nel caso in cui la tabella sia costituita da un numero piuttosto alto di elementi e/o sia soggetta a frequenti variazioni di contenuto si ricorre al caricamento da flusso esterno, cioè il caricamento risulta essere una fase vera e propria del programma in cui i dati da inserire in tabella, letti da un flusso, vengono caricati dopo un eventuale controllo: l'operazione di caricamento della tabella fa parte di solito delle operazioni di inizio programma.

Esempi:

a. Supponiamo che al fine di ottenere i risultati previsti da un programma, si debba utilizzare in memoria una tabella tipo quella rappresentata in fig. 8.1: ogni elemento di tabella contiene la matricola ed il nominativo di un dipendente, informazioni che leggiamo da un flusso il cui tracciato record è riportato in fig. 8.2 (i record relativi agli elementi di tabella hanno codice record CSK = "TA").

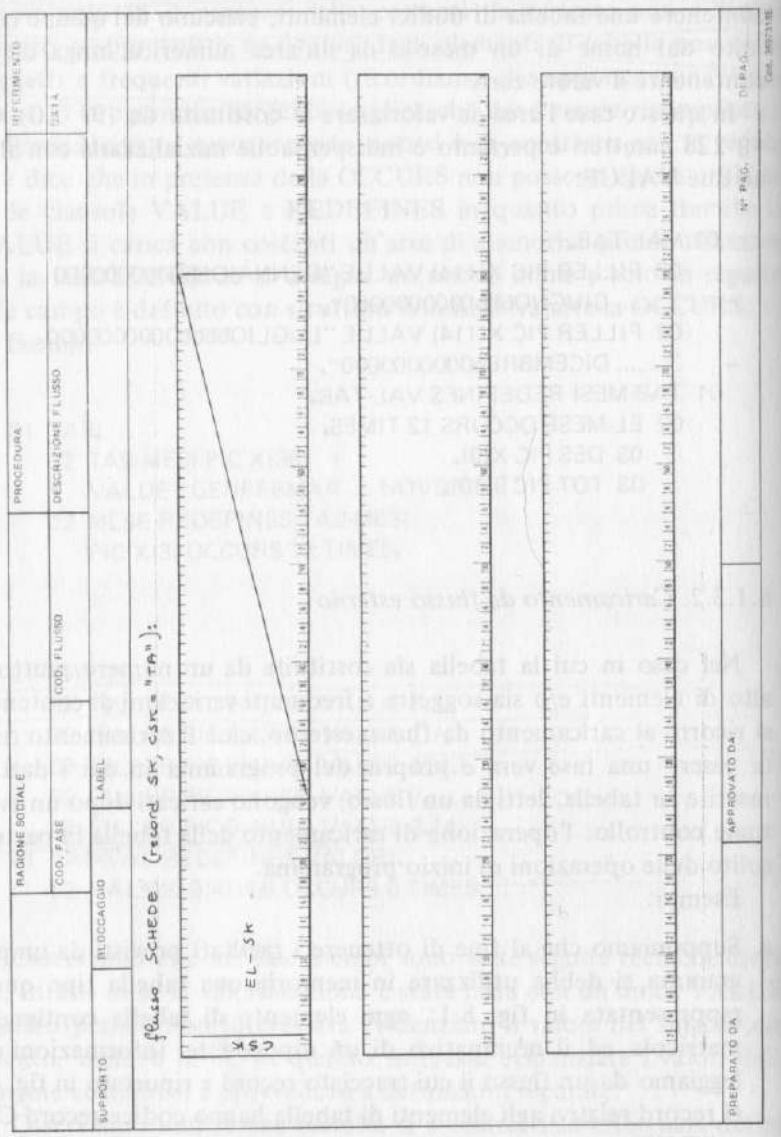


Fig. 8.2

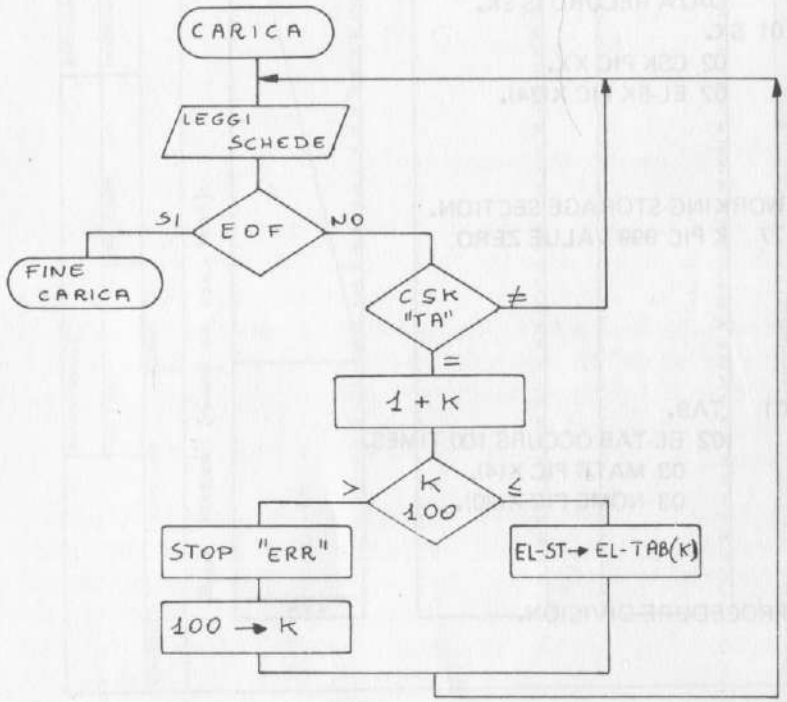
Fig. 8.1

In fig. 8.3 è riportato il flow-chart relativo al caricamento della tabella in questione; questa è la codifica degli elementi necessari per la definizione della tabella ed il caricamento:

```
FILE-CONTROL.
    SELECT SCHEDE ASSIGN TO lettore.
    .
    .
    .
FILE SECTION.
FD  SCHEDE LABEL RECORD IS OMITTED
    DATA RECORD IS SK.
01 SK.
    02 CSK PIC XX.
    02 EL-SK PIC X(24).
    .
    .
    .
WORKING-STORAGE SECTION.
77  K PIC 999 VALUE ZERO.
    .
    .
    .
01  TAB.
    02 EL-TAB OCCURS 100 TIMES.
       03 MATR PIC X(4).
       03 NOME PIC X(20).
    .
    .
    .
PROCEDURE DIVISION.
    .
    .
    .
```

```
CARICA SECTION.
INIZIO.
    READ SCHEDE AT END GO TO FINE.
    IF CSK NOT = "TA" GO TO INIZIO.
    ADD 1 TO K
    IF K > 100 STOP "ERR"
    MOVE 100 TO K ELSE
    MOVE EL-ST TO EL-TAB (K).
    GO TO INIZIO.
FINE. EXIT.
    .
    .
    .
```

Fig. 8.3



Osserviamo che al termine del caricamento l'indice K contiene il numero degli elementi effettivamente caricati.
Ricordiamo che talvolta è necessario effettuare una sequenza di controlli al momento del caricamento, ad esempio: validità codice, numericità di alcuni campi, ordinamento, ecc.

b. La tabella che si vuole caricare in memoria sia l'insieme dei record registrati sequenzialmente sul flusso FL-TAB su disco magnetico. Gli elementi di tabella possano essere al massimo 50 con codice compreso fra 301 e 375: in questo caso è forse conveniente riservare un'area di memoria atta a contenere 75 elementi e procedere al caricamento (e alla ricerca) in accesso diretto (il valore dell'indice è uguale al valore del codice meno 300).
In fig. 8.4 è riportato il tracciato del record che contiene il generico elemento di tabella e in fig. 8.5 il flow-chart di caricamento: si osservi che l'unico controllo che viene effettuato è che non ci siano due elementi con lo stesso codice (non vengono effettuati altri controlli in quanto immaginiamo che siano stati caricati su disco solo record formalmente corretti).

```
FILE-CONTROL.
    SELECT FIL-TAB ASSIGN TO disco.
    .
    .
    .
FILE SECTION.
FD  FL-TAB
    LABEL RECORD IS STANDARD
    BLOCK CONTAINS 12 RECORDS
    DATA RECORD IS REC-TAB.
01  REC-TAB.
    02 COD PIC 9(3).
    02 DES PIC X(25).
    .
    .
    .
WORKING-STORAGE SECTION.
77  IND PIC 99.
    .
    .
    .
```

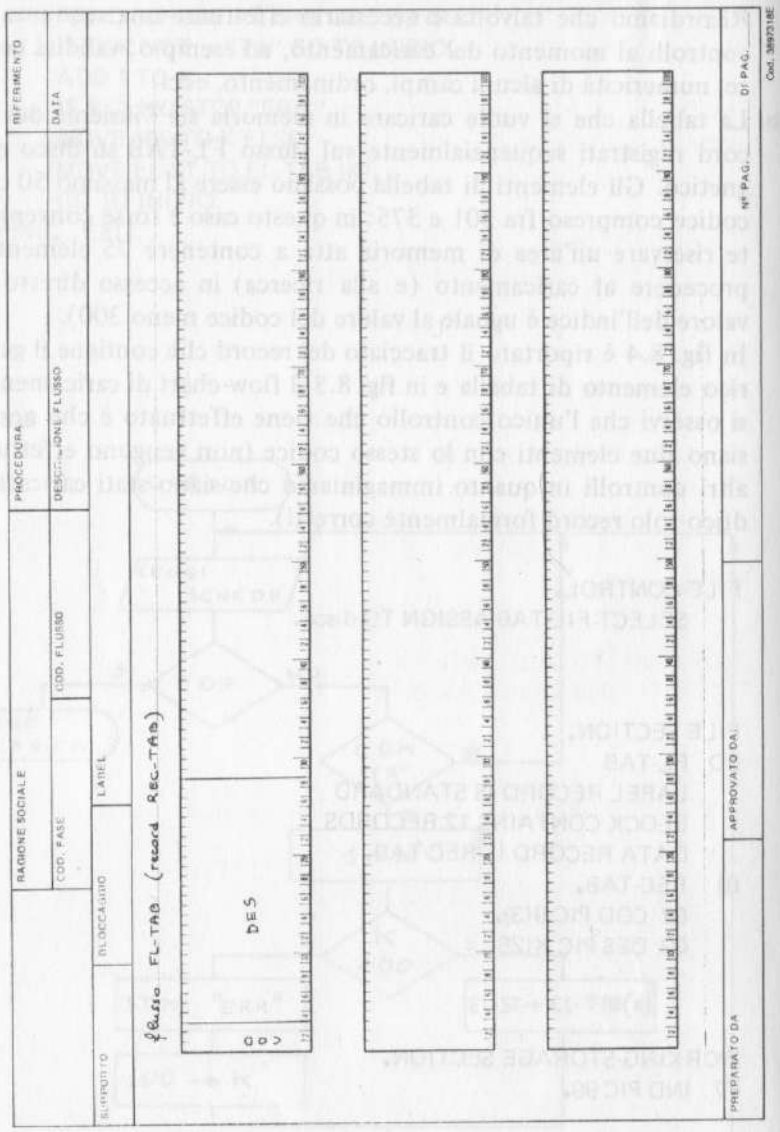
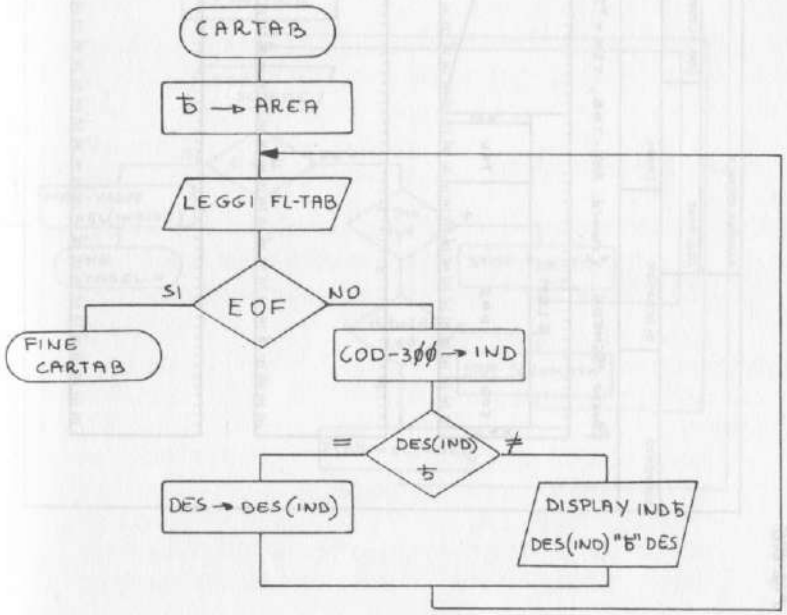


Fig. 8.4


```
01    AREA.
      02 DES PIC X(25) OCCURS 75 TIMES.
      :
      :
PROCEDURE DIVISION.
      :
      :
CARTAB SECTION.
VIA.  MOVE SPACE TO AREA.
LEGGI. READ FL-TAB AT END GO TO EX-CARTAB.
      SUBTRACT 300 FROM COD GIVING IND
      IF DES (IND) = SPACE
      MOVE DES TO DES (IND)
      ELSE
      DISPLAY IND "b" DES (IND) "b" DES.
      GO TO LEGGI.
EX-CARTAB.  EXIT.
```

Fig. 8.5



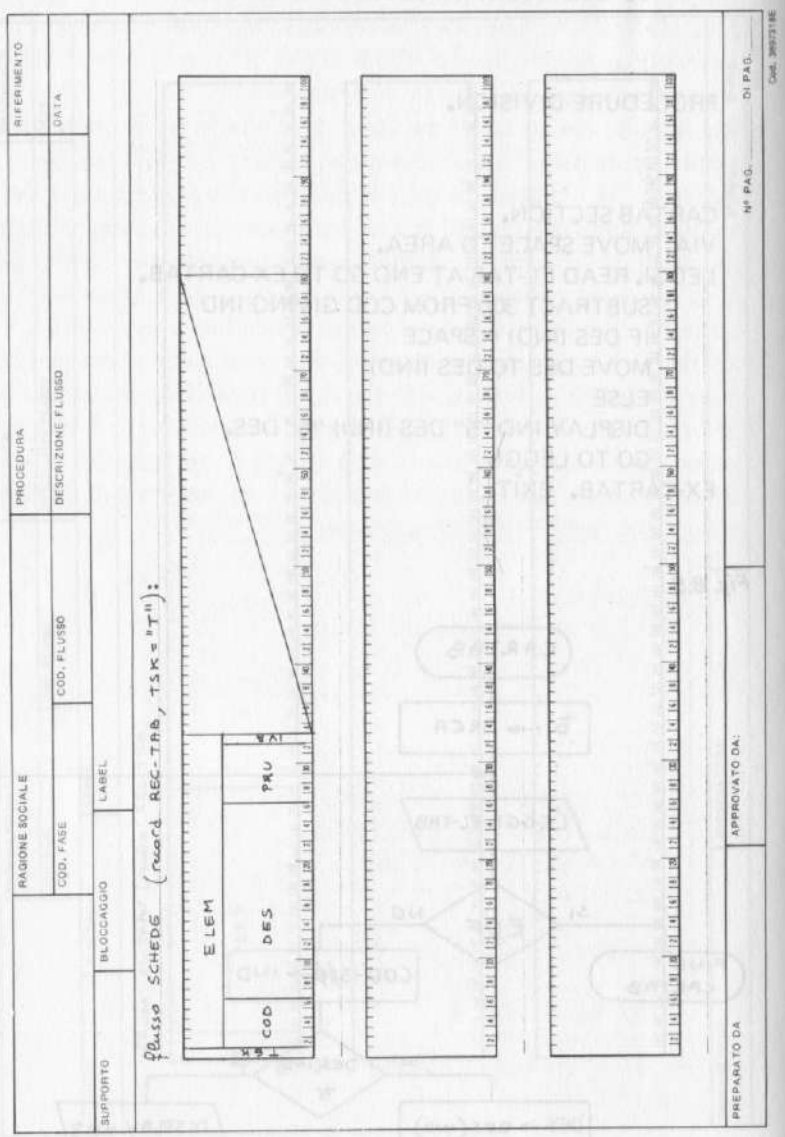
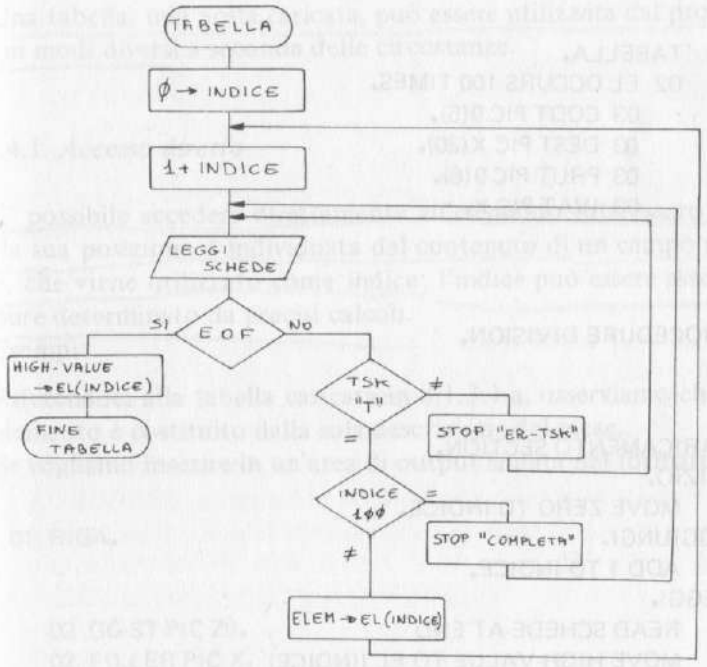


Fig. 8.6

In questo caso l'abblencamento della zone riservata per la tabella è stato fatto all'inizio del caricamento; si poteva anche assegnare il nome AREA come la ridefinizione di un nome-dato di PIC X(1875) con VALUE SPACE.

c. vogliamo caricare in un'area di memoria opportunamente definita una tabella costituita al più da 100 elementi: dopo l'ultimo elemento effettivamente caricato si vuole inserire un elemento fittizio di valore HIGH-VALUE; in fig. 8 6 è rappresentato il tracciato scheda relativo al flusso contenente gli elementi di tabella (gli elementi di tipo scheda TSK ≠ "T" non devono essere caricati in tabella), in fig. 8.7 è riportato il flow-chart di caricamento.

Fig. 8.7



```
FILE-CONTROL.
    SELECT SCHEDE ASSIGN TO lettore-schede.
    .
    .
    .
FILE SECTION.
FD  SCHEDE LABEL RECORD IS OMITTED
    DATA RECORD IS REC-TAB.
01  REC-TAB.
    02 TSK PIC X.
    02 ELEM PIC X(32).
    .
    .
    .
WORKING-STORAGE SECTION.
77  INDICE PIC 999.
    .
    .
    .
01  TABELLA.
    02 EL OCCURS 100 TIMES.
       03 CODT PIC 9(5).
       03 DEST PIC X(20).
       03 PRUT PIC 9(6).
       03 IVAT PIC X.
    .
    .
    .
PROCEDURE DIVISION.
    .
    .
    .
CARICAMENTO SECTION.
INIZIO.
    MOVE ZERO TO INDICE.
AGGIUNGI.
    ADD 1 TO INDICE.
LEGGI.
    READ SCHEDE AT END
    MOVE HIGH-VALUE TO EL (INDICE)
    GO TO EX-TABELLA.
    IF TSK NOT = "T" STOP "ERR-TSK"
    GO TO LEGGI.
```



```
    IF INDICE = 100 STOP "COMPLETA"
    GO TO LEGGI.
    MOVE ELEM TO EL (INDICE)
    GO TO AGGIUNGI.
EX-TABELLA. EXIT.
```

Caricare un elemento di chiave massima è utile non solo nel caso di tabelle ordinate, infatti nel caso di tabelle disordinate l'elemento contenente HIGH-VALUE rappresenta il fine tabella ed è un'informazione alternativa rispetto al numero degli elementi caricati.

8.1.4. *Modalità di utilizzo di una tabella*

Una tabella, una volta caricata, può essere utilizzata dal programma in modi diversi a seconda delle circostanze.

8.1.4.1. *Accesso diretto*

E' possibile accedere direttamente all'elemento interessato quando la sua posizione è individuata dal contenuto di un campo numerico, che viene utilizzato come indice: l'indice può essere assegnato oppure determinato da precisi calcoli.

Esempi:

a. Riferendoci alla tabella caricata in 8.1.3.1.a. osserviamo che ogni elemento è costituito dalla sola descrizione del mese.
Se vogliamo inserire in un'area di output la data nel formato:

```
01 RIGA.
   .
   .
   .
   02 GG-ST PIC Z9.
   02 FILLER PIC X.
   02 MM-ST PIC XXX.
   02 FILLER PIC X.
   02 AA-ST PIC 99.
   .
   .
   .
```

facendo riferimento alla data registrata su un record di input:

```
01 REC-IN.
   .
   .
   .
   02 GG PIC 99.
   02 MM PIC 99.
   02 AA PIC 99.
   .
   .
   .
```

in PROCEDURE DIVISION prepareremo la riga di stampa con questa sequenza di istruzioni:

```
.
.
.
MOVE SPACE TO RIGA
.
.
.
MOVE GG TO GG-ST
MOVE MESE (MM) TO MM-ST
MOVE AA TO AA-ST
.
.
.
```

b. Riferendoci alla tabella caricata in 8.1.3.2.b. osserviamo che l' area di memoria è stata riempita parzialmente (in modo lacunare) e che la corrispondenza tra codice dell'elemento e sua posizione in tabella è diretta, analizziamo le istruzioni di PROCEDURE che ci consentono di prelevare dalla tabella la descrizione dell'elemento di codice CH-IN:


```
.
.
.
SUBTRACT 300 FROM CH-IN GIVING DEP
IF DES (DEP) = SPACE
DISPLAY CH-IN "NON TROVATO"
ELSE
DISPLAY CH-IN "b" DES (DEP).
.
.
.
```

8.1.4.2. *Ricerca tabellare*

Quando non è nota la posizione dell'elemento interessato è necessario effettuare una vera e propria ricerca tabellare.

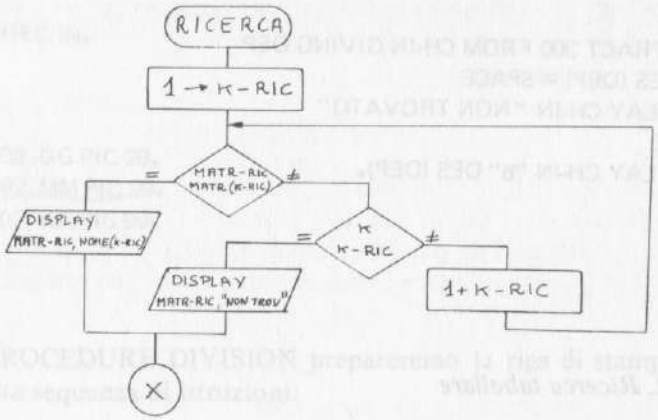
Le modalità con cui si sviluppa la ricerca dipendono essenzialmente dal fatto che la tabella sia ordinata o disordinata, infatti se la tabella è ordinata la ricerca si spinge al massimo fino all'elemento di codice maggiore di quello cercato, mentre, per tabella disordinata la ricerca di un elemento si deve spingere alla fine della tabella.

Esempi:

a. Riferendoci alla tabella caricata in 8.1.3.2.a. e supponendola disordinata, dovendo ricercare l'elemento di codice MATR-RIC procediamo come indicato in fig. 8.8 (ricordiamo che K alla fine del caricamento indica il numero di elementi effettivamente caricati).

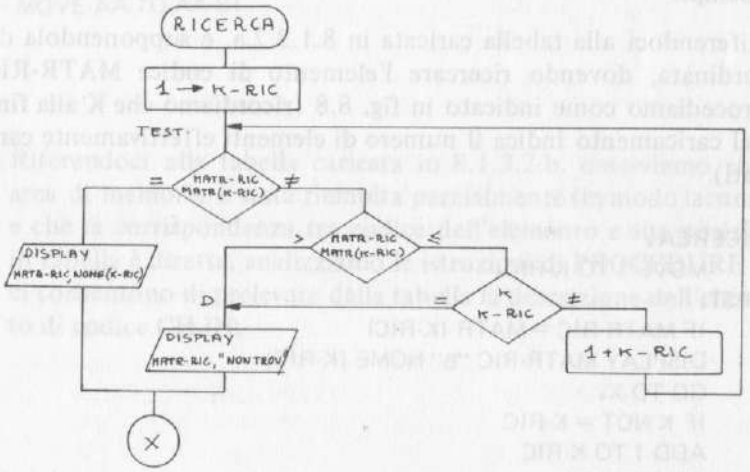
```
RICERCA.
    MOVE 1 TO K-RIC.
TEST.
    IF MATR-RIC = MATR (K-RIC)
    DISPLAY MATR-RIC "b" NOME (K-RIC)
    GO TO X.
    IF K NOT = K-RIC
    ADD 1 TO K-RIC
    GO TO TEST.
    DISPLAY MATR-RIC "NON TROVATO".
X. .........
```

Fig. 8.8



b. Riferendoci alla tabella caricata in 8.1.3.2.a., e supponendola ordinata per matricola crescente, per ricercare l'elemento con matricola MATR-RIC procediamo come indicato in fig. 8.9.

Fig. 8.9




```
RICERCA.
    MOVE 1 TO K-RIC.

TEST.
    IF MATR-RIC = MATR (K-RIC)
    DISPLAY MATR-RIC "b" NOME (RIC)
    GO TO X.
    IF MATR-RIC > MATR (K-RIC)
    GO TO D.
    IF K NOT = K-RIC
    ADD 1 TO K-RIC
    GO TO TEST.
D.  DISPLAY MATR-RIC "NON TROVATO".
X.  .......
```

c. Riferendoci alla tabella caricata in 8.1.2.3.2.c e supponendola disordinata, dovendo ricercare un elemento di codice COD procediamo come indicato in fig. 8.10 (il ruolo che assume l'elemento di valore HIGH-VALUE è lo stesso di K in a.)

```
RICERCA.
    MOVE 1 TO INDICE.
TEST.
    IF COD = COD-T (INDICE)
    DISPLAY EL (INDICE)
    GO TO X.
    IF COD-T (INDICE) NOT = HIGH-VALUE
    ADD 1 TO INDICE
    GO TO TEST.
    DISPLAY COD "b NON TROVATO".
X.  .........
```

d. Riferendoci alla tabella caricata in 8.1.3.2.c e supponendola ordinata, dovendo ricercare un elemento di codice COD procediamo come in fig. 8.11 (osserviamo che la presenza dell'elemento di valore HIGH-VALUE provoca automaticamente l'uscita di "NON TROVATO" in caso di fine tabella).

Fig. 8.10

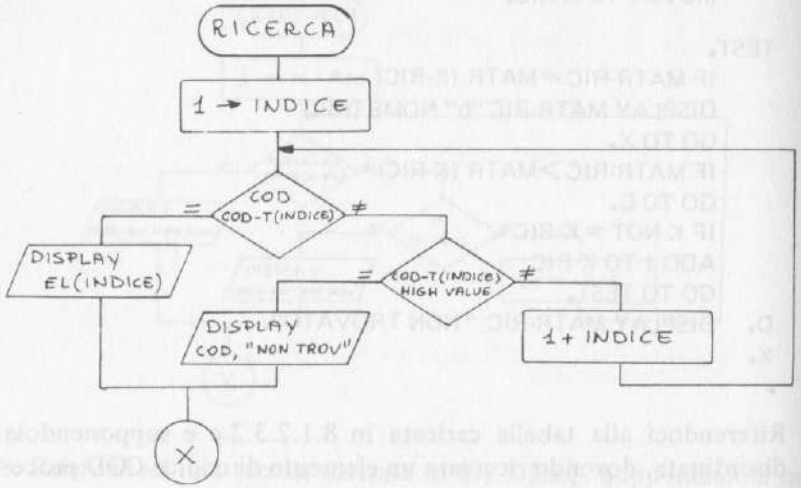
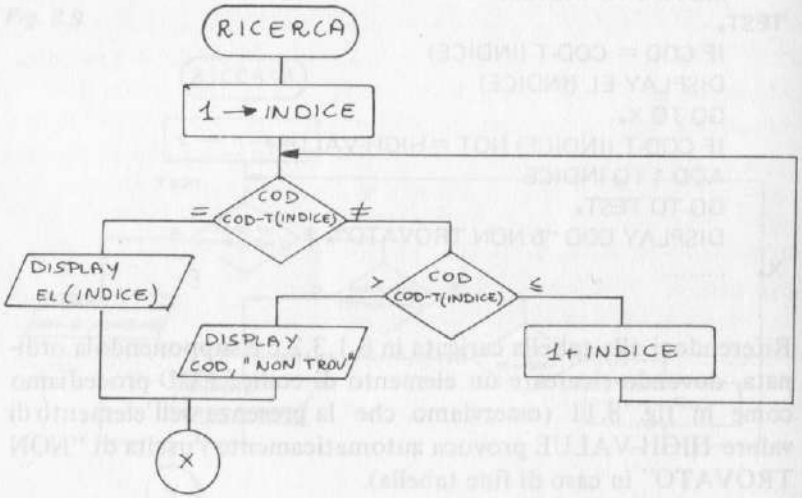


Fig. 8.11




```
RICERCA.
    MOVE 1 TO INDICE.
TEST.
    IF COD = COD-T (INDICE)
    DISPLAY EL (INDICE)
    GO TO X.
    IF COD NOT > COD-T (INDICE)
    ADD 1 TO INDICE
    GO TO TEST.
    DISPLAY COD "b NON TROVATO".
X.  .........
```

8.2. Tabelle gestite dal sistema

Il metodo indexing facilita alcune operazioni come la ricerca tabellare ed il trattamento di campi specifici.

8.2.1. *Definizione delle tabelle*

Il metodo indexing richiede, rispetto al metodo subscripting, un formato della OCCURS leggermente modificato, infatti occorre assegnare uno o più nomi-indice che saranno utilizzati nella gestione della tabella.

Esaminiamo un semplice formato:

numero-di-livello nome-dato OCCURS intero TIMES
INDEXED BY nome-indice$_1$ [nome-indice$_2$] ....

Il nome-indice non richiede alcuna descrizione separata, la sua definizione è a carico del compilatore (la sua allocazione in memoria ed il suo formato sono dipendenti dall'hardware).

Durante l'esecuzione del programma, all'atto del riferimento ad una posizione di tabella, il contenuto del campo nome-indice consente di individuare un elemento di tabella.

I nomi-indice possono essere gestiti da un numero ristretto di istruzioni Cobol e precisamente:

a. SET, SEARCH, PERFORM...VARYING... di cui si parlerà più avanti;
b. IF per confronti di relazione.

8.2.2. *Riferimenti a campi di tabella*

Per riferirsi ad un campo di tabella valgono le stesse considerazioni fatte a proposito del metodo subscripting: in questo caso la particolarizzazione del nome del generico campo deve essere fatta o tramite una costante numerica oppure un nome-indice associato alla tabella nella OCCURS.

Esempio:

Riferendosi alla tabella "indexata":

```
01 TAB.
   02 CAMPO OCCURS 50 TIMES INDEXED BY IND.
      03 CODTAB PIC X(4).
      03 DESTAB PIC X(25).
```

se si vuole indicare la ventesima DESTAB si scrive:

```
DESTAB (20)
```

oppure:

```
DESTAB (IND)
```

dove IND è stato preventivamente caricato in modo da puntare al ventesimo elemento. Ricordiamo che i nomi-indice prima di essere utilizzati nei riferimenti e campi di tabella devono essere inizializzati (normalmente tramite il verbo SET e la PERFORM... VARYING).

Anche per tabelle da trattare con il metodo indexing, quando il programmatore vuole esplicitamente fare riferimento ad un elemento di tabella, deve preoccuparsi che il contenuto del nome-indice (o la costante) non assuma valori minori di uno o maggiori del massimo degli elementi previsti per la tabella (ovvero l'"intero" della OCCURS).

E' sempre possibile utilizzare dei nomi-dato per fare riferimento a campi di tabella "indexate" senza che ciò alteri i nomi-indice ad esse associati.

Un campo di tabella può essere puntato con nome-indice solo se il nome-indice è stato associato alla tabella con la clausola INDEXED BY.

8.2.3. *Indirizzamento relativo*

E' possibile, utilizzando il metodo indexing, utilizzare l'indirizzamento relativo rispettando il seguente formato:

nome-dato (nome-indice [{+ / -} intero])

questo consente di puntare ad un elemento diverso da quello individuato dal nome-indice lasciando inalterato il contenuto di nome-indice.

8.2.4. *USAGE IS INDEX*

Si è detto che i nomi-indice non possono essere descritti come dati in un programma Cobol e che possono essere trattati solo dalle istruzioni SET, SEARCH, PERFORM... VARYING... ed IF e che il loro formato dipende dall'hardware dell'elaboratore.

La clausola

USAGE IS INDEX

permette di definire un campo-dato in DATA DIVISION con lo stesso formato e le stesse caratteristiche dei nomi-indice.

Un campo di questo tipo (che possiamo chiamare nome-dato-formato-indice) contiene un valore che corrisponde ad un numero di posizione di un elemento di tabella: ricordiamo però che non può essere associato ad un nome-dato di un elemento di tabella a cui possono riferirsi solo i nomi-indice precisati al momento della definizione.

Un nome-dato-formato-indice ha una struttura che non dipende da scelte del programmatore e quindi deve essere omessa la PICTURE.

I nomi-dato-formato-indice possono essere trattati solo tramite SET, SEARCH ed IF e nella loro definizione oltre alla PICTURE non sono ammesse le clausole VALUE, JUSTIFIED e BLANK WHEN ZERO.

8.2.5. *Caricamento delle tabelle*

Anche per il caricamento valgono, per le tabelle trattate con il metodo indexing, gli stessi principi esposti per il metodo subscripting.

Sono al solito disponibili, a seconda delle esigenze, i due tipi di caricamento:

a. caricamento con VALUE,
b. caricamento da flusso esterno: ricordiamo che i contenuti dei nomi-indice possono essere modificati solo tramite SET e PERFORM e non tramite ADD, MOVE, ecc....

8.2.6. *Il verbo SET*

Il verbo SET è utilizzato per inizializzare i nomi-indice, i nomi-dato-formato-indice ed i campi-dati (in riferimento alla gestione delle tabelle con metodo indexing).

I formati di questo verbo consentono numerose operazioni, esaminiamo il primo formato:

$$\underline{\text{SET}} \left\{ \begin{matrix} \text{nome-indice}_1 \\ \text{identificatore}_1 \end{matrix} \right. \left. \begin{matrix} [\text{nome-indice}_2] \ldots \\ [\text{identificatore}_2] \ldots \end{matrix} \right\} \underline{\text{TO}} \left\{ \begin{matrix} \text{nome-indice}_3 \\ \text{identificatore}_3 \\ \text{costante} \end{matrix} \right\}$$

in questo formato identificatore indica sia nome-dato sia nome-dato-formato-indice.

Il nome-indice$_1$ (nome-indice$_2$, ..) assume il valore corrispondente al numero di posizione espresso da nome-indice$_3$ o da identificatore$_3$ oppure dalla costante.

