P6060

Manuale generale del sistema

olivetti

LP Code 3974650 V (0)

PREFAZIONE

Questa pubblicazione ha lo scopo di fornire informazioni sulla struttura di base del Sistema P6060.

Essa è destinata agli utenti che intendono fare un uso sofisticato ed ottimizzato della macchina base e dei supporti periferici, fornendo le informazioni sulle caratteristiche di struttura e di interfaccia fra i componenti del Sistema che approfondiscono quanto riportato sugli specifici manuali utente già emessi.

Riferimenti : P6060 Personal Minicomputer manuale generale (GR codice 3940910 P) - P6060 Opzione Plotter manuale del programmatore (GP codice 3973700 R) - P6060 I/O con periferiche esterne manuale del programmatore (GP codice 3973710 E)

Distribuzione : Su licenza (L)

Prima edizione: Settembre 1978

PUBBLICAZIONE EMESSA DA:

Ing. C. Olivetti & C., S.p.A.
Direzione Marketing Centrale
Servizio Documentazione
77, Via Jervis - 10015 IVREA (Italy)

INDICE

INDICE DELLE FIGURE

1. INTRODUZIONE

Il sistema Olivetti P6060 consiste in un minicomputer programmabile in linguaggio BASIC, orientato alla programmazione scientifica e tecnica. Il sistema è dotato di un software di base efficacemente strutturato, che ne costituisce l'interfaccia accessibile all'utente: questi può perciò utilizzarlo senza doverne conoscere le caratteristiche hardware, colloquiando con il software di base per mezzo di un insieme di comandi ad alto livello.

Il software di base è costituito di un insieme di programmi scritti in un linguaggio comprensibile per l'unità centrale ma non utilizzabile dall'utente. Una parte di tali programmi, e specificamente quella che si occupa della gestione delle risorse del sistema, prende il nome di sistema operativo. Tutte le elaborazioni si svolgono sotto il diretto controllo dell'utente e la supervisione del sistema operativo.

La memoria centrale del sistema è realizzata parte in ROM (Read Only Memory) e parte in RAM (Random Access Memory). Quest'ultima la sola accessibile dallo utente, è destinata a contenere sia programmi applicativi sia opportuni sottoinsiemi del software di base. Dal punto di vista dell'utente, il sistema P6060 è monoprogrammato: istante per istante, infatti, la memoria centrale può contenere, oltre ai programmi di software di base, un solo programma applicativo.

Il sistema P6060 è in grado di registrare gli eventi esterni e di generare interruzioni. E' altresì in grado di isolare e segnalare gli errori che si verificano nell'esecuzione di programmi o di comandi.

Il sistema può essere collegato con numerose unità periferiche. In particolare è in grado di gestire in modo logico un'unità a disco come memoria di massa in linea.

Il software di base è per l'appunto registrato su una memoria di massa, che può essere costituita da floppy disks (FDU), disco a testina mobile (DCU), disco a testine fisse (HDU). Si parla rispettivamente di sistema P6060 FDU, P6060 DCU, P6060 HDU. I sistemi DCU e HDU sono estensioni del sistema FDU; possono pertanto operare in modo identico al sistema FDU.

Dal supporto fisico su cui è registrato il software di base, opportuni suoi sottoinsiemi vengono trascritti in memoria centrale al momento dell'accensione della macchina oppure quando risulta necessario durante la esecuzione delle operazioni richieste.

In questo manuale si descrive il modo in cui il software di base si interpone tra l'hardware e l'utente, realizzando la macchina virtuale sulla quale l'utente opera; in particolare vengono descritti i singoli componenti del sistema operativo e le funzioni da essi espletate.

2. CONFIGURAZIONE HARDWARE DEL SISTEMA P6060

Configurazione base ed estensioni

Il sistema P6060 è composto dalla memoria centrale, da una unità centrale di elaborazione (CPU), da canali e da periferiche di I/O collegate di norma ai canali attraverso unità di controllo; può essere collegato con altri sistemi attraverso canali.

Il sistema P6060 è modulare: partendo da una configurazione hardware minima lo si può estendere e potenziare in vario modo, così da adattarlo alle diverse esigenze dei singoli utenti.

Nella configurazione minima il sistema comprende:

- memoria interna (48K byte di cui 16K byte riservati all'utente)
- CPU
- console
- tastiera
- display
- unità disco (FDU, DCU, HDU)

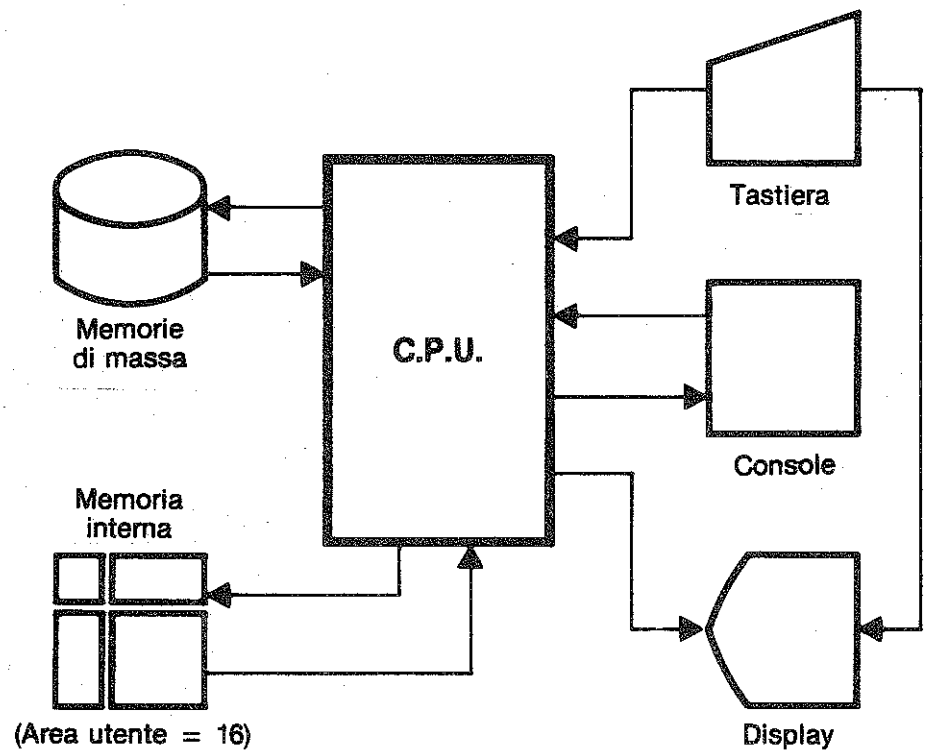


Figura 2-1 Configurazione minima

Il disco contiene il software di base (viene perciò detto disco sistema) e può essere utilizzato come memoria di massa in linea.

Il sistema minimo può essere potenziato con l'ampliamento della memoria interna riservata all'utente (fino a 48K byte) con il collegamento alle seguenti unità periferiche:

- stampante termica

- fino a 2 FDU + 2 DCU oppure 2 HDU

- unità collegate attraverso canale IPSO, oppure EIA RS 232. In questo modo possono essere collegate al sistema P6060 unità di qualsiasi tipo:

  . PN 20 - perforatore di nastro

Nel sistema P6060 sono disponibili 13 posti piastra.

La configurazione base del sistema è così composta:

- tastiera

- console

- stampante

- governo floppy-disk (2 piastre)

- unità centrale

- una piastra di memoria utente (RAM) con capacità variabile tra 16 e 32K byte

Gli ultimi 4 posti-piastra possono essere occupati da altrettanti governi linea, di tipo IPSO, EIA RS 232, Current Loop. Sono ammessi al più 2 governi linea per ciascun tipo. L'ultima piastra può essere occupata indifferentemente da un governo linea o da una piastra di memoria RAM aggiuntiva. Nel caso il sistema possieda 4 governi linea, la memoria utente resta necessariamente limitata a 32K byte.

- LN 20 - lettore di nastro perforato

- CTU 1000 / CTU 1010 - cassetta magnetica

- PR 1220; PR 1230; PR 1240; PR 1350; PR 1370 - stampanti veloci

- SV 160; SV 160/1, Sv 160/2 - stampante ausiliaria

- ICU 600 - interfaccia strumenti di misura

- CR 300 - lettore di schede

- PCU 600 - plotter

- XU 3500 - lettore di banda perforata (codice mnemonico non disponibile)

- UN 812 (NRZ/7; NRZ/9; PHE/9) - unità a nastro magnetico

Le caratteristiche tecniche di ciascuna di esse sono descritte nei rispettivi manuali.

Memoria interna

La memoria interna è un insieme di locazioni direttamente indirizzabili, ad accesso veloce.

Dati e programmi devono venire caricati in memoria interna prima di essere elaborati.

La memoria interna è costituita quasi interamente di RAM (Random Access Memory) e di una piccola parte di ROM (Read Only Memory). La RAM è destinata a contenere il programma utente, i dati da elaborare e le opportune routine di software di base; la ROM contiene soltanto un programma di software di base, il bootstrap, necessario per l'inizializzazione del sistema. L'unità di base indirizzabile della memoria è il byte, che è composto di 8 bit.

Ad ogni byte di memoria è associato un numero progressivo intero positivo, a partire da 0, che individua univocamente il suo indirizzo.

I byte possono essere considerati individualmente oppure raggruppati in campi. Un campo di memoria viene indirizzato attraverso il primo byte del campo.

CPU

La CPU (Central Processing Unit) è il centro di controllo e di elaborazione del sistema. Essa controlla l'esecuzione delle istruzioni, la sequenza in cui tale esecuzione deve avvenire e il verificarsi di interruzioni. La CPU si compone di 2 elementi:

- l'unità di controllo, che esegue i controlli su tutte le operazioni del sistema;

- l'unità aritmetico-logica, che esegue le operazioni aritmetiche ed effettua le scelte logiche tra le varie alternative di esecuzione

Per esemplare la sua attività la CPU utilizza parti di memoria interna (registri, program status word, ecc.).

Operazioni I/O

Le operazioni I/O effettuano il trasferimento di informazioni tra memoria centrale e unità periferiche. Le periferiche collegate al sistema P6060 sono divise in 2 classi:

- periferiche gestite in modo logico

- periferiche gestite fisicamente

Periferiche gestite in modo logico

Le periferiche gestite in modo logico sono indirizzate in modo privilegiato dall'utente per mezzo di comandi o istruzioni BASIC, senza che ne debbano essere prese in considerazione le caratteristiche hardware. Le periferiche gestite in modo logico sono:

- tastiera (può anche essere gestita come periferica esterna)
- console
- display
- avvisatore microacustico
- unità disco (FDU, DCU, HDU)
- stampante termica
- stampante esterna
- video display
- plotter

Periferiche gestite fisicamente

Le periferiche gestite fisicamente sono collegate al sistema P6060 attraverso dispositivi di interfaccia detti canali.

In questo caso, poichè la CPU non è costretta a servire immediatamente le interruzioni di I/O, il sistema può eseguire contemporaneamente istruzioni o comandi ad una o più operazioni di I/O. La gestione di queste periferiche viene effettuata per mezzo di istruzioni di I/O generale. Il collegamento logico tra l'unità centrale e i vari canali è realizzato dal sistema operativo; l'indirizzamento alle periferiche associate ai singoli canali deve essere effettuato esplicitamente, eventualmente per mezzo di opportune istruzioni nei programmi applicativi.

I canali collegabili al sistema P6060 sono:

- interfaccia IPSO

- interfaccia EIA RS 232

- tastiera

3. IL SOFTWARE DI BASE

I componenti

Il software di base comprende tutti i programmi che consentono all'utente di colloquiare in modo semplice, per mezzo di comandi e utilizzando un linguaggio di programmazione ad alto livello (il BASIC), con l'hardware del sistema P6060. Il software di base libera l'utente dalla necessità di fare riferimento alle caratteristiche fisiche del sistema e al linguaggio dell'unità centrale.

Il software di base può essere suddiviso, in funzione della sua allocazione in memoria interna, in quattro parti: software residente, il software non residente, il software opzionale, i programmi di utilità.

Il software di base è registrato su disco, ad eccezione di una piccola parte (il caricatore iniziale o bootstrap) che è registrato nella memoria ROM del sistema. Dal disco viene quindi trascritto in memoria interna, una parte all'atto dell'accensione, le altre ogni qualvolta si rendono necessarie per l'espletamento dell'operazione in corso. Le diverse parti di cui si compone il software vengono allocate in zone ben definite della memoria RAM del sistema. La memoria RAM viene utilizzata per contenere sia il software di base sia i programmi introdotti dall'utente. Si divide in due parti:

- memoria riservata al sistema
- memoria utente

Memoria riservata al sistema: E' la parte di memoria su cui vengono allocate esclusivamente parte di software di base. La sua dimensione è di 28K byte. Al suo interno una zona di circa 4K byte viene gestita parte come area di overlay e parte come stack. La dimensione dello stack varia nel tempo.

Memoria utente: E' la parte di memoria destinata a contenere i programmi utente. Può contenere inoltre parti del software di base. La sua dimensione minima

è di 16K byte è estendibile fino a 48K byte mediante l'introduzione di moduli di memoria, ciascuno della dimensione di 8K byte. (Vedi capitolo creazione di testi e programmi ed editing).

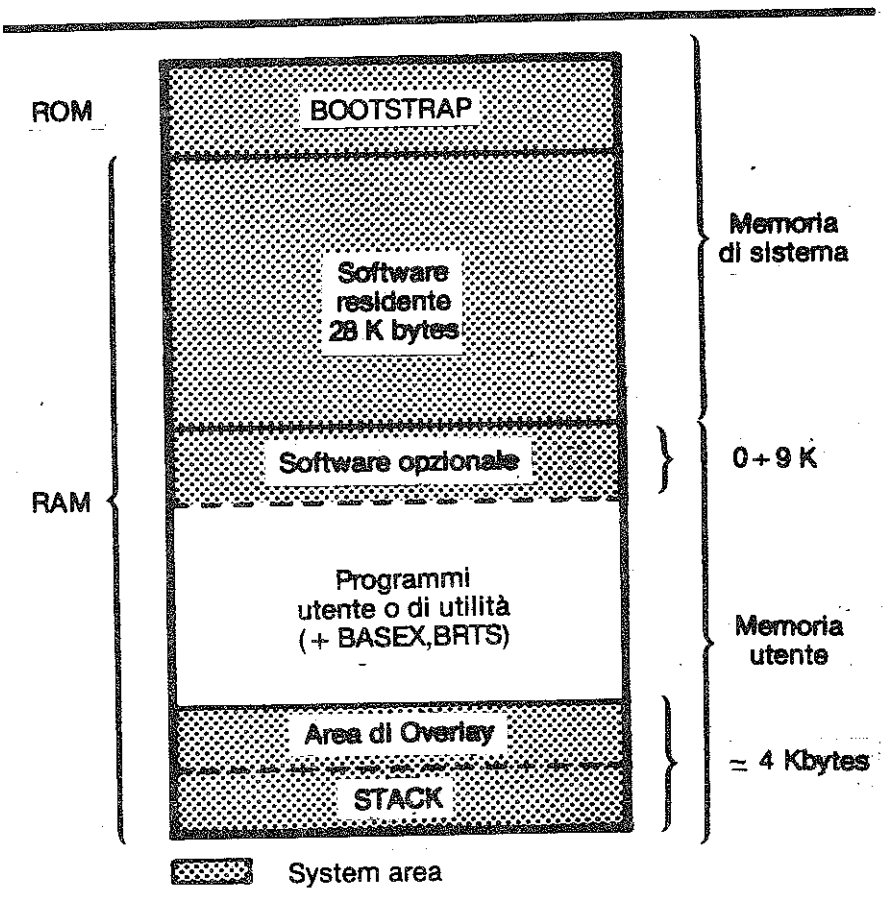


Figura 3-1 System Area

Si definisce software residente quella parte del software di base che viene caricata in memoria al momento dell'accensione della macchina e non ne viene più rimossa. Il software residente consta delle procedure che realizzano l'interfaccia virtuale fra l'hardware della macchina ed il resto del software (sia di base che applicativo), e di parte del sistema operativo (cioè delle procedure che gestiscono le risorse del sistema). Sono permanentemente residenti in memoria interna quelle parti del sistema operativo che risultano sempre indispensabili al funzionamento del sistema, con l'eccezione di alcune parti necessarie alla esecuzione dei programmi BASIC (BASEX e BRTS) (vedi cap. 5), che vengono caricate nella memoria utente in fase di preesecuzione del programma e vi restano per tutta la durata dell'esecuzione. Il software residente comprende i seguenti moduli:

- PIH (Program Interrupt Handler)

- PIOCS (Physical I/O Control System)

- MONITOR

- il modulo per la gestione del sistema nello stato "Calculator Mode"

Il software opzionale

Il software opzionale è costituito dalle estensioni del software base destinate a soddisfare particolari esigenze dei programmi dell'utente. In particolare consente di ottenere le seguenti estensioni (tra parentesi viene indicata l'occupazione approssimativa di memoria centrale):

- trattamento delle stringhe (2K byte)

- trattamento delle matrici (1K byte)

- uso della stampante integrata come plotter (2K byte)

- collegamento di periferiche su canale EIA RS 232 (2.5K byte)

- collegamento a video di pagina (1K byte)

Il software opzionale viene allocato nella memoria utente. Ogni opzione viene caricata su esplicita richiesta dell'utente; a seconda delle opzioni richieste la memoria utente assume quindi configurazioni diverse, e varia anche la parte di memoria disponibile per l'introduzione di un programma.

Le opzioni scelte dall'utente sono residenti finchè questi non ne richieda esplicitamente altre (comando OPTION o CONFIGURE). Quando ciò avviene il sistema carica le opzioni richieste nella memoria utente distruggendone il contenuto attuale.

Software non residente

Le restanti parti del sistema operativo (software non residente vengono caricate nella memoria riservata al sistema solo quando sono necessarie per espletare l'operazione richiesta in quel momento: in tal caso vengono caricate nell'area di overlay. L'area di overlay si trova all'interno della memoria riservata al si-

stema, ed ha una dimensione di circa 4K byte. Terminata la funzione richiesta l'area occupata in zona di overlay viene resa nuovamente disponibile e può ospitare nuove parti di sistema operativo.

Programmi di utilità

I programmi di utilità sono programmi che svolgono complesse funzioni su file, librerie, ecc. Consentono all'utente una gestione semplice delle informazioni memorizzate sui dischi.

I programmi di utilità, quando ne viene richiesta la esecuzione, sono caricati nella memoria utente. I programmi di utilità vanno a ricoprire l'area su cui è memorizzato il programma utente e non l'area eventualmente occupata dal software opzionale: si comportano perciò come un programma utente.

Le parti di software di base richiamate dal processor del programma di utilità vengono caricate in area di overlay. Al termine dell'esecuzione l'area occupata viene resa nuovamente disponibile per accogliere un nuovo programma di utilità oppure un programma utente.

Funzioni del sistema operativo

Le funzioni del sistema operativo sono:

- la gestione delle risorse del sistema (memoria, processor, operazioni di I/O, files)

- la gestione dei programmi utente (introduzione, compilazione, esecuzione, ecc.)

Per il sistema P6060 (monoprogrammato e multiprocessor) la gestione della memoria e del processor consiste nella distribuzione delle risorse tra il software di base ed un unico programma utente.

Realizzazione delle funzioni richieste al sistema operativo

Ogni operazione richiesta al sistema viene realizzata attivando un elemento del sistema operativo che funge da supervisore dell'operazione richiesta, coordinando l'esecuzione delle funzioni elementari di cui si compone l'operazione. Ogni funzione elementare può essere a sua volta scissa in sottofunzioni. Ad ognuna delle funzioni e delle sottofunzioni corrisponde un elemento del sistema operativo. Gli elementi che realizzano fisicamente tali funzioni vengono detti moduli: essi

sono le unità base del sistema operativo (d'ora in poi parleremo sempre di moduli per intendere gli elementi del sistema operativo).

Organizzazione in livelli del sistema operativo

Ad ogni elemento del sistema operativo può essere associato un livello più o meno alto a seconda della maggiore o minore complessità della funzione svolta: il sistema operativo risulta quindi suddiviso in una serie di livelli organizzati gerarchicamente.

Ogni livello è caratterizzato da un insieme di operazioni (le singole funzioni o sottofunzioni realizzate dal sistema operativo) e dall'insieme di dati su cui queste operazioni vengono svolte.

Si parla di organizzazione gerarchica dei livelli per significare che funzioni di un certo livello (e quindi i moduli che le realizzano) possono richiamare solo funzioni (moduli) dello stesso livello o di livello inferiore. Il livello più elevato è quello che interfaccia direttamente con l'utente.

I livelli gerarchici sono quattro: dall'esterno verso l'interno, distinguiamo un livello "utente" (livello 1), un livello "comandi" (livello 2), un livello "funzioni comuni" (livello 3), un "nucleo" (livello 4).

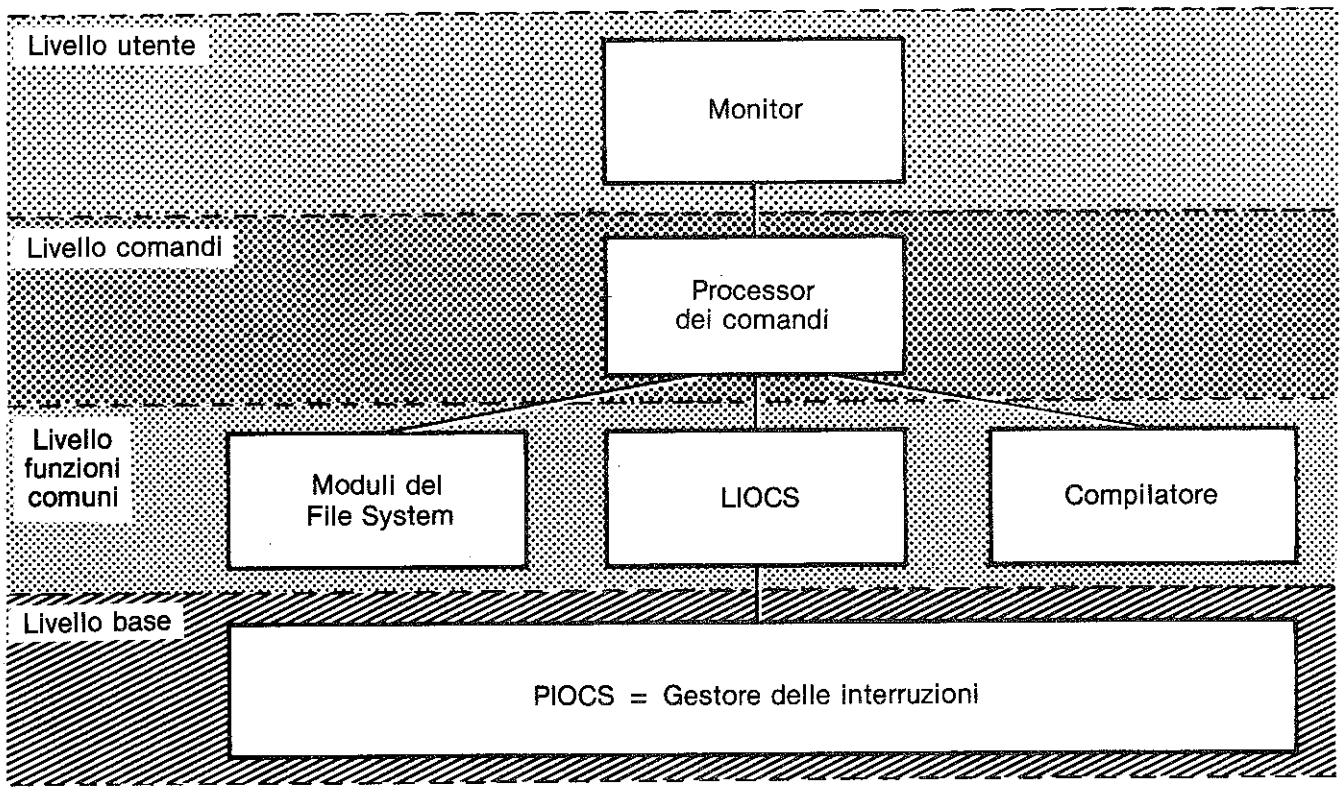


Figura 3-2 Organizzazione in livelli del Sistema Operativo

Livello utente: Il livello "utente" interfaccia direttamente con l'utente ed ha la funzione di riconoscere quanto viene digitato sulla tastiera. L'input è rappresentato dalla stringa di caratteri che viene volta per volta introdotta nel buffer di tastiera. Al livello utente appartiene soltanto un modulo residente, il MONITOR. Il MONITOR identifica (in base ai primi 3 caratteri) il comando introdotto (le istruzioni BASIC sono assimilate ai comandi) e cede il controllo, dopo averne lanciato il caricamento se non residente, al modulo preposto al suo trattamento; al termine delle operazioni il controllo ritorna al MONITOR.

Livello comandi: Ogni modulo di questo livello gestisce l'esecuzione di un singolo comando. L'esecuzione di un comando comporta in genere la suddivisione della funzione richiesta in più sottofunzioni. Il processor del comando richiamerà quindi in modo appropriato i moduli che realizzano le sottofunzioni richieste e ne supervedrà l'esecuzione. Terminata l'esecuzione del comando il controllo viene restituito al MONITOR.

Livello funzioni comuni: Al livello "funzioni comuni" appartengono alcune sottofunzioni comuni a più comandi (è il caso, ad esempio, dell'operazione di ricerca nel catalogo dei files, che può essere richiesta dai comandi OLD, PURGE, CATALOG, ecc.). Appartengono a questo livello il compilatore e il decompilatore BASIC, i moduli del File System, il LIOCS (Logical I/O Control System).

I moduli che appartengono al livello "funzioni comuni" trattano essenzialmente i programmi e i dati contenuti nell'area utente o nei files su disco.

Nucleo: A questo livello appartengono i moduli che costituiscono l'estensione immediata dell'hardware del sistema, realizzando funzioni elementari comuni a tutte o quasi le attività svolte dal sistema.

Fanno parte di questo livello i moduli del PIOCS (Physical I/O Control System), i moduli per la gestione delle parti del sistema operativo non residenti e per la gestione delle interruzioni.

Dati e livello loro associato

Anche i dati su cui i moduli operano possono essere associati a dei livelli. Mentre però i moduli sono associati ad un solo livello i dati possono essere di due tipi:

- dati associati ad un livello; sono di norma associati ad un singolo modulo

- dati associati a due livelli; costituiscono l'interfaccia tra moduli che non appartengono allo stesso livello: così per esempio, le informazioni di controllo sul working file fanno parte dell'interfaccia tra moduli del livello comandi e moduli del livello funzioni comuni

La struttura del software di base

Ciascuno dei programmi di cui è costituito il software di base realizza una particolare funzione del sistema e costituisce un modulo. Uno o più moduli sono organizzati in una struttura superiore: il segmento.

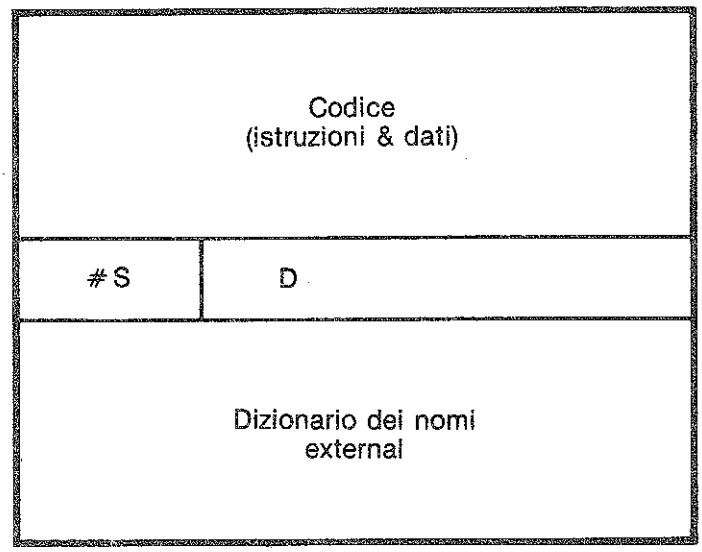
I moduli

I moduli sono le unità elementari di cui è costituito il software. Un modulo è un'unità logica che esegue

una o più funzioni correlate. Funzioni distinte vengono realizzate da programmi allocati in moduli separati, secondo uno schema di programmazione modulare.

Struttura dei moduli

I moduli sono costituiti da una control section (cioè una unità di codice comprendente istruzioni e dati) e da un dizionario contenente la raccolta dei nomi external riferiti (riferimenti ad altri moduli). In ogni control section vi è un solo nome external: il nome della control section stessa.



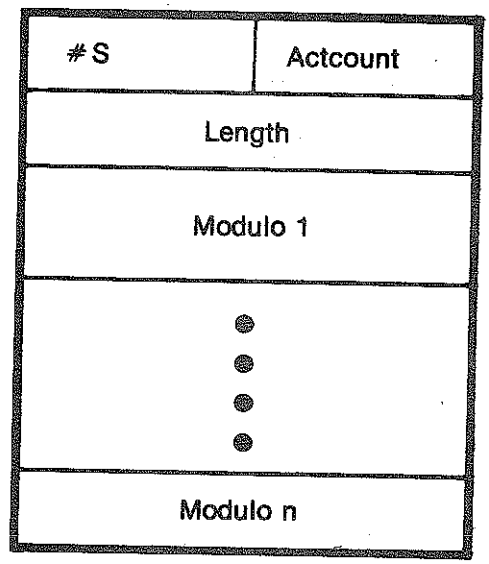
Ad ogni nome external corrisponde un elemento dei dizionario che contiene il suo indirizzo, nella forma indicata in figura. Il valore # S indica il segmento all'interno del quale si trova il modulo; D indica lo spiazzamento del modulo rispetto all'origine del segmento.

I moduli sono rilocabili (non contengono quindi costanti indirizzo) e possono perciò essere caricati in un punto qualsiasi della memoria. I dati che sono utilizzati in sola lettura (tabelle, costanti, ecc.) sono collocati insieme al codice relativo alle istruzioni. Le aree di lavoro il cui contenuto è significativo soltanto per l'esecuzione della procedura che le contiene (e quindi non lo è per le altre procedure) vengono allocate sullo stack all'inizio dell'esecuzione, rimosse quando questa ha termine. I dati condivisi da più moduli vengono inseriti in un'area di

comunicazione del sistema (COMAREA), allocata in memoria nella fase di inizializzazione del sistema.

I segmenti

Il segmento è la più piccola unità del software di base che, dal supporto dove è registrata, viene caricata in memoria. Ciò vuol dire che quando un modulo si trova nella memoria interna, vi si trovano anche tutti gli eventuali altri moduli che insieme fanno parte del segmento al quale il modulo appartiene.



Il significato dei vari campi è il seguente:

- #S — nome del segmento
- ACTCOUNT — contatore dei moduli del segmento che sono attivi
- LENGTH — lunghezza del segmento
- Modulo 1, ..., n — moduli che compongono il segmento

Il campo ACTCOUNT ha senso relativamente ai segmenti che vengono caricati in area di overlay. Si dice che un modulo è attivo quando vengono eseguite istruzioni appartenenti ad esso, oppure è attivo un modulo da esso richiamato e dal quale deve riprendere il controllo. Si dice che un segmento è attivo quando è attivo almeno uno dei moduli che ad esso appartengono. Finchè un segmento è attivo non può essere ricoperto da altri segmenti caricati in area di overlay; se un

segmento non è più attivo l'area da esso occupata può essere invece utilizzata per il caricamento di altri segmenti.

Un segmento è costituito di norma da moduli che si richiamano l'un l'altro, concorrendo ad eseguire una unica funzione. Poichè quando un modulo ne richiama un altro appartenente allo stesso segmento questo è già disponibile in memoria, viene così risparmiato il tempo del caricamento da disco.

La tabella dei segmenti (SEGTAB)

I segmenti sono registrati su disco sistema in modo contiguo, ed ognuno di essi è allineato al settore.

La tabella dei segmenti (SEGTAB), che ne raccoglie gli indirizzi, è anch'essa registrata sul disco sistema. La SEGTAB viene caricata in memoria in fase di inizializzazione; qui gli indirizzi di disco dei segmenti del software residente ed opzionale vengono sostituiti dagli indirizzi di memoria interna dei segmenti stessi. Per gli altri segmenti l'indirizzo riportato è sempre quello del disco.

Il passaggio di controllo tra un modulo e l'altro viene realizzato per mezzo di un meccanismo che utilizza come dati fondamentali l'indirizzo del modulo chiamato (informazione presente nel Dizionario dei Nomi External del modulo chiamante) e la tabella dei segmenti (SEGTAB). Questo meccanismo viene utilizzato per tutti i moduli di software, ad eccezione del BASEX e dei moduli di BRTS.

Se il modulo chiamato appartiene ad un segmento di software residente, la SEGTAB contiene l'indirizzo di memoria del segmento. L'aggancio con il modulo viene quindi realizzato passando direttamente all'esecuzione dell'istruzione che si trova all'indirizzo I + D, dove I è l'indirizzo indicato nella SEGTAB, e D è lo spiazzamento indicato nell'elemento del Dizionario dei Nomi External corrispondente al modulo riferito. I moduli che appartengono ai segmenti di software non residente vengono caricati in area di overlay. L'area di overlay viene gestita in modo tale da:

- non ricaricare in memoria un segmento già caricato e che vi è ancora registrato

- mantenere contigua l'area occupata dai segmenti attivi

- mantenere i segmenti attivi in testa all'area di overlay

Due puntatori indicano l'inizio dell'area di overlay e l'indirizzo di impianto dell'ultimo segmento che vi è registrato. Nella figura è schematizzata una possibile configurazione dell'area di overlay.

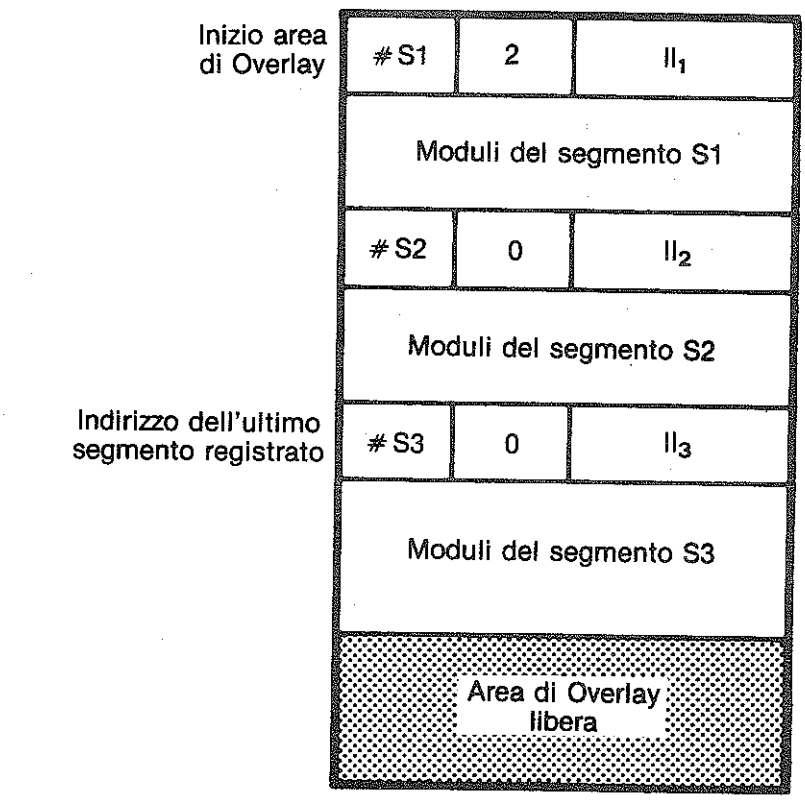


Figura 3-3 L'area di overlay

I segmenti presenti sono 3 (#S1, #S2, #S3) di cui il primo è attivo (campo ACTCOUNT $\neq \emptyset$) e gli altri no.

Per l'aggancio di moduli di software non residente, oltre alle informazioni presenti nel Dizionario dei Nomi External e nella SEGTAB, viene esaminata anche l'area di overlay per verificare se il segmento che contiene il modulo non sia già in memoria.

Nel caso della figura 3-3, il richiamo (da parte del segmento #S1) di moduli appartenenti ai segmenti S1

e #S2 provoca il passaggio del controllo e l'incremento del campo ACTCOUNT (il segmento #S2 è a questo punto un segmento attivo). Il richiamo di moduli appartenenti al segmento #S3 provoca oltre al passaggio del controllo e all'incremento del campo ACTCOUNT, uno spostamento del segmento in coda al segmento #S1: il segmento #S2 a questo punto non è più presente nell'area di overlay.

Il richiamo di moduli appartenenti ad un segmento non registrato in memoria provoca il caricamento di questo ultimo in coda al segmento #S1 e quindi le stesse operazioni descritte per i casi precedenti. I segmenti #S2 e #S3 a questo punto non sono più registrati in memoria.

I processor dei programmi di utilità vengono caricati nell'area utente.

Gestione dello stack e dell'area di overlay

L'area di overlay e lo stack sono adiacenti. Ad essi viene riservata una zona di memoria la cui estensione varia al variare delle funzioni del sistema. L'area di overlay viene occupata dall'alto verso il basso, lo stack in senso opposto. Si ha condizione di errore solo quando una delle due aree cresca fino a invadere la parte effettivamente impegnata dall'altra. Quando il programma utente è a livello di editing, l'area di overlay viene occupata dai segmenti del sistema operativo e lo stack viene occupato dai dati locali del moduli. In questa fase l'area di overlay + stack ha una dimensione di 4K byte.

Quando il programma utente è in fase di esecuzione (processor ALP attivo) l'area di overlay viene occupata da moduli del LIOCS; lo stack viene occupato da indirizzi, valori, contesti locali, ecc. In questo caso l'area di overlay + stack ha una dimensione pari a tutta l'area utente rimasta libera dopo la fase di preesecuzione del programma (per la preesecuzione e l'esecuzione di un programma BASIC, vedi cap. 5).

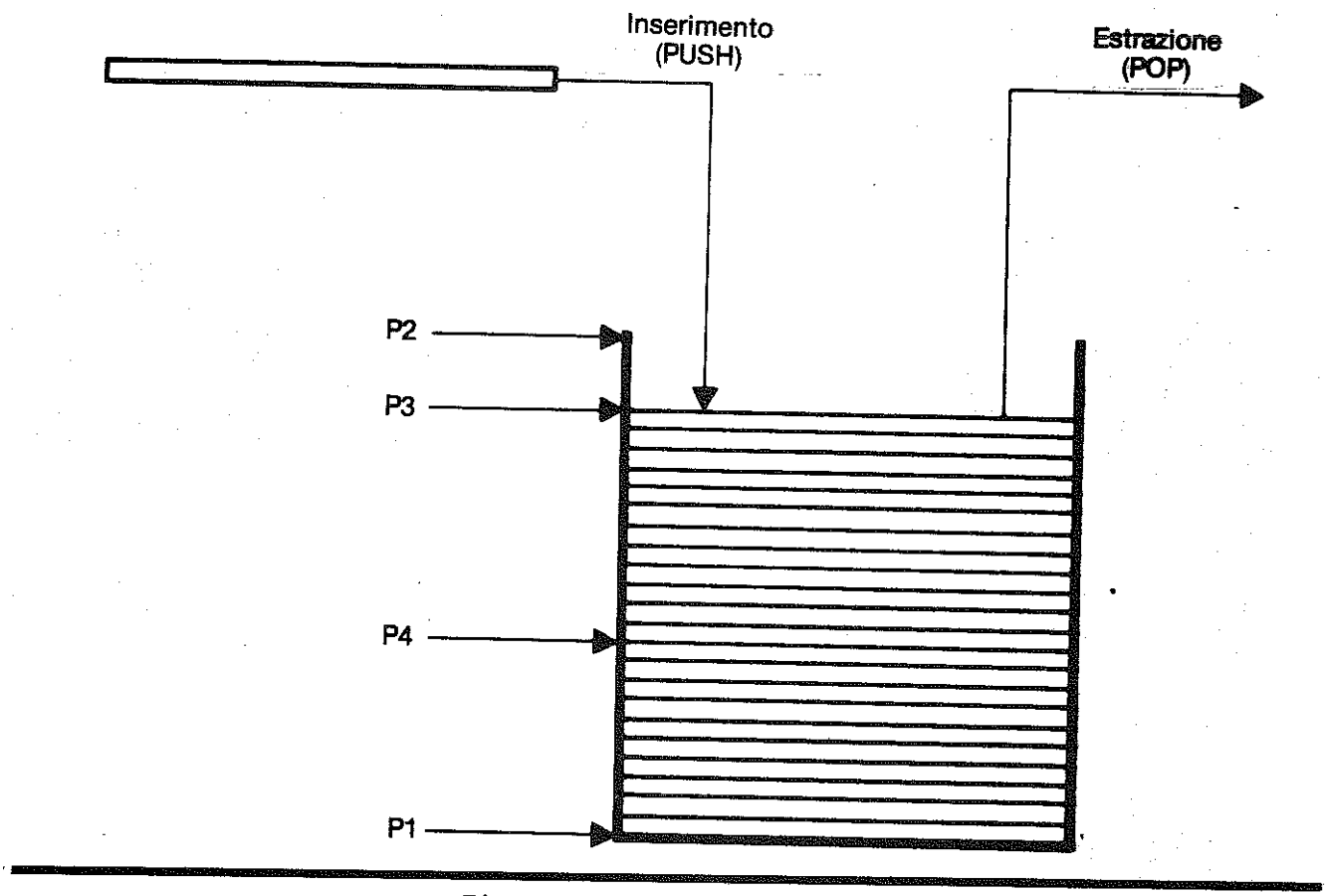


Figura 3-4 Principio di funzionamento dello STACK

I puntatori sono:

P_1 = Bottom: punta alla base dello stack

P_2 = Top: punta al limite superiore dello stack

P_3 = Tos (TOP OF STACK): punta al primo byte libero dello stack

P_4 = Lenv (Local Environment): punta al byte più basso nel contesto locale allocato

Non esiste un disegno di strutturazione gerarchica degli overlay: la radice di overlay è il software residente, mentre i rami di overlay (i segmenti) si possono agganciare tra loro richiamandosi a vicenda in memoria (senza cioè dover passare per il richiamo dalla radice di overlay, ma riferendosi soltanto alla SEGTAB).

Le chiamate rivolte da un modulo residente a un modulo esterno sono fatte per indiretezza attraverso la SEGTAB, ove sono conservate le informazioni che con-

sentono di reperire il modulo esterno richiamato sia che si trovi in memoria interna sia che si trovi in memoria esterna. Nella SEGTAB è conservato anche l'indirizzo di alcune funzioni built-in.

Mentre in una struttura ad overlay di tipo statico è possibile valutare, già in fase di lancio del linkage editor, l'occupazione massima di memoria, e prevenire eventuali overflow, in una struttura siffatta, con aggancio dinamico l'occupazione di memoria è difficilmente valutabile a priori, poichè dipende da una serie di quantità che variano nel corso dell'elaborazione (come il numero di funzioni, built-in oppure utente, che vengono richiamate da una stessa istruzione). Inoltre, poichè l'area di overlay compete con lo stack, l'eventuale overflow di memoria può essere determinato, in ogni momento, dall'occupazione dinamica dello stack (che dipende anch'essa di quantità che variano nel corso dell'elaborazione, come il numero di cicli contemporaneamente nidificati). Va rilevato che l'area di overlay compete con lo stack non soltanto in fase di esecuzione, ma anche in fase di compilazione e decompilazione.

Le macchine virtuali

Sull'unità centrale del sistema P6060 sono state definite due macchine virtuali con i rispettivi linguaggi: la macchina SLP (System Language Processor) e la macchina ALP (Algebric Language Processor). Le macchine virtuali eseguono i programmi espressi nel loro linguaggio.

La macchina SLP viene programmata utilizzando la codifica simbolica del linguaggio macchina, i tempi di risposta dell'unità centrale sono in questo caso molto brevi. La macchina ALP viene programmata in linguaggio BASIC, ed è quindi destinata all'esecuzione dei programmi utente. E' dotata di aritmetica floating-point e di istruzioni di controllo complesse.

Un sottoinsieme del software di base scritto nel linguaggio della unità centrale, realizza l'emulazione delle macchine virtuali ALP e SLP; esse non hanno alcun legame di dipendenza con il resto del software di base. Poichè anche la parte che realizza l'emulazione delle macchine virtuali viene caricata in memoria da disco, è possibile incrementare le funzioni del sistema semplicemente modificando il software, senza dover

modificare l'hardware della macchina: ogni utente può quindi avere a sua disposizione la versione più aggiornata del sistema semplicemente introducendo il dischetto o il disco che la contiene. La funzione delle macchine virtuali è chiarita nel disegno seguente.

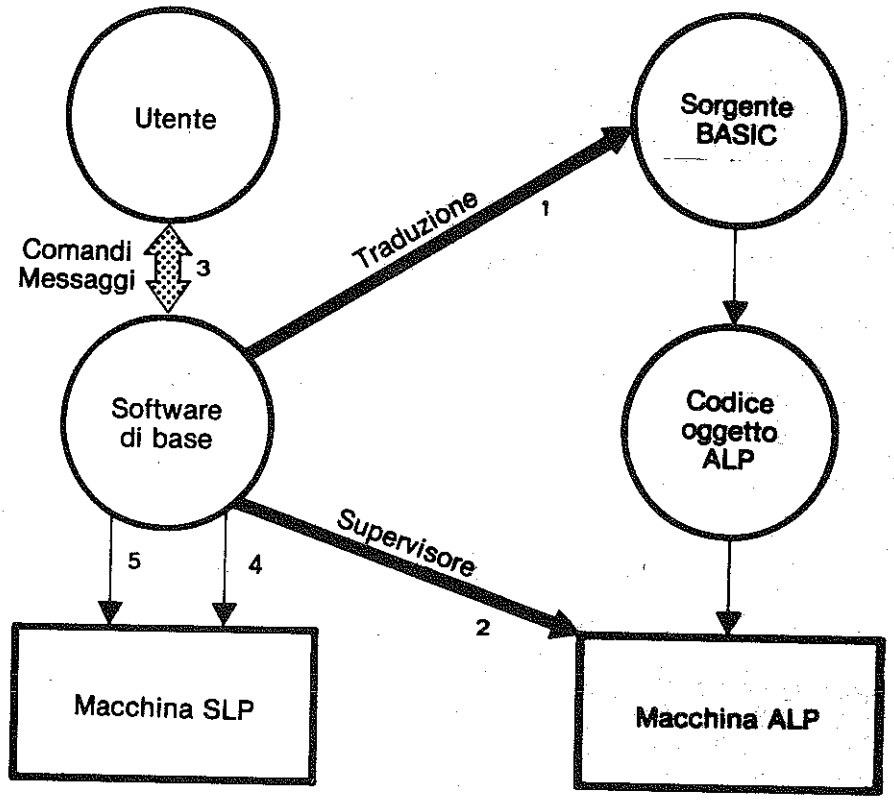


Figura 3-5 Le macchine virtuali

Lo schema di principio è il seguente:

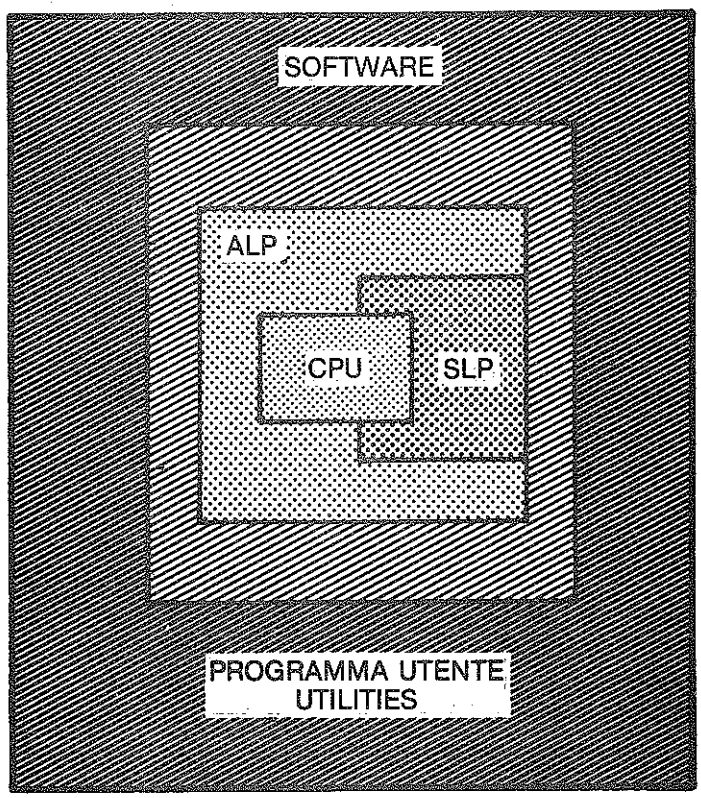
1. Il software di base, che gira sulla macchina SLP, traduce il sorgente BASIC in codice oggetto ALP.

