Consiglio Nazionale delle Ricerche

ISTITUTO DI ELABORAZIONE
DELLA INFORMAZIONE

PISA

LINKER P6060: UN LINKAGE EDITOR PER UN ASSEMBLER RILOCABILE

A. Andronico, B. Balatresi, B. Cipolla
P. Stiavetti, O. Stock

Nota interna B79-31

LINKER P6060: Un linkage editor per un assembler rilocabile.

A. Andronico (*)
B. Balatresi (**)
B. Cipolla (**)
P.Stiavetti (*)
O. Stock (**)

(*+ Istituto di Elaborazione dell'INformazione - C.N.R. - PISA

(**) Consulente Olivetti nell'ambito della Convenzione Olivetti-CNR.

2 - Posizione del problema

Assumiamo come primitivo il concetto di azione nel senso di considerarla un qualcosa che si compie su un universo di oggetti ben definiti, ha durata finita e produce un effetto (risultato) definito.

Assumiamo anche l'esistenza di un qualche linguaggio L atto a descrivere azioni e sia A un interprete di L.

Quando L è un linguaggio di programmazione un insieme finito S di descrizioni di azioni in L lo diremo "pezzo di software" e A sarà quindi una macchina.

La risoluzione di un problema P mediante elaboratore passa attraverso la stesura e l'uso di $k \geq 1$ pezzi di software S_i, $i = 1, 2, \ldots, k$.

Non esiste difficoltà di principio per cui gli S_i siano scritti in più di un linguaggio di programmazione. A seconda dei punti di vista e del tipo di programmazione (sequenziale, concorrente, ecc...) gli S_i si identificheranno con sottoprogrammi, procedure, moduli ecc.

Intuitivamente l'insieme $\Pi = \{S_i,\ i=1,2,..k\}$ degli S necessari e sufficienti a risolvere P lo diremo programma.

Per ogni S risultano definiti due insiemi N_x e N_y, i cui elementi sono detti "nomi", e simboleggiano rispettivamente dati e risultati delle azioni di S. Inoltre con N_s indicheremo l'insieme dei possibili identificatori degli elementi di S.

L'insieme $N = N_x \cup N_y \cup N_s$ viene detto spazio logico di S.

L'insieme M delle locazioni di memoria in cui le informazioni possono effettivamente venire immagazzinate si chiamo spazio fisico.

La applicazione $r: N \rightarrow M$ viene detta legamento degli indirizzi, e l'insieme $r(N)$, immagine di N tramite r, si dice spazio fisico di S.

Dato S siano $s_1, s_2, \ldots, s_n$, $n \geq 1$ le azioni che lo compongono e supponiamo che esistono in S due azioni privilegiate dette rispettivamente iniziale e finale. Non si perde di generalità assumendo che l'azione iniziale sia s_1 e quella finale s_n.

Diciamo che s_i è la successiva di s_j se l'interpretazione di s_j avviene immediatamente prima di s_i.

Una successione $C = \{ s_{i_1}, s_{i_2}, \ldots s_{i_k}, \ldots\ldots\ldots\ldots\ldots \}$

di azioni di S tali che $s_{i_1} = s_1$ e s_{i_j} è successiva di $s_{i_{j-1}}$ $(j > 1)$ si dice catena su S. Se C è finita e il suo ultimo elemento coincide con s_n allora diremo C catena di interpretazione di S.

Diciamo che s^* è azione di chiamata se essa denota l'inserimento di una catena.

Dati S_i, S_j diciamo che S_i chiama S_j e scriviamo $S_i \prec S_j$ se esiste una C_i catena di interpretazione di S_i contenente almeno un'azione di tipo s^* che fa inserire in C_i una qualche catena C_j di S_j.

Nel caso generale ($i \neq j$) affinché l'azione di chiamata sia effettuabile è necessaria una preventiva combinazione degli spazi logici di S_i e S_j in un unico spazio logico composto S(i,j). Questa operazione di combinazione (legamento) di spazi logici si chiama linking.

Linking di un $\tilde{\Pi} = \{ S_i, i=1, \ , k \}$ è dunque l'operazione che combina (lega) i singoli spazi logici degli S_i in un unico spazio logico di $\tilde{\Pi}$; essenzialmente ciò corrisponde a fissare le loro posizioni relativamente a un'arbitraria base comune.

In generale il linking può avvenire in una, o tra due successive, delle operazioni basilari della programmazione: codifica del programma, traduzione, caricamento in memoria centrale,

esecuzione. Viene qui considerato il caso del linking successivo alla traduzione e precedente al caricamento in memoria centrale. Ciò presenta i seguenti vantaggi:

a) flessibilità, data della indipendenza dalle altre fasi,

b) possibilità di introdurre un singolo pezzo di software,

c) possibilità di legare dei pezzi di software "di libreria", ossia messi a disposizione dal sistema,

d) possibilità di segmentare il programma Π , in modo da avere in memoria a ogni istante dell'esecuzione solo la parte necessaria.

Quest'ultimo punto può essere realizzato come funzione secondaria del dispositivo che opera il linking. Precisiamo i termini qui di seguito. E' caratteristico dei moderni sistemi operativi avere un Overlay Supervisor, programma preposto a gestire un'area di memoria (memoria di overlay), in cui vengono, a tempo d'esecuzione, caricate parti di programma, necessarie solo in particolari momenti e non nell'intero corso dell'esecuzione; una volta "eseguita" la parte necessaria del pezzo di software T caricato per ultimo, la relativa area di memoria overlay viene resa disponibile per altri pezzi di software. I pezzi di software T che vengono caricati nella memoria di overlay, verranno detti pezzi di software overlay. I pezzi di software che invece restano in memoria centrale per tutto il corso dell'esecuzione di vengono detti residenti.

E' quindi realizzabile che il dispositivo operante il linking, operi anche una segmentazione del programma $\Pi = \{S_i,\ i=1, \ldots, k\}$ in un insieme di pezzi di software $\{T_j,\ j=1, \ldots, n\}$ in cui ogni pezzo di software T_j è specificato essere di tipo overlay o residente. Ciò, sensatamente, può avvenire guidato dall'utente attraverso una serie D

di statements di controllo della segmentazione.

In questa situazione complessa la funzione r di legamento di N, spazio logico di al suo spazio fisico M viene radicalmente cambiata, diventando funzione parametrica, ad ogni istante, delle azioni di fin qui eseguite e della segmentazione (in overlay e residenti) di e operando effettivamente ad ogni istante solo su un sottoinsieme di N.

3 - Schema generale di progetto

3.1 - L'ambiente P6060

Diciamo macchina base P6060 e la indichiamo con M_o l'insieme composto da una unità centrale e una unità Floppy Disk (FDU).

Per la modularità del sistema si possono definire le seguenti configurazioni Hardware:

1 - $M_1 = M_o + FDU$

2 - $M_2 = M_o + DCU^{(*)}$

3 - $M_3 = M_1 + DCU$

Ovviamente $M_o \subset M_1 \subset M_2 \subset M_3$.

Dal punto di vista software gli ambienti da considerare sono due: Compilazione (C) e Esecuzione (E).