Se l'identificatore$_1$ (identificatore$_2$ ...) è un nome-dato-formato indice il comportamento è lo stesso, se invece è un nome-dato il formato è corretto solo se il campo emittente è un nome-indice.

Osserviamo che questo formato del verbo SET serve per effettuare delle operazioni di trasferimento ma non impatta con le funzioni della MOVE.

Analizziamo ora il secondo formato del verbo SET:

$$\underline{\text{SET}}\ \text{nome-indice}_1\ [\text{nome-indice}_2] \ldots \left\{ \begin{matrix} \underline{\text{UP BY}} \\ \underline{\text{DOWN BY}} \end{matrix} \right\} \left\{ \begin{matrix} \text{nome-dato} \\ \text{costante} \end{matrix} \right\}$$

Questo formato del verbo SET consente di incrementare (UP BY) o decrementare (DOWN BY) i nomi-indice dichiarati a primo membro, con i valori della costante o di nome-dato.

Esempi:

a. SET X1 TO 4

in questo caso X1, che deve essere un nome-indice, viene posizionato a 4.

b. SET X1 TO X2

in questo caso X1 assume il valore espresso da X2. Osserviamo che vi sono delle limitazioni relative ai formati di X1 e X2:

- se X1 è un nome-indice X2 può essere un nome-indice, un nome-dato-formato-indice oppure un nome-dato,
- se X1 è un nome-dato-formato-indice X2 può essere un nome-indice oppure un nome-dato-formato-indice,
- se X1 è un nome-dato X2 può essere solo un nome-indice.

c. SET INDICE UP BY 3

il valore del campo nome-indice INDICE è incrementato di 3.

d. SET CAMPO DOWN BY VAL

il valore del campo nome-indice CAMPO è decremenato del contenuto del campo nome-dato di nome VAL.

8.2.7. *Il verbo SEARCH*

Il metodo indexing permette, come abbiamo detto, di agevolare le operazioni di ricerca in tabella: questo avviene utilizzando il verbo SEARCH che opera una vera e propria ricerca automatica su una dimensione di una tabella "indexata".

Il formato del verbo SEARCH è:

SEARCH nome-dato [VARYING {nome-indice / identificatore}]

[AT END istruzioni$_1$]

WHEN condizione$_1$ {istruzioni$_2$ / NEXT SENTENCE}

[WHEN condizione$_2$ {istruzioni$_3$ / NEXT SENTENCE}] ...

Quando una delle condizioni espresse dopo la parola WHEN è verificata la ricerca si arresta e il nome-indice associato alla tabella resta fissato con il valore della posizione dell'elemento su cui è terminata la ricerca. Nome-dato non deve avere l'indicazione dell'indice ma deve essere necessariamente descritto con la clausola OCCURS ... INDEXED BY.

La SEARCH, nel formato riportato, opera una ricerca seriale sulla dimensione interessata facendo variare un solo indice.

Se alla partenza della SEARCH il nome-indice associato al nome-dato punta ad un numero di posizione maggiore dell'ultimo elemento di tabella, la ricerca è immediatamente terminata, in questo caso se è indicata l'opzione AT END viene pasato il controllo ad istruzioni$_1$, altrimenti il controllo passa alla frase immediatamente successiva.

Se alla partenza della SEARCH il nome-indice associato al nome-dato si riferisce ad un elemento di tabella, vengono testate, nell'ordine in cui sono elencate, le condizioni specificate: se nessuna di esse è verificata, il nome-indice è automaticamente incrementato in modo da puntare all'elemento seguente, quindi vengono nuovamente testate le condizioni, e così via. Tale processo ha termine quando una delle condizioni è verificata (in tal caso il controllo passa alle istruzioni specificate dopo la condizione) oppure per il rilevamento del fine tabella.

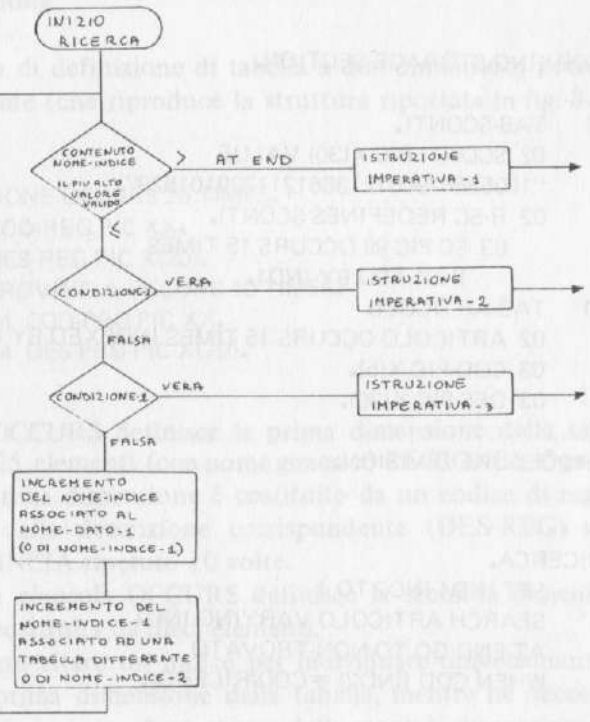
Utilizzando l'opzione VARYING con identificatore, che deve essere un nome-dato-formato-indice oppure un nome-dato numerico intero, l'identificatore è aggiornato parallelamente al nome-indice associato a nome-dato.

Se l'opzione VARYING è riferita a nome-indice e nome-indice è stato elencato insieme ad altri nomi nella clausola INDEXED BY associata all'elemento oggetto della SEARCH, esso costituisce l'indice di controllo della ricerca, cioè sarà il solo ad essere variato durante l'esecuzione della SEARCH; al di fuori di questo caso la SEARCH utilizza automaticamente il primo nome-indice associato alla tabella.

Se l'opzione VARYING è riferita ad un nome-indice associato ad un'altra tabella, anche tale nome-indice sarà automaticamente incrementato durante la ricerca: questo consente di pilotare, nell'ambito di una sola ricerca il posizionamento di indici di tabelle interdipendenti.

In fig. 8.12 è riportato il flow-chart delle operazioni eseguite da una SEARCH con un formato comprensivo di due condizioni.

Fig. 8.12



8.2.8. *Esempi*

a.
```
WORKING-STORAGE SECTION.
.........
01   TABELLA.
     02 ARTICOLO OCCURS 20 TIMES INDEXED BY IND.
          03 COD PIC X(5).
          03 DES PIC X(20).
.........
PROCEDURE DIVISION.
.........
.........
RICERCA.
     SET IND TO 1
     SEARCH ARTICOLO AT END GO TO NON-TROV
     WHEN COD (IND) = COD-RICERCA GO TO TROV.
.........
.........
```

b.
```
WORKING-STORAGE SECTION.
.........
01   TAB-SCONTI.
     02 SCONTI PIC X(30) VALUE
     "100535479007213861211709101529".
     02 R-SC REDEFINES SCONTI.
        03 SC PIC 99 OCCURS 15 TIMES
              INDEXED BY IND1.
01   TAB-ARTICOLI.
     02 ARTICOLO OCCURS 15 TIMES INDEXED BY IND2.
     03 COD PIC X(5).
     03 DES PIC X(30).
.........
PROCEDURE DIVISION.
.........
.........
RICERCA.
     SET IND1 IND2 TO 1
     SEARCH ARTICOLO VARYING IND1
     AT END GO TO NON-TROVATO
     WHEN COD (IND2) = COD-RICERCA
```



```
     MOVE COD (IND2) TO COD-ST
     MOVE DES (IND2) TO DES-ST
     MOVE SC (IND1) TO SC-ST
     WRITE RIGA BEFORE 2 MOVE SPACE TO RIGA
     GO TO LETTURA.
.........
```

8.3. Tabelle multidimensionali

Le tabelle di cui si è finora trattato sono ad una sola dimensione e sono descritte tramite una sola clausola OCCURS.

In Cobol è possibile gestire anche tabelle a più dimensioni, definite tramite più clausole OCCURS riportate a livelli via via crescenti: analizziamo brevemente le caratteristiche su un esempio.

8.3.1. *Subscripting*

Un esempio di definizione di tabella a due dimensioni potrebbe essere il seguente (che riproduce la struttura riportata in fig. 8.13):

```
01 STATO.
   02 REGIONE OCCURS 25 TIMES.
      03 COD-REG PIC XX.
      03 DES-REG PIC X(20).
      03 PROVINCIA OCCURS 10 TIMES.
         04 COD-PRO PIC XX.
         04 DES-PRO PIC X(20).
```

La prima OCCURS definisce la prima dimensione della tabella costituita da 25 elementi (con nome generico REGIONE). Ogni elemento della prima dimensione è costituito da un codice di regione (COD-REG), dalla descrizione corrispondente (DES-REG) e dal campo PROVINCIA ripetuto 10 volte.

La seconda clausola OCCURS definisce la seconda dimensione della tabella, costituita da dieci elementi.

E' sufficiente citare un indice per individuare univocamente un campo della prima dimensione della tabella, mentre ne occorrono due per fare riferimento ad un campo della seconda dimensione.

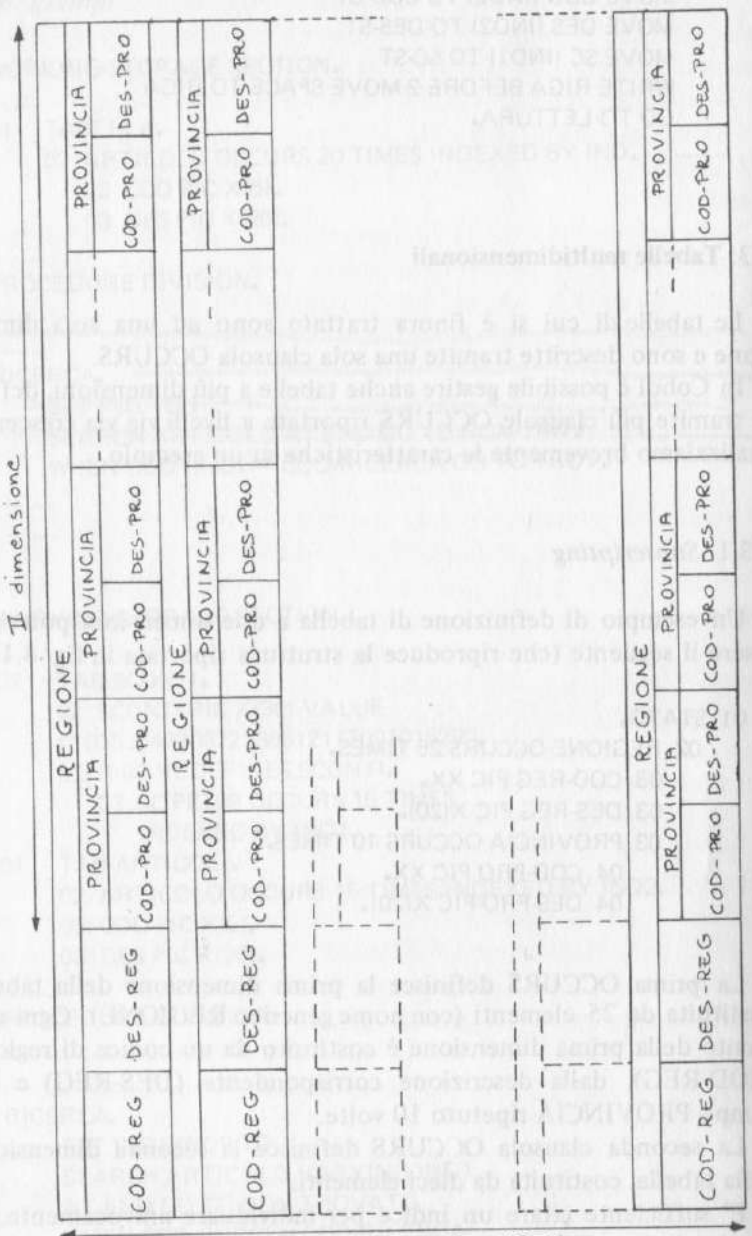


Fig. 8.13

Come indici possono essere utilizzati nomi dato oppure costanti, quando bisogna fare riferimento anche al secondo indice, i due indici si scrivono fra parentesi rotonde e separati da uno spazio; prima si scrive l'indice della dimensione superiore e quindi quello della seconda dimensione, ecc...

Esempi:

```
COD-REG (IND1)
COD-PRO (5 VAL)
DES-PRO (IND2 6)
DES-PRO (IND1 IND2)
```

8.3.2. *Indexing*

Quanto esposto per le tabelle trattate con il metodo subscripting è estendibile con le opportune variazioni, alle tabelle trattate con il metodo indexing.

Esempio:

```
01 STATO.
   02 REGIONE OCCURS 25 TIMES
      INDEXED BY NUM1.
      03 COD-REG PIC XX.
      03 DES-REG PIC X(20).
      03 PROVINCIA OCCURS 10 TIMES
         INDEXED BY NUM2.
         04 COD-PRO PIC XX.
         04 DES-PRO PIC X(20).
```

NUM1 è il nome-indice associato alla prima dimensione della tabella, NUM2 alla seconda dimensione.

Nei riferimenti ai campi della seconda dimensione possono essere utilizzate sia costanti che nomi-indice, mai però nomi-dato e nomi-indice insieme.

Ricordiamo che per ricerca con il metodo indexing si utilizza il verbo SEARCH che, come è già stato detto, lavora su una sola dimensione per volta.

8.4. PERFORM... VARYING...

Oltre ai formati già analizzati per il verbo PERFORM ne esiste uno che consente di variare automaticamente un nome-dato o un nome-indice (durante l'esecuzione della PERFORM, fino al verificarsi di una certa condizione.

Questo è un formato di codifica:

$$\underline{\text{PERFORM}}\ \text{nome-procedura}_1 \left[\left\{ \begin{matrix} \underline{\text{THROUGH}} \\ \underline{\text{THRU}} \end{matrix} \right\} \text{nome-procedura}_2 \right]$$

$$\underline{\text{VARYING}} \left\{ \begin{matrix} \text{nome-dato}_1 \\ \text{nome-indice}_1 \end{matrix} \right\} \underline{\text{FROM}} \left\{ \begin{matrix} \text{nome-dato}_2 \\ \text{nome-indice}_2 \\ \text{costante}_1 \end{matrix} \right\}$$

$$\underline{\text{BY}} \left\{ \begin{matrix} \text{nome-dato}_3 \\ \text{costante}_2 \end{matrix} \right\} \underline{\text{UNTIL}}\ \text{condizione.}$$

Il nome-dato$_1$ o il nome-indice$_1$ sono settati inizialmente ad un certo valore (FROM) e sono variati, dopo ogni esecuzione delle istruzioni richiamate, con un certo passo (BY).

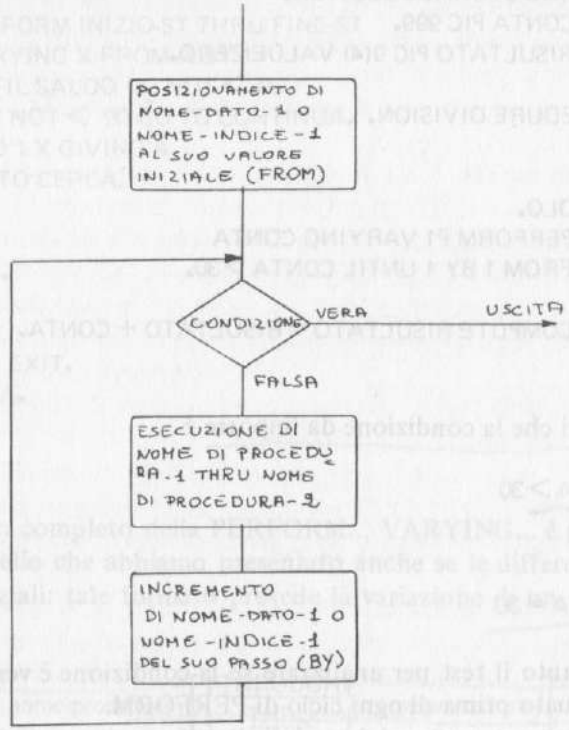
Ricordiamo che:

a. tutti i nomi-dato citati sono campi numerici elementari definiti in DATA DIVISION e le costanti sono numeriche;
b. se VARYING fa riferimento ad un nome-indice, si ha che nome-dato$_2$ e nome-dato$_3$ devono essere interi ed inoltre le costanti devono essere intere positive;
c. se la parola FROM fa riferimento ad un nome-indice, nome-dato$_1$, nome-dato$_2$ e costante$_2$ devono essere interi;
d. nome-dato$_3$ non può assumere il valore zero;
e. la condizione può essere semplice o composta. In fig. 8.14 è rappresentato il flow-chart relativo al funzionamento della PERFORM nel formato proposto.

Esempi:

```
PERFORM PRX VARYING CONTATORE
    FROM 100 BY -10 UNTIL CONTATORE = -30.
PERFORM STA VARYING CONT1 FROM -100
    BY -20 UNTIL ALFA = -200.
PERFORM CT THRU EX-CT VARYING INDICE
    FROM A BY PASSO UNTIL INDICE <C.
```


Fig. 8.14



E' responsabilità del programmatore assicurarsi, per evitare di ciclare indefinitamente, che la condizione imposta possa verificarsi effettivamente ed evitare quindi di scrivere istruzioni così:

```
PERFORM CALC VARYING A FROM 1 BY 2
    UNTIL  A = 30.
```

Esempi:

a. Volendo ottenere nel campo RISULTATO la somma dei primi 30 numeri interi, possiamo scrivere:

```
.........
WORKING-STORAGE SECTION.
77    CONTA PIC 999.
77    RISULTATO PIC 9(4) VALUE ZERO.
.........
PROCEDURE DIVISION.
.........
.........
CALCOLO.
      PERFORM P1 VARYING CONTA
      FROM 1 BY 1 UNTIL CONTA >30.
.........
P1.   COMPUTE RISULTATO = RISULTATO + CONTA.
.........
```

NB.

Si noti che la condizione da imporre è

CONTA > 30

e non

CONTA = 30

in quanto il test per analizzare se la condizione è verificata viene effettuato prima di ogni ciclo di PERFORM.

b. Vogliamo stampare dei moduli di sollecito per tutti i clienti della tabella TAB-SALDI che hanno un saldo negativo. E' stato caricato, come ultimo elemento di tabella un cliente fittizio con saldo zero.

```
.........
WORKING-STORAGE SECTION.
77    X PIC 99.
77    A PIC 99 VALUE 1.
.........
01 TAB-SALDI.
      02 CLIENTE OCCURS 91.
            03 COD PIC X(5).
            03 SALDO PIC S9(7).
.........
PROCEDURE DIVISION.
.........
```



```
.........
CERCA.
      PERFORM INIZIO-ST THRU FINE-ST
      VARYING X FROM A BY 1
      UNTIL SALDO (X) NOT < 0.
      IF X NOT < 90 GO TO CONTINUA.
      ADD 1 X GIVING A
      GO TO CERCA.
.........
.........
INIZIO-ST.
.........
.........
FINE-ST.  EXIT.
CONTINUA.
.........
.........
```

Il formato completo della PERFORM... VARYING... è più complesso di quello che abbiamo presentato anche se le differenze non sono sostanziali: tale formato prevede la variazione di un massimo di tre indici:

$$\underline{\text{PERFORM}}\ \text{nome-procedura}_1 \left[\left\{ \begin{matrix} \underline{\text{THROUGH}} \\ \underline{\text{THRU}} \end{matrix} \right\} \text{nome-procedura}_2 \right]$$

$$\underline{\text{VARYING}} \left\{ \begin{matrix} \text{nome-dato}_1 \\ \text{nome-indice}_2 \end{matrix} \right\} \underline{\text{FROM}} \left\{ \begin{matrix} \text{nome-dato}_2 \\ \text{nome-indice}_2 \\ \text{costante}_1 \end{matrix} \right\}$$

$$\underline{\text{BY}} \left\{ \begin{matrix} \text{nome-dato}_3 \\ \text{costante}_2 \end{matrix} \right\} \underline{\text{UNTIL}}\ \text{condizione}_1$$

$$\left[\underline{\text{AFTER}} \left\{ \begin{matrix} \text{nome-dato}_4 \\ \text{nome-indice}_3 \end{matrix} \right\} \underline{\text{FROM}} \left\{ \begin{matrix} \text{nome-dato}_5 \\ \text{nome-indice}_4 \\ \text{costante}_3 \end{matrix} \right\} \right.$$

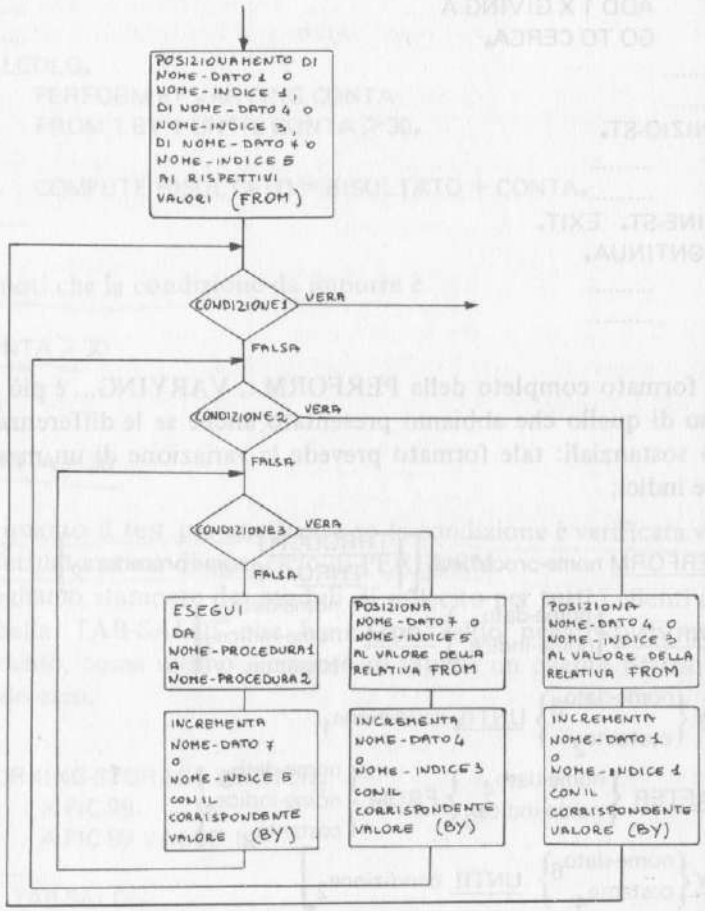
$$\left. \underline{\text{BY}} \left\{ \begin{matrix} \text{nome-dato}_6 \\ \text{costante}_4 \end{matrix} \right\} \underline{\text{UNTIL}}\ \text{condizione}_2 \right]$$

$$\left[\underline{\text{AFTER}} \left\{ \begin{matrix} \text{nome-dato}_7 \\ \text{nome-indice}_5 \end{matrix} \right\} \underline{\text{FROM}} \left\{ \begin{matrix} \text{nome-dato}_8 \\ \text{nome-indice}_6 \\ \text{costante}_5 \end{matrix} \right\} \right.$$

$$\left. \underline{\text{BY}} \left\{ \begin{matrix} \text{nome-dato}_9 \\ \text{costante}_6 \end{matrix} \right\} \underline{\text{UNTIL}}\ \text{condizione}_3 \right]$$

In fig. 8.15 è rappresentato il flow-chart relativo al funzionamento della PERFORM... VARYING in caso di formato completo.

Fig. 8.15



8.5. Un esempio

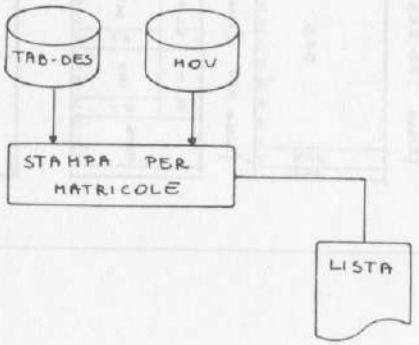
Vogliamo stampare dei moduli a partire da un flusso sequenziale su disco (MOV) ordinato su codice matricola (MATR); in fig. 8.16 è presentato l'organigramma della fase, nelle figg. 8.17 e 8.18 i tracciati dei flussi.

Il flusso MOV contiene uno o più record (REC-M) per matricola, ogni record può contenere, oltre al tipo record (TR) e alla matricola (MATR), fino ad un massimo di 10 elementi costituiti da un riferimento voce (RV) e da un importo (IMP), il primo elemento da non considerare, nel caso in cui su un record siano meno di 10, è quello con RV = "0"; per una stessa matricola possono esserci più record non completamente utilizzati.

Il flusso TAB-DES consiste di una serie di record (REC-T) di cui quelli con TST = "V" sono necessari per la costituzione di una tabella di conversione riferimento-voce/descrizione-voce: si è sicuri che il numero massimo di elementi con TST = "V" è 60, i codici degli elementi utili di tabella siano tutti distinti e assumano valori compresi fra 1 e 99.

Il flusso LISTA è costituito da una serie di moduli (uno per matricola) come da tracciato. La descrizione della voce deve essere prelevata dalla tabella, se in tabella non è presente stampare al suo posto dei punti. La riga contenente il totale importo deve essere stampata a rottura di controllo su matricola. Per semplicità suppo-

Fig. 8.16



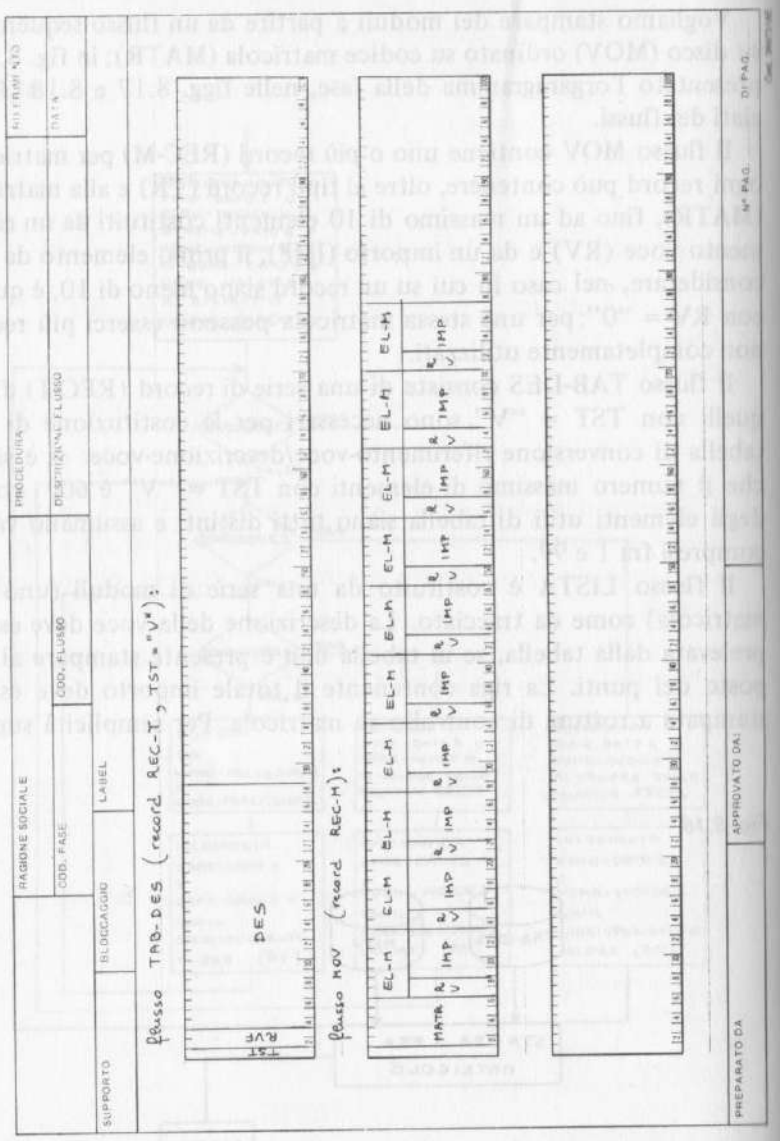


Fig. 8.17

niamo che il numero di righe dettaglio per matricola sia inferiore al numero massimo di righe dettaglio che possono essere scritte nel modulo.

In fig. 8.19 è rappresentato il flow-chart del problema proposto e in fig. 8.20 la relativa codifica.

Fig. 8.18

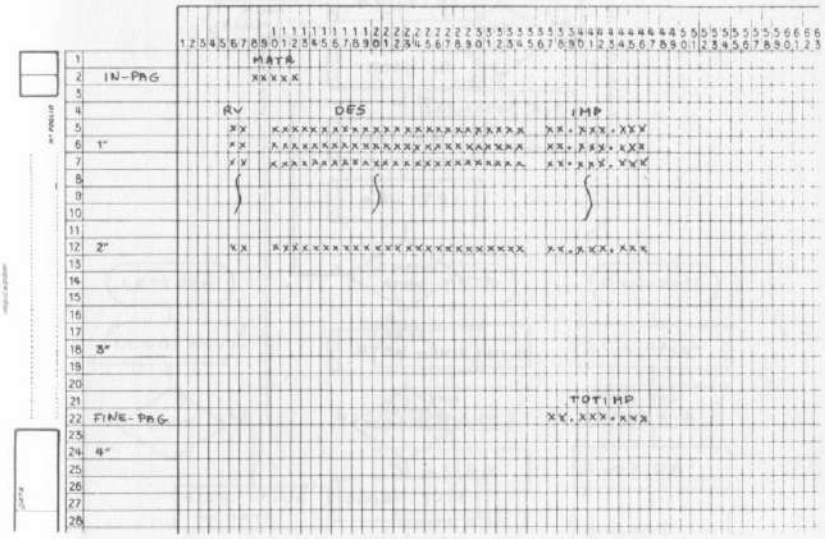


Fig. 8.19a

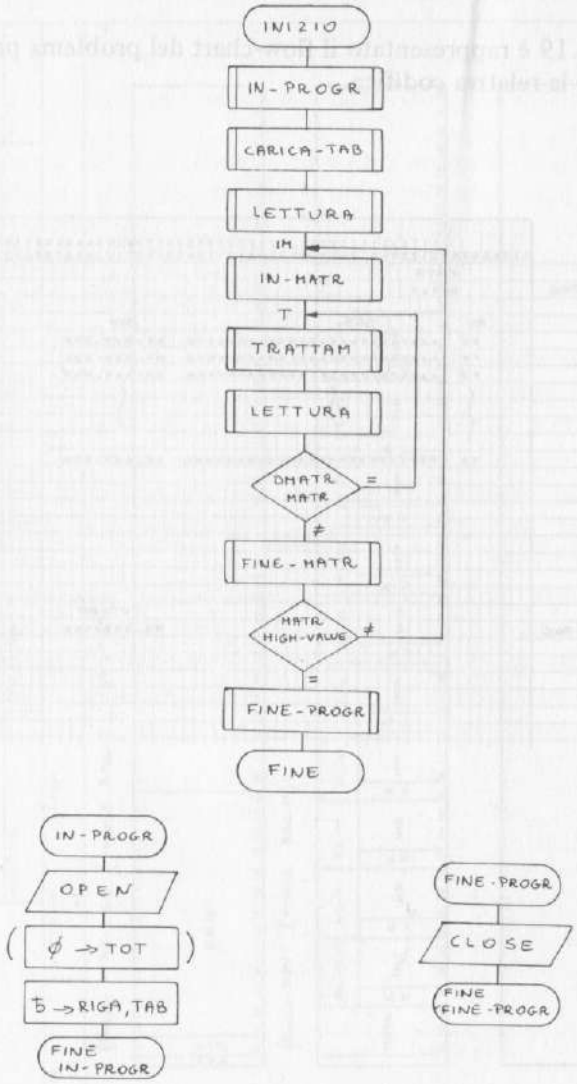


Fig. 8.19b

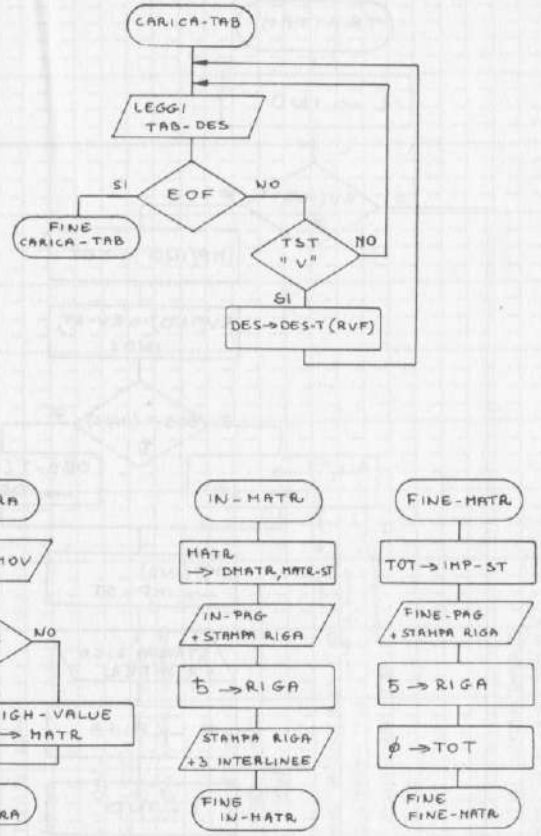


Fig. 8.19c

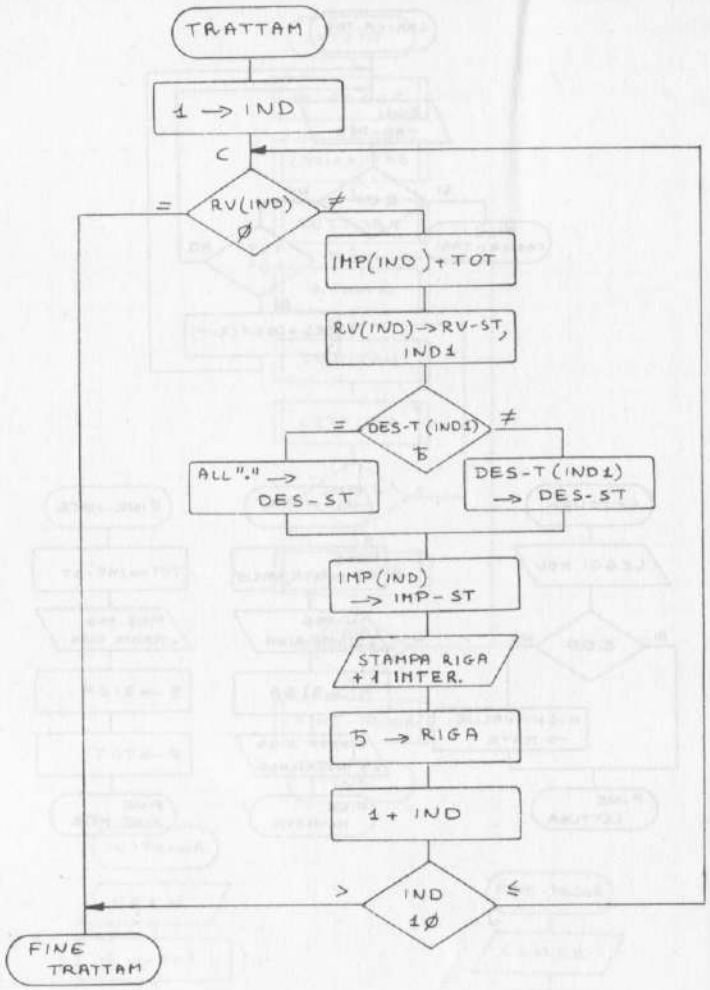


Fig. 8.20a

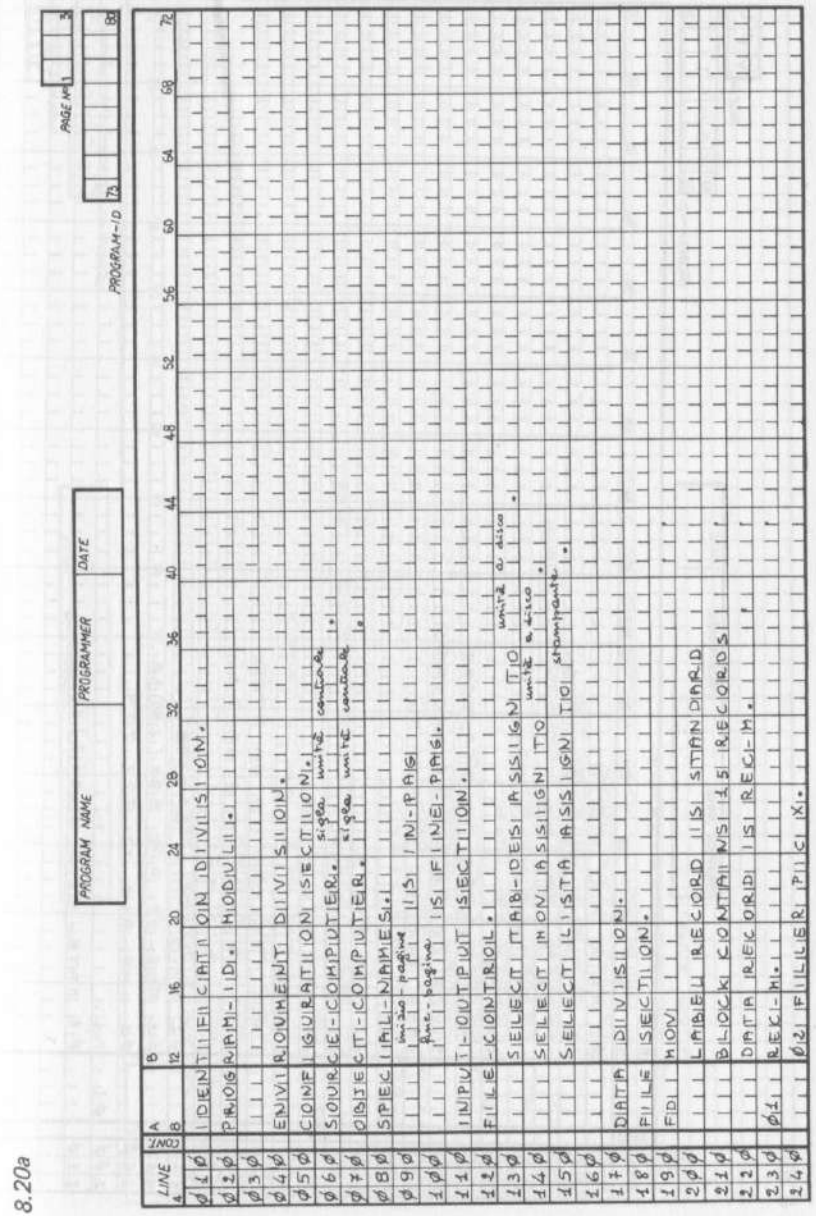


Fig. 8.20b

PAGE Nº 002

PROGRAM NAME	PROGRAMMER	DATE

PROGRAM-ID MODUL1

```
LINE  CONT.  A
4     B      B
                12   16   20   24   28   32   36   40   44   48   52   56   60   64   68   72
010            02 MATRI PICX(15).
020            02 ELIMI OCCURS 10 TIMES.
030               03 RIV PIC 99.
040               03 IMPIPIC 9(5)V.
050      FD    TABDESS
060            LABELRECORDS ARE STANDARD
070            BLOCKCONTAINS 10 RECORDS
080            DATA RECORDS ARE REC-T.
090      02    REC-T.
100            02 TSTP PIC X.
110            02 RVF PIC 99.
120            02 DESP PIC X(255).
130      FD    LISTA
140            LABELRECORDS OMITTED
150            DATA RECORDS ARE RIGA.
160      02    RIGA.
170            02 RVI-ST PIC 99.
180            02 DESI-STP PIC BBX(25)BB.
190            02 IMP PIC ZZ.ZZZ.ZZ9.
200      02    RIG.
210            02 MATRI-STP PIC B(72)X(15).
```