2. Il programma BASIC tradotto in codice oggetto, gira sulla macchina ALP supervisionato dal software di base.

3. L'utilizzatore colloquia con la macchina tramite il software di base.

4. Gestione dell'I/O e delle interruzioni.

5. Calculator Mode.



Il linguaggio macchina SLP

Il linguaggio di sistema, accettato dalla macchina SLP, è dotato di:

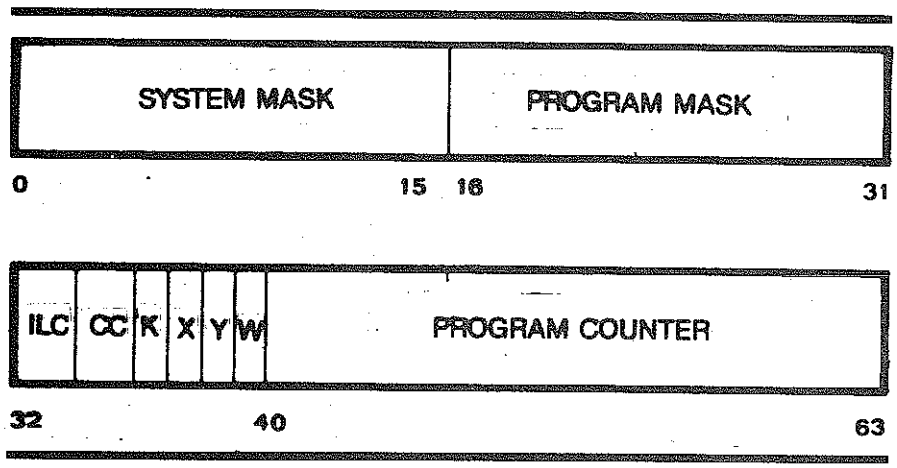
- istruzioni per le operazioni di aritmetica binaria
- istruzioni per le operazioni logiche
- istruzioni per la gestione dello stack (per l'allocazione e il rilascio di aree, e l'acquisizione di valori passati nella COMAREA)
- istruzioni per l'attivazione e l'aggancio di moduli di sistema (per la gestione dinamica dei moduli caricati in area di overlay)
- istruzioni per operazioni di conversione (di dati nel formato desiderato)
- istruzioni per operazioni di ricerca tabellare: (per ricerche tabellari dicotomiche sequenziali, sequenziali con maschera)
- istruzioni per la gestione dell'I/O: (per i parame-

tri di controllo dei canali di I/O sono conservati nelle parole di controllo di canale CCW)

- istruzione per il passaggio alla macchina ALP: ad essa corrisponde un'istruzione simmetrica nella macchina ALP. Le istruzioni macchina SLP vengono interpretate dal software di base. Il valore del program counter, contenuto nella parola di stato (PSW), permette di posizionarsi sull'istruzione da eseguire; l'istruzione viene poi riconosciuta e servita da una specifica routine.

La Program Status Word (PSW)

La Program Status Word è un'area di memoria lunga 64 bit che contiene le informazioni necessarie all'esecuzione delle istruzioni del programma. La PSW attiva è detta "PSW corrente". Il formato della PSW è il seguente:



I campi hanno il seguente significato:

System Mask: E' la maschera delle interruzioni da unità periferiche (interruzioni esterne). Ogni bit è posto in corrispondenza biunivoca con una periferica integrata oppure con un canale: le interruzioni sono abilitate se il bit corrispondente è uguale ad 1.

Program Mask: E' la maschera delle interruzioni di programma. Ogni bit è posto in corrispondenza biunivoca con un tipo di interruzione (interruzioni dovute ad errore o a chiamate del supervisore). Le interruzioni sono abilitate se il bit corrispondente è uguale ad 1.

Instruction Length Code (ILO): Contiene la lunghezza dell'istruzione da eseguire.

Condition Code (CC): Contiene il codice di condizione.

K,X,Y,P: X e Y sono disponibili per usi futuri. K indica se la memoria è impaginata oppure no. P indica se è attiva la macchina virtuale SLP o quella ALP.

Program Counter: Contiene l'indirizzo dell'istruzione successiva a quella in corso di esecuzione. La PSW, nel caso si verifichi un'interruzione di tipo abilitato, viene memorizzata e può essere successivamente analizzata. Al suo posto viene caricata una nuova PSW che lancia l'esecuzione delle routine di gestione delle interruzioni.

Il linguaggio macchina ALP

Il linguaggio macchina ALP, o linguaggio algebrico, è orientato all'elaborazione di espressioni algebriche, e permette di ottenere una traduzione compatta del programma BASIC in linguaggio oggetto. Il programma oggetto è inoltre reversibile: può essere convertito nuovamente in formato sorgente passando attraverso una fase di decompilazione.

In tal modo si ottengono tre vantaggi:

- minore occupazione di memoria esterna (dovendosi conservare il solo formato oggetto)
- minore occupazione di memoria centrale (essendo la traduzione assai compatta)
- possibilità di effettuare le operazioni di editing direttamente sul programma pronto per l'esecuzione, o addirittura in esecuzione

Quest'ultima caratteristica fa sì che il P6060 risulti uno dei più comodi e fruibili strumenti per lo sviluppo del software.

Il set di istruzioni consente di eseguire calcoli con numeri decimali in semplice e doppia precisione, di elaborare stringhe, di operare su matrici, di realizzare strutture di controllo, di comunicare con la macchina SLP.

Il linguaggio algebrico è privo di istruzioni I/O. Per tali funzioni il sistema fa ricorso alla macchina SLP. Le istruzioni accette dalla macchina ALP sono di 4

tipi e si defferenziano a seconda della lunghezza: 1, 2, 3 oppure N bytes ($3 \leq N \leq 256$). Le istruzioni di lunghezza 1 e 2 sono composte dal solo codice operativo. Le istruzioni di lunghezza 3 hanno il codice operativo sul primo byte e un operando esplicito sugli altri due. Nelle istruzioni a lunghezza variabile il primo byte contiene il codice operativo, il secondo il descrittore; l'operando vero e proprio occupa i byte restanti.

La macchina virtuale ALP

La macchina virtuale ALP è destinata all'elaborazione di espressioni aritmetiche espresse in notazione polacca inversa. Le operazioni vengono eseguite sullo stack.

La macchina ALP è costituita da tre componenti principali: la memoria, lo stack e il processor.

La memoria

La memoria contiene i dati e il programma scritto in linguaggio algebrico. Sotto il controllo del processor i dati possono essere poi scambiati tra memoria e stack. La memoria della macchina virtuale ALP è così composta:

Area di Definizione del Working File (WFDA): E' posta in un'area di comunicazione dati del sistema (COMAREA) e contiene i puntatori alle tabelle e alle aree usate dai programmi BASIC.

Area del Programma: Contiene il programma utente.

Area dei Descrittori: Contiene le tabelle dei descrittori delle variabili semplici e multiple e le tabelle usate per la compilazione e l'esecuzione dei programmi utente.

Area delle Variabili: Contiene i valori delle variabili semplici e multiple, numeriche e/o stringa.

Lo stack

Lo stack è una zona di memoria in modo implicito da parte delle istruzioni ALP. E' gestito dal processor per mezzo di una serie di puntatori che permettono di controllare tutte le fasi di allocazione e deallocazione di aree (vedi "Gestione dello stack e dell'area di overlay). Sullo stack vengono allocati:

- indirizzi e valori per la valutazione di espressioni

- informazioni di controllo per il richiamo di funzioni (contesti di function)

- informazioni di controllo ed aree delle variabili locali e degli argomenti di chiamata necessari per la loro valutazione (contesti locali)

- strutture di controllo per chiamate a subroutine ed esecuzione di cicli FOR/NEXT (contesti di subroutine e contesti di loop)

Il processor

Il processor della macchina ALP è costituito dalla parte di sistema che realizza le istruzioni di linguaggio algebrico e permette di gestire lo stack. Il processor può accedere ad aree di lavoro, poste nella COMAREA, dove sono contenute informazioni come:

- la PSW, le cui informazioni sono in comune tra la macchina ALP e la macchina SLP

- il contatore per l'annidamento dei cicli FOR/NEXT

- il contatore per l'annidamento delle funzioni

- il tipo di operazione eseguita nell'ultima istruzione

Il valore del program counter, contenuto nella parola di stato (PSW) consente il posizionamento sull'istruzione da eseguire. Seguono una fase di posizionamento sul valore degli operandi ed una fase di esecuzione dell'operazione (vedi figura pag. 3-17). L'esecuzione delle operazioni provoca, generalmente, una modifica dello stack.

Le istruzioni che, in accordo con la struttura polacca del codice, preparano gli operandi per le successive operazioni algebriche, caricano lo stack con:

- l'indirizzo, nel caso che l'operando sia costituito da una variabile

- il valore, nel caso che l'operando sia costituito da una costante.

Le istruzioni che realizzano l'operazione algebrica vera e propria prelevano dallo stack i valori degli operandi (o il loro indirizzo) e depositano, al loro posto, il risultato dell'operazione. Il riferimento ad una variabile di programma viene risolto in due fasi: dapprima viene caricato sullo stack l'indirizzo della variabile; poi avviene il posizionamento sul valore della variabile e la sua utilizzazione.

Si è già detto che la macchina ALP dipende dalla macchina SLP per le operazioni di I/O. E' inoltre la macchina SLP che provvede a compilare con le informazioni appropriate l'Area dei Descrittori e l'Area di Definizione del Working File (WFDA), utilizzate poi dalla macchina ALP.

Le macchine SLP ed ALP possono scambiarsi informazioni attraverso le risorse in comune: lo stack e la memoria.

Infine le due macchine virtuali possono cedersi il controllo a vicenda. Il controllo può essere ceduto in entrambe le direzioni in modo diretto (con apposite istruzioni), oppure in modo indiretto, dal processor ALP al processor SLP, con un meccanismo di interruzione comune ad entrambe le macchine.

Lo "stato del sistema"

Il sistema P6060 accetta come input stringhe di caratteri introdotte per mezzo della tastiera. Ogni stringa che viene processata, dando luogo ad una azione, lascia una traccia all'interno del sistema.

Definiamo "stato del sistema" l'insieme delle informazioni che rappresentano la storia delle informazioni che rappresentano la storia delle operazioni precedenti e condizionano l'azione del sistema a fronte della introduzione di una nuova stringa di caratteri.

All'utente il sistema P6060 si presenta come una macchina a stati. Le informazioni connesse con lo stato del sistema sono registrate in due campi di memoria all'interno della COMAREA: la Parola di Stato del Sistema (SSW) e l'Area di Definizione del Working File (WFDA).

Livelli del sistema

Gli stati del sistema possono essere raggruppati in due livelli:

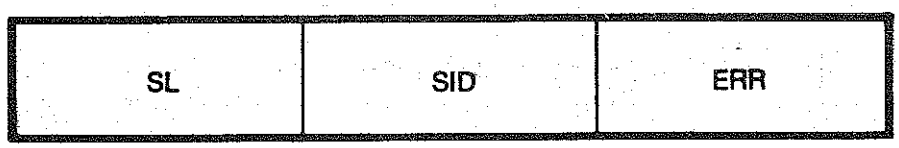
- livello comandi: esecuzione dei comandi di creazione e di editing di programma o di testo, esecuzione dei comandi di gestione delle librerie, uso della macchina come calcolatrice

- livello programma: esecuzione del programma e dei comandi che consentono di controllare l'esecuzione e il debugging del programma stesso

Un solo stato, OPERATOR CALL, è comune ai due livelli. I due livelli si differenziano per il formato del programma utente presente in memoria centrale. A livello comandi il programma utente è in formato output del compilatore, o formato di editing; a livello programma è in formato output del preesecutore, o formato eseguibile.

La Parola di Stato del Sistema (SSW)

La Parola di Stato del Sistema indica istante per istante lo stato presente del sistema. Il suo formato è il seguente:



il significato dei singoli campi è il seguente:

- SL : Status Level; indica il livello del sistema (comandi/programma)

- SID : Status Identifier; contiene il codice dello stato riferito al livello del sistema

A livello comandi si distinguono gli stati:

- CM : Calculator Mode

- PCE : programma utente in fase di creazione o di editing (Program Creation Editing)

- TCE : testo in fase di creazione o di editing (Text Creation Editing)

- ICEX: esecuzione dei comandi interrompibili Catalog o List (Interruptable Command Execution)

- SE : errore di sintassi (Syntax Error)

- DS : deterioramento del sistema (Dead System)

- OPC_1 : richiesta di intervento dell'operatore (Operator Call)

A livello programma si distinguono gli stati:

- DEBUG: programma utente in debugging (DEBUGing)

- PEX : programma utente in fase di esecuzione (Program Execution)

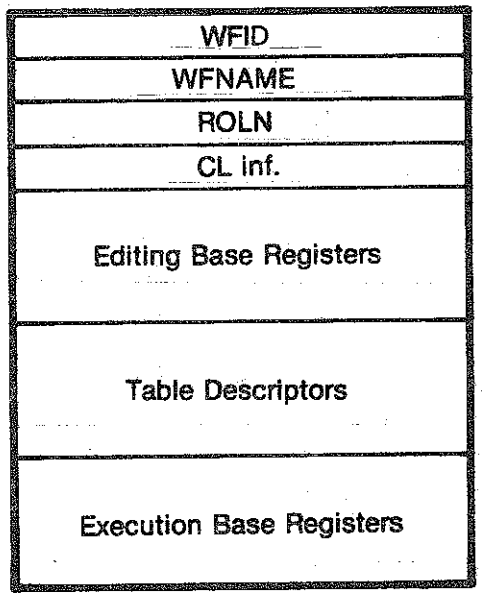
- PIW : programma utente in attesa di input (Program Input Waiting)

- OPC_2 : richiesta di intervento dell'operatore (Operator Call)

- ERR : contiene il codice dell'errore eventualmente riscontrato

Area di Definizione del Working File (WFDA)

Il Working File è la parte della memoria interna riservata per contenere il programma o il testo introdotto dall'utente. Il campo di memoria WFDA (Working File Definition Area) contiene tutte le informazioni necessarie ad operare sul programma o sul testo introdotto. Il suo formatc è il seguente:



Il significato dei singoli campi è il seguente:

WFID (Working File Identifier): Identifica il contenuto del working file (vuoto, programma, testo) ed eventualmente lo stato del programma utente contenuto in memoria (editing, debugging, esecuzione).

WFNAME (Working File NAME): Contiene il nome del programma o del testo presente nel working file.

ROLN (Read Only Line Number): Numero di linea da cui comincia la protezione per i programmi soggetti a protezione.

CL inf. (Current Line information): Contiene il numero di linea e l'indirizzo della linea corrente.

A seconda del tipo di operazione in corso la linea corrente può essere definita come:

- l'ultima linea introdotta senza errore
- l'ultima linea visualizzata
- l'ultima linea stampata
- l'ultima linea di cui è stata intrapresa l'esecuzione

Editing Base Registers: L'insieme degli indirizzi delle tabelle che sono utilizzate in fase di introduzione ed editing di un programma o di un testo.

Table descriptors: L'insieme dei descrittori di tali tabelle.

Execution Base Register: L'insieme degli indirizzi del codice di un programma e delle tabelle che sono necessarie in esecuzione del programma stesso.

ICEX (Instruction Counter in esecuzione): Puntatore all'ultima istruzione eseguita o a quella in corso di esecuzione.

La conoscenza della struttura di quella parte della memoria che viene occupata dal programma utente sarà utile all'utente stesso per valutare di volta in volta lo stato complessivo della macchina virtuale algebrica, cioè delle variabili del sistema.

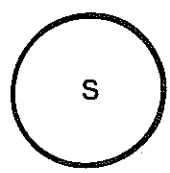
Diagrammi degli stati

Il diagramma degli stati contiene le informazioni necessarie per conoscere, in ogni momento, lo stato del sistema. Più precisamente, nel diagramma degli stati sono riportati:

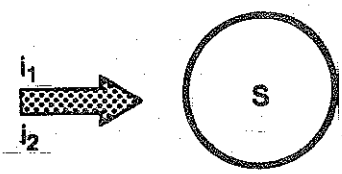
- l'illustrazione dei possibili stati
- l'elenco degli ingressi che il sistema può accettare in ogni stato
- l'evidenziazione dei cambiamenti di stato che il sistema subisce a seguito dell'introduzione dei comandi.

Il diagramma costituisce, in pratica, una descrizione sintetica e formalizzata delle caratteristiche funzionali principali del sistema.

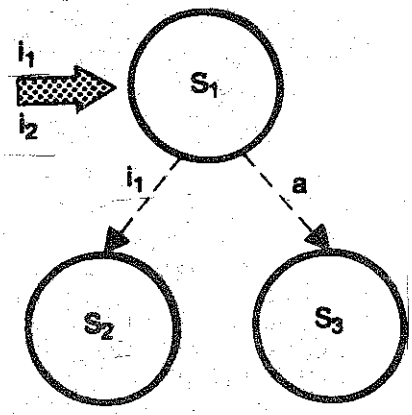
Simbologia adottata



indica lo stato del sistema denominato 'S'.



indica che il sistema, nello stato 'S', può accettare gli ingressi 'i_1' ed 'i_2'.



indica che il sistema che nello stato 'S_1' può accettare gli ingressi 'i_1' e 'i_2', passa allo stato 'S_2' per effetto dell'ingresso 'i_1' (freccia di collegamento continua tra i due stati), passa invece allo stato 'S_3' per effetto di un'azione interna 'a' di sistema, come per esempio OPERATOR CALL, o di programma, come per esempio una richiesta di dati da tastiera (freccia di collegamento tratteggiata). Infine l'ingresso 'i_2', anch'esso accettato nello stato 'S_1' non provoca cambiamenti di stato: esso non compare associato ad alcuna freccia di collegamento con altri stati.

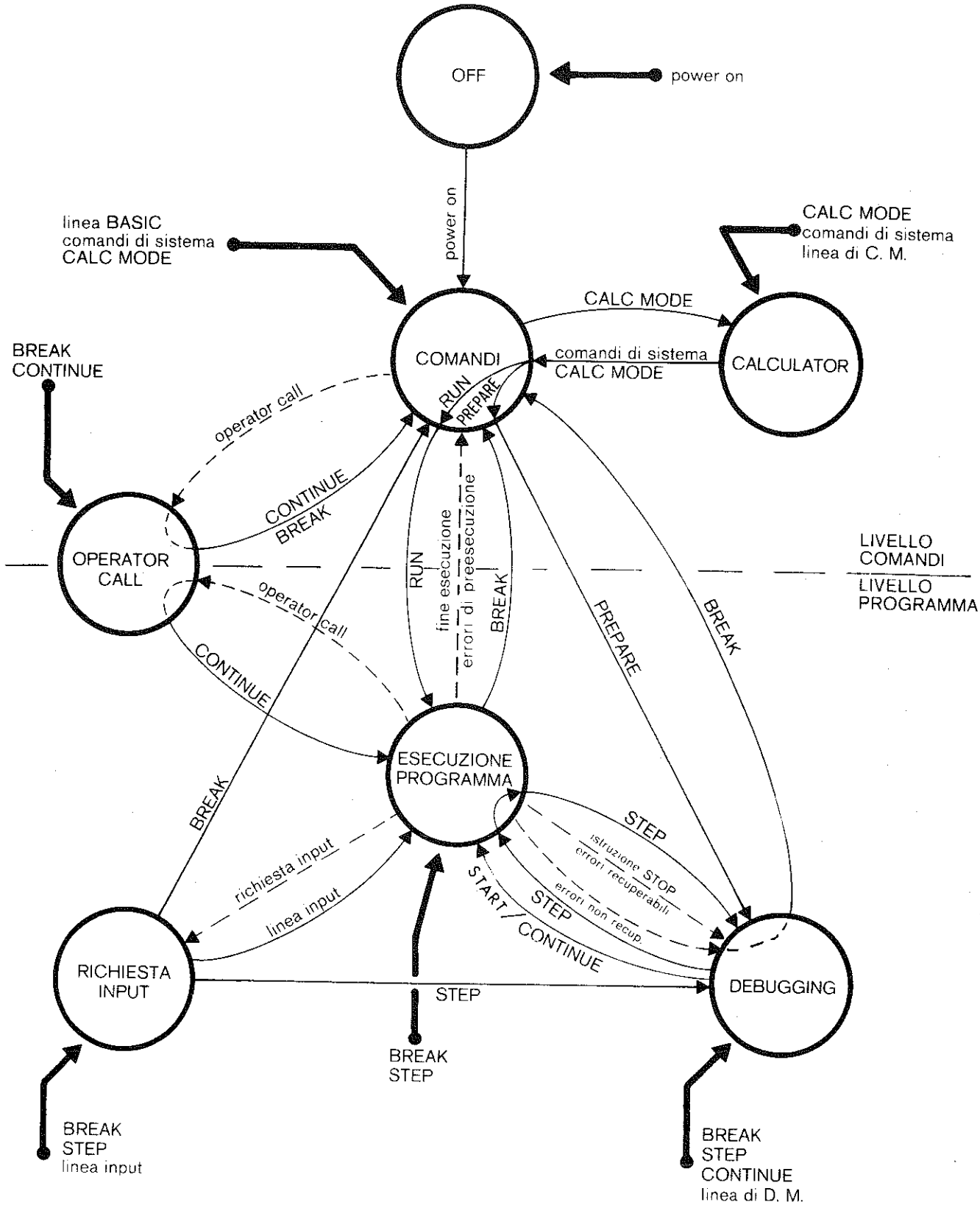


Figura 3-6 Diagramma completo degli stati

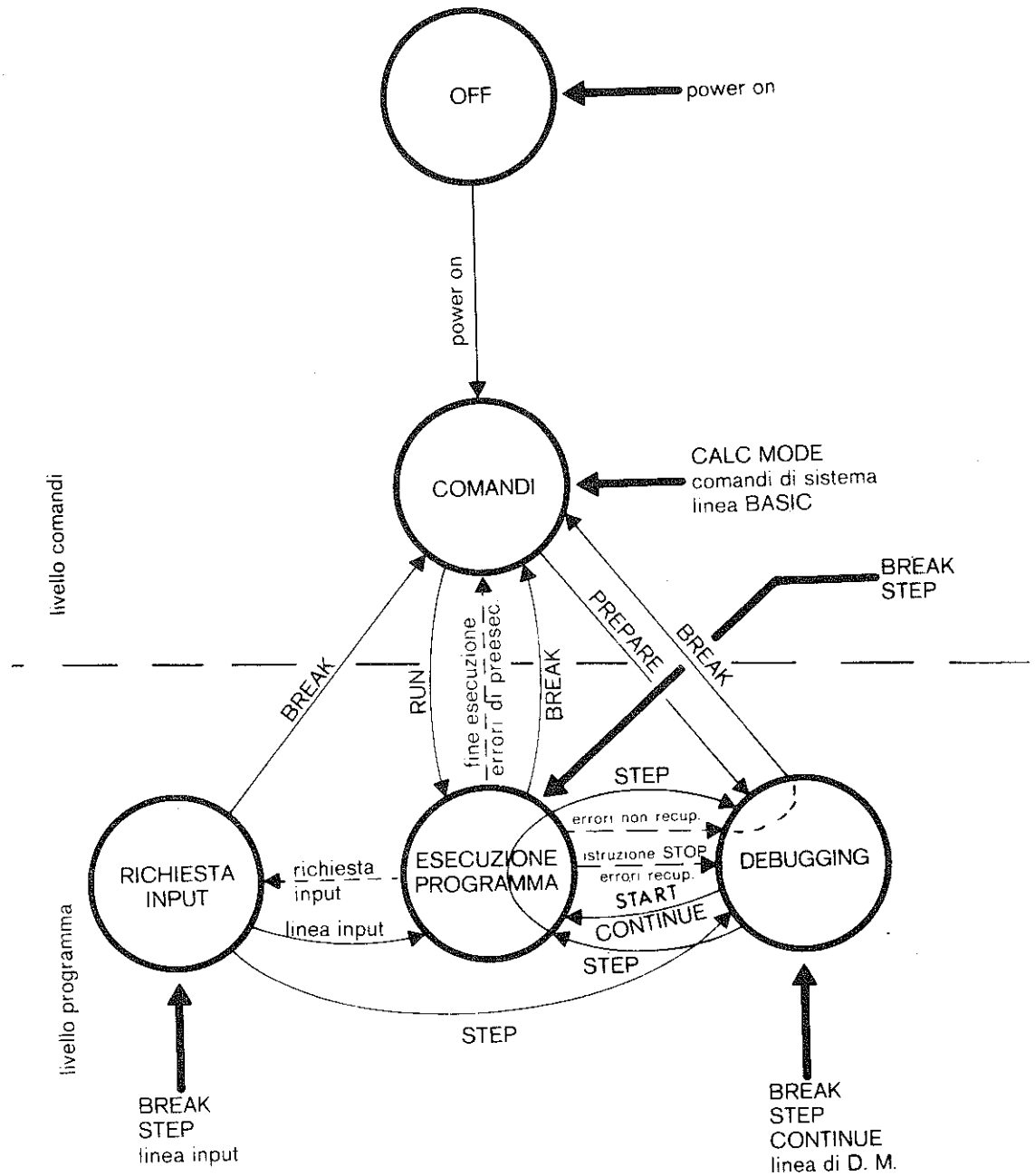


Figura 3-7 Stato 'ESECUZIONE PROGRAMMA'

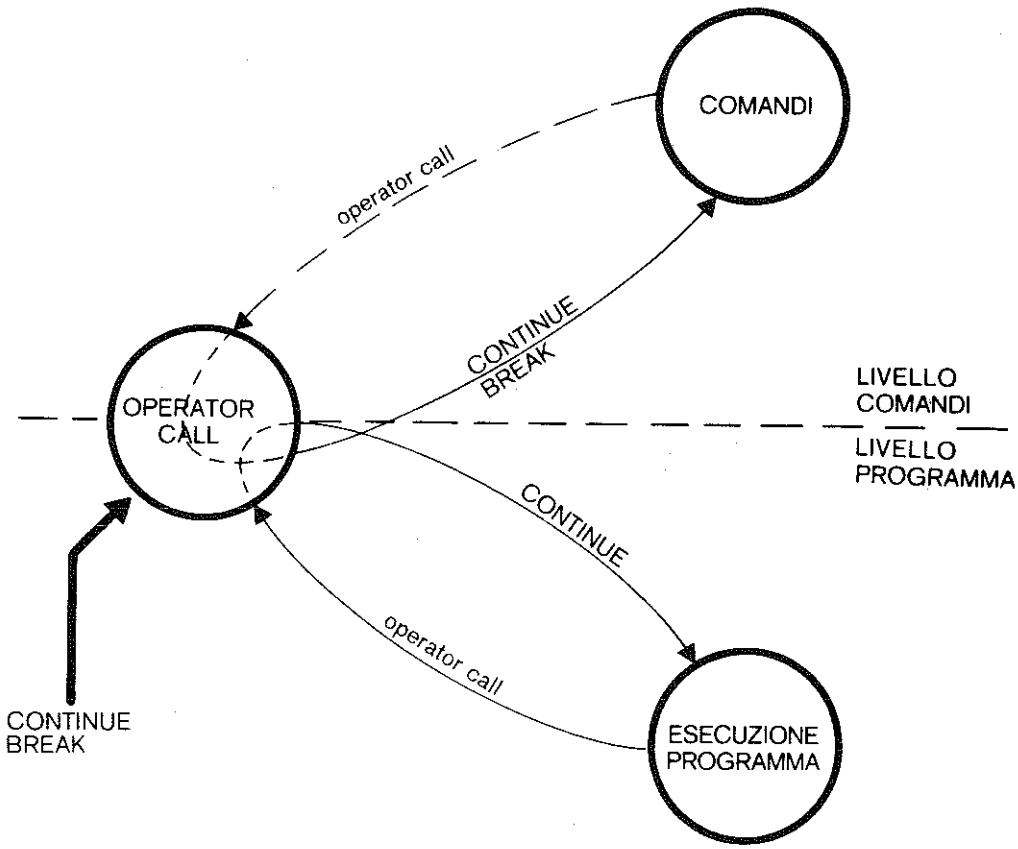


Figura 3-8 Stato 'OPERATOR CALL'

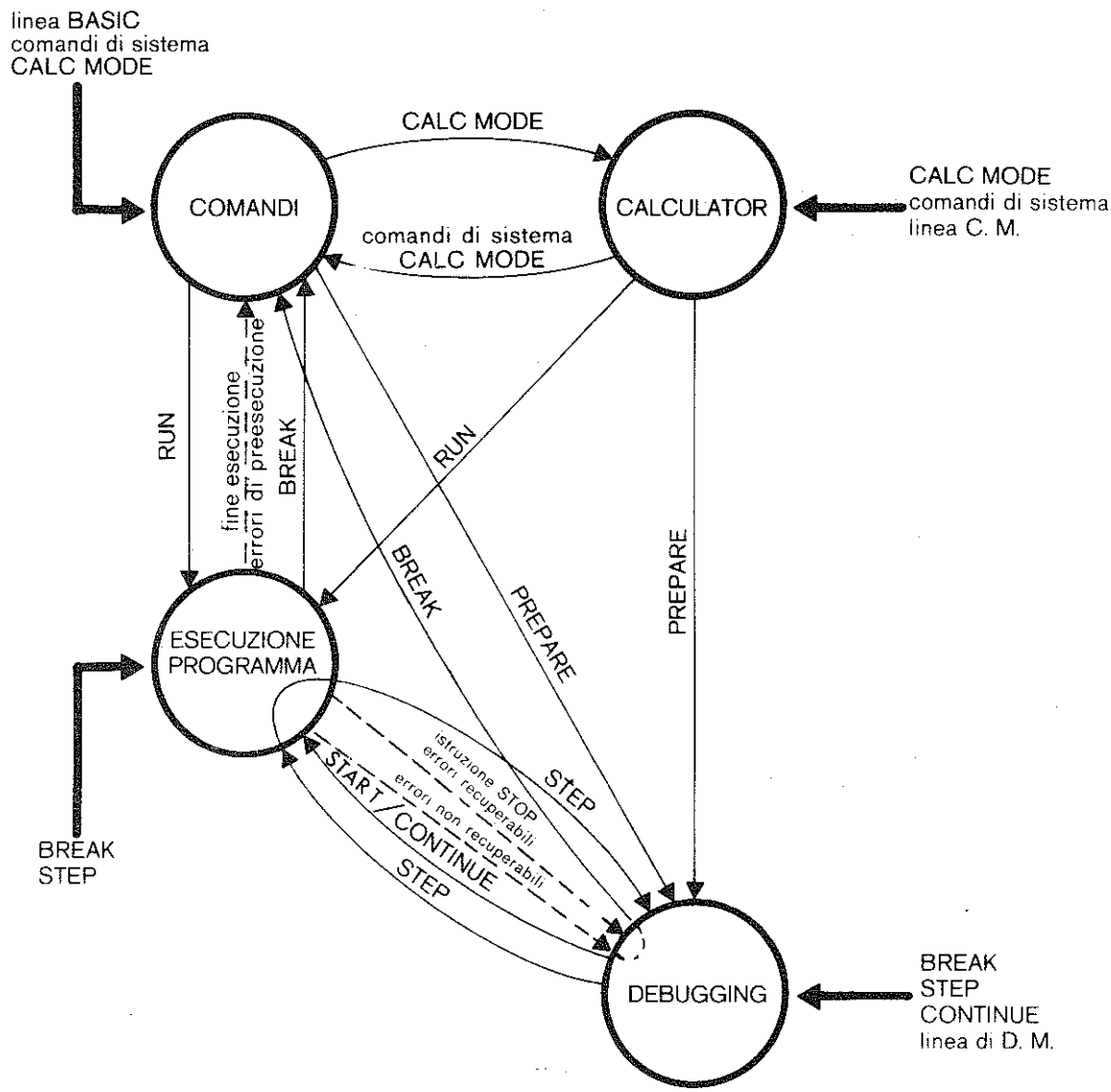


Figura 3-9 Stato 'CALCULATOR'

Va rilevato che nello stato 'calculator' l'utente colloquia direttamente con la macchina virtuale SLP: in tal modo è possibile ottenere la massima rapidità di calcolo quando occorre valutare espressioni aritmetiche. La stessa facility è disponibile nello stato di debugging.

La seguente tabella fornisce, per ogni stato, la configurazione delle luci che lo individua. I simboli utilizzati hanno il seguente significato:

- Ø : luce spenta
- 1 : luce accesa (nel caso di running, indifferentemente fissa o lampeggiante)
- * : luce accesa lampeggiante
- ⊡ : luce accesa fissa

LUCE \ STATO	OFF	COMANDI	CALCULATOR	OPERATOR CALL	ESECUZIONE PROGRAMMA	DEBUGGING	RICHIESTA INPUT
RUNNING	0	1	1	⊡	*	1	⊡
STEP	0	0	0	0	0	1	0
CONTINUE	0	0	0	0	1	0	0
CM→COM	0	0	1	0	0	0	0
BREAK	0	0	0	1	1	1	1

Nota: Le luci di Console possono assumere altre configurazioni, in relazione a condizioni di errore. Le informazioni al riguardo sono reperibili nel cap. 4.

4. INIZIALIZZAZIONE DEL SISTEMA

L'inizializzazione del sistema è conseguente all'accensione della macchina oppure ad una riconfigurazione a seguito di comando OPTIONS o CONFIGURE.

Al termine dell'inizializzazione il sistema è disponibile per l'introduzione di comandi, linee di programma o di testo, linee di calculator mode.

L'inizializzazione avviene in tre fasi: una prima fase, portata a termine da un modulo di software di base registrato in memoria ROM, consiste in una verifica della funzionalità hardware del sistema e nella successiva eventuale autodiagnostica. La seconda fase è espletata dal bootstrap, un modulo anch'esso residente in ROM, che provvede a caricare in memoria alcuni altri moduli di software di base, tra i quali il Preset, che porta a termine la terza fase dell'inizializzazione. Durante questa fase vengono caricati in memoria altri moduli di software, e alcune aree di memoria vengono inizializzate in modo opportuno. Al termine dell'operazione la memoria ha l'assetto tipico degli stati del sistema che appartengono al "livello comandi" (programma utente in fase di editing). Il Preset cede infine il controllo al modulo software responsabile della gestione degli input introdotti dall'utente.

Autodiagnostici

Il modulo che svolge la prima fase di inizializzazione esegue dei controlli diagnostici preliminari su:

Console e Cicalino: Vengono accese le 11 lampade di Console e viene fatto suonare il cicalino, per consentire all'utente di verificare il corretto funzionamento della Console. Le lampade rimangono accese per la durata fisica del caricamento, e vengono completamente spente alla fine. Se vi sono state anomalie od errori, le lampade vengono lasciate accese in particolari configurazioni (vedi configurazioni errori sulla Console).

Unità Centrale: Viene testato il corretto funzionamento delle micro-istruzioni dell'Unità Centrale (per esempio vengono caricati due registri con uguale contenuto e viene poi eseguito il confronto: se l'esito è positivo, le micro-istruzioni di assegnazione e controllo funzionano correttamente). L'anomalo funzionamento di una o più micro-istruzioni viene segnalato sulla Console.

Memoria: La memoria viene sondata dall'indirizzo ØØØØ su moduli di 2K byte verificando se vi è ROM, RAM o nulla. Nel primo caso viene eseguito il calcolo del CRC (codice ciclico), nel secondo il prova RAM, e nel terzo si passa all'analisi dei 2K byte successivi sino alla fine della memoria presente.

Caricamento da FDU: Viene verificata la presenza sulla unità Ø (inferiore) del dischetto Sistema; se questo è presente viene letto e caricato in Memoria il Software Residente. Se il dischetto nell'unità Ø è assente o non è un dischetto Sistema, le verifiche vengono svolte sull'unità 1 (superiore). Durante questa fase vengono fatti controlli diagnostici anche su:

- funzionamento dell'Unità floppy disk

- disponibilità di Memoria per il caricamento del Software Residente

- validità delle tracce interessate del dischetto

Spegnimento lampade di Console: Lo spegnimento delle lampade di Console segnala la fine del caricamento del Software Residente.

Lancio del bootstrap: Dopo i controlli diagnostici sulla memoria viene ceduto il controllo all'inizializzatore vero e proprio (bootstrap).

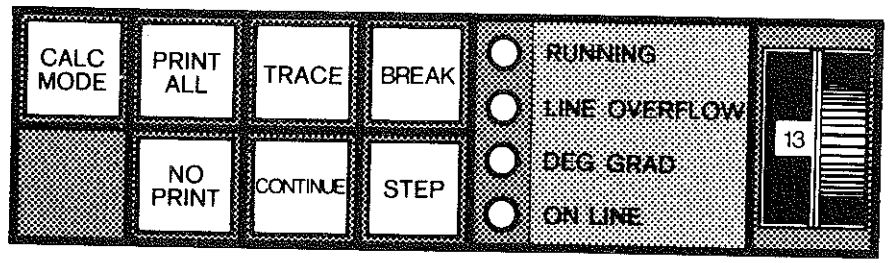


Figura 4-1 La console

Le configurazioni di luci di console indicano durante la fase di controlli diagnostici particolari tipi di errori sia Hw sia Sw.

Per rendere più semplice la spiegazione si suddividono le luci di console come in figura 4-2 e gli errori in gruppi.

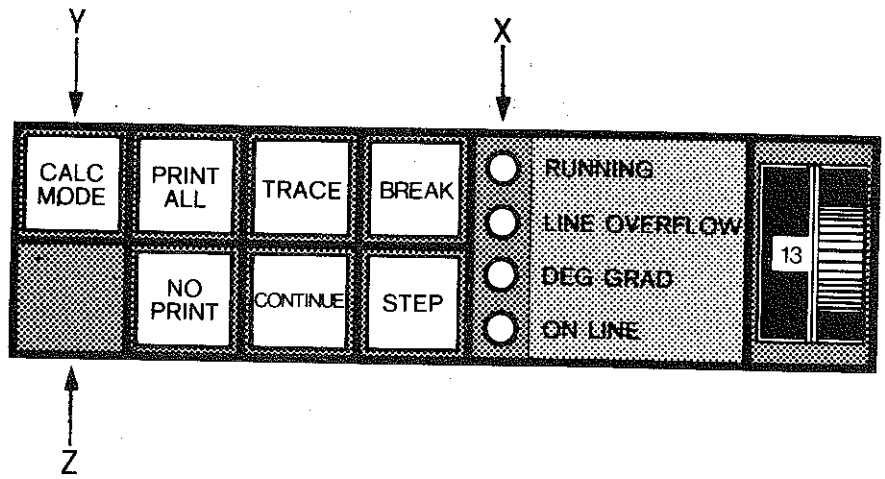


Figura 4-2 Console di macchina

Nel seguito, una proposizione del tipo Z = 4 starà ad indicare che il tasto NO PRINT del gruppo di luci Z è acceso; Z = 6 significherà che sono accesi sia il tasto NO PRINT che il tasto CONTINUE .

Gruppo A:

Z = 1
Y e X variabili

Assenze di risposta in prova memoria. Occorre l'intervento del tecnico; le configurazioni riscontrate permettono di identificare le piastre degredate.

Gruppo B:

Z = 2
Y e X variabili

Guasto a parte delle ROM. Occorre l'intervento del tecnico.

Gruppo C:

Z = 3
Y e X variabili

Piastre RAM guaste. Occorre l'intervento del tecnico.

Gruppo D:

Z = 4
X = Ø ÷ 8
Y variabile

Governo FDU guasto. Occorre l'intervento del tecnico.

Gruppo E:

Z = 5 ÷ 6
Y e X variabili

Errore nel trasferimento del software di floppy disk in memoria interna; se l'errore si verifica ripetendo l'operazione di inizializzazione, occorre far intervenire il tecnico.

Gruppo F:

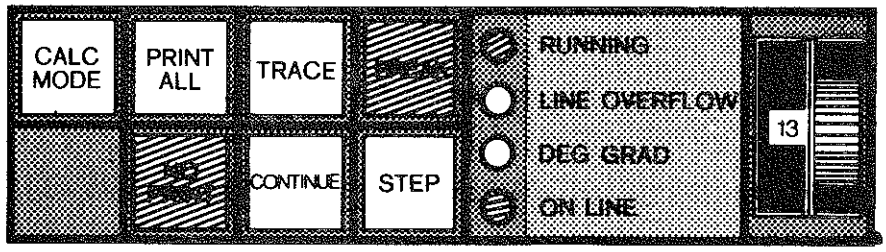
Z = 7
X = Ø
Y = 1 ÷ 8

Guasto all'unità centrale. Occorre l'intervento del tecnico.

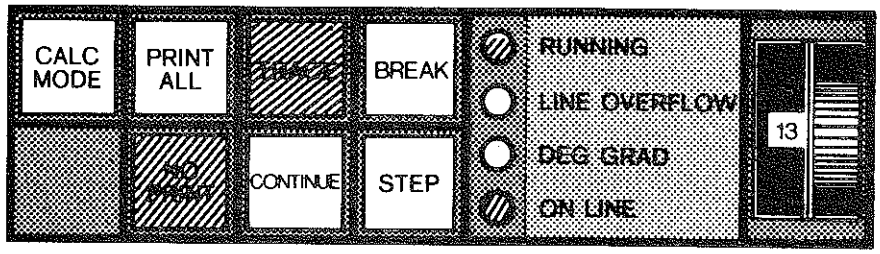
Gruppo G:

errori facilmente rimediabili da parte dell'utente (i tasti retinati indicano luce accesa).

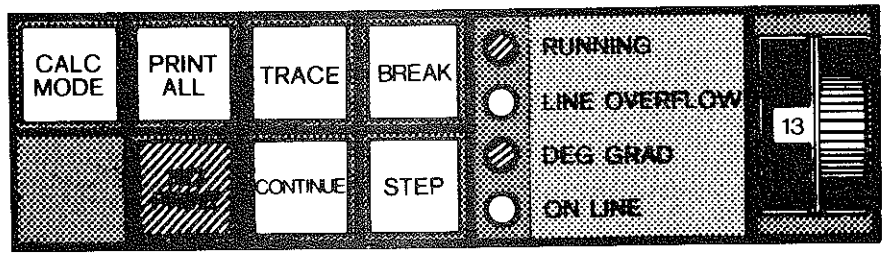
1. Manca il Floppy-Disk su di una unità.



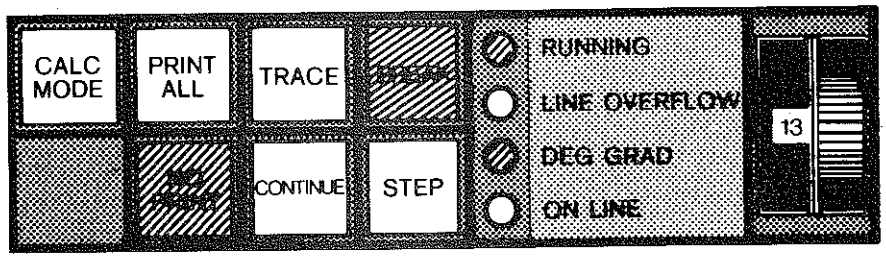
2. E' inserito un Floppy-Disk utente anzichè Floppy-Disk Sistema.



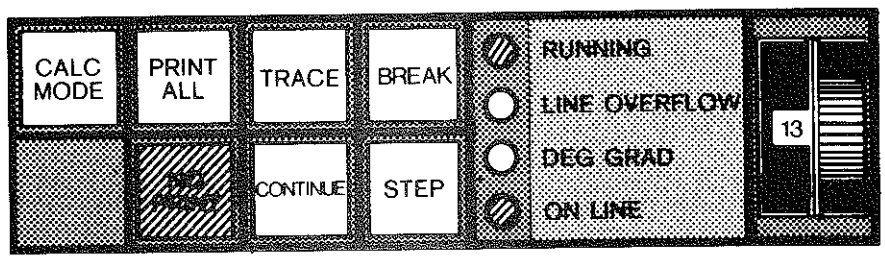
3. Floppy-Disk Sistema non corretto.



4. Floppy-Disk Sistema non corretto.



5. Sportello unità Floppy-Disk aperto.



Funzioni svolte dal bootstrap

Il bootstrap provvede a caricare in memoria:

- la SEGTAB (vedi cap. 3)
- la tabella dove sono registrate le stringhe di caratteri associate ai tasti funzione (FKEYTAB)

- il modulo software PRESET

- il segmento di software residente che contiene il PIH e i moduli di PIOCS

Il bootstrap carica inoltre dai corrispondenti campi del disco sistema l'ultima descrizione delle opzioni specificata dall'utente e l'ultima data introdotta. Viene quindi esaminata la configurazione hardware per verificare quali periferiche sono collegate all'unità centrale (display, stampante, canali IPSO, ecc.). Una sua descrizione viene posta nell'area di comunicazione dati del sistema (COMAREA). Ultimate queste operazioni, il bootstrap cede il controllo al PRESET. Nella fig. 4-3 è schematizzata la configurazione della memoria in questo istante.

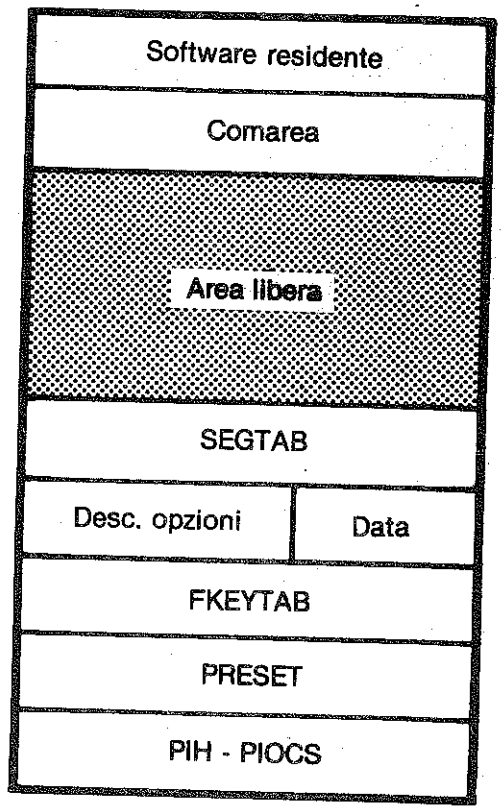


Figura 4-3 La memoria RAM all'atto della chiamata del PRESET

Funzioni svolte dal PRESET

Il modulo PRESET svolge le seguenti funzioni:

- rilocazione delle tabelle SEGTAB e FKEYTAB nelle zone di memoria interna loro riservate

- analisi della congruenza tra le opzioni prescelte dall'utente e la configurazione hardware del sistema

- caricamento in memoria interna delle parti di S.O. relative alle opzioni scelte dall'utente

- caricamento in memoria della parte restante di software residente. Viene caricato il MONITOR, cui in seguito verrà ceduto il controllo, ed il supervisore dello stato "CALCULATOR MODE" (Calculator Mode Handler o CMH)

- registrazione nella tabella SEGTAB degli indirizzi di memoria dei segmenti di software residente ed opzionale relativi alle opzioni scelte e caricate in memoria

- inizializzazione, con valori appropriati, dei campi appartenenti alla COMAREA (descrittori di tabelle, ecc.)

- lettura dei parametri relativi alle librerie utente in linea e loro registrazione nella COMAREA

Al termine di questa serie di operazioni l'inizializzazione del sistema è terminata. Viene perciò abilitata la tastiera, che durante tutto questo periodo era rimasta disabilitata, e il controllo viene ceduto al MONITOR.

Nella figura 4-4 è schematizzata la configurazione della memoria in questo istante.

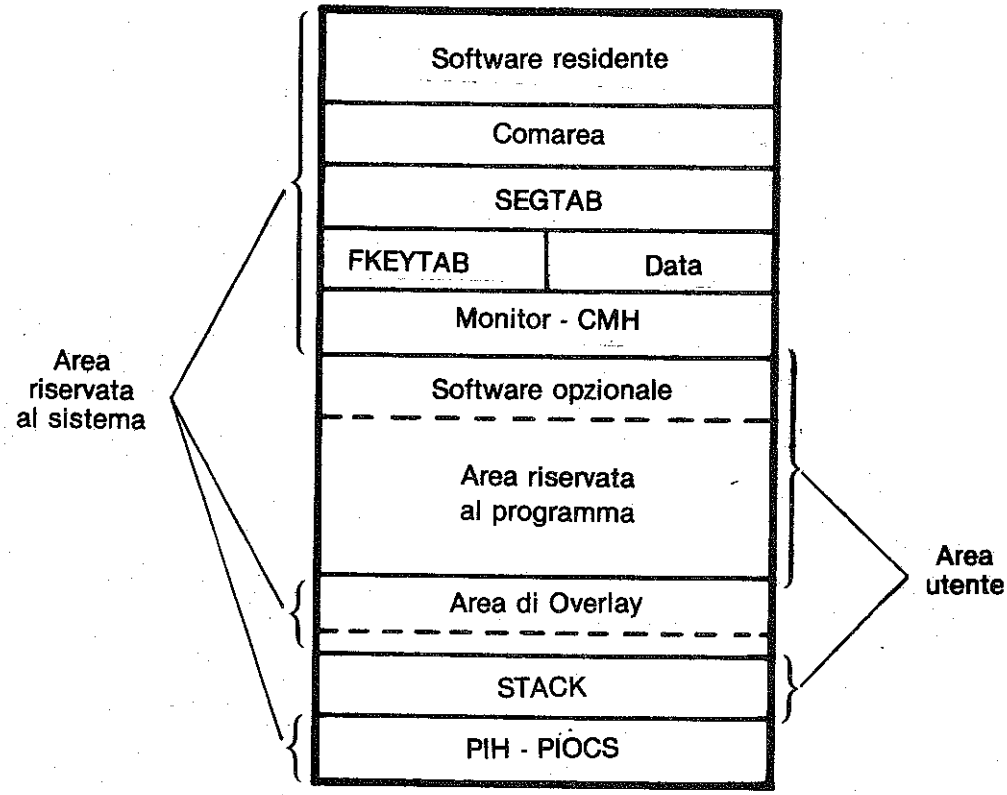


Figura 4-4 La memoria RAM al termine dell'inizializzazione del sistema

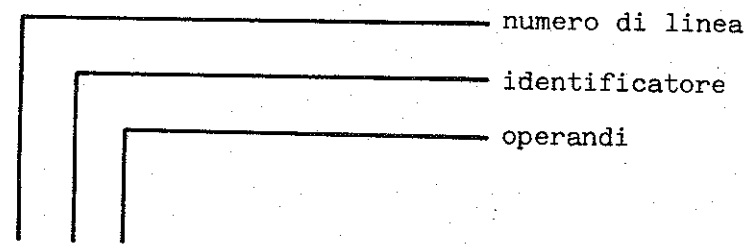
5. CREAZIONE ED EDITING DI UN PROGRAMMA E DI UN TESTO

Struttura di un programma BASIC

Un programma BASIC è costituito da un insieme di linee ciascuna delle quali consta di una e una sola istruzione. Ogni istruzione è composta di più elementi ordinati come segue:

numero linea IDENTIFICATORE..... operandi.....

