P6066

Ambiente di programmazione in Assembler
Guida del programmatore

Edizione preliminare

oliveti

GR Code 3978380 T (0)

PREFAZIONE

Questa pubblicazione è indirizzata a programmatori ed analisti di sistema che intendono creare moduli Assembler da richiamare in programmi BASIC. Prerequisiti necessari alla comprensione di questo manuale sono la conoscenza del linguaggio Assembler, del linguaggio BASIC e del manuale del sistema.

Riferimenti:

P6066 - Manuale generale
(GR code 3946860Y)

P6066 - Manuale del sistema
(LP code 39784605)

P6066 - Linguaggio Assembler
(GR code 3978440Z).

Distribuzione: Generale (G)

Edizione Preliminare: Marzo 1980

PUBBLICAZIONE EMESSA DA:

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

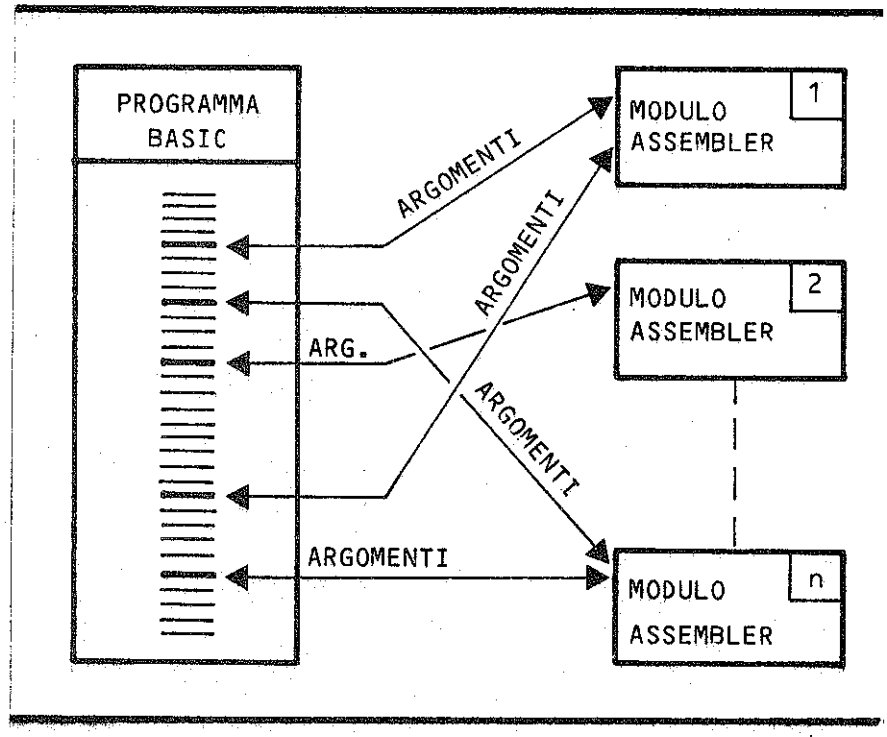
3978380 T

INDICE

INDICE DELLE FIGURE

1. INTRODUZIONE

Un programma BASIC che utilizzi moduli Assembler deve avere la seguente struttura:



ovvero:

- il programma BASIC deve poter trasferire ad un modulo Assembler alcuni argomenti per ottenerne un'elaborazione rapida;

- il programma BASIC deve poter utilizzare più volte lo stesso modulo come pure utilizzare più moduli distinti;

- al termine di ogni modulo utilizzato, il controllo ritorna al programma BASIC.

Un modulo Assembler, per poter essere utilizzato da un programma BASIC, deve avere le seguenti caratteristiche:

- poter prelevare gli argomenti forniti dal programma BASIC e, nel caso si tratti di argomenti numerici, convertirli in formato binario (vedi Appendice B) onde renderli elaborabili in ambiente Assembler (fig. 1-1).