Poiché ciascuno di questi ambienti può essere applicato a Assembler (A) o Basic (B) si hanno i quattro ambienti seguenti: CA, CB, EA, EB per cui si può avere lo schema seguente

Fig. 1

(*) Disk Control Unit

Per le configurazioni M_o e M_1 è esclusa la convivenza Basic e Assembler per cui sul grafo di Fig. 1 i percorsi sono i seguenti:

a) I, CA, EA
b) I, CB, EB
c) I, EA
d) I, EB

Il LINKER P6060 interviene nei percorsi a) e c).

Per le configurazioni M_2 e M_3 si possono costituire raggruppamenti di ambienti operativi nei quali sia possibile la convivenza Assembler e Basic e che includono come sottocasi a), b), c), d).

Lo schema generale di tali raggruppamenti sarà del tipo:

$I, CX_1, CX_2, \ldots, CX_k, EX_1, \ldots EX_k$

dove X_i = A oppure B; $(i = 1, 2, \ldots, k)$.

3.2 - Strutture fondamentali interessanti Linker P6060

Definiamo qui gli elementi fondamentali di cui il LINKER P6060 fa uso.

3.2.1 - Modulo

L'elemento più semplice a cui fa riferimento il LINKER P6060 è il modulo.

Poichè un modulo, nella forma utile al LINKER, è una entità che è già stata trattata da un compilatore (Assembler, Basic o altro), in quanto segue esso sarà denominato Modulo Oggetto (e verrà indicato genericamente con M). La struttura del generico M è quella di figura 2.

Fig. 2

3.2.2 - Segmento

Diciamo segmento Q un insieme h $\geq$ 1 finito di moduli che rappresenti logicamente la più piccola unità di trasferiemnto di enti eseguibili. Un tale elemento può essere considerato ai fini di eventuali overlay di memoria centrale. La struttura del generico Q è quella di figura 3.

Fig. 3

3.2.3 - I Files nel Sistema P6060

Nel sistema P6060 un "file" è un corpo, estendentesi su di un numero intero di settori, preceduto da un "file header" che contiene informazioni necessarie al sistema quali l'allocazione del file in memoria secondaria, il tipo di file ecc.

I tipi di files interessanti LINKER P6060 sono:

1) Files di tipo TEXT
2) Files di tipo OBJECT
3) Files di tipo EXECUTE

Un file di tipo text ha il corpo costituito da un numero qualsiasi di record con i primi 4 caratteri numerici e i restanti 76 appartenenti al set ISO.

Un file di tipo object presenta come corpo un modulo oggetto.

Un file di tipo execute ha per corpo una struttura del tipo specificato in figura 4.

Fig. 4

dove

APD : Assembler Program Directory, contiene informazioni relative alla struttura del file execute.

$SEG_1 \ldots SEG_n$: Segmenti costituiti dal Linker.

SEGTAB : Tabella dei segmenti, contiene l'OFFSET dei segmenti relativo al primo byte del file execute espresso in numero di blocchi di k byte dove k è detto fattore di allineamento e viene specificato in input.

OD : Object Dictionary, è un particolare segmento che viene incluso,(su richiesta) e contiene, per ogni modulo incluso da Linker nel modulo eseguibile, il nome ed il suo indirizzo, espresso come numero di segmento ed OFFSET all'interno del segmento.

3.3 - Funzioni e Struttura di Linker P6060

Linker P6060,utility fornita insieme al sistema operativo, svolge le funzioni descritte in generale nel paragrafo 2.I "pezzi di software" di tipo S, ivi descritti, si identificano con "moduli", e quelli di tipo T con "segmenti".

Esso, dunque:

A) raggruppa un insieme di moduli object in segmenti secondo statements di controllo (Direttive cfr. 3.4) forniti dall'utente su di un file di tipo text.

B) risolve gli indirizzamenti tra moduli provenienti da diverse compilazioni.

C) costruisce con i segmenti così ottenuti e con eventuali segmenti di sistema, richiesti nelle direttive, un file execute.

Lo schema di base del funzionamento del linker è in figura 5.

Il blocco 5 sta ad indicare che durante il processo di linking, il linker può interagire con l'utente. Il blocco 7 specifica che su stampatrice appaiono informazioni standard ed opzionali riguardanti il processo di linking:

Messaggi di errore
Listing
XREF TABLE
etc.

Fig. 5

4 - Modalità d'uso di Linker P6060

4.1 - Direttive

Attraverso statements di controllo detti "direttive" l'utente definisce il formato dei segmenti che dovranno costituire il programma eseguibile e la struttura di questo (corpo di file execute). Le direttive sono preparate dall'utente in un file di tipo text.

Nei paragrafi seguenti sono definiti i tipi di direttive e la sintassi relativa.

4.1.1 - Direttiva di Definizione di Segmenti Utente

$$\left\{\begin{matrix}O\\R\end{matrix}\right\}[num]: extname_1\ (intname_1)\ [*],\ldots,extname_n(intname_n)\ [*]$$

dove

O = Overlay

R = Residente

indicano come il segmento deve essere caricato in memoria centrale

num : numero intero compreso fra 15 e 254 indica il numero di segmento. Se num è omesso si assume automaticamente il primo numero non già impegnato e maggiore uguale al valore base definito dalle eventuali direttive di origine numerazione segmenti (cfr. 3.4.2).

$extname_i$:name del file che contiene il modulo oggetto

$intname_i$:nome del modulo oggetto

:indicazione di priorità in caso di modulo multiplato (multiplato è detto un modulo che compare in più di un segmento.)

La selezione delle occorrenze del modulo multiplato in caso di riferimento da moduli esterni, avviene secondo i seguenti criteri:

1) Se esiste una occorrenza del modulo multiplato nello stesso segmento a cui appartiene il modulo chiamante, viene utilizzata questa occorrenza.

2) Se non è verificato il caso 1) viene selezionata l'occorrenza del modulo per la quale è stata data una segnalazione di priorità o, in caso di mancata segnalazione, viene considerata la prima occorrenza del modulo nella definizione delle direttive.

4.1.2 - Direttiva di definizione dell'origine di numerazione dei segmenti

Ha il formato:

B:num

num : intero compreso tra 15 e 254 che viene assunto come origine per la numerazione dei segmenti rilocabili.

In un file di direttive può comparire un numero qualsiasi di direttive di tipo B. Ciascuna di queste rimane operativa per i segmenti successivi di tipo utente, per i quali non sia stato definito il campo num, fino alla successiva eventuale direttiva B.

4.1.3 - Direttiva di inclusione segmenti di sistema

Ha il seguente formato:

D: mod_1, mod_2, mod_3,....., mod_n

I moduli mod_i, sono moduli di sistema.

A seguito di questa direttiva il linker sviluppa automaticamente la inclusione, nel file execute, dei segmenti di sistema che contengono i moduli specificati.

I numeri di tali segmenti non possono essere utilizzati per segmenti utente.

4.1.4 - Direttiva di inclusione di mappa logica

Ha il formato:

M:num

dove

num : numero del segmento in cui si vuole che la mappa logica dei moduli all'interno di corpo di file execute si trovi.