Esempio:




```
1ØØ LET A=5
11Ø LET B=C=A
12Ø PRINT A,B,C
```

Il numero di linea è un intero N ($1 \leq N \leq 9999$) che denota la posizione logica di un'istruzione all'interno di un programma: le istruzioni si intendono ordinate per numero di linea crescente, indipendentemente dall'ordine in cui sono state introdotte.

Il numero di linea costituisce, inoltre, l'etichetta dell'istruzione: con esso è possibile riferirsi alla istruzione sia da parte di altre istruzioni del programma, sia nelle operazioni di sistema. Non possono coesistere due istruzioni aventi lo stesso numero di linea.

L'identificatore, costituito da una, due o tre parole riservate, determina il tipo dell'istruzione; esso può essere omesso nel caso dell'istruzione LET.

Dopo ogni parola riservata possono comparire uno o più operandi, separati tra loro da elementi separatori; un operando può consistere in una costante, una variabile, un'espressione, una relazione di confronto. Gli spazi

in genere, non sono significativi nelle istruzioni; fanno eccezione gli spazi contenuti nelle costanti alfanumeriche e nelle immagini della linea di stampa. Gli spazi non sono invece ammessi nei seguenti casi:

- all'interno del numero di linea

- all'interno di una parola riservata

- all'interno di un nome di variabile o di funzione (standard di sistema o definita dall'utente)

- all'interno di un dato numerico

In particolare, tra il numero di linea e l'identificatore, tra l'identificatore e un operando, tra i singoli operandi, può essere inserito un numero arbitrario di spazi (eventualmente nessuno).

L'esecuzione delle istruzioni del programma avviene, di norma, secondo numero di linea crescente. Tale ordine di esecuzione può essere alterato mediante istruzioni di controllo. Le istruzioni contenute in un programma possono essere esecutive oppure non esecutive. Un'istruzione è esecutiva quando, se incontrata nel corso dell'esecuzione del programma, dà luogo ad una ben determinata operazione; un'istruzione è non esecutiva quando ha una funzione dichiarativa o di commento, e comunque, se incontrata nel corso dell'esecuzione del programma, non dà luogo ad alcuna azione da parte del sistema.

Struttura di un testo

Un testo è costituito da un insieme di linee dotate di numero di linea e ordinate, a somiglianza delle linee di programma, per un numero di linee crescente; le linee di testo possono contenere qualsiasi cosa sia esprimibile per mezzo di codici ISO.

Questo tipo di struttura ordinata consente, tra l'altro, di effettuare numerosi tipi di analisi linguistiche (locuzioni ricorrenti, frequenza di sintagmi, ecc.), nonchè di archiviare sotto forma di testo qualsiasi tipo di comunicazione epistolare, commerciale oppure no. In particolare, un testo può essere costituito da una sequenza di istruzioni in linguaggio BASIC. Esse vengono però conservate in formato sorgente, e possono successivamente essere compilate mediante il comando

COMPILE; soltanto in questo momento verrà effettuata, sulle linee del testo, l'analisi sintattica che si effettua in programmi BASIC.

Le linee contenute nei file testo e le linee contenute nei file programma hanno, come caratteristica comune, il numero di linea. Questa caratteristica consente di effettuare nello stesso modo sui file di programma e sui file testo un certo numero di operazioni, come, per esempio, l'editing, la visualizzazione, l'ordinamento automatico delle nuove linee introdotte, la cancellazione di linee, il listing, totale o selettivo. Inoltre i file testo, così come i file programma, possono essere registrati su una memoria esterna, e successivamente ricaricati da libreria in memoria interna.

Working file e linea corrente

Il working file è quella parte di memoria interna che il sistema riserva al programma, oppure al testo, sul quale l'utente sta, al momento, operando. Si tratta di un'area di lavoro nella quale l'utente può introdurre un programma o un testo, editarlo, effettuare su di esso altre operazioni di vario genere. Il working file, essendo in memoria interna, è labile: cioè se la macchina si spegne il suo contenuto va perduto.

La linea corrente è quella linea, presente nel working file (e quindi già introdotta nel sistema e da esso accettata), alla quale si può accedere con alcune operazioni privilegiate: in particolare, FETCH senza argomento. A seconda della operazioni cui ci si riferisce, la linea corrente può essere definita in modo diverso come:

- l'ultima linea introdotta senza errore
- l'ultima linea visualizzata per mezzo di comandi: FETCH, [↑], [↓]
- l'ultima linea stampata a seguito del comando LIST
- l'ultima linea di programma di cui è stata intrapresa l'esecuzione

Introduzione di un programma o di un testo

Subito dopo l'accensione della macchina, così come dopo l'esecuzione del comando OPTIONS, il working file risulta vuoto, ed il sistema è predisposto per l'introduzione da tastiera di un nuovo programma.

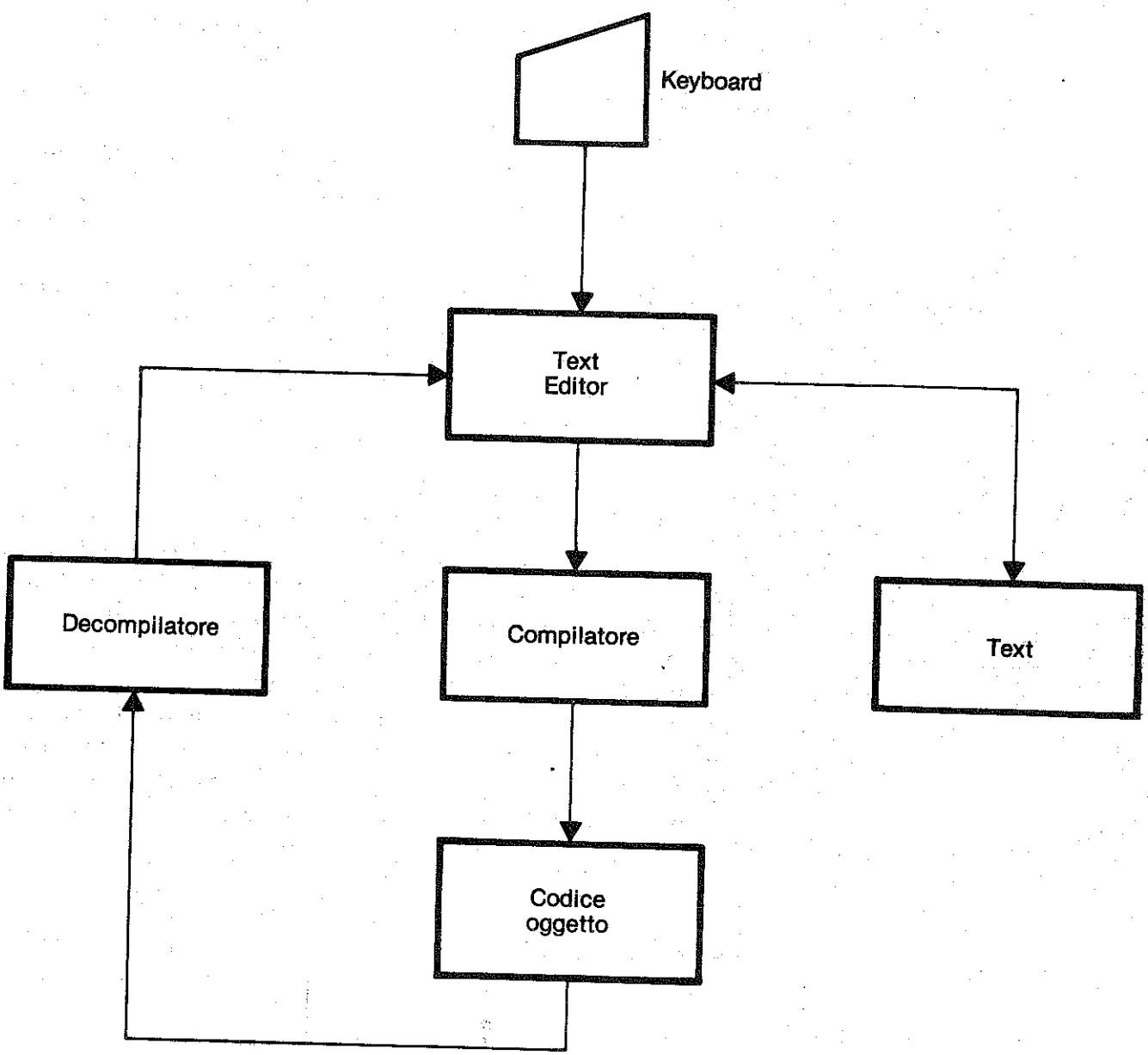
Quando invece il working file contiene un testo o un programma introdotti in precedenza, il sistema deve essere inizializzato per mezzo di uno dei comandi NEW, TEXT, OLD, RUN.

L'introduzione di un programma o di un testo da tastiera nel working file avviene linea per linea. Se corretta sintatticamente, la linea digitata, dopo il comando di fine linea, viene acquisita dal sistema e compilata. Poichè il compilatore vede ogni singola istruzione BASIC come un input a se stante, una modifica apportata ad un programma non comporta la ricompilazione dell'intero programma, ma soltanto la ricompilazione delle istruzioni BASIC interessate dalla modifica.

Operazioni di editing

I blocchi di codice in memoria sono ordinati sequenzialmente secondo numeri di linea crescenti; la struttura dei programmi BASIC e dei testi rende particolarmente semplici le operazioni di inserimento, sostituzione, cancellazione e rinumerazione delle linee. Queste operazioni vengono effettuate sia in programmi compilati, sia su programmi in formato sorgente (testi).

Il modulo che svolge le funzioni di editing è l'EDITOR.



Inserimento

L'inserimento di una nuova linea viene effettuato semplicemente introducendo la linea desiderata, dotata di un numero di linea maggiore di quello dell'istruzione che deve precederla e minore di quello dell'istruzione che deve seguirla. L'EDITOR crea una nuova entry nella LNT (vedi cap. 5) e inserisce il blocco di codice al posto che gli compete per ordinamento naturale sui numeri di linea. Aggiorna poi nella LNT i campi indirizzo delle istruzioni spostate.

Sostituzione

Per sostituire una linea è sufficiente introdurre la linea desiderata con lo stesso numero di linea della linea che si vuole sostituire; quest'ultima risulta così automaticamente cancellata dal working file. L'EDITOR sostituisce il codice del tipo di istruzione, modificando eventualmente gli indirizzi riferiti nei vari blocchi di codice. L'entry sostituita viene ricompilata con le informazioni della nuova linea.

Cancellazione

La cancellazione di una o più linee è resa possibile dall'uso del comando DELETE. L'EDITOR cerca nella LNT l'entry corrispondente all'istruzione cancellata, la marca come non definita e provvede a compattare il codice e ad aggiornare gli indirizzi della LNT.

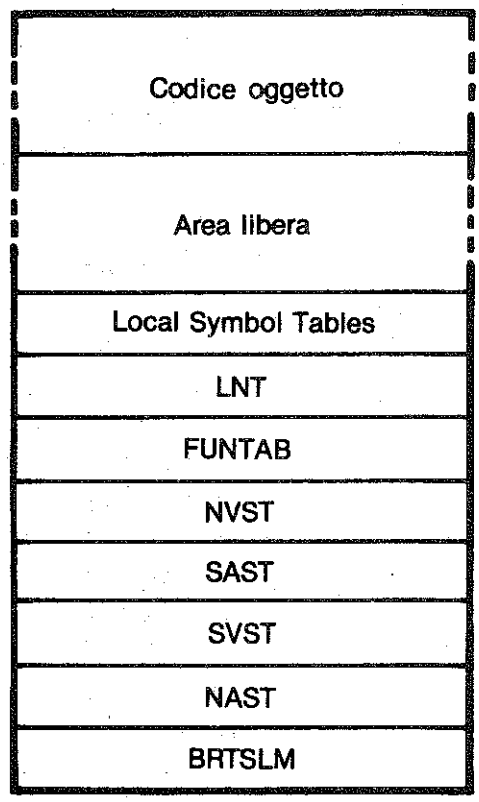
Rinumerazione

L'uso del comando RESEQUENCE consente, nello stato "COMANDI", di ristabilire il "passo" desiderato fra i numeri di linea delle linee introdotte, conservandone l'ordine sequenziale. E' così possibile, soprattutto nel caso di inserimento di più linee consecutive, evitare noiose operazioni di riscrittura oppure l'introduzione di istruzioni di controllo non indispensabili. Nella rinumerazione di un programma, il comando RESEQUENCE provvede automaticamente a correggere i riferimenti, adeguandoli alla nuova numerazione. Ciò non accade nella rinumerazione di un testo.

Quando le linee BASIC vengono introdotte e immediatamente compilate, l'EDITOR aggiorna, inoltre, i contatori di riferimento delle varie tabelle dei simboli.

Mappa di memoria a edit-time

Lo schema seguente rappresenta la mappa della memoria a EDIT-TIME.



Oltre al codice oggetto, sono presenti le tabelle prodotte in fase di compilazione ed editing ed un'area libera. Gli elementi delle tabelle NVST, SVST, NAST, SAST sono le variabili globali del programma, mentre gli elementi della tabella LVST sono le variabili locali.

La tecnica del Character Processing

Le parti non residenti nel sistema operativo vengono caricate in memoria soltanto quando si rendono necessarie: questo, in particolare, avviene per i processor dei comandi e per i moduli del compilatore.

Una sofisticata tecnica di esame della stringa in ingresso, denominata character processing, permette di evitare che l'esecuzione di un comando o la processazione di una linea BASIC venga ritardata del tempo occorrente per il caricamento da disco dell'opportuno modulo di software. Il ritardo viene annullato sovrapponendo il tempo di lettura da disco alla fase di introduzione del comando da parte dell'utente.

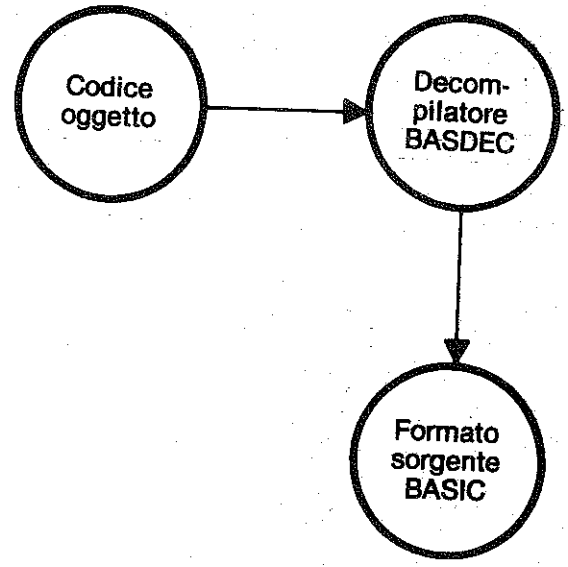
La tecnica consiste nel cercare di riconoscere il comando prima che la sua introduzione sia terminata. Ciò viene effettuato eseaminando i primi caratteri introdotti. Dall'esame del primo carattere (lettera o numero) il sistema è in grado di riconoscere un comando o una linea BASIC. Dall'esame dei primi 3 caratteri (se comando) oppure dei primi 3 caratteri dopo il numero di linea (se linea BASIC), il sistema è in grado di riconoscere il tipo di comando o di istruzione.

Il caricamento dei moduli di software occorrenti avviene quindi in questa fase, senza che sia necessario attendere l'introduzione del carattere (EOL) di fine linea: il tempo di caricamento risulta così invisibile all'utente, e il comando viene servito (o la linea processata) immediatamente. Nel caso in cui la linea input venga modificata, il suo esame viene ripetuto. Il Character Processing viene effettuato dal MONITOR, che provvede al riconoscimento del tipo di comando, e dal BASCOMP che provvede al riconoscimento del tipo di istruzione. Il MONITOR richiama in memoria il processor del comando che è stato riconosciuto (eventualmente il DRIVER del compilatore); il DRIVER del compilatore legge invece il modulo preposto alla compilazione dell'istruzione BASIC introdotta.

Decompilazione

Come per la fase di compilazione, esiste anche nella fase di decompilazione un insieme di moduli del sistema operativo ai quali compete la riconversione in linee sorgente BASIC del codice oggetto generato durante la fasi di compilazione.

L'insieme di questi moduli costituisce il decompilatore BASIC o BASDEC (Basic Decompilator):



Il BASDEC effettua la conversione da codice oggetto (in notazione polacca inversa) a sorgente BASIC; il formato finale è equivalente, ma non necessariamente identico, a quello originalmente digitato dall'utente. Così, ad esempio, se l'utente digita la linea

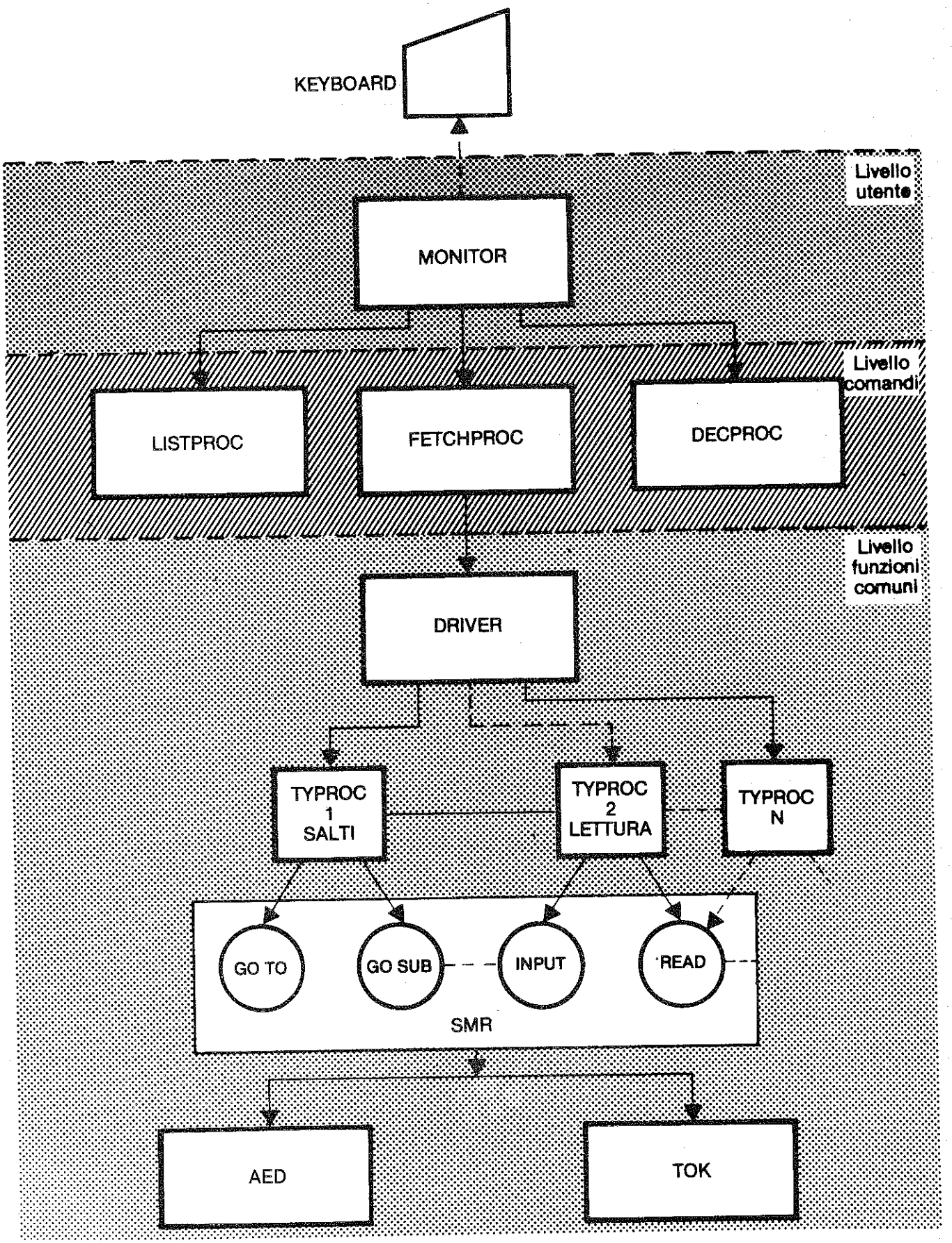
```
4Ø A=B+(C)
```

dopo la decompilazione la linea stessa sarà convertita nel formato

```
4Ø  LET  A=B+C
```

Il decompilatore elimina infatti tutti gli elementi non necessari introdotti dall'utente (spazi superflui, parentesi inutili, ecc.) e introduce alcuni elementi in modo standard (parole chiave opzionali, spaziature, ecc.).

Grazie alle funzioni svolte dal decompilatore, anzichè memorizzare due versioni di un programma (il formato sorgente e il formato oggetto) se ne conserva una soltanto, con conseguente risparmio di occupazione su disco. Il decompilatore opera nella fase di editing, ed ha quindi a disposizione, oltre al codice oggetto, tutte le tabelle che ne costituiscono il supporto.


KEYBOARD
Livello utente
MONITOR
Livello comandi
LISTPROC
FETCHPROC
DECPROC
Livello funzioni comuni
DRIVER
TYPROC 1 SALTI
TYPROC 2 LETTURA
TYPROC N
GO TO
GO SUB
INPUT
READ
SMR
AED
TOK

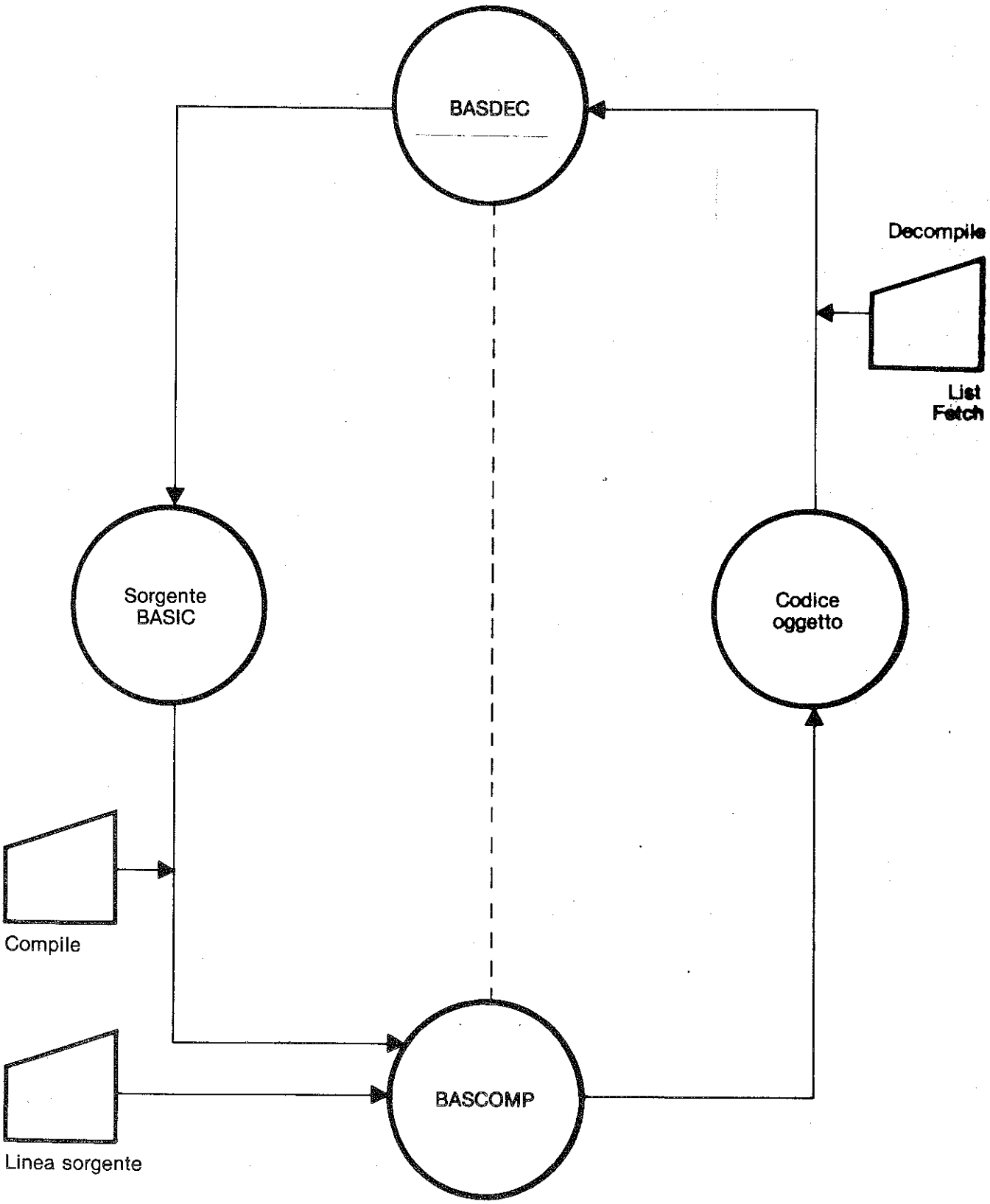
- Driver: è il modulo che interfaccia con il MONITOR. Ha la funzione di riconoscere il tipo di istruzione BASIC e di cedere il controllo all'opportuno modulo di TYPROC

- TYPROC (Type Processor): sono un insieme di moduli, ognuno dei quali, lanciato dal DRIVER, riconosce l'istruzione che deve essere decompilata e provvede a lanciare la relativa SMR

- SMR (Statement Management Routines): sono una serie di moduli ognuno dei quali è specializzato per il trattamento di un solo tipo di istruzione

- TOK (Tokenizer): è un modulo che scinde i blocchi di codice oggetto costituiti di espressioni algebriche nella serie di singoli elementi (operandi, operatori) di linguaggio algebrico che lo compongono

- AED (Algebric Expression Decompiler): è il modulo che, sulla base dell'input fornito dal TOK, provvede alla corretta decompilazione del codice oggetto


BASDEC
Decompile
List
Fetch
Sorgente
BASIC
Codice
oggetto
Compile
BASCOMP
Linea sorgente

Utilizzazione

Il BASDEC viene richiamato dai comandi LIST, FETCH e DECOMPILE. In quest'ultimo caso il codice oggetto esistente viene sostituito, al momento della decompilazione, con il nuovo codice sorgente prodotto. Il codice oggetto potrà essere ottenuto nucvamente con una ulteriore fase di compilazione.

6. COMPILAZIONE, PREESECUZIONE ED ESECUZIONE PROGRAMMI

Testi e programmi

Il sistema P6060 accetta linee scritte in linguaggio BASIC di tre tipi differenti:

1. Linee di BASIC immediato (CALCULATOR-MODE e DEBUGGING-MODE): linee contenenti espressioni aritmetiche eventualmente contenenti richiami di funzioni BUILT-IN, riferimenti a risultati precedentemente ottenuti, assegnazione dei tasti funzione, gran totale, richiami di funzione utente e (solo in Debugging-Mode) riferimenti a variabili utente.

2. Linee appartenenti ad un programma BASIC, introdotte da tastiera e processate (analizzate e compilate) in modo incrementale.

3. Linee BASIC appartenenti ad un programma memorizzato in un file testo (le linee BASIC vengono memorizzate direttamente in codice ISO), che devono ancora essere processate in modo completo.

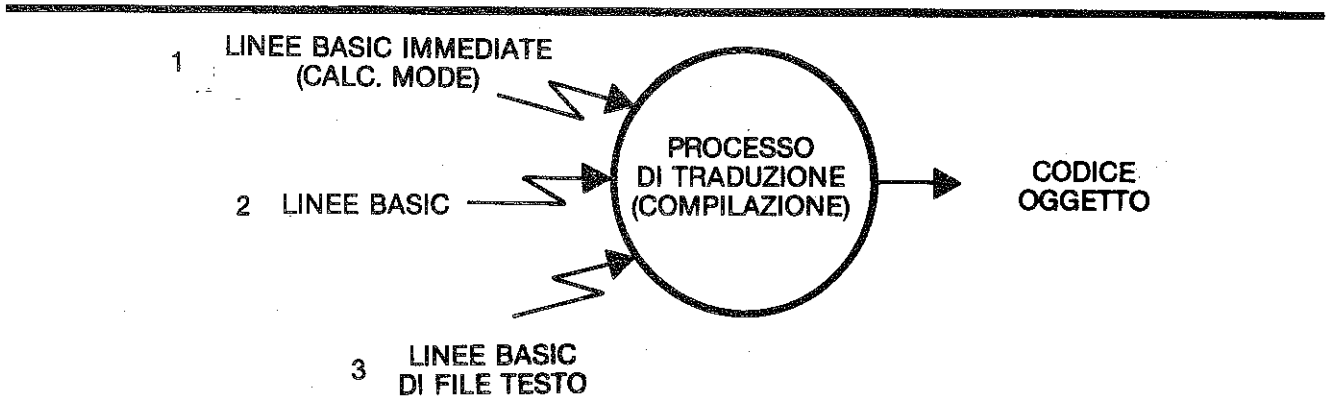
Un modulo di interfaccia con l'utente (MONITOR), entra in azione ogniqualvolta si verifica un'introduzione da tastiera (linea BASIC, COMANDO, C.M.). Il monitor riconosce il tipo di input e passa il controllo ad un modulo di compilazione (BASCOMP).

Compilazione ed editing

Il BASCOMP è l'insieme dei moduli del sistema operativo che provvedono alla compilazione delle linee BASIC. Il BASCOMP opera in modo incrementale: ogni linea di input viene cioè esaminata singolarmente, indipendentemente dalle altre. Ogni linea BASIC in input è una sequenza di caratteri ISO, ed è detta linea in formato sorgente, o linea sorgente. L'insieme delle istruzioni BASIC in formato sorgente costituisce un programma sorgente.

Tutte le volte che viene introdotta una linea il compilatore ne esegue il riconoscimento, e successivamente richiama in memoria le opportune routine di compi-

lazione. In un'unica passata sul programma sorgente il compilatore produce un codice in formato oggetto (codice oggetto).



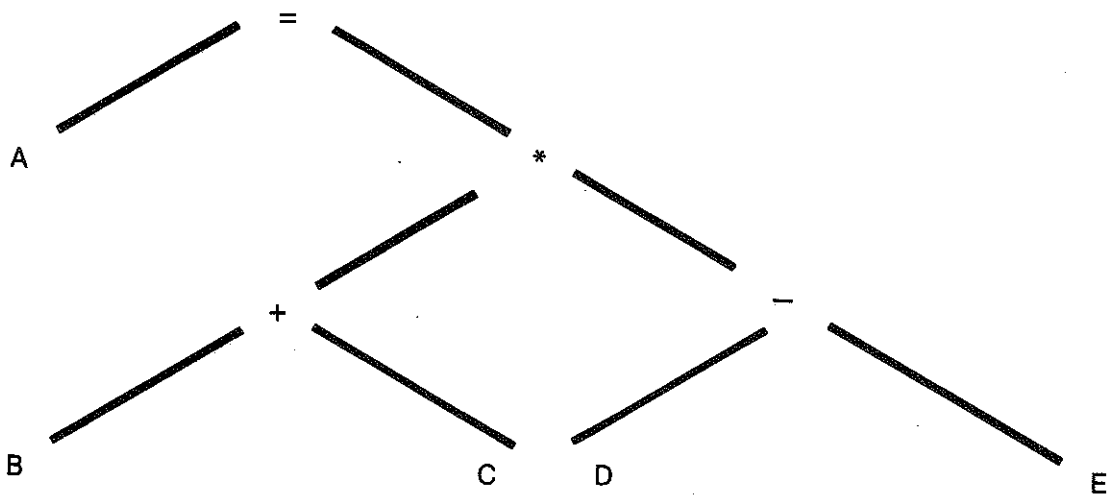
Il codice oggetto è scritto in linguaggio algebrico (ALP) secondo la notazione polacca inversa.

Il suo funzionamento è basato sull'uso di uno stack; ciò consente di risparmiare parte delle memorizzazioni che si rendono necessarie ad ogni passo della compilazione incrementale (vedi cap. 3).

Un esempio di traduzione:

10 LET A = (B + C) * (D – E)

ALBERO SINTATTICO



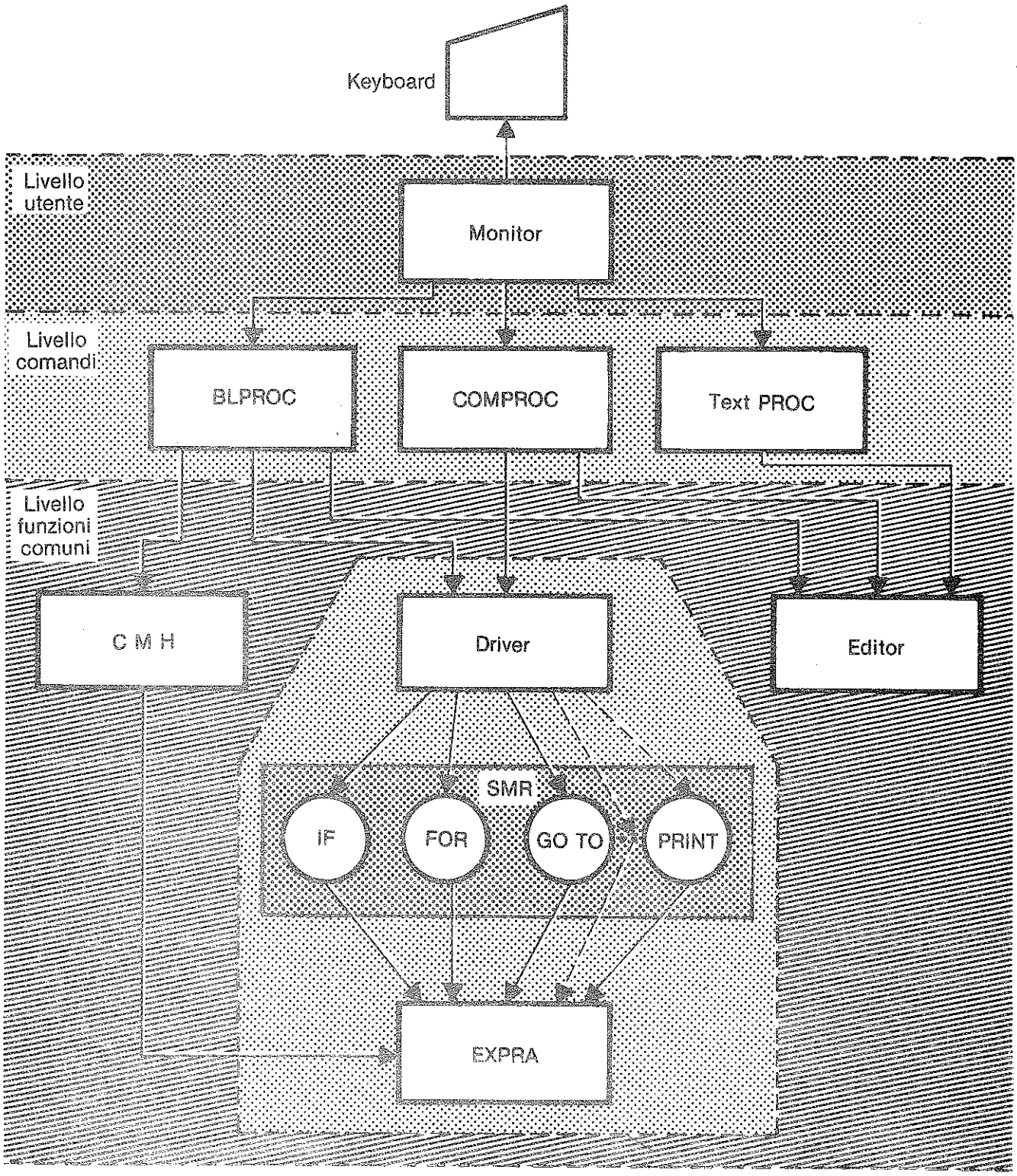
FORMA POLACCA INVERSA

ABC + DE – * =

CODICE ALP

```
PUSH B
PUSH C
ADD
PUSH D
PUSH E
SUB
MUL
```

Oltre al codice oggetto il compilatore produce anche una serie di tabelle necessarie per l'editing e l'esecuzione.


Keyboard
Livello utente
Monitor
Livello comandi
BLPROC
COMPROC
Text PROC
Livello funzioni comuni
C M H
Driver
Editor
SMR
IF
FOR
GO TO
PRINT
EXPRA

BASCOMP

Architettura del compilatore (BASCOMP)

Il BASCOMP risulta composto da diverse routine suddivise in tre gerarchici:

- Driver
- Statement Management Routine (SMR)
- EXPRA

1° livello: Driver: E' il modulo di compilazione che interfaccia con il sistema. Ha il compito di riconoscere il tipo di istruzione e di trasferire il controllo alla routine adibita al trattamento dell'istruzione individuata.

2° livello: Statement Management Routine (SMR): Ognuna di queste routine è adibita al trattamento di un solo tipo di statement. Queste routine sono richiamate dal driver e ad esso rendono il controllo.

3° livello:.....(EXPRA): E' un modulo di utilità che provvede alla compilazione delle espressioni algebriche. Viene chiamato da routine SMR e ad esse rende il controllo.

In figura è illustrata l'architettura del compilatore BASIC (BASCOMP) e le relazioni che lo legano ad altri moduli del sistema operativo. I moduli BLPROC (Basic Line Processor) e COMPROC (Compile Processor) supervedono rispettivamente alla compilazione di un programma BASIC nei casi di introduzione degli statement dopo il comando NEW oppure di compilazione di un testo con il comando COMPILE.

Struttura del codice oggetto

Per struttura del codice oggetto si intende l'insieme delle informazioni generate con la compilazione di un programma. Tali informazioni rendono possibile la corretta interpretazione del programma nei vari momenti della sua evoluzione.

Il programma BASIC che dà origine a questa struttura è composto da istruzioni ognuna delle quali, sottoposta a compilazione, dà luogo alla generazione di un blocco di codice oggetto e di tabelle. La struttura del blocco prodotto dalla compilazione di una istruzione BASIC è la seguente:

Header — E' composto da 3 campi.

Il primo campo contiene il codice operativo dell'istruzione di richiamo del supervisore dell'esecuzione (BASEX). Il secondo campo contiene il numero di linea dell'istruzione BASIC ed un flag che dice se si tratta di un'istruzione eseguibile oppure no. Il terzo campo (presente solo nel caso di istruzioni non esegibili) contiene la lunghezza più uno del CODE BLOCK. Se l'istruzione è non eseguibile, questa informazione permette di oltrepassare il CODE BLOCK e posizionarsi direttamente sull'istruzione successiva.

Code Block — E' un insieme di frasi del linguaggio algebrico che realizzano la funzione indicata nell'istruzione sorgente.

Produzione tabelle

Il compilatore, oltre ai blocchi di codice oggetto, produce 8 tabelle:

- LNT
- BRTSLM
- FUNTAB
- LVST
- NVST
- NAST
- SVST
- SAST

Le ultime 6 contengono i simboli (variabili e funzioni) che compaiono nel programma. Le tabelle vengono utilizzate dal sistema in diverse fasi operative.

Nella fase di editing, insieme ai blocchi di codice, consentono di ritornare alle linee BASIC in formato sorgente attraverso la fase di decompilazione (vedi cap. 6). In preesecuzione vengono utilizzate per la allocazione di spazio alle variabili di programme e per altre funzioni (vedi "Preesecuzione").

Nella fase di esecuzione il posizionamento sui valori di alcuni tipi di variabile avviene attraverso gli indirizzi contenuti nelle corrispondenti tabelle (vedi § "Caricamento variabili nello stack"). Quando dalla fase di esecuzione si passa allo stato di DEBUGGING le tabelle risultano infine necessarie per la corretta compilazione, e conseguente esecuzione,

delle linee di DEBUGGING-MODE (vedi § "Esecuzione").

Line Number Table (LNT): E' una tabella che contiene le informazioni relative alle istruzioni BASIC intese come linee di programma. Ad ogni linea di programma corrisponde un elemento della tabella. Ogni elemento è lungo 7 byte e contiene le seguenti informazioni:

- numero di linea dell'istruzione
- tipo di istruzione (IF, GOTO, ecc.)
- indirizzo di memoria del codice dell'istruzione

A queste informazioni principali ne sono associate altre utilizzate per i controlli globali in fase di preesecuzione. La LNT è l'unica tabella che viene generata anche nel caso di introduzione di un programma BASIC sotto forma di testo. La tabella LNT viene rimossa dalla memoria prima della fase di esecuzione.

Basic Run Time Support Loading Mask (BRTSLM): E' un campo di memoria lungo 16 byte posto nella WFDA. Ogni bit è posto in corrispondenza con un modulo di BRTS. I bit con valore 1 indicano i moduli di BRTS che devono essere associati al programma per mezzo del linkage editor.

Function Table (FUNTAB): E' una tabella contenente le informazioni relative alle definizioni di funzione all'interno del programma. Ad ogni funzione corrisponde un elemento della tabella. Ogni elemento è lungo 9 byte e contiene le seguenti informazioni:

- nome della funzione
- tipo di funzione (numerica o stringa)
- funzione mono o multilinea
- numeri di linea di inizio e fine della definizione di funzione

A queste informazioni principali ne sono associate altre, utilizzate per i controlli globali in fase di preesecuzione.

Local Variable Symbol Table (LVST): Vi è una tabella per ogni definizione di funzione multilinea presente all'interno del programma. Ogni tabella è costituita di 16 elementi, dei quali 15 contengono informazioni riguardanti le variabili locali della funzione (ogni funzione multilinea può utilizzare al più 15 variabi-

li locali; se le variabili locali sono in numero minore, alcuni elementi della tabella non vengono utilizzati), mentre il sedicesimo contiene il valore di ritorno della funzione. Ogni elemento della tabella è lungo 2 byte e contiene le seguenti informazioni:

- nome della variabile locale
- tipo di variabile (numerica o stringa)

A queste informazioni principali ne sono associate altre, utilizzate per i controlli globali in fase di preesecuzione. La tabella LVST viene rimossa dalla memoria prima della fase di esecuzione.

Numeric Variable Symbol Table (NVST): E' una tabella contenente le informazioni che riguardano le variabili numeriche semplici del programma. Ogni elemento della tabella è lungo 2 byte e contiene le seguenti informazioni:

- nome della variabile
- tipo di variabile (singola o doppia precisione)

A queste informazioni principali ne sono associate altre, utilizzate per i controlli globali in fase di preesecuzione.

Numeric Array Symbol Table (NAST): E' una tabella contenente le informazioni che riguardano le variabili numeriche multiple (matrici o vettori) del programma. Ogni elemento della tabella è lungo 10 byte e contiene le seguenti informazioni:

- nome della variabile

- tipo degli elementi della variabile (semplice o doppia precisione)

- numero di dimensioni (1 o 2, a seconda che si tratti di un vettore o di una matrice)

- indirizzo di memoria dello spazio allocato per la variabile (l'allocazione viene effettuata in fase di preesecuzione)

A queste informazioni principali ne sono associate altre, utilizzate per i controlli globali in fase di preesecuzione.

String Variable Symbol Table (SVST): E' una tabella contenente le informazioni che riguardano le variabili alfanumeriche semplici del programma. Ogni elemento della tabella è lungo 7 byte e contiene le seguenti informazioni:

- nome della variabile

- lunghezza di allocazione

- indirizzo di memoria dello spazio allocato per la variabile (l'allocazione viene effettuata in fase di preesecuzione)

A queste informazioni principali ne sono associate altre, utilizzate per i controlli globali in fase di preesecuzione.

String Array Symbol Table (SAST): E' una tabella contenente le informazioni che riguardano le variabili numeriche multiple (matrici o vettori) del programma. Ogni elemento è lungo 10 byte e contiene le seguenti informazioni:

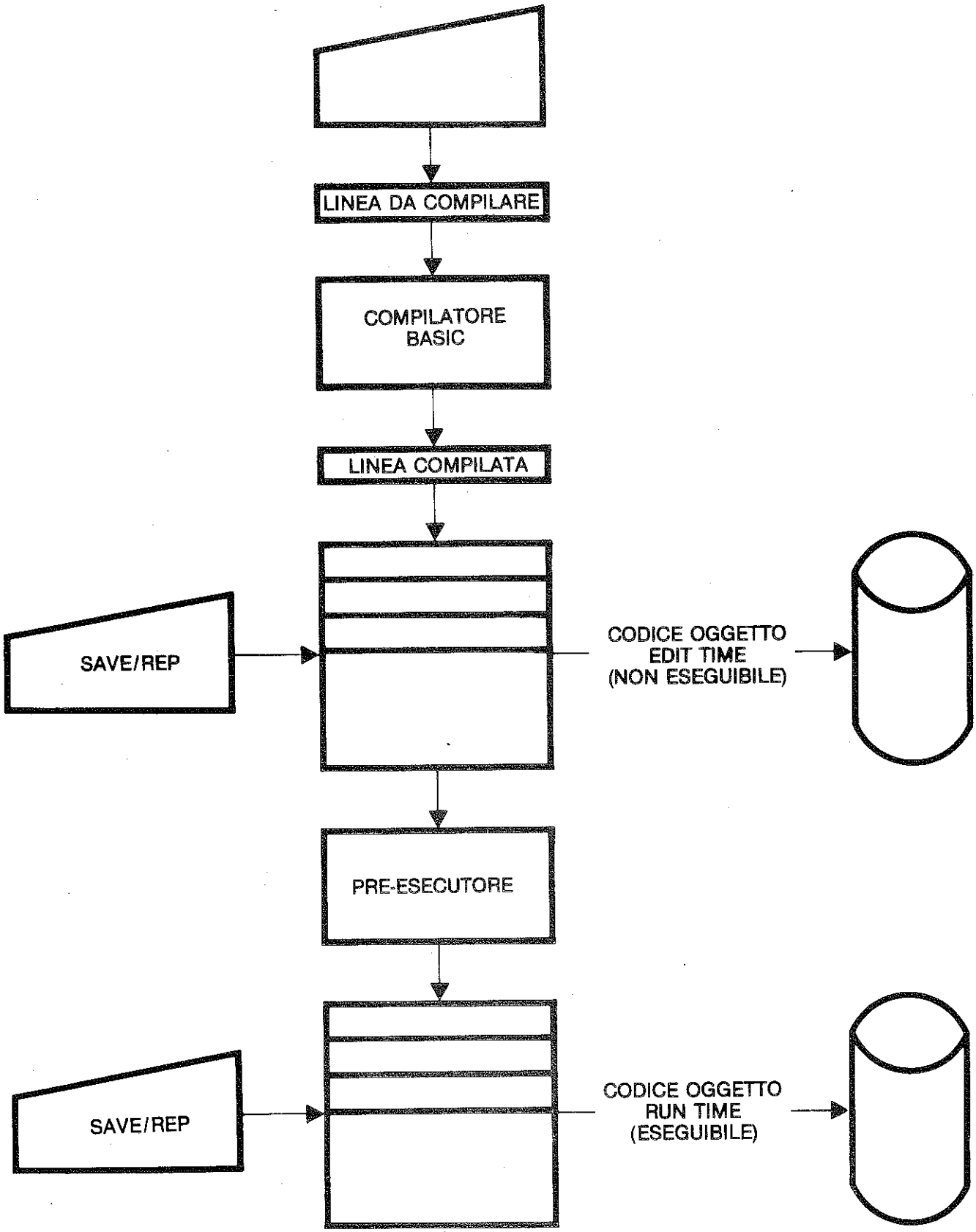
- nome della variabile

- tipo degli elementi della variabile (semplice o doppia precisione)

- numero di dimensioni (1 o 2, a seconda che si tratti di un vettore o di una matrice)

- indirizzo di memoria dello spazio allocato per la variabile (l'allocazione viene effettuata in fase di preesecuzione)

A queste informazioni principali ne sono associate altre, utilizzate per i controlli globali in fase di preesecuzione.


LINEA DA COMPILARE
COMPILATORE BASIC
LINEA COMPILATA
SAVE/REP
CODICE OGGETTO EDIT TIME (NON ESEGUIBILE)
PRE-ESECUTORE
SAVE/REP
CODICE OGGETTO RUN TIME (ESEGUIBILE)

Preesecuzione

Dopo la fase di compilazione il programma BASIC non è eseguibile. Il BASCOMP vede infatti come input le singole istruzioni, e rende in output un oggetto rivolto a soddisfare le operazioni richieste in fase di editing. Perchè il programma sia eseguibile deve prima essere effettuata una serie di operazioni consistenti nell'esaminare in modo globale le istruzioni, e nel dotare il programma di tutte le parti necessarie alla esecuzione. Questo risultato viene raggiunto sottoponendo il programma in formato output del compilatore ad una fase di preesecuzione. Il PREX è il modulo del sistema operativo che svolge questo compito.

Funzioni eseguite

Le funzioni che il PREX deve svolgere possono essere raggruppate in cinque categorie:

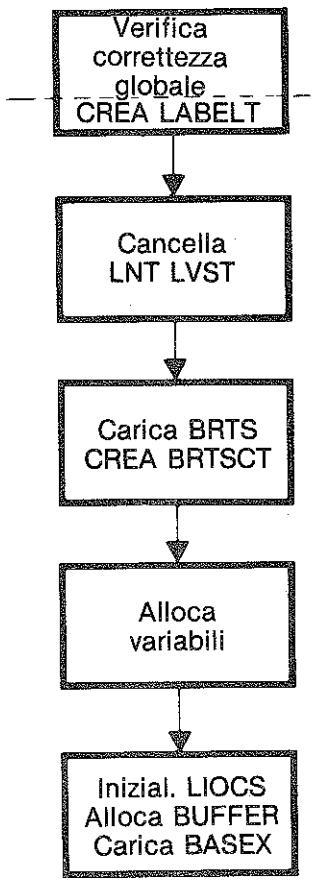
1. Controlli globali di correttezza del programa.