Fig. 8.20c

PAGE Nº 003

PROGRAM NAME	PROGRAMMER	DATE

PROGRAM-ID MODUL1

```
LINE  CONT.  A
4     B      B
                12   16   20   24   28   32   36   40   44   48   52   56   60   64   68   72
010      WORK  INGSTORAGE SECTION.
020      77    INDP PIC 99.
030      77    INDP PIC 99.
040      77    DMATR PIC X(5).
050      77    TOTP PIC 9(8) VALUE ZERO.
060      01    TAB
070            02 DESS-T PIC X(25) OCCURS 99 TIMES.
080
090      PROCEDURE DIVISION.
100      PROG-PRINCIPAL SECTION.
110      PERFORM INI-PROGR
120      PERFORM CARICA-TAB
130      PERFORM LETTURA.
140      IMP.  PERFORM INI-MATR.
150      T.    PERFORM TRATTAMI PERFORM LETTURA
160            IF DMATR=MATR GOTO TOTT.
170            PERFORM FINE-MATR
180            IF MATR MOTT=HIGH-VALUE GOTO TOTTIMP.
190            PERFORM FINE-PROGR
200            STOP RUN.
```

Fig. 8.20d

PAGE Nº 004

PROGRAM NAME	PROGRAMMER	DATE

PROGRAM-ID MODUL1

```
LINE  CONT  A
4     B     B
      B     12   16   20   24   28   32   36   40   44   48   52   56   60   64   68   72
010         IN-PROGR SECTION.
020         OPEN INPUT TABI-DESMON
030         OPEN OUTPUT LISTA
040         MOVE SPACES TO RIGA TABI.
050         EXIT-IN-PROGR. EXIT.
060
070         CARICA-TABI SECTION.
080         READ TABI-DES AT END GOTO EX-CARICA-TABI.
090         IF TSTNOT = "V" GOTO CARICA-TABI.
100         MOVE DES-TI(RVE) GOTO CARICA-TABB.
110         EXIT-CARICA-TABI. EXIT.
120
130         LETTURA SECTION.
140         READ MON AT END MOVE HIGH-VALUE TO MATR.
150         EXIT-LETTURA. EXIT.
160
170         IN-MATR SECTION.
180         MOVE MATR TO DMATR MATR-ST
190         WRITE RIGA AFTER ADVANCING M-PAG
200         MOVE SPACE TO RIGA
210         WRITE RIGA AFTER ADVANCING 3 LINES.
220         EXIT-IN-MATR. EXIT.
230
240
```

Fig. 8.20e

PAGE Nº 005

PROGRAM NAME	PROGRAMMER	DATE

PROGRAM-ID MODUL1

```
LINE  CONT  A
4     B     B
      B     12   16   20   24   28   32   36   40   44   48   52   56   60   64   68   72
010         TRATT TAM SECTION.
020         MOVE 1 TO INDI.
030         C.  IF RV (INDI) = 10 GOTO EX-TRATTAM.
040         ADD IMP (INDI) TO TOT
050         MOVE RV (INDI) TO RV-STI (INDI)
060         IF DES-TI (INDI) = SPACE MOVE ALL "." TO DES-STI
070         ELSE MOVE DES-TI (INDI) TO DES-STI.
080         MOVE IMP-STI (INDI) TO IMP-STI
090         WRITE RIGA BEFORE 1 LINE
100         MOVE SPACE TO RIGA.
110         ADD 1 TO INDI
120         IF INDI NOT > 100 GOTO C.
130         EXIT-TRATTAM. EXIT.
140
150         FINE-MATR-SEGTION.
160         MOVE TOT TO IMP-STI
170         WRITE RIGA AFTER FINE-PAG
180         MOVE SPACE TO RIGA
190         MOVE ZERO TO TOT.
200         EXIT-FINE-MATR. EXIT.
210
220         FINE-PROGRAM SECTION.
230         CLOSE LISTA MON TABI-DES.
240         EXIT-FINE-PROGR. EXIT.
```

9. ORGANIZZAZIONE SEQUENZIALE

Nei capitoli precedenti abbiamo introdotto alcuni elementi relativi alla gestione dell'input/output per flussi con organizzazione sequenziale: come abbiamo più volte ricordato, le affermazioni ed i formati introdotti erano parziali in quanto venivano forniti solo alcuni elementi necessari per poter scrivere (a livelli diversi) dei programmi esemplificativi.

In questo capitolo, e così pure in 10 ed 11, l'interesse si pone soprattutto sulle forme di organizzazione previste dallo standard Cobol e sugli accessi previsti e gestibili con istruzioni del linguaggio: è per questi motivi che gli esempi che introdurremo d'ora in poi saranno semplici e brevi.

9.1. Concetti generali

Un flusso organizzato in modo sequenziale è strutturato in modo tale che ogni record (ad eccezione di uno: "il primo") è preceduto da un solo record ed inoltre ogni record (ad eccezione di uno: "l'ultimo") è seguito da un solo record: la relazione predecessore-successore è stabilita dall'ordine di registrazione impartito quando il flusso viene generato. Si stabilisce inoltre che la relazione predecessore-successore non può cambiare, ad eccezione del caso in cui vengano accodati dei record (ovvero registrati dopo l'ultimo).

La definizione di flusso organizzato sequenzialmente fa riferimento all'attività di registrazione, tale attività va intesa in senso lato, ovvero sia come registrazione di un flusso su nastro o su disco, sia come perforazione (in linea) di schede, sia come stampa, ed infine anche come predisposizione manuale di record.

L'unica forma di accesso consentita in questo tipo di organizzazione è l'accesso sequenziale: i record possono rendersi disponibili solo nel modo con cui sono stati registrati.

9.2. Riferimenti in ENVIRONMENT DIVISION

In ENVIRONMENT DIVISION i riferimenti ai flussi sono riportati nei paragrafi FILE-CONTROL ed I-O-CONTROL della INPUT-OUTPUT SECTION.

In FILE-CONTROL per ogni flusso di elaborazione viene scritta una SELECT che, per flussi organizzati in modo sequenziale, ha la seguente struttura (notiamo che presenteremo d'ora innanzi dei formati per noi definitivi ma non sempre completi in quanto è nostro interesse evidenziare le particolarità che ci sembrano più significative):

```
SELECT [OPTIONAL] nome-flusso
    ASSIGN TO nome-periferica
    [ORGANIZATION IS SEQUENTIAL]
    [ACCESS MODE IS SEQUENTIAL]
    [FILE STATUS IS nome-dato] .
```

In questo formato si presentano alcuni elementi nuovi: quando una o entrambe le clausole ORGANIZATION e ACCESS non sono specificate viene assunto automaticamente che l'organizzazione e l'accesso siano sequenziali. Queste due clausole, di tipo documentativo, assumono un'importanza particolare (sempre a livello documentativo) quando il flusso è registrato su un supporto che consente anche l'accesso diretto.

Osserviamo che il ruolo delle due clausole è di tipo diverso in quanto la ORGANIZATION fa riferimento alla struttura logica del flusso, mentre la ACCESS al tipo di trattamento.

Quando è precisata la clausola STATUS il sistema operativo provvede ad inserire nel campo specificato, che deve essere un "X(2)" definito in DATA DIVISION ma non in FILE SECTION o in RE-

PORT SECTION (vedi appendice B), un valore standard, dopo l'esecuzione di una qualsiasi operazione di input oppure di output, in relazione al fatto che:

a. l'operazione sia andata a buon fine,
b. si sia riscontrata la fine del flusso,
c. si riscontri un errore permanente nell'operazione dopo che il sistema operativo ha tentato più volte di eseguirla.

In I-O-CONTROL si stabiliscono i punti in cui deve essere memorizzata la situazione del sistema per poter provvedere al ripristino del programma in caso di interruzione forzata, si possono inoltre definire delle aree che due o più flussi devono utilizzare in comune e si forniscono delle indicazioni circa il trattamento di volumi multiflusso.

9.3. Riferimenti in DATA DIVISION

Il "momento" della DATA DIVISION in cui sono interessate le descrizioni dei flussi sono le FD (File Description) di FILE SECTION.

In FD vengono specificati:

a. la presenza o meno delle etichette,
b. la lunghezza dei record logici e fisici,
c. le informazioni contenute nei record-etichetta associati ad un flusso,
d. i nomi dei record che compaiono nel flusso e che sono descritti in dettaglio nei successivi livelli 01.

```
FD nome-flusso
  [BLOCK CONTAINS [intero1 TO] intero2 {RECORDS    }]
                                       {CHARACTERS}
  [RECORD CONTAINS [intero3 TO] intero4 CHARACTERS]
  LABEL {RECORD IS  } {STANDARD}
        {RECORDS ARE} {OMITTED }
```



```
[DATA {RECORD IS  } nome-record1 [nome-record2] ... ].
      {RECORDS ARE}
```

Osserviamo che la clausola RECORD CONTAINS ha puramente valore documentativo in quanto la dimensione dei singoli record viene rilevata dalla loro definizione (infatti è la somma dei campi elementari che li costituiscono).

9.4. Riferimenti in PROCEDURE DIVISION

Per quanto riguarda i verbi di PROCEDURE DIVISION si presentano sostanzialmente solo alcune novità rispetto alle note introduttive sull'input per flussi sequenziali (vedi 5 e 7).

I flussi organizzati in modo sequenziale si prestano ad essere trattati anche in output in accodamento, come era già previsto dalla definizione (9.1), e in input/output, infatti è consentita la riregistrazione dei record (e precisamente dell'ultimo record letto).

9.4.1. *Il verbo OPEN*

Questo è il formato completo del verbo OPEN:

```
      {INPUT nome-flusso1 [REVERSED      ][nome-flusso2 [REVERSED      ]]...
      {                   [WITH NO REWIND]               [WITH NO REWIND]
OPEN  {OUTPUT nome-flusso3 [WITH NO REWIND] [nome-flusso4 [WITH NO REWIND]]...
      {I-O nome-flusso5 [nome-flusso6] ...
      {EXTEND nome-flusso7 [nome-flusso8] ...
```

L'istruzione OPEN inizia l'elaborazione del flusso e precisamente consente il controllo e/o la registrazione delle etichette di flusso e di altre operazioni di input-output.

L'esecuzione dell'istruzione OPEN con risultato positivo determina la disponibilità del flusso e rende disponibile al trattamento l'area record del flusso.

Fig. 9.1

ORGANIZATION IS SEQUENTIAL

ACCESS MODE IS SEQUENTIAL

Istruzioni	Tipi di OPEN			
	INPUT	OUTPUT	EXTEND	I-O
READ	X			X
WRITE		X	X	
REWRITE				X

La OPEN deve essere eseguita prima di un qualunque trattamento del flusso, la scelta del tipo di OPEN determina la possibilità ed i limiti di elaborazione: in fig. 9.1 sono schematizzate le possibilità operative in relazione alle diverse OPEN previste.

Analizziamo ora gli aspetti nuovi del verbo OPEN che, per quanto riguarda i formati INPUT e OUTPUT, è già stato ampiamente trattato (7).

Nel caso di OPEN EXTEND si ha un posizionamento sul flusso nella posizione seguente l'ultimo record logico caricato e quindi ogni successivo ordine di registrazione "aggiunge" un record al flusso già esistente.

In questo caso di OPEN se la clausola LABEL RECORD indica che le etichette di flusso sono presenti si ha la seguente sequenza di operazioni:

a. le etichette di inizio flusso sono trattate solo in caso di un flusso monovolume,
b. se si tratta di un flusso multivolume sono trattate solo le etichette di inizio del volume che deve essere soggetto ad accodamento (l'ultimo),
c. se ci sono delle etichette di fine-flusso sono elaborate ma vengono cancellate al momento della prima registrazione di accodamento.

Nel caso di OPEN I-O, permessa solo per flussi registrati su disco magnetico, se le etichette sono presenti, l'esecuzione di una OPEN

I-O provoca la lettura ed il controllo con una eventuale riregistrazione, se il sistema operativo prevede che le etichette di flusso debbano essere aggiornate.

9.4.2. *Il verbo CLOSE*

Per quanto riguarda questo verbo ci limitiamo a ritrascrivere il formato generale in quanto la descrizione fornita in 7. è sostanzialmente completa:

```
                    [ { REEL } [WITH NO REWIND]
                      { UNIT } [FOR REMOVAL]
CLOSE nome-flusso1  [ WITH { NO REWIND }
                           { LOCK     }  ]

[ nome-flusso2 [ { REEL } [WITH NO REWIND]
                 { UNIT } [LOCK]
               [ WITH { NO REWIND }
                      { LOCK     }  ] ....
```

9.4.3. *Il verbo READ*

L'istruzione di lettura rende disponibile, in area record del flusso, un record logico.

Nel caso di flussi con organizzazione sequenziale l'istruzione ha come formato completo quello presentato in 7.:

```
READ nome-flusso [INTO nome-dato][AT END ... .]
```

Osserviamo che questa istruzione può essere utilizzata sia dopo una OPEN INPUT che dopo una OPEN I-O.

Ricordiamo che quando è specificata la INTO dopo un comando di lettura il record letto è disponibile sia in area record sia nell'area precisata.

Se durante l'esecuzione di una READ viene rilevata la fine del volume (per flussi multivolume) allora viene eseguita questa sequenza di operazioni:

a. esecuzione della procedura standard di fine volume, (o bobina),
b. rilascio del volume (o della bobina),
c. trattamento iniziale del nuovo volume (o bobina),
d. trasferimento in area record del primo record del nuovo volume (o bobina).

Per flussi in input di tipo OPTIONAL, assenti durante l'elaborazione, si ha durante l'esecuzione della prima istruzione di lettura, un posizionamento simulato in posizione EOF (end of file).

Quando durante l'esecuzione dell'istruzione READ viene riscontrata la fine del flusso, e non è stata specificata la clausola AT END, allora deve essere prevista una USE (vedi 9.4.6) per la gestione da programma di questa situazione.

Ricordiamo infine che non è possibile effettuare nuove letture dopo l'uscita per fine flusso.

9.4.4. *Il verbo WRITE*

Il formato che abbiamo presentato in 7. può ritenersi completo per quanto riguarda gli elementi più significativi, ci limitiamo ora a ripresentarlo e a fare qualche considerazione supplementare.

```
WRITE nome-record [FROM nome-dato]
[ { BEFORE } ADVANCING { { nome-dato } { LINE  } } ]
  { AFTER  }           { { intero    } { LINES } }
                       { { nome-mnemonico }      }
                       { { PAGE           }      }
```

Ricordiamo che quando viene utilizzata l'opzione FROM viene definita in modo implicito una MOVE dell'area nome-dato all'area nome-record.

Dopo l'esecuzione di una istruzione di output non si ha alcuna garanzia sul contenuto della relativa area record (su taluni sistemi è riempita di spazi); se è presente la clausola FROM l'area nome-dato non viene alterata.

Quando (per flussi multivolume) viene riscontrata la fine fisica di un volume (o bobina) viene automaticamente effettuata questa sequenza di operazioni:

a. sul volume (o bobina) è registrata una etichetta di fine volume (o bobina),
b. viene rilasciato il volume (o bobina) esaurito,
c. vengono eseguite le procedure standard di inizio volume (o bobina).

Se durante l'esecuzione di un comando di registrazione viene rilevato un errore fisico di input/output (ad esempio si tenta di registrare su un flusso monovolume completo), si ha la seguente sequenza di operazioni:

a. posizionamento opportuno del FILE STATUS,
b. – se non è stata inserita una USE (vedi 9.6) il risultato dell'operazione è indefinito,
   – se invece è prevista una USE vengono eseguite le operazioni relative che sono state previste per questo caso d'errore.

9.4.5. *Il verbo REWRITE*

Questo verbo consente di sostituire un record di un flusso sequenziale su disco, il suo formato è

```
REWRITE nome-record [FROM nome-dato]
```

Il flusso a cui questa istruzione si riferisce deve essere su disco magnetico e deve essere stato trattato con una OPEN I-O.

L'ordine di riregistrazione riguarda (dal punto di vista fisico) l'area disco che contiene l'ultimo record letto, pertanto la lunghezza del record nome-record deve essere uguale alla lunghezza dell'ultimo record letto.

Nel caso in cui sia utilizzata l'opzione FROM è implicitamente definita una MOVE da nome-dato a nome-record.

9.4.6. *DECLARATIVES*

9.4.6.1. *Generalità*

Il linguaggio Cobol prevede che all'inizio della PROCEDURE DIVISION possa essere scritto un insieme di sezioni, una o più, raggruppate sotto il nome procedurale DECLARATIVES: tale insieme è costituito da istruzioni che devono essere eseguite solo al verificarsi di particolari condizioni durante l'esecuzione del programma.

Ad esempio, se durante l'esecuzione di una operazione di input/output viene rilevato un errore di tipo fisico e se nelle DECLARATIVES è stata specificata una procedura di trattamento per questo tipo di errore, allora il sistema operativo, invece di sospendere l'esecuzione, passa il controllo al programma e precisamente alla procedura di DECLARATIVES specificata.

Sappiamo che la PROCEDURE DIVISION può essere suddivisa in sezioni i cui nomi sono scelti dal programmatore; quando in un programma vengono utilizzate le DECLARATIVES, le sezioni relative alle varie procedure di controllo devono comparire all'inizio della PROCEDURE DIVISION globalmente precedute dalla parola Cobol DECLARATIVES (da colonna 8, seguita dal punto) e seguite dalle parole Cobol END DECLARATIVES (da colonna 8, seguita dal punto).

Ogni sezione di cui si compongono le DECLARATIVES deve iniziare con una frase USE che specifica al sistema operativo in quali circostanze devono essere eseguite le istruzioni contenute in quella sezione.

Esempio:

```
PROCEDURE DIVISION.
DECLARATIVES.
ERR1 SECTION.
     USE..... .
A1. .....
.....
ERR2 SECTION.
     USE ..... .
B1. .....
.....
```



```
B2. .....
END DECLARATIVES.
PROGRAMMA SECTION.
.
.
.
.
.
.
```

9.4.6.2. *Il verbo USE*

Questo verbo consente di identificare le procedure di trattamento degli errori di input/output, procedure che devono essere eseguite dopo quelle standard previste dal sistema operativo.

Il formato della USE è:

```
                    { EXCEPTION }
USE AFTER STANDARD  { ERROR     }
                    { nome-flusso1 [nome-flusso2]... }
                    { INPUT                          }
          PROCEDURE ON { OUTPUT                      }
                    { I-O                            }
                    { EXTEND                         }
```

L'istruzione USE, quando è presente, deve essere la prima istruzione di una sezione delle DECLARATIVES e deve terminare con un punto"." seguito da almeno uno spazio. Il resto della sezione è costituito da uno o più paragrafi che nel complesso costituiscono l'insieme di istruzioni da eseguire in determinate circostanze.

In particolare si può richiedere il richiamo della procedura in caso di errore per uno o più flussi i cui nomi sono specificati, per i flussi aperti in INPUT oppure in OUTPUT oppure in I-O oppure EXTEND.

I termini ERROR e EXCEPTION sono sinonimi e possono essere intercambiabili.

9.5. Un esempio

Il programma che presentiamo provvede all'aggiornamento di un archivio ARK mediante informazioni registrate su un flusso movimenti MOV.

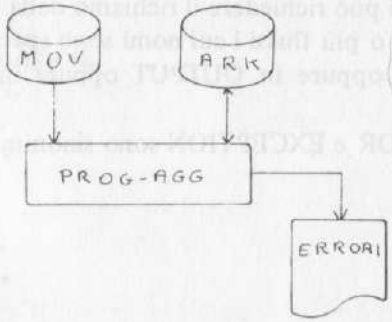
Il flusso archivio ARK è ordinato sul codice COD-ARK e ogni suo record contiene un campo TOTALE; ARK non contiene record con codice uguale.

Il flusso movimenti MOV è ordinato sul codice COD-MOV e ogni suo record contiene un campo IMPORTO; per ogni codice possono esserci uno o più record.

L'elaborazione consiste nello scorrere parallelamente i due flussi: se c'è corrispondenza fra codice archivio e codice movimento bisogna aggiornare TOTALE aggiungendogli la somma dei valori di IMPORTO e quindi riregistrare il record aggiornato; se un record di archivio non ha corrispondenti movimenti non deve essere trattato; se infine dei movimenti non hanno corrispondente record in archivio deve essere stampata per ogni movimento una segnalazione (COD-MOV e IMPORTO).

In fig. 9.2 riportiamo l'organigramma del problema proposto (si noti che il flusso ARK è indicato in aggiornamento), in fig. 9.3 sono riprodotti i tracciati dei flussi di elaborazione (non è riportato il tracciato stampa perchè procederemo con DISPLAY di messaggi), nelle figg. 9.4 e 9.5 sono indicati rispettivamente il flow-chart descrittivo della soluzione del problema e la relativa codifica.

Fig. 9.2



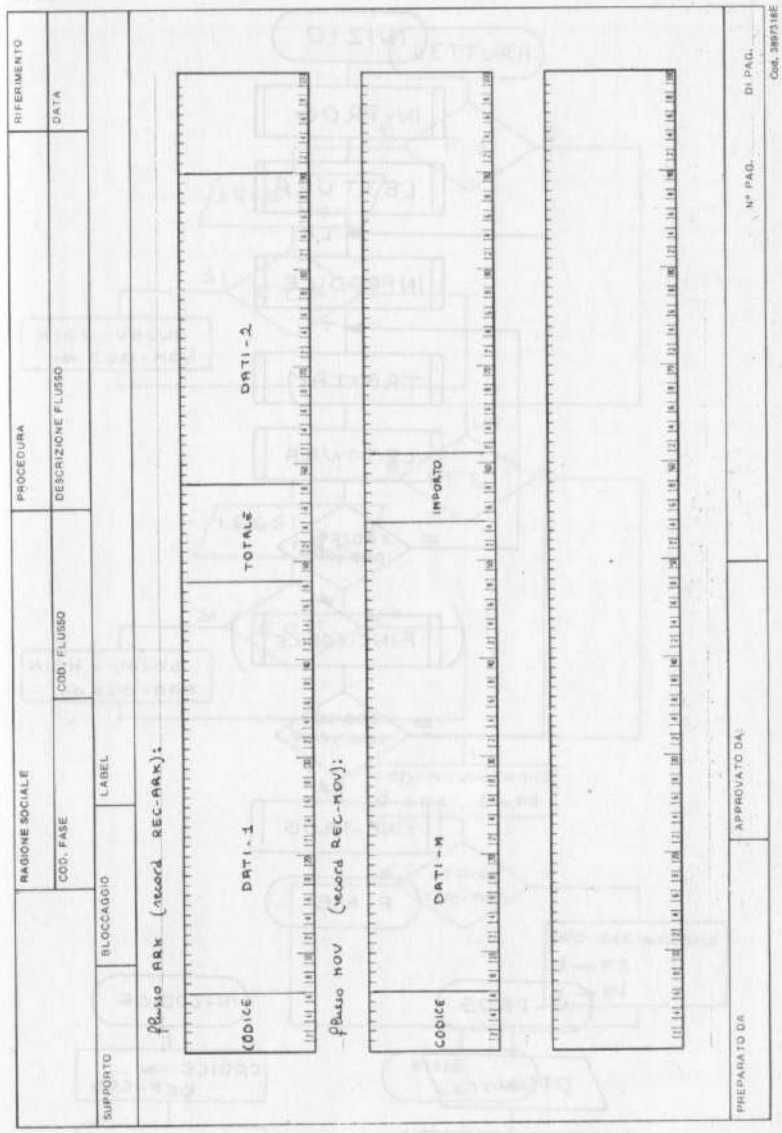


Fig. 9.3

Fig. 9.4a

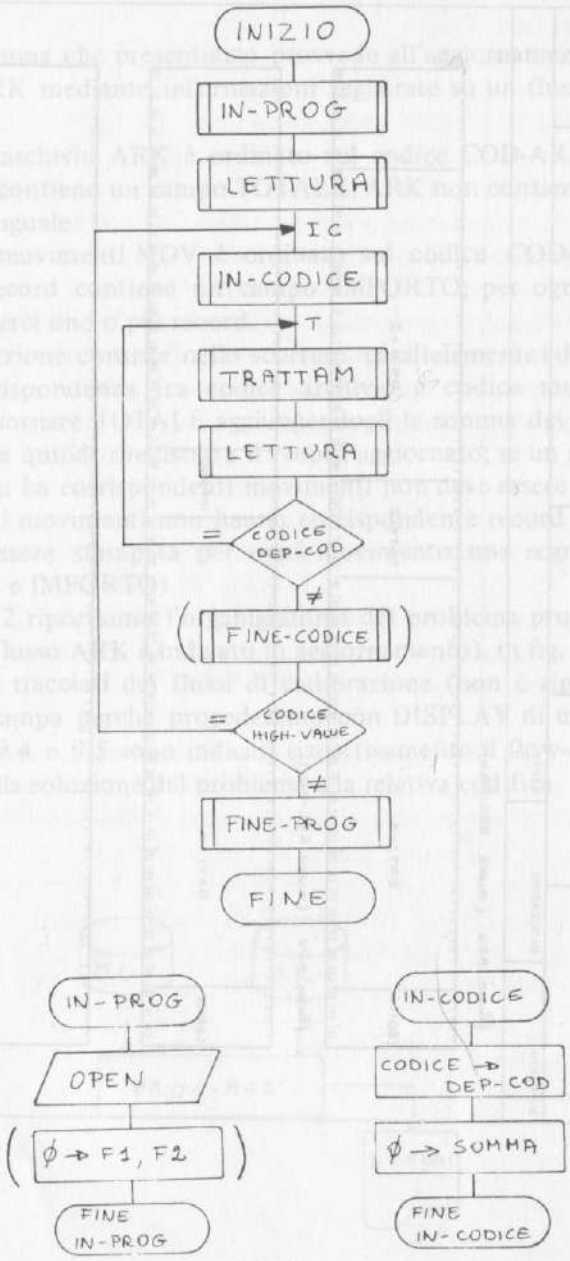


Fig. 9.4b

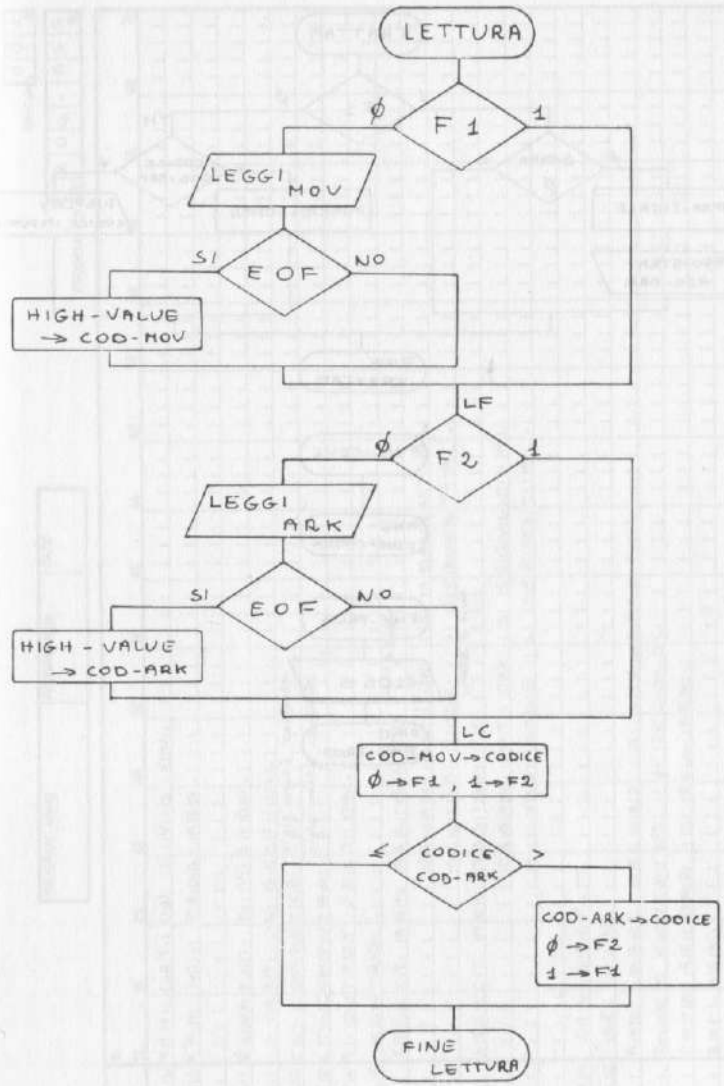


Fig. 9.4c

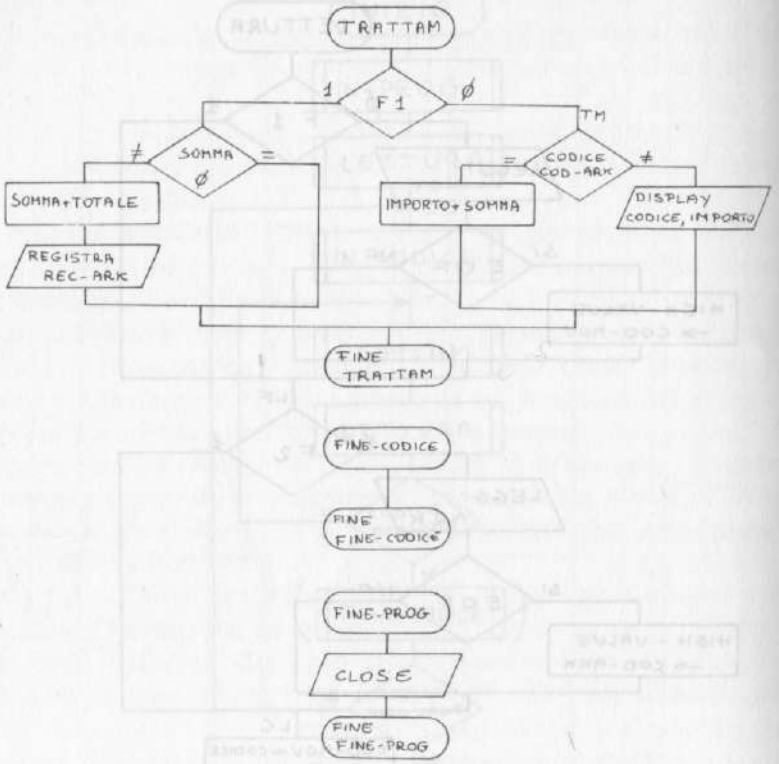


Fig. 9.5a

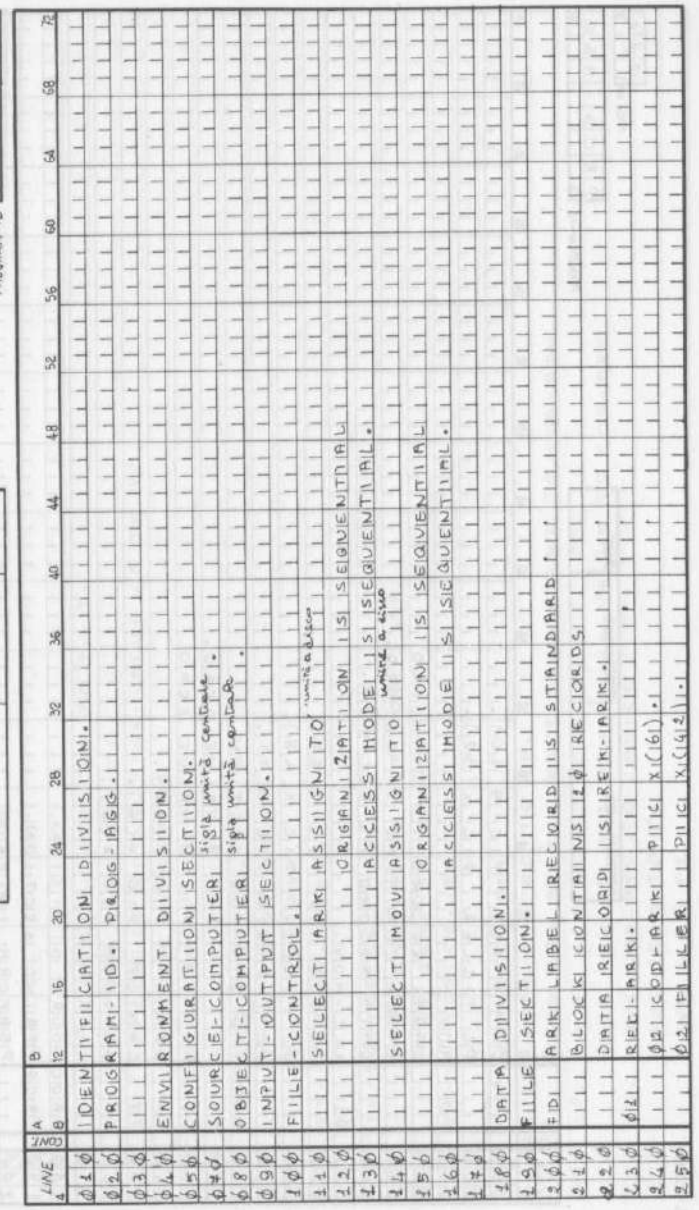


Fig. 9.5b

PROGRAM NAME | PROGRAMMER | DATE

PAGE NO. 002

PROGRAM-ID PROG-AGG

LINE	CONT.	A	B
010			02 TOTALE PICX(131).
020			02 FILLER PICX(132).
030		FD	MON
040			LABEL RECORD IS STANDARD
050			BLOCK CONTAINS 18 RECORDS
060			DATA RECORD IS REC-MOV.
070		02	REC-MOV.
080			02 CODMOV PICX(6).
090			02 FILLER PICX(48).
100			02 IMPORTO PIC9(7).
110		WORKING-STORAGE SECTION.	
120	77	77	PIC9 VALUE ZERO.
130	77	77	PIC9 VALUE ZERO.
140	77	SOMMA	PIC9(10).
150	77	CODICE	PICX(6).
160	77	DEP-COD	PICX(6).
170			
180	PROCEDURE DIVISION.		
190	PROG-PRINCIPALE SECTION.		
200		PERFORM IN-PROG	
210		PERFORM LETTURA.	
220	IC.	PERFORM IN-CODICE.	
230	T.	PERFORM TRATTAM	
240		PERFORM LETTURA	