4.1.5 - Direttiva di definizione dell'entry point

Ha il formato:

E:mod [(seg_n)]

dove

mod : nome del modulo a cui deve essere passato il controllo quando il file execute va in esecuzione.

seg_n: (opzionale) indica il segmento su cui si trova il modulo.
E' utile in caso di modulo multiplato. In tale caso, se seg_n non è specificato si intende il primo riferimento a mod indicato nelle direttive.

4.2 - Chiamata

Per le configurazioni Hardware definite in 3.1 prive di DCU il LINKER viene richiamato dal comando:

EXEC LNK, IN = [SYSLIB / USLIB], $filename_1$, OUT = [SYSLIB / USLIB], $name_2$ [,ALG=k] [,SYSID=sysid] [, XREF / NOXREF] [, LIST / NOLIST] [, OD / NOOD]

dove:

IN = ... : indica il disco su cui risiede il file di direttive di nome "$filename_1$" (INPUT)

$filename_1$: file di tipo text a partire dal quale si ottiene il file di nome "$filename_2$".

OUT = ... : specifica il disco su cui allocare il file di nome "$filename_2$" (OUTPUT)

$filename_2$: file di tipo execute

k : 8, 16, 32 .... indica il fattore di allineamento dei blocchi logici del file execute.
Per default k = 128

SYSID : identificatore di sistema

XREF : opzione che permette di ottenere la stampa della CROSS REFERENCE TABLE

LIST : opzione che permette di ottenere la stampa della lista della struttura del corpo del file execute

OD : opzione mediante la quale l'OBJECT DICTIONARY viene incluso nel file execute.

I parametri sottolineati sono quelli assunti per default: SYSLIB, NOXREF, NOLIST, NOOD.

5 - Metodologia di realizzazione

Per lo sviluppo del progetto di LINKER P6060 si è usato un metodo di tipo top-down articolato su cinque livelli e precisamente:

5.1 - Livello 1:

Definizione delle funzioni svolte da LINKER P6060 (si possono assumere quelle di paragrafo 3.3).

5.2 - Livello 2:

Definizione dei passi da compiere per sviluppare LINKER P6060 dove ciascun passo si configura come una "scatola nera": assumiamo come definizione dei passi la seguente:

a - Analisi del comando
- a.1 - Analisi sintattica del comando
- a.2 - Le informazioni vengono memorizzate in una tabella.
- a.3 - Analisi semantica del comando. Tutti gli errori provocano la terminazione.

b - Analisi delle direttive

b.1 - Si carica il file delle direttive e si fa l'analisi sintattica e una prima analisi semantica delle direttive, mettendole in forma codificata in tabelle.

b.2 - Viene attribuita la numerazione.

b.3 - Viene controllata la coerenza dei moduli multiplati.

b.4 - Viene controllata l'esistenza dei file object, accedendo ai directory (unico accesso).

b.5 - Acquisizione delle lunghezze dei segmenti di sistema.

b.6 - Acquisizione delle lunghezze di codice dei moduli. Accessi di files, controlli e memorizzazione delle informazioni in una tabella.

c - Valutazioni preliminari al linking

c.1 - Valutazione delle dimensioni dei segmenti, del file execute e verifiche.

c.2 - Allocazione del file execute.

d - Linking

d.1 - Costruzione di tabelle da inserire nel file execute.

d.2 - Costruzione dei segmenti, secondo le direttive. Risoluzione delle indirettezze e memorizzazione dei segmenti nel file execute.

d.3 - Generazione e stampa delle tabelle opzionali.

e - Si passa il controllo al monitor.

5.3 - Livello 3

Vengono definite a parole le funzioni dei vari passi individuati a livello 2, si specificano le strutture dei dati e dei risultati parziali e/o totali denominando quelli fondamentali;

definizione parallela del trattamento degli errori.

A titolo d'esempio vediamo lo sviluppo del punto b del livello 2 dove per i nomi simbolici introdotti si veda Appendice A.

b - Analisi delle direttive.

b.1 - Analisi sintattica e semantica (parziale) delle direttive. Si carica il file delle direttive.

L'analisi sintattica di ogni direttiva è effettuata chiamando per ogni direttiva SYNCHK (Syntax Checker).

Direttiva tipo 0 o R: si creano tante righe di TASFLE quanti sono i moduli, si riempiono i campi: NULINEA (numero linea di inizio direttiva), MODNAME, FILENAME, LIBID (identificatore di libreria), i campi di TINYINF relativi a OVRES,ASTER , PASTER, LAST, TIPO (per informazioni su questi campi, vedere l'allegato sulle tabelle).

Se num è definito, viene trascritto in SEGNUM e viene memorizzato in SEGTB1 il puntatore a TASFLE.

Direttiva tipo D: Si riempiono K righe di TASFLE guardando sul SOD e riempiendo le relative posizioni di SEGTB1 con puntatori a TASFLE e un bit per segnalare che il segmento è di sistema

1) MODNAME con il nome del modulo.
2) SEGNUM con il numero di segmento.
3) FILEXIST viene messo ad 1 ad indicare che il file esiste.

Viene inoltre memorizzato nel campo MODLEN che è sovrapposto a parte di FILENAME (vedi tabelle), il displacement del modulo espresso in bytes. Successivamente vengono scritte in TASFLE le informazioni relative agli altri moduli citati nella direttiva e appartenenti quindi allo stesso

segmento.

Se nella direttiva sono citati moduli di sistema che sono già apparsi in precedenti direttive di tipo D, questi non sono ritrascritti.

Direttiva tipo M: il valore di num viene salvato nel campo MAPPA. Se tale campo ha il valore 0 allora la mappa non è stata richiesta.

Direttiva tipo E: il nome del modulo viene messo nel campo MENTRYP (Module ENTRY Point), il numero del segmento in cui è l'ENTRY POINT viene salvato nel campo SENTRYP (Segment ENTRY Point).

Se non esiste nessuna direttiva di tipo E nel file delle direttive, in MENTRYP viene messo il nome del primo modulo citato nella prima direttiva.

b.2 - Numerazione dei segmenti

L'algoritmo di numerazione attribuisce i numeri ai segmenti che ne sono privi tra due direttive B. L'algoritmo esclude i segmenti per i quali è già stato definito NUM e non dà errori nel caso che due direttive di tipo B siano date conseguentemente o che partano dallo stesso segmento. In questo caso ha valore l'ultimo. Durante la numerazione, (se non esisteva una direttiva di tipo E), viene riempito il campo SENTRYP per l'entry point.

b.3 - Controllo delle molteplicità.

Si considera la colonna degli asterischi, si controlla che due moduli asteriscati . si controlla che due moduli asteriscati non abbiano lo stesso nome.

Predisposizione per la ricerca futura, per ogni modulo asteriscato si cerca la prima occorrenza e si setta il campo PASTER. Se la prima occorrenza è quella con

l'asterisco non si setta.

b.4 - Ricerca dei file object in Directory e memorizzazione degli indirizzi

Si carica in core il file directory: per ogni pezzo, per i moduli di tipo utente, si controlla la presenza dei filenames di TASFLE, tra i file object; si sostituisce l'indirizzo del file in FILENAME, campo FILEAD.