2. Creazione delle tabelle necessarie a RUN-TIME e SWAMP-OUT delle tabelle superflue di EDIT-TIME.

3. Link del BASIC RUN-TIME SUPPORT.

4. Allocazione delle aree necessarie e in fase di esecuzione.

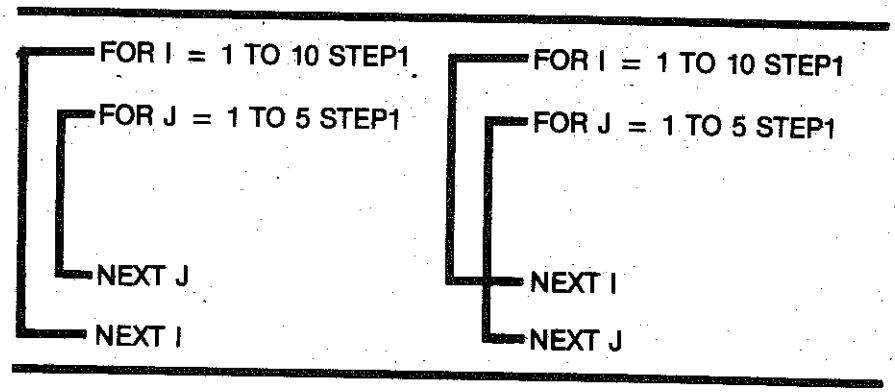
5. Inizializzazione del LIOCS, qualora sia richiesta dal programma.


Verifica
correttezza
globale
CREA LABELT
Cancella
LNT LVST
Carica BRTS
CREA BRTSCT
Alloca
variabili
Inizial. LIOCS
Alloca BUFFER
Carica BASEX

Fase 1 In questa fase si eseguono gli accertamenti di correttezza globale del programma. In particolare si verifica che:

- le istruzioni appartenenti a funzioni multilinea siano compilate correttamente. A seguito di operazioni di editing una funzione multilinea può essere stata definita più volte: le istruzioni che ne fanno parte, relativamente alle variabili che sono argomenti di chiamata a variabili locali, devono essere compilate a rispetto all'ultima definizione data. Se così non è, devono essere ricompilate

- i cicli FOR/NEXT siano annidati correttamente



- le funzioni multilinea non contengano al loro interno altre definizioni di funzioni

- i salti ad istruzioni vengano effettuati in modo corretto (non sono consentiti salti dall'esterno all'interno di cicli FOR/NEXT, ecc.)

Si effettua inoltre la concatenazione dei dati presenti nelle istruzioni DATA che costituiscono il file dati interno.

Fase 2 In questa fase vengono rimosse dalla memoria le tabelle LNT e LVST che servono soltanto in fase di editing. Lo spazio da esse occupato risulta quindi nuovamente utilizzabile.

Viene creata anche la tabella dei riferimenti (LABEL TABLE). La tabella contiene un elemento per ogni linea riferita in istruzioni di salto (lunghezza di un elemento: 3 byte). Le definizioni di funzione sono equiparate a linee riferite. Ogni elemento contiene l'indirizzo di memoria dell'istruzione riferita ed il li-

vello di annidamento.

Fase 3 Il BASIC RUN TIME SUPPORT (BRTS) è l'insieme dei moduli di software che provvedono alla esecuzione delle operazioni di I/O indicate nelle istruzioni BASIC e che la macchina ALP non è in grado di eseguire. Il loro aggancio nella fase di esecuzione avviene con istruzioni del tipo "salto a subroutine"; gli eventuali argomenti di chiamata si trovano nello stack. Istruzioni speciali consentono al modulo di accedere agli argomenti e di ritornare alla macchina algebrica.

Il link del BRTS consta delle seguenti operazioni:

- caricamento in memoria dei moduli di BRTS indicati nella tabella BRTSLM

- creazione della tabella BRTSCT (BRTS Call Table), ogni elemento della quale contiene l'indirizzo di memoria del relativo modulo di BRTS (lunghezza di un elemento: 3 byte)

Quando, in un'operazione di editing, viene cancellata una linea in cui è specificata un'istruzione che in fase di preesecuzione implica un richiamo di BRTS, esempio:

```
1Ø  A=B=5
2Ø  INPUT F
3Ø  C=A+B
4Ø  PRINT C
5Ø  END
DELETE 2Ø
```

il bit della BRTS relativa resta attivato (bit a 1); cioè non viene rimosso l'aggancio alla BRTS relativa, sebbene il compilatore provveda ad eliminare il codice relativo alla linea stessa del codice oggetto prodotto; la BRTS in questione viene rilasciata solo al momento di una DECOMPILE (bit a Ø).

Fase 4 Viene allocato lo spazio per le variabili numeriche e alfanumeriche.

Per ogni variabile numerica presente nella tabella NAST vengono riservati 8 byte . Questo consente, nel caso delle variabili numeriche, un indirizzamento veloce al valore della singola variabile, effettuato

tenendo conto soltanto della posizione della variabile all'interno della tabella e non della singola o doppia precisione delle variabili che la precedono. L'indirizzo di inizio dello spazio allocato per tutte le variabili numeriche viene messo in un apposito campo di memoria. Per ogni variabile multipla numerica (matrice o vettore) vengono riservati N campi (N = numero degli elementi della variabile) di 4 o 8 byte a seconda che sia in semplice o doppia precisione. L'indirizzo di inizio dello spazio allocato per ogni matrice o vettore viene trascritto nell'apposito campo dell'elemento corrispondente della tabella NAST. Ad ogni variabile alfanumerica viene allocato un campo di lunghezza pari alla lunghezza dichiarata della variabile. L'indirizzo di inizio dello spazio allocato per ogni variabile alfanumerica viene trascritto nell'apposito campo dell'elemento corrispondente della tabella SVST. Per ogni variabile multipla alfanumerica, vengono riservati N campi (N = numero elementi della variabile), ciascuno di lunghezza pari alla lunghezza dichiarata per la variabile. L'indirizzo di inizio dello spazio allocato per ogni variabile alfanumerica viene trascritto nell'apposito campo dell'elemento corrispondente della tabella SAST.

Tabella riassuntiva della occupazione

	OCCUPAZIONE
Variabile Numerica	8 Byte
Variabile Multipla Numerica	(N*4) Byte (singola precisione) (N*8) Byte (doppia precisione)
Variabile Alfanumerica	1 Byte
Variabile Alfanumerica Multipla	(N*l) Byte

N = numero degli elementi della variabile
l = lunghezza dichiarata

Fase 5 — Inizializzazione del LIOCS (o di parte di esso) se necessario per l'esecuzione del programma (operazioni su file su disco).

Allocazione dei buffer (uno per ogni file dichiarato nell'istruzione FILES): in pratica vengono predisposti dei canali per operare su file. Per ogni buffer vengono riservati 172 byte per il sistema FDU e 292 byte per il sistema DCU/HDU. Vengono poi allocati i buffer per le operazioni di I/O con periferiche esterne. Le dimensioni di questi ultimi sono indicate con l'apposita istruzione BASIC. Sempre per le periferiche esterne (anche per una sola) vengono allocati 480 byte così ripartiti:

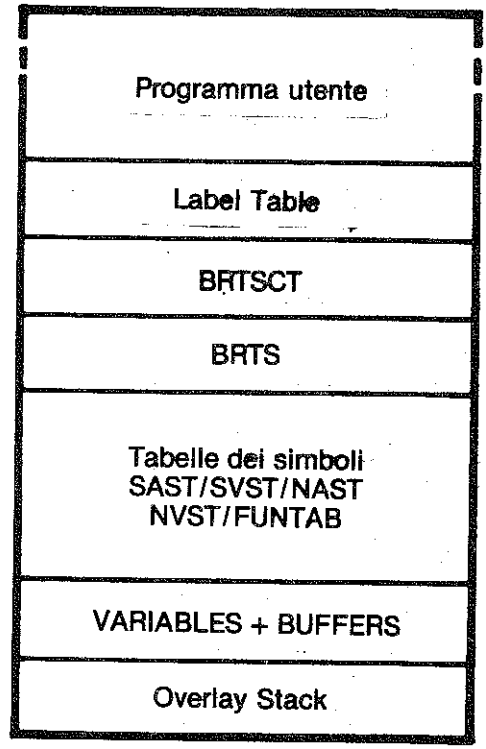
- 32 byte: deposito di stato corrente (2 byte per canale)
  - 16 byte deposito di stato
  - 16 byte separatore

- 448 byte: area di lavoro (28 canali, 16 byte per canale)

Dopo queste fasi viene caricato il supervisore dell'esecuzione (BASEX).

Mappa di memoria a RUN-TIME

La figura mostra la mappa della memoria utente dopo la fase di preesecuzione.



A questo punto il programma è in formato eseguibile e il sistema è nello stato PEX (Program Execution) o DEBUG, entrambi appartenenti al livello programma.

Preesecuzione veloce

Al termine della prima fase di preesecuzione (immediatamente prima della creazione della LABEL TABLE) il preesecutore considera il programma "parzialmente preeseguito". Questa informazione può essere memorizzata (comandi SAVE, REPLACE) tra le caratteristiche del programma.

A fronte di una nuova esecuzione, la preesecuzione del programma "parzialmente preeseguito" tralascia le operazioni della prima fase, ad eccezione della creazione della LABEL TABLE.

Questo tipo di preesecuzione viene detta "preesecuzione veloce", poichè consente un notevole risparmio di tempo. Un programma viene considerato "parzialmente preeseguito" finchè non intervengono operazioni di editing, nel qual caso si rende nuovamente necessaria una preesecuzione completa.

Registrazione del programma in formato di editing

Con la creazione della LABEL TABLE e le successive operazioni di preesecuzione, il programma passa dal formato di editing (non eseguibile) al formato eseguibile, che manterrà durante tutto il periodo dell'esecuzione. Il formato eseguibile non consente di riottenere il formato di editing; per questo motivo sul disco in linea viene registrato il formato di editing: è così possibile eseguire operazioni di editing al termine dell'esecuzione. La registrazione del programma non avviene nel caso in cui ne sia stata comandata l'esecuzione o la preesecuzione con il comando:

```
    RUN filename
PREPARE filename
```

In questo caso infatti il programma è già registrato in memoria.

Il preesecutore causa la transizione dal "livello comandi" al "livello programma" e il TERMINATOR è il modulo del sistema operativo che ripristina la configurazione di editing del programma provocando la transizione inversa.

Le BRTS possono usare, per l'esecuzione delle operazioni di I/O con periferiche interne, sia il LIOCS sia il PIOCS. Alcuni moduli di BRTS possono inoltre richiamare, durante l'esecuzione, parti di LIOCS che vengono caricate nell'area di OVERLAY, e vi restano per il tempo necessario all'esecuzione dell'istruzione.

Supervisore esecuzione (BASEX)

Il modulo che svolge i compiti di supervisore della fase di esecuzione è il BASEX; viene richiamato in area utente e ne occupa 1380 byte .

Il BASEX è composto dall'insieme delle routines di trattamento delle SVC (Supervisor Call) provenienti dal programma utente, e può accedere al campo LOPTYPE (Last Operation Type), contenuto nell'area di comunicazione COMAREA. Questo campo contiene il tipo di operazione eseguita nell'istruzione immediatamente precedente.

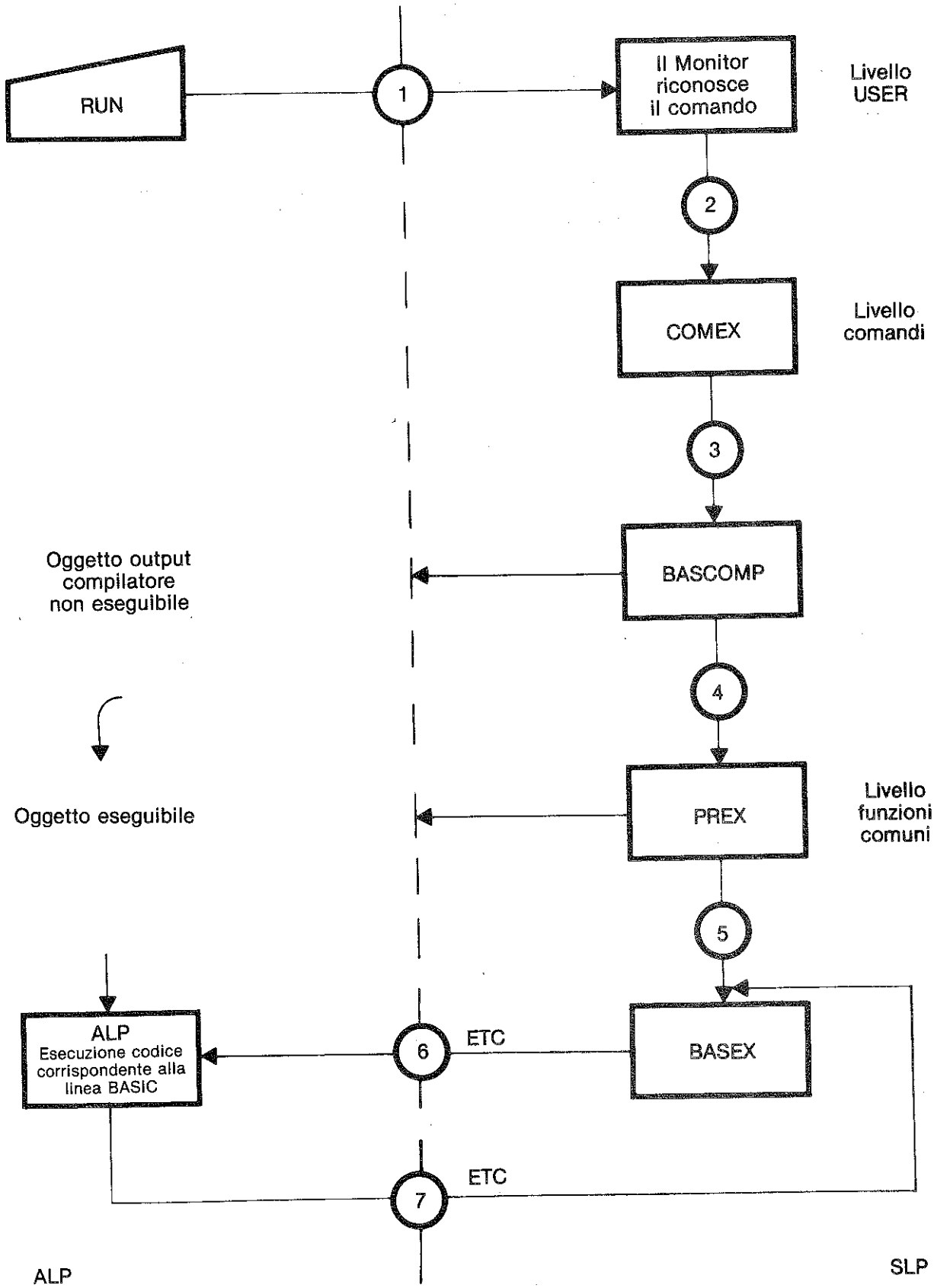
In particolare interessa distinguere:

- esecuzione in sequenza

- GO TO
- GO SUB
- riferimento a funzione
- RETURN
- NEXT

Il modulo processa le seguenti SVC:

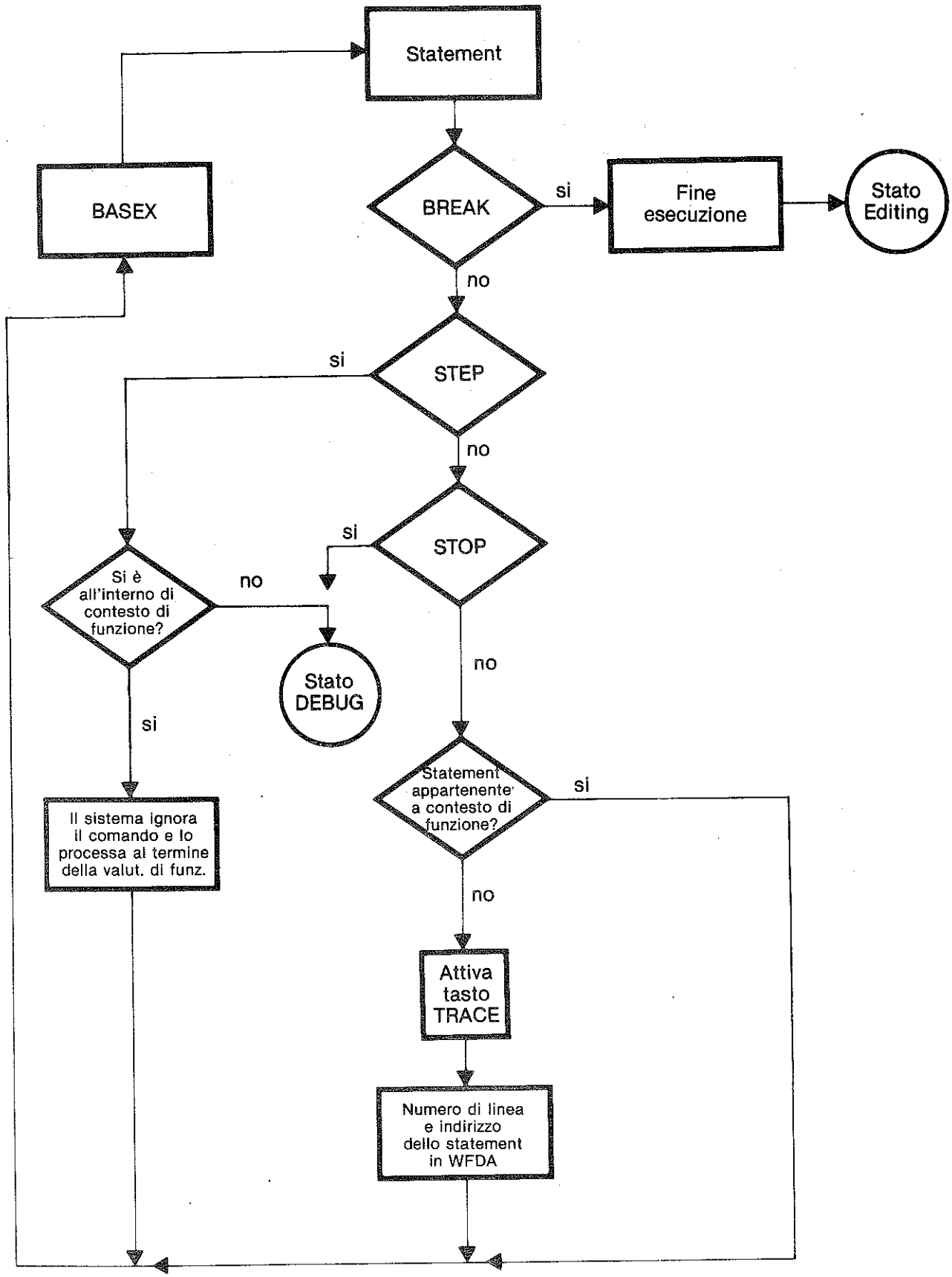
- SVA anteposta al codice oggetto di ogni statement
- istruzione STOP
- istruzione END.


RUN
1
Il Monitor riconosce il comando
Livello USER
2
COMEX
Livello comandi
3
BASCOMP
Oggetto output compilatore non eseguibile
4
Oggetto eseguibile
PREX
Livello funzioni comuni
5
ALP
Esecuzione codice corrispondente alla linea BASIC
6
ETC
BASEX
ETC
7
ALP
SLP

Inizio istruzione

Anzitutto il BSEX ccntrolla che non si siano verificati errori durante l'esecuzione dell'istruzione precedente. In caso affermativo esegue il trattamento di errcre; in caso verifica se l'istruzione successiva è eseguibile oppure no.

Un'istruzione non eseguibile non dà luogo ad alcuna azione da parte del sistema. Se incontrata viene oltrepassata, e viene eseguita l'istruzione successiva. Un'istruzione eseguibile dà luogo invece ad un'opportuna azione da parte del sistema.


Statement
BASEX
BREAK
si
Fine esecuzione
Stato Editing
no
si
STEP
no
si
STOP
Si è all'interno di contesto di funzione?
no
Stato DEBUG
no
si
Statement appartenente a contesto di funzione?
si
Il sistema ignora il comando e lo processa al termine della valut. di funz.
no
Attiva tasto TRACE
Numero di linea e indirizzo dello statement in WFDA

Interruzione esecuzione

Se l'istruzione è eseguibile il BASEX, prima di lasciarne l'esecuzione, controlla che l'utente non abbia inviato da tastiera un comando BREAK o STEP, o che la esecuzione non debba essere sospesa per effetto di un comando STOP.

Processo di BREAK: Se l'utente ha premuto il tasto BREAK, il BASEX esegue la stessa azione attivata dalla istruzione END.

Processo di STEP: Se è attivato il tasto STEP si distinguono due casi: se ci si trova all'interno di una funzione, il BASEX ignora lo STEP, per poi processarlo al termine della valutazione della funzione; altrimenti il modulo invia sul display il numero di linea dell'istruzione da eseguire, e attiva i tasti STEP e CONTINUE ponendo il sistema nello stato di DEBUGGING.

Processo di STOP: Se il numero di linea dell'istruzione da eseguire coincide con quello specificato in un precedente comando STOP, il modulo invia su display il messaggio STOP ed esegue le azioni proprie del comando STEP tranne la visualizzazione.

Lancio dell'istruzione: Se non v'è stata interruzione dell'esecuzione, a seguito di STEP o STOP, il modulo BASEX distingue ancora fra istruzioni che non appartengono a funzioni e istruzioni che ne fanno parte.

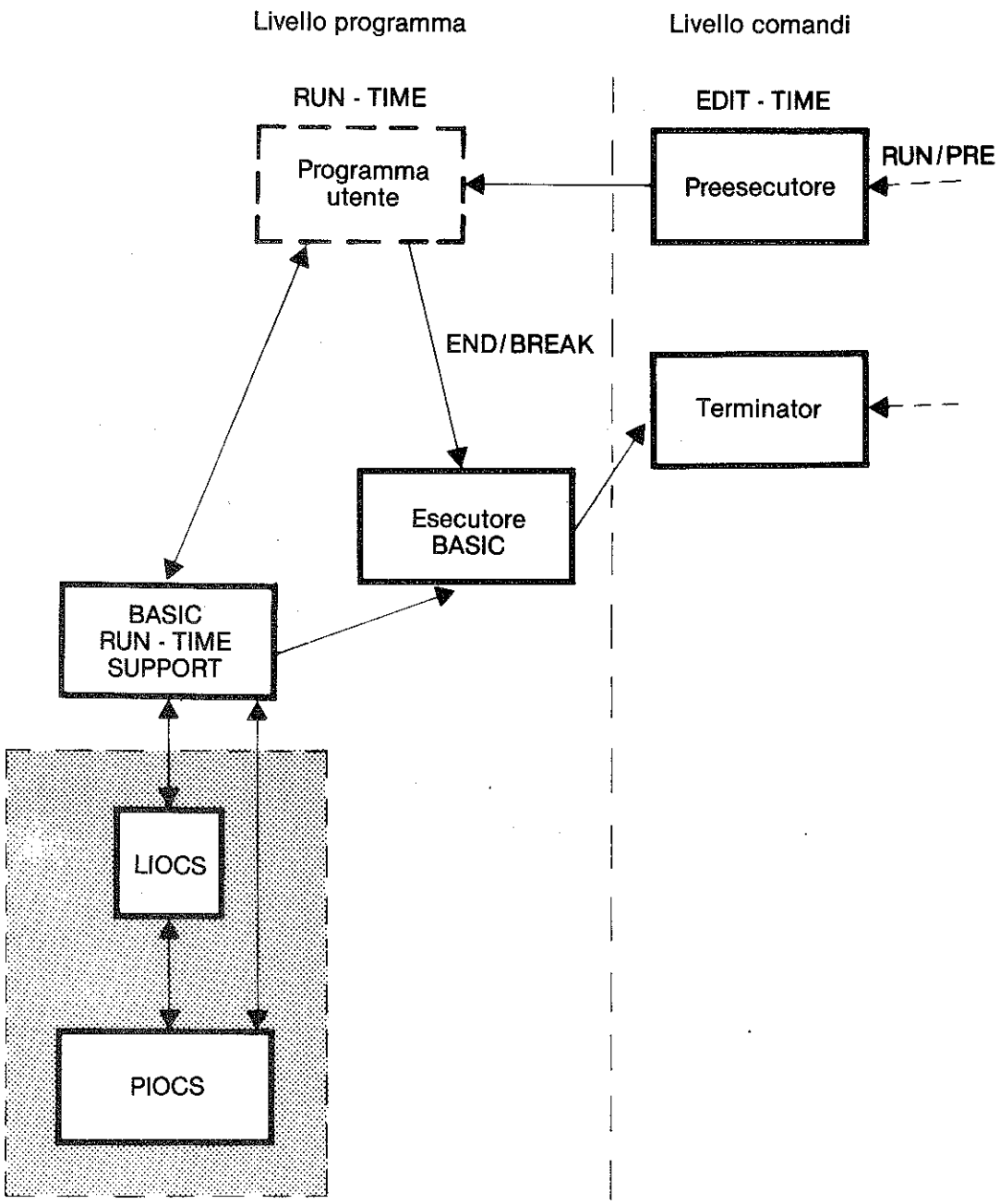
Se l'istruzione non appartiene ad una funzione si eseguono i seguenti passi:

- tracciamento, regolato dal tasto TRACE della console

- il numero di linea dell'istruzione ed il suo indirizzo vengono memorizzati in alcuni campi del WFDA (informazioni sulla linea corrente vedi capitolo 3)

- il controllo è restituito al programma utente

Se l'istruzione fa parte di una funzione multilinea il controllo viene restituito direttamente al programma utente.


Livello programma
Livello comandi
RUN - TIME
EDIT - TIME
Programma utente
Preesecutore
RUN/PRE
END/BREAK
Terminator
Esecutore BASIC
BASIC RUN - TIME SUPPORT
LIOCS
PIOCS

Esecuzione

L'esecuzione di un programma BASIC consiste nella scansione del codice oggetto del programma e nella sua interpretazione da parte del software di base che realizza la macchina virtuale ALP.

Nel corso della scansione, quando necessario, vengono inoltre richiamate le routines di BRTS ed il supervisore dell'esecuzione (BASEX: Basic Executor) per l'espletamento delle funzioni loro demandate.

Organizzazione del sistema operativo con il sistema a livello programma

Quando il programma è in esecuzione il sistema si trova in uno stato appartenente al "livello programma". L'organizzazione del sistema operativo in questo caso è illustrata nella figura precedente.

Trattamento errori

Vengono distinti i seguenti tipi di errore:

- errori recuperabili
- errori non recuperabili

Per tutti viene inviato su display il messaggio opportuno. Gli errori recuperabili sono contraddistinti dal fatto che per essi il sistema assume un'azione standard di recupero dell'errore, giungendo al termine dell'esecuzione dell'istruzione. A fronte di questo tipo di errori vengono abilitati i tasti STEP e CONTINUE e il sistema è posto in stato di debugging.

Gli errori non recuperabili determinano invece la fine dell'esecuzione con un'azione analoga a quella della istruzione END, BREAK.

Istruzione STOP: Viene visualizzato il messaggio di STOP. Il sistema è posto nello stato di debugging e vengono abilitati i tasti STEP e CONTINUE.

Istruzione END: Il controllo viene ceduto al TERMINATOR, che finalizza eventuali operazioni di I/O (chiusura file, ecc.) e trascrive nuovamente il programma in memoria in configurazione di editing.

L'esecuzione delle istruzioni algebriche

Ogni istruzione BASIC viene tradotta in una serie di istruzioni in linguaggio algebrico. La macchina virtuale ALP utilizza lo stack per l'esecuzione di queste istruzioni. Lo stack viene gestito dal processor ALP

per mezzo di puntatori che permettono di controllare tutte le operazioni di allocazione e rimozione di aree (vedi cap. 3).

I puntatori sono:

1. BOTTOM : punta alla base dello stack

2. TOP : punta al limite superiore dello stack

3. TOS (Top of Stack) : punta al 1° byte libero dello stack

4. LENV (Local Environment): punta alla base dell'ultimo contesto locale allocato, dove per contesto locale si intendono le informazioni memorizzate sullo stack all'atto dell'esecuzione di una definizione di funzione BASIC

I primi due puntatori vengono controllati ogni volta che si opera una allocazione o deallocazione sullo stack. Il terzo puntatore serve per la gestione corrente degli elementi sullo stack. Il quarto puntatore serve come indirizzo di base per ogni riferimento a variabili locali della procedura in atto. Le operazioni di I/O (vedi anche cap. 8) vengono eseguite richiamando le apposite routines di BRTS. In questa circostanza per mezzo di apposite istruzioni il controllo viene ceduto al processor SLP, e da questo restituito al processor ALP.

Le istruzioni del linguaggio algebrico possono essere così suddivise:

- istruzioni per il caricamento di variabili nello stack

- caricamento costanti nello stack

- istruzioni con operando implicito nello stack

- istruzioni di salto

- istruzioni di subroutine e function

- altre istruzioni

L'allocazione di aree sullo stack e la loro rimozione avvengono a run time a seguito dell'esecuzione di istruzioni BASIC (ad esempio, l'istruzione GOSUB provoca l'allocazione di aree sullo stack, mentre l'istruzione RETURN ne provoca la rimozione).

Caricamento variabili nello stack

Queste istruzioni operano sullo stack, permettendo di forzare l'indirizzo di una variabile specificata. Il set di istruzioni interessa le variabili numeriche e alfanumeriche sia globali che locali. L'esecuzione di ogni istruzione aggiorna il puntatore di gestione dello stack (TOS) e controlla un eventuale overflow di stack. L'indirizzo delle variabili numeriche sia locali che globali viene caricato in maniera indiretta, allocando sullo stack l'indirizzo implicato (1 byte) stesso dell'istruzione, dato che identifica univocamente l'indirizzo della variabile. Per la variabile stringa sia globali che locali, viene caricato sullo stack l'indirizzo implicito del descrittore. L'indirizzo è dato come valore relativo all'inizio della tabella SVST, oppure all'inizio dello spazio allocato sullo stack (per i descrittori delle variabili locali). I descrittori contengono poi l'indirizzo del valore della variabile. Per l'indirizzo di array viene caricato sullo stack l'indirizzo implicito del descrittore.

Caricamento costanti nello stack

Queste istruzioni operano sullo stack in modo simile alle precedenti: viene caricato sullo stack un valore costante (numerico o alfanumerico); viene aggiornato il puntatore di gestione di stack; viene controllato l'overflow di stack.

Istruzioni con operando implicito nello stack

Queste istruzioni sono prive di campo operando, poichè operano implicitamente sugli elementi già allocati sullo stack. Per ogni istruzione sono previsti il tipo operandi che si trovano sullo stack e l'opportuno aggiornamento del puntatore di gestione dello stack medesimo.

Fanno parte di questo set:

- OPERATORI NUMERICI
- OPERATORI DI STRINGA
- OPERATORI DI INDICIAMENTO
- OPERATORI DI ARRAY (VARIABILE MULTIPLA)
- OPERATORI DI CONFRONTO

Operatori numerici

Gli operatori numerici possono operare sugli ultimi due elementi dello stack (assegnazione, somma, sottrazione, ecc.), sull'ultimo (ON GOTO), oppure sugli ultimi N (assegnazione multipla).

Gli operandi sono variabili numeriche o temporanei numerici (i temporanei numerici sono valori numerici sullo stack). Quando l'azione dà un risultato numerico, questo è un temporaneo che sostituisce gli elementi su cui opera. Le operazioni di assegnazione scaricano lo stack senza depositarvi temporanei.

Operatori di stringa

Gli operatori di stringa operano sugli ultimi due operandi (concatenazione e assegnazione) oppure sugli ultimi N operandi (assegnazione multipla) dello stack. Gli operandi sono variabili o temporanei alfanumerici (i temporanei alfanumerici sono valori alfanumerici sullo stack). Il risultato dell'operazione viene allocato sullo stack al posto degli operandi. Le operazioni di assegnazione scaricano lo stack senza depositarvi temporanei.

Operatori di indiciamento

Gli operatori di indiciamento permettono di caricare sullo stack l'indirizzo di un elemento di una variabile multipla. Sullo stack si trovano l'indice (nel caso di un vettore) o gli indici (nel caso di una matrice) che identificano l'elemento stesso.

Operatori di array

Gli operatori di array operano sugli ultimi due operandi dello stack (vettori) oppure sugli ultimi tre (matrici). Gli operandi sono indirizzi impliciti di variabili multiple. Il risultato dell'operazione è contenuto nella variabile multipla il cui indirizzo era l'operando a livello più basso.

Operatori di confronto

Gli operatori di confronto operano sugli ultimi due elementi dello stack (ambedue numerici o alfanumerici). A seconda che la condizione che esaminano sia verificata oppure no, al posto degli operatori viene caricato sullo stack il temporaneo numerico 1 o Ø.

Istruzioni di salto

Fanno parte di questo gruppo le istruzioni di SALTO INCONDIZIONATO (GOTO), SALTO CONDIZIONATO (IF/THEN) e SALTO CALCOLATO (ON/GOTO). L'istruzione GOTO non fa uso dello stack; le altre operano invece sull'ultimo elemento allocato sullo stack, che contiene le seguenti informazioni:

1. L'indirizzo dell'istruzione riferita e uno switch che indica se il salto va effettuato oppure no (nel caso di IF/THEN).

2. I numeri di linea delle istruzioni riferite e un contatore che indica a quale di essi si deve saltare (nel caso di ON/GOTO).

Se il salto avviene dall'interno all'esterno di un ciclo FOR/NEXT vengono rimossi dallo stack i contesti di loop (vedi § istruzioni varie) da cui si esce. All'uscita dell'ON GO TO le informazioni sulla linea successiva sono contenute nel Program Counter.

Esempio:

```
100   FOR I=1 TO N STEP 5
.....
150   FOR J=1 TO M STEP 1
.....
180   GO TO 460
.....
200   NEXT J
.....
.....
.....
.....
400   NEXT I
.....
460   PRINT J
470   END
```

All'esecuzione dell'istruzione avente numero di linea 180 vengono rimossi due contesti di loop. L'indirizzo di memoria dell'istruzione cui si salta viene ricavato

dal successivo esame della LABEL TABLE.

Subroutine e function

Di questo gruppo fanno parte due tipi di istruzioni:

1. Quelle che passano il controllo a subroutine e funzioni.

2. Quelle che restituiscono il controllo al programma chiamante.

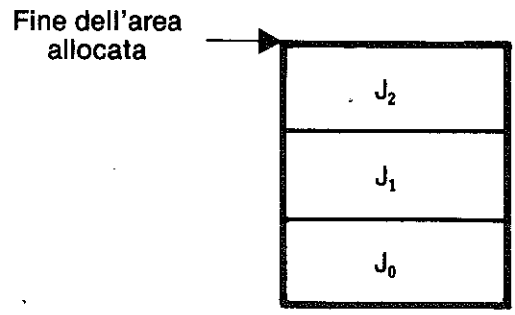
Si tratta ancora di istruzioni di salto. Quelle del 1° tipo, con l'esecuzione del salto, devono salvare sullo stack l'indirizzo dell'istruzione successiva (contenuto dal Program Counter), il livello di annidamento, il puntatore all'ultimo contesto locale allocato per una funzione (il puntatore deve essere disponibile per la nuova allocazione) e il N°di argomenti trasmessi (Ø nel caso di SUBROUTINE). Quelle del 2° tipo recuperano tali informazioni di ritorno ripristinando la situazione di stack precedente la chiamata. Nel § salto a subroutine vengono esposte in modo dettagliato le procedure di richiamo.

Istruzioni varie

Fanno parte di questo insieme: istruzioni che determinano la fine di un programma (STOP/END)); istruzioni che permettono di cedere il controllo al processor SLP; altre istruzioni, tra le quali la FOR e la NEXT che consentono di instaurare un ciclo di esecuzione.

Esecuzione di un ciclo

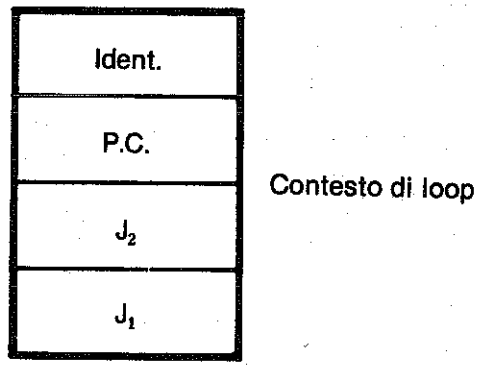
L'istruzione FOR opera sullo stack utilizzando gli ultimi tre elementi sullo stack e memorizzandovi il Program Counter. All'esecuzione del FOR (es.: FOR I=J_0 TO J_1 STEP J_2) lo stack si trova come indicato appresso:



1. Si scandiscono J_o , J_1 , J_2 e in luogo dei loro indirizzi si caricano i loro valori temporanei.

2. Si testa se il valore iniziale della variabile di controllo del ciclo è maggiore di quella finale (e il passo è positivo) oppure minore (e il passo è negativo) nel qual caso si passa ad eseguire l'istruzione successiva alla NEXT. Altrimenti si incrementa il valore del livello di annidamento per le istruzioni, si rimuove il valore J_o (già assegnato ad I) e si memorizza sullo stack il Program Counter (PC). Il Program Counter contiene l'indirizzo dell'istruzione successiva alla FOR, informazione che viene utilizzata per poter rieseguire tale istruzione dopo la NEXT. Inoltre alloca sullo stack anche un identificatore del ciclo FOR/NEXT. Queste informazioni costituiscono il "contesto di loop".

   All'esecuzione della NEXT, lo stack è così configurato:



L'istruzione opera con i seguenti passi:

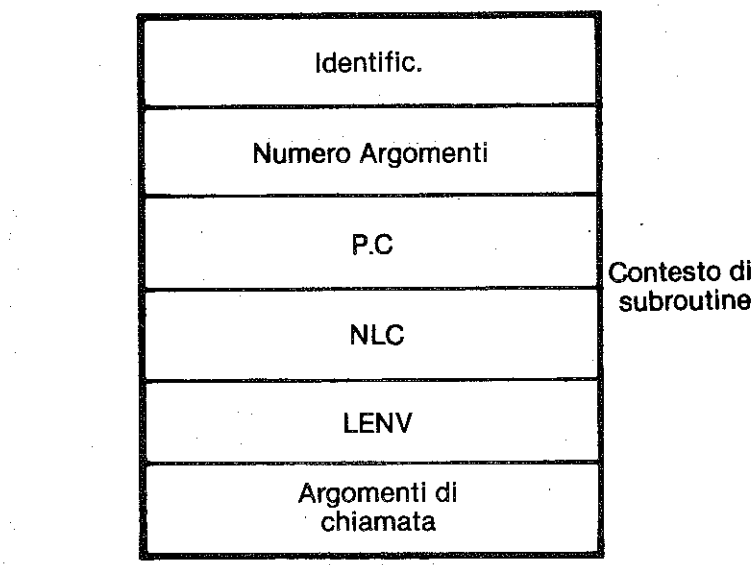
1. Incrementa il valore della variabile di controllo.

2. Testa se il valore della variabile di controllo è maggiore di J_1 . Se è maggiore decrementa il livello di annidamento delle istruzioni. Si rimuove il contesto di loop posizionando il TOS sotto l'elemento che contiene J_1 e si passa il controllo all'istruzione successiva alla NEXT. Se non è maggiore si carica sul program counter l'indirizzo della prima istruzione successiva alla FOR e si cede il controllo a tale istruzione.

Salto a subroutine

Questa istruzione permette di alterare il flusso normale dell'esecuzione del programma, andando ad eseguire una parte di programma a partire da un certo numero di linea in poi.

Al termine si vuole rientrare all'istruzione successiva a quella di salto; perciò, prima di effettuare il salto, occorre caricare nello stack il contenuto del program counter (indirizzo dell'istruzione successiva), onde poterlo recuperare al termine dell'esecuzione della subroutine. Si caricano poi sullo stack il LENV, il NLC ed il numero di argomenti (Ø nel caso di GOSUB). Queste informazioni, insieme al program counter e ad un identificatore, costituiscono il "contesto di subroutine". Si esegue poi il salto normalmente. Si compila infine con l'opportuno valore il campo LOPTYPE.



Rientro da subroutine

Al rientro da un salto a subroutine, si dealloca l'identificatore del contesto di subroutine; si carica sul program counter l'indirizzo di rientro del contesto di subroutine; si restituisce il valore salvato di NLC e di LENV e si assegna l'appropriato valore al campo LOPTYPE deallocando poi l'intero contesto di subroutine.

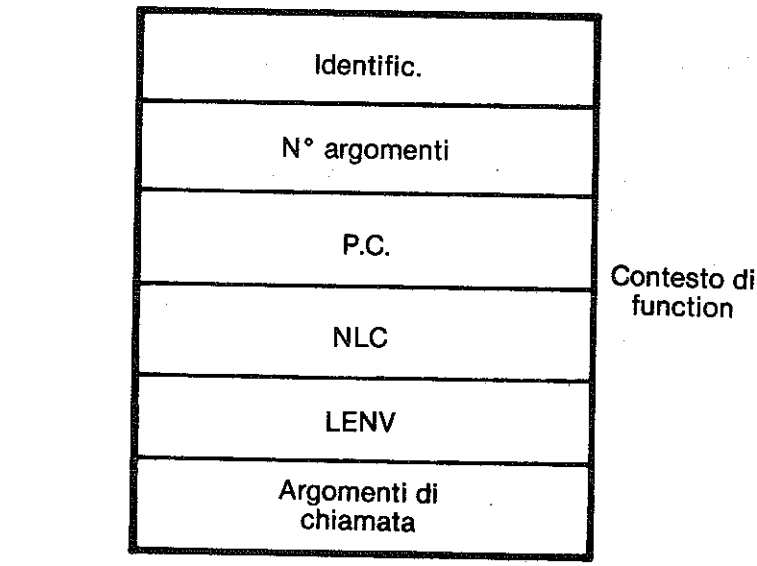
Chiamata di funzione di sistema

Questa istruzione permette di saltare all'esecuzione di un modulo di BRTS per l'espletamento di funzioni complesse per cui non esistono istruzioni dirette di linguaggio algebrico (istruzioni di I/O).

Nello stack di carica il "contesto di function", cioè:

- il numero degli argomenti, il contenuto del program counter, il livello dell'NCL, il puntatore all'ultima area locale (LENV) e un identificatore

Si preleva quindi l'indirizzo della funzione di sistema, con una ricerca sulla tabella BRTSCT. Si carica nel program counter, si posiziona il LENV sul 1° argomento della funzione e si passa il controllo al modulo di BRTS.



Rientro da funzione di sistema

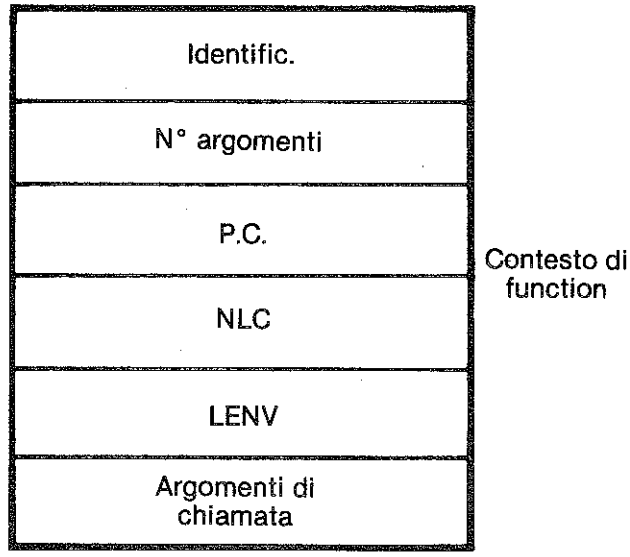
Al rientro da una funzione, si dealloca il contesto di function usato per l'esecuzione della funzione e, in base alle informazioni di rientro allocate sullo stack, si attribuiscono a LENV, NLC e program counter i valori precedenti. Il program counter contiene così l'indirizzo di rientro per l'esecuzione dell'istruzione successiva a quella contenente la chiamata di funzione.

Chiamata di funzione

Questa istruzione permette di saltare all'esecuzione di una funzione BASIC. L'istruzione contiene il numero dei parametri che vengono trasmessi alla funzione e il suo indirizzo implicito. I parametri devono essere depositati sullo stack, per poi essere utilizzati dalla istruzione di definizione di funzione. La chiamata di funzione opera un salvataggio delle informazioni di ritorno sullo stack; memorizza cioè:

- il numero dei parametri trasmessi
- l'indirizzo dell'istruzione successiva (contenuto nel program counter)
- il livello di loop (contenuto di NLC)
- il puntatore di base della precedente area locale (LENV)

Queste informazioni costituiscono, insieme ad un identificatore, il "contesto di function". Infine aggiorna il LENV sul primo argomento caricato sullo stack. Dopo questa fase si procede alla valutazione della funzione, cioè all'esecuzione delle istruzioni che la definiscono.



Definizione di funzione

Questa istruzione definisce una funzione BASIC, fornendo le informazioni necessarie per la sua successiva valutazione. La definizione di funzione alloca sullo stack un contesto locale di cui fanno parte i seguenti

elementi:

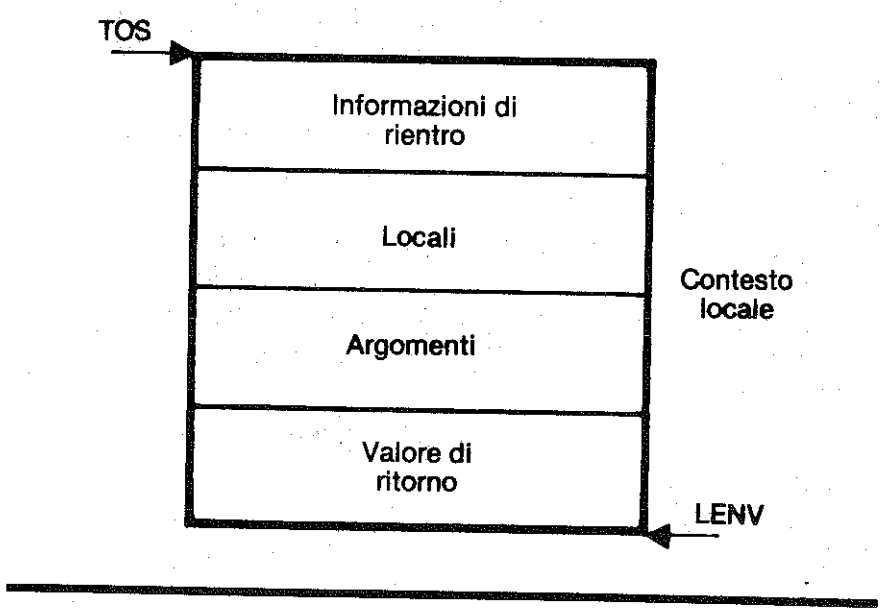
- valore di ritorno della funzione

- argomenti (formati solo da valori)

- locali (descrittori di variabili locali e valori delle stringhe locali)

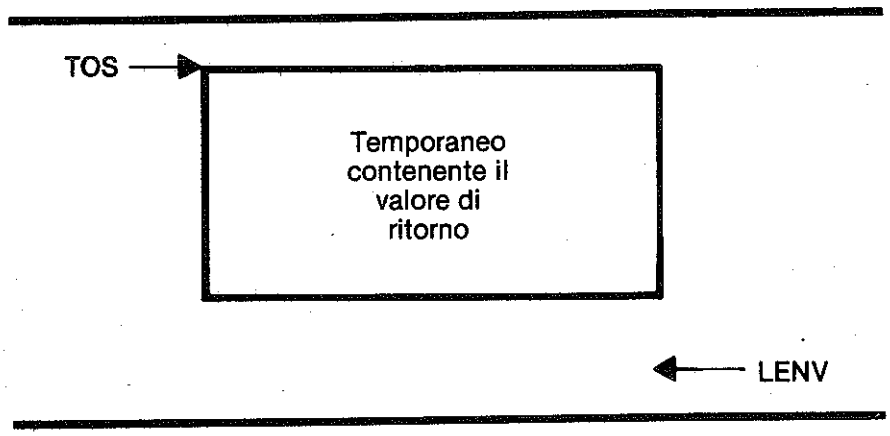
- informazioni di rientro della funzione (praticamente le informazioni che fanno parte di un contesto di function come descritto nella figura precedente)

Il TOS punta alla testa delle informazioni di rientro, mentre il LEV punta alla base del valore di ritorno della funzione.



Rientro da funzione

Al termine della valutazione della funzione avviene il ripristino dei precedenti valori di LENV, TOS, Program Counter e la deallocazione del contesto locale. Alla fine il TOS punta al valore di ritorno della funzione (temporaneo numerico o alfanumerico), mentre il LENV punta al contesto precedente.