Fig. 9.5c

PROGRAM NAME | PROGRAMMER | DATE

PAGE NO. 003

PROGRAM-ID PROG-AGG

LINE	CONT.	A	B
010			IF CODICE = DEP-CODI GOTO T.
020			PERFORM FINE-CODICE
030			IF CODICE NOT = HIGH-VALUE GOTO IC.
040			PERFORM FINE-PROG
050			STOP RUN.
060			
070		IN-PROG SECTION.	
080			OPEN INPUT MOV
090			OPEN I-O ARK.
100	EX-IN	N-PROG.	EXIT.
110			
120	IN-CODICE SECTION.		
130		MOVE CODICE TO DEP-CODI	
140		MOVE ZERO TO SOMMA.	
150	EX-IN	N-CODICE.	EXIT.
160			
170	LETTURA SECTION.		
180		IF F1 = 1 GO TO LF.	
190		READ MOV AT END MOVE HIGH-VALUE TO CODI-MOV.	
200	LF.	IF F2 = 1 GO TO LC.	
210		READ ARK AT END MOVE HIGH-VALUE TO CODI-ARK.	
220	LC.	MOVE CODI-MOV TO CODICE	
230		MOVE ZERO TO F2	
240		MOVE 1 TO F2	

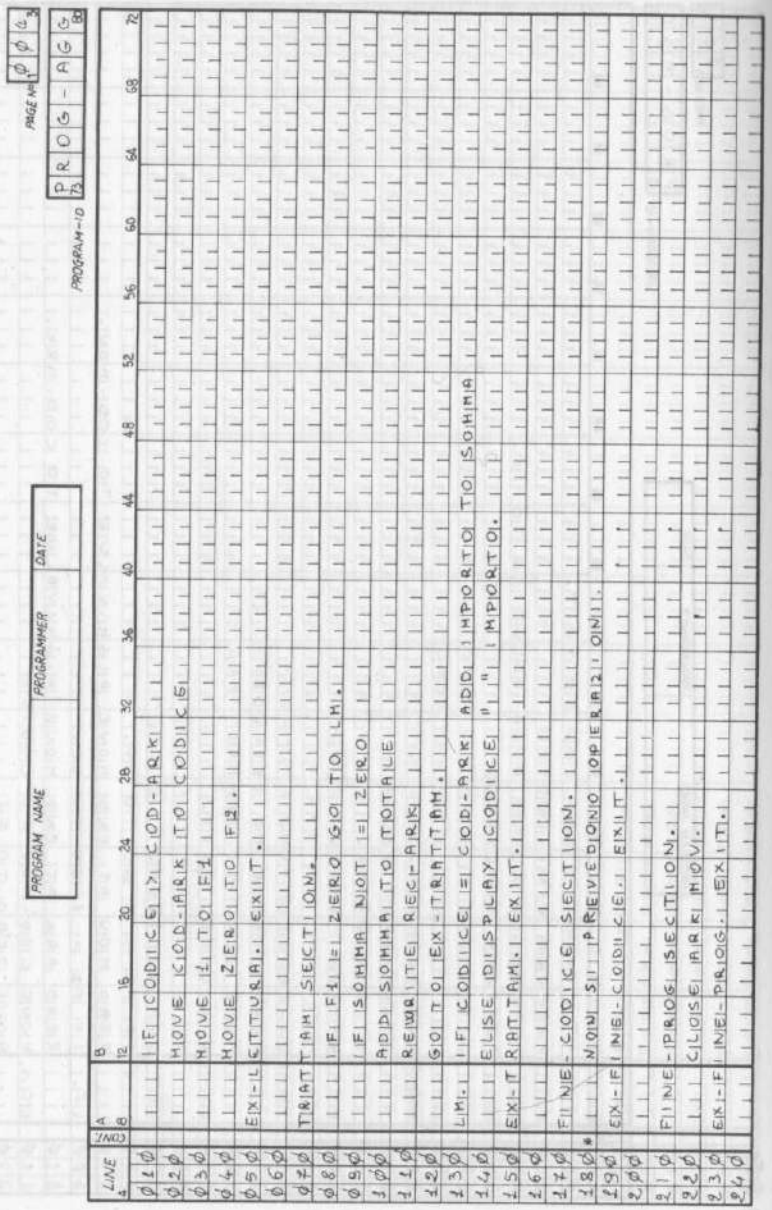


Fig. 9.5d

9.6. Riepilogo

In questo paragrafo vogliamo brevemente schematizzare le possibilità offerte dall'organizzazione sequenziale (ricordiamo che è consentito solo l'accesso sequenziale).

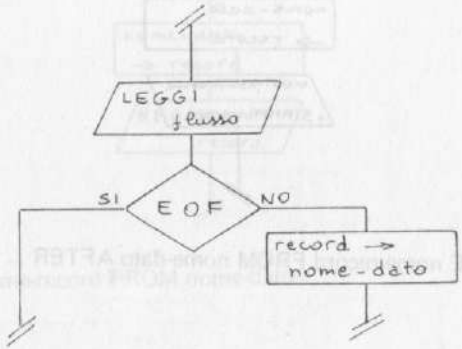
Tutte le volte che compare la frase AT END questa può essere sostituita da una opportuna procedura di USE.

a. OPEN INPUT

1.

READ nome-flusso AT END ... • ...

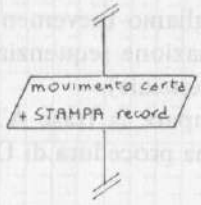
2.



READ nome-flusso INTO nome-dato AT END ... • ...

b. OPEN OUTPUT

1.



WRITE nome-record AFTER ...

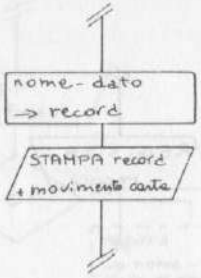
2.

WRITE nome-record BEFORE....

3.

WRITE nome-record FROM nome-dato AFTER ...

4.



WRITE nome-record FROM nome-dato BEFORE ....

5.

WRITE nome-record

6.

WRITE nome-record FROM nome-dato

c. OPEN EXTEND

1.

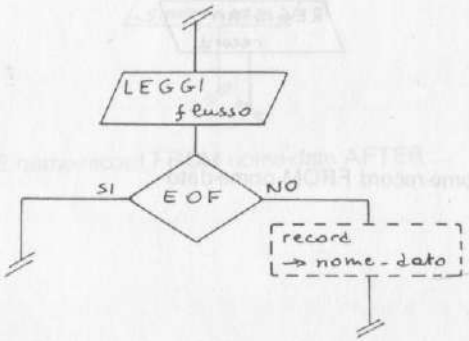
WRITE nome-record

2.

WRITE nome-record FROM nome-dato

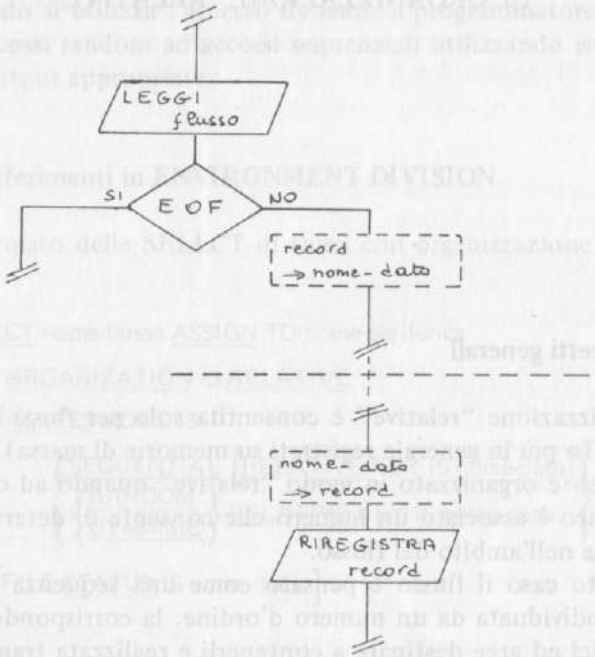
d. OPEN I-O

1.



READ nome-flusso INTO nome-dato [AT END... .]...

2.



REWRITE nome-record [FROM nome-dato]

10. ORGANIZZAZIONE "RELATIVE"

10.1. Concetti generali

L'organizzazione "relative" è consentita solo per flussi su disco magnetico (o più in generale registrati su memorie di massa).

Un flusso è organizzato in modo "relative" quando ad ogni suo record logico è associato un numero che consente di determinarne la posizione nell'ambito del flusso.

In questo caso il flusso è pensato come una sequenza di aree, ciascuna individuata da un numero d'ordine: la corrispondenza tra record logici ed aree destinate a contenerli è realizzata tramite una formula che consente di assegnare ad ogni record logico la sua posizione nell'ambito del flusso.

E' utilizzando i numeri (relativi) di record che è possibile determinare dove deve essere registrato (o ricercato) un record logico.

Gli accessi ad un flusso "relative" possono essere:

a. sequenziale,
b. random,
c. dynamic.

Quando l'accesso è sequenziale, la sequenza con cui si accede ai record è quella corrispondente ai numeri relativi di record crescenti, in lettura vengono presi in considerazione solo i record che sono effettivamente registrati sul flusso, nel senso che se su qualche zona record del supporto non è stato registrato un record logico, tale area viene ignorata.

In accesso random la sequenza di accesso ai record è controllata dal programmatore che determina, in relazione alla chiave da trattare, il corrispondente numero di record.

Quando si utilizza l'accesso dynamic il programmatore può alternare accessi random ad accessi sequenziali utilizzando istruzioni di input-output appropriate.

10.2. Riferimenti in ENVIRONMENT DIVISION

Il formato delle SELECT di flussi con organizzazione "relative" è:

```
SELECT nome-flusso ASSIGN TO nome-periferica
       ORGANIZATION IS RELATIVE
       [ACCESS MODE IS
          { SEQUENTIAL [RELATIVE KEY IS nome-dato] }
          { { RANDOM  } RELATIVE KEY IS nome-dato  }
          { { DYNAMIC }                            }  ]
       [FILE STATUS IS nome-dato1] .
```

Ogni flusso "relative" utilizzato nel programma deve avere una SELECT di questo tipo: se non è specificato il tipo di accesso è inteso SEQUENTIAL.

L'organizzazione specifica la struttura logica del flusso e pertanto una volta che per un flusso è stata definita (ed il flusso è stato generato) non può essere cambiata; il tipo di accesso può invece variare in relazione alle esigenze di trattamento del flusso.

La RELATIVE KEY è quel campo, che non può coincidere con una porzione dell'area record, in base al quale viene determinato il numero del record nell'ambito del flusso: come si può osservare deve essere specificato nel caso di accesso RANDOM o DYNAMIC, mentre nel caso di SEQUENTIAL è opzionale.

La RELATIVE KEY è obbligatoria nel caso di accesso RANDOM in quanto l'accesso al record è diretto sulla base della posizione calcolata ed inserita nella RELATIVE KEY; per quanto riguarda l'accesso DYNAMIC bisogna ricordare che questo tipo di accesso consente di trattare sequenzialmente porzioni di flusso dopo dei posizionamenti in accesso diretto sul primo record che interessa.

Il campo nome-dato$_1$, che deve essere definito in DATA DIVI-

SION ma non in FILE SECTION o in REPORT SECTION (vedi appendice B.), deve essere un "X(2)", e viene aggiornato dopo ogni operazione di input/output.

L'analisi di nome-dato$_1$ consente di rilevare una delle seguenti situazioni:

a. operazione eseguita con esito positivo,
b. rilevazione del fine flusso,
c. operazione eseguita una con esito negativo (esamineremo i possibili eventi quando tratteremo le singole istruzioni),
d. errore permanente di input/output.

Per quanto riguarda la I-O-CONTROL osserviamo che per un flusso "relative" è possibile specificare le modalità che regolano la memorizzazione dello stato del sistema nell'eventualità di dovere ripristinare il lavoro in caso di sospensione forzosa; inoltre è possibile assegnare la stessa area record o la stessa area blocco a differenti flussi.

10.3. Riferimenti di DATA DIVISION

Per quanto riguarda la DATA DIVISION le modalità di scrittura delle FD e delle descrizioni dei record restano invariate, bisogna solamente ricordare che il nome-dato specificato nella RELATIVE KEY non deve essere il nome di un campo dell'area record del flusso in oggetto.

10.4. Riferimenti in PROCEDURE DIVISION

I flussi organizzati in modo "relative" si prestano, come abbiamo già detto, a diversi tipi di accessi: il tipo di OPEN fissato consente di determinare il modo in cui deve avvenire l'accesso al flusso e di conseguenza il tipo di istruzioni che possono essere utilizzate e le loro caratteristiche operative.

10.4.1. *Il verbo OPEN*

Il formato del verbo OPEN per flussi organizzati in modo "relative" è:

```
        { INPUT  nome-flusso1 [nome-flusso2] ... }
OPEN    { OUTPUT nome-flusso3 [nome-flusso4] ... }
        { I-O    nome-flusso5 [nome-flusso6] ... }
```

L'esecuzione della OPEN inizia il trattamento del flusso, abilitando i tipi di operazioni possibili, e controlla, e se è il caso genera, le etichette di flusso.

L'esecuzione dell'istruzione OPEN con risultato positivo determina la disponibilità del flusso ed abilita al trattamento l'area record del flusso.

La OPEN deve essere eseguita prima di ogni altro trattamento del flusso; in fig. 10.1 è riportata la tabella dei possibili trattamenti in relazione alla OPEN utilizzata.

10.4.2. *Il verbo CLOSE*

Questo verbo consente di terminare l'elaborazione di un flusso utilizzando il formato:

```
CLOSE nome-flusso1 [WITH LOCK]
      [nome-flusso2 [WITH LOCK]]...
```

I flussi, i cui nomi sono citati nella CLOSE, non occorre che abbiano la stessa organizzazione: perchè possa essere eseguita questa istruzione è sufficiente che il flusso sia stato trattato con una OPEN.

L'opzione LOCK consente di impedire che il flusso possa essere oggetto di una successiva OPEN durante l'esecuzione della stessa unità di elaborazione.

Fig. 10.1

TIPI DI ACCESSO	ISTRUZIONI	TIPI DI OPEN		
		INPUT	OUTPUT	I-O
SEQUENTIAL	READ	X		X
	WRITE		X	
	REWRITE			X
	START	X		X
	DELETE			X
RANDOM	READ	X		X
	WRITE		X	X
	REWRITE			X
	START			
	DELETE			X
DYNAMIC	READ	X		X
	WRITE		X	X
	REWRITE			X
	START	X		X
	DELETE			X

10.4.3. *Il verbo READ*

L'istruzione di lettura, nel caso di accesso sequenziale, rende disponibile il "prossimo" record logico in area record; nel caso di accesso random, l'istruzione di lettura rende disponibile "un record specificato" in area record; per quanto riguarda l'accesso dynamic è sufficiente tenere presente che è possibile un alternarsi di accessi sequenziali e random.

I formati della READ per flussi con organizzazione "relative" sono due:

Formato 1:

READ nome-flusso [NEXT] RECORD

[INTO nome-dato]

[AT END ....•]

Formato 2:

READ nome-flusso RECORD [INTO nome-dato]

[INVALID KEY ....•]

La frase INTO, in entrambi i formati, non può essere usata quando il flusso contiene record di lunghezza diversa.

Il formato 1 deve essere usato per tutti i flussi con accesso sequenziale (in questo caso non viene indicata la parola NEXT).

Il formato 1 con la parola NEXT deve essere usato per flussi con accesso dynamic quando il trattamento dei record è sequenziale.

Il formato 2 è utilizzato per flussi a cui si accede in modo random oppure per ricerca random in flussi con accesso dynamic.

Le frasi INVALID KEY ed AT END sono obbligatorie a meno che non sia specificata una opportuna USE (nelle DECLARATIVES) con i relativi trattamenti.

Il verbo READ può essere utilizzato solo per flussi preventivamente trattati con una OPEN INPUT oppure con una OPEN I-O.

Per quanto riguarda il formato 1 possiamo affermare che:

a. Viene reso disponibile il "prossimo" record logico, nel senso di "quello fisicamente successivo all'ultimo trattato"; naturalmente è inteso che:
   - per flussi con accesso sequenziale, il primo record da trattare dopo la OPEN è il primo record fisicamente registrato sul flusso;
   - per flussi con accesso dynamic il primo record da trattare è quello cui si "punta" dopo l'esecuzione di una START (istruzione di posizionamento che verrà esaminata più avanti) oppure di un ordine di lettura formato 2 terminato con esito positivo;

– i record eventualmente presenti ma cancellati logicamente vengono ignorati automaticamente: la READ consente quindi di analizzare solo i record logicamente presenti sul flusso ignorando anche le aree record non contenenti alcun record logico;
– per quanto riguarda gli ordini di lettura successivi al primo, il record reso disponibile è il "prossimo" a meno che, per l'accesso dynamic, non sia stata fatta nuovamente una START oppure una READ di formato 2;
– se al momento dell'esecuzione dell'istruzione di lettura non è definito quale sia il record da rendere disponibile, allora l'esecuzione dell'operazione non può avvenire correttamente (è il caso di una READ... NEXT, per flussi in accesso dynamic, non preceduta da una START eseguita con esito positivo oppure da un'altra READ... NEXT che non riscontri l'indicatore di EOF);

b. quando viene incontrato l'identificativo di fine flusso (e vengono eseguite le operazioni specificate dopo la frase AT END oppure nella USE relativa) è possibile dare una nuova istruzione di lettura solo in questi casi:
– per accesso sequenziale, dopo l'esecuzione di una CLOSE e di una successiva OPEN che abiliti alla lettura in sequenziale;
– per accesso dynamic, dopo l'esecuzione con esito positivo di una nuova START oppure di una READ di formato 2;

c. se nel formato 1 è specificata la RELATIVE KEY viene continuamente aggiornata sulla base del numero relativo del record che viene reso disponibile.

Per quanto riguarda il formato 2 osserviamo che: o viene fornito il record richiesto (la cui posizione è determinata sulla base della RELATIVE KEY, che è predisposta dal programmatore con calcoli fatti a partire dalla chiave del record), oppure, nel caso in cui il record non venga trovato nella posizione richiesta, vengono eseguite le istruzioni scritte dopo la frase INVALID KEY, o, in alternativa, la procedura precisata da una USE appropriata.

10.4.4. *Il verbo WRITE*

Il verbo WRITE consente di registrare un record logico su flussi aperti in output oppure in input/output: la registrazione, come al solito, può interessare direttamente la periferica di output oppure semplicemente opera un trasferimento dall'area record all'area blocco del flusso.

Il formato di questo verbo è:

```
WRITE nome-record [FROM nome-dato]
      [INVALID KEY .... .]
```

Quando si utilizza l'opzione FROM bisogna tenere presente che nome-dato non può essere un identificativo dell'area record o di parte di essa e che al momento dell'esecuzione della WRITE si ha un trasferimento del contenuto di nome-dato in area record e quindi avviene la fase vera e propria di registrazione (nel senso visto or ora).

Quando l'accesso al flusso è di tipo sequenziale i record vengono registrati per numeri relativi crescenti, se la RELATIVE KEY è associata al flusso viene continuamente aggiornata con il numero relativo dell'"ultimo" record registrato.

Per flussi con accesso sequenziale la WRITE può essere utilizzata solo nel caso di OPEN OUTPUT.

Per flussi con accesso random o dynamic la WRITE può essere utilizzata per flussi aperti in output oppure in input/output: quando si utilizzano questi tipi di accesso il valore della RELATIVE KEY deve essere opportunamente calcolato per poter indirizzare la posizione in cui il record deve essere registrato.

Si è in condizione di INVALID KEY (nel senso che vengono eseguite le istruzioni scritte dopo questa frase oppure, se non è presente, quelle indicate nelle DECLARATIVES da una opportuna USE) quando si verifica una delle seguenti circostanze:

a. l'accesso al flusso è random o dynamic e la RELATIVE KEY indirizza una posizione del flusso già occupata da un altro record;
b. quando si tenta di registrare oltre i limiti fisici del flusso.

Quanto specificato al precedente punto a. conferma che il Cobol

non prevede la gestione dei sinonimi: ricordiamo che nella descrizione della organizzazione "relative" era stato detto che ad ogni record deve essere assegnata da programma una posizione nell'ambito del flusso e pertanto il programmatore deve preoccuparsi della gestione di record a cui la formula di corrispondenza associa lo stesso numero di record se non si può fare in modo che questo non succeda.

10.4.5. *Il verbo REWRITE*

Questo verbo consente di sostituire un record esistente su un flusso (registrato su una memoria di massa); a tale scopo si utilizza il formato:

```
REWRITE nome-record [FROM nome-dato]
    [INVALID KEY ... .]
```

Il record che deve sostituire quello esistente può essere preparato in area record oppure in un'area di memoria diversa dall'area record.

La frase INVALID KEY deve essere omessa nel caso in cui l'accesso al flusso sia sequenziale: per questo tipo di accesso infatti il record interessato dalla registrazione è l'ultimo record letto.

La frase INVALID KEY (o una corrispondente USE) nel caso di accesso random o dynamic consente di precisare quali procedure debbano essere eseguite nel caso in cui non venva trovato il record da sostituire: con questi tipi di accesso è infatti possibile richiedere l'esecuzione di una REWRITE senza aver effettuato un'operazione di lettura del record che si vuole modificare.

Ricordiamo che il record che viene registrato al posto di uno già esistente deve avere lo stesso numero di caratteri.

10.4.6. *Il verbo START*

Questo verbo consente di effettuare dei posizionamenti all'interno del flusso per poi procedere in modo sequenziale; il formato di questa istruzione è:


```
                  [ IS EQUAL TO      ]
                  [ IS =             ]
START nome-flusso [ KEY IS GREATER TO ] nome-dato ]
                  [ IS >             ]
                  [ IS NOT LESS THAN ]
                  [ IS NOT <         ]

[ INVALID KEY ... . ]
```

Il flusso, che deve essere trattato in accesso sequenziale o dynamic, deve essere stato aperto in INPUT oppure in I-O.

Nome-dato specifica, se l'opzione KEY è presente, il nome del campo dichiarato come RELATIVE KEY della SELECT del flusso.

Se l'opzione KEY è presente il tipo di KEY specificata consente di effettuare il posizionamento in corrispondenza della posizione indicata dalla RELATIVE KEY (=), della prima posizione occupata dopo quella indicata dalla RELATIVE KEY (>), oppure, della prima posizione occupata a partire da quella indicata dalla RELATIVE KEY (NOT <): quest'ultima forma è quella normalmente utilizzata.

Nel caso non sia presente l'opzione KEY è sottinteso "KEY IS EQUAL TO nome-dato".

Se la ricerca del record su cui posizionarsi ha esito negativo allora vengono eseguite le istruzioni scritte dopo la frase INVALID KEY oppure, se questa non è indicata, viene eseguita la procedura prevista da una USE opportuna che deve sopperire alla mancanza dell'INVALID KEY.

10.4.7. *Il verbo DELETE*

Questo verbo ha la funzione di cancellare logicamente un record del flusso; il suo formato è:

```
DELETE nome-flusso RECORD [INVALID KEY ... .]
```

La frase INVALID KEY non deve essere specificata quando l'ac-

cesso al flusso è sequenziale, infatti il record trattato con una DELETE, per flussi con questo tipo di accesso, è l'ultimo record letto.

Per flussi che vengono elaborati con accesso random o dynamic la frase INVALID KEY (o una USE sostitutiva) deve essere indicata, infatti è possibile dare l'ordine di cancellazione senza avere eseguito preventivamente una READ, e pertanto il flusso potrebbe non contenere un record valido nella posizione indicata dalla RELATIVE KEY.

Dopo l'esecuzione di una DELETE il record trattato risulta logicamente cancellato dal flusso e non può più essere trattato da programma.

L'esecuzione di una DELETE non altera l'area record del flusso in trattamento.

10.4.8. *Il verbo USE*

Il verbo USE specifica le procedure che devono essere eseguite in concomitanza di errori nel trattamento dell'input/output, sia quando l'errore è di sistema e viene prima trattato dalle routines standard, sia quando è un errore o una situazione particolare rilevata da una istruzione di input/output e sono state omesse le opzioni INVALID KEY oppure AT END.

Il formato della USE è:

$$\underline{\text{USE}}\ \underline{\text{AFTER}}\ \text{STANDARD} \left\{ \begin{array}{l} \underline{\text{EXCEPTION}} \\ \underline{\text{ERROR}} \end{array} \right\}$$

$$\underline{\text{PROCEDURE}}\ \text{ON} \left\{ \begin{array}{l} \text{nome-flusso}_1 \left[\text{nome-flusso}_2 \right] \ldots \\ \underline{\text{INPUT}} \\ \underline{\text{OUTPUT}} \\ \underline{\text{I-O}} \end{array} \right\}$$

e le caratteristiche (con le opportune limitazioni del formato) sono le stesse viste per i flussi con organizzazione sequenziale (vedi 9.4.6.).

10.5. Un esempio

Un flusso FL-1 organizzato in modo "relative" debba essere aggiornato in base a movimenti registrati su un flusso FL-2 sequenziale.

L'aggiornamento consiste in inserimenti, variazioni e cancellazioni, per quanto riguarda le variazioni non entriamo nel merito su "cosa si debba fare" in quanto il problema dal nostro punto di vista si riduce alla ricerca e, a modifica effettuata, alla riregistrazione del record.

Per quanto riguarda la cancellazione vedremo, in questo caso, che la gestione sarà a carico del programmatore in quanto FL-1 contiene dei sinonimi e gestire le cancellazioni da programma, senza l'uso della DELETE, agevola il trattamento dei sinonimi.

Supponiamo che il flusso FL-1 sia costituito al più da 1200 records e che il resto della divisione della chiave del record (numerica, se non lo fosse potremmo pensare al suo valore binario) per 997 sia distribuito in modo abbastanza uniforme: le chiavi dei record siano tali da dare origine a circa 900 resti diversi, il numero massimo di sinonimi di una stessa chiave (chiavi con uguale resto) sia piuttosto basso.

I presupposti enunciati implicano che mediamente una sola chiave su tre ammette un sinonimo.

Analizziamo la tecnica che si è stabilito di seguire in questo caso per la determinazione del "numero di record" a partire dalla chiave del record.

Ogni record di FL-1 è pensato costituito da due parti:

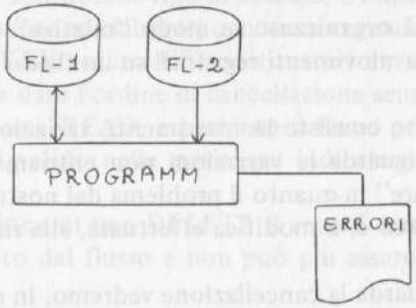
a. le informazioni che costituiscono il record logico vero e proprio (REC-VP)
b. informazioni supplementari per la gestione del record nel flusso.

In fig. 10.2 è descritto l'organigramma del problema ed in fig. 10.3 il tracciato dei flussi.

L'area di disco abilitata a contenere il flusso FL-1 sia definita in modo tale da poter contenere 1500 record REC-1:

– 997 sono le locazioni a cui si accede direttamente dopo il calcolo del resto (assumiamo che se il resto è zero viene considera-

Fig. 10.2



to pari a 997), di queste posizioni, in base alle ipotesi fatte, ne sono impegnate 900;

– 300 è il numero delle locazioni atte a contenere i sinonimi;

– 202 sono locazioni di scorta;

– 1 locazione (la 998-esima) è destinata a contenere un numero che indica la prima posizione libera dell'area sinonimi (precisamente nella zona relativa al codice).

La registrazione di un record sul flusso avviene secondo queste modalità:

a. viene calcolato il resto della divisione della chiave per 997,

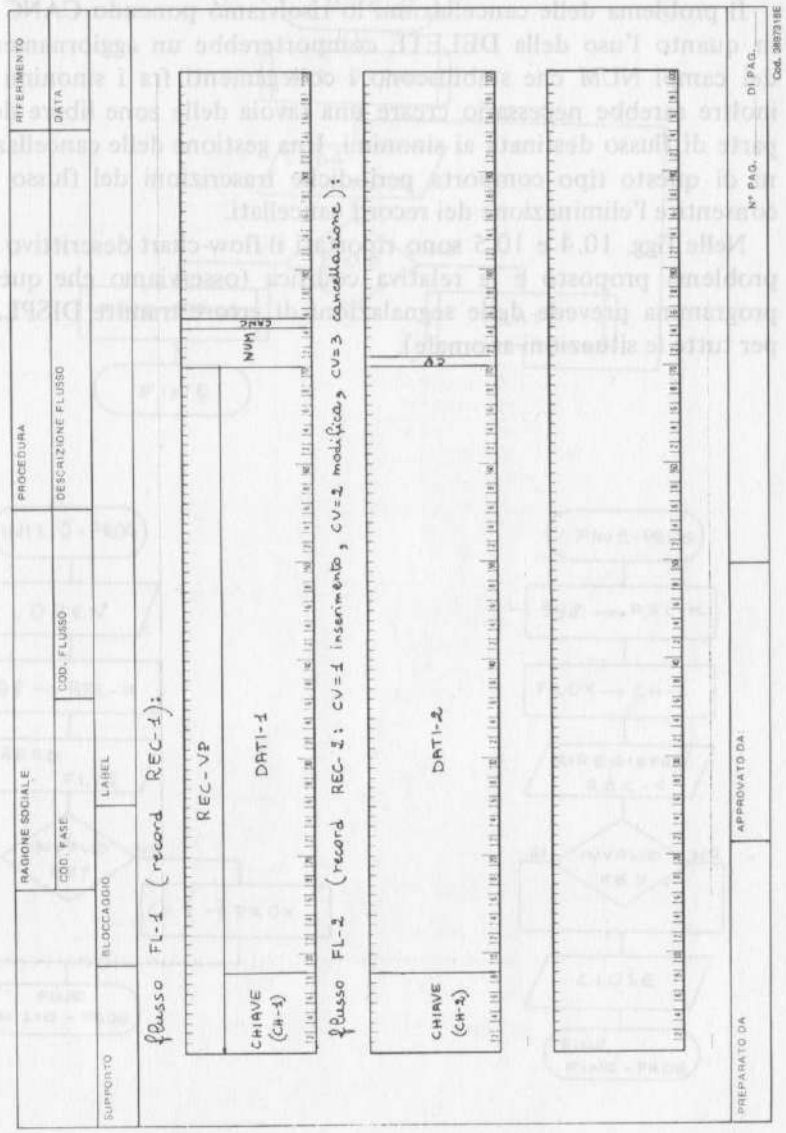
b. se tale resto identifica una zona libera il record viene registrato in quella posizione con il campo NUM a zero e CANC a zero.

c. se il resto identifica una zona occupata da un altro record, e quest'ultimo contiene zero in NUM, viene inserito in NUM il valore corrispondente alla prima zona libera e in tale zona viene registrato il nuovo record, ponendo zero in CANC e NUM, e si aggiorna il valore corrispondente alla prossima zona libera.

d. se il resto identifica una zona occupata da un altro record e quest'ultimo contiene in NUM un valore diverso da zero si va a reperire in accesso diretto il record identificato da NUM e si procede in questo modo finché non si individua un record che è l'ultimo della catena dei sinonimi in quanto ha il suo campo NUM uguale a zero e quindi si procede come in c.

Alla fine del programma bisogna riregistrare in posizione 998 il

Fig. 10.3



valore della prima zona libera (la posizione 998 al momento della prima generazione del flusso contiene il valore 999).

Il problema delle cancellazioni lo risolviamo ponendo CANC a 1 in quanto l'uso della DELETE comporterebbe un aggiornamento dei campi NUM che stabiliscono i collegamenti fra i sinonimi ed inoltre sarebbe necessario creare una tavola delle zone libere della parte di flusso destinata ai sinonimi. Una gestione delle cancellazioni di questo tipo comporta periodiche trascrizioni del flusso per consentire l'eliminazione dei record cancellati.

Nelle figg. 10.4 e 10.5 sono riportati il flow-chart descrittivo del problema proposto e la relativa codifica (osserviamo che questo programma prevede delle segnalazioni di errore tramite DISPLAY per tutte le situazioni anomale).