Si indica con l'accensione del bit FILEXIST l'avvenuto ritrovamento del file. Alla fine si deve avere una colonna di tutti 1. Viene anche indicata in VOLID l'Unità fisica.

b.5 - Acquisizione delle lunghezze dei segmenti di Sistema

Si accede a SEGTAB e si calcola (origine + dato di SEGTAB) l'indirizzo dei segmenti di sistema e la si mette in FILEAD. Si sistema in VOLID l'identificatore relativo al disco di sistema. Si segna se O/R in TASFLE.

In SEGTAB si mette la lunghezza (calcolata da SEGTAB) dei segmenti di sistema.

b.6 - Acquisizione delle lunghezze di codice dei moduli.

Per ogni File Object interessato: si accede al file.
Si memorizza il FAT nell'AREAFAT della riga di TASFLE.
Si controlla l'esistenza del modulo indicato in TASFLE.
Si prende la lunghezza A-C (codice + indirettezze) e la si riporta in TASFLE nel campo MODLEN.

5.4 - Livello 4:

Sviluppo in linguaggio tipo Algol delle procedure associate ai passi descritti al livello 3 introducendo:

a - le variabili in forma simbolica necessaria alla descrizione dell'algoritmo.

b - Le funzioni di utilità autonome o connesse a più procedure.

c - tutte le funzioni di sistema necessarie alla esecuzione dell'algoritmo.

d - commenti atti a documentare lo svolgimento della procedura.

In Appendice B è riportato un elenco delle procedure fondamentali. A titolo di esempio e ricollegandoci allo sviluppo del punto b del livello 3 riportiamo la struttura e la descrizione della procedura SISED come ulteriore raffinamento del punto b.1.

La procedura SISED richiama le procedure:

GETDIR (Acquisizione di una direttiva)
ONESID (Analisi sintattica e semantica parziale di una direttiva)
MORDIR (Verifica se il buffer delle direttive contiene ancora direttive complete, dopo l'ultima esaminata)
ORSISD (Analisi di un campo di direttiva di tipo O/R)
DSISD (Analisi di un campo di direttiva di tipo D)
BSISD (Analisi di direttiva tipo B)
MSISD (Analisi di direttiva tipo M)
ASISD (Analisi di direttiva tipo E)
FUSOD (Determinazione del segmento di appartenenza di un modulo di sistema)

e inoltre TERMIN e ERRORI citati in Appendice B.

La figura 6 mostra le connessioni fra SISED e le procedure ad essa collegate.

Fig. 6

Lo sviluppo di SISED in linguaggio tipo ALGOL è riportato qui di seguito:

```
Procedure SISED;

begin IOFILE (1, FATAD, VOL, indirizzo (indirizzo file) / * Dummy
            FAT *, FATINFH, FATL; /* Carico il FAT del file direttive */
      PDIRFILE: = 0;
      PTASFLE: = "inizializzato";
      SODIN: = 0;                /* serve solo al FUSOD. Indica che il SOD
                                    non è ancora in core */
      While "non s'è raggiunto EOF"
          do begin IOFILE(1,DIRBUF,VOL,FATAD,PDIRFILE,LDIRBUF).
                   PDIRBUF: = 0;
                   NEXTEND: = 1;
                   While NEXTEND = 1   /* Ci sono direttive intere nel
                      buffer da controllare (vedi MORDIR)*/
                      do begin GETDIR;
                               if ERRFOUND = 0
                                 then ONESID  /* Si può passare a ONESID
     solo se GETDIR non ha dato error. Se no PDIRBUF:=fine direttiva*/
                                 else begin MORDIR;
                                            PDIRBUF:=PNEXTEND;
                                            ERRFOUND:=0;
                                      end;
                                      if DIRBUF(PDIRBUF+1)=EOF
                                         then "uscita"
                                         else MORDIR /* serve per il test
                                                        del while */
                           end;
                   PDIRFILE:=PDIRFILE+PDIRBUF /* Caricherò così a partire
    dall'inizio della prima direttiva non completa che già si trovava in
    DIRBUF (vedi MORDIR) */
              end;
end.
```

5.5 - Livello 5:

Codifica in linguaggio Assembler P6060 mantenendo sia nomi simbolici di variabili e di funzioni definiti a livello 4 come pure tutti i commenti a scopo di documentazione.

Tra il livello 3 e il livello 4 sono stati inoltre sviluppati alcuni strumenti atti alla comparazione fra i due livelli.

La determinazione dei tipi di errori (vedi Appendice C) fa parte di tali strumenti.

6 - Un esempio

Consideriamo i moduli di tipo object A,B,C,D e siano dati le relazioni seguenti:

A $\prec$ B , A $\prec$ C , A $\prec$ D, D $\prec$ C, B $\prec$ SISMOD (SISMOD = modulo di sistema).

Tali relazioni sono messe in evidenza nella figura 7, 8, 9, 10, dove la struttura è quella di figura 2

Fig. 7
Fig. 8
Fig. 9
Fig. 10

Siano inoltre AFILE, BFILE, CFILE, DFILE e FILEC i nomi dei file utente contenenti i moduli dati e di C ne esistano due versioni in CFILE e FILEC.

Mostriamo la costruzione di un file execute, diciamolo EXAMP, con l'uso di LINKER P6060,avendo definito il seguente file DIR di direttive:

```
B : 33
R : AFILE(A)
O : BFILE(B), CFILE(C) *
O(56) : DFILE(D), FILEC(C)
E : A(33)
D : SISMOD
```

Il comando

```
EXEC LNK, IN = USLIB, DIR, OUT = USLIB, EXAMP
```

assumerà DIR come file direttive e produrrà il file EXAMP, il cui contenuto è quello di figura 11, dove testata di segmento è una coppia (n_s, t_s) con n_s = numero di segmento, t_s = lunghezza del segmento arrotondata a un multiplo di k, che occupa uno spazio costante h.

Fig. 11

Per indicare una testata di segmento scriveremo T_{n_s}.

Osserviamo che i campi indirettezze sono risolti in tutti i moduli, e che il corpo del file è strutturato in quattro segmenti di numero rispettivamente 33, 34, 56, 81 (ove i primi due sono determinati dalla direttiva B, il terzo è dato esplicitamente e il quarto corrisponde all'effettivo numero del segmento di sistema cui appartiene SISMOD).

Il segmento 33 contenente il modulo A è residente come da direttiva e i segmenti 34 e 56 (contenenti rispettivamente i moduli B, C e D, C) sono overlay. Il segmento 81 è residente in quanto specificato tale nel sistema e di questo è usato solo il modulo SISMOD dei due (SISMO e SISMOD) presenti nel segmento considerato.

Si noti che dei due moduli C (contenuti in FILEC e CFILE) per effetto dell'asterisco, pur essendo presenti in EXAMP entrambi, nel caso generale l'indirettezze vengono risolte rispetto al modulo asteriscato (cfr. 4.1) tranne nel caso di appartenenza di un modulo multiplato al segmento del modulo chiamante.

Nella tabella APD un campo indicherà che il modulo A del segmento 33 verrà considerato modulo iniziale come indicato nella direttiva E. Si noti che questa direttiva è inutile: lo stesso risultato, in questo caso, si ottiene per default.