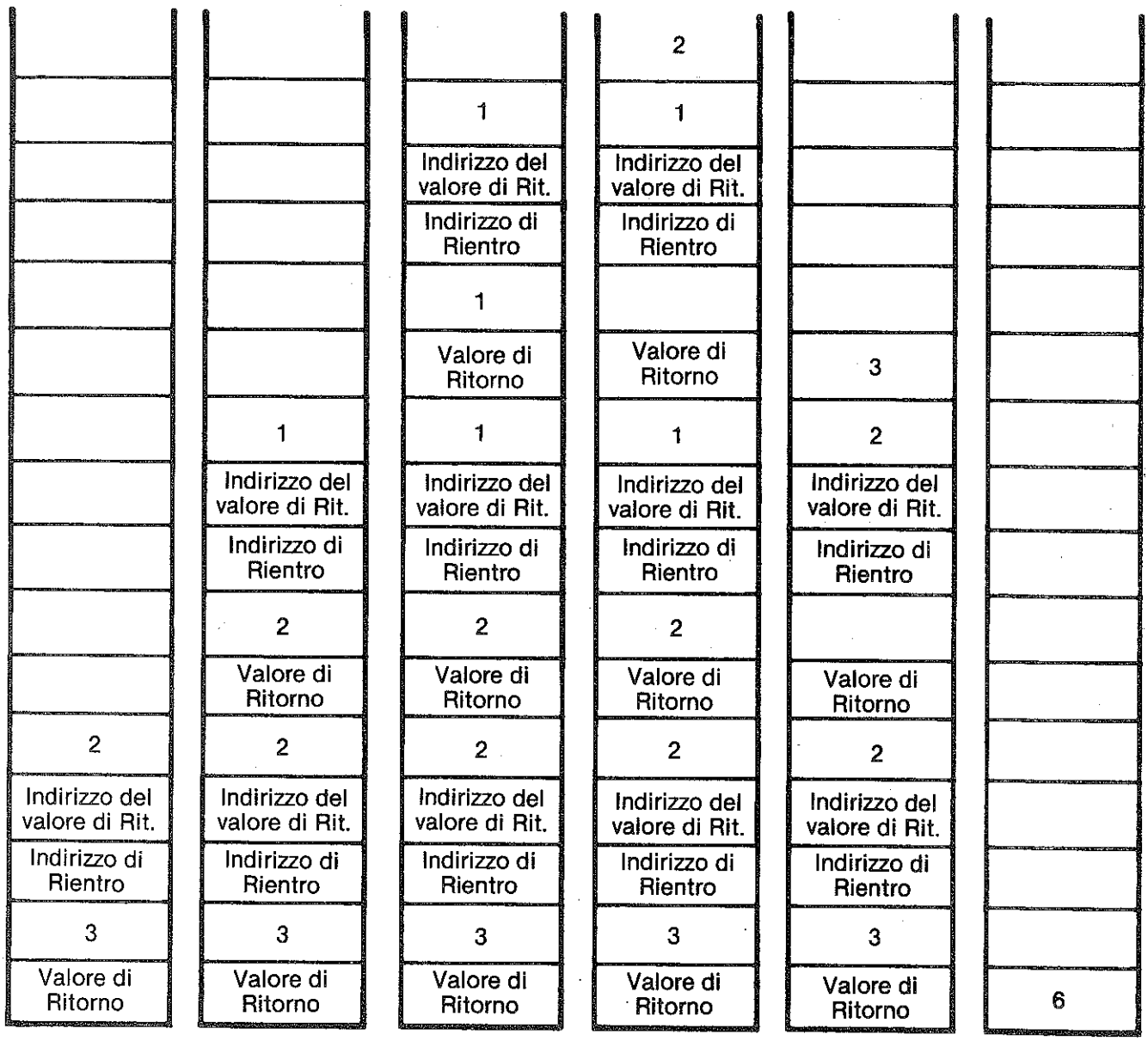
Calcolo fattoriale in una funzione ricorsiva

Il calcolo del fattoriale in una funzione ricorsiva avviene, come per tutte le funzioni, sullo stack. Sullo stack vengono allocati i contesti locali; il numero dei contesti allocati equivale al fattoriale da calcolare (così se si vuole calcolare il fattoriale di 5, vengono allocati 5 contesti locali).

Se si vuole calcolare, ad esempio, con la seguente funzione:

```
10  DEF  FNA (X)
20  IF X=1 THEN 50
30  FN*=FNA(X-1)*X
40  GO TO 60
50  FN*=1
60  FNEND
```

sullo stack il calcolo viene eseguito come illustrato nello schema A (per X=3).


2
Indirizzo del valore di Rit.
Indirizzo di Rientro
Valore di Ritorno
1
2
3
6

7. IL FILE SYSTEM

Generalità

File System

Definiamo File System l'organizzazione dei dati e di ogni altra informazione in file e librerie, nonchè l'insieme dei moduli del Software di base preposti alla loro gestione.

Record logico

Definiamo record logico l'unità elementare d'informazione che l'utente può memorizzare e alla quale può accedere. Sul sistema P6060 la struttura del record logico è arbitraria, e la sua lunghezza indefinita.

Record fisico

Definiamo record fisico o blocco una sequenza di dati che può essere trasferita dalla memoria interna a quella esterna e viceversa per mezzo di un'unica operazione di I/O. La lunghezza massima di un record fisico è di 128 byte su sistemi FDU e 256 su sistemi DCU/HDU. Un record fisico può contenere uno o più record logici, ovvero può essere un sottoinsieme proprio di un record logico. In questo caso ha la lunghezza massima possibile.

Una collezione di record logici costituisce un file. Il sistema P6060 è in grado di riconoscere tre differenti tipi di file : file programma, file testo e file dati. A loro volta i file di tipo dati possono essere distinti in tre categorie, a seconda delle modalità di accesso che consentono:

- file dati ad accesso sequenziale (S)
- file dati ad accesso diretto (S)
- file dati di tipo plotter (Z)

In quest'ultimo caso la chiave di accesso al record desiderato è il numero ordinale della parola (per parola si intende un gruppo di 4 byte successivi) ove è allocato il primo carattere del record stesso; tale

numero può essere calcolato facilmente ricordando che:

- ad una costante numerica in singola precisione viene allocata una parola

- ad una costante numerica in doppia precisione vengono allocate due parole

- ad una costante alfanumerica di N caratteri vengono allocate M=INT ((N-1)/4+2) parole.

Librerie

Una libreria è un insieme catalogato di file. Il sistema può gestire una libreria utente per ogni unità FD ed un massimo di 35 librerie utente per ogni unità DC o HD.

All'interno di ogni singola libreria, il sistema può gestire al più una sottolibreria di ciascuno dei 3 tipi seguenti:

- tipo package
- tipo comune
- tipo utente

Il tipo di sottolibreria è individuato dal prefisso associato ai nomi dei file ad essa appartenenti (rispettivamente *,+,Ø).

I tre tipi di sottolibrerie si differenziano per il diverso livello di protezione a cui sono soggetti:

- nella sottolibreria package è vietato inserire, modificare o cancellare entry in directory

- nella sottolibreria comune è vietato modificare o cancellare entry in directory

La protezione diventa operativa dopo il lancio di un apposito programma di utilità (LBPROTECT) e non può più essere rimossa. Per la sottolibreria utente non è precisato alcun regime di protezione.

Alle librerie accessibili dall'utente va aggiunta la libreria sistema, che contiene i moduli del software di base: in tale libreria non è consentita all'utente alcuna operazione. Ogni sottolibreria può contenere sia file programma, sia file testo, sia file dati.

In una sottolibreria non possono coesistere più file con lo stesso nome; file con lo stesso nome possono però essere presenti in sottolibrerie o libreria differenti.

Su tutte le libreria è previsto un secondo livello di protezione, operante a livello file dopo che il file stesso è stato caricato in memoria: possono essere inibite le operazioni di decompilazione, linking listing, visualizzazione ed editing sui file programma e testo, e può essere inibito l'uso dell'utility FLPRINT per i file dati. La protezione è applicabile ai file delle librerie di tipo utente per mezzo del comando SECURE; è automatica per le librerie di tipo package e di tipo comune.

L'organizzazione delle librerie nel sistema P6060 è stata concepita in modo tale da garantire una completa compatibilità tra il sistema FDU e il sistema DCU/HDU, sia a livello file. Una libreria è individuata da un nome, specificato nell'etichetta di libreria e dalla periferica su cui è allocata.

Al momento della creazione di una libreria è possibile indicare una parola chiave che consente in seguito di disciplinare l'accesso. Inoltre in fase di installazione del sistema viene specificata una "parola chiave di sistema" che consente l'accesso a tutte le librerie.

La gestione delle librerie (allocazione, rilascio, compattamento, apertura, chiusura) è consentita da un ampio set di comandi e di utilities (nel sistema FDU è disponibile un sottoinsieme del set stabilito per il sistema DCU/HDU). E' possibile dichiarare aperte, e quindi operare contemporaneamente su di esse, al più 6 librerie fra quelle esistenti sui dischi in linea. Nel sistema FDU esiste una sola libreria per ciascun floppy-disk; le due librerie (la libreria, nel caso di sistema monodisco) risultano sempre aperte. Se il sistema è stato generato con parola chiave, e la libreria creata anch'essa con parola chiave, e si vuole effettuare la sostituzione di un disco, la libreria risulterà non aperta. Nel sistema DCU/HDU una o più librerie possono essere dichiarate, mediante il comando LBSTORE, librerie di default: verranno cioè aperte automaticamente all'accensione della macchina.

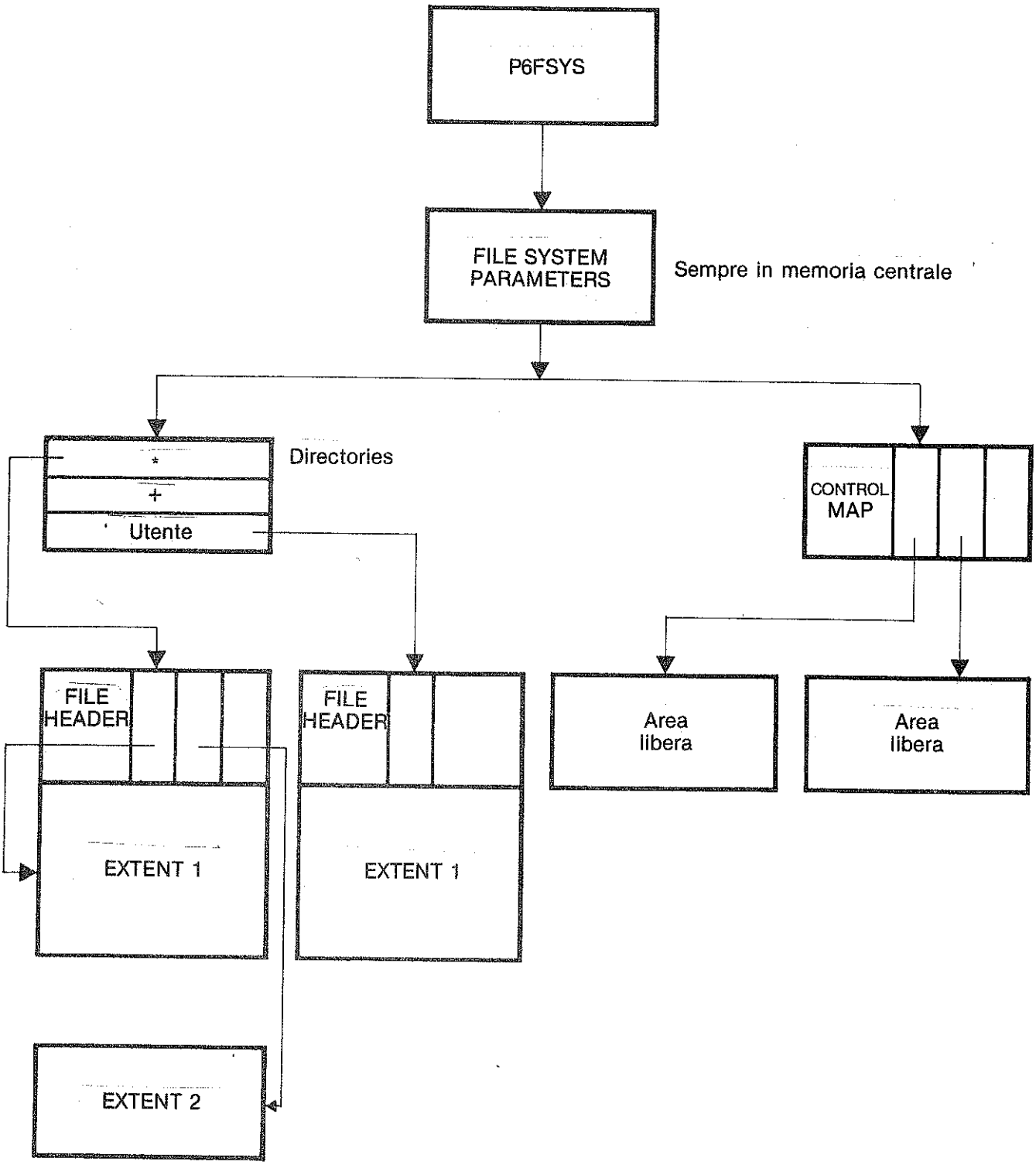
Nel sistema FDU la libreria su disco sistema e la libreria su disco utente sono considerate concatenate per l'istruzione di ricerca di file: per i file programma e i file testo la priorità viene data alla libreria su disco sistema; per i file dati la priorità è inversa.

Nel sistema DCU/HDU le librerie aperte sono sempre considerate concatenate. L'ordine di priorità nelle operazioni di ricerca è lo stesso ordine in cui sono state dichiarate. In questo caso la stessa priorità vale anche per i file di dati.

Le informazioni relative alle:

- tracce alternative
- all'indirizzo e all'ampiezza delle directories
- all'indirizzo e all'ampiezza della control map

Sono presenti in memoria centrale. La ricerca di un file o di un'area libera avviene partendo da tali informazioni.


P6FSYS
FILE SYSTEM PARAMETERS
Sempre in memoria centrale
*
+
Utente
Directories
CONTROL MAP
FILE HEADER
FILE HEADER
Area libera
Area libera
EXTENT 1
EXTENT 1
EXTENT 2

I supporti

La memoria di massa del sistema P6060 è costituita da unità floppy disk (FDU), unità a dischi mobili (DCU), unità dischi fissi (HDU).

Al sistema P6060 possono essere collegate un'unità floppy disk con uno o due drivers, e/o un'unità DCU (con un driver mobile e uno fisso) oppure un'unità HDU (di capacità variabile).

Il floppy disk dispone di una sola superficie utilizzabile per la memorizzazione delle informazioni; la memorizzazione viene effettuata su 77 tracce concentriche, numerate a partire dall'esterno, delle quali:

- le tracce 0 ÷ 73 sono utilizzabili
- la traccia 74 è riservata al sistema
- le tracce 75 ÷ 76 sono riservate al recupero di tracce deteriorate

La traccia indice (traccia ØØ) contiene le etichette di volume e di libreria, le informazioni di controllo per gestire le tracce alternative e le etichette del software di base residente e opzionale.

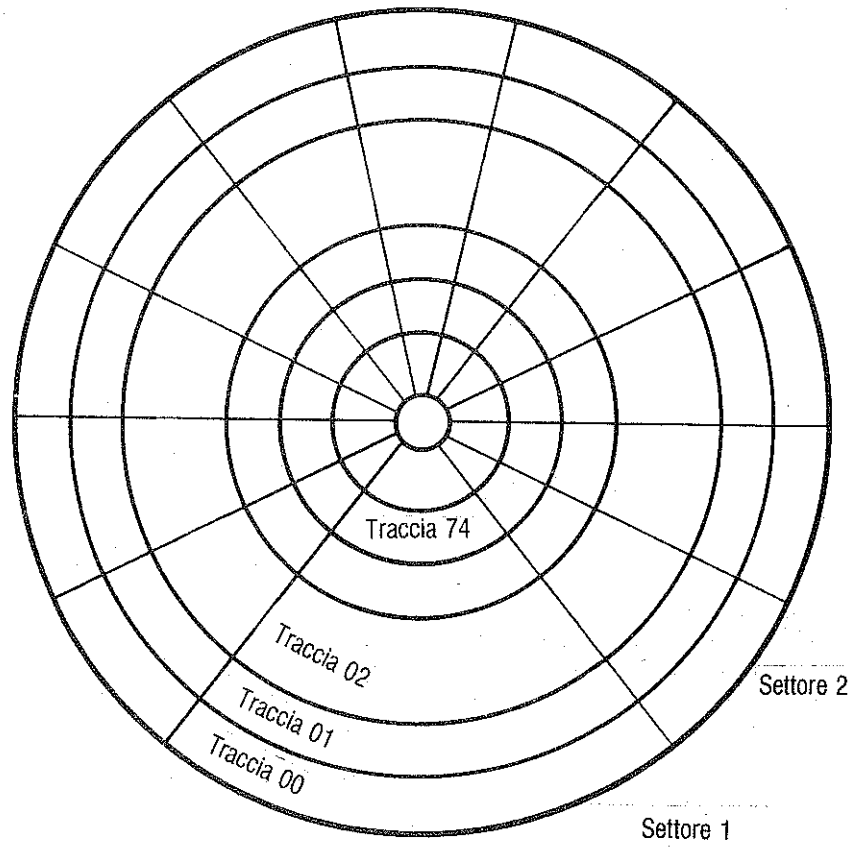


Figura 7-1 Schema del floppy disk

Ogni traccia è a sua volta suddivisa in 26 settori; il formato è descritto nella figura seguente:

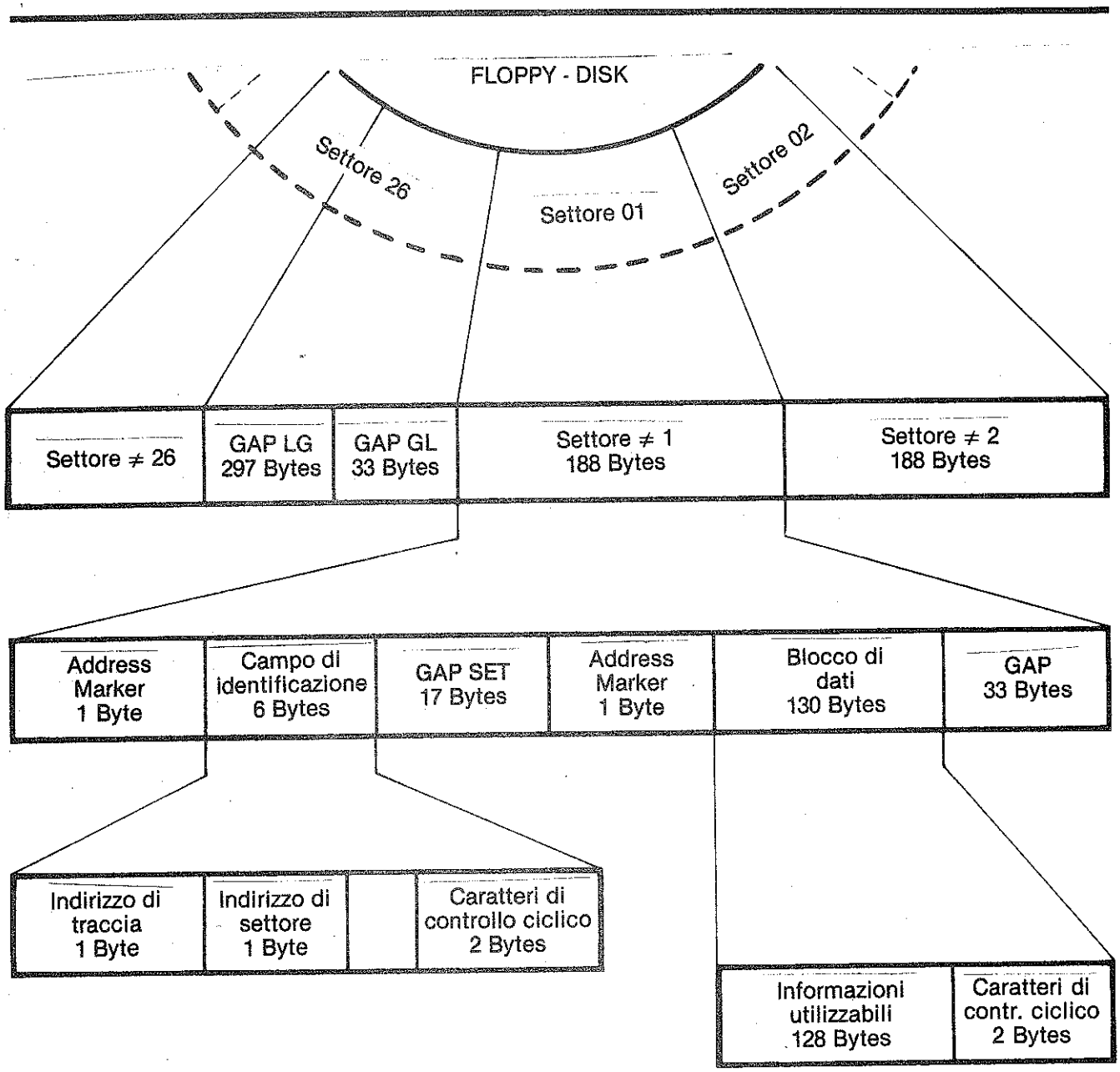


Figura 7-2 Formato del sistema

Disco sistema

A seconda del suo contenuto, un floppy disk prende il nome di disco sistema oppure di disco utente. Un disco sistema contiene il sistema operativo ed il resto del software di base. Funzionalmente si presenta come un disco contenente quattro file etichettati conformemente agli standard Olivetti, contenenti rispettiva-

mente:

- il software residente
- il software opzionale
- il software non residente
- il file system (d,e,f,h), che a sua volta contiene:

  . la sottolibreria Package
  . la sottolibreria Comune
  . la sottolibreria Utente
  . alcune informazioni di servizio (directory, control map)

La struttura del disco sistema è mostrata nella figura seguente:

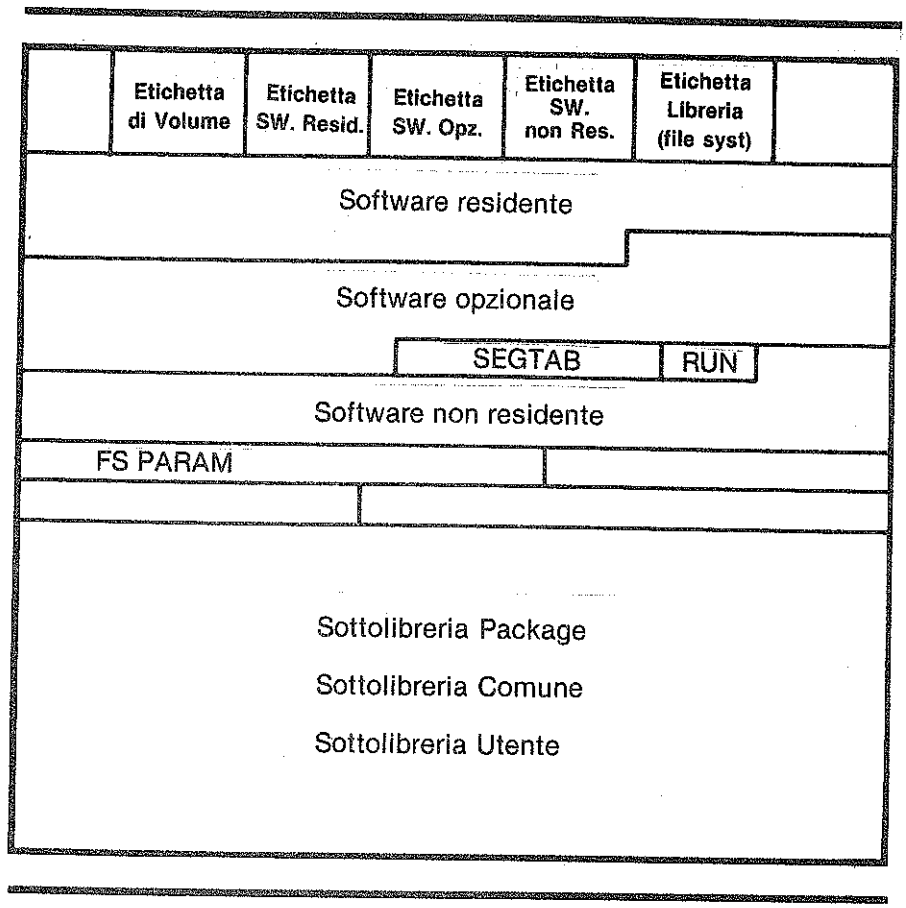


Figura 7-3 Struttura del disco sistema

Disco utente

Un disco utente contiene in traccia Ø una sola etichetta valida: quella del file system. La sua struttura è mostrata in figura.

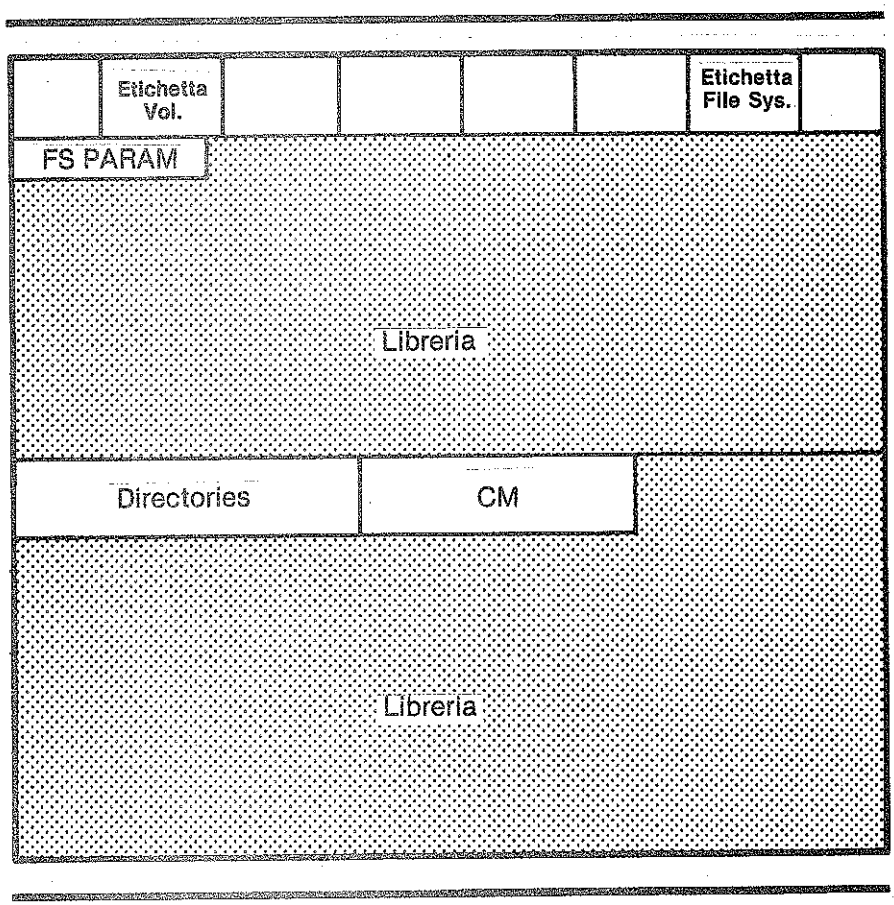


Figura 7-4 Struttura della traccia Ø

Il significato dei campi è il seguente:

Etichetta di volume: contiene le seguenti informazioni:

- nome del volume
- nome del proprietario del volume
- protezione cui è soggetto il volume

Etichette di libreria ed etichette del software di base: Contengono le seguenti informazioni:

- nome della libreria
- data di creazione della libreria
- tipo di libreria
- indirizzo del primo settore della libreria
- indirizzo dell'ultimo settore della libreria
- protezione cui è soggetta la libreria

SEGTAB (lunghezza: 4 settori): E' la tabella dei segmenti della libreria di sistema (vedi cap. 3). Contiene 256 entry ciascuna delle quali indica, in due byte , l'indirizzo di inizio di un segmento.

FUN (lunghezza: 2 settori): Memorizza il contenuto standard dei tasti funzione.

FS PARAM (lunghezza: settori): Contiene i parametri del file system e cioè:

- tipo di protezione
- indirizzo del settore delle directory
- numero di settori della directory della sottolibreria Package
- numero di settori delle directory della sottolibreria Comune
- numero di settore delle directory della sottolibreria utente
- indirizzo del settore della control-map
- numero di settori della control-map

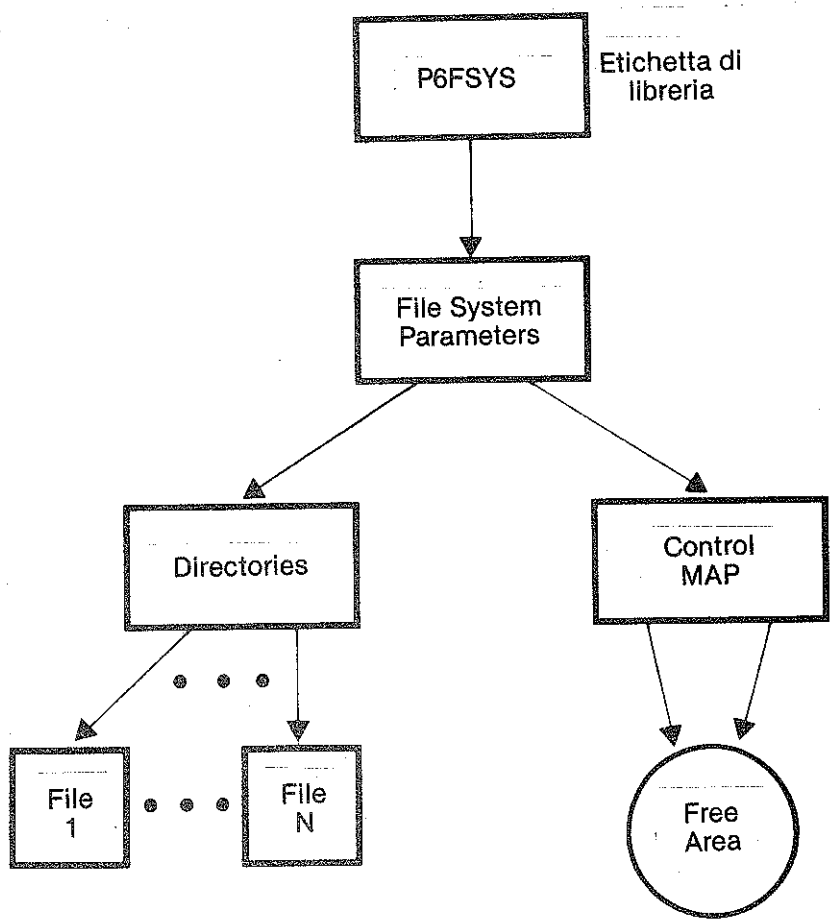


Figura 7-5 Disco utente

Si noti che nel disco sistema la directory è collocata in posizione intermedia tra la libreria sistema e la libreria utente; in tal modo si riducono sensibilmente i tempi di accesso. Per lo stesso motivo (ottimizzazione degli accessi) la posizione della directory è immutata nel disco utente. Il contenuto dei campi directory e control map verrà descritto nel seguito (pag. 7-19).

DCU

Il sistema P6060 DCU comprende 1 o 2 doppi drivers per dischi, comprendenti ciascuno un disco fisso e uno mobile. Ciascun piatto ha entrambe le superfici registrabili. Ogni superficie contiene 408 tracce registrabili, disposte su anelli concentrici. Più tracce che hanno lo stesso raggio ma appartengono a superfici diverse costituiscono un cilindro. Il concetto di cilindro è limitato ad un solo disco, anche se il meccanismo d'accesso è unico per entrambi i dischi.

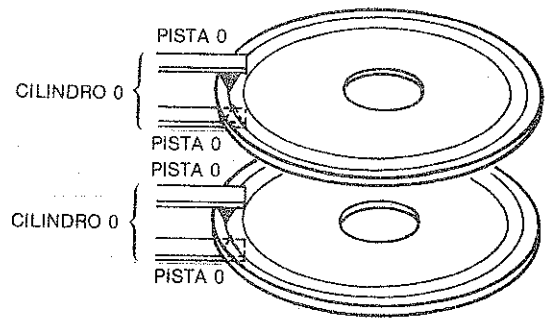


Figura 7-6 Concetto di cilindro

Le unità DCU

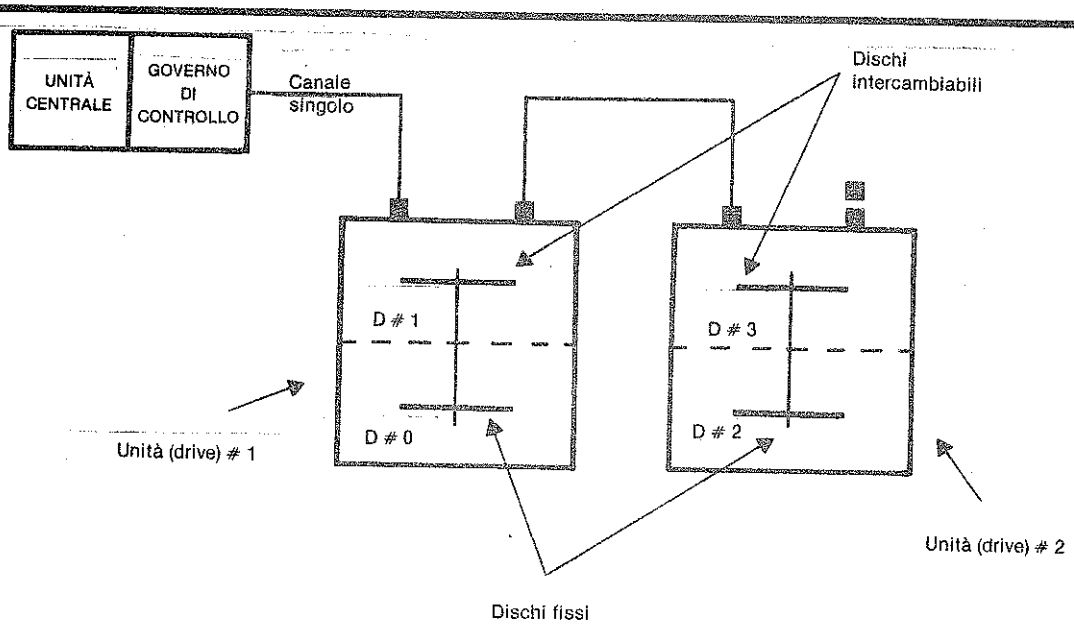


Figura 7-7 Collegamento a due unità DCU

Ogni cilindro è a sua volta suddiviso in 48 settori di 256 byte ciascuno. I settori 0-23 appartengono alla superficie inferiore di ciascun disco; i settori 24-47

a quella superiore. Dei 408 cilindri, i cilindri 0-399 sono utilizzabili per la normale registrazione dei dati; gli altri 8 cilindri sono riservati, traccia per traccia, alla sostituzione di tracce deteriorate. L'unità fisica indirizzabile sul disco è il settore.

Formato della traccia

Il formato della traccia è definito nella figura seguente:

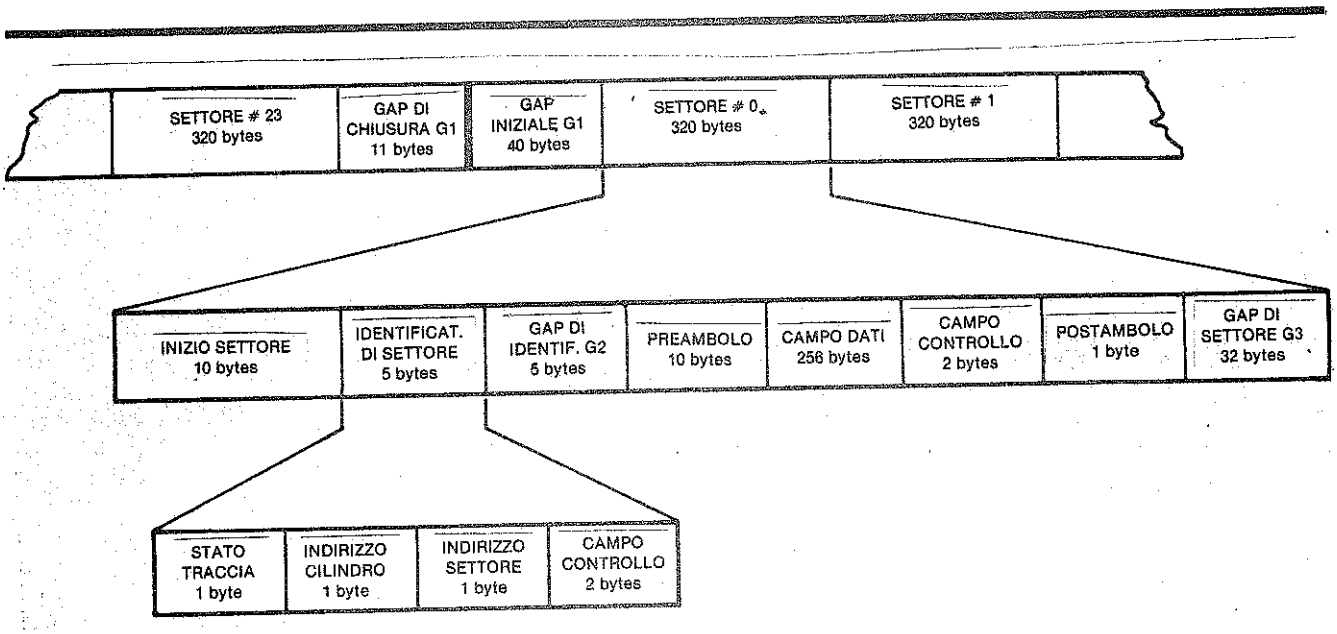


Figura 7-8 Formato traccia

Sul cilindro ØØ sono memorizzate l'etichetta di volume, le etichette del software di base e le etichette di libreria (al più 35 librerie utente per ogni disco). Le etichette sono analoghe a quelle usate nel sistema FDU. Anche per il sistema DCU si parla di disco sistema e di disco utente, con prerogative simili ai corrispondenti dischetti nel sistema FDU.

HDU

Il sistema P6060 HDU comprende 1 o 2 unità HDU, ciascuna dotata di 1 o 2 drivers per un disco a testine mobili. Ogni disco può essere registrabile su una sola faccia oppure su entrambe.

La superficie di ciascun disco è suddivisa in 2ØØ anelli concentrici o tracce registrabili. Più tracce che hanno lo stesso raggio ma appartengono a superfici diverse constituiscono un cilindro. Il concetto di

cilindro è limitato ad un solo disco, anche se il meccanismo d'accesso è unico per tutti e quattro i dischi.

Le unità HDU

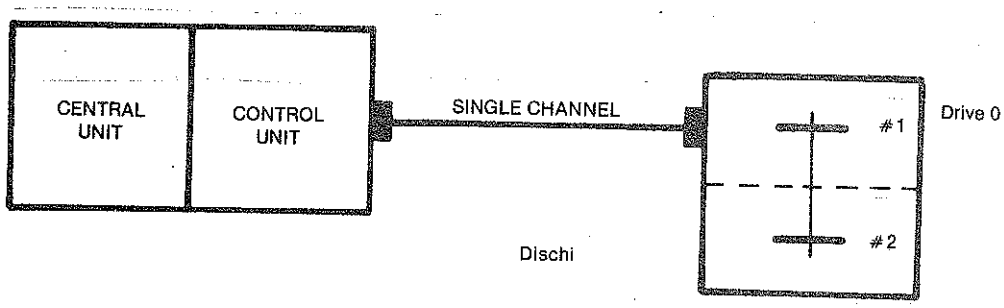


Figura 7-9 Collegamento ad una unità HDU

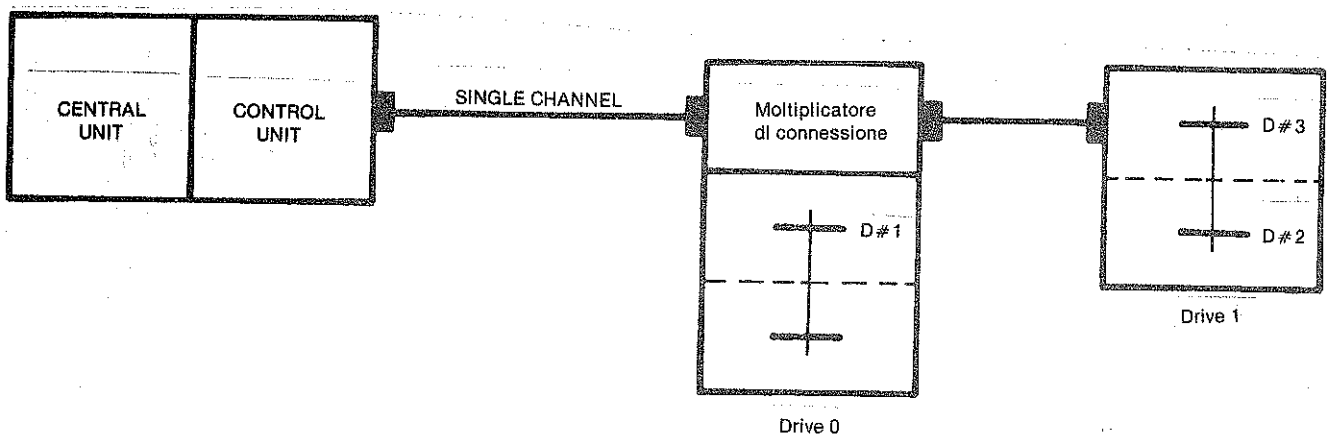


Figura 7-10 Collegamento a due unità HDU

Ogni cilindro è a sua volta suddiviso in 96 o 192 settori di 256 byte ciascuno.

I settori Ø-47 (Ø-95 nel caso siano in tutto 192) appartengono alla superficie inferiore di ciascun disco; i settori 48-95 (96-191) a quella superiore.

Ogni 48 settori registrabili, sulla traccia è posto un settore alternativo, riservato alle sostituzioni di settori deteriorati.

L'unità fisica indirizzabile sul disco è il settore.

Formato della traccia

Il formato della traccia è definito nella figura 7-11.

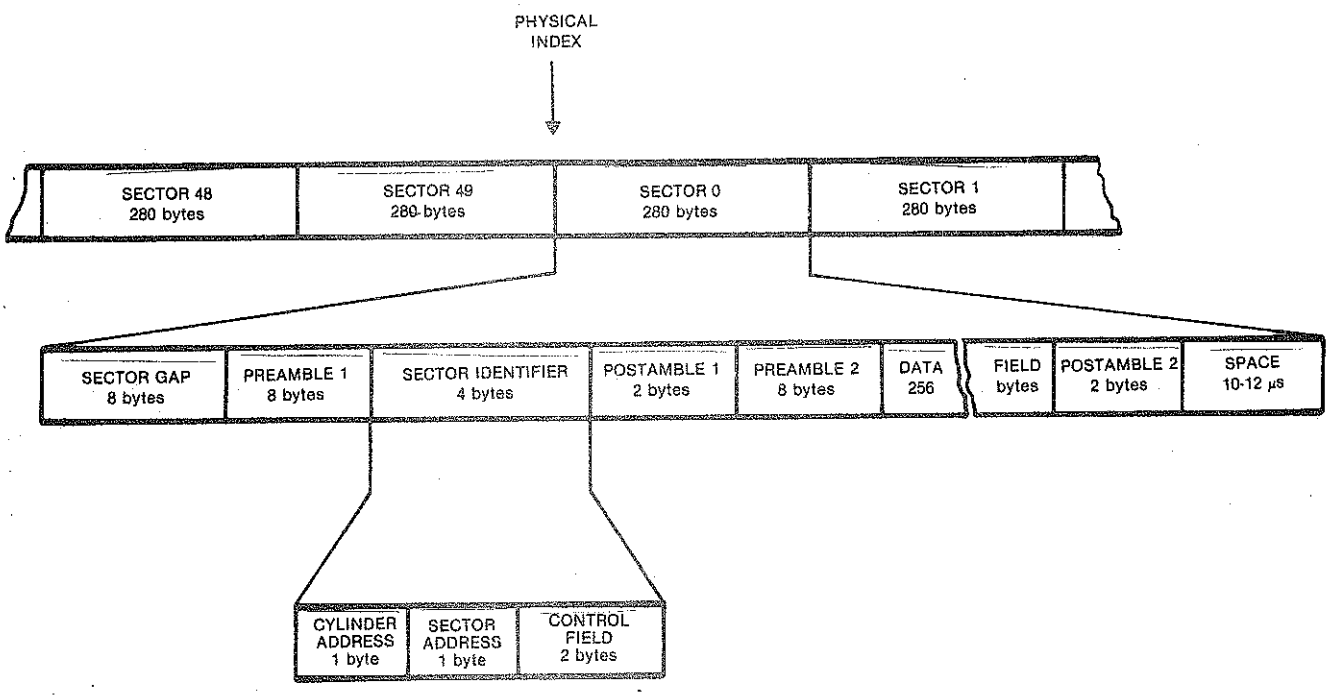


Figura 7-11 Formato traccia

Sul cilindro ØØ sono memorizzate l'etichetta di volume, le etichette del software di base e le etichette di libreria (al più 35 librerie utente per ogni disco).

Le etichette sono analoghe a quelle usate nel sistema FDU.

Anche per il sistema HDU si parla di disco sistema e di disco utente, con prerogative simili ai corrispondenti dischetti nel sistema FDU.

Sostituzione di un disco

La sostituzione di un disco mentre il sistema è in funzione è reso possibile dal comando DCH.

Il comando DCH acquisisce tutte le opportune informazioni al riguardo di:

- utilizzazione delle tracce alternative
- indirizzo e ampiezza delle directories
- indirizzo e ampiezza dlle Control Map

La sostituzione di un disco,effettuata senza la preventiva esecuzione del comando DCH, può comportare l'alterazione permanente di uno o più file a seguito dei comandi SAVE, CREATE, MODIFY, REPLACE, PREPARE, RUN.

Abbiamo definito File System la struttura dei file all'interno di una o più librerie: l'organizzazione è gerarchica, nel senso che in ogni libreria vi è un catalogo dei file contenuti (directory), e per ogni file un sommario del contenuto e delle sue allocazioni (file header).

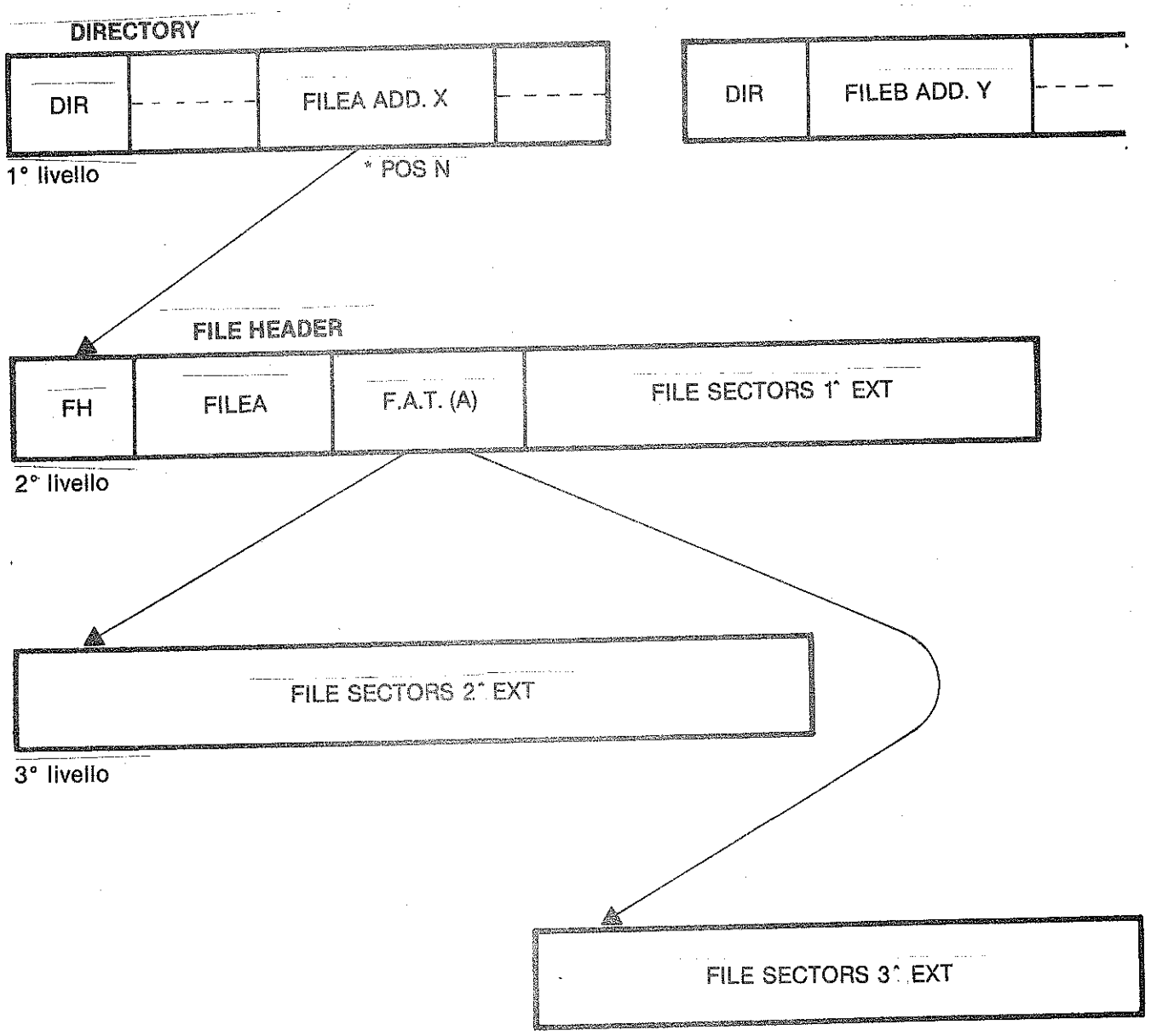


Figura 7-12 Organizzazione gerarchica delle librerie

I tre livelli su cui sono allocate le informazioni che si riferiscono ad un singolo file sono collegati tra loro per mezzo di una serie di puntatori che associano in modo biunivoco tali informazioni.