Fig. 10.4a

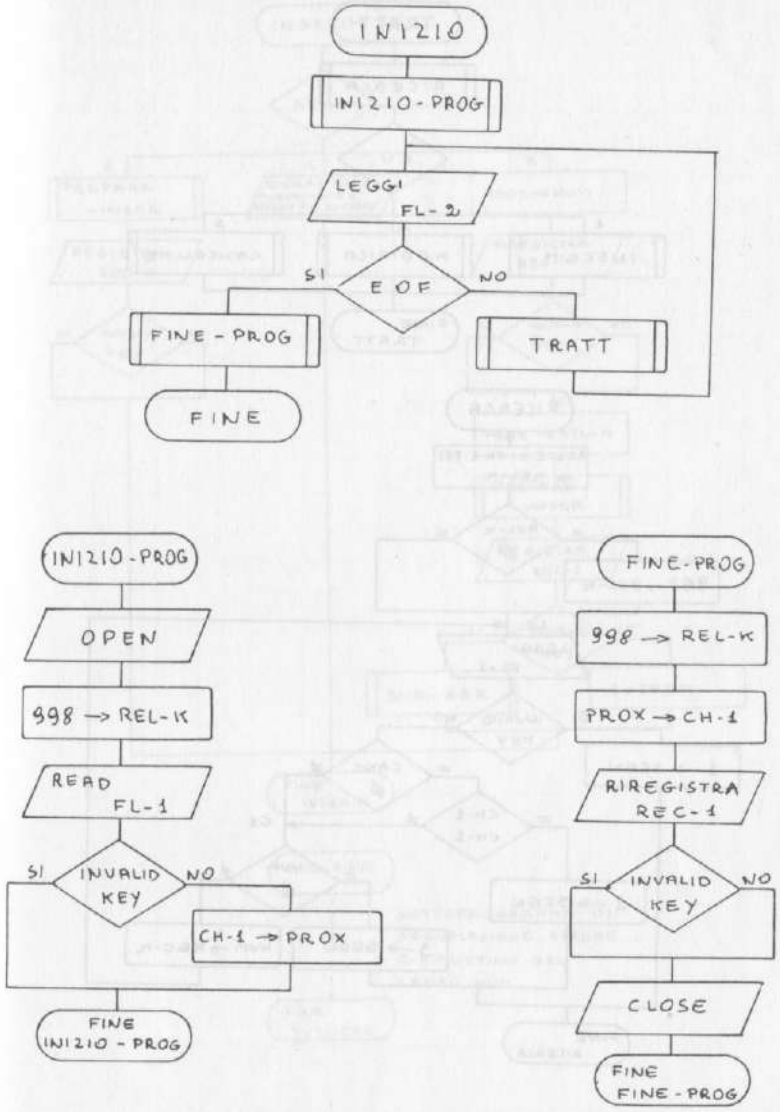


Fig. 10.4b

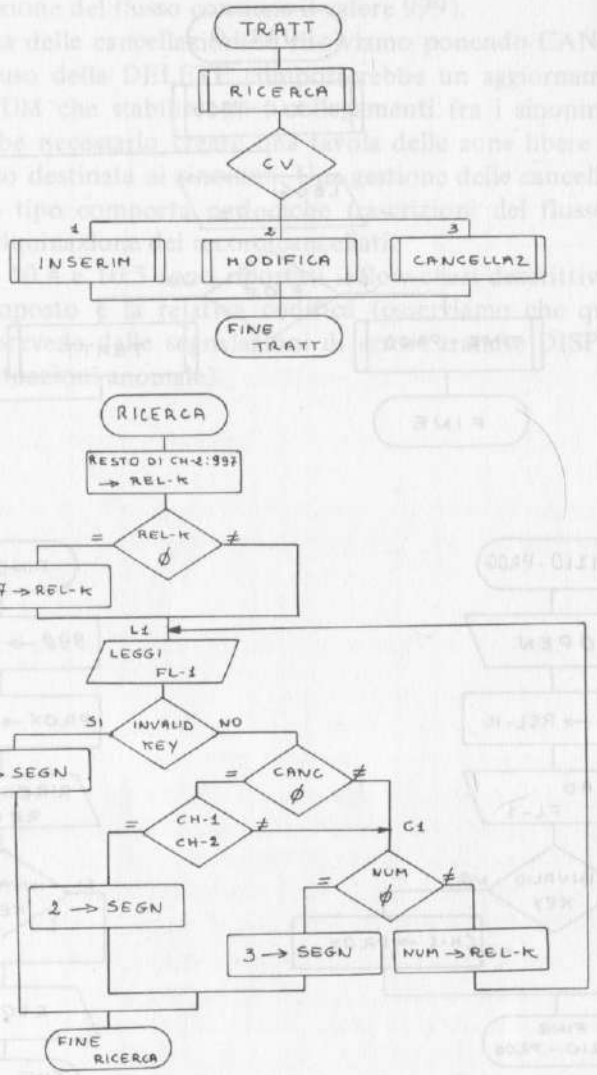


Fig. 10.4c

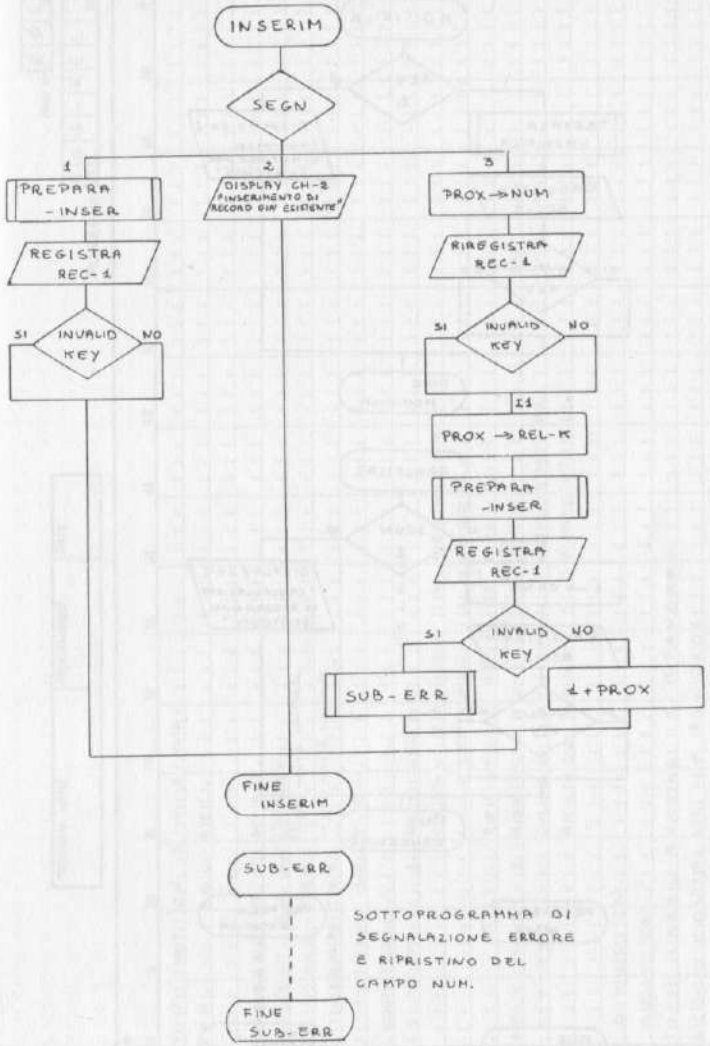


Fig. 10.4d

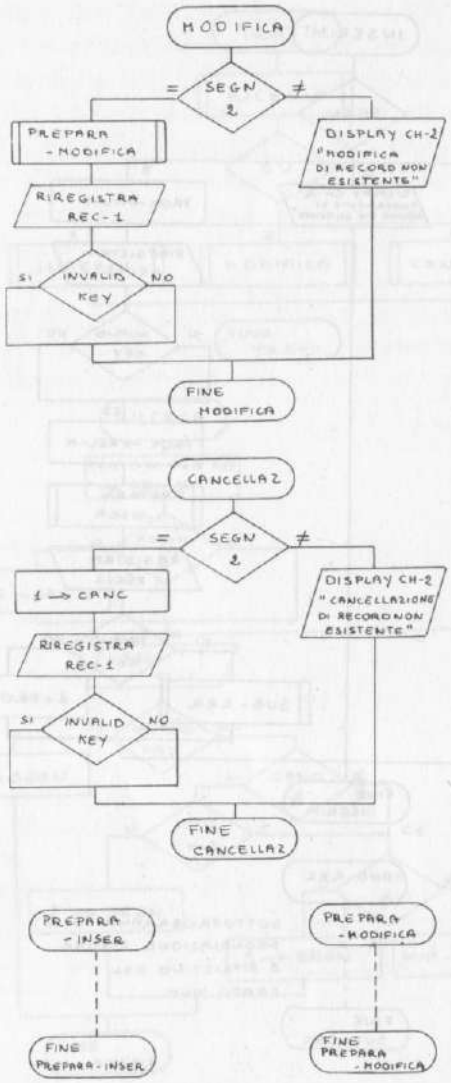


Fig. 10.5a

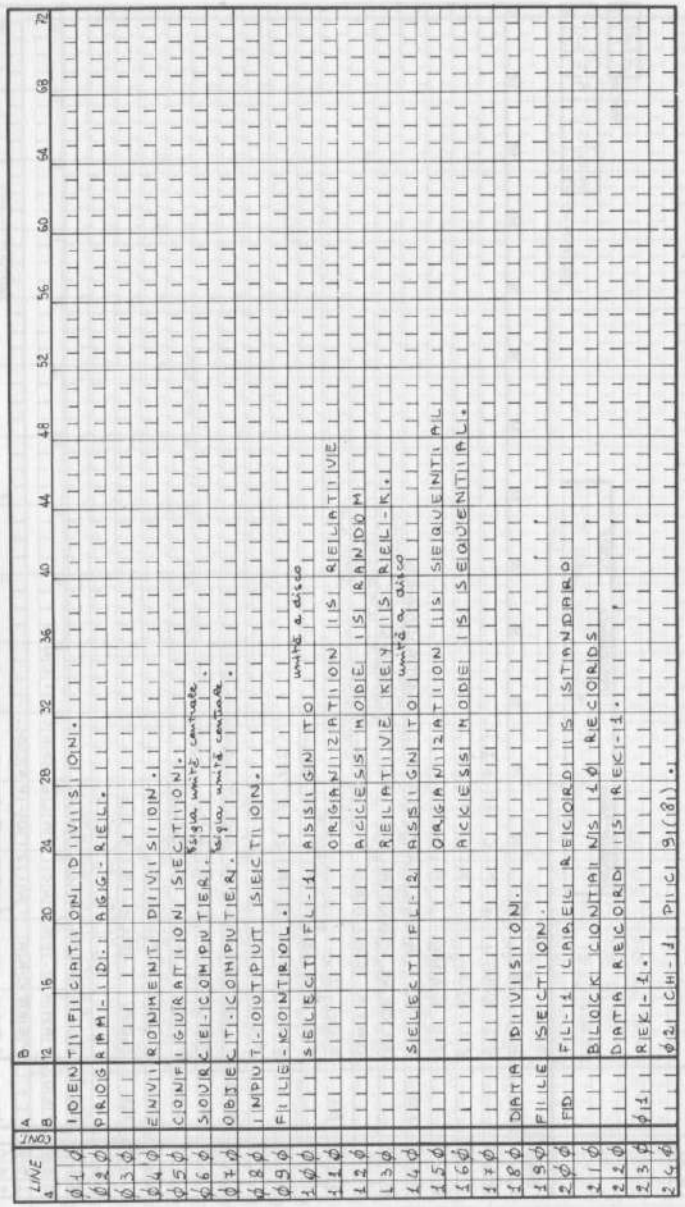


Fig. 10.5b

PROGRAM NAME	PROGRAMMER	DATE

PAGE Nº 002

PROGRAM-ID AGG-REL

```
LINE  CONT.  A
4     8      B
                12        16        20        24        28        32        36        40        44        48        52        56        60        64        68        72
010          |  |  |  |02| DATII-1| PI| CI IX(621).|  |  |  |
```

Fig. 10.5d

PAGE No. 004

PROGRAM NAME | PROGRAMMER | DATE

PROGRAM-ID AGG-REL

LINE	CONT.	A B	B 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72
010		RICE	RCA SECTION.
020			DIVIDE 9871 INTO CH-2 GIVING DEP
030			REMAINDER RELI-K
040			IF RELI-K =ZERO MOVE 9917 TO RELI-K.
050		L1.	READ FILE-1 INVALID KEY MOVE 1 TO SEGN
060			GO TO EX-RICERCA.
070			IF CANC NOT =ZERO GO TO C1.
080			IF CH-1 NOT =CH-2 GO TO C1.
090			MOVE 2 TO SEGN
100			GO TO EX-RICERCA.
110		C1.	IF NUM NOT = ZERO MOVE NUM TO RELI-K
120			GO TO L1.
130			MOVE 3 TO SEGN.
140		EX-R	KERCA. EXIT.
150			
160		INSE	RIM SECTION.
170			GO TO S1 S2 S3 DEPENDING SEGM.
180		S1.	PERFORM PREPARA-INSER
190			WRITE RECI-1 INVALID KEY GO TO EX-INSERIM.
200			GO TO EX-INSERIM.
210		S2.	DISPLAY CH-2 " 3 NSERIMENTO DI RECORD GIA' ESISTENTE"
220			GO TO EX-INSERIM.
230		S3.	MOVE PRIOX TO NUM
240			REWRITE RECI-1 INVALID KEY GO TO EX-M

Fig. 10.5e

PAGE No. 005

PROGRAM NAME | PROGRAMMER | DATE

PROGRAM-ID AGG-REL

LINE	CONT.	A B	B 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72
010		I1.	MOVE PRIOX TO RELI-K
020			PERFORM PREPARA-INSER
030			WRITE RECI-1 INVALID KEY
040			PERFORM SUB-ERRI THRU EX-SUB-ERRI
050			GO TO EX-INSERIM.
060			ADD 1 TO PRIOX.
070		EX-I	NSERIM. EXIT.
080			
090		MODI	FICA SELECTION.
100			IF SEGN NOT =2
110			DISPLAY CH-2 " MODIFICA DI RECORD INON ESISTENTE"
120			ELSE PERFORM PREPARA-MODIFICA
130			REWRITE RECI-1 INVALID KEY GO TO EX-MODIFICA.
140		EX-M	ODIFICA. EXIT.
150			
160		CANC	ELLAZIONE.
170			IF SEGN NOT =2
180			DISPLAY CH-2 " CANCELLAZIONE DI RECORD NON ESISTENTE"
190			ELSE MOVE 1 TO CANC
200			REWRITE RECI-1 INVALID KEY GO TO EX-CANCELLAZ.
210		EX-C	ANCELLAZ. EXIT.
220			
230	*	LES	ECTION: PREPARA-INSER: PREPARA-MODIFICA E SUB-ERRI NON SONO
240	*	SVIL	UPPATE

10.6. Riepilogo

In questo paragrafo vogliamo brevemente schematizzare le possibilità offerte dall'organizzazione "relative" in relazione ai vari tipi di accessi.

Tutte le volte che compaiono la frase AT END o la INVALID KEY questa può essere sostituita da una opportuna procedura di USE.

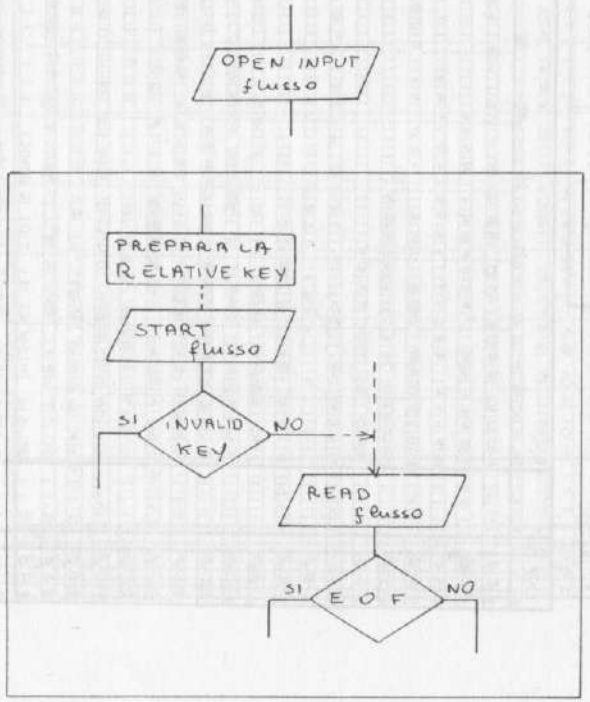
Tenere presente la possibilità delle opzioni FROM ed INTO che non sono esaminate in questa sede.

10.6.1. *Accesso sequenziale*

a. OPEN INPUT

In fig. 10.6 sono schematizzate le operazioni che possono essere

Fig. 10.6



eseguite su un flusso organizzato in modo "relative" trattato in accesso sequenziale con una OPEN INPUT.

Come si può osservare, oltre alle operazioni di OPEN INPUT e CLOSE, può essere utilizzata la READ... AT END... preceduta oppure no da una operazione di posizionamento (nel caso in cui la START sia presente la READ seguente può essere fatta con risultati attendibili solo se il posizionamento ha avuto esito positivo.

b. OPEN OUTPUT

Come si può vedere dall'esame di fig. 10.7 l'unica operazione che si può fare con questo tipo di OPEN è la WRITE.

c. OPEN I-O

Nel caso di OPEN I-O per flussi con organizzazione "relative" ed accesso sequenziale sono possibili molte operazioni ma con vincoli ben precisi (fig. 10.8).

L'operazione di lettura può essere fatta preceduta o meno da una START (nel caso sia inserita la START il posizionamento deve avere esito positivo; per quanto riguarda la REWRITE e la DELETE queste operano sull'ultimo record letto e pertanto devono essere precedute da un'operazione di lettura eseguita con esito positivo.

Fig. 10.7

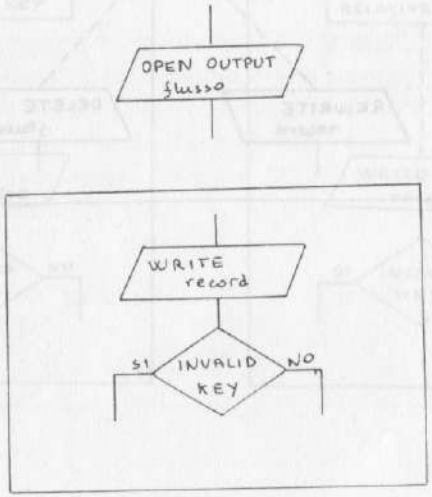
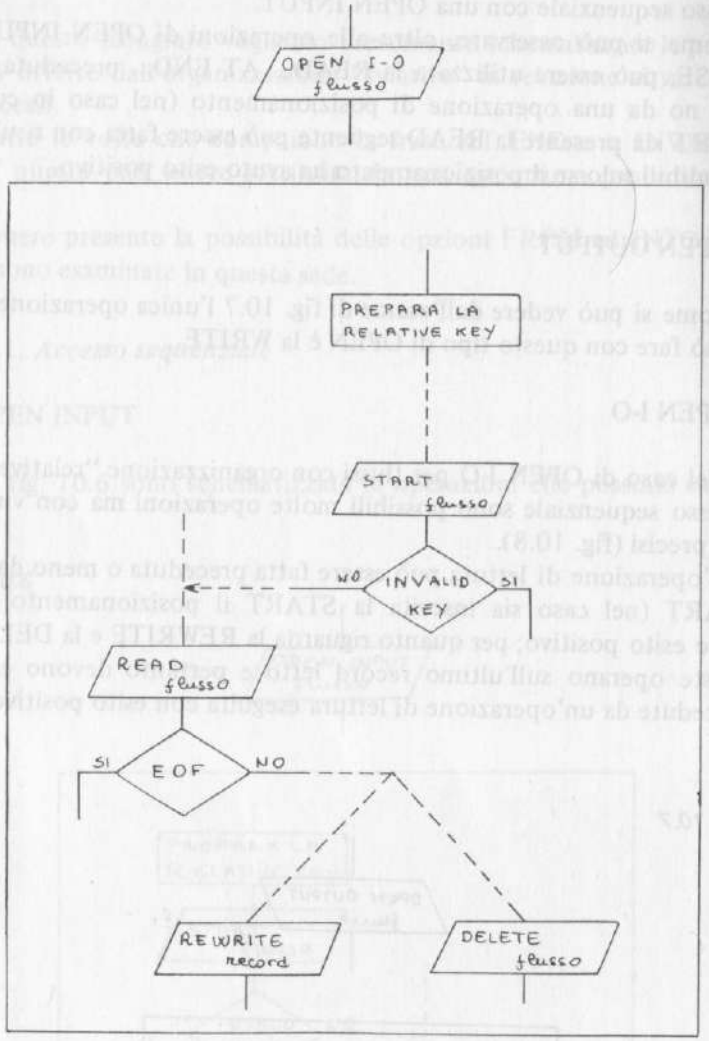


Fig. 10.8



10.6.2. *Accesso random*

a. OPEN INPUT

La fig. 10.9 mostra che l'unica operazione possibile per un flusso con organizzazione "relative" e accesso random trattato con una OPEN INPUT è la lettura in accesso diretto previo posizionamento ad un valore opportuno della RELATIVE KEY.

b. OPEN OUTPUT

Utilizzando questo tipo di OPEN, per flussi "relative" con accesso random, l'unica operazione possibile è la registrazione in accesso diretto, che viene eseguita previo controllo della disponibilità dell' area indirizzata dal valore della RELATIVE KEY (fig. 10.10).

Fig. 10.9

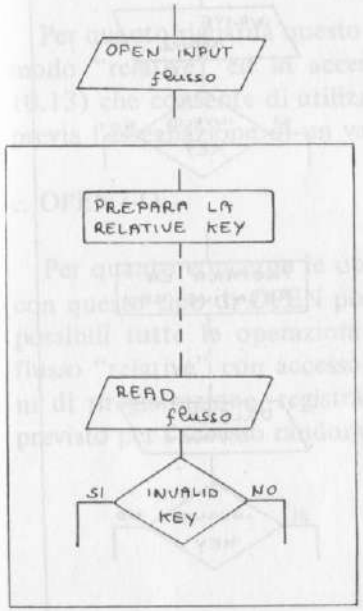
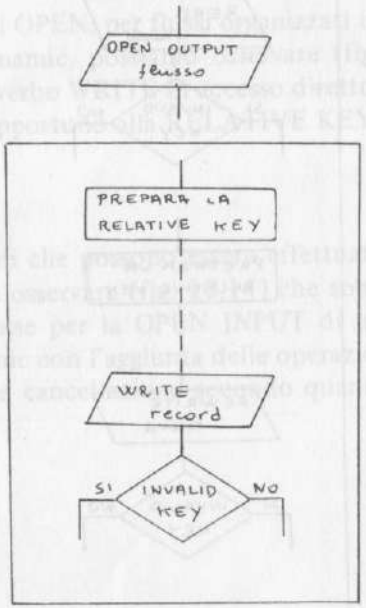


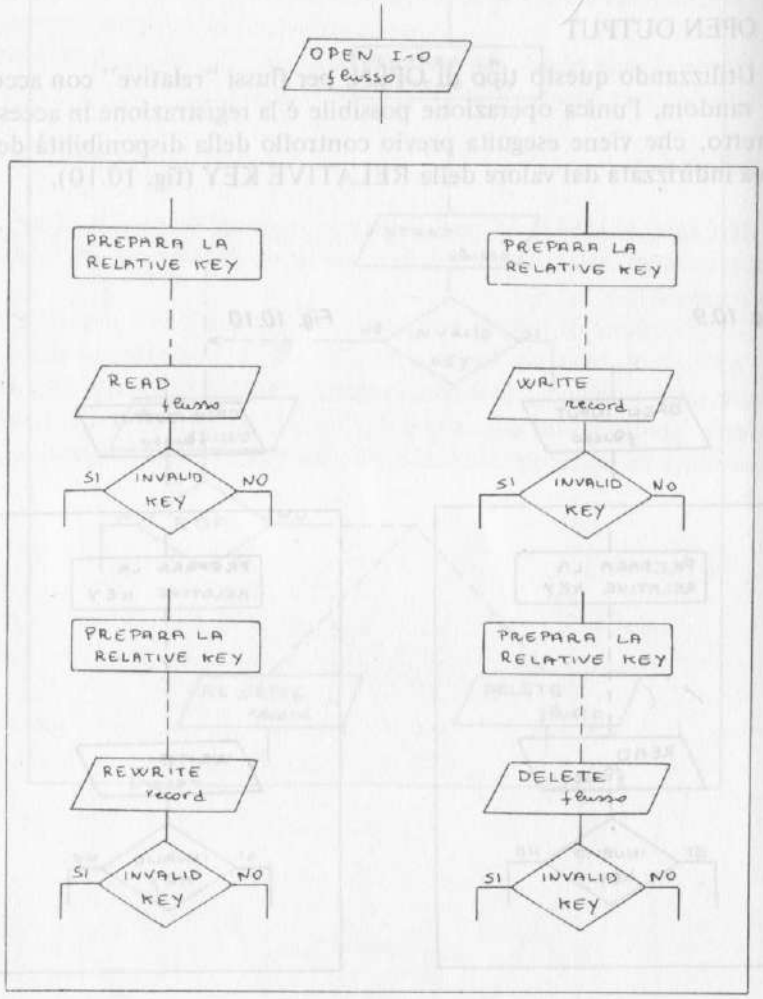
Fig. 10.10



c. OPEN I-O

Come mostra la fig. 10.11 utilizzando questo tipo di OPEN sono possibili le operazioni di lettura, registrazione, riregistrazione e cancellazione; tali operazioni vengono eseguite in modo diretto, deve cioè essere preparata opportunamente la RELATIVE KEY.

Fig. 10.11



10.6.3. *Accesso dynamic*

a. OPEN INPUT

Da fig. 10.12 si può rilevare che la READ random può essere comunque fatta dopo aver preparato opportunamente la RELATIVE KEY ed in particolare se l'operazione di lettura ha esito positivo può servire come posizionamento per delle READ di tipo sequenziali (a partire dal prossimo record registrato).

Oltre che con la tecnica esposta precedentemente, per effettuare dei posizionamenti all'interno del flusso, per poi leggere sequenzialmente, si può utilizzare la START: in questo caso il primo record letto tramite una READ NEXT è proprio quello puntato.

L'operazione READ NEXT dopo un posizionamento iniziale può essere iterata senza ripetere posizionamenti (ricordiamo che la OPEN INPUT opera automaticamente un posizionamento sul primo record valido).

b. OPEN OUTPUT

Per quanto riguarda questo tipo di OPEN, per flussi organizzati in modo "relative" ed in accesso dynamic, possiamo osservare (fig. 10.13) che consente di utilizzare il verbo WRITE in accesso diretto, previa l'assegnazione di un valore opportuno alla RELATIVE KEY.

c. OPEN I-O

Per quanto concerne le operazioni che possono essere effettuate con questo tipo di OPEN possiamo osservare (fig. 10.14) che sono possibili tutte le operazioni ammesse per la OPEN INPUT di un flusso "relative" con accesso dynamic con l'aggiunta delle operazioni di riregistrazione, registrazione e cancellazione secondo quanto previsto per l'accesso random.

Fig. 10.12

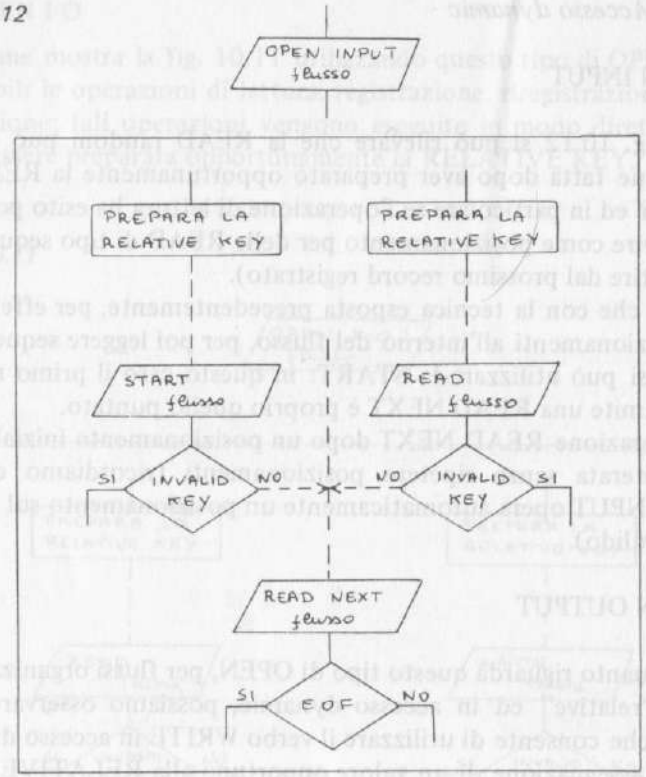


Fig. 10.13

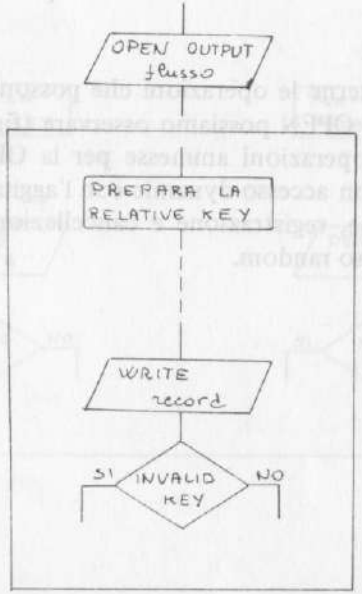
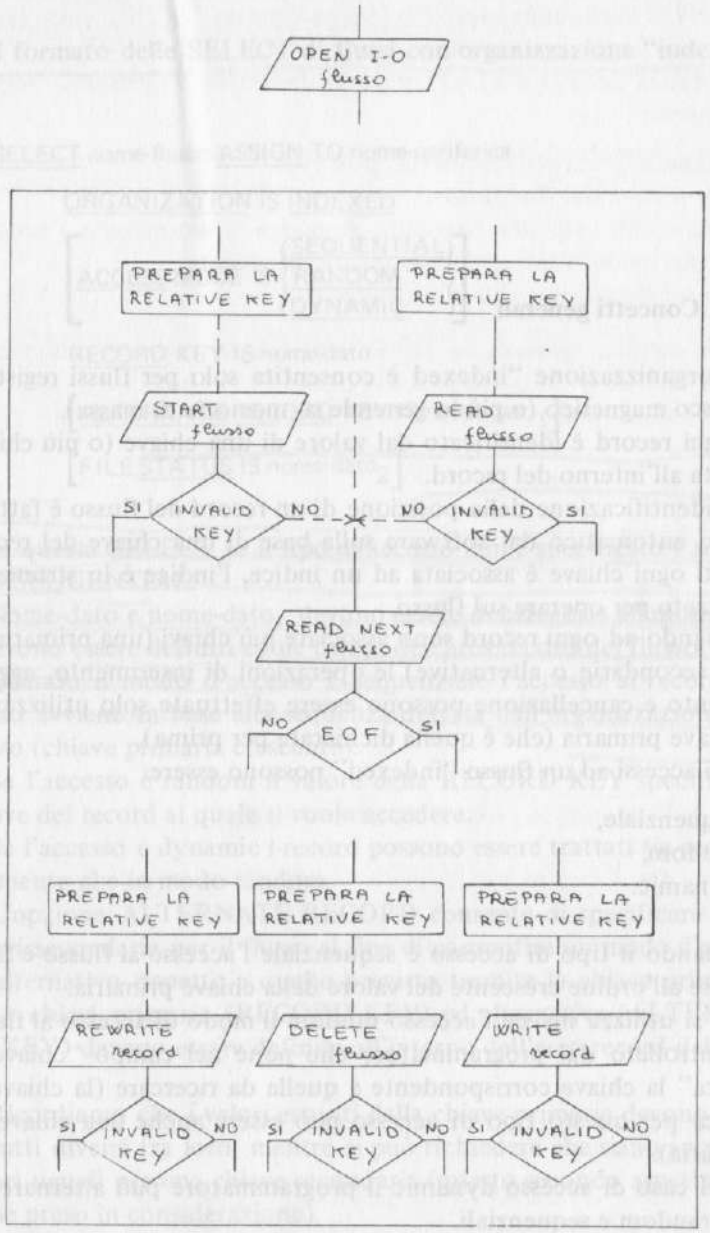


Fig. 10.14



11. ORGANIZZAZIONE "INDEXED"

11.1. Concetti generali

L'organizzazione "indexed è consentita solo per flussi registrati su disco magnetico (o più in generale su memorie di massa).

Ogni record è identificato dal valore di una chiave (o più chiavi) situata all'interno del record.

L'identificazione della posizione di un record del flusso è fatta in modo automatico dal software sulla base di una chiave del record, infatti ogni chiave è associata ad un indice, l'indice è lo strumento utilizzato per operare sul flusso.

Quando ad ogni record sono associate più chiavi (una primaria ed altre secondarie o alternative) le operazioni di inserimento, aggiornamento e cancellazione possono essere effettuate solo utilizzando la chiave primaria (che è quella dichiarata per prima).

Gli accessi ad un flusso "indexed" possono essere:

a. sequenziale,
b. random,
c. dynamic.

Quando il tipo di accesso è sequenziale l'accesso al flusso è fatto in base all'ordine crescente del valore della chiave primaria.

Se si utilizza invece l'accesso random il modo di accesso al flusso è controllato dal programmatore che pone nel campo "chiave di ricerca" la chiave corrispondente a quella da ricercare (la chiave di ricerca per questo tipo di accesso può essere anche una chiave secondaria).

Nel caso di accesso dynamic il programmatore può alternare accessi random e sequenziali.

11.2. Riferimenti in ENVIRONMENT DIVISION

Il formato delle SELECT di flussi con organizzazione "indexed" è:

```
SELECT nome-flusso ASSIGN TO nome-periferica
    ORGANIZATION IS INDEXED
    [ACCESS MODE IS { SEQUENTIAL | RANDOM | DYNAMIC }]
    RECORD KEY IS nome-dato
    [ALTERNATE RECORD KEY IS nome-dato1] ...
    [FILE STATUS IS nome-dato2]
```

In questa SELECT se il tipo di accesso non è specificato è sottinteso SEQUENTIAL.

Nome-dato e nome-dato$_1$ devono essere di categoria alfanumerica e devono essere definiti come campi dell'area record del flusso.

Quando il modo d'accesso è sequenziale l'accesso ai record del flusso avviene in base alla sequenza dettata dall'organizzazione del flusso (chiave primaria crescente).

Se l'accesso è random il valore della RECORD KEY specifica la chiave del record al quale si vuole accedere.

Se l'accesso è dynamic i record possono essere trattati sia sequenzialmente che in modo random.

L'opzione ALTERNATE RECORD consente di specificare delle chiavi secondarie per il flusso al fine di consentire un modo d'accesso alternativo rispetto a quello previsto tramite la chiave primaria.

Le chiavi primaria (RECORD KEY) ed alternative (ALTERNATE KEY) devono essere definite all'interno dell'area record del flusso.