B I B L I O G R A F I A

1. Andronico,A. "Un metodo per l'ordinamento dei sottoprogrammi e l'ottimizzazione della assegnazione della memoria in un programma assemblatore", Calcolo Vol. 5, N. 3, Luglio 1968.

2. Andronico,A. , Balatresi,B., Cipolla,B., Stiavetti,P., Stock,O. "Specifiche funzionali del Linker P6060", Olivetti 1978.

3. Andronico,A. , Balatresi,B., Cipolla,B., Stiavetti,P., Stock,O. "Specifiche di progetto del Linker P6060 - Olivetti 1978.

4. Presser,L., White,J., "Linkers and Loaders", Computing Surveys, Vol. 4, N. 3, Settembre 1972.

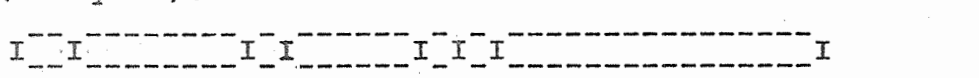
5. Wirth,N. "Systematic Programming: An Introduction", Prentice-Hall, 1973.

Appendice A

TABELLE UTILIZZATE DA LINKER-P6060

1) TASFLE:

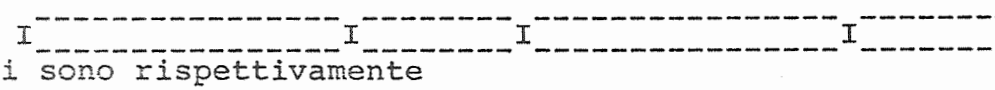
a) Tabella Struttura File execute (TASFLE), può contenere da 200 (versione minima) a 600 (versione massima) linee con questo formato (in bytes):



NOMI E CONTENUTI DEI CAMPI:

1) NULINEA: (2 bytes) contiene il numero della linea della direttiva,
2) MODNAME: (8 bytes) contiene il nome del modulo
3) SEGNUM: (1 byte) contiene il numero del segmento
4) FILENAME: (6 bytes) contiene il nome del file e poi altri dati (*)
5) LIBID: (1 byte) contiene l'identificatore di libreria (**).
6) TINYINF: (1 byte) contiene informazioni varie, vedi (***).
7) AREAFAT: (16 bytes) file allocation table.

(*) Il tracciato successivo del campo è (in bits):



dove i campi sono rispettivamente

1) MODLEN: 2 bytes che contengono la lunghezza del modulo (espressa in bytes).
2) Campo non utilizzato lungo 16 bits.
3) FILEAD: 2 bytes per identificare l'indirizzo del file.
4) VOLID: 1 byte per identificare il volume.

(**) Nel caso DTM ci sarà una tabella di decodifica.

(***) Il campo TINYNF contiene otto bits che servono rispettivamente a:

1) Campo PASTER : se ad uno indica che esiste un modulo "prioritario" con quel nome.
2) Campo ASTER: indica che il modulo in oggetto è prioritario.
3) Campo TIPO: indica se il modulo è utente (^{0}) o sistema (1).
4) Campo OVRES: indica se il modulo è overlay (0) o residente (1).

5) Campo LAST: indica che il modulo è l'ultimo del segmento attuale.
6) Bit non utilizzato.
7) Bit non utilizzato.
8) Bit FILEXIST: indica (quando si è fatta la ricerca sulla esistenza dei files) se il file esiste (1) o no (0), la colonna FILEXIST di TASFLE deve essere tutta ad 1.

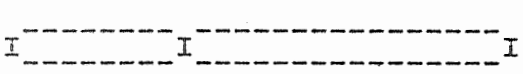
2) SEGTB e SEGTB1:

Le Tabelle SEGTB e SEGTB1 sono costituite ciascuna da 256 elementi di due bytes, il loro formato (in bits) è:

Dove i campi hanno rispettivamente i significati:

1) Primo bit: Per SEGTB indica se il segmento è overlay (0) o residente (1), per SEGTB1 indica se il segmento è di sistema (1) o no (0).
2) Restanti 15 bits: per SEGTB1 contengono un puntatore a TASFLE, per SEGTB contengono prima la lunghezza del segmento (espressa in multipli del fattore di allineamento (ALIGN) poi il displacement (espresso sempre in multipli di ALIGN).
3) TANUM:

La tabella TANUM è costituita da 16 elementi di 3 bytes, ognuno dei quali ha il seguente formato (in bits):



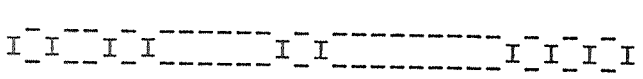
La tabella TANUM serve per memorizzare i numeri di segmento contenuti nelle direttive tipo B fornite dall'utente, il suo primo elemento è inizializzato con i valori 16 e 1, i successivi elementi vengono riempiti con: numero fornito dalla direttiva B e puntatore alla riga di TASFLE (PTASFLE) da cui si deve iniziare a numerare i segmenti partendo dalla base di numerazione fornita dalla direttiva B.

I campi di ogni elemento contengono quindi:

1) BANUM: un byte con la base di numerazione.
2) PNTR: due bytes con puntatore a TASFLE.

4) TABELIN:

La tabella TABELIN contiene, in forma codificata, le informazioni presenti nella chiamata del LINKER; il formato della tabella (in bytes) è il seguente:



I nomi ed i contenuti dei campi sono nell'ordine:

1) campo LIBD, contiene l'indirizzo della libreria in cui è il file delle direttive,
2) campo FILED, contiene l'indirizzo del file delle direttive,
3) campo LIBEX, contiene l'indirizzo della libreria che conterrà il file execute prodotto dal LINKER.
4) campo FILEX, contiene il nome del file execute,
5) campo ALIGN, contiene il fattore di allineamento per il file execute,
6) campo SYSID, contiene il System Identifier,
7) campo XREF, flag che sagnala la richiesta della CROS REFERENCE TABLE,
8) campo LIST, flag che segnala la richiesta della lista, (SI= 1, NO= 0),
9) campo OD, flag che segnala la richiesta dell'OBJECT DICTIONARY.

5) "TABELLE VARIE"

Vi sono all'interno del LINKER, varie altre aree che servono per memorizzare informazioni varie, queste aree vanno sotto il nome di "TABELLE VARIE".

Le aree utilizzate sono le seguenti:

1) BODYLEN: 4 bytes che contengono la lunghezza del corpo del file execute.

2) MAPPA: un byte che contiene il numero del segmento in cui va inclusa la mappa logica, (se MAPPA=0 allora niente mappa logica).
3) MENTRYP: 8 bytes che contengono il nome del modulo scelto come ENTRY POINT (direttiva tipo E, in mancanza di tale direttiva, viene preso come entry point module il primo del file execute).
4) SENTRYP: un byte che contiene il numero del segmento contenente il modulo entry point.
5) WENTRYP: un byte che dice se l'Entry Point è originato da una direttiva di tipo ENTRY (E) o no.

Appendice B

LIVELLO 4 - ELENCO PROCEDURE FONDAMENTALI

1) - "VERTICALI"

LNK

Inizializzazione, chiamata procedure "orizzontali", controllo del flusso.

ERRORI

Stampa dei messaggi d'errore.

TERMIN

Terminazione.