Control Map

La Control Map è la mappa delle aree libere su disco. E' registrata su tre settori, più o meno al centro del disco (vedi figure A-11 e A-12) ed ha il seguente formato:

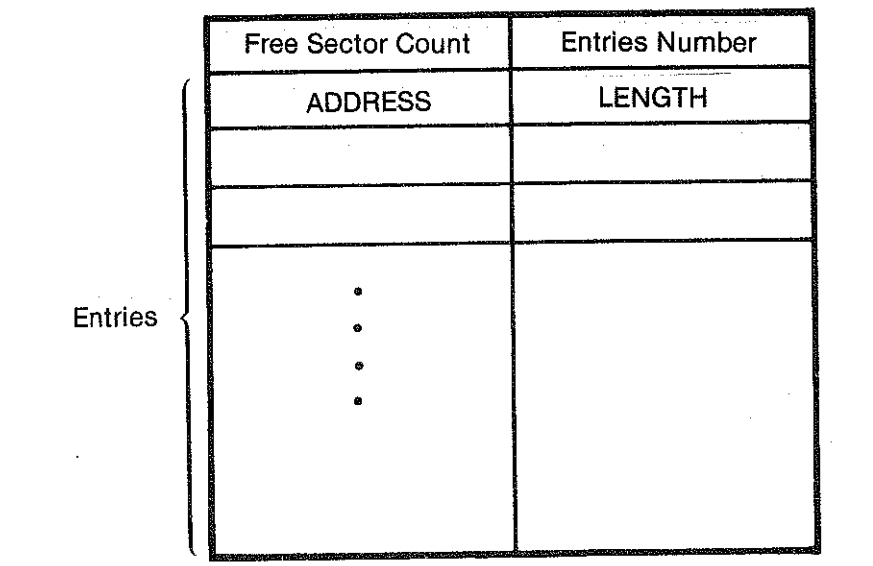


Figura 7-13 Formato dei settori della Control Map

Il significato dei singoli campi è il seguente:

- Free Sector Count: numero totale dei settori liberi
- Entries Number: numero totale delle aree mappate
- Control Map Entry: ad ogni entry sono associate le informazioni che si riferiscono ad un'area libera su disco: l'indirizzo del primo settore dell'area libera e la sua lunghezza in byte

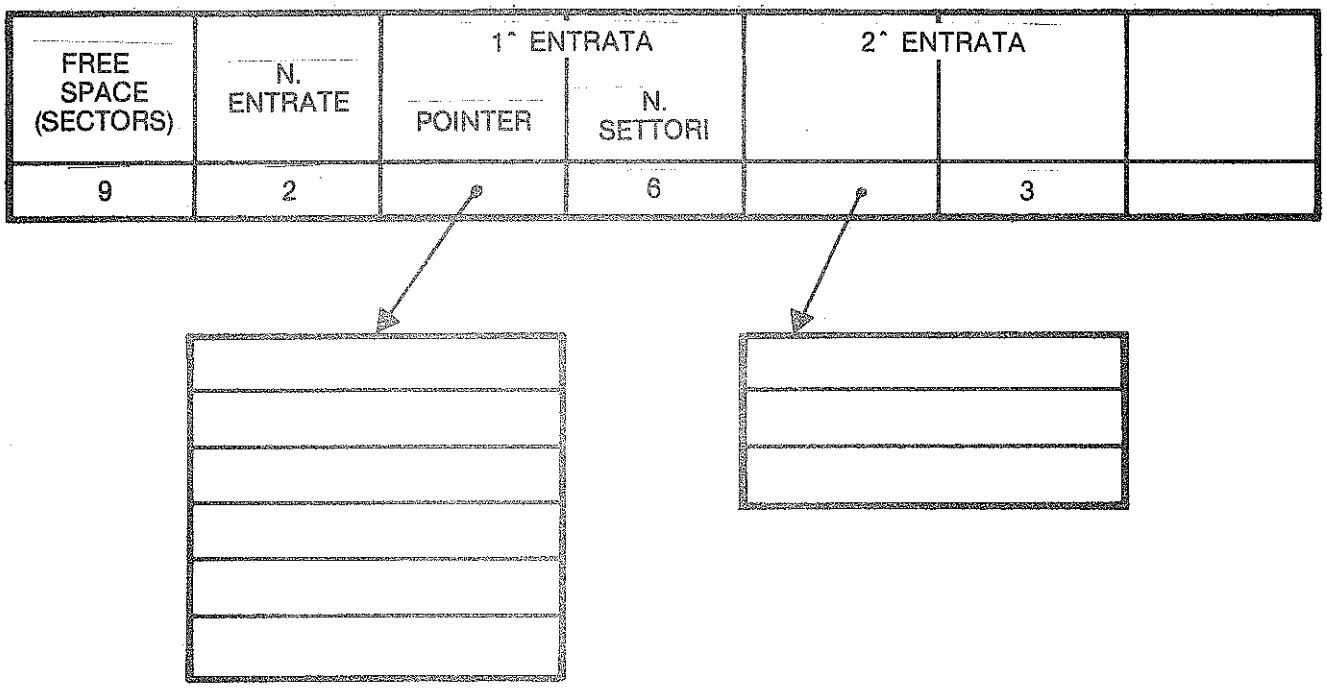


Figura 7-14 Struttura Control Map

Le directories sono strutture di controllo non accessibili all'utente, che possono occupare uno o più settori di disco contigui. Appartengono al primo livello della gerarchia del File System, detto "livello dei nomi simbolici". Nella directory ad ogni nome di file è associato l'indirizzo di un settore di controllo (File header) che ne contiene un sommario. In ciascuna libreria esistono 3 directories, una per ciascuna delle sottolibrerie. Per quanto riguarda la libreria del software di base, funzioni analoghe a quelle della directory sono svolte dalla SEGTAB. Il formato della directory delle sottolibrerie di tipo Package, Comune e Utente è descritto in figura 7-15.

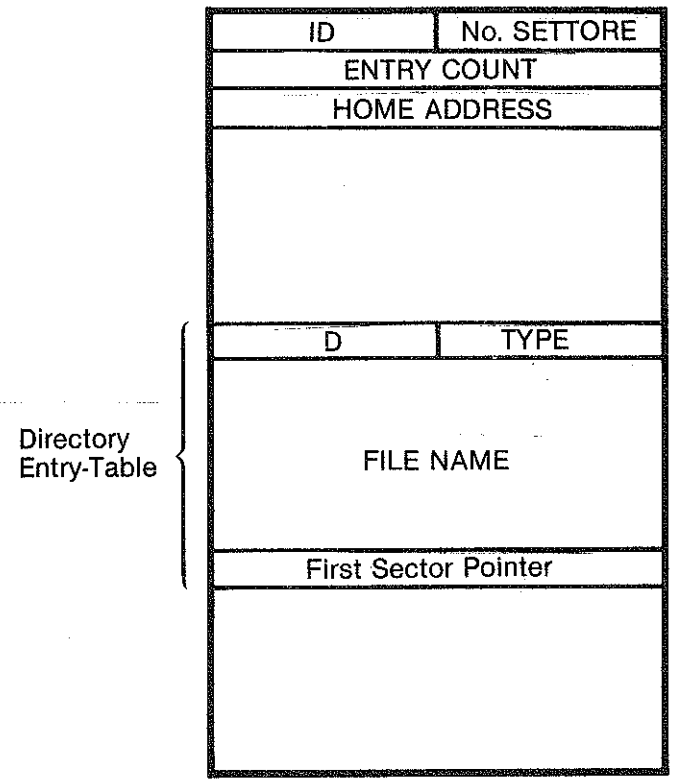


Figura 7-15 Formato dei settori di directory

Il significato dei campi è il seguente:

- ID: è una targa di identificazione che specifica il settore in questione è un settore di directory

- N°: è il numero progressivo del settore

- HOME ADDRESS: è l'indirizzo del settore che la directory occupa su disco

- ENTRY COUNT: è il contatore delle entry contenute in questo settore. Un settore di floppy disk contiene 13 entry, un settore di disco (sistemi DCU/FDU) ne contiene 26

- ENTRY TABLE: è una tabella che contiene un entry per ogni file. Ha memorizzate le seguenti informazioni:

  . D: è un flag di 1 bit che viene acceso quando l'entry è in uso

. TYPE: è un indicatore del tipo di file (progr., testo, dati)

. FILE NAME: contiene il nome del file

- FIRST SECTOR POINTER: indica l'indirizzo del primo settore del file associato alla entry della directory (il settore di FH)

File Header

Nel file header sono contenute le informazioni che riguardano direttamente il tipo e il contenuto del file e ne consentono la gestione. Ogni file che non appartiene alla libreria di sistema ha un proprio file header.

Il file header occupa fisicamente un settore. Appartiene al 2° livello della gerarchia del File System, detto "livello degli attributi". Il formato del file header è descritto in figura 7-16.

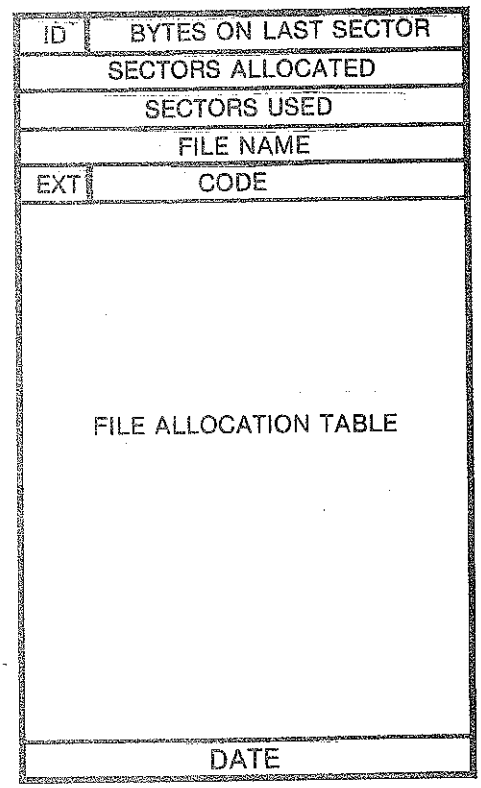


Figura 7-16 Formato del settore di FH

Il significato dei campi è il seguente:

- ID: è un indicatore che specifica che il settore in questione è un file header

- BYTES ON LAST PAGE: numero di byte utilizzati sullo ultimo settore del file. Questa informazione non è usata per file dati di tipo random

- PAGES ALLOCATED: numero di settori assegnato al file

- PAGES USED: numero di settori effettivamente usati. Informazione non utilizzata per file dati di tipo random

- FILE NAME: nome del file

- NO. ZONE: numero delle estensioni su cui è allocato il file

- TYPE: codice che indica il tipo di file

- F.A.T.: tabella in cui è specificata l'allocazione fisica del file sul disco. Ogni file può occupare da 1 a 4 extension (aree disgiunte). Ogni extension è individuata da:

  . indirizzo su disco del settore iniziale

  . lunghezza espressa in numero di settori

- DATES:

  . data della creazione del file

  . data dell'ultima modifica

  Ogni data è espressa da 6 caratteri numerici nella forma:

  dd mm yy

File Sector

I file sector costituiscono il terzo livello della gerarchia del File System, detto "livello dei dati". Essi contengono i dati effettivi del file allocato. Il numero dei dati che possono essere memorizzati in un settore dipende dalle dimensioni del settore stesso e

dal tipo di dato (variabile numerica, stringa, porzioni di testo o di programma).

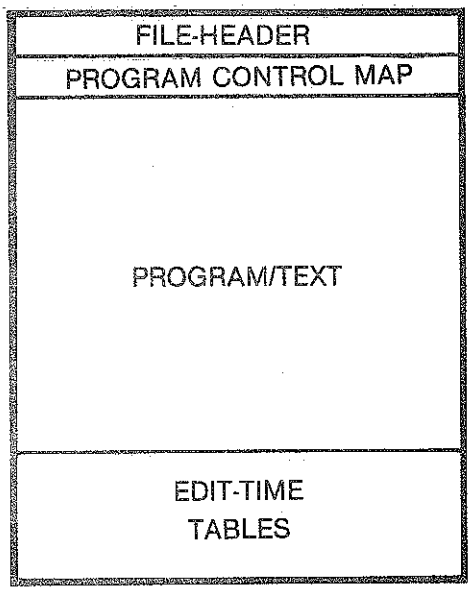


Figura 7-17 Struttura file programma e testo

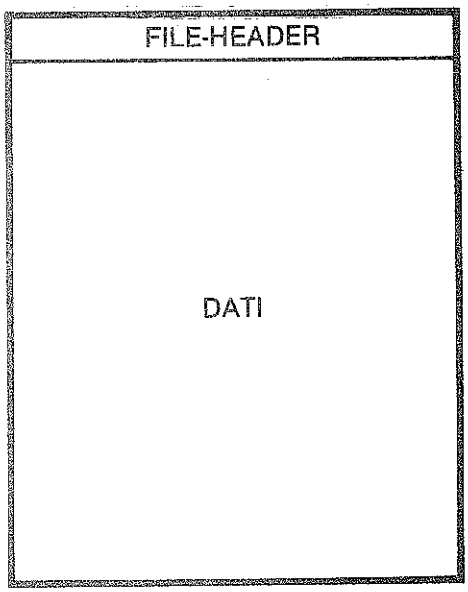


Figura 7-18 Struttura file S, R, Z

Struttura delle librerie

Nei sistemi DCU/HDU possono essere allocate fino a 35 librerie per disco, a differenza dell'unica libreria contenuta in un floppy disk; in tali sistemi è pertanto differente il meccanismo di ricerca di un file.

Su floppy disk il sistema si posiziona sull'unica etichetta di libreria e acquisisce le informazioni necessarie.

Su DCU e HDU viene conservata in memoria una tabella contenente le necessarie informazioni al riguardo delle librerie (al più sei) dichiarate aperte. Il sistema si posiziona sull'etichetta della prima di esse, ed esamina poi nell'ordine di apertura dichiarato le etichette delle librerie aperte finchè rintraccia la libreria sulla quale è richiesta l'informazione. Le etichette delle librerie sono fisicamente memorizzate sulla traccia ØØ, e occupano su DCU/HDU 35 settori, su floppy disk 1 settore. Un apposito puntatore (BOE) è posizionato sull'area del disco in cui sono contenuti i parametri della libreria, le directory e la Control Map, così strutturati:

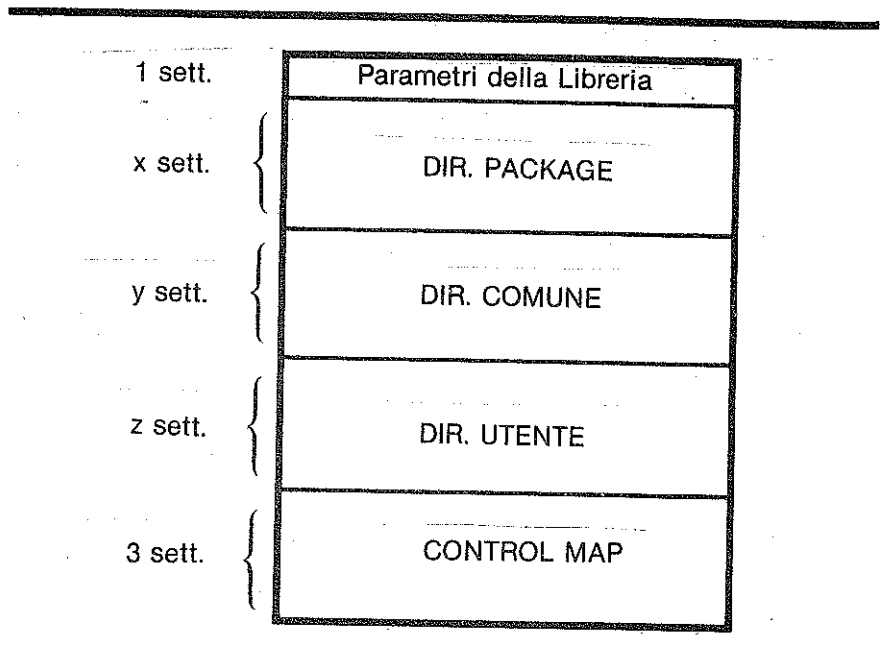


Figura 7-19 Area di disco contenente le informazioni riguardanti una libreria

Strategia di allocazione dei File

I file possono occupare da 1 a 4 estensioni. Quando deve essere memorizzato un file, viene ricercata la più piccola zona libera del disco in grado di contenere interamente il file in un'unica estensione, si cerca di allocarlo nel minor numero possibile di estensione. Se lo spazio complessivo disponibile è superiore alla lunghezza del file, ma non esistono quattro estensioni la cui lunghezza complessiva sia sufficiente a controllarlo, si rende necessaria un'opera-

zione di compattazione della libreria. L'utente può sapere su quanti estensioni è allocato il file per mezzo del comando CATALOG. Le librerie del software di base sono allocate su un'unica estensione.

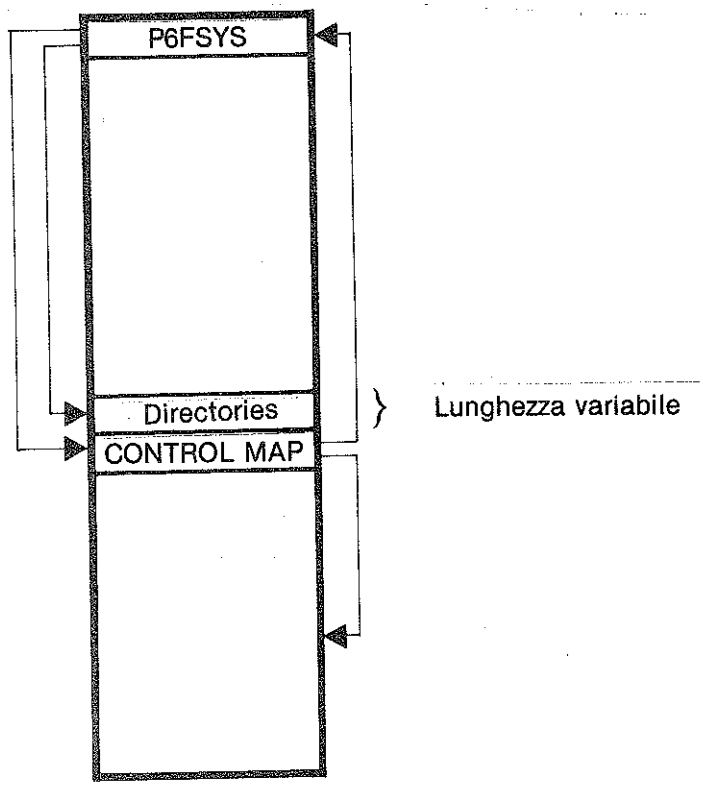


Figura 7-20 Struttura disco utente dopo LBC

Schema di allocazione

Un file può essere allocato su 1,2,3 o 4 extent. I file su più di un extent possono essere compattati con l'utility LIBCOPY.

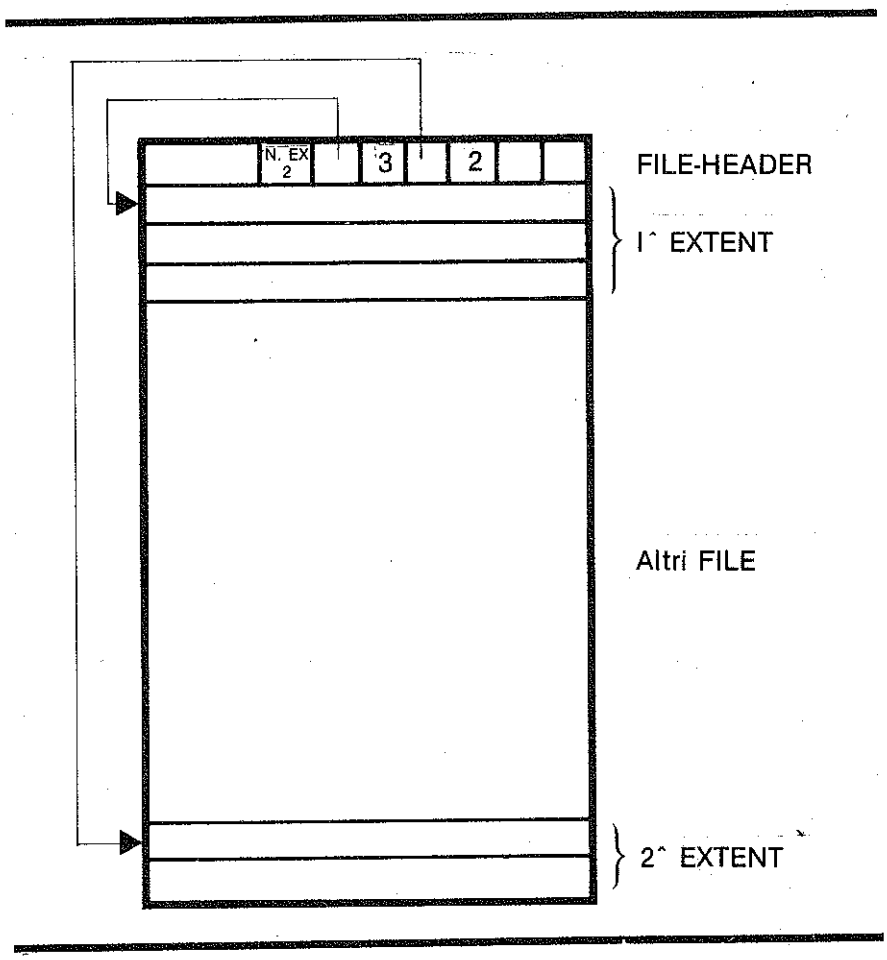


Figura 7-21 Schema di allocazione

Si può avere un file su più extent a seguito di comandi SAVE, CRE, MOD, REP.

Gestione delle librerie e struttúra del File System

Le routine del sistema operativo che si occupano della memorizzazione delle informazioni e dell'accesso alle informazioni archiviate operano sui 3 livelli gerarchici del File System.

1° livello: livello dei nomi simbolici dei file. I moduli agiscono a livello di directory. Le operazioni consentite sono:

- inserimento di un nome simbolico di file nella directory
- cancellazione di un nome simbolico di file dalla directory
- inizializzazione della directory

- scansione della directory

- variazione delle dimensioni dell'area allocata alla libreria

2° livello: livello degli attributi del file: I moduli agiscono a livello di File Header. Le operazioni consentite sono:

- creazione di un file header

- eliminazione di un file header

- troncamento di un file e nuova registrazione del file header

- estensione della dimensione di un file e nuova registrazione del file header

3° livello: livello dei dati: I moduli agiscono direttamente per allocare spazio. Le operazioni consentite sono:

- allocazione dello spazio sul disco

- riallocazione dello spazio su disco a seguito di operazione di troncamento o estensione.

Operazioni del File System

Le operazioni consentite dai moduli del sistema operativo che agiscono sul File System possono essere così riassunte:

- creazione
- cancellazione
- modifica del nome
- modifica della lunghezza
- ricerca
- catalogo

Ogni operazione logica è scissa, quando viene eseguita, in al più 3 fasi, corrispondenti ai 3 livelli del File System sui quali viene effettuata.

Esempio: COMANDO CREATE

1 (liv. 1)
- cerca il file
- inserisci il nuovo file nella directory corrispondente

2 (liv. 2) crea un file header

3 (liv. 3) alloca lo spazio

8. LE OPERAZIONI DI INPUT/OUTPUT

Introduzione

I trasferimenti di informazioni da o verso la memoria centrale sono detti "operazioni di INPUT/OUTPUT (I/O)".

Organizzazione dell'INPUT/OUTPUT

Il sistema P6060 può ricevere richieste di operazioni di I/O sia durante l'esecuzione di comandi di sistema (FETCH, LIST, ecc.), sia durante l'esecuzione di programmi BASIC (istruzione PRINT, READ, ecc.).

I comandi di sistema svolgono operazioni di I/O solamente con le seguenti periferiche gestite logicamente:

- FDU
- DCU
- HDU
- Console
- Printer
- Tastiera
- Display
- Stampante esterna
- Video esterno

I programmi BASIC possono invece svolgere operazioni di I/O sia con periferiche gestite logicamente sia con periferiche gestite fisicamente. A seconda dei casi vengono richiamate parti diverse del Sistema Operativo, a cui è delegata la gestione dell'I/O. Anche questa parte del S.O. è organizzata in livelli gerarchici come indicato in Figura 8-1.

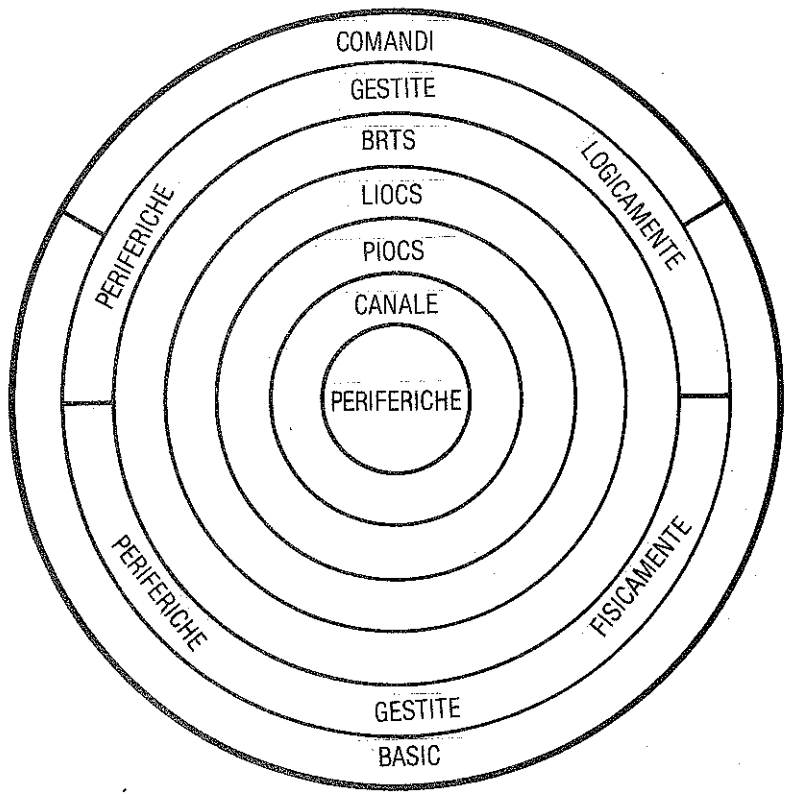


Figura 8-1 Livelli gerarchici

Il Basic RUN-TIME Support (BRTS)

Si tratta di un gruppo di routine che hanno lo scopo di convertire i dati dal formato adatto alla elaborazione da parte del programma BASIC (formato BASIC) al formato in cui i dati stessi sono rappresentati sulle periferiche (formato di periferica) e viceversa. Il formato BASIC e i formati di periferica vengono trattati in modo più approfondito nel paragrafo "Formati e Conversioni" di questo capitolo. I comandi di sistema non utilizzano le routine di BRTS, poichè i dati da loro trattati non devono essere convertiti in formato BASIC.

Il Logical INPUT/OUTPUT Control System (LIOCS)

Ogni periferica è in grado di trattare contemporaneamente solo un insieme ridotto di caratteri che definiamo "record".

Il LIOCS, ricevuta una sequenza di caratteri nel formato della periferica, provvede ad organizzarli in record e a trasferirli alla periferica interessata per

mezzo del PIOCS. Viceversa, ricevuta la richiesta di trasferire al BRTS un certo numero di caratteri, il LIOCS li colleziona richiedendo al PIOCS un numero di record sufficiente a soddisfare la richiesta. Per eseguire queste operazioni il LIOCS ha bisogno di un'area di lavoro; per le periferiche gestite logicamente l'area di lavoro viene automaticamente allocata dal Sistema Operativo; per le periferiche gestite fisicamente l'area di lavoro viene assegnata da programma BASIC (istruzioni BUILD, ASSIGN, ecc.).

Il Physical INPUT/OUTPUT Control System (PIOCS)

Il PIOCS provvede a gestire il trasferimento dei record da o per il canale. Prima che l'operazione di trasferimento abbia inizio, il PIOCS verifica:

- l'assenza di uno stato di anomalia sul canale
- la disponibilità del canale ad intraprendere una nuova operazione
- il corretto collegamento del canale con la periferica a cui è indirizzato il record

Quindi avvia l'operazione di trasferimento, e quando questa è conclusa verifica che sia stata eseguita correttamente.

Per eseguire queste operazioni il PIOCS si serve di una memoria di transito (buffer) specifica per ogni canale. Per le periferiche gestite logicamente il buffer viene allocato automaticamente dal sistema operativo. Per le periferiche gestite fisicamente il buffer viene allocato da programma BASIC tramite l'istruzione BUFFER.

Il canale

Il canale è un dispositivo di INPUT/OUTPUT le cui operazioni non interferiscono con le operazioni che si svolgono su altri canali. Il canale è formato dal Governo Hardware delle Unità Periferiche e da moduli di Sistema Operativo che attuano lo scambio di informazioni tra Unità Centrale e Unità Periferiche. La seguente tabella riporta i canali disponibili e i loro parametri caratteristici:

Nome Canale	Modo di funzionamento	Frequenza MAX possibile sul canale Kbyte/sec.	Frequenza MAX protetta da Sist. Operat. Kbyte/sec.	Carico I/O V=Velocità di trasmissione dati della per. Kbyte/sec.
IPSO	MULTIPLEX	2Ø	4	V/4Ø
	SINGOLO	2Ø	2Ø	V/1ØØ
RS232	MULTIPLEX	2	2	V/4Ø (se non FREE RUNNING)
CURRENT LOOP	MULTIPLEX	2	2	V/4 (se FREE RUNNING)
IEEE-488	DMA	1ØØØ	3ØØ	V/2ØØØ
FDU	SINGOLO	3Ø	3Ø	Ø.3Ø
DTM	DMA	13Ø	13Ø	Ø.15
PRINTER	MULTIPLEX			
TASTIERA DISPLAY CONSOLE	MULTIPLEX			

Modo di funzionamento del Canale: Il sistema P6060 consente di gestire fino a 16 canali scelti fra quelli precedentemente elencati. In base al modo di funzionare si possono distinguere tre tipi di canale:

- Multiplex: è un canale che può essere attivo insieme ad altri dello stesso tipo

- Singolo : è un canale che se è attivo obbliga altri canali dello stesso tipo ad attendere per poter iniziare l'operazione di I/O

- DMA : è un canale in cui lo scambio dei dati viene fatto direttamente con la memoria senza interessare la CPU. Questo tipo di canale può essere attivo insieme ad altri dello stesso tipo

I tre tipi di canali suddetti possono essere attivi contemporaneamente; il Sistema Operativo ne garantisce

però il funzionamento per velocità di scambio dati inferiori a quelle indicate in tabella. E' consentito attivare il canale anche a velocità superiori, fino a quella massima consentita sul canale. In tal caso sarà cura dell'utente verificare che il carico di I/O non risulti eccessivo (maggiore di 1), calcolandola secondo la tabella precedente (carico I/O).

Esempio di calcolo del CARICO di I/O: La velocità di trasmissione dei dati sulla periferica è espressa sempre in bit/sec. Siccome il Carico di I/O è espresso in Kbyte (chilo byte), per calcolare la velocità effettiva (v) di trasmissione l'utente dovrà eseguire le seguenti operazioni:

supponendo di avere:

Canale IPSO

Modo di funzionamento MULTIPLEX
Velocità di trasmissione 16ØØ bit/sec.
16ØØ : 1Ø = 16Ø byte
16Ø : 1.ØØØ = Ø.16 Kbyte (V=velocità effettiva)
Carico di I/O
Ø16 : 4Ø = Ø.ØØ4

Formati e conversioni

Sia A$ una variabile alfanumerica (variabile stringa) a cui sia stato assegnato un valore qualsiasi, per esempio "PIPPO".

La rappresentazione in memoria (Formato BASIC) della variabile A$ consiste di tre elementi:

- il suo nome (A$)

- gli attributi di memoria della variabile

  . lunghezza di allocazione
  . indirizzo di memoria della prima locazione allocata

- il suo valore ("PIPPO"), con i suoi attributi:

  . valore valido/stringa non assegnata
  . lunghezza attuale, 5 caratteri

NOME .	ATTRIBUTI DI MEMORIA	VALORE

A seconda del tipo di periferica su cui si vuole inviare questa variabile, deve essere modificata la sua rappresentazione, al fine di renderla compatibile con le caratteristiche della periferica. Ad esempio se si vuole stampare **A$** deve essere inviato alla stampante il solo valore "PIPPO" di **A$**.

Nel caso di registrazione su disco, al contrario, poichè quanto viene registrato dovrà in futuro essere riletto non è sufficiente salvare il solo valore della variabile, ma bisogna corredarlo di tutte quelle informazioni (separatori) che permetteranno di riconoscere ed identificare tale valore durante le operazioni di lettura.

Un problema analogo si pone quando si deve riconoscere un valore da attribuire ad una variabile entro una stringa di caratteri inviata da una periferica; anche in questo caso un separatore consente di isolare il valore a cui si vuole accedere.

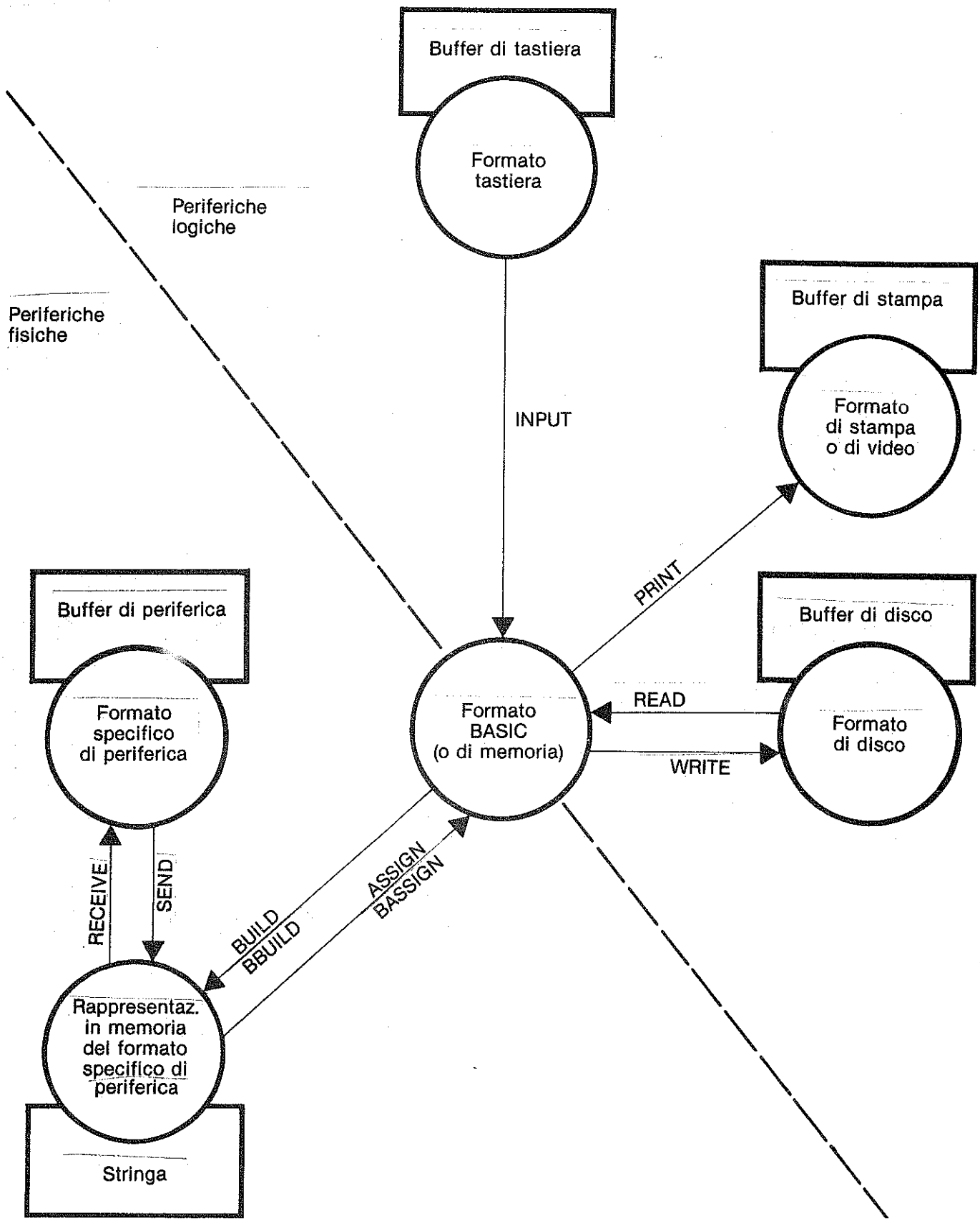


Figura 8-2 Trasformazione dei dati da formato di periferica a formato BASIC e viceversa

Per le periferiche gestite logicamente la rappresentazione dei dati è univoca; il sistema operativo trasforma i dati dal formato di periferica a quello BASIC e viceversa.

Per le periferiche gestite fisicamente, invece, una rappresentazione univoca dei dati risulta impossibile; del sistema operativo fanno parte alcune istruzioni BASIC che consentono di eseguire le opportune trasformazioni di formato. L'appendice C riporta i vari formati per le periferiche gestite logicamente e quelli generati dalle istruzioni BASIC.

9. PROGRAMMI DI UTILITA'

Le utility o programmi di utilità, costituiscono parte integrante, sia logicamente che fisicamente, delle librerie di sistema. Ogni utility consiste in un segmento. La differenza principale tra i programmi di utilità e gli altri segmenti del sistema operativo consiste nel fatto che i primi vengono caricati in memoria interna nell'area utente: il caricamento di un'utility comporta pertanto la distruzione del precedente contenuto del working file. Ogni utility realizza in modo completo un'operazione di servizio del sistema (ad esempio, la verifica della leggibilità di un dischetto, la copia di un disco o di una libreria, e così via).

Caricamento UTILITY

Le utility vengono richiamate con il comando EXEC. Il processor del comando EXEC contiene una tabella nella quale sono specificati i nomi di tutte le utility esistenti in libreria, e provvede a scandire questa tabella fino a trovare il nome dell'utility interessata.

Gli indirizzi di disco delle utility sono ordinati nella tabella SEGTAB secondo l'ordine della tabella dei nomi: una volta trovato il nome, risulta noto anche l'indirizzo di disco dell'utility.

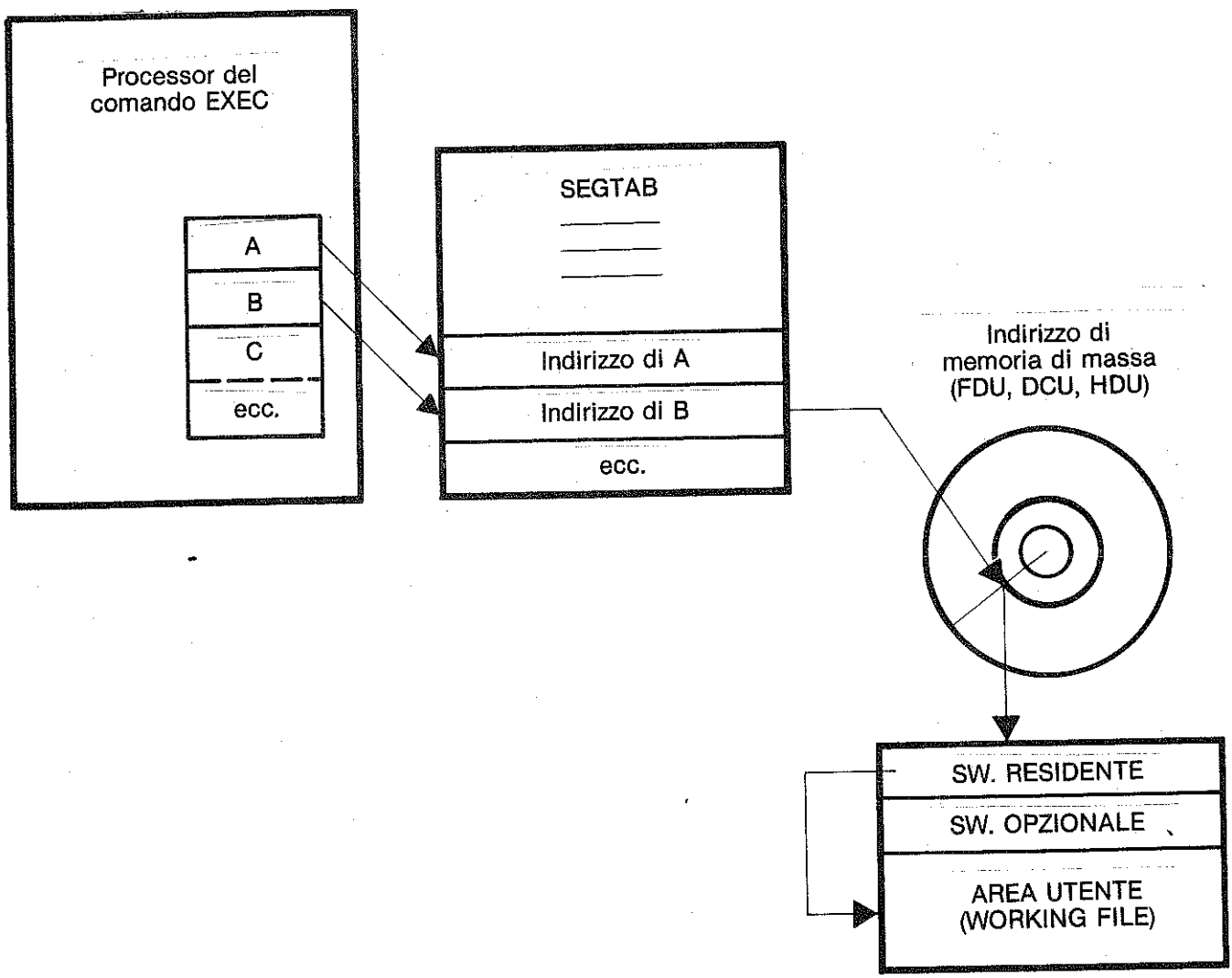


Figura 9-1 Processor del comando EXEC e SEGTAB

L'operazione descritta si svolge nel modo seguente:

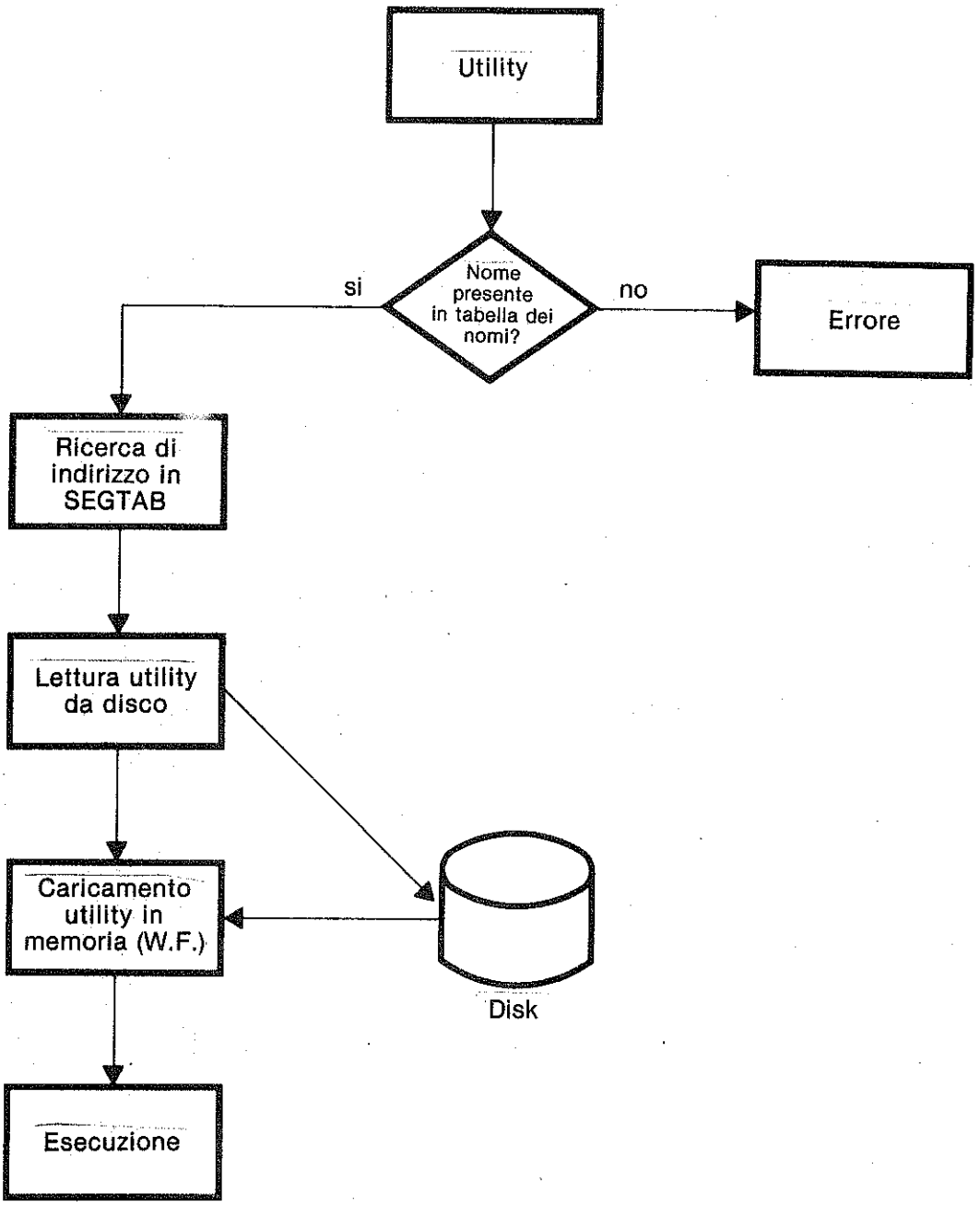


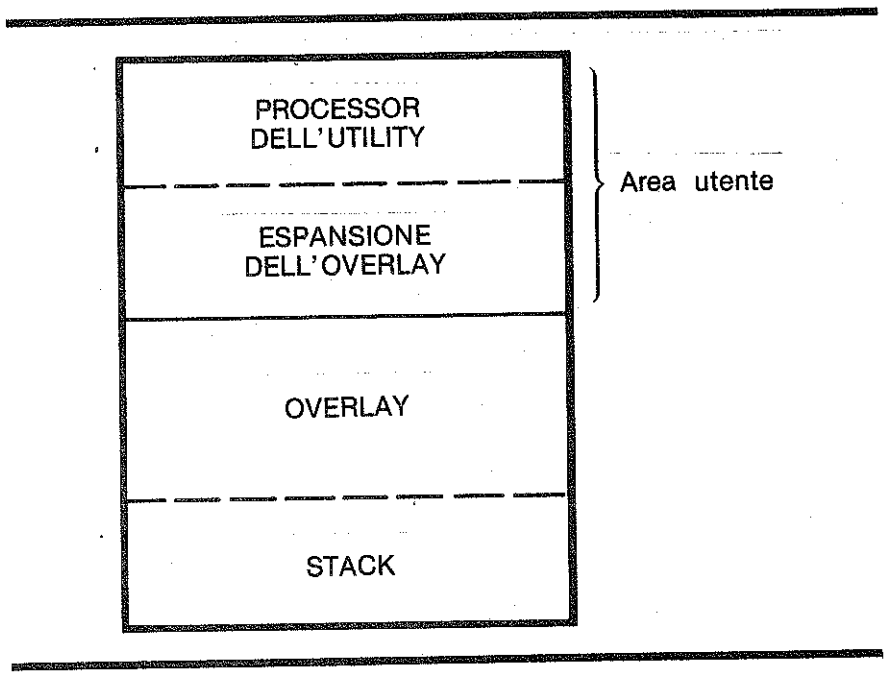
Figura 9-2 . Caricamento di una utility da disco in memoria

Il comando EXEC carica quindi l'utility da disco in memoria, nell'area utente (WORKING-FILE), distruggendone così il contenuto. Il controllo dell'esecuzione viene quindi passato direttamente all'utility.

L'esecuzione di un'utility richiede, come quella di un comando, una serie di funzioni comuni a più utility

e comandi (ricerca dei nomi dei file , letture da disco, ecc.). Tali funzioni vengono svolte a seguito del caricamento in area di OVERLAY degli opportuni moduli di S.O.

Alcune utility richiedono, nel corso dell'esecuzione, la sostituzione del disco utente o del disco sistema. In quest'ultimo caso tutti i moduli richiamati vengono caricati in area di OVERLAY (che pertanto viene espansa per contenerli) ove rimangono anche se momentaneamente inutilizzati:



Essendo i moduli già presenti in memoria al momento del richiamo non è può necessario leggerli dal disco sistema, ed in luogo di questo può essere montato un disco differente.