Ricordiamo che i valori assunti dalla chiave primaria devono essere tutti diversi tra loro, mentre si può richiedere che siano accettati valori uguali per una chiave secondaria (questo secondo aspetto non viene preso in considerazione).

Il campo nome-dato$_2$ (FILE STATUS), che deve essere definito

in DATA DIVISION, ma non in FILE SECTION o in REPORT SECTION (vedi appendice B.), deve essere un "X(2)" e viene aggiornato dopo ogni operazione di input/output.

L'analisi del FILE STATUS consente di rilevare una delle seguenti situazioni:

a. operazione eseguita con esito positivo,
b. rilevazione del fine flusso,
c. operazione eseguita con esito negativo (esamineremo i possibili eventi quando tratteremo le singole istruzioni),
d. errore permamente di input/output.

Per quanto riguarda la I-O-CONTROL osserviamo che per un flusso "indexed" valgono le stesse possibilità e modalità di specificazione che avevamo enunciato per flussi organizzati in modo "relative".

11.3 Riferimenti in DATA DIVISION

Le modalità di scrittura delle FD e delle descrizioni record di DATA DIVISION restano invariate, bisogna solamente ricordare che devono fare parte delle descrizioni dei record anche le aree citate in RECORD KEY ed in ALTERNATE RECORD.

11.4. Riferimenti in PROCEDURE DIVISION

Per i flussi organizzati in modo "indexed" sono possibili gli stessi tipi di accesso che abbiamo già analizzato per i flussi organizzati in modo "relative": il tipo di OPEN consente di determinare il modo in cui deve avvenire l'accesso al flusso e di conseguenza il tipo di istruzioni che possono essere utilizzate e le loro caratteristiche operative.

11.4.1. *Il verbo OPEN*

Il formato del verbo OPEN per flussi organizzati in modo "indexed" non differisce da quello che abbiamo presentato per flussi con organizzazione "relative":


```
        INPUT nome-flusso1 [nome-flusso2]...
OPEN {  OUTPUT nome-flusso3 [nome-flusso4]... }
        I-O nome-flusso5 [nome-flusso6]...
```

In fig. 11.1 è riportata la tabella dei possibili trattamenti in relazione alla OPEN utilizzata.

Fig. 11.1

TIPI DI ACCESSO	ISTRUZIONI	TIPI DI OPEN		
		INPUT	OUTPUT	I-O
SEQUENTIAL	READ	X		X
	WRITE		X	
	REWRITE			X
	START	X		X
	DELETE			X
RANDOM	READ	X		X
	WRITE		X	X
	REWRITE			X
	START			
	DELETE			X
DYNAMIC	READ	X		X
	WRITE		X	X
	REWRITE			X
	START	X		X
	DELETE			X

11.4.2. *Il verbo CLOSE*

Anche il formato previsto per il verbo CLOSE non differisce da quello utilizzato nel caso di un flusso con organizzazione "relative":

CLOSE nome-flusso$_1$ [WITH LOCK]
[nome-flusso$_2$ [WITH LOCK]] ...

I flussi, i cui nomi sono citati nella CLOSE, non occorre che abbiano lo stesso tipo di organizzazione: perchè possa essere eseguita questa istruzione è sufficiente che i flussi siano stati trattati con una OPEN.

L'opzione LOCK consente di impedire che il flusso possa essere oggetto di una successiva OPEN durante l'esecuzione della stessa unità di elaborazione.

11.4.3. *Il verbo READ*

L'istruzione di lettura offre le stesse possibilità esaminate nel caso dell'organizzazione "relative" con la differenza che nelle considerazioni dell'organizzazione "indexed" interviene la chiave del record piuttosto che il numero di record ed inoltre, nell'accesso random la ricerca può essere effettuata in base al valore di una chiave secondaria.

Nel caso di accesso sequenziale una istruzione di lettura rende disponibile in area record il "prossimo" record logico, mentre nel caso di accesso random rende disponibile in area record "un record la cui chiave è stata specificata"; per quanto riguarda l'accesso dynamic è sufficiente tenere presente che intervengono alternativamente accessi sequenziali e random.

I formati della READ per flussi con organizzazione "indexed" sono due:

Formato 1:

READ nome-flusso [NEXT] RECORD
[INTO nome-dato]
[AT END ... .]

Formato 2:

READ nome-flusso RECORD [INTO nome-dato]
[KEY IS nome-dato$_1$]
[INVALID KEY ... .]

La frase INTO, in entrambi i formati, non può essere utilizzata quando il flusso contiene record di lunghezza diversa.

Il formato 1 deve essere usato (senza la parola NEXT) tutte le volte che l'accesso è sequenziale.

Il formato 1, con la parola NEXT, è usato per lettura sequenziale quando l'accesso è dynamic.

Il formato 2 è utilizzato per flussi a cui si accede in modo random oppure per ricerca random in flussi con accesso dynamic.

Le frasi INVALID KEY e AT END sono obbligatorie a meno che non sia specificata una opportuna USE (nelle DECLARATIVES) per i relativi trattamenti.

Il verbo READ può essere utilizzato solo per flussi preventivamente trattati con una OPEN INPUT oppure con una OPEN I-O.

Per quanto riguarda il formato 1 possiamo affermare che:

a. Viene reso disponibile il "prossimo" record logico, nel senso di "quello con chiave immediatamente superiore" all'ultimo record trattato, naturalmente è inteso che:
   - per flussi con accesso sequenziale il primo record da trattare dopo la OPEN è il record di chiave minore (la sua posizione è rilevata dall'indice),
   - per flussi con accesso dynamic il primo record da trattare è

quello a cui si "punta" dopo l'esecuzione con esito positivo di una START (nei formati che vedremo più avanti);
- i record eventualmente presenti ma cancellati logicamente vengono ignorati automaticamente: la READ consente quindi di analizzare solo i record logicamente presenti sul flusso;
- per quanto riguarda gli ordini di lettura successivi al primo, il record reso disponibile è il "prossimo record logico", a meno che, per l'accesso dynamic, non sia stata fatta nuovamente una START;
- se al momento dell'esecuzione dell'istruzione di lettura non è definito quale sia il record da rendere disponibile, allora l'esecuzione della operazione non può avvenire correttamente;

b. quando viene incontrato l'identificativo di fine flusso (e vengono eseguite le operazioni specificate dopo la frase AT END o nella USE relativa) è possibile dare una nuova istruzione di lettura solo in questi casi:
- per accesso sequenziale, dopo l'esecuzione di una CLOSE e di una successiva OPEN che abiliti alla lettura in sequenziale;
- per accesso dynamic, dopo l'esecuzione con esito positivo di una nuova START oppure di una READ di formato 2.

Per quanto riguarda il formato 2, se non è stata specificata l'opzione KEY, è stabilito che la chiave di ricerca sia quella dichiarata in RECORD KEY sia nel caso di accesso random che dynamic e tale chiave per l'accesso dynamic è utilizzata anche per le esecuzioni seguenti di READ di formato 1 finché non venga stabilita una diversa chiave di ricerca.

Se nel formato 2 è presente l'opzione KEY la chiave specificata viene utilizzata come chiave di ricerca sia per accesso random che dynamic ed in particolare, per quest'ultimo, tale chiave viene utilizzata come chiave di ricerca per ogni esecuzione seguente di READ di formato 1 finché non venga stabilita una diversa chiave di ricerca.

L'esecuzione di una READ di formato 2 fornisce il record di chiave uguale a quella richiesta (che deve essere preparata nell'opportuno campo dell'area record) oppure, per non trovato, provoca

l'esecuzione delle istruzioni indicate dopo la frase INVALID KEY o nella procedura di trattamento della USE relativa.

11.4.4. *Il verbo WRITE*

Il verbo WRITEconsente di registrare un record logico su un flusso aperto in output oppure in input/output: la registrazione, come al solito, può interessare direttamente la periferica in output oppure semplicemente opera un trasferimento dall'area record all'area blocco del flusso.

Il formato di questo verbo è:

```
WRITE nome-record [FROM nome-dato]

[INVALID KEY ... .]
```

Quando si utilizza l'opzione FROM bisogna tenere presente che nome-dato non può essere un identificativo dell'area record o di parte di essa e che, al momento dell'esecuzione della WRITE si ha un trasferimento del contenuto di nome-dato in area record e quindi avviene la fase vera e propria di registrazione (nel senso visto or ora).

Il valore della chiave primaria di un record deve essere diverso per i vari record del flusso.

Prima dell'esecuzione della WRITE il valore dell'area stabilita come chiave primaria deve essere caricato da programma.

Se l'accesso è sequenziale i record in registrazione devono essere trattati per chiavi crescenti, se l'accesso è sequenziale o dynamic i record possono essere registrati in un ordine qualunque.

Durante l'esecuzione di una WRITE si può avere l'INVALID KEY in concomitanza dei seguenti eventi:

a. quando l'accesso è sequenziale (flusso aperto in output) ed il valore della chiave primaria non è maggiore del valore della chiave primaria del record precedentemente registrato;
b. quando il flusso è aperto in input/output ed il valore della chiave

primaria è uguale al valore della chiave primaria di un altro record già registrato;

c. quando lo spazio disponibile per la registrazione è completamente esaurito.

11.4.5. *Il verbo REWRITE*

Questo verbo consente di sostituire un record esistente su un flusso (registrato su una memoria di massa); a tale scopo si utilizza il formato:

```
REWRITE nome-record [FROM nome-dato]
        [INVALID KEY ... .]
```

Il record che deve sostituire quello esistente può essere preparato in area record oppure in un'area diversa dall'area record.

La frase INVALID KEY può essere omessa solo nel caso in cui si sia indicata nelle DECLARATIVES una USE appropriata.

Per flussi in accesso sequenziale il record interessato alla riregistrazione è l'ultimo record letto.

Per flussi in accesso random o dynamic il record interessato alla riregistrazione è quello con la chiave primaria con lo stesso valore del record che lo deve sostituire.

La frase INVALID KEY (o una corrispondente USE) consentono di indicare le procedure da utilizzare:

a. nel caso di accesso sequenziale qualora il contenuto della chiave primaria del record da inserire sia diverso dal contenuto della chiave primaria dell'ultimo record letto;

b. nel caso in cui non venga trovato sul flusso un record con chiave uguale a quella del record preparato per la sostituzione.

Ricordiamo che il record che viene registrato al posto di uno già esistente deve avere lo stesso numero di caratteri.

11.4.6. *Il verbo START*

Questo verbo consente di effettuare dei posizionamenti all'interno del flusso per poi procedere in modo sequenziale; il formato di questa istruzione è:

```
START nome-flusso [KEY { IS EQUAL TO
                       IS =
                       IS GREATER THAN
                       IS >
                       IS NOT LESS THAN
                       IS NOT <       } nome-dato]
[INVALID KEY ... .]
```

Il flusso, che deve essere trattato in accesso sequenziale o dynamic, deve essere stato aperto con una OPEN INPUT oppure con una OPEN I-O.

Nome-dato, se l'opzione KEY è presente, specifica il nome dell'area dichiarata come RECORD KEY nella SELECT del flusso.

Quando viene indicata la clausola KEY, il suo formato specifica le modalità del posizionamento, infatti questo avviene in corrispondenza della posizione indicata dalla RECORD KEY (=) oppure in corrispondenza del primo record con chiave immediatamente maggiore di quella indicata dalla RECORD KEY (>) oppure in coincidenza del primo record di chiave maggiore o eguale (NOT <) di quella indicata: se la ricerca del record su cui posizionarsi ha esito negativo allora vengono eseguite le istruzioni scritte dopo la frase INVALID KEY oppure, se questa non è indicata, viene eseguita la procedura prevista da una opportuna USE che deve sopperire alla mancanza dell'INVALID KEY.

Se è omessa la clausola KEY è sottinteso che il posizionamento avviene (se possibile) in coincidenza del record di chiave uguale a quella contenuta nella RECORD KEY; di solito la clausola KEY viene utilizzata nel formato "NOT <": ricordiamo che l'uscita per INVALID KEY non da origine ad alcun posizionamento.

11.4.7. *Il verbo DELETE*

Questo verbo ha la funzione di cancellare logicamente un record del flusso; il suo formato è:

DELETE nome-flusso RECORD [INVALID KEY ... .]

La frase INVALID KEY non deve essere specificata quando l'accesso al flusso è sequenziale, infatti il record trattato con una DELETE, per flussi trattati in accesso sequenziale, è l'ultimo record letto.

Per flussi che vengono elaborati in accesso random o dynamic la frase INVALID KEY (o una USE sostitutiva) deve essere indicata, infatti è possibile dare l'ordine di cancellazione senza avere eseguito preventivamente una READ e pertanto il flusso potrebbe non contenere un record valido nella posizione indirzzata sulla base della RECORD KEY.

Dopo l'esecuzione di una DELETE il record trattato risulta logicamente cancellato dal flusso e non può più essere elaborato da programma.

L'esecuzione di una DELETE non altera l'area record del flusso in trattamento.

11.4.8. *Il verbo USE*

Il verbo USE specifica le procedure che devono essere eseguite in concomitanza di errori nel trattamento dell'input/output, sia quando l'errore è di sistema e viene prima trattato dalle routines standard, sia quando è un errore o una situazione particolare rilevata durante l'esecuzione di una istruzione di input/output e sono state omesse le opzioni INVALID KEY oppure AT END.

Il formato della USE è:

USE AFTER STANDARD { EXCEPTION / ERROR }

PROCEDURE ON { nome-flusso$_1$ [nome-flusso$_2$] ... / INPUT / OUTPUT / I-O }

e le caratteristiche (con le opportune limitazioni di formato) sono le stesse viste per i flussi con organizzazione sequenziale (vedi 9.4.6).

11.5. Un esempio

Il programma CARICHI si propone di aggiornare un archivio prodotti (PROD) mediante un flusso di movimenti (MOV).

In fig. 11.2 è riportato l'organigramma di questo problema.

Il flusso MOV è organizzato in modo sequenziale e viene trattato in INPUT: è il flusso guida dell'elaborazione.

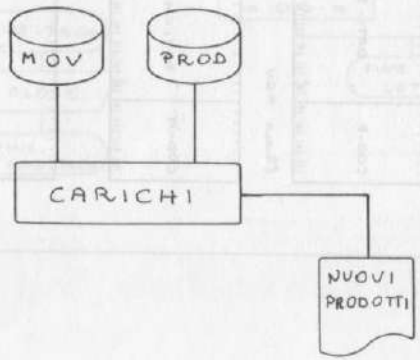
Il flusso PROD è organizzato in modo indexed e viene trattato in accesso random in aggiornamento.

In fig. 11.3 sono schematizzati i tracciati record dei due flussi.

L'elaborazione in quanto questo programma è un pretesto per utilizzare i verbi di input/output, consiste in:

a. leggere in modo sequenziale i record di MOV,
b. se il campo QTA contiene il valore zero deve essere eliminato da PROD il record con COD-P uguale COD-M (segnalarne l'eventuale assenza),
c. se il campo QTA contiene un valore diverso da zero ed in archivio è presente un record con COD-P uguale a COD-M aggiornarlo nel campo GIAC sommandovi QTA,
d. se il campo QTA contiene un valore diverso da zero ed in archivio non è presente un record con COD-P uguale a COD-M bisogna

Fig. 11.2



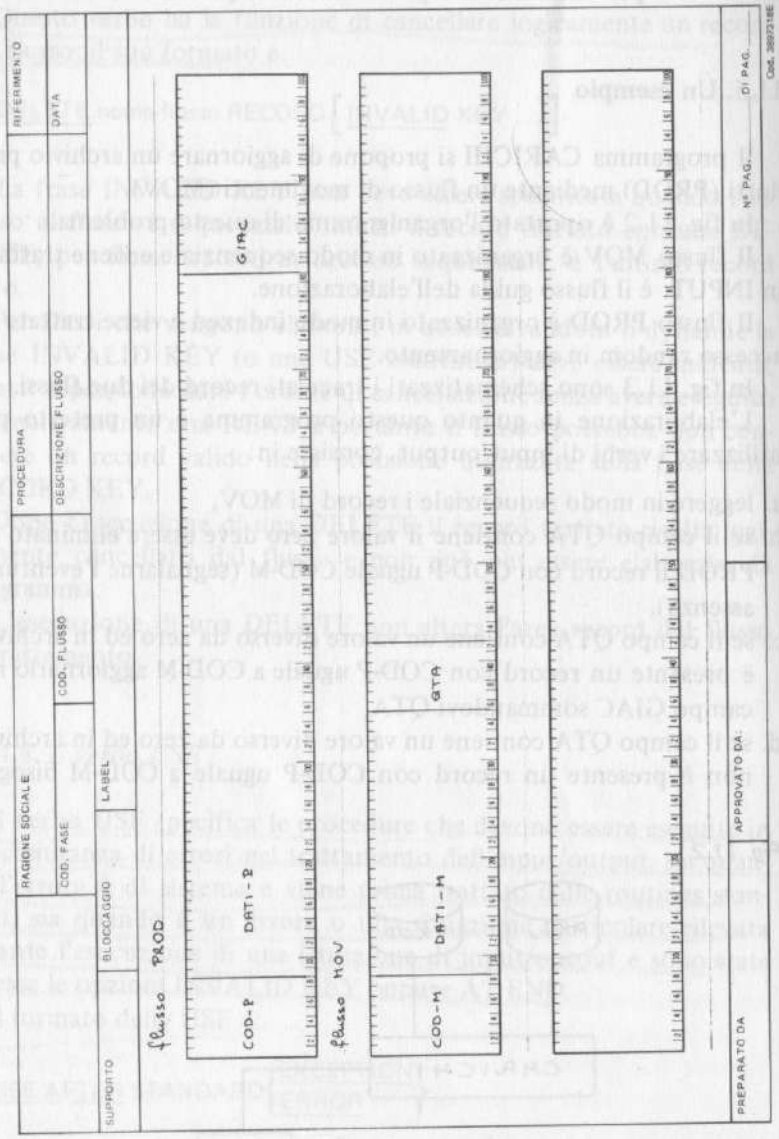


Fig. 11.3

inserire un nuovo record archivio con GIAC uguale a QTA e DATI-P a spazio ed inoltre bisogna segnalare l'inserimento.

Nelle figg. 11.4 ed 11.5 sono riportati rispettivamente il flowchart e la codifica dell'esempio proposto.

Fig. 11.4a

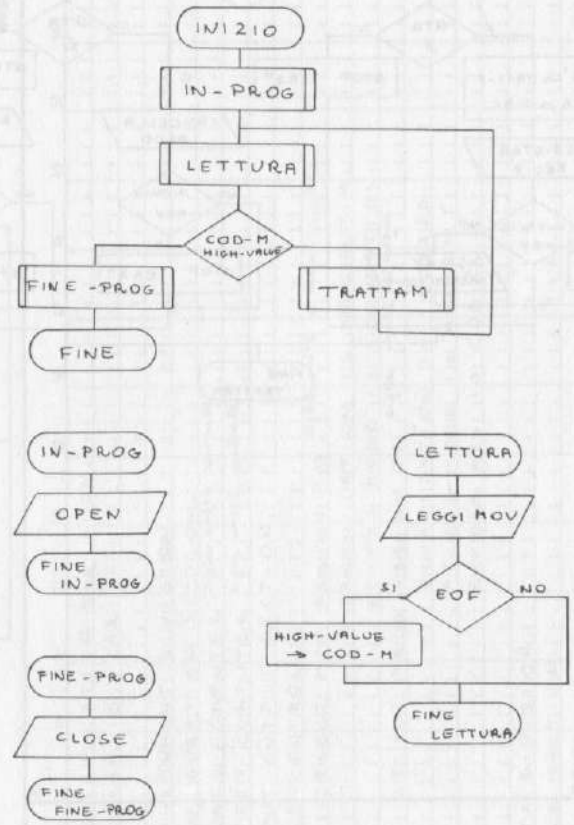


Fig. 11.4b

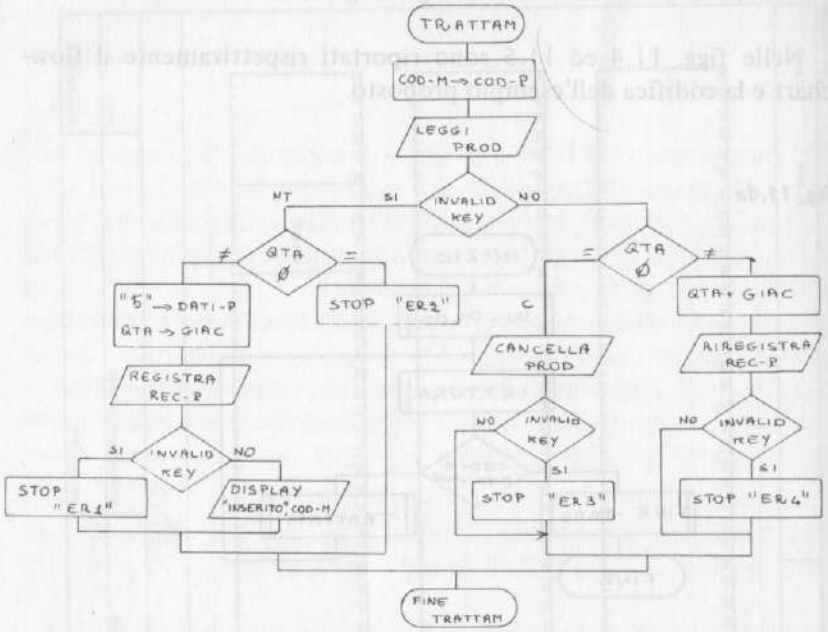


Fig. 11.5a

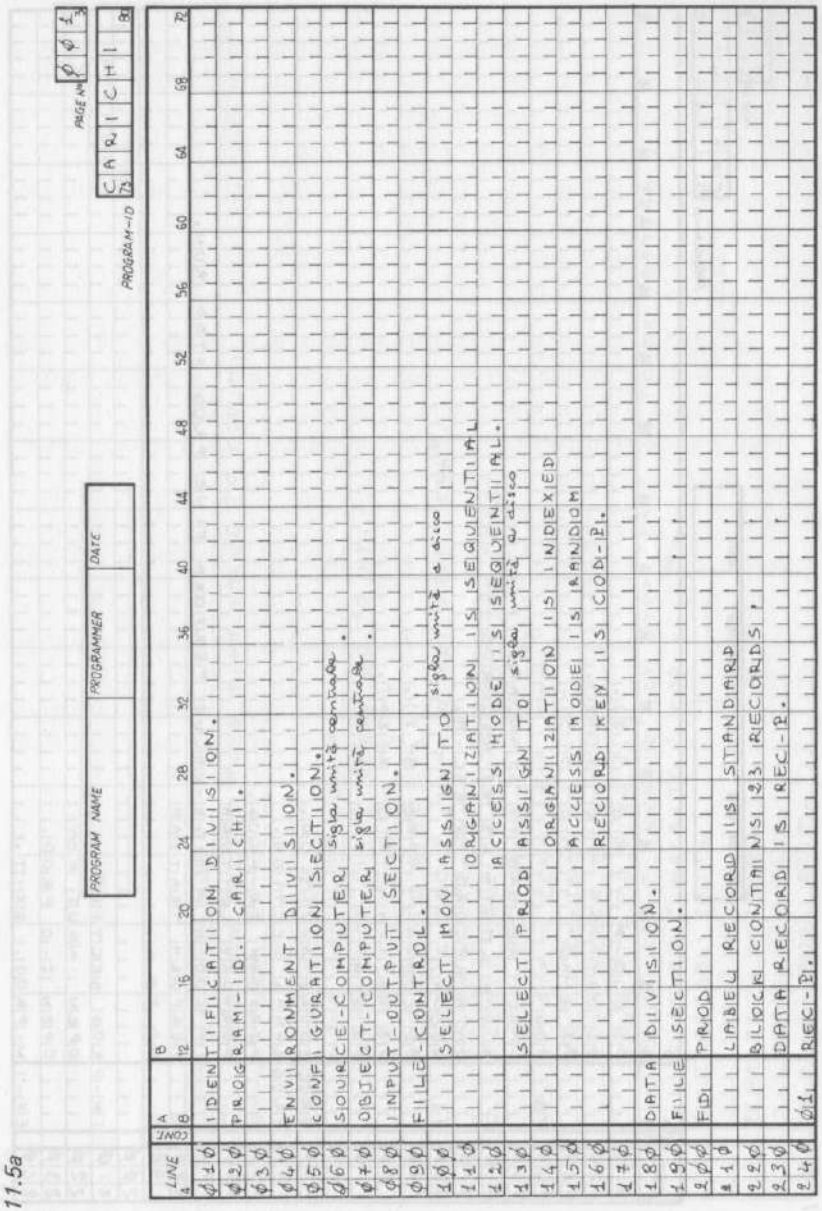


Fig. 11.5b

PROGRAM NAME	PROGRAMMER	DATE

PAGE No. 003

PROGRAM-ID CARICHI

```
LINE  CONT.  A
4     B      B
            12        16        20        24        28        32        36        40        44        48        52        56        60        64        68        72
010          012 CODI-P    PICI      9(18)1.
020          012 DATII-P   PICI      X(721).
030          012 GIIACI    PICI      9(19)1.
040     FDI  MOVI
050          CIBEL RECORD IS STANDARD
060          BLOCK CONTAINS 18 RECORDS
070          DATA RECORDS IS RECI-M.
080     013  RECI-M.
090          012 CODI-M    PICI      9(18)1.
100          012 DATII-M   PICI      X(271).
110          012 QTA       PICI      9(7)1.
120
130     PROC EDURE DIVISION.
140     PROGRAMMA SECTION.
150          PERFORM INI-PROG.
160     L.   PERFORM LETTURA
170          IF CODI-M = HIGH-VALUE PERFORM FINE-PROG STOP RUN.
180          PERFORM TRATTAM
190          GO TO L.
200
210     INI-PROG SECTION.
220          OPEN INPUT MOV
230          OPEN I-O PROD.
240     EXI-I NI-PROG. EXIT.
```

Fig. 11.5c

PROGRAM NAME	PROGRAMMER	DATE

PAGE No. 003

PROGRAM-ID CARICHI

```
LINE  CONT.  A
4     B      B
            12        16        20        24        28        32        36        40        44        48        52        56        60        64        68        72
            LETTURA SECTION.
            READ MOV AT END MOVE HIGH-VALUE TO CODI-M.
     EXI-L  ETTURA. EXIT.

            TRATTAM SECTION.
            MOVE CODI-M TO CODI-P
            READ PROD INVALID KEY GO TO NT.
            IF QTA = ZERO GO TO CI.
            ADD QTA TO GIAC
            REWRITE RECI-P INVALID KEY STOP "ERR4".
            GO TO EXI-TRATTAM.
     CI.    DELETE PROD INVALID KEY STOP "ERR3".
            GO TO EXI-TRATTAM.
     NT.    IF QTA = ZERO STOP "ERR2"
            GO TO EXI-TRATTAM.
            MOVE SPACE TO DATII-P
            MOVE QTA TO GIAC
            WRITE RECI-P INVALID KEY STOP "ERR1" GO TO EXI-TRATTAM.
            DISPLAY "INSERITO" CODI-M.
     EXI-T  RATTAM. EXIT.

     FINE   -PROG SECTION.
            CLOSE MOV PROD.
     EXI-F  INE-PROG. EXIT.
```

11.6. Un altro esempio

Il programma che presentiamo ora esemplifica un caso di lettura fra limiti: in fig. 11.6 è rappresentato l'organigramma del problema.

Il flusso ARK-ART, organizzato INDEXED, è costituito da record (REC) che ai fini di questa elaborazione contengono:

a. la chiave (CHIAVE) sottodefinita nei campi settore merceologico (SETM) e progressivo (PROGR),
b. il prezzo unitario (PRU),
c. la giacenza.

I record del flusso guida (GUIDA) su schede (TSK = "L") contengono i valori di due settori merceologici SET1 e SET2, il primo minore o uguale del secondo (controllarlo).

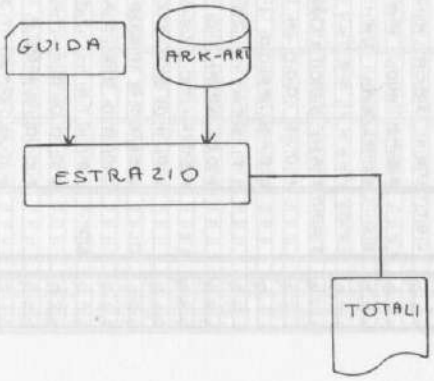
L'elaborazione consiste nella stampa di una riga per ogni record valido del flusso guida, precisamente:

a. i valori di SET1 e SET2,
b. la valorizzazione di tutti gli articoli compresi fra il primo articolo del settore SET1 e l'ultimo di SET2.

Per quanto riguarda la stampa non ci preoccuperemo di problemi di impaginazione.

In fig. 11.7 sono descritti i tracciati dei record dei due flussi in input, nelle figg. 10.8 e 10.9 sono riportati rispettivamente il flowchart e la codifica del problema proposto.

Fig. 11.6



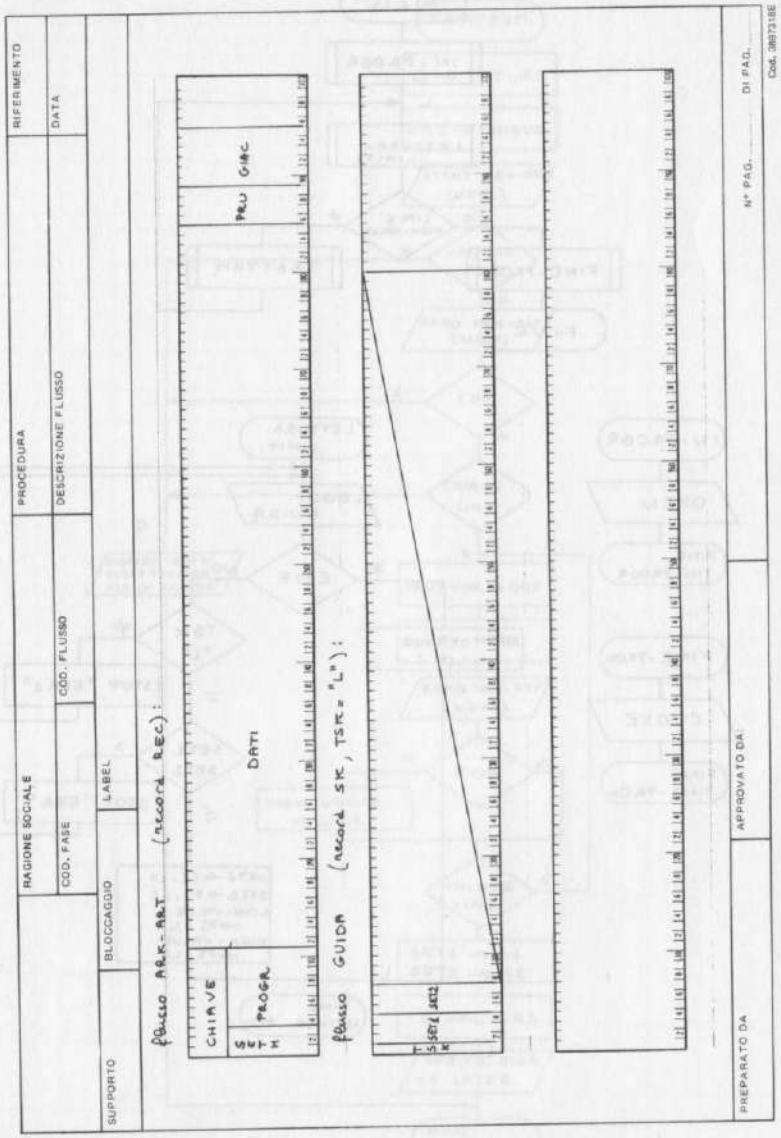


Fig. 11.7

Fig. 11.8a

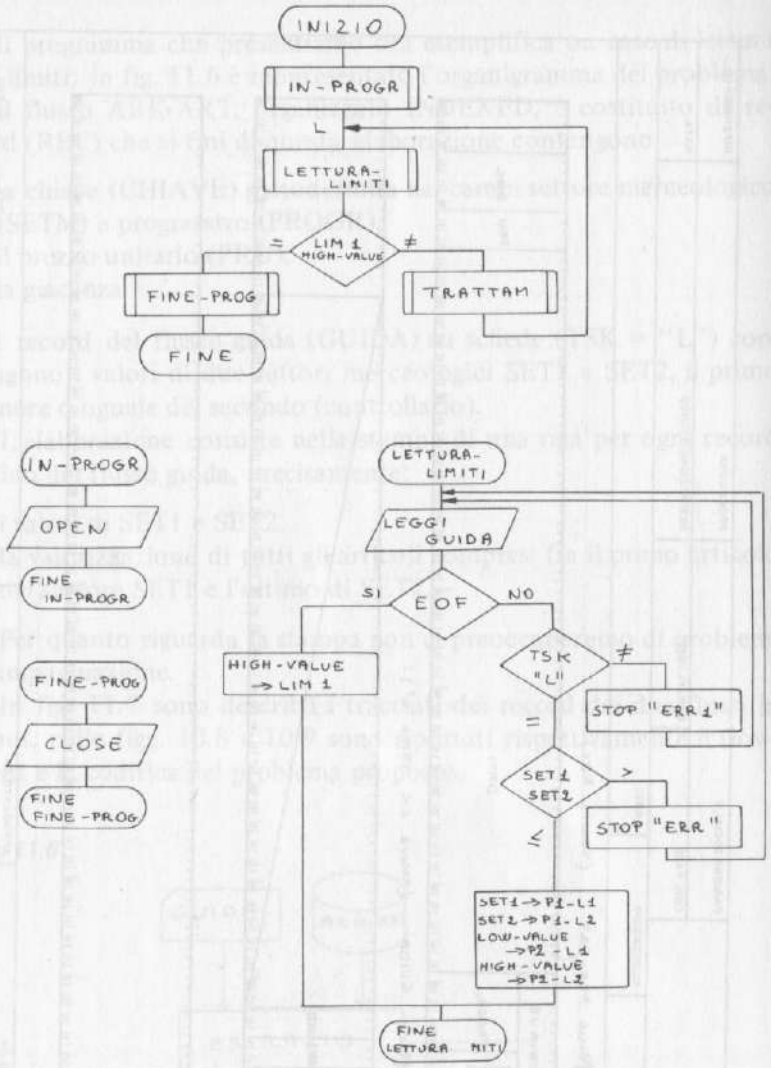


Fig. 11.8b

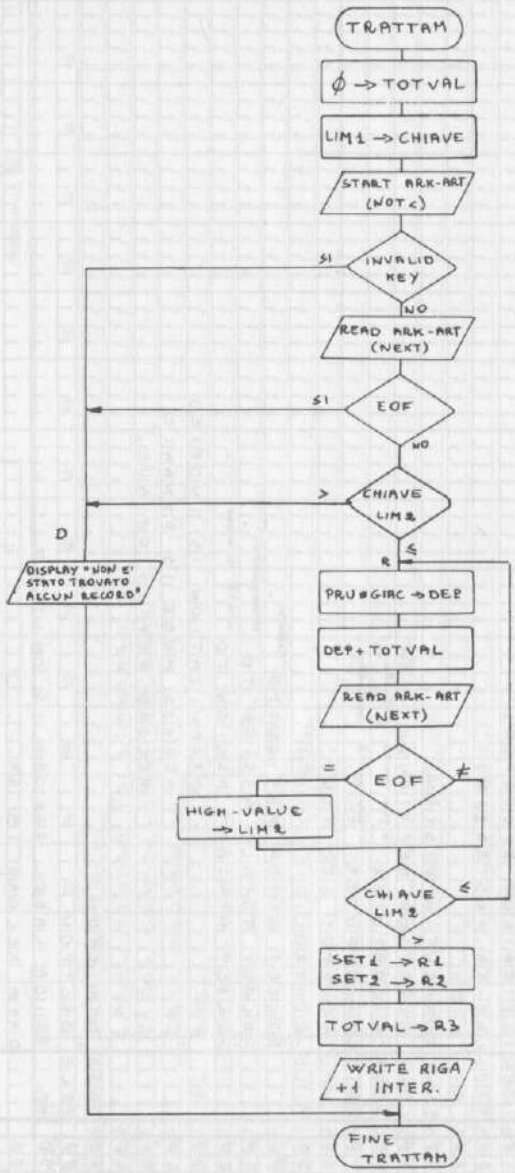


Fig. 11.9a

PROGRAM NAME	PROGRAMMER	DATE

PAGE Nº 001

PROGRAM-ID ESTRAZIO

```
LINE  CONT.  A  B
4     8      8  12   16   20   24   28   32   36   40   44   48   52   56   60   64   68   72
Ø1Ø          IDENTIFICATION DIVISION.
Ø2Ø          PROGRAMMA-IDE. ESTRAZIO.
Ø3Ø
Ø4Ø          ENVIRONMENT DIVISION.
Ø5Ø          CONFIGURATION SECTION.
Ø6Ø          SOURCE-COMPUTER.  sigla unità centrale.
Ø7Ø          OBJECT-COMPUTER.  sigla unità centrale.
Ø8Ø          INPUT-OUTPUT SECTION.
Ø9Ø          FILE-CONTROL.
1Ø0          SELECT GUIDA ASSIGN TO  lettore.
11Ø          SELECT TOTAL ASSIGN TO  stampante.
12Ø          SELECT ARK-ART ASSIGN TO  unità a disco
13Ø          ORGANIZATION IS INDEXED
14Ø          ACCESS MODE IS DYNAMIC
15Ø          RECORD KEY IS CHIAVE.
16Ø
17Ø          DATA DIVISION.
18Ø          FILE SECTION.
19Ø          FD GUIDA RECORD IS OMITTED
2Ø0          DATA RECORD IS SK.
21Ø          Ø1 SK.
22Ø          Ø2 TSK PIC X.
23Ø          Ø2 SET1 PIC 9(13).
24Ø          Ø2 SET2 PIC 9(13).
```