2) - "ORIZZONTALI"

SISEC (KEYBUF, TABELIN)

Analisi sintattica e semantica comando.

SISED (LIBID,FILED,SOD,FILBUF,TASFLE,SEGTB1,TANUM)

Analisi sintattica e semantica parziale direttive.

NUSEG (TASFLE,SEGTB1,TANUM)

Numerazione segmenti.

COMOL (TASFLE)

Controllo molteplicità.

SETFAD (TASFLE,DIRBUF)

Ricerca file object in directory e memorizzazione indirizzo.

SSEGLN (SSEGTAB,SEGTB,TASFLE,SEGTB1)

Acquisizione lunghezza segmenti di sistema.

MODLUN (TASFLE,MODBUF)

Acquisizione lunghezza codice moduli.

DIMCHK (SEGTB,TASFLE,SEGTB1,DIRBUF)

Valutazione lunghezza segmenti e file execute, allocazione file execute.

SETAPD (TASFLE,TABELIN,SEGTB,APDBUF)

Costruzione e memorizzazione APD.

CREAFE (TASFLE,SEGTB1,MODBUF,SEGBUF,SEGTB,TABELIN)

Costruzione segmenti, risoluzione indirettezze, memorizzazione.

PRTOPT (TABELIN,TASFLE,SEGTB)

Generazione e stampa delle tabelle opzionali.

Appendice C

TIPI DI ERRORE

Ci sono quattro diversi tipi di errore:

1) Codice di severità = 16

Questo tipo di errore provoca il rientro immediato al MONITOR.

2) Codice di severità = 12

Questo tipo di errore provoca la terminazione del programma alla fine della fase del corso.

3) Codice di severità = 8

Prosecuzione dell'analisi sintattica e semantica senza costruzione del file execute.

4) Codice di severità = 4

Warnings.

Per gli errori di severità 16 il linker rientra al MONITOR e compare su display l'informazione adeguata ad individuare l'errore.

Gli errori di questo tipo sono i sintattici e semantici del comando e gli overflow di core.

Gli errori di severità 12 si individuano principalmente come errori sintattici o semantici delle direttive.

In questi casi l'analisi prosegue se possibile fino alla valutazione delle dimensioni dei segmenti e del file execute.

In presenza di errori di tipo 8 il linker continua le sue funzioni ma non viene costruito il file execute. Si verifica la correttezza dei riferimenti e si eseguono le stampe opzionali.

Gli errori di severità 4 non modificano il flusso del programma. L'output può anche risultare non eseguibile sulla macchina su cui è stato eseguito il linking.

I messaggi di errore vengono emessi immediatamente dopo essere stati individuati. L'utente può interrompere l'esecuzione premendo il tasto BREAK.

Il test al Break viene fatto all'interno della procedura LNK al termine di ogni routine.

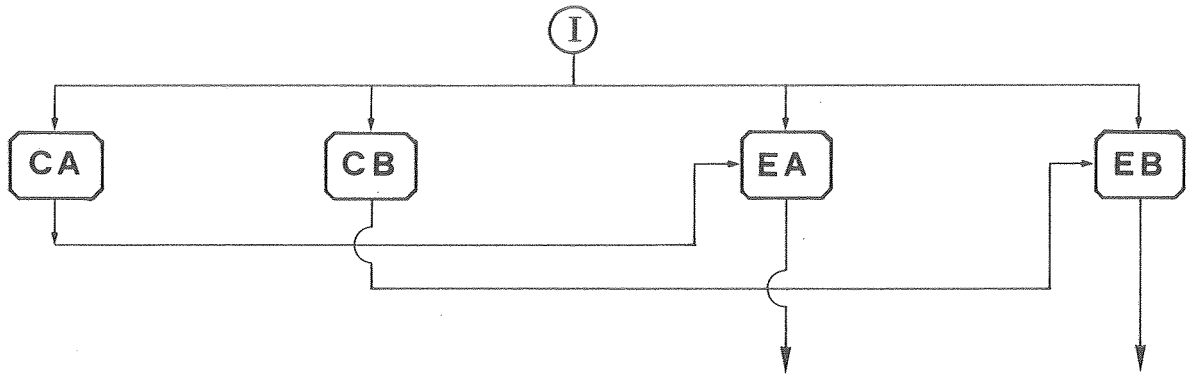


FIG.1

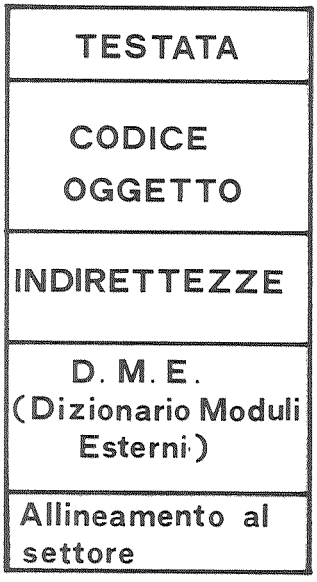

TESTATA
CODICE OGGETTO
INDIRETTEZZE
D. M. E. (Dizionario Moduli Esterni)
Allineamento al settore

FIG. 2

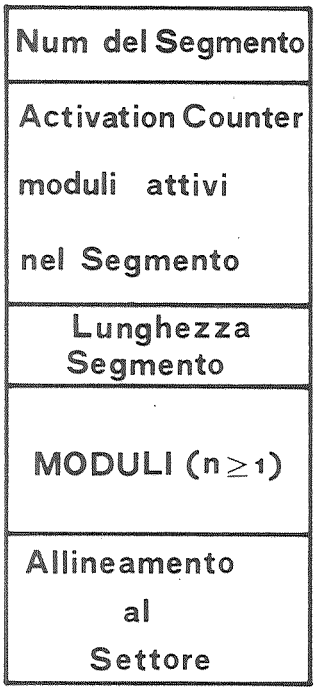

Num del Segmento
Activation Counter
moduli attivi
nel Segmento
Lunghezza
Segmento
MODULI (n ≥ 1)
Allineamento
al
Settore