- elaborare gli argomenti prelevati;

- poter ritornare al programma BASIC gli argomenti elaborati (previa loro conversione in formato memoria - vedi Appendice B - onde renderli utilizzabili in ambiente BASIC - fig. 1-2 - nel caso si tratti di argomenti numerici).

I soggetti specifici di questo manuale saranno pertanto:

- la struttura di un modulo Assembler;

- la creazione di un modulo Assembler;

- il richiamo di moduli Assembler da programma BASIC;

- la messa a punto (debugging) del programma BASIC completo di modulo Assembler.

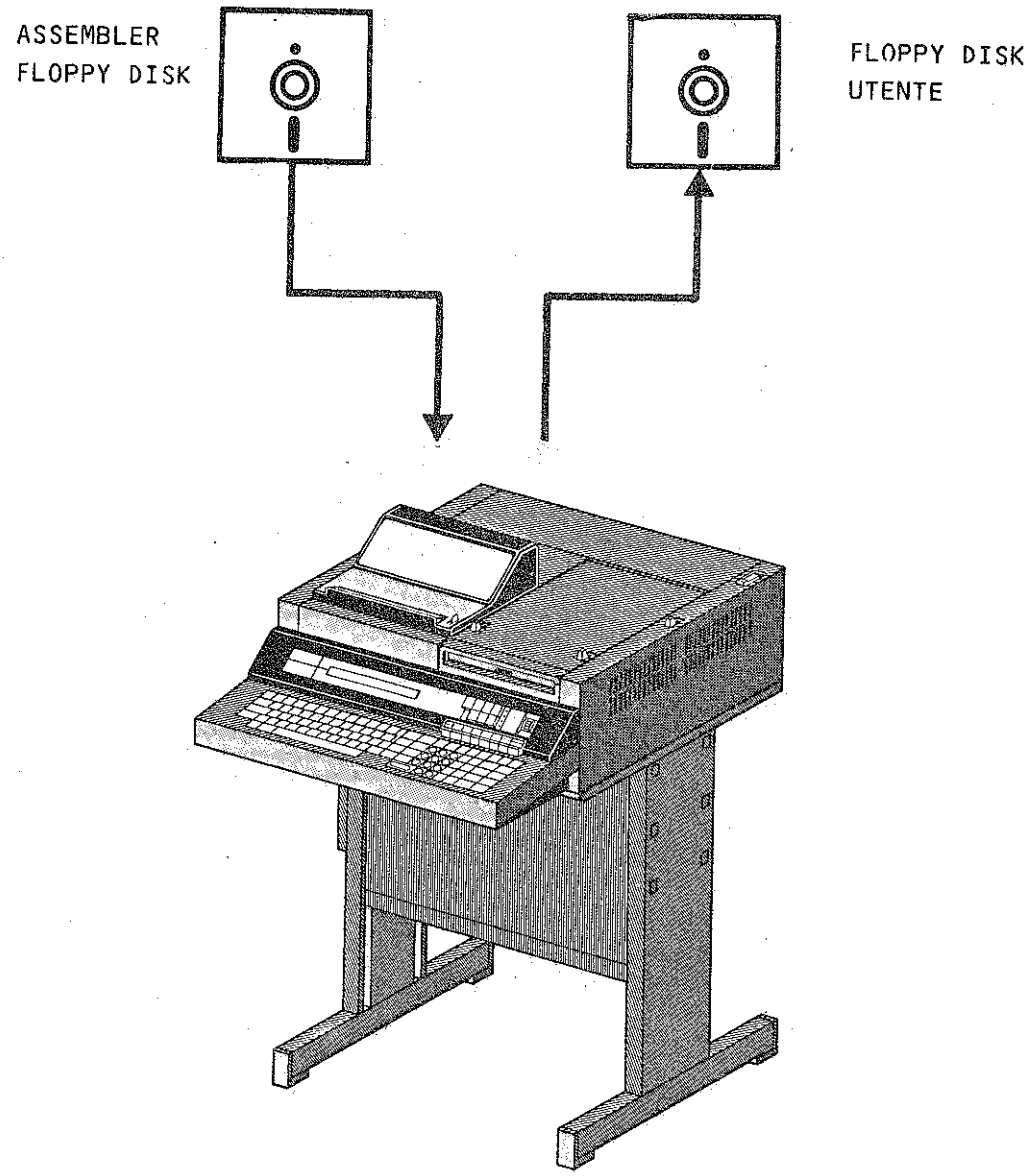
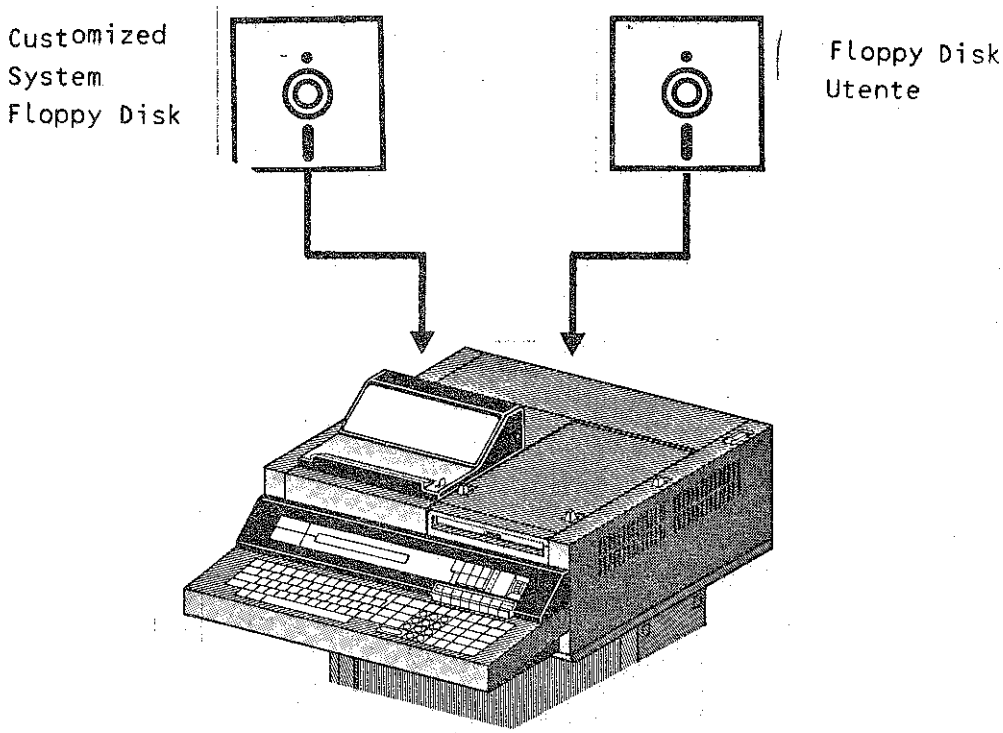