Fig. 11.9b

PROGRAM NAME	PROGRAMMER	DATE

PAGE Nº 002

PROGRAM-ID ESTRAZIO

```
LINE  CONT.  A  B
4     8      8  12   16   20   24   28   32   36   40   44   48   52   56   60   64   68   72
1Ø0          FD ARK-ART
11Ø          LABEL RECORD IS STANDARD
12Ø          BLOCK CONTAINS 21 RECORDS
13Ø          DATA RECORD IS REK.
14Ø          Ø1 REC.
15Ø          Ø2 CHIAVE.
16Ø          Ø3 SETIMP PIC 9(13).
17Ø          Ø3 PROGR PIC 9(8).
18Ø          Ø2 DATI PIC X(174).
19Ø          Ø2 PRU PIC 9(14).
2Ø0          Ø2 GIAC PIC 9(16).
21Ø          FD TOTAL
22Ø          LABEL RECORD OMITTED
23Ø          DATA RECORD IS RIGA
24Ø          Ø1 RIGA.
25Ø          Ø2 R1 PIC B(5)999.
26Ø          Ø2 R2 PIC BB999B(5).
27Ø          Ø2 R3 PIC Z(10).
28Ø          WORKING-STORAGE SECTION.
29Ø          Ø1 CIM1.
3Ø0          Ø2 P1-LI1 PIC 9(13).
31Ø          Ø2 P2-LI1 PIC 9(18).
32Ø          Ø1 CIM2.
33Ø          Ø2 P1-LI2 PIC 9(13).
```

Fig. 11.9c

PAGE No 003

PROGRAM NAME	PROGRAMMER	DATE

PROGRAM-ID ESTRAZIO

```
LINE  CONT.  A  B
4     8      8  12  16  20  24  28  32  36  40  44  48  52  56  60  64  68  72
010          |  012 P2-L2 PIC 9(8).
020          012 DEP PIC 9(9).
030          012 TOTVAL PIC 9(10).
040
050          PROCEDURE DIVISION.
060          PROGRAMMA SECTION.
070          PERFORM IN-PROG.
080          L. PERFORM LETTURA-LIMITI
090          IF LIMIT = HIGH-VALUE PERFORM FINE-PROG
100          STOP RUN.
110          PERFORM TRATTAMI
120          GO TO L.
130
140          IN-PROG SECTION.
150          OPEN INPUT GUIDA
160          OPEN OUTPUT TOTAL
170          OPEN I-O PARK-ART.
180          EXIT-IN-PROG. EXIT.
190
200          FINE-PROG SECTION.
210          CLOSE GUIDA TOTAL PARK-ART.
220          EXIT-FINE-PROG. EXIT.
230
240
```

Fig. 11.9d

PAGE No 004

PROGRAM NAME	PROGRAMMER	DATE

PROGRAM-ID ESTRAZIO

```
LINE  CONT.  A  B
4     8      8  12  16  20  24  28  32  36  40  44  48  52  56  60  64  68  72
010          LETTURA-LIMITI SECTION.
020          READ GUIDA AT END MOVE HIGH-VALUE TO LIMIT
030          GO TO EXIT-LETTURA LIMITI.
040          IF TSK NOT = "L" STOP "ERR1"
050          GO TO LETTURA-LIMITI.
060          IF SET1 > SET2 STOP "ERR"
070          GO TO LETTURA-LIMITI.
080          MOVE SET1 TO P1-L1
090          MOVE SET2 TO P1-L2
100          MOVE LOW-VALUE TO P2-L1
110          MOVE HIGH-VALUE TO P2-L2.
120          EXIT-LETTURA LIMITI. EXIT.
130
140          TRATTAMI SECTION.
150          MOVE ZERO TO TOTVAL
160          MOVE LIMIT TO CHIAVE
170          START PARK-ART KEY IS NOT < CHIAVE
180          INVALID KEY GO TO D.
190          READ PARK-ART NEXT AT END GO TO D.
200          IF CHIAVE > LIM2 GO TO D.
210          MULTIPLY PRU BY GIAC GIVING DEP
220          ADD DEP TO TOTVAL
230          READ PARK-ART NEXT
240          AT END HIGH VALUE TO LIM2.
```

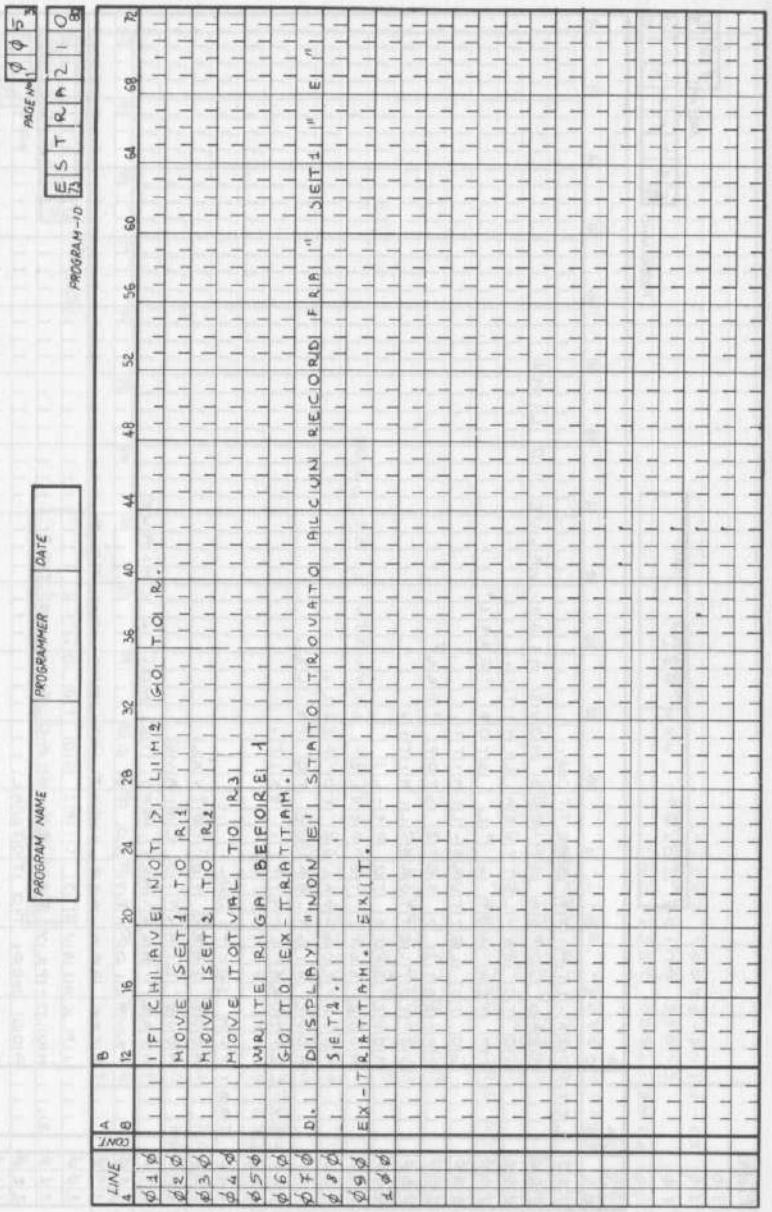


Fig. 11.9e

11.7. Riepilogo

Come già è stato fatto in 9. e 10. anche in questo capitolo schematizziamo le possibilità offerte dall'organizzazione in esame (indexed) in relazione ai vari tipi di accessi.

Occorre precisare che non sono prese in esame le opzioni FROM ed INTO e che, dove sono riportate la clausola AT END oppure la INVALID KEY, resta inteso che possono essere omesse provvedendo all'inserimento nelle DECLARATIVES di una USE appropriata.

11.7.1. *Accesso sequenziale*

a. OPEN INPUT

In fig. 11.10 si può osservare che un flusso indexed in accesso sequenziale trattato con una OPEN INPUT può essere elaborato solo dall'istruzione READ... AT END... preceduta o meno da una operazione di posizionamento all'interno del flusso tramite una START (in questo caso è possibile dare un ordine di lettura solo se non si è usciti per INVALID KEY).

b. OPEN OUTPUT

L'unica operazione consentita quando si tratta in output un flusso indexed con accesso sequenziale è la WRITE (fig. 11.11).

c. OPEN I-O

Nel caso di OPEN I-O per flussi con organizzazione indexed sono possibili le stesse operazioni previste in questa circostanza per un flusso organizzato in modo "relative" con la sola differenza che l'istruzione REWRITE ha il test di INVALID KEY che serve per cautelarsi da variazioni delle chiavi sul record che viene riregistrato (fig. 11.12).

Fig. 11.10

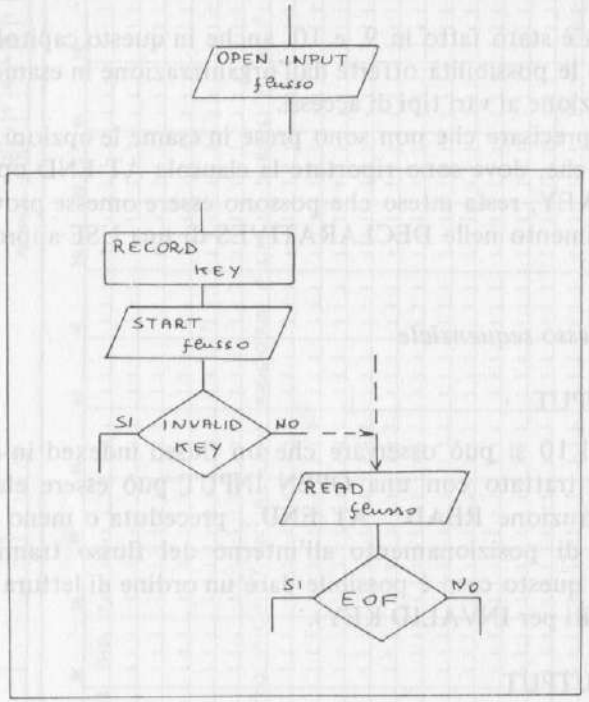


Fig. 11.11

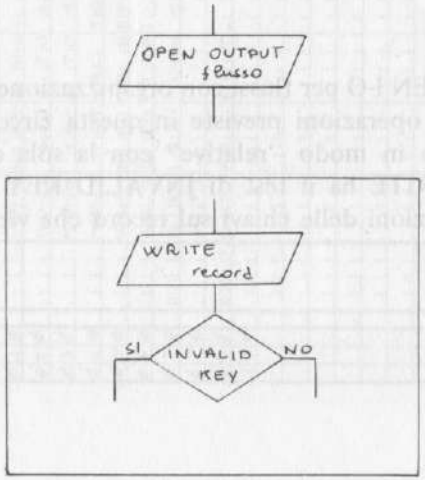
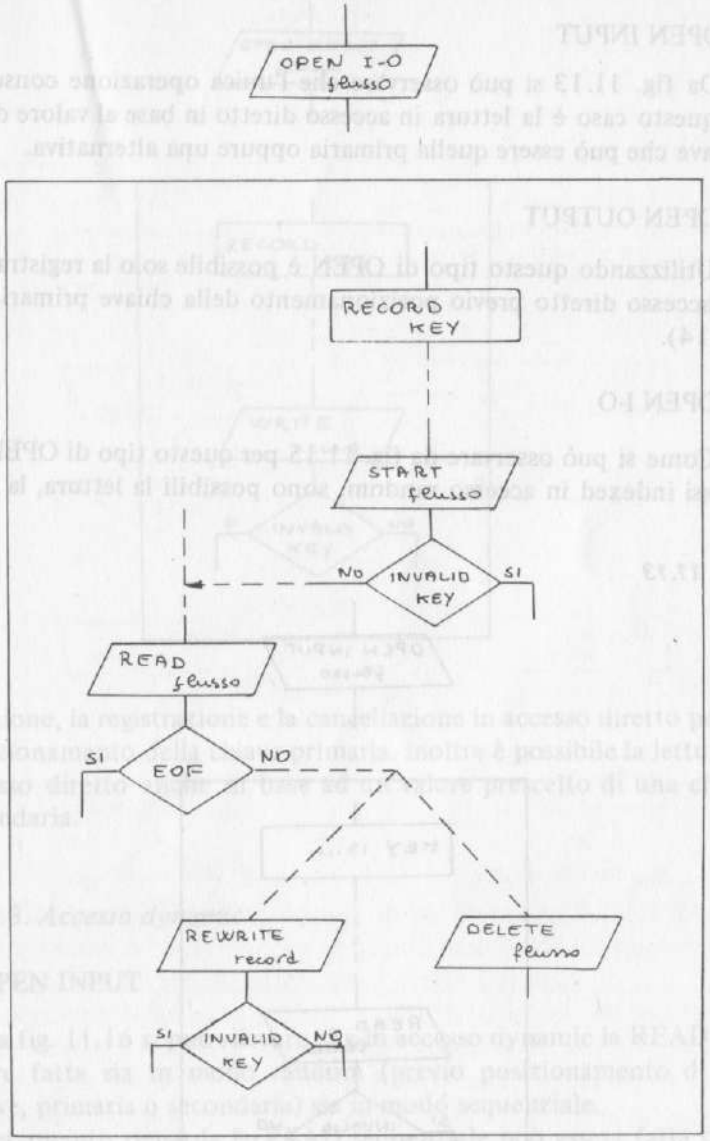


Fig. 11.12



11.7.2. *Accesso random*

a. OPEN INPUT

Da fig. 11.13 si può osservare che l'unica operazione consentita in questo caso è la lettura in accesso diretto in base al valore di una chiave che può essere quella primaria oppure una alternativa.

b. OPEN OUTPUT

Utilizzando questo tipo di OPEN è possibile solo la registrazione in accesso diretto previo posizionamento della chiave primaria (fig. 11.14).

c. OPEN I-O

Come si può osservare da fig. 11.15 per questo tipo di OPEN, per flussi indexed in accesso random, sono possibili la lettura, la riregi-

Fig. 11.13

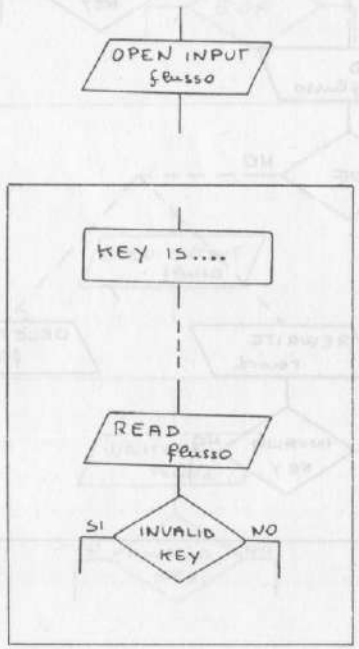
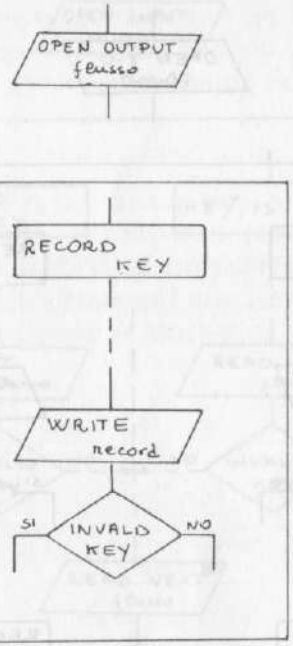


Fig. 11.14



strazione, la registrazione e la cancellazione in accesso diretto previo posizionamento della chiave primaria, inoltre è possibile la lettura in accesso diretto anche in base ad un valore prescelto di una chiave secondaria.

11.7.3. *Accesso dynamic*

a. OPEN INPUT

Da fig. 11.16 si può rilevare che in accesso dynamic la READ può essere fatta sia in modo random (previo posizionamento di una chiave, primaria o secondaria) sia in modo sequenziale.

Per quanto riguarda la READ sequenziale può essere fatta dopo un'operazione di posizionamento con esito positivo (sulla base della chiave primaria) e in questo caso la READ sequenziale opera a

Fig. 11.15

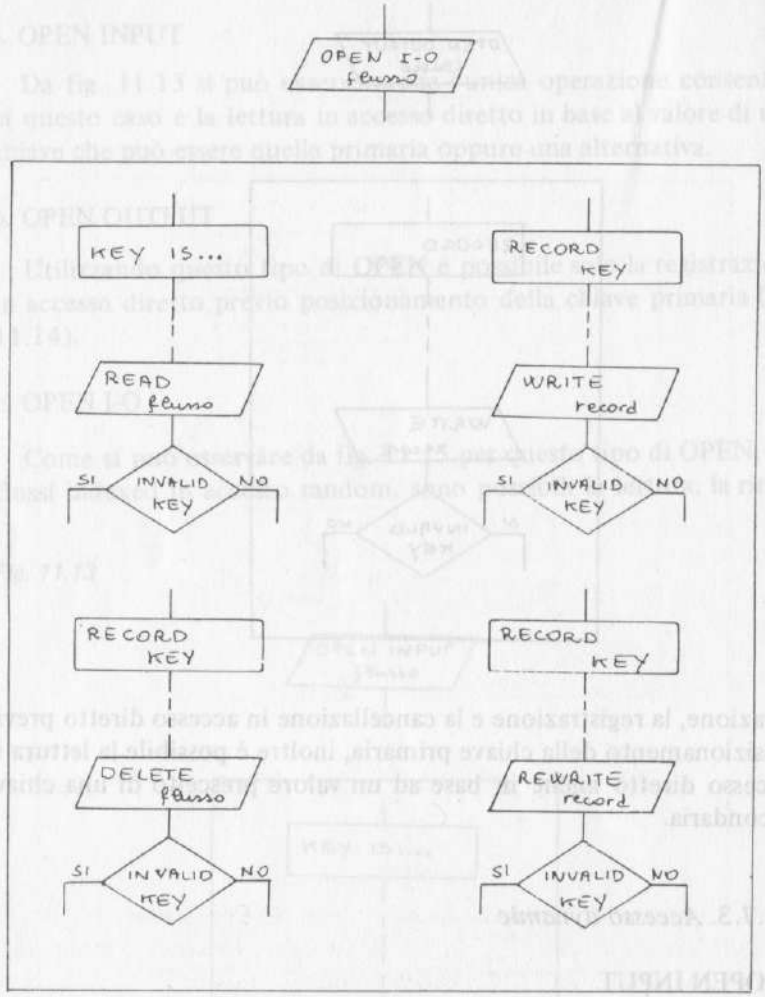
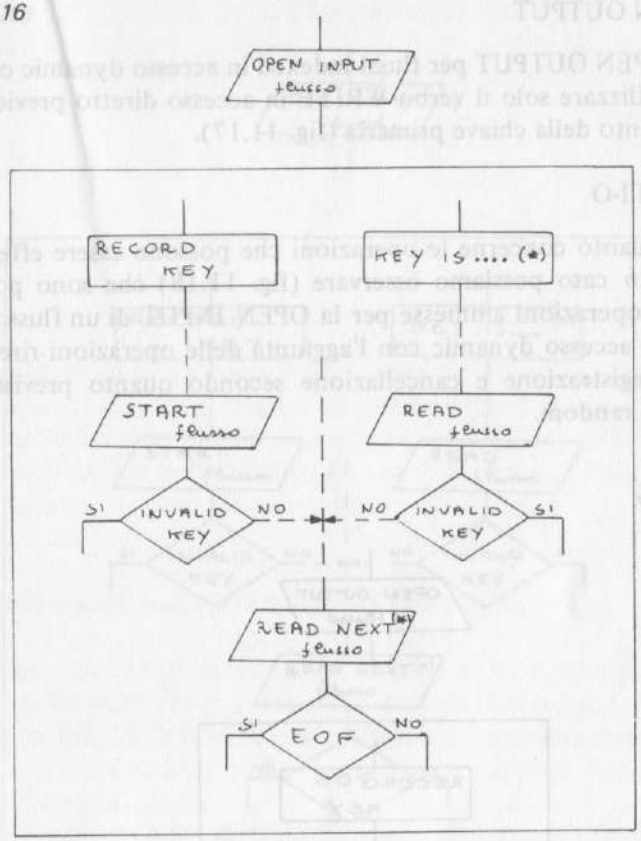


Fig. 11.16



partire dal record puntato, oppure dopo un'operazione di lettura random andata a buon fine, ed in questo caso la READ sequenziale opera a partire dal record successivo a quello trattato.

In particolare nel caso in cui una READ sequenziale segua una READ random la chiave secondo cui vengono forniti i record in sequenza è la stessa utilizzata nella READ random, finché non venga effettuato un posizionamento con la START o non venga eseguita un'altra READ random.

b. OPEN OUTPUT

La OPEN OUTPUT per flussi indexed in accesso dynamic consente di utilizzare solo il verbo WRITE in accesso diretto previo posizionamento della chiave primaria (fig. 11.17).

c. OPEN I-O

Per quanto concerne le operazioni che possono essere effettuate in questo caso possiamo osservare (fig. 11.18) che sono possibili tutte le operazioni ammesse per la OPEN INPUT di un flusso indexed con accesso dynamic con l'aggiunta delle operazioni riregistrazione, registrazione e cancellazione secondo quanto previsto per l'accesso random.

Fig. 11.17

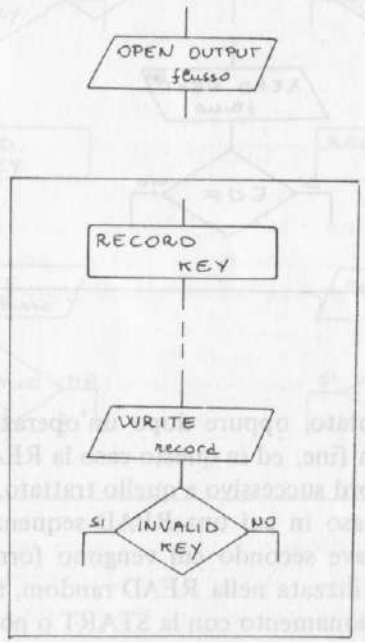
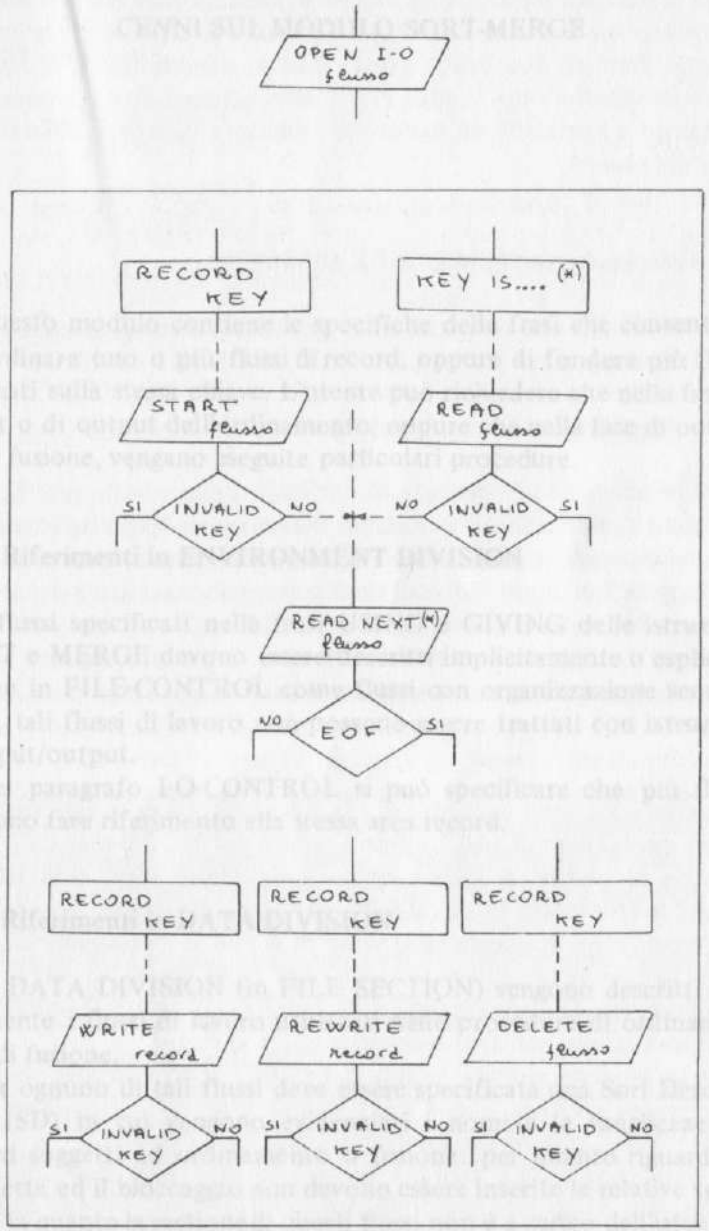


Fig. 11.18



APPENDICE A

CENNI SUL MODULO SORT-MERGE

Questo modulo contiene le specifiche delle frasi che consentono di ordinare uno o più flussi di record, oppure di fondere più flussi ordinati sulla stessa chiave. L'utente può richiedere che nella fase di input o di output dell'ordinamento, oppure che nella fase di output della fusione, vengano eseguite particolari procedure.

A.1. Riferimenti in ENVIRONMENT DIVISION

I flussi specificati nelle frasi USING e GIVING delle istruzioni SORT e MERGE devono essere descritti implicitamente o esplicitamente in FILE-CONTROL come flussi con organizzazione sequenziale, tali flussi di lavoro non possono essere trattati con istruzioni di input/output.

Nel paragrafo I-O-CONTROL si può specificare che più flussi possono fare riferimento alla stessa area record.

A.2. Riferimenti in DATA DIVISION

In DATA DIVISION (in FILE SECTION) vengono descritti parzialmente i flussi di lavoro utilizzati nelle procedure di ordinamento e di fusione.

Per ognuno di tali flussi deve essere specificata una Sort Description (SD) in cui vengono evidenziati i nomi e le lunghezze dei record soggetti ad ordinamento o fusione: per quanto riguarda le etichette ed il bloccaggio non devono essere inserite le relative specifiche in quanto la gestione di questi flussi non è a carico dell'utente.

L'indicatore SD è seguito immediatamente dal nome del flusso al quale si riferisce e può essere inoltre seguito da frasi che specificano la dimensione dei record e che forniscono il nome dei record che saranno descritti di seguito: queste frasi sono però di tipo documentativo in quanto i tipi record che costituiscono il flusso compaiono di seguito a livello 01 ed inoltre la lunghezza si deduce dalle descrizioni dei record.

A.3. Riferimenti in PROCEDURE DIVISION

Le istruzioni che consentono di gestire le funzioni di ordinamento e di fusione sono: RELEASE, RETURN, SORT e MERGE.

a. RELEASE nome-record [FROM nome-dato]

è l'istruzione che consente di trasferire i record alla fase iniziale di una operazione di sort, dopo l'esecuzione di questa istruzione il record non è più disponibile e, quando il controllo passa alla procedura di input il flusso risulta costituito da tutti i record che sono stati trattati da un'istruzione RELEASE;

b. RETURN nome-flusso [INTO nome-dato]
AT END ... • ...

questa istruzione consente di ottenere i record dalla fase finale di una operazione di sort oppure i record già fusi durante un'operazione di merge: il flusso citato deve essere stato descritto in DATA DIVISION in una descrizione flussi di tipo sort-merge.
L'esecuzione di questa istruzione fa si che il "prossimo record" dell'ordinamento, ottenuto sulla base delle chiavi su cui si fa l'operazione di ordinamento o di fusione, sia reso disponibile per l'elaborazione nell'area record associata al flusso.
Quando viene rilevata la fine del flusso, dopo l'esecuzione delle istruzioni relative, non è più possibile utilizzare l'istruzione RETURN nell'ambito della corrente procedura di output;

c. L'istruzione SORT crea un flusso ordinato sulla base di chiavi precisate, eseguendo eventuali procedure di input, e rende disponibili i record ordinati su un flusso oppure per il trattamento da parte di procedure di output:

SORT nome-flusso$_1$

ON {ASCENDING / DESCENDING} KEY nome-dato$_1$ [nome-dato$_2$] ...

[ON {ASCENDING / DESCENDING} KEY nome-dato$_3$ [nome-dato$_4$] ...] ...

{INPUT PROCEDURE IS nome-sezione$_1$ [{THROUGH / THRU} nome-sezione$_2$]}
{USING nome-flusso$_2$ nome-flusso$_3$ ....}

{OUTPUT PROCEDURE IS nome-sezione$_3$ [{THROUGH / THRU} nome-sezione$_4$]}
{GIVING nome-flusso$_4$}

Il flusso$_1$ deve essere descritto in una Sort Description mentre i flussi 2, 3 e 4 devono essere descritti con delle File Description.
I vari nomi-dato che vengono utilizzati per indicare le chiavi di ricerca devono essere stati descritti come parte dell'area record del flusso$_1$.
La procedura di input e quella di output consistono di una o più sezioni contigue (e non devono avere delle parti in comune).
La procedura di input deve contenere l'istruzione RELEASE per poter trasferire i record da ordinare al flusso$_1$, questa procedura può contenere istruzioni atte a modificare, selezionare e generare record ma è soggetta ad alcune limitazioni, e precisamente non può contenere istruzioni SORT o MERGE e non è possibile il trasferimento del controllo (tramite GO TO, PERFORM, ALTER) fuori della procedura stessa.
Quando è utilizzata una procedura di input il controllo vi passa prima che il flusso$_1$ sia ordinato, il compilatore inserisce un meccanismo di rientro alla fase di ordinamento alla fine dell'ultima sezione di tale procedura.
Per quanto riguarda la procedura di output si hanno limitazioni dello stesso tipo ed inoltre, quando tale procedura è utilizzata, il controllo vi passa dopo che il flusso$_1$ è stato ordinato. Una procedura di output deve contenere l'istruzione RETURN che consente di prelevare i record ordinati dalla fase di ordinamento per elaborarli nella procedura di output.

d. L'istruzione MERGE crea un flusso ordinato effettuando la fusione dei record di due o più flussi precedentemente ordinati sulla stessa chiave e rende disponibili i record che escono dalla fase di fusione per eventuali trattamenti di una procedura di output: le osservazioni sono analoghe a quelle fatte per l'operazione di SORT.

```
MERGE nome-flusso1
    [ON {ASCENDING} KEY nome-dato1 [nome-dato2] ...
        {DESCENDING}
    ON {ASCENDING} nome-dato3 [nome-dato4] ...] ...
        {DESCENDING}
    USING nome-flusso2 nome-flusso3 ...
    {OUTPUT PROCEDURE IS nome-sezione1 [{THROUGH} nome-sezione2]}
    {GIVING nome-flusso4                {THRU}                  }
```

APPENDICE B

CENNI SUL MODULO REPORT WRITER

Il modulo Report Writer contiene la descrizione delle frasi che il linguaggio Cobol prevede per la generazione automatica dei tabulati.

Per la definizione dell'organizzazione logica di un tabulato è utilizzata una gerarchia di livelli: gruppo di elementi di inizio tabulato, gruppo iterativo nel tabulato ed infine il gruppo di chiusura del tabulato; il gruppo iterativo è a sua volta pensato suddiviso in parte di inizio pagina, righe dettaglio e parte di fine pagina.

In corrispondenza della parola riservata LINE-COUNTER è generato uno speciale registro per ogni Report definita in DATA DIVISION, tale registro è gestito dal sistema per determinare i posizionamenti verticali, il valore del LINE-COUNTER può essere esaminato in PROCEDURE DIVISION ma non può essere modificato dall' utente.

PAGE-COUNTER è il nome di un registro speciale definito automaticamente dal sistema e aggiornato in relazione alla variazione delle pagine: può essere esaminato, ma non modificato, dall'utente.

La Report Writer consente di generare flussi sequenziali in output, in FILE-CONTROL tali flussi devono essere implicitamente o esplicitamente definiti con organizzazione sequenziale.

B.1. Riferimenti in DATA DIVISION

In FILE SECTION per il flusso in stampa deve essere scritta una FD in cui invece della frase DATA RECORD compare la frase obbligatoria:

$$\left\{\begin{array}{l}\underline{\text{REPORT}}\ \text{IS}\\ \underline{\text{REPORTS}}\ \text{ARE}\end{array}\right\}\ \text{nome-tabulato}_1\ \left[\text{nome-tabulato}_2\right]\ldots$$

che consente di precisare il numero di tabulati che sono definiti in REPORT SECTION, la FD di un flusso di questo tipo non è seguita da descrizioni di record.

In REPORT SECTION in corrispondenza di ogni tabulato il cui nome compare in REPORT IS di FD deve essere scritta una RD (Report Description).

Nella RD di un tabulato vengono specificati i limiti fisici della pagina del tabulato, la posizione della prima riga della testata del foglio, della prima e dell'ultima riga dettaglio e quella della prima riga del gruppo di fine-foglio.