FIG. 3

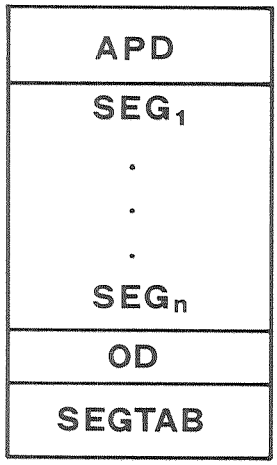


FIG.4

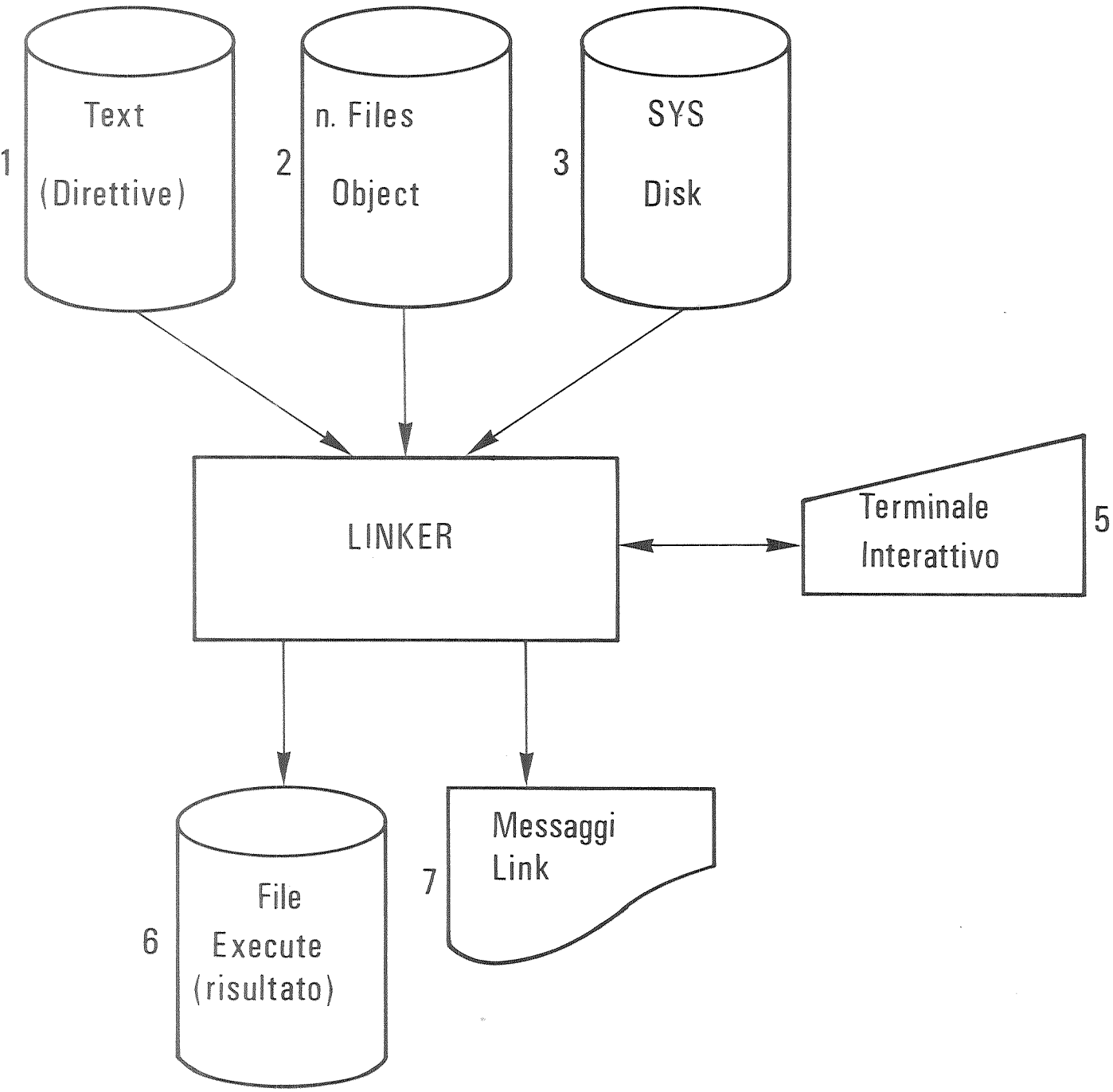

1
Text
(Direttive)
2
n. Files
Object
3
SYS
Disk
LINKER
Terminale
Interattivo
5
6
File
Execute
(risultato)
7
Messaggi
Link