Fig. 1-1 Ambiente Assembler.

Si noti che il Floppy Disk Utente è necessario poichè in esso verranno registrati i moduli Assembler che saranno successivamente richiamati in ambiente BASIC.

a) Sistema a Floppy disk



b) Sistema a disco

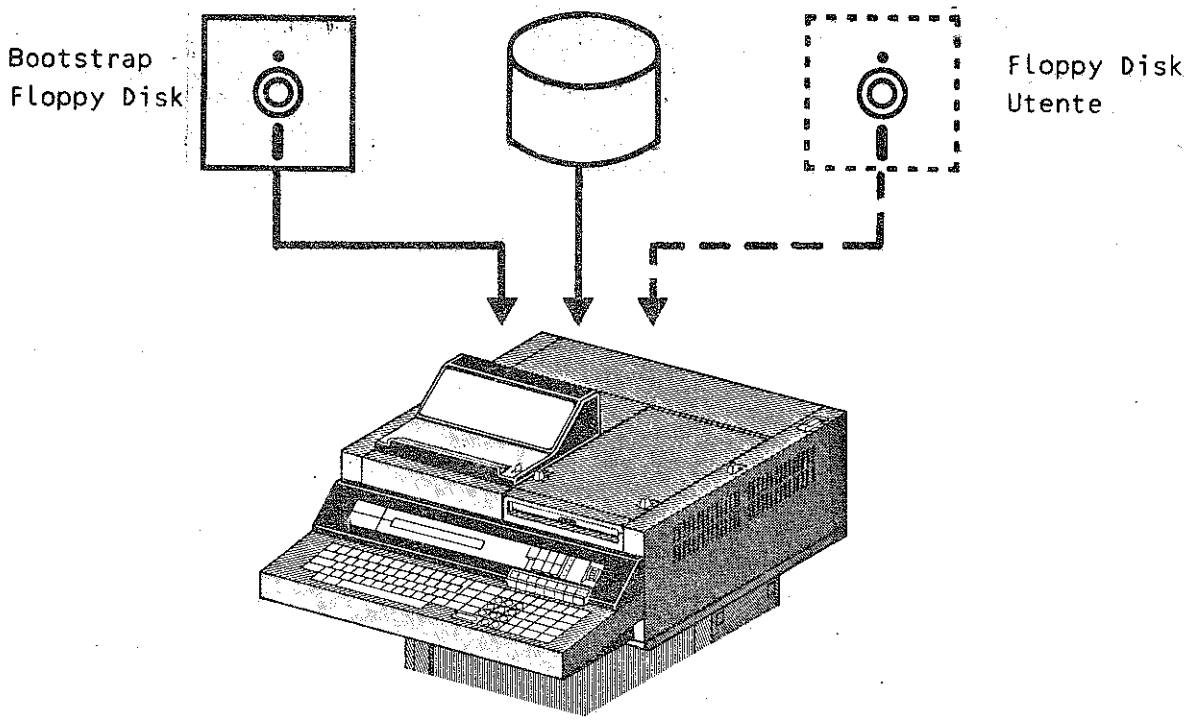


Fig. 1-2 Ambiente BASIC

2. STRUTTURA DI UN MODULO ASSEMBLER

Caratteristiche di un modulo Assembler

Un modulo in linguaggio Assembler può essere preparato (modulo sorgente) sia in ambiente Assembler che BASIC (figg. 1-1, 1-2); può essere creato (modulo oggetto) solo in ambiente Assembler e può essere messo a punto e gestito esclusivamente in ambiente BASIC.
L'impiego di un modulo Assembler da parte di un programma BASIC impone pertanto che la struttura del modulo rispetti alcune convenzioni particolari.
Di seguito verranno dapprima richiamati alcuni concetti sulla struttura di una frase Assembler, quindi le convenzioni per strutturare un modulo Assembler.

Struttura di una frase Assembler

Il formato di una frase Assembler è composto dai seguenti campi:

```
line-num [nome] operazione [operandi][commento]
```

dove:

line-num — è un numero compreso tra 1 e 9999 che indica la linea della frase (ovvero dell'istruzione Assembler);

nome — è una espressione alfanumerica, composta da 1 a 8 caratteri, che indica il nome simbolico della frase;

operazione — è un simbolo alfanumerico che indica il codice operativo di una delle istruzioni Assembler;

operandi — è una espressione alfanumerica che indica gli operandi della frase; questo campo può assumere formati diversi, dipendenti dal tipo di frase, per i quali si rimanda al manuale P6066 Linguaggio Assembler (GR code 3978440Z);

commento — è una espressione alfanumerica che contiene delle note alla frase; il numero di caratteri di questo campo è limitato dal solo fatto di poter essere contenuti negli 80 caratteri del buffer di riga.

Note

1. I campi di una frase Assembler devono essere separati tra di loro da almeno uno spazio.
   In assenza del campo nome, gli spazi tra line-num e operazione devono essere almeno due.

2. Se la frase Assembler è di tipo "commento" (caratterizzata cioè dai soli campi line-num e commento),il campo line-num deve essere seguito immediatamente da uno spazio e dal carattere *.

Struttura di un