Oltre a queste caratteristiche di tipo generale (relative al tabulato) nella RD si passa alla descrizione individuale degli elementi caratteristici di ogni riga specificando la posizione di stampa di ogni singolo campo, la PICTURE, il nome del campo di riferimento per determinare qual'è il dato da stampare in quella posizione, la sua posizione di stampa (assoluta nell'ambito della pagina o relativa rispetto ad altre righe) ed infine il tipo di gruppo di righe al quale appartiene.

Tutte le descrizioni di cui abbiamo accennato sono piuttosto pesanti ma consentono di risparmiare molto nella codifica della PROCEDURE DIVISION che risulta molto più lineare e sintetica.

B.2. Riferimenti in PROCEDURE DIVISION

Il dettaglio della descrizione dei formati delle righe e della struttura del tabulato viene "premiato" in PROCEDURE DIVISION dove il programmatore può gestire con semplicità la stampa del tabulato.

Dopo la OPEN del flusso a cui la Report appartiene il programmatore "apre" il tabulato con un'istruzione INITIATE e alla fine della stampa di quel tabulato farà eseguire una TERMINATE: dopo l'ultima TERMINATE di tabulati associati ad un flusso bisogna dare l'ordine di CLOSE per quel flusso.

L'istruzione di stampa riga è la GENERATE e può essere riferita solo a righe dettaglio, la gestione automatica del LINE-COUNTER e

le specifiche di descrizione del tabulato consentono la generazione automatica, al momento opportuno, delle righe relative all'inizio e alla fine sia del tabulato sia della pagina.

Anche per il modulo Report Writer è prevista la possibilità di utilizzare una USE:

<u>USE</u> <u>BEFORE</u> <u>REPORTING</u> identificatore.

che consente di fare eseguire la procedura a cui si riferisce prima che sia prodotta la riga identificata; tale procedura non deve contenere nessuna della istruzioni INITIATE, GENERATE e TERMINATE, ma può contenere l'istruzione SOPPRESS che inibisce la stampa delle righe di un gruppo di stampa.

APPENDICE C

CENNI SUL MODULO SEGMENTATION

In questo modulo vengono fornite le specifiche che lo standard Cobol prevede per la segmentazione dei programmi.

La segmentazione di un programma Cobol è attuata utilizzando la struttura a sezioni: ogni sezione viene numerata con un valore compreso fra 0 e 99:

nome-sezione SECTION [numero].

Se il numero è omesso è inteso uguale a zero: le sezioni che hanno un numero assegnato compreso tra zero e 49 concorrono insieme alle zone dati e alle tavole flussi a formare quella porzione di programma che risiede, durante l'elaborazione, in modo permanente in memoria (segmenti permanenti), mentre le sezioni che hanno assegnato un numero compreso tra 50 e 99 vengono alternativamente richiamate in memoria a seconda delle esigenze di elaborazione (segmenti indipendenti).

Quando l'insieme dei segmenti residenti e dell'area corrispondente al segmento indipendente di lunghezza massima supera la dimensione della memoria disponibile si possono adottare diverse soluzioni:

a. rinumerare le sezioni in modo da ridurre il numero dei segmenti residenti,
b. riorganizzare il programma,
c. utilizzare (in OBJECT-COMPUTER) una opzione che consente di variare il set di valori a cui corrispondono segmenti residenti:

[SEGMENT-LIMIT IS numero-di-segmento]

il numero citato diventa la soglia oltre la quale una sezione dà origine ad un segmento indipendente.

Naturalmente la segmentazione implica alcune considerazioni ed alcune limitazioni: circa il tipo di flow-chart utilizzato:

a. se la GO TO su cui opera un verbo ALTER,che serve per modificare il nome procedura di una istruzione di salto, è situata in un segmento indipendente le due istruzioni devono trovarsi nello stesso segmento;
b. una PERFORM che si trova in un segmento residente può richiamare sezioni che si trovano in segmenti residenti oppure sezioni che si trovano in un singolo segmento indipendente;
c. una PERFORM che si trova in un segmento indipendente può richiamare parte del residente oppure paragrafi situati nello stesso indipendente;
d. una istruzione di SORT (oppure di MERGE) che si trova in un segmento residente e che richiama procedure di input o di output (di output) può richiamarle solo se appartengono al residente o sono contenute in un solo segmento indipendente;
e. una istruzione di SORT (oppure di MERGE) che si trova in un segmento indipendente e che richiama procedure di input o di output può richiamarle solo se appartengono al residente oppure se sono contenute nello stesso segmento indipendente.

APPENDICE D

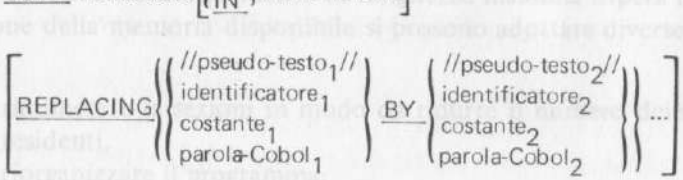
CENNI SUL MODULO LIBRARY

In questo modulo sono fornite le specifiche che consentono di indicare che un testo specificato deve essere copiato da una libreria.

Le librerie Cobol contengono dei testi che sono disponibili affinché il compilatore li prelevi al momento della compilazione: le porzioni di testo inserite nel programma sorgente vengono interpretate come una parte di programma sorgente.

L'importanza delle funzioni di questo modulo è rilevante in quanto possono essere codificate una volta per tutte alcune parti ripetitive, quali le descrizioni dei flussi che vengono utilizzati in più programmi oppure alcune routines di trattamento di tipo standard: questo consente non solo di risparmiare tempo in codifica ma garantisce una omogeneità di descrizioni.

L'istruzione che consente di inserire porzioni di sorgente, prelevate da una libreria, in un programma è:



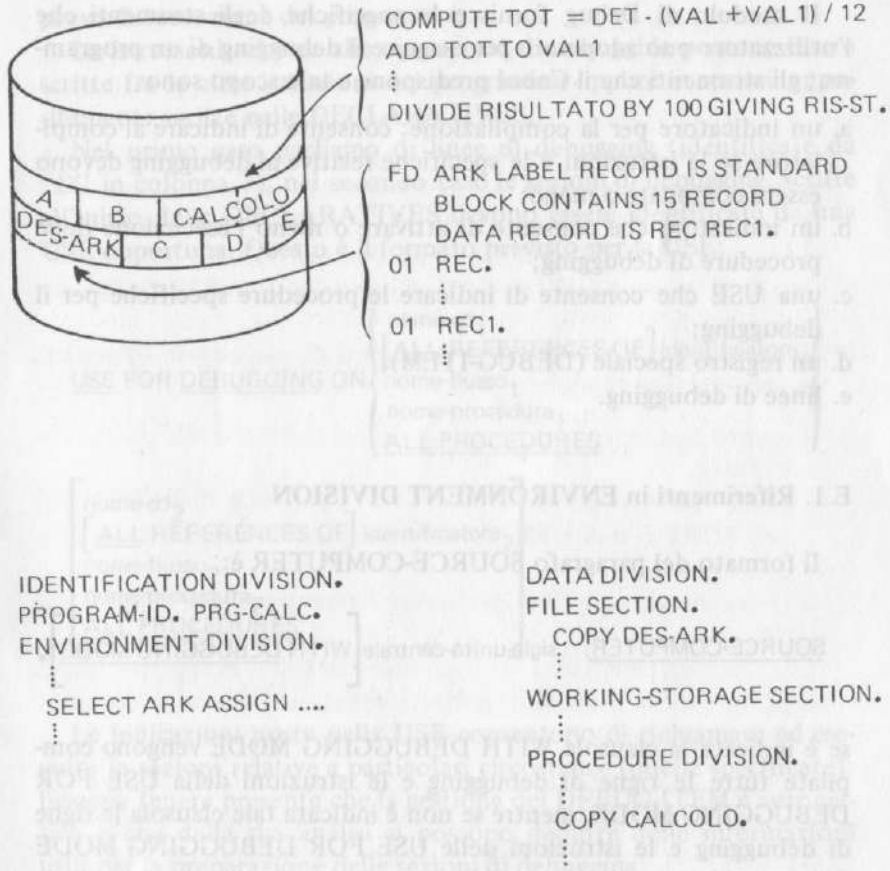
Nome-testo è il nome con cui la parte che si vuole inserire nel programma è memorizzata in libreria (se è ammesso che ci siano più librerie bisogna specificare il nome di quella a cui ci riferisce).

Se non è specificata l'opzione REPLACING il testo citato è copiato integralmente, nel caso contrario vengono effettuate le sostituzioni citate.

Per pseudo-testo si intende una qualunque sequenza di caratteri (pseudo-testo$_1$ non può essere omesso e neppure può essere costituito da soli spazi).

In fig. D.1. è riportato un esempio di utilizzo dell'istruzione COPY.

Fig. D.1.



APPENDICE E

CENNI SUL MODULO DEBUG

Il modulo di Debug fornisce le specifiche degli strumenti che l'utilizzatore può adoperare per eseguire il debugging di un programma; gli strumenti che il Cobol predispone a tale scopo sono:

a. un indicatore per la compilazione: consente di indicare al compilatore se le istruzioni e le specifiche relative al debugging devono essere compilate o meno;
b. un indicatore che consente di attivare o meno l'esecuzione delle procedure di debugging;
c. una USE che consente di indicare le procedure specifiche per il debugging;
d. un registro speciale (DEBUG-ITEM);
e. linee di debugging.

E.1. Riferimenti in ENVIRONMENT DIVISION

Il formato del paragrafo SOURCE-COMPUTER è:

```
SOURCE-COMPUTER. sigla-unità-centrale [WITH DEBUGGING MODE].
```

se è indicata la clausola WITH DEBUGGING MODE vengono compilate tutte le righe di debugging e le istruzioni della USE FOR DEBUGGING MODE, mentre se non è indicata tale clausola le righe di debugging e le istruzioni delle USE FOR DEBUGGING MODE sono considerate come linee di commento.

E.2. Riferimenti in PROCEDURE DIVISION

Quando un programma viene compilato WITH DEBUGGING MODE viene generato automaticamente un registro speciale (DEBUG-ITEM) atto a contenere informazioni utili per il debugging del programma.

Il DEBUG-ITEM è sottodefinito in una serie di aree specializzate a contenere determinate informazioni ed identificate da parole riservate.

Il programmatore facendo dei test sul contenuto del DEBUG-ITEM, o semplicemente stampandolo, può seguire l'andamento della prova di esecuzione del programma.

Le istruzioni relative alle operazioni di debugging possono essere scritte fra le altre istruzioni del programma oppure in sezioni appositamente scritte nelle DECLARATIVES.

Nel primo caso parliamo di linee di debugging (identificate da "D" in colonna 7), nel secondo caso le sezioni di debugging, scritte all'inizio delle DECLARATIVES devono essere identificate da una USE opportuna. Questo è il formato previsto per la USE:

```
                    { nome-cd1                                  }
                    { [ALL REFERENCES OF] identificatore1       }
USE FOR DEBUGGING ON { nome-flusso1                              }
                    { nome-procedura1                           }
                    { ALL PROCEDURES                            }

[ nome-cd2                                                      ]
[ [ALL REFERENCES OF] identificatore2                           ]
[ nome-flusso2                                                  ] ...
[ nome-procedura2                                               ]
[ ALL PROCEDURES                                                ]
```

Le indicazioni poste nella USE consentono di richiamare ed eseguire le sezioni relative a particolari circostanze (quelle specificate): bisogna tenere presente che la gestione del DEBUG-ITEM è automatica e che dalla sua analisi si possono dedurre delle informazioni utili per la preparazione delle sezioni di debugging:

a. nome-cd è il riferimento delle procedure che devono essere ese-

guite dopo istruzioni relative al modulo di trasmissione dati (vedi appendice G): ENABLE, DISABLE, SEND, RECEIVE (con uscita diversa da NO DATA), IF MESSAGE: in questi casi il DEBUG-ITEM contiene il numero identificativo della linea del sorgente relativa a nome-cd, il nome-cd ed i contenuti dell'area associata a nome-cd;

b. quando è indicato un identificatore con l'opzione ALL le procedure specificate vengono eseguite dopo una qualunque istruzione che lo coinvolga; se invece l'opzione ALL non è indicata, le procedure specificate vengono eseguite dopo istruzioni che modifichino il contenuto di identificatore e dopo una WRITE o una REWRITE che faccia riferimento all'identificatore.
In questo caso il DEBUG ITEM contiene il numero di riga di riferimento nel sorgente, il nome dell'identificatore ed il suo contenuto dopo l'esecuzione dell'istruzione che lo ha coinvolto;

c. quando è indicato nome-flusso le procedure indicate sono eseguite dopo le istruzioni OPEN, CLOSE, READ (per uscita diversa da INVALID KEY o AT END), DELETE, START. Le informazioni contenute nel DEBUG-ITEM, sono il numero di riga del sorgente relativa a nome-flusso, nome-flusso, e, nel caso di READ, l'intero record letto;

d. l'indicazione nome-procedura consente di eseguire le istruzioni identificate dalla USE:
   - immediatamente dopo il passaggio del controllo alla procedura indicata,
   - prima dell'esecuzione di una GO TO, di una PERFORM, di una SORT o una MERGE che trasferisca il controllo alla procedura indicata,
   - prima di una ALTER che alteri nome-procedura in una GO TO,
   - prima dell'esecuzione di una USE relativa a nome-procedura,
   - prima che la procedura indicata venga eseguita la prima volta (se tale procedura non fa parte delle DECLARATIVES).

   In questi casi il DEBUG-ITEM contiene delle indicazioni sul numero di riga del sorgente relativo a nome-procedura, il nome della procedura ed eventuali altre informazioni diversificate a seconda dei casi.

e. quando è indicato ALL PROCEDURES gli effetti descritti in d. si hanno in corrispondenza di ogni nome-procedura (ad eccezione delle procedure di debugging) che compare nel programma, e pertanto, la stampa del DEBUG-ITEM, richiesta nella sezione della relativa USE, non è altro che la stampa del percorso del programma.

APPENDICE F

CENNI SUL MODULO INTER-PROGRAM COMMUNICATION

In questo modulo sono descritti gli elementi che sono previsti dal Cobol standard per la gestione delle comunicazioni tra un programma "chiamante" ed uno o più programmi "chiamati".

Le procedure di comunicazione consentono di abilitare il passaggio di controllo da un programma ad un'altra unità di elaborazione e di abilitare entrambi i programmi ad accedere ad uno stesso insieme di dati.

F.1. Riferimenti in DATA DIVISION: LINKAGE SECTION

La LINKAGE SECTION in un programma è significativa solo se il programma deve essere eseguito sotto il controllo di un ordine di chiamata scritto in un programma chiamante e se è previsto che i due programmi utilizzino dati comuni.

La LINKAGE SECTION è utilizzata per descrivere nel programma chiamato i dati che sono in comune con il programma chiamante, per tali dati non viene riservato spazio durante la compilazione ma le loro descrizioni servono per compilare le istruzioni di PROCEDURE DIVISION che ad essi fanno riferimento.

In LINKAGE SECTION possono essere definite anche aree record di flussi definiti nel programma chiamante; se interessa definire la struttura di più tipi record dello stesso flusso bisogna procedere con delle ridefinizioni in quanto ogni livello 01 di LINKAGE SECTION definisce (senza riservare) zone di memoria diverse.

Le regole di scrittura della LINKAGE-SECTION sono le stesse presentate per la WORKING-STORAGE SECTION ad eccezione del fatto che la clausola VALUE può essere utilizzata solo nella definizione di nomi condizione a livello 88.

Se nella LINKAGE SECTION vengono definite delle tabelle da trattare in modo indexing bisogna tenere presente che non è possibile creare una corrispondenza fra l'index-name del programma chiamato e del programma chiamante.

E' utile osservare, per poter ben strutturare la LINKAGE SECTION che la corrispondenza fra campi o aree record tra programma chiamante e programma chiamato può essere fatta solo utilizzando nomi identificati dal numero di livello 01 oppure 77.

F.2. Riferimenti in PROCEDURE DIVISION

Nel programma chiamato la PROCEDURE DIVISION è identificata dal nome divisione seguito eventualmente dall'opzione USING che ha lo scopo di indicare quali elementi descritti nella LINKAGE SECTION sono i corrispondenti di elementi descritti nel programma chiamante e riportati in una USING che compare nell'istruzione di chiamata.

PROCEDURE DIVISION [USING identificatore$_1$ [identificatore$_2$] ...].

Identificatore$_1$, identificatore$_2$,... sono i nomi definiti in LINKAGE SECTION a livello 01 oppure 77. Quando è indicata la USING nel programma chiamato nell'istruzione di chiamata del chiamante deve essere inserita una USING che contiene i nomi del programma chiamante che devono essere messi in corrispondenza con quelli del chiamato: nomi corrispondenti devono individuare aree della stessa lunghezza fra loro compatibili.

Quando non è indicata la USING nel programma chiamato significa che non deve essere creata alcuna corrispondenza di aree con il chiamante e pertanto non deve essere utilizzata la LINKAGE SECTION e il programma chiamante non deve avere la USING nell'istruzione di chiamata.

F.2.1. *Il verbo CALL*

Questo è il formato dell'istruzione che consente di richiamare un programma durante l'esecuzione di un altro programma:

$$\underline{\text{CALL}}\left\{\begin{matrix}\text{nome-dato}\\ \text{costante}\end{matrix}\right\}\left[\underline{\text{USING}}\ \text{identificatore}_1\ \left[\text{identificatore}_2\right]\ldots\right]$$

$$\left[\text{ON}\ \underline{\text{OVERFLOW}}\ldots\ .\right]$$

Come si può vedere dal formato il nome del programma richiamato può essere espresso direttamente (costante) oppure attraverso un nome-dato, questa seconda possibilità consente di gestire da programma le modalità di richiamo inserendo in nome-dato il nome del programma da richiamare.

L'opzione USING, come si è detto, consente di creare una corrispondenza fra nomi del programma chiamato e chiamante.

Quando è indicata l'opzione ON OVERFLOW... vengono eseguite le istruzioni indicate di seguito se al momento dell'esecuzione della CALL la memoria disponibile non è sufficiente per contenere il programma chiamato; se ci si trova in questa situazione e non è stata inserita l'opzione gli effetti dell'esecuzione della CALL sono quelli previsti da chi ha preparato il software dell'elaboratore che si sta utilizzando.

F.2.2. *Il verbo EXIT*

La frase EXIT PROGRAM indica la fine logica di un programma chiamato, quando viene incontrata il controllo viene restituito al programma chiamante e viene eseguita l'istruzione immediatamente seguente l'istruzione di chiamata.

L'istruzione

EXIT PROGRAM.

deve essere l'unica istruzione di un paragrafo.

Il verbo CANCEL

Questo verbo utilizzato nel formato

$$\underline{\text{CANCEL}}\left\{\begin{matrix}\text{nome-dato}_1\\ \text{costante}_1\end{matrix}\right\}\left[\begin{matrix}\text{nome-dato}_2\\ \text{costante}_2\end{matrix}\right]\ldots$$

consente di rendere disponibile al sistema operativo le aree di memoria occupate da programmi chiamati i cui nomi sono citati direttamente (costante$_1$, costante$_2$,...) o indirettamente attraverso dei nomi-dato che contengono nomi di programmi.

F.2.3. **Osservazioni**

a. L'uso delle specifiche relative al modulo di Inter-program communication non solo è interessante per la gestione dei sottoprogrammi esterni ma è importante quando un programma di notevoli dimensioni viene preparato da persone diverse: basta infatti cambiare in un programma chiamato alcuni elementi per poterlo provare in modo autonomo (si tratta in sostanza di fare qualche modifica nella definizione dei dati sopprimendo la LINKAGE SECTION e distribuendo i suoi elementi in WORKING-STORAGE SECTION e in FILE SECTION e preparando le SELECT relative ai flussi del programma chiamante che vengono utilizzati nel programma chiamato e di sopprimere la USING di PROCEDURE DIVISION).

b. Di seguito è riportato in modo schematizzato un esempio di richiamo di sottoprogramma esterno.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. CHIAMA.
ENVIRONMENT DIVISION.
      ⋮
      SELECT FL-1.......
      ⋮
DATA DIVISION.
FILE SECTION.
      ⋮
FD  FL-1....
    ....... DATA RECORD IS REC.
01  REC.
```

```
WORKING-STORAGE SECTION.
77  CONT .....
77  TOT .....
01  AREA .....
PROCEDURE DIVISION.
  CALL CHIAMATO USING REC CONT TOT AREA
IDENTIFICATION DIVISION.
PROGRAM-ID. CHIAMATO.
DATA DIVISION.
LINKAGE SECTION.
77  CTR .....
77  VALORE .....
01  CAMPO .....
01  ANAGRAFICO.
PROCEDURE DIVISION USING ANAGRAFICO CTR VALORE CAMPO.
```

APPENDICE G

CENNI SUL MODULO COMMUNICATION

Il modulo Communication contiene la descrizione delle specifiche delle frasi Cobol che abilitano l'accesso, l'elaborazione e la creazione di messaggi o di porzioni di messaggi per la comunicazione con periferiche e terminali remoti.

G.1. Riferimenti in DATA DIVISION

In un programma Cobol che prevede la gestione della trasmissione dati deve essere presente la COMMUNICATION SECTION: in questa sezione vengono inserite delle Communication Description (CD) in cui viene specificata l'interfaccia tra il programma Cobol e le routines di gestione della trasmissione dati.

Tale interfaccia è costituita da una serie di aree in cui possono essere indicate le code e le sottocode di input, i terminali destinatari di messaggi, la data e l'ora del messaggio, la lunghezza, lo status, ecc....

In particolare si ha una descrizione separata delle interfacce per l'input e per l'output in quanto si presentano problemi di tipo diverso, infatti per l'output è sufficiente indicare il destinatario del messaggio, mentre per l'input bisogna indicare la coda o anche una sottocoda (anche per vari livelli) di prelievo e quindi individuare, tramite valori che vengono caricati nell'interfaccia, il terminale da cui proviene il messaggio.

Per quanto riguarda le aree adibite al ricevimento dei messaggi o alla preparazione dei messaggi da inviare, vengono definite in WORKING-STORAGE SECTION.

G.2. Riferimenti in PROCEDURE DIVISION

Esaminiamo ora le istruzioni che possono essere utilizzate in PROCEDURE DIVISION per la gestione della trasmissione dei dati:

a. l'istruzione

IF MESSAGE FOR nome-cd

consente di determinare in un campo apposito dell'interfaccia di input il numero dei messaggi in attesa in una coda oppure in una particolare sottocoda.
Nome-cd è il nome mnemonico assegnato in CD ad una interfaccia di input;

b. Le istruzioni ENABLE e DISABLE consentono di abilitare o disabilitare una specificata sorgente o destinazione di messaggi;

c. per l'acquisizione di messaggi da una coda o da una sottocoda viene utilizzata l'istruzione RECEIVE in cui devono essere specificate: l'interfaccia di input in cui il programmatore ha posto le specifiche che consentono di individuare la coda o la sottocoda che interessa, l'area di WORKING-STORAGE SECTION in cui deve essere trasferito il messaggio in input ed infine le procedure che devono essere utilizzate nel caso in cui la coda indirizzata non contenga messaggi;

d. l'istruzione SEND consente di rilasciare alla coda specifica di un terminale (o alle code di più terminali) un messaggio o una porzione di messaggio.

INDICE ANALITICO

A (simbolo di PICTURE), 58
ACCEPT, 75
ACCESS MODE, 279, 303
ADD, 85, 165, 173
ADVANCING, 80, 159, 284
AFTER, 80, 159, 196, 198, 200, 265, 284, 287, 298, 312, 344, 384
Alfabetico, 70, 82
Alfanumerico, 70, 82
ALL, 27, 99, 196, 198, 200, 202
ALL PROCEDURES, 383
ALL REFERENCES, 383
ALPHABETIC, 84, 177
ALTER, 187, 373, 379
ALTERNATE, 335
AND, 178
ASCENDING, 373, 374
ASSIGN, 42, 279, 303, 335
Asterischi (**) - operatore, 168
Asterisco (*) - commento, 31
Asterisco (*) - operatore, 168
Asterisco (*) - simbolo di PICTURE, 127
AT END, 79, 157, 256, 283, 297, 300, 307, 339, 372, 384
AUTHOR, 37

B (simbolo di PICTURE), 128
Barra (/) - commento, 31
Barra (/) - operatore, 168
Barra (/) - simbolo di PICTURE, 128
BEFORE, 80, 159, 196, 198, 200, 284, 298
BLANK WHEN ZERO, 130, 137
BLOCK CONTAINS, 49, 280

CALL, 388
CANCEL, 388
Categorie, 58, 129, 160
CD, 391
CHARACTER(S), 40, 49, 136, 196, 198, 200, 280
CLOSE, 78, 156, 283, 305, 338, 376, 384
COMMUNICATION SECTION, 47, 391
COMPUTATIONAL, 141
COMPUTE, 186
CONFIGURATION SECTION, 37, 39
COPY, 380
CORRESPONDING, 173, 174
Costanti, 26, 70
COUNT, 204
CR (simbolo di PICTURE), 126
CURRENCY SIGN, 41, 124

D, 31, 383
DATA DIVISION, 34, 47, 280
DATA RECORD, 49, 280, 375
DATE, 75
DATE-COMPILED, 37
DATE-WRITTEN, 37
DAY, 75
DB (simbolo di PICTURE), 126
DEBUG-ITEM: 382

DEBUGGING MODE, 382
DECIMAL-POINT IS COMMA, 41, 122, 124
DECLARATIVES, 286, 383, 384
DELETE, 311, 344, 384
DELIMITED, 202, 204
DELIMITER, 204
DEPENDING ON, 186
DESCENDING, 373, 374
DISABLE, 384, 392
DISPLAY, 76, 141
DIVIDE, 90, 166
Divisione (DIVISION), 33, 35
Dollaro ($) - simbolo di PICTURE, 123
DOWN BY, 255
DYNAMIC, 303-335

Edizione (simboli), 120
ELSE, 92, 181
ENABLE, 384, 392
END DECLARATIVES, 286
ENTER, 207
ENVIRONMENT DIVISION, 34, 37, 279
EQUAL TO, 93, 176, 311, 343
ERROR, 287, 312
EXCEPTION, 287, 312, 344
EXIT, 96, 189, 388
EXTEND, 281, 287, 300

FD, 48, 280
FILE-CONTROL, 42, 279
FILE SECTION, 47, 48
FILE STATUS, 279, 285, 303, 335
FILLER, 59
FIRST, 198, 200
FOR, 196, 200
FROM, 75, 87, 159, 262, 265, 284, 285, 298, 300, 301, 309, 310, 326, 341, 342, 372

GENERATE, 376
GIUSTIFIED, 134
GIVING, 85, 87, 89, 90, 165, 166, 167, 373, 374
GO TO, 94, 186, 187, 373, 379, 384,
GREATER THAN, 93, 176, 311, 343

HIGH-VALUE(S), 27

IDENTIFICATION DIVISION, 34, 36
IF, 92, 139, 181
IF MESSAGE, 384, 392
IN, 170, 204, 380
INDEXED, 251, 335
INITIAL, 196, 198, 200
INITIATE, 376
INPUT, 78, 155, 281, 287, 297, 305, 312, 326, 329, 331, 337, 344, 361, 364, 365
INPUT-OUTPUT SECTION, 39-42
INPUT PROCEDURE, 373
INSPECT, 196, 198, 200
INSTALLATION, 37
INTO, 90, 157, 202, 204, 283, 297, 300, 307, 326, 339
INVALID KEY, 307, 309, 310, 311, 339, 341, 342, 343, 344, 372, 384
I-O, 281, 287, 300, 305, 312, 327, 330, 331, 337, 344, 361, 364, 368
I-O-CONTROL, 42, 280, 304, 371

KEY, 311, 335, 339, 343, 373, 374

LABEL RECORD, 48, 280
LEADING, 136, 196, 198, 200
LEFT, 135
LESS THAN, 93, 176, 311, 343
LINE(S), 80, 284
LINE-COUNTER, 375, 376
LINKAGE SECTION, 47, 386
Literal, 26
Livello (numeri di), 50, 54, 69
LOCK, 156, 283, 305, 338
LOW-VALUE(S), 27

Maggiore (>), 93, 176, 311, 343
MEMORY SIZE, 40
Meno (–) - continuazione, 31
Meno (–) - operatore, 168
Meno (–) - simbolo di PICTURE, 125
MERGE, 374, 379, 384
Minore (<), 93, 176, 311, 343
MODULES, 40
MOVE, 82, 160, 174
MULTIPLY, 88, 166

NEGATIVE, 178
NEXT, 181, 256, 307, 339
NO DATA, 384
Nome - condizione (88), 139, 178
NOT, 93, 94, 176, 177, 178, 311, 343
Nove (9) - Simbolo di PICTURE, 58
NUMERIC, 94, 177
Numerico, 70, 82
Numero (di livello), 50, 54, 69

OBJECT-COMPUTER, 39, 40, 378
OCCURS, 229, 251, 259
OFF, 170, 380
OMITTED, 48, 280
ON SIZE ERROR, 163
OPEN, 78, 155, 281, 297, 298, 300, 337, 376, 384
OPTIONAL, 42, 158, 279, 284
OR, 178, 204
ORGANIZATION, 279, 303
OUTPUT, 78, 155, 281, 287, 298, 305, 312, 327, 329, 331, 337, 344, 361, 364, 368
OUTPUT PROCEDURE, 373, 374
OVERFLOW, 202, 204, 388

PAGE, 159, 284
PAGE-COUNTER, 375
Paragrafo, 35, 73
Parentesi"(,)", 168
Parole, 21, 27
PERFORM, 96, 188, 191, 192, 193, 262, 265, 273, 369, 384
PICTURE, 57, 69
Più (+) - operatore, 168
Più (+) - simbolo di PICTURE, 125
POINTER, 202, 204
POSITIVE, 178
PROCEDURE DIVISION, 34, 73, 281, 387
PROCEDURE ON, 287, 312, 344
PROGRAM-ID, 36
Punto (·) - simbolo di PICTURE, 41, 121
Punto (·) - Separatore, 26, 28, 37
Punto e virgola (;), 28

Qualificazione, 170, 233
QUOTE(S), 27

RANDOM, 303, 335
RD, 376
READ, 79, 157, 283, 297, 300, 339, 384
RECEIVE, 384, 392
RECORD CONTAINS, 280
RECORDS, 49
REDEFINES, 132, 234
REEL, 156, 283
RELATIVE, 303
RELEASE, 372
REMAINDER, 163
REMOVAL, 156, 283
RENAMES (66), 137, 170
REPLACING, 198, 200, 380
REPORT(S), 376
REPORT SECTION, 47, 376
RETURN, 372
REVERSED, 155, 281
REWIND, 155, 156, 283
REWRITE, 285, 301, 310, 342, 384
RIGHT, 135
ROUNDED, 162
RUN, 97

S (Simbolo di PICTURE), 118
SD, 371

SEARCH, 256, 261
SECURITY, 37
SEGMENT-LIMIT, 378
Segno, 26, 118
SELECT, 42, 48, 279, 303, 335
SEND, 384, 392
SEPARATE, 136
SEQUENTIAL, 279, 303, 335
SET, 252, 254
Sezione, 35, 73, 378
SIGN, 136
SIZE, 202
SIZE ERROR, 163
SOPPRESS, 377
SORT, 373, 379, 384
SOURCE-COMPUTER, 39, 382
SPACE(S), 27
SPECIAL-NAMES, 40, 80, 122
STANDARD, 48, 280, 344
START, 311, 343
STOP, 97
STRING, 202
Struttura, 33, 35
SUBTRACT, 87, 166, 173
SYNCHRONIZED, 135

TALLYING, 196, 200, 204
TERMINATE, 376
THROUGH (THRU), 96, 138, 139, 188, 191, 192, 262, 265, 273
TIME(S), 75, 191, 229, 251
TRAILING, 136

Uguale (=), 93, 168, 176, 311
UNIT, 156, 283
UNSTRING, 204
UNTIL, 192, 262, 265
UP BY, 255
USAGE, 141
USAGE IS INDEX, 253
USE, 284, 312, 326, 344, 361
USE BEFORE REPORTING, 377
USE FOR DEBUGGING, 382
USING, 373, 374, 387, 388

V (simbolo di PICTURE), 118
VALUE, 70, 133, 139, 233, 386
VARYING, 252, 256, 262, 265
Verbo, 22
Virgola (,) - separatore, 28, 41
Virgola (,) - simbolo di PICTURE, 41, 121
Virgolette (", ,,), 26

WHEN, 256
WORDS, 40
WORKING-STORAGE SECTION, 47, 386, 391
WRITE, 80, 159, 284, 298, 309, 341, 384

X (Simbolo di PICTURE), 58

Z (Simbolo di PICTURE), 122
ZERO-ZEROS-ZEROES, 27
ZERO (0) - Simbolo di PICTURE, 123.

1290. *Quaderni di informatica, diretti da Franco Filippazzi*

1. Franco Filippazzi, *Tecnologia dell'elaboratore elettronico. Situazione attuale e prospettiva*
2. Enrico Spoletini, *Il Basic: teoria ed esercizi*
3. Enrico Spoletini, *Il Cobol: teoria ed esercizi*
4. Gaetano A. Lanzarone, Marco Maiocchi, Roberto Polillo, *Introduzione alla programmazione strutturata*
5. D. Ferrari, G. Serazzi, A. Zeigner, *Le prestazioni degli elaboratori elettronici. Misura/Valutazione/Ottimizzazione*
6. Giulio Occhini, *L'informatica nella gestione aziendale. Aspetti e prospettive d'impiego*
7. Alessandra ed Enrico Spoletini, *Il linguaggio Fortran*
8. Dorothy A. Walsh, *La programmazione in Pascal*
9. Trevor Housley, *Sistemi di trasmissione dati e di teleelaborazione*
10. Enrico Spoletini (a cura di), *La produzione del software applicativo. Evoluzione e prospettive*
11. Grayce M. Booth, *I sistemi distribuiti*
12. Dimitris N. Chorafas, *Progettazione e sviluppo di reti locali*
13. Jerome Kanter, *Sistemi informativi manageriali*
14. Antonio Pizzarello, *Ingegneria del software. Metodi e tecnologie per lo sviluppo di grandi sistemi informativi*
15. Andrew Monk, *Interazione uomo-computer. Elementi di base*
16. Angelo Maiocchi, *Il controllo di qualità del software*
17. Tom Conlon, *Imparare Micro-Prolog. Un approccio orientato alle soluzioni di problemi*

700. *Collana di informatica, diretta da Arrigo Frisiani*

1. Aldo de Luca, Luigi Maria Ricciardi, *Introduzione alla cibernetica*
2. Daniela Da Riva, Gabriele Ricci, *Elementi di organizzazione delle informazioni (Information retrieval)*
3. Arrigo Frisiani, *Calcolatori elettronici. Introduzione alla teoria delle reti logiche*
4. Massimiliano Lunelli, *Una biblioteca di sottoprogrammi dell'algebra lineare*
5. Nils J. Nilsson, *Metodi per la risoluzione dei problemi nell'intelligenza artificiale*
6. David Gries, *Principi di progettazione dei compilatori*
7. Arrigo Frisiani, Luigi Gilli, *Introduzione alle reti logiche*
8. Umberto Bussolati, Giancarlo Martella, Carlo Petronio, *La sicurezza nei sistemi informativi*
9. Giampio Bracchi, Gianmario Motta, *Sistemi informativi e imprese*