FIG.5

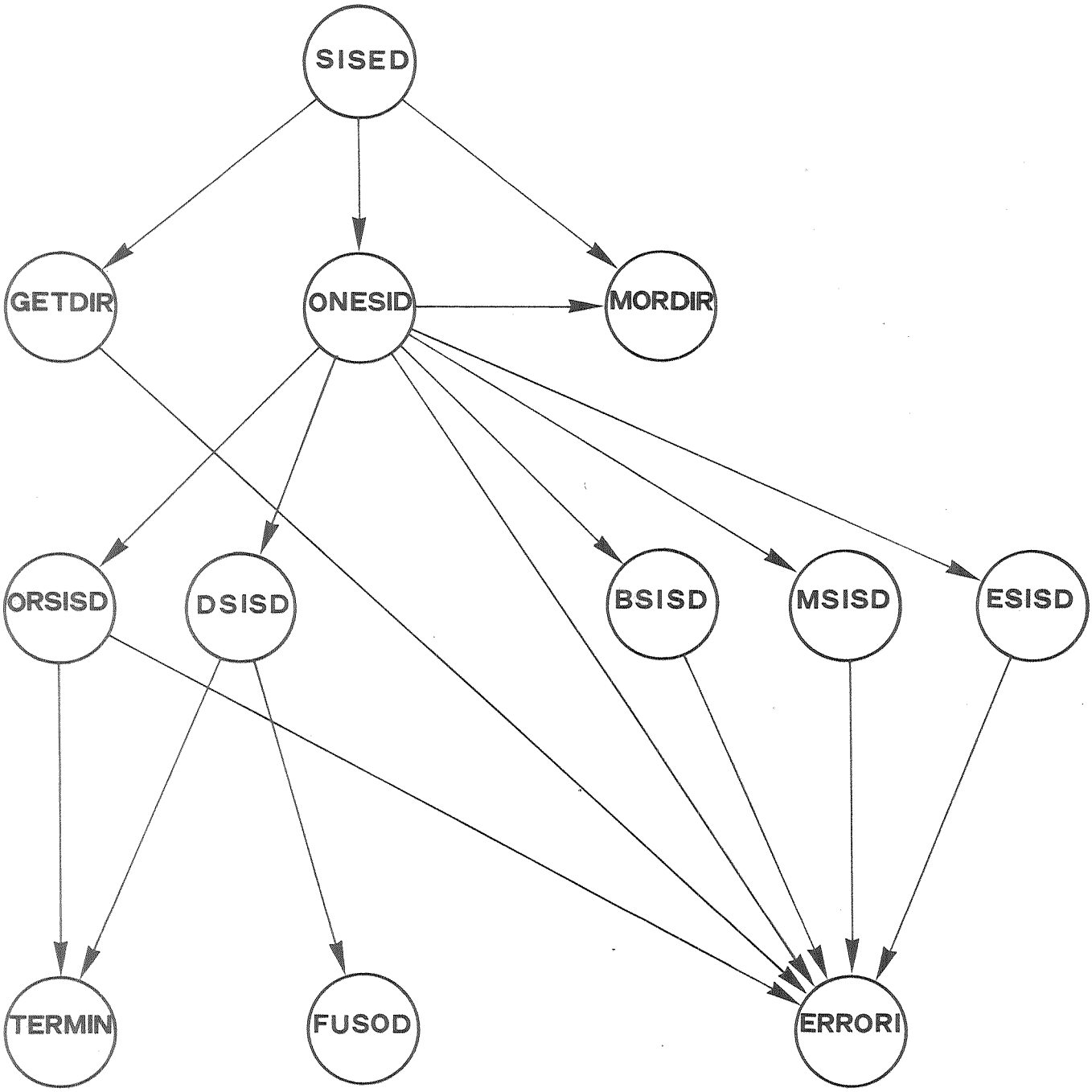

SISED
GETDIR
ONESID
MORDIR
ORSISD
DSISD
BSISD
MSISD
ESISD
TERMIN
FUSOD
ERRORI

FIG.6

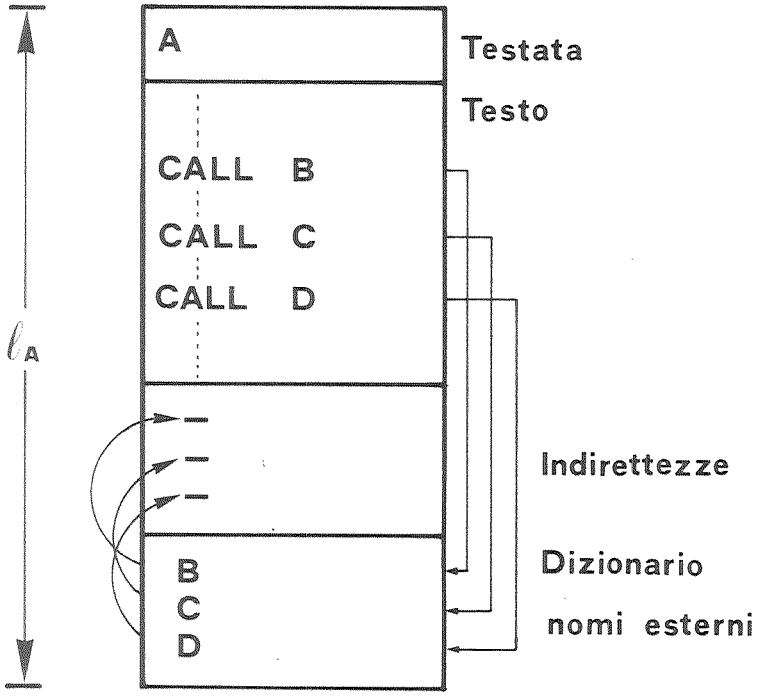

A
Testata
Testo
CALL B
CALL C
CALL D
ℓ_A
Indirettezze
Dizionario
nomi esterni
B
C
D

FIG.7

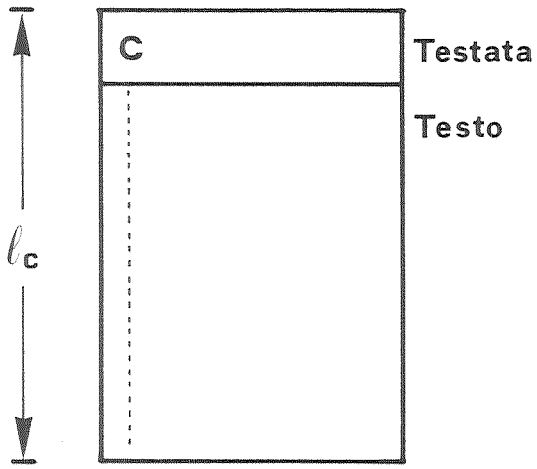


FIG. 9

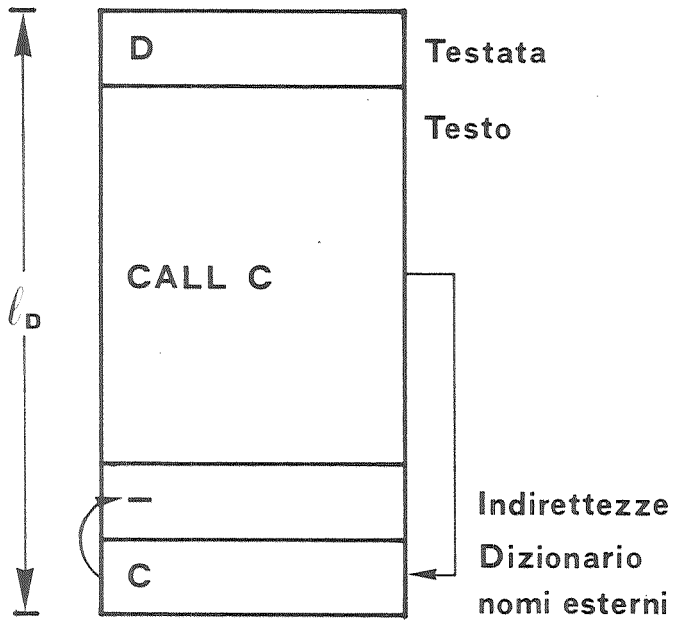
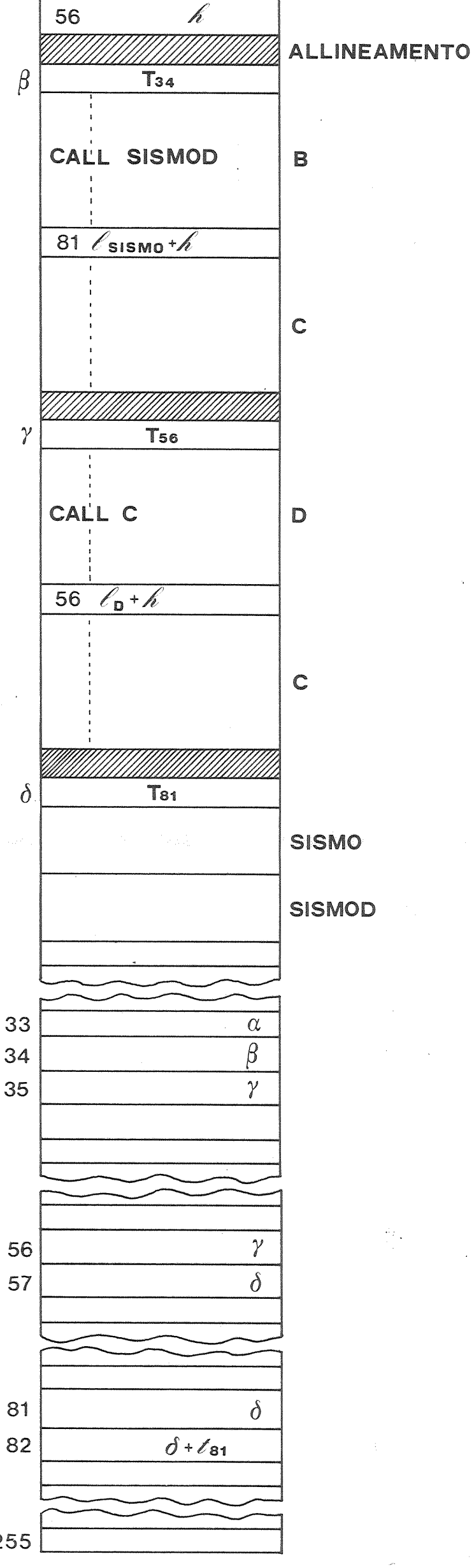

D
Testata
Testo
CALL C
ℓ_D
–
C
Indirettezze
Dizionario
nomi esterni

FIG.10


56 h
ALLINEAMENTO
β
T34
CALL SISMOD
B
81 ℓ SISMO + h
C
γ
T56
CALL C
D
56 ℓ D + h
C
δ
T81
SISMO
SISMOD
33 α
34 β
35 γ
56 γ
57 δ
81 δ
82 δ + t81
255