Esecuzione

L'esecuzione di un'utility comporta un'interazione diretta operatore-sistema: il sistema invia all'utente una serie di messaggi, cui l'utente deve rispondere con le azioni prescritte. L'esecuzione di un'utility si sviluppa attraverso una serie di passi successivi, descritti nella figura seguente, dove con * sono indicate le operazioni eseguite manualmente dall'utente:

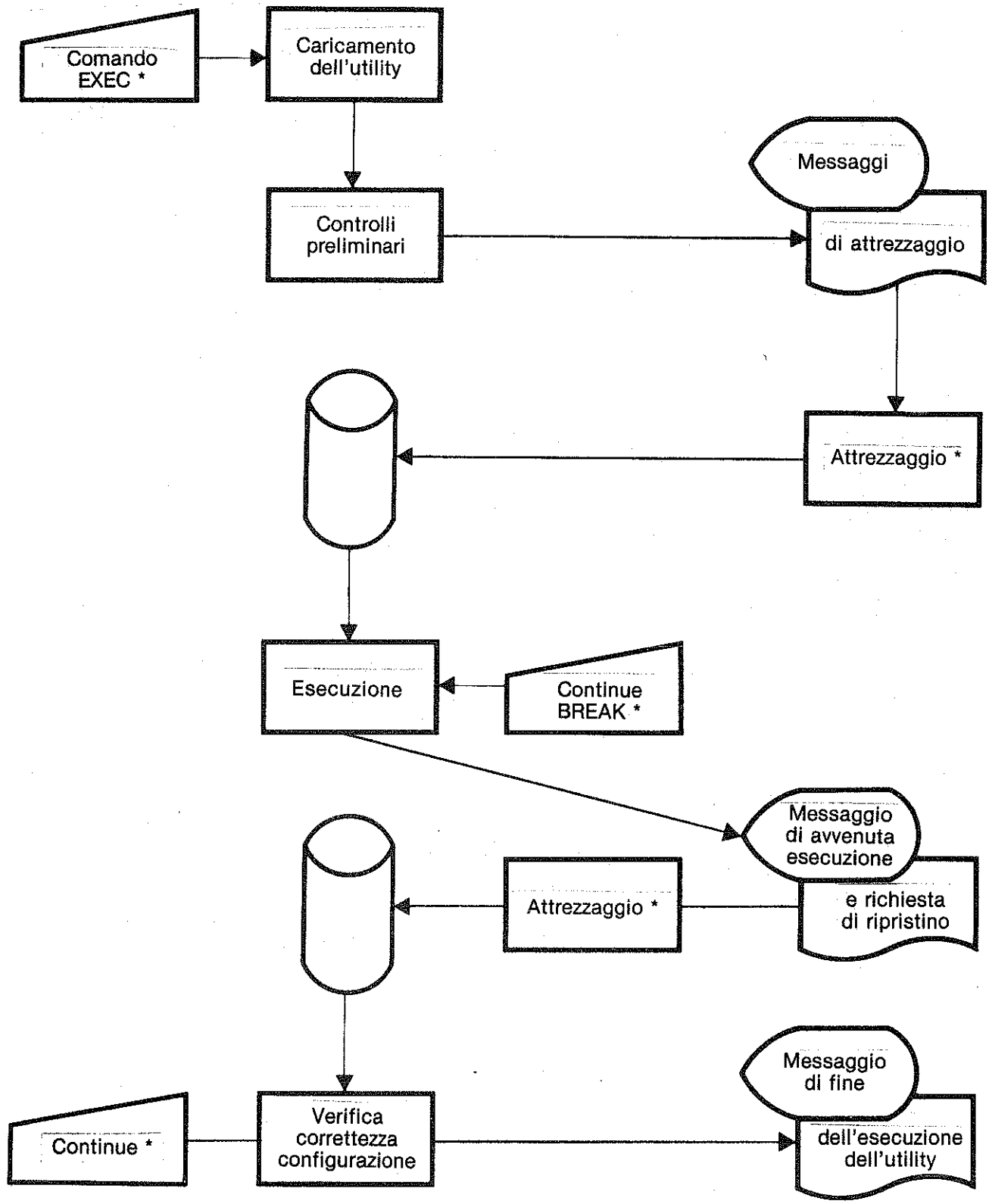


Figura 9-3 Esecuzione di una utility

Controlli preliminari

I controlli preliminari dipendono dal tipo di utility richiesta, e possono consistere nel verificare se c'è spazio sufficiente su disco (es.: copie file o librerie), se le operazioni richieste sono congruenti con la configurazione del sistema (es.: operazioni su disco utente con sistema FDU configurato monodisco), se è stata introdotta la parola chiave richiesta (libreria con parola chiave su sistema DCU). Se i controlli hanno esito positivo, l'utility verifica se l'esecuzione richiede un attrezzaggio particolare del sistema: in questo caso viene visualizzata la richiesta di attrezzaggio all'operatore; in caso contrario continua l'esecuzione.

Attrezzaggio del sistema

La richiesta di attrezzaggio del sistema avviene attraverso messaggi inviati su display ed eventualmente su stampante. L'operatore deve rispondere al messaggio eseguendo quanto richiesto ed inviando il comando CONTINUE (l'uso del comando BREAK blocca l'esecuzione dell'utility). Durante questa fase il sistema è in stato OPERATOR-CALL.

Se vi è stata una richiesta di attrezzaggio, la prima operazione eseguita in questa fase è una verifica delle operazioni compiute dall'operatore. Viene verificato infatti se il tipo di dischi richiesti in linea sono effettivamente presenti (in caso contrario viene ripetuta la richiesta di attrezzaggio); se le operazioni richieste comportano una distruzione delle informazioni su un disco di output (es.: copia disco), l'utility indica quali sono le informazioni che verranno distrutte. L'operatore può così verificare che non si perdano informazioni ancora valide e proseguire nell'esecuzione con il comando CONTINUE.

Ripristino configurazione valida di dischi

Alla fine dell'esecuzione, quando siano state eseguite sostituzioni di disco, l'utility chiede il ripristino di una configurazione di dischi che comprenda almeno un disco sistema in linea.

L'utente deve attrezzare correttamente il sistema, e inviare il comando CONTINUE. Durante questa fase il sistema è stato OPERATOR-CALL.

Dopo l'introduzione del comando CONTINUE viene verificata la configurazione dei dischi:

- se questa è valida, il controllo viene ceduto al MONITOR (in caso vi siano più dischi sistema in linea, per disco sistema FDU viene assunto quello inferiore e per disco sistema DCU/HDU quello dichiarato in fase di inizializzazione. I dischi in sovrappiù vengono considerati dischi utente non inizializzati).

- se la configurazione non è valida il sistema ricicla sulla richiesta

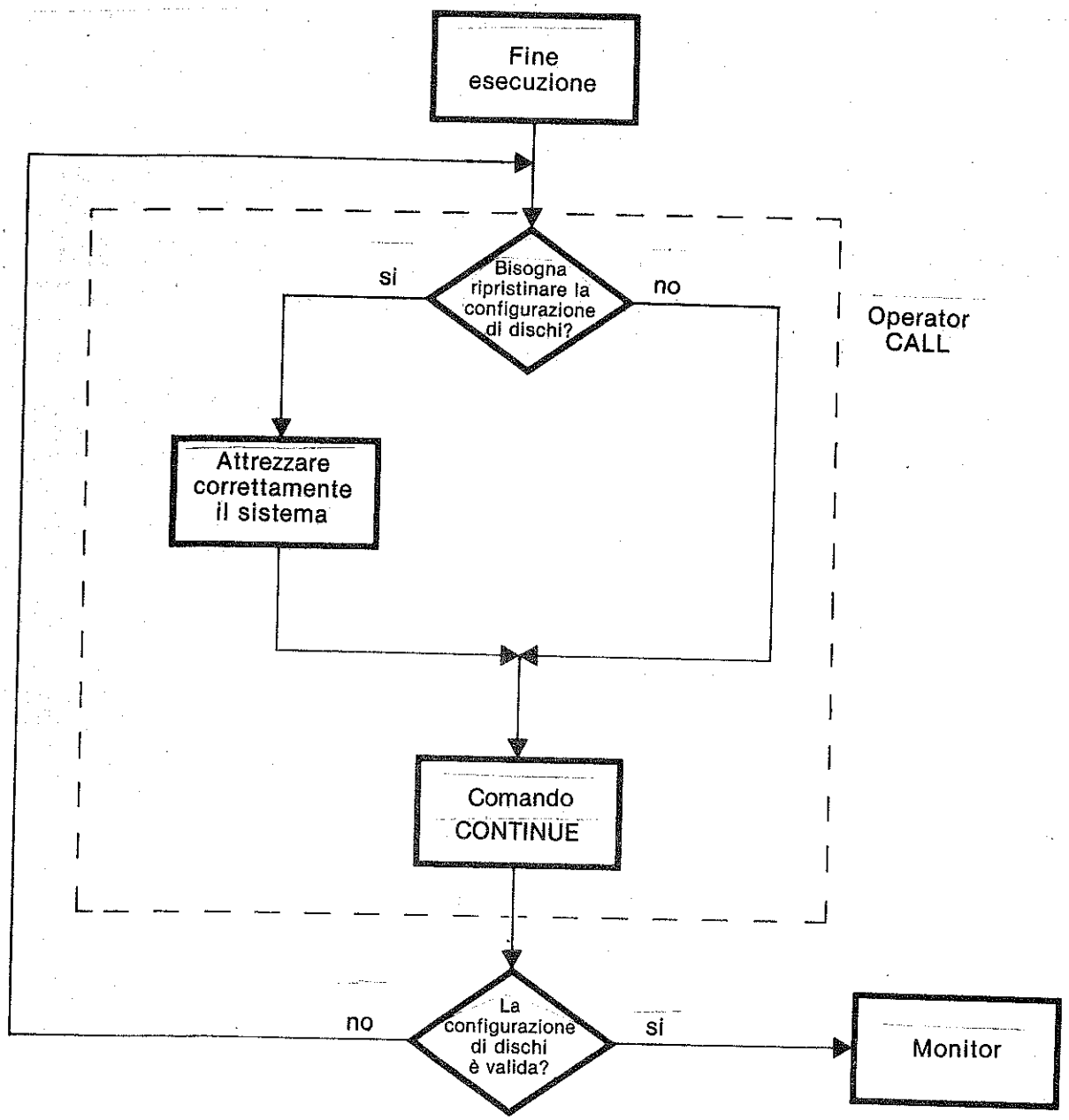


Figura 9-4 Ripristino configurazione valida di dischi

10. LA STAMPANTE INTEGRATA COME PLOTTER

In questo capitolo vengono descritti i metodi di creazione, archiviazione e stampa di immagini. L'archiviazione delle immagini viene effettuata su file dati, mentre la stampa può essere ottenuta sulla stampante integrata oppure su un'unità PLOTTER esterna al P6060. Un'estensione del BASIC consente di ottenere queste prestazioni in modo semplice ed efficace.

I programmi di tipo PLOTTER

Per programma PLOTTER si intende un insieme di istruzioni che permettono di ottenere una immagine.

Compilazione di istruzioni PLOTTER

Le istruzioni rivolte al PLOTTER (vedi: "Opzione PLOTTER" cod. GP39737ØØR) facenti parte di un programma vengono compilate in modo diverso a seconda del tipo di periferica su cui si vuole ottenere la rappresentazione dell'immagine; per la compilazione viene richiamata una funzione di sistema (BRTS) di supporto al programma BASIC, specifica per ciascuna istruzione.

Preesecuzione di istruzioni PLOTTER

Nella fase di preesecuzione, il preesecutore controlla se all'interno del programma è presente la istruzione EXTERNAL-PLOTTER, che indica che la figura deve essere tracciata con periferiche di tipo PLOTTER

- se l'istruzione non è presente (immagine tracciata su stampante integrata) vengono agganciate al programma le funzioni di sistema specifiche del PLOTTER che sono state richiamate dal compilatore (BRTSLM)

- se l'istruzione è presente viene agganciata al programma un'unica funzione di sistema responsabile del collegamento tra istruzioni PLOTTER e il driver della particolare unità esterna PLOTTER, su cui si vuole ottenere l'immagine. Questo driver è una funzione multilinea del BASIC (funzione P) che normalmente viene fornita dall'Olivetti, ma che può anche essere scritta dall'utente

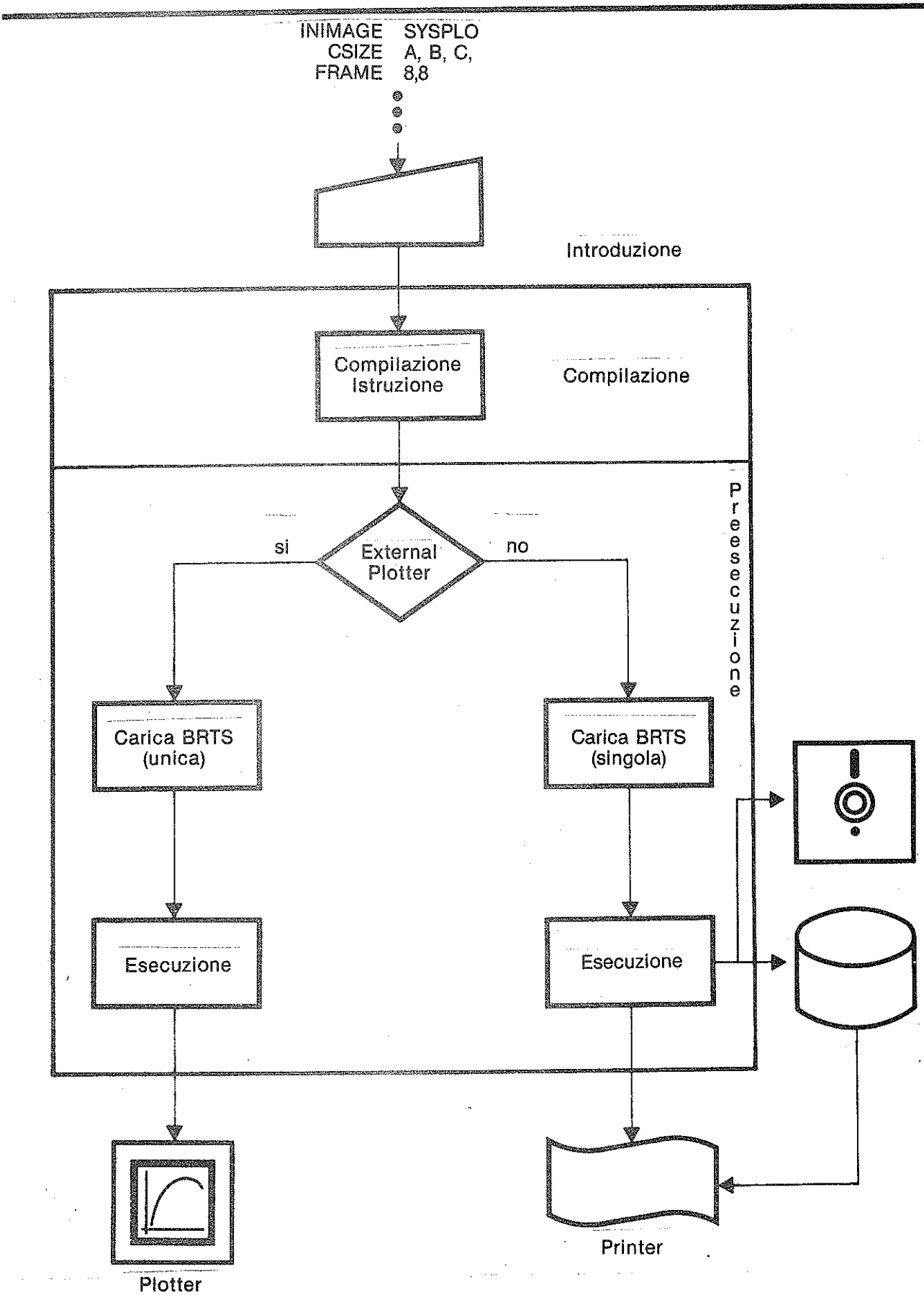


Figura 10-1 Esecuzione dei programmi di tipo PLOTTER

Esecuzione di programmi di tipo PLOTTER

Il tracciamento dell'immagine su periferica PLOTTER avviene contemporaneamente alla esecuzione delle istruzioni del programma. Per il tracciamento di punti, linee o caratteri avviene uno spostamento del pennino sulla carta nelle posizioni indicate dalle istruzioni.

La stampante integrata non può tracciare linee dal basso verso l'alto, perchè è vincolata al movimento della carta, e inoltre durante l'esecuzione può essere impiegata per visualizzare messaggi estranei alle operazioni di tracciamento (per es. a seguito di istruzioni PRINT); per questi due motivi l'immagine viene completamente sviluppata e memorizzata prima che venga iniziata qualsiasi operazione di tracciamento.

Uso della stampante integrata

L'area (FRAME) su cui viene rappresentata l'immagine è vista come un insieme discreto di punti. Ogni pollice dichiarato nella istruzione FRAME corrisponde a 70 punti. Per immagazzinare l'immagine prima della stampa, sono necessari:

- un buffer in memoria interna sul quale generare la immagine. Il buffer in questione può essere insufficiente a contenere l'intera immagine definita dalle istruzioni del programma, e perciò, ogni qualvolta si satura la capacità del buffer, diventa necessario registrare il contenuto (immagine parziale) su di un file

- un file in memoria esterna (FDU, DCU) che deve poter contenere tutte le immagini parziali generate dal programma

Le dimensioni del buffer e il nome del file devono essere indicati dall'utente prima di ogni altra operazione, utilizzando le apposite istruzioni INIMAGE e LDIMAGE.

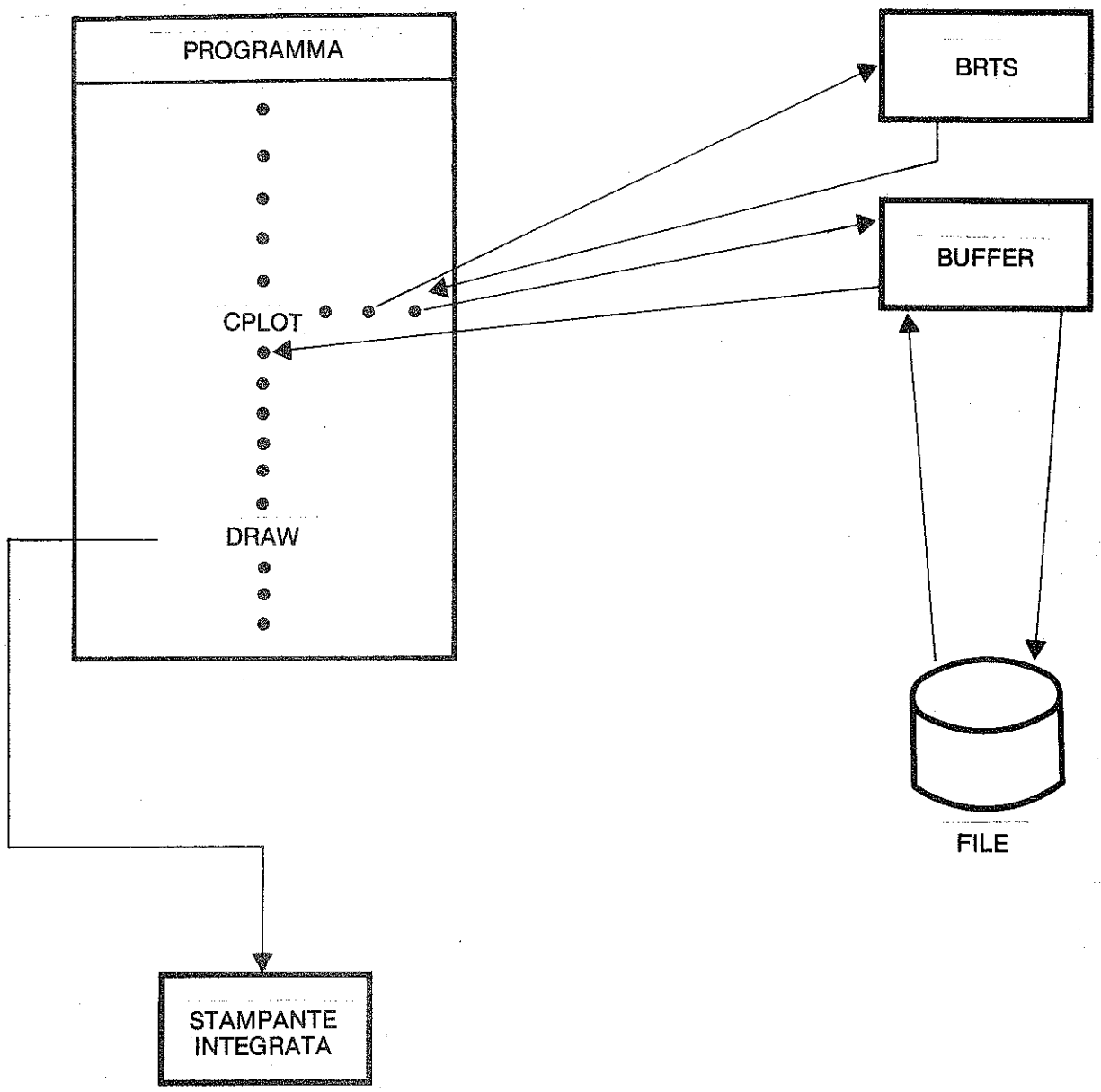


Figura 10-2 Uso della stampante integrata

Inizializzazione del Buffer e del File

Il buffer ed il file vengono inizializzati tramite la istruzione INIMAGE o LDIMAGE.

I byte che compongono il buffer hanno 4 possibili formati:

1)

1	0	0	0	0	0	0	0

- segnala il primo e l'ultimo byte della parte di buffer usata per descrivere l'immagine

2)

1	X	X	X	X	X	X	X

- segnala la presenza di una colonna di 7 punti di cui quelli corrispondenti ai bit con valore 1 sono marcati

3)

0	n

con n = numero binario su 7 bit

- segnala la presenza di n colonne di 7 punti non marcati

4)

0	0	0	0	0	0	0	0

- indica che il byte al momento non è utilizzato per la descrizione (byte vuoto)

L'immagine viene descritta dai byte di formato 2 e 3. L'uso di byte di formato 3 permette di compattare la descrizione dell'immagine. Il sistema si riserva 256 byte del buffer come area di lavoro per la generazione dell'immagine. Il contenuto del buffer dopo l'inizializzazione è:

Anche sul file il sistema si riserva 384 byte:

- i primi 256 contengono l'ultima area di lavoro

- i restanti 128 contengono le informazioni relative alle immagini parziali contenute nel file stesso

Uso del Buffer e del File per il marcamento dei punti

Per comprendere il processo di generazione della immagine, si deve immaginare di partire da una figura tutta bianca e di voler marcare il 100° punto della prima riga. Si deve ottenere una immagine così composta:

- 1 byte di formato 3 che descrive 99 colonne di punti non marcati

- 1 byte di formato 2 che descrive la colonna di punti il primo dei quali marcato

- restanti byte di formato 3 fino al completamento dell'immagine

Il marcamento di ogni punto comporta un rifacimento della descrizione dell'immagine:

- vengono ridefiniti i contenuti dei byte di formato 3 che la rappresentano

- viene inserito un nuovo byte di formato 2 nel posto appropriato

E' chiaro che marcando nuovi punti diminuiscono i byte vuoti e aumentano quelli di formato 2. Quando non ci sono più byte vuoti, qualora debbano essere marcati altri punti il contenuto del buffer viene registrato sul file, il buffer di memoria viene riinizializzato e il tracciamento riprende. All fine della generazione dell'immagine il file su PLOTTER contiene tutte le immagini parziali tranne l'ultima, che al momento è ancora registrata sul buffer in memoria.

Algoritmo per il marcamento dei punti

Le dimensioni della immagine da creare sono dichiarate nell'istruzione FRAME e hanno i seguenti valori massimi:

- 8 pollici per la larghezza
- 936 pollici per la lunghezza

L'istruzione SCALE consente di definire i valori massimi per le ascisse e le ordinate relative al FRAME dichiarato e definisce perciò implicitamente l'origine del sistema e le unità di misura lungo gli assi cartesiani. Diminuendo o aumentando l'unità di misura degli assi cartesiani, si possono modificare a piacimento le dimensioni dell'immagine e la sua sistemazione nel FRAME.

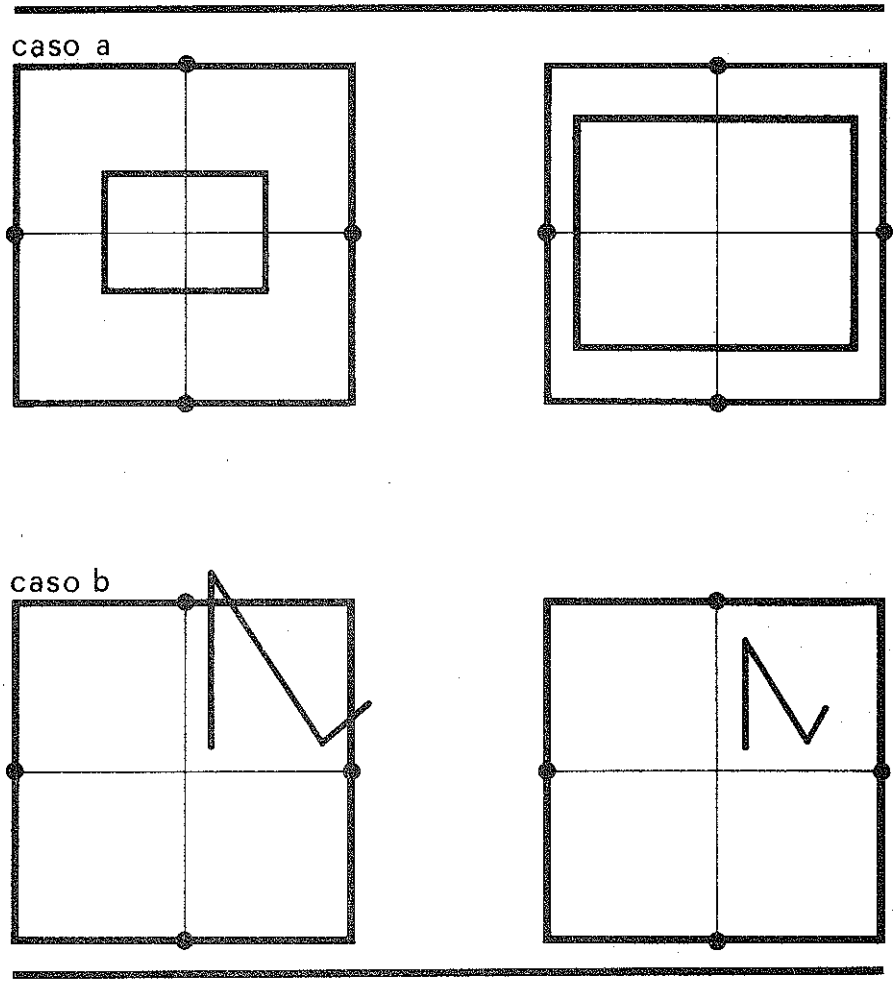


Figura 10-3 Cambiamento dell'unità di misura degli assi cartesiani

Come risulta in Figura 10-3 (caso a), modificando l'unità di misura degli assi cartesiani si possono ingrandire immagini, oppure in Figura 10-3 (caso b) riportare un'immagine entro i confini del FRAME dichiarato.

Le unità periferiche di tipo PLOTTER per il tracciamento di immagini o figure dispongono di un pennino. La stampante integrata non ha un pennino; la funzione equivalente viene svolta dal sistema, che va a marcare i bit opportuni nel buffer che rappresenta l'immagine. Il pennino (reale o virtuale, prima dell'invio del primo gruppo di coordinate) è posizionato al centro dell'area definita. Ogniqualvolta occorre tracciare un segmento, si calcola per via analitica l'equazione della retta passante per gli estremi del segmento e vengono marcati tutti i punti compresi fra tali estremi secondo lo schema riportato in Figura 10-4.

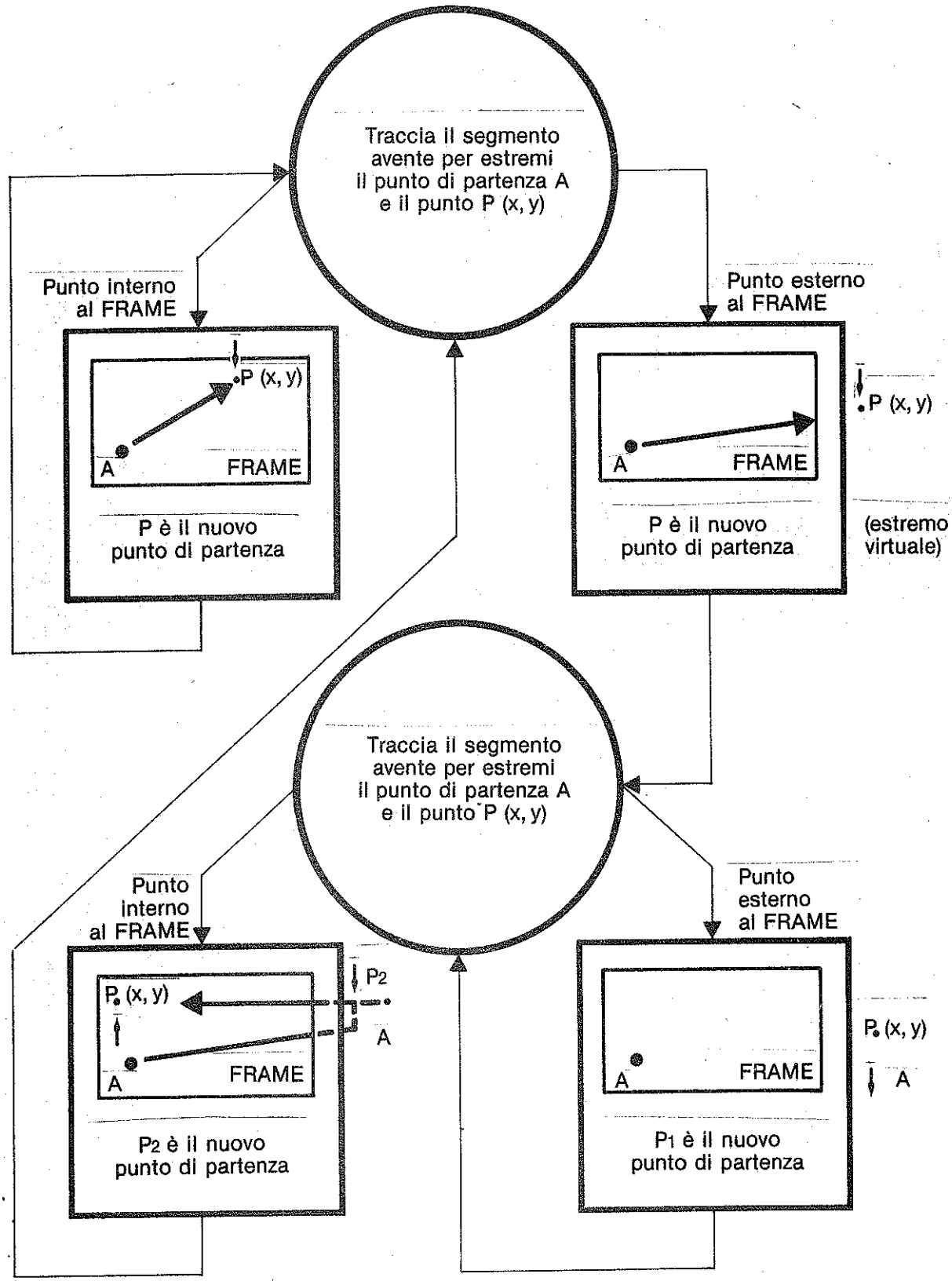
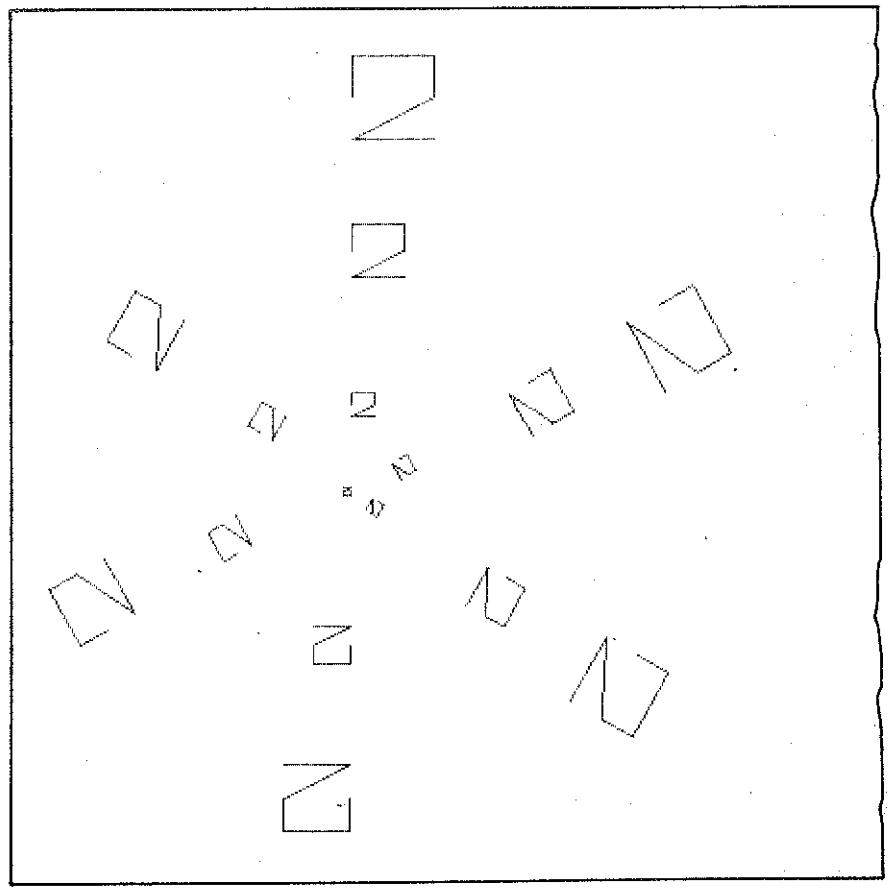


Figura 10-4 Traccia di un segmento

Se il punto di arrivo è interno all'area definita, il sistema sposta il pennino (che in Figura 10-4 è rappresentato da) fino a raggiungere il punto.

Viceversa, se il punto di arrivo è esterno all'area definita, il sistema sposta il pennino fino a raggiungere il bordo dell'area, e memorizza le coordinate del punto di arrivo. Se il nuovo punto di arrivo è interno all'area, il sistema esegue una interpolazione, posiziona il pennino sul primo punto interno all'area e sposta il pennino fino a raggiungere il punto di arrivo. Se il nuovo punto è ancora esterno, il sistema memorizza le nuove coordinate del punto e non fa eseguire alcuno spostamento al pennino.

Oltre a segmenti, possono anche essere rappresentati caratteri alfabetici, numerici e alcuni caratteri speciali. Il carattere è visto come un insieme di segmenti su una matrice di 3 x 3 punti. La sua rappresentazione può essere ruotata rispetto all'asse delle ascisse di un angolo la cui ampiezza è definibile da programma tramite l'istruzione CSIZE (vedere Opzione Plotter P6060 - GR Code 3973700 R (1)).



Stampa dell'immagine

L'esecuzione di un'istruzione DRAW causa la registrazione sul file dell'ultima immagine parziale e della area di lavoro.

Il buffer viene poi utilizzato per realizzare l'unione (unione logica di tipo OR) di tutte le immagini parziali descritte su disco.

Nel caso illustrato in figura 10-5, sul disco sono state registrate 3 immagini parziali: il buffer viene perciò diviso in 3 parti.

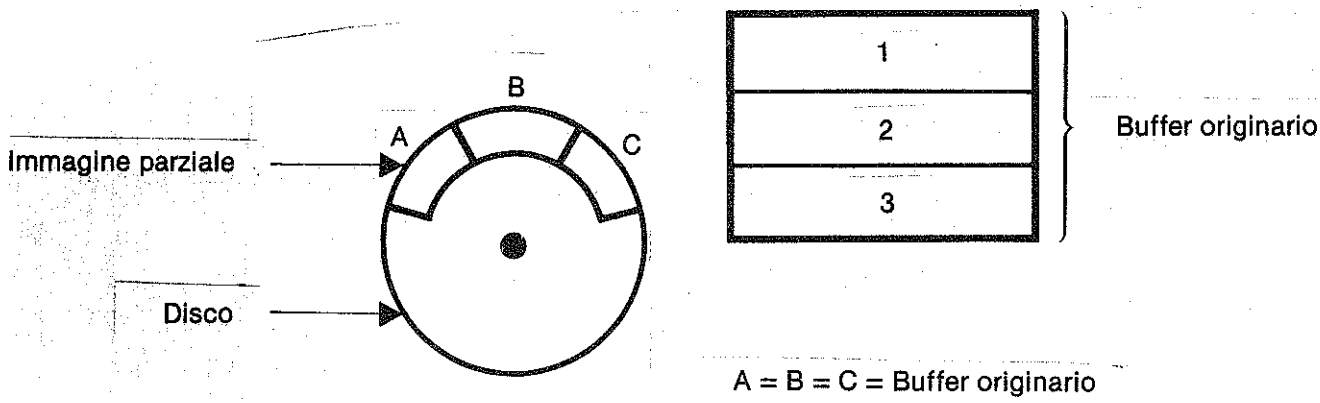


Figura 10-5 Registrazione di immagini parziali sul buffer

Su ognuno di questi buffer di transito viene letta da disco la prima parte della corrispondente immagine parziale (vedi figura 10-6).

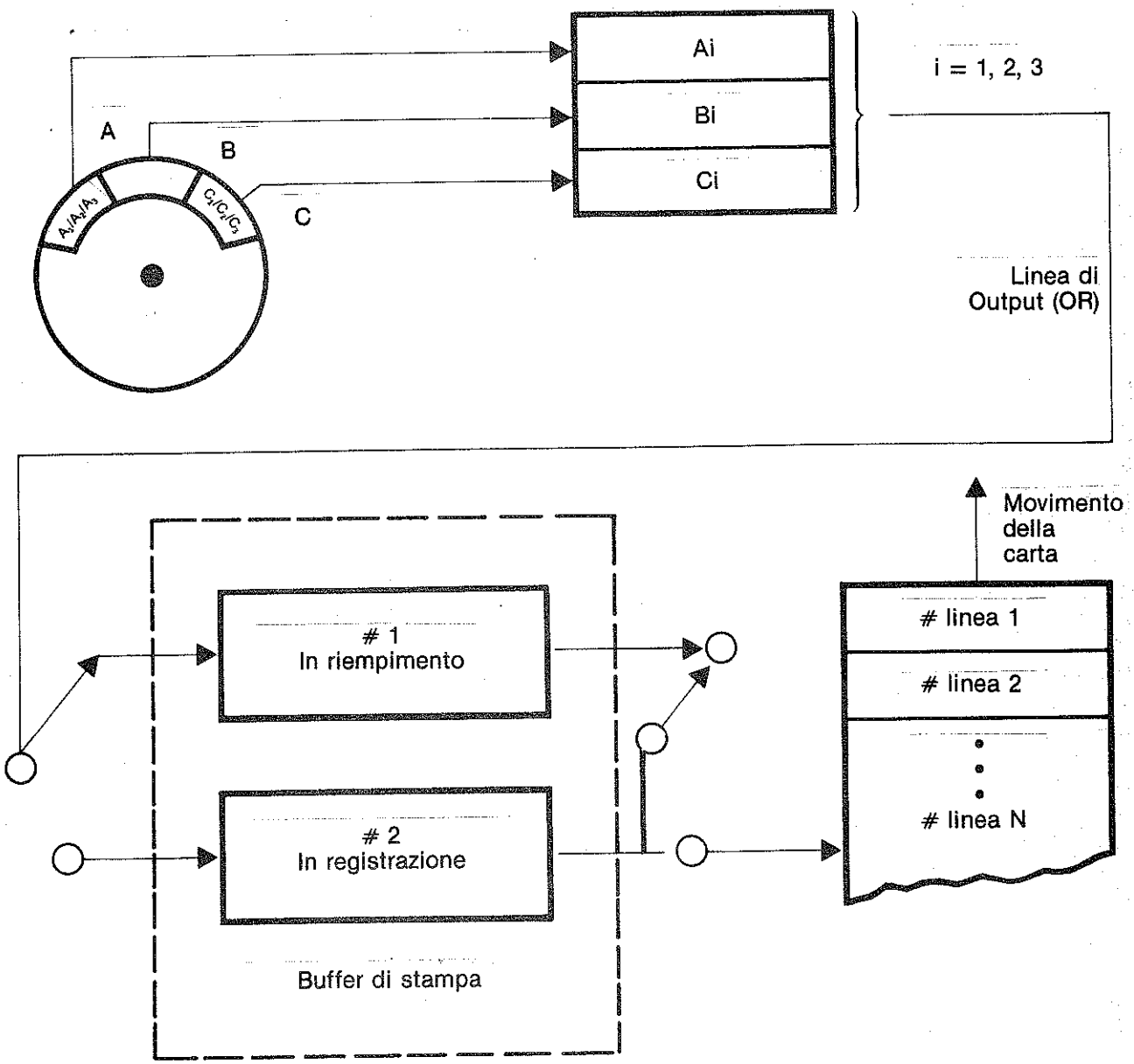


Figura 10-6 Operazione di OR

Viene quindi iniziata l'operazione di OR tra i byte che costituiscono le descrizioni A1, B1, C1. Quando una di queste è esaurita (date le caratteristiche di compattazione, ciò non avviene in generale contemporaneamente) viene rimpiazzata dalla successiva.

Quando si è sviluppato l'OR per una riga di stampa, l'output viene trasferito sul buffer di stampa e la stampa viene lanciata. Vengono utilizzati 2 buffer di stampa che sono allocati sullo stack al fine di consentire la sovrapposizione temporale tra l'ope-

razione di OR e quella di stampa. Nel caso in cui lo spazio di memoria non sia sufficiente, il sistema emette una segnalazione di errore.

Dimensionamento di Buffer e File

Daremo in questa sede alcune indicazioni su due argomenti di notevole interesse operativo:

- determinazione delle dimensioni ottimali del buffer in memoria e del file su disco in dipendenza della immagine
- costruzione dell'immagine in modo da consentire la minima occupazione di spazio su disco e la massima velocità di esecuzione

L'occupazione sul file è influenzata:

1. Dalle dimensioni del FRAME.
2. Dal numero delle immagini parziali che occorre memorizzare, e perciò dal rapporto nero/bianco della immagine complessiva.
3. Dalla disposizione dei punti sull'immagine. Infatti tra due figure formate ambedue dallo stesso numero di punti, occupa mediamente più spazio quella avente i punti uniformemente distribuiti e meno quella avente i punti in un'area ridotta dell'immagine.

E' possibile dare le seguenti relazioni orientative sul dimensionamento del file:

1. FRAME * 112 = MAX. D.F. (massima dimensione del file)

   in cui:

   FRAME ⟶ dimensioni dell'area dell'immagine in cm^2

   112 ⟶ N° di byte occorrenti per rappresentare un centimetro quadrato di superficie dell'immagine

2. MAX. D.F. * FA = DIM.R.F. (dimensioni reali del file)

   in cui:

   FA ⟶ fattore di annerimento $\frac{\text{N° punti neri}}{\text{N° punti bianchi}}$

L'estensione del buffer può essere al più pari alla estensione del file di output; in questo caso l'elaborazione non dà luogo all'archiviazione di immagini parziali. Tuttavia, dimensionando il buffer, si deve di

norma tener conto dei vincoli di spazio in memoria; il buffer di norma dovrà risultare una frazione intera del file.

Nello stabilire le sue dimensioni si tenga conto che:

Maggiori dimensioni del buffer implicano	
Vantaggi	Svantaggi
- Minori registrazioni sul file	- Grande occupazione di memoria
- Minor tempo per la stampa finale	- Minore velocità nel marcare i punti

Minori dimensioni del buffer implicano	
Vantaggi	Svantaggi
- Maggiore velocità nel marcare i punti	- Maggiori registrazioni sul file
- Piccola occupazione di memoria	- Maggior tempo per la stampa finale

Il dimensionamento del buffer dovrebbe essere tale che, detto K il numero delle immagini parziali generate ($K \leqslant 8$), il tempo di stampa risulti invariato, e il tempo di registrazione delle K immagini parziali risulti inferiore al guadagno di tempo nella generazione dell'immagine. Risulta perciò:

DIM.R.F/K = DIM.BUF. (dimensioni del buffer)

Nella messa a punto di un programma, può essere sufficiente registrare l'immagine solo sul buffer in memoria; in tal caso l'immagine coincide con l'immagine parziale, ed al termine dell'esecuzione o dopo una istruzione DRAW non è più disponibile.

Se c'è la necessità di riprendere una immagine generata da un programma già precedentemente eseguito, si può rendere nuovamente operabile l'immagine archiviandola per mezzo dell'istruzione LDIMAGE, senza che per questo occorra rieseguire l'intero programma.

Uso di PLOTTER esterni

Per i programmi rivolti a PLOTTER esterni, l'organo che realizza il collegamento tra le varie istruzioni di PLOTTER e il driver dell'unità PLOTTER (funzione P), è la funzione di sistema (BRTS). I parametri dell'istruzione di PLOTTER diventano gli argomenti di chiamata della funzione P, a cui vengono passati dalla funzione di sistema, il controllo passa poi alla funzione P, che alla fine lo restituisce all'istruzione del programma successiva a quella eseguita.

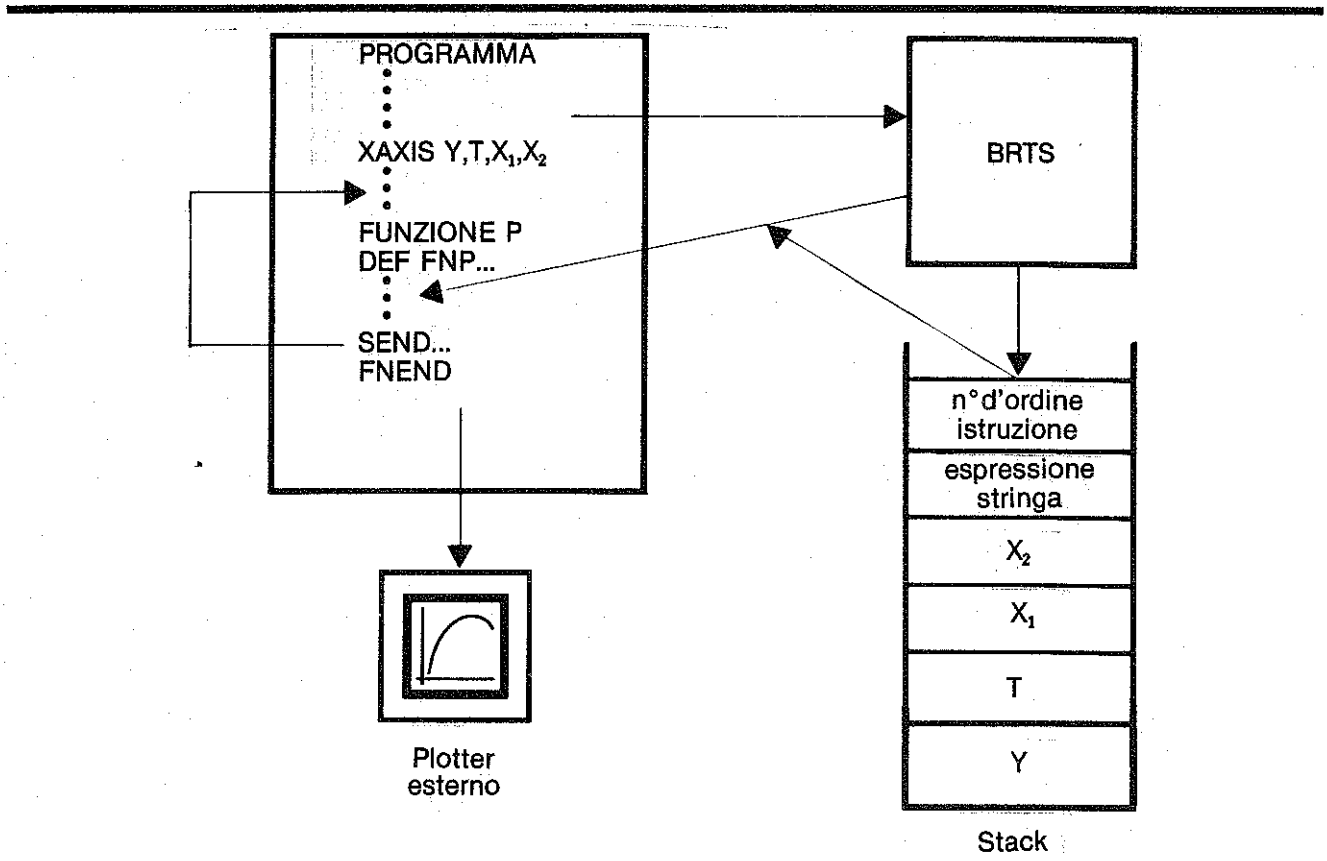


Figura 10-7 Uso dei PLOTTER esterni

La funzione P è di tipo numerico: ed ha 6 argomenti dei quali:

- i primi 4 rappresentano i possibili parametri numerici delle istruzioni (x_1, x_2, x_3, x_4)
- il quinto rappresenta il valore dell espressione stringa presente nella istruzione CPLOT (per tutte le altre istruzioni, questo argomento è inizializzato con il valore "stringa nulla")
- il sesto rappresenta il numero d'ordine attribuito al tipo di istruzione PLOTTER che deve essere ...-

guita. In base a questo valore è possibile, nell'ambito della funzione P, individuare univocamente il tipo di istruzione, e interpretare in modo appropriato gli argomenti di chiamata

L'immagine viene tracciata man mano che le singole istruzioni vengono eseguite, e non stampata dopo che è stata costruita. Le istruzioni INIMAGE, LDIMAGE e DRAW non hanno in questo caso alcun effetto.

11. NOTE DI PROGRAMMAZIONE

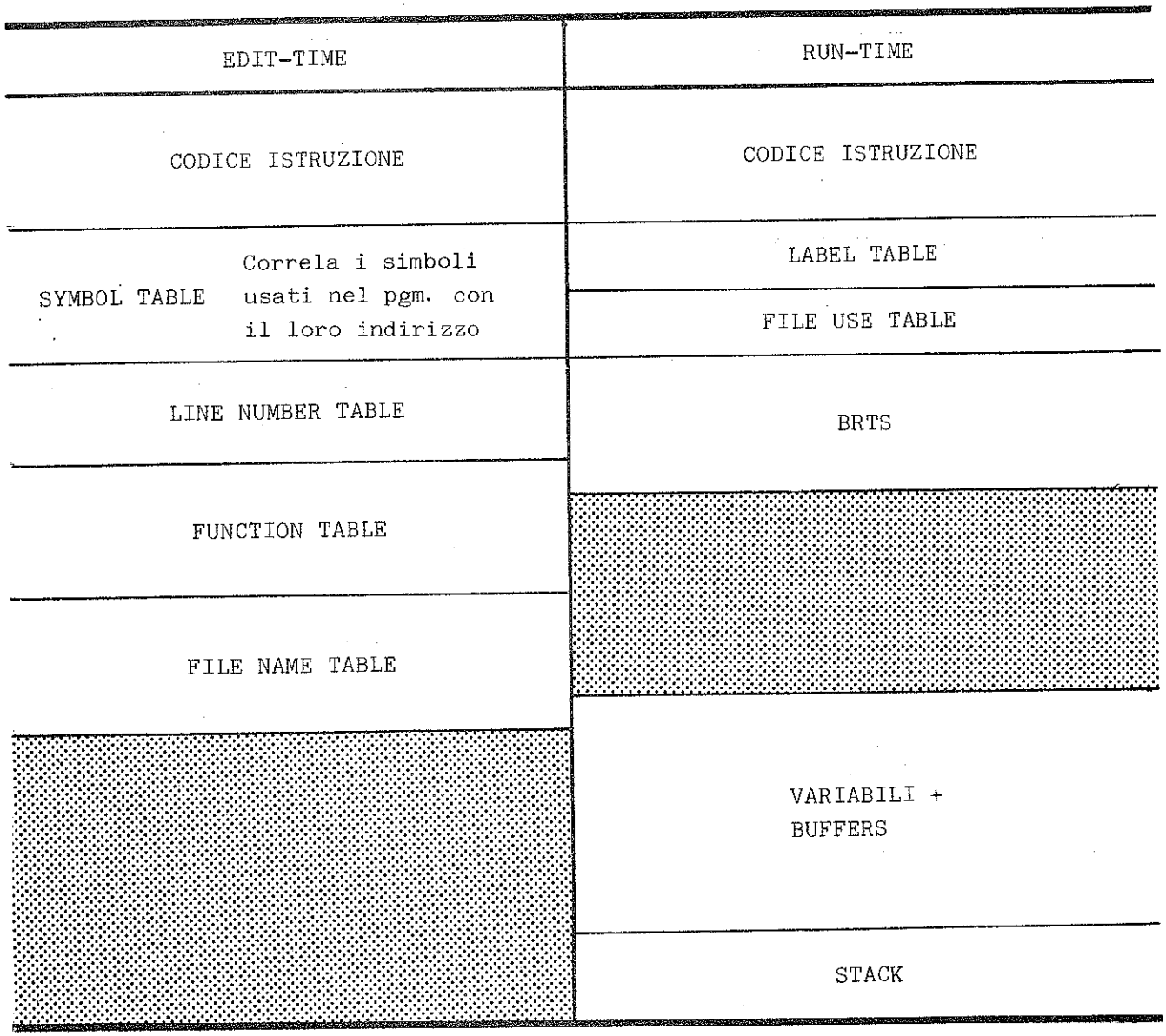
Generalità

Dai capitoli precedenti può essere desunta una serie di informazioni che riguardano l'occupazione della memoria interna durante le varie fasi operative, nonchè le modalità con cui avvengono la preesecuzione e l'esecuzione dei programmi. Tali informazioni risultano utili per ottenere un miglioramento delle tecniche di programmazione mediante l'uso di alcuni accorgimenti che contribuiscono a ridurre l'occupazione di memoria e ad abbreviare i tempi di esecuzione. In questo capitolo vengono schematizzate sia le informazioni riguardanti l'occupazione di memoria sia gli accorgimenti utilizzabili nell'implementazione dei programmi.

Nella tabella seguente è indicata l'occupazione di memoria del software residente e delle opzioni che possono essere richieste dall'utente mediante il comando OPTION.

COMAREA	308 496
MONITOR	1360 1400
PIH	464 500
PIOCS	1350 1650
DRIVER COMPILATORE	836 836
EXPRA	2744 2744
opzioni	
STR	2K BYTE
MAT	1.5K BYTE
PLO	2K BYTE

La somma dell'occupazione del software residente e del software opzionale prescelto fornisce l'occupazione statica totale di memoria interna da parte del sistema operativo. La composizione del codice in memoria utente negli stati "Creazione ed edit" ed "Esecuzione" è indicata nello schema seguente.



Diamo ora il significato e i criteri di valutazione dell'estensione dei singoli componenti.

Codice istruzioni

E' il codice direttamente ricavato dalla compilazione delle istruzioni che compongono il programma utente. La sua estensione di byte è da considerarsi pari a circa l'80% dell'estensione (in numero di caratteri) del sorgente. Il codice istruzioni rimane inalterato nei due stati.

Stato Editing

Symbol Tables: Tabelle simboli.

Line number Table: Tabella dei numeri di linea. Occupazione: 7 byte per linea.

Function Table: Tabella associata alle funzioni definite dall'utente (tipo FNx). Occupazione: 7 byte per funzione.

File name Table: Contiene i nomi dei file di tipo 'dati' riferiti nel programma. Occupazione: 7 byte per file.

Stato Running

Label Table: Gli elementi di questa tabella sono associati alle linee di programma riferite. Occupazione: 3 byte per linea riferita.

File use Table: Contiene per ogni file l'indirizzo di disco del relativo "file header". Occupazione: 2 byte per file riferito.

BRTS (Basic Run Time Support): L'insieme delle routines che realizzano funzioni di I/O associate a istruzioni BASIC. Ciascuna routine, associata ad un certo tipo di statement, è utilizzata da tutti gli statement dello stesso tipo presenti nel programma.

NOME DELLA ROUTINE BRTS	OCCUPAZIONE IN BYTE	
	MIN.	MAX.
CONVERT STR. TO VECT.	252	260
CONVERT VECT. TO STR.	324	332
BUILD	340	348
build using	304	376
ASSIGN	364	396
BBUILD	316	380
BASSIGN	388	396
TRACE ON-OFF	76	84
DELAY	132	140
RKB	364	412
INPUT	660	696
READ DATA	484	492
PRINT	268	276
PRINT USING	340	348
DISP	348	356
DISP USING	348	396
RESTORE	204	212
FILE	456	464
READ EXTERNAL FILE	856	880
write external file	460	512
SCRATCH EXTERNAL FILE	144	152
APPEND EXTERNAL FILE	144	152
CHAIN EXTERNAL FILE	356	364
SETW EXTERNAL FILE	216	248
MAT I/O	1052	1052
BEEP	44	44
IPSO	2232	2240
WHERE	860	868
LDIMAGE	872	880
SCALE	84	92
CSIZE	180	188
OFFSET	132	144
MOVE DOT PLOT	380	388
XAXIS	84	92
YAXIS	84	92
CTAB	84	92
FRAME	100	108
CPLOT	76	84
IDOT IPLOT	380	388
DRAW	140	148
INIMAGE	400	408
EXTERNAL PLOTTER	704	712

Precisione

Le operazioni tra un operando in semplice precisione ed uno in doppia precisione sono in generale da evitare. Ogni operazione di conversione DOPPIA ⟷ SINGOLA dura infatti circa 200 μsec.

Così la somma: $S_1 = S_2 + D_2$

dove

- S_1 e S_2 sono in singola precisione
- D_2 è in doppia precisione

si svolge nel modo seguente:

1. S_2 viene convertita in doppia precisione.

2. Viene effettuato il calcolo in doppia precisione.

3. Il risultato viene convertito in singola precisione.

La durata dell'operazione di somma è di ~ 1 m sec: di questo tempo le conversioni occupano 400 μsec. E' dunque buona norma eseguire operazioni tra operandi con lo stesso grado di precisione; in questo caso alcune operazioni aritmetiche (ad esempio +, -) sono scarsamente influenzata dalla precisione degli operandi; altre (ad esempio *, /) sono estremamente più veloci se operate in singola precisione.

A proposito del prodotto, l'algoritmo di implementazione è tale che:

a * b e b * a

vengono eseguite in tempi differenti. Il prodotto avviene in maniera efficiente se il moltiplicando ha un numero di cifre significative inferiore al moltiplicatore.

Esempio:

MOLTIPLICANDO X MOLTIPLICATORE
(meno cifre) (più cifre)

Costanti e variabili

Come regola generale, nella stesura di un programma BASIC P6060, l'uso delle costanti dovrebbe essere ri-

dotto al minimo indispensabile. Usare variabili al posto delle costanti comporta i seguenti vantaggi e svantaggi.

VANTAGGI	CAUSE
Minore occupazione del codice	Le costanti numeriche vengono codificate in oggetto ogni volta che vengono riferite; la costante occupa $5 + \text{INTEGER}\left(\frac{\text{N°cifre}}{2}\right)$ Byte Una variabile occupa 1 byte per ogni riferimento (indirizzamento posizionato su 1 byte).
Maggiore velocità di esecuzione	Se la variabile che sostituisce la costante è dichiarata in singola precisione, è possibile effettuare i calcoli in singola precisione. Al contrario, una costante coinvolta in calcoli, è integralmente mossa nello stack (MOVE da memoria a memoria) e viene sempre elaborata in doppia precisione

SVANTAGGI	CAUSE
Minore leggibilità del programma	Usando opportuni accorgimenti (ad esempio, KØ=Ø, K1=1,ecc.) si può ridurre questo inconveniente, naturalmente il programma resta meno leggibile.
Minor numero di variabili a disposizione	

Ciclo FOR/NEXT

Per l'esecuzione dei cicli FOR/NEXT è bene tener conto di alcuni accorgimenti fondamentali:

- usare tutte le variabili in semplice precisione
- non usare lo STEP I di default

Esempio:

2.5 ms/LOOP

NO

```
DCL SINGLE
  •
  •
FOR I = 1 TO 10 STEP 1
  •
  •
  •
NEXT I
```

1.9 ms/LOOP

SI

```
DCL SINGLE
C1 = 1
D1 = 10
FOR I = C1 TO D1 STEP C1
  •
  •
  •
NEXT I
```

Infatti nello stack le costanti occupano non meno di 5 byte, mentre le variabili riferite, per nome, occupano soltanto uno o due byte. Utilizzando variabili in luogo di costanti, oltre a ridurre l'occupazione dinamica dello stack, si riduce il tempo necessario per l'allocazione e la rimozione dei dati. Il tempo di esecuzione delle due sezioni di codice esemplificate differisce di 6 msec.

Variabili multiple

Operare con elementi di variabili multiple (vettori o matrici) è più laborioso che non operare tra variabili semplici (vedi § sullo stack). Fatto pari a 1 il tempo di operazione su scalare, si hanno i seguenti tempi:

SCALARE	ELEMENTO DI VETTORE	ELEMENTO DI MATRICE
1	1Ø	2Ø

Quindi, ove possibile:

- usare più variabili scalari invece di un vettore

- usare più vettori invece di una matrice

- non usare all'interno del ciclo FOR/NEXT assegnazioni del tipo:

```
FOR I = C  TO C  STEP C
 .
 .
A(I)  = V (K)
 .
 .
 .
NEXT I
```

con K costante all'interno del loop, ma usare:

```
T = V(K)
FOR I = C  TO C  STEP C
 .
 .
A(I) = T
 .
 .
NEXT I
```

Salto

ON.....(x)..... GO TO.....

ON.....(x)..... GO SUB....

L'uscita più veloce è quella sequenziale (x=Ø). Esiste un rapporto 1/10 tra questa uscita ed una qualsiasi delle altre. Se nel flow logico di un programma esiste una strada preferenziale, questa deve pertanto essere collocata immediatamente dopo la istruzione ON X GO TO (GO SUB).

L'istruzione ON X GO TO è più veloce di una serie di IF....THEN.

Chiavi funzioni

Per quanto riguarda le chiavi funzioni:

- le stringhe associate ai tasti funzione mediante i-

struzioni BASIC FKEY # vengono memorizzate in ordine ascendente del numero di chiave, indipendentemente dall'ordine con cui sono state definite. Così se la definizione avviene mediante il seguente programma:

```
10 FKEY # I ABC
20 FKEY # 8,DEF
          .
          .
          .
          .
100 FKEY # 2,KLM
```

l'ordine di memorizzazione definito è:

FKEY # I, FKEY # 2,.....FKEY # 8.

conviene pertanto definire le chiavi in ordine sequenziale; in caso contrario, per ogni chiave definita al di fuori dell'ordine sequenziale viene effettuato uno shift multiplo di memoria.

BRTS

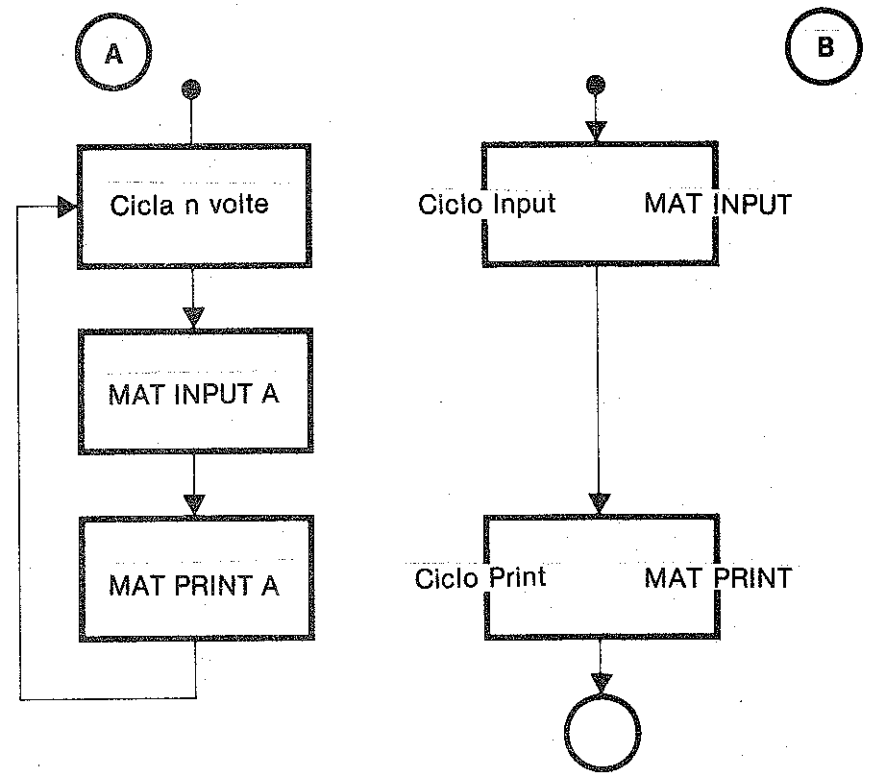
Le routine di Basic Run Time Supported richiamano in memoria un segmento in area di OVERLAY. Istante per istante in memoria può essere presente un solo segmento di OVERLAY. I moduli di OVERLAY non si accodano. La tecnica di programmazione deve tendere a minimizzare lo swap IN/OUT di questi moduli. E' bene perciò che l'utente eviti dei loop contenenti istruzioni appartenenti a segmenti diversi.

SEGMENTI DI OVERLAY

Segmento 1	BUILD USING
	INPUT
	PRINT
	PRINT USING
	DISP
	DISP USING
	READ
	WRITE
	SETW
	MAT READ
	MAT WRITE
	MAT PRINT
Segmento 2	RESTORE
	FILE

Segmento 3	SCRATCH FILE
Segmento 4	APPEND
Segmento 5	MAT INPUT MAT PRINT USING
Segmento 6	LDIMAGE
Segmento 7	SCALE
Segmento 8	CSIZE
Segmento 9	XAXIS
Segmento 10	YAXIS
Segmento 11	CTAB
Segmento 12	FRAME
Segmento 13	CPLOT
Segmento 14	DRAW
Segmento 15	INIMAGE

Una volta caricato in memoria un segmento di OVERLAY conviene usarlo il più possibile.



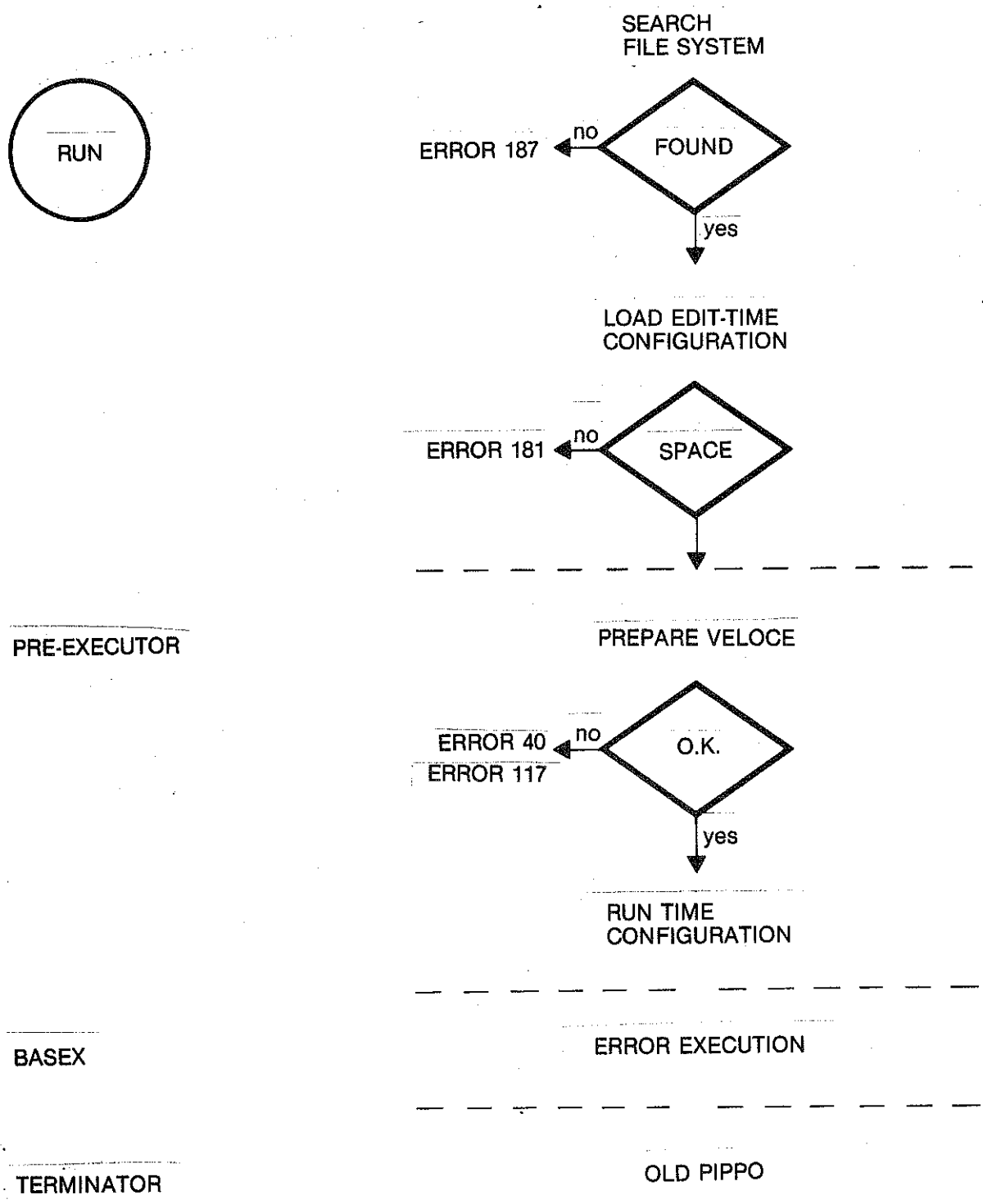
In figura:

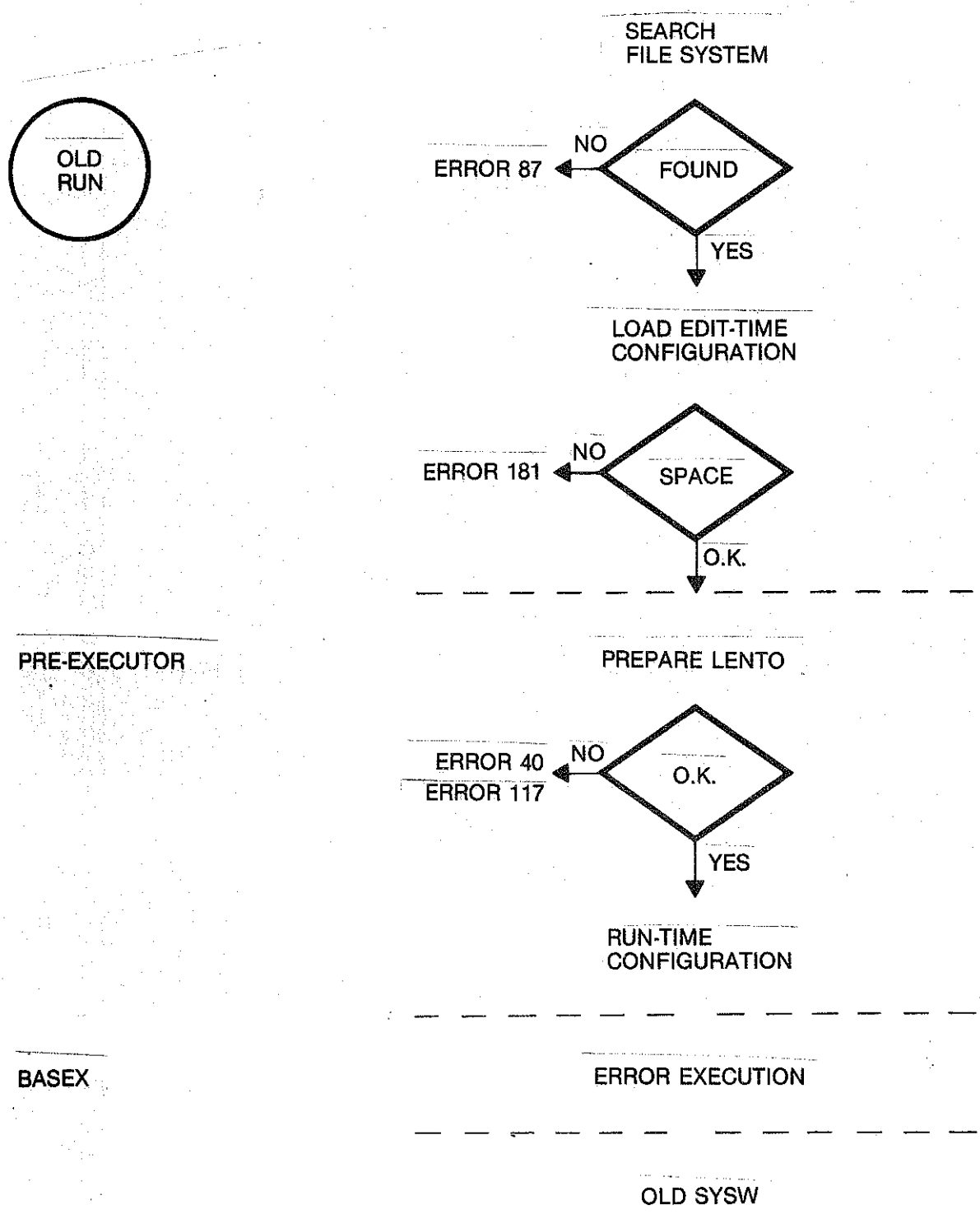
- nel caso A la programmazione è inefficiente, perchè in tutto vengono caricati in memoria 2*N segmenti di OVERLAY

- nal caso B è efficiente perchè in tutto vengono caricati in memoria 2 segmenti di OVERLAY.

Richiamo e lancio della esecuzione

L'esecuzione del programma risulta abbreviata se il programma è stato precedentemente salvato nella forma preeseguita e se viene richiamato mediante il comando RUN FILENAME. Così, nei due esempi successivi, risulta più efficiente la procedura seguita nel primo caso (a sinistra).


RUN
SEARCH
FILE SYSTEM
ERROR 187
no
FOUND
yes
LOAD EDIT-TIME
CONFIGURATION
ERROR 181
no
SPACE
PRE-EXECUTOR
PREPARE VELOCE
ERROR 40
ERROR 117
no
O.K.
yes
RUN TIME
CONFIGURATION
BASEX
ERROR EXECUTION
TERMINATOR
OLD PIPPO


SEARCH
FILE SYSTEM
OLD
RUN
NO
ERROR 87
FOUND
YES
LOAD EDIT-TIME
CONFIGURATION
NO
ERROR 181
SPACE
O.K.
PRE-EXECUTOR
PREPARE LENTO
ERROR 40
ERROR 117
NO
O.K.
YES
RUN-TIME
CONFIGURATION
BASEX
ERROR EXECUTION
OLD SYSW

Buffer

Ad ogni file dati aperto viene allocato un buffer, con un riferimento ad un record fisico.

Stack

L'occupazione dello stack non è statica ma varia istante per istante durante l'esecuzione. I valori qui riportati sono relativi ad un istante in cui si supponga di aver interrotto l'esecuzione del programma. La riduzione della occupazione dello stack consente di evitare l'errore di overflow memory in fase di preesecuzione (error 65); inoltre la velocità di esecuzione diminuisce al crescere della occupazione dello stack. Infatti lo stack è una area di memoria, la sua velocità di caricamento e svuotamento è relativamente bassa rispetto alla velocità di elaborazione della Unità Centrale.

Alcune occupazioni tipiche di Stack

1. SUBROUTINE (9 BYTE) 4 byte per indirizzo di rientro; 4 byte per pointer all'area locale precedente (lenv); 1 byte per identificatore

2. FUNCTION 9 byte come per la subroutine; 8 byte per il valore restituito; 8 byte per ogni variabile locale o per ogni argomento.

   Minimo 17 byte

3. FOR/NEXT (21 byte) 8 byte per valore superiore del loop; 8 byte per valore step; 4 byte per indirizzo di richiusura del loop; 1 byte per identificatore

4. BUILT-IN stessa occupazione delle Function + area locale della Built-in.

5. CHIAMATA BRTS stessa occupazione della Subroutine + lista argomenti + AREA LOCALE della BRTS.

6. BUILT-IN STRINGA particolarmente onerose ai fini della occupazione dello stack. L'argomento stringa viene infatti passato per valore (come tutti gli altri argomenti).

A. LE ISTRUZIONI ALP

Nelle seguenti tabelle vengonc descritte le istruzioni del linguaggio Algebrico ALP, l'elemento BASIC a loro corrispondente e il codice generato.

Elemento Linguaggio BASIC	Codice Generato
Variabile Numerica A = B	1 byte
Variabile Stringa A$ = B$	2 byte
Variabile Locale Numerica DEF FNA (A,B) C C = B+A	1 byte
Variabile Locale Stringa DEF FNA (A$ + B$) C$ C$ = B$+A$	2 byte
Array Numerico V(I,J) = F(A,B)	1 byte
Array Stringa V$(I,J) = F$(A,B)	2 byte

Indirizzo di Array Numerico MAT (A = B)	2 byte
Indirizzo di Array Stringa MAT READ A$	2 byte
Funzione Matriciale MAT A = CON (M,N)	2 byte
Costante Numerica A = 123	5 + INT $\frac{N}{2}$ byte N = numero dei caratteri
Costante Stringa A$ = "ABC"	(N+2) byte N = numero dei caratteri
Operatore di Somma A = A + B	1 byte
Operatore di Sottrazione A = A - B	1 byte
Operatore di Moltiplicazione A = A * B	1 byte
Operatore di DIvisione A = A / B	1 byte

Operatore di Assegnazione A = B	1 byte
Operatore di Elevamento a potenza A = 5 ↑ 5	1 byte
Operatore di Assegnazione Numerica Multipla A = B = C = 123	2 byte
Meno Unario A = - x	1 byte
Concatenamento A$ + B$	1 byte
Assegnazione di Stringa A$ + "ABC"	1 byte
Assegnazione di Stringa Multipla A$ = B$ = C$ = "ABC"	2 byte
Confronto Numerico su Condizione IF I ≷ 10 THEN	2 byte
Confronto fra Stringhe su Condizione IF A$ ≷ "ABC" THEN	2 byte
Carica Codice di Riempimento PAD A$, 32	2 byte

Estrai Codice di Riempimento DEPAD A$, 32	2 byte
Carica COdice di Riempimento Binario BPAD A$	1 byte
Salto Incondizionato GO TO ...	2 byte
Salto Calcolato ON A GO TO ...	N+3 byte N=numero degli offset
Salto a Subroutine GO SUB ...	2 byte
Salto Condizionato IF A=B THEN	2 byte
Rientro da Subroutine RETURN	1 byte
Chiamata di Funzione A = FNA (....)	2 byte
Chiamata di Funzione di Sistema (BRTS) PRINT DISP :	3 byte

Definizione di Funzione DEF FNA (X)	4 + 2N byte N = numero di variabili locali stringa
Rientro di Funzione FNEND	1 byte
Chiamata di Funzione BUILT-IN SIN ABS	2 byte
Salto Calcolato a Subroutine Subroutine ON X GOSUB	N + 3 byte N = numero de-offset
Inizio Loop FOR I = ... THEN	4 byte
Fine Loop Next I	2 byte
Salto a Executor 1 ogni linea BASIC	1 byte + 2 byte
Interruzione STOP	2 byte
Fine Programma END	2 byte

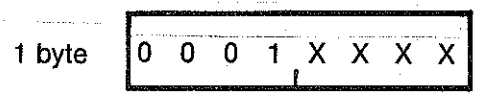
Chiave di Funzione FKEY =	3+N byte N = numero caratteri della stringa che vengono associati alla chiave funzione

B. ELEMENTI SULLO STACK

Di seguito sono raccolti i formati e le dimensioni delle aree allocate sullo stack in presenza dei vari elementi:

Variabile numerica:

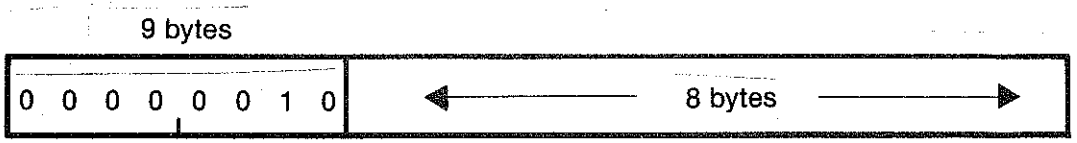
Variabile locale numerica:



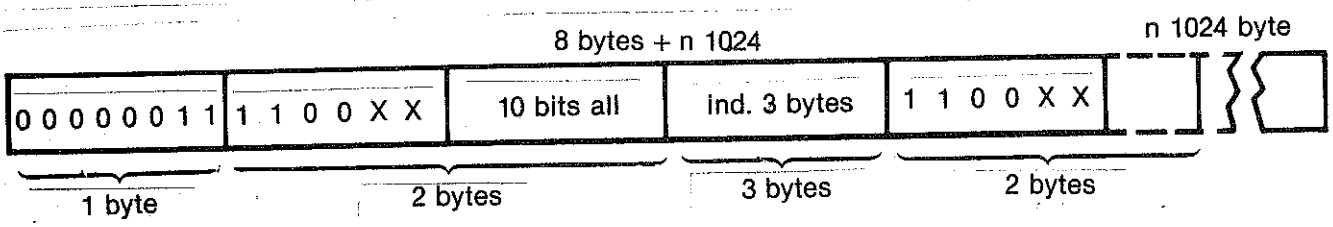
Variabile stringa:

Variabile locale stringa:

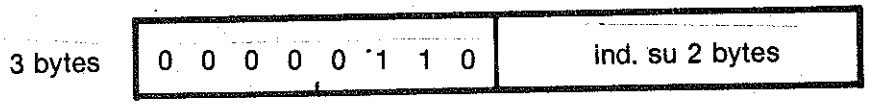
Temporaneo numerico:



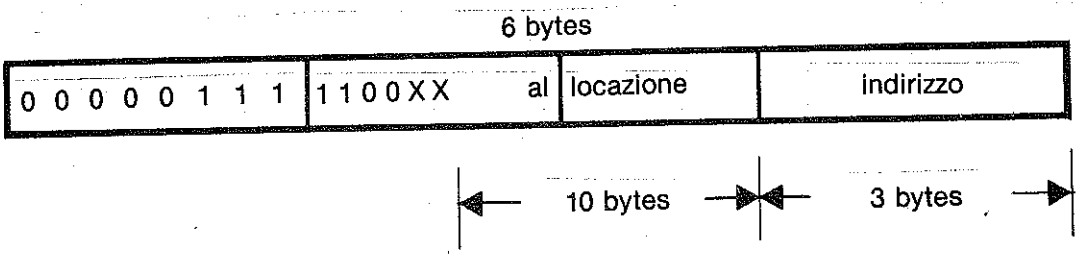
Temporaneo stringa:



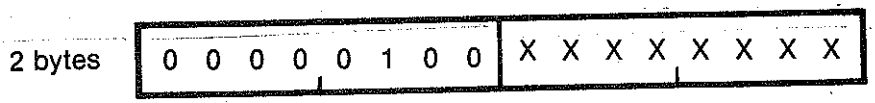
Indirizzo di memoria: (relativo ad elemento di array numerico)



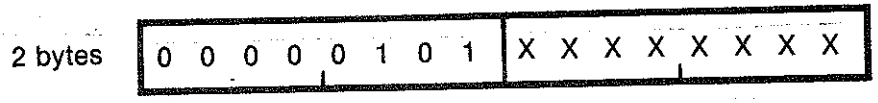
Indirizzo di memoria: (relativo ad elemento di array stringa)



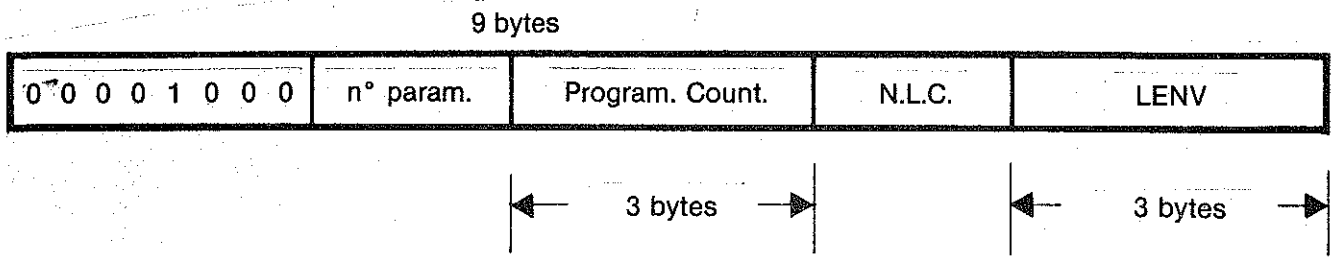
Nome di array numerico:



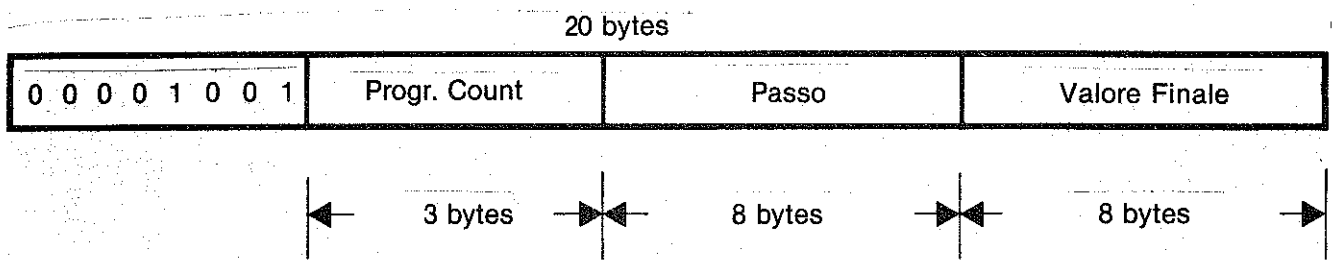
Nome di array stringa:



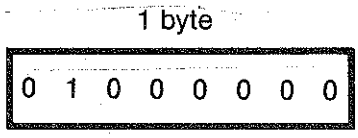
Contesto di subroutine e function:



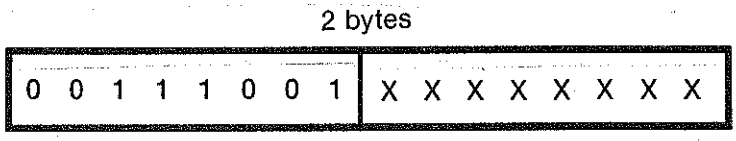
Contesto di loop:



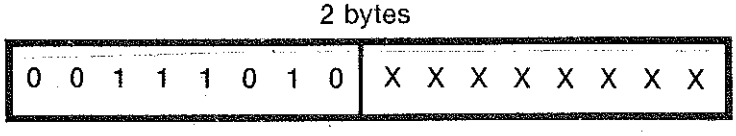
NOP di Stack: (serve ad ottenere gli indirizzi allineati)



Costante DATA: (residente in memoria e utilizzata dal S.O.)



Costante PLOTTER: (residente in memoria e utilizzata dal S.O. per operazioni di PLOTTER)



C. FORMATI DI RAPPRESENTAZIONE DEI DATI E CONVERSIONI

Generalità

Il sistema P6060 presenta vari formati di rappresentazione dei dati, a seconda che questi siano in fase di elaborazione in memoria centrale o su unità periferiche. Qualora si trovino su unità periferiche, anche in questo caso esistono varie possibilità di formato di rappresentazione, in funzione del tipo di periferica.

Di seguito useremo la seguente classificazione dei formati di dati:

- formato di memoria interno
- formato di memoria esterno
- formato ISO

e la seguente suddivisione, già nota, per le unità periferiche:

- periferiche gestite logicamente
- periferiche gestite fisicamente

Formato ISO

Si tratta della rappresentazione di dati secondo il codice ISO (vedi P6060 Manuale Generale). Le periferiche VIDEO/TASTIERA/DISPLAY esigono tale formato.

Formato interno

Si tratta del formato di rappresentazione in memoria delle variabili sia numeriche che stringa dotate di tutti gli attributi necessari alla loro elaborazione (vedi Capitolo 6).

Formato di memoria esterno

Si tratta del formato con cui è rappresentata su unità periferica l'immagine di memoria per tutti i tipi di dati, privati degli attributi necessari unicamente all'elaborazione (vedi Capitolo 7).

Periferiche gestite logicamente

In questa categoria (vedi Capitolo 2) sono raggruppate le periferiche video, tastiera, console, dischi, stampanti. Da un punto di vista dei formati dei dati, tali periferiche hanno la caratteristica di fornire tali formati in modo predefinito, cioè indipendentemente dalle applicazioni. Di conseguenza il sistema operativo può fornire all'utente un insieme di primitive di Input/Output che lo svincolano da ogni definizione e gestione di formati di dati.

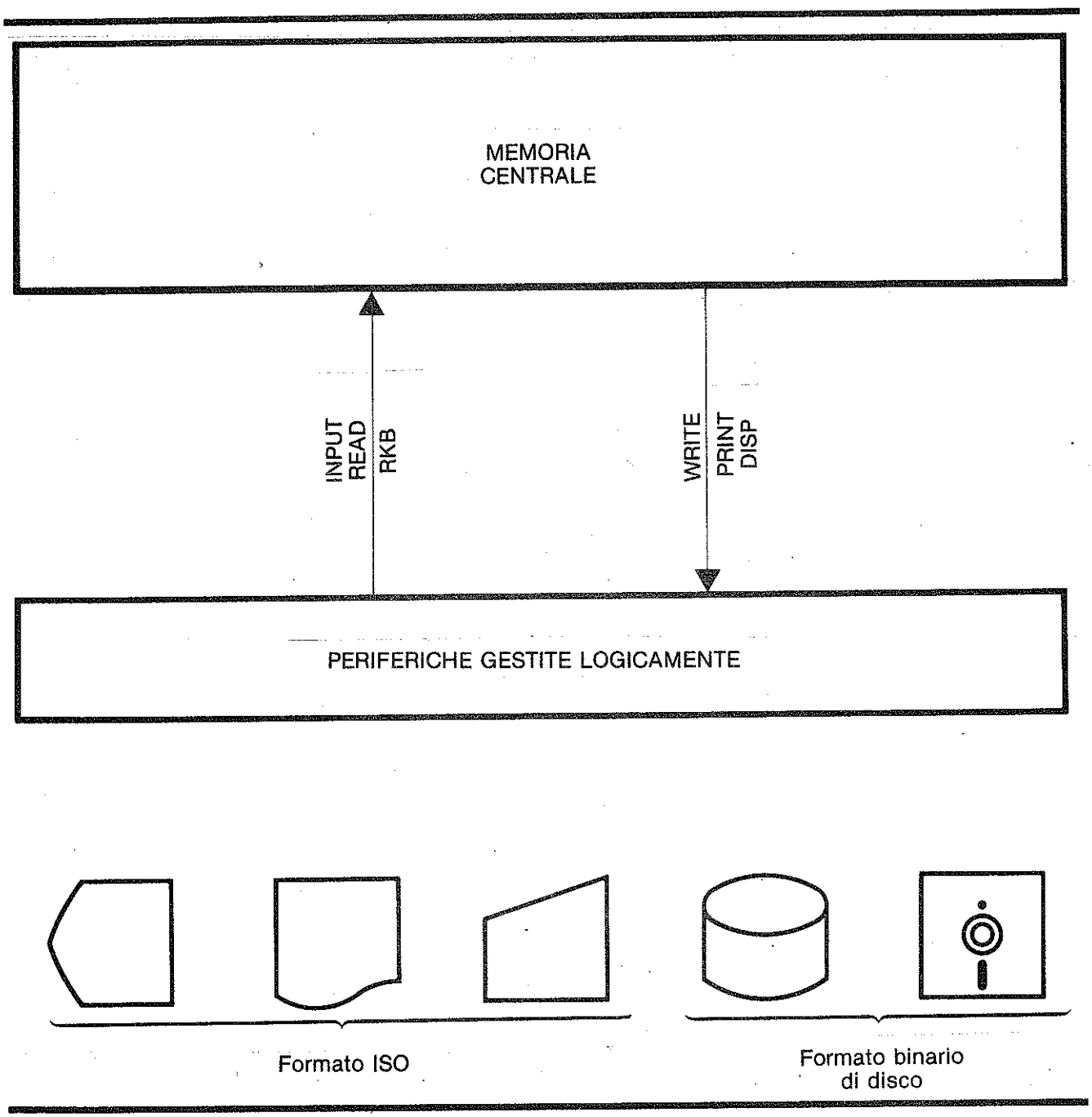


Figura C-1 Periferiche gestite logicamente

Periferiche gestite fisicamente

Le periferiche gestite fisicamente (vedi capitolo 2) sono collegate al sistema P6060 attraverso dispositivi di interfaccia, i canali. La gestione di queste periferiche viene effettuata per mezzo di istruzioni di I/O generale.

L'indirizzamento alle periferiche associate ai singoli canali deve essere effettuato esplicitamente, eventualmente, per mezzo di opportune istruzioni nei programmi applicativi. I canali collegabili al sistema P6060 sono:

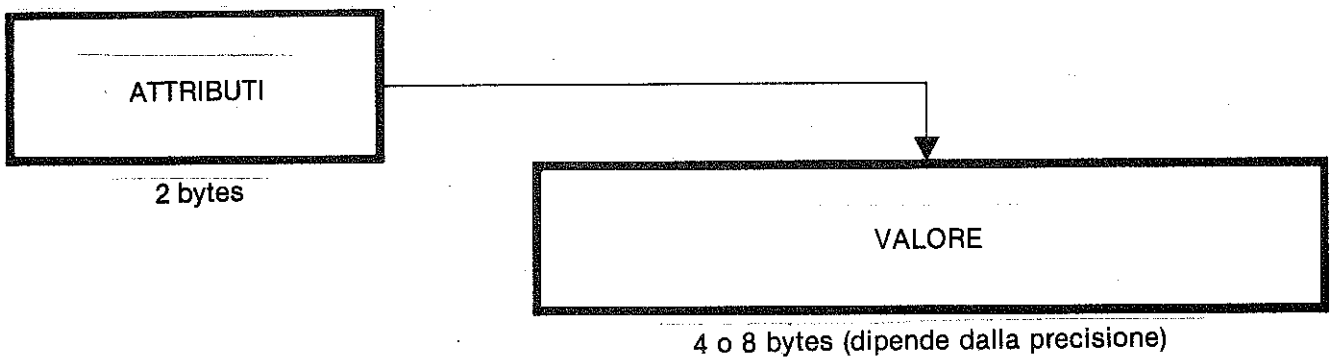
- interfaccia IPSO
- interfaccia EIA RS 232
- tastiera
- interfaccia IEEE 488-1975

In tal caso il formato dei dati è non più predefinito dal sistema operativo, ma dal programmatore, che quindi deve crearlo e gestirlo. Il sistema mette in tal caso a disposizione dell'utente primitive di invio e ricezione dei record (SEND, RECEIVE) (vedi P6060 I/O con periferiche esterne) e istruzioni per la creazione di variabili stringa destinate all'output (BUILD, BBUILD) e per l'acquisizione di dati di input (ASSIGN, BASSIGN).

Tutto questo è valido anche per le periferiche precedentemente definite come "gestite logicamente" qualora il progrommatore, per motivi di efficienza, intenda gestire tali periferiche in modo fisico utilizzando cioè le primitive appena ricordate.

Le istruzioni BUILD, BBUILD, ASSIGN, BASSIGN

La funzione di queste istruzioni è nota (vedi P6060 Manuale Generale). In questa sede è interessante rilevare l'azione di tali istruzioni sui formati delle variabili numeriche e stringhe. Di seguito saranno trattate soltanto le istruzioni BUILD e BBUILD, tralasciando le ASSIGN e la BASSIGN perfettamente simmetriche.



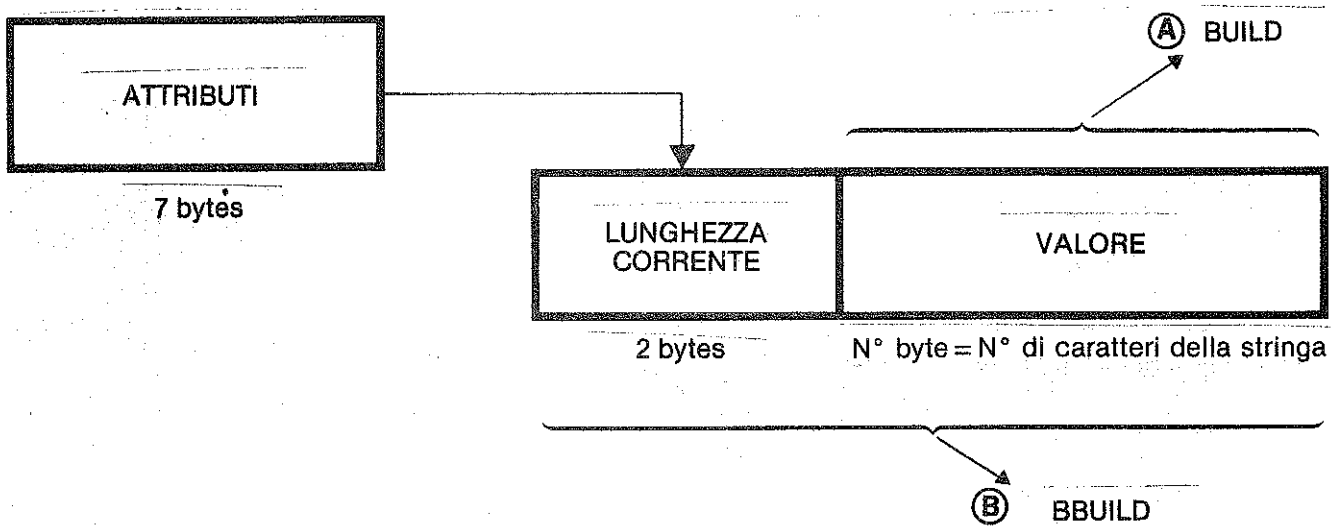
Sia la BUILD e la BBUILD intervengono solo sul campo "valore". La BUILD lo trasforma in una variabile stringa di caratteri in codice ISO. La BBUILD trasferisce il campo "valore" in una variabile stringa mantenendone lo stesso formato interno; in entrambi i casi la variabile stringa prodotta può essere trasferita su periferica attraverso un'istruzione SEND; nel primo caso le tipiche periferiche di output saranno:

- video
- stampante

nel secondo caso tutte le periferiche gestite fisicamente.

Variabili stringa

Le due istruzioni hanno un'azione diversa qualora si tratti di variabili stringa.



A: Viene convertito in ISO solo il valore della variabile stringa. Per il resto vale tutto quanto detto precedentemente.

B: Definisce una variabile stringa in formato interno, che comprende sia il valore che l'Attributo di Lunghezza Corrente. Questo fatto garantisce una migliore efficienza di gestione su memoria di massa di questo tipo di dati, in quanto viene evitato il continuo ricalco della lunghezza corrente. Su disco, la variabile stringa occupa quattro (4) byte più il numero di caratteri della stringa arrotondati alla parola. Il diverso comportamento di BUILD e BBUILD comporta che la sequenza di istruzioni

```
1) BUILD        2)BBUILD
   WRITE          SEND
   READ           RECEIVE
   ASSIGN         BASSIGN
```

applicate, ad esempio, allo stesso valore

123.4567895555 comporta:

- nel primo caso, l'ottenimento del valore 123.4567900000 (a causa delle normalizzazioni introdotte dalla conversione operata dalla istruzione BUILD, vedi P6060 Manuale Generale)

- nel secondo caso, il riottimento del valore iniziale, in quanto ogni istruzione di questa sequenza comporta solo il trasferimento della immagine di memoria verso e da unità periferica

Il delimitatore

Come è già stato detto, nella gestione fisica della periferica si ha un completo controllo del programmatore anche sulla struttura dei file di dati residenti sull'unità periferiche. In tale occasione l'uso del delimitatore ha la funzione di separatore dei campi che compongono i record trasmessi (o ricevuti) ad unità periferiche sotto forma di variabili stringa.

Convert

Fra le istruzioni che agiscono sul formato dei dati, un posto particolare è riservato alla CONVERT, che trasforma caratteri alfanumerici negli equivalenti codici numerici ISO, e viceversa. Tale istruzione ha lo scopo di poter trasformare dati in formato binario qualsiasi (ad esempio, provenienti da uno strumento di misura) in dati in formato ISO adatti ad essere visualizzati o stampati, e viceversa.

Printed in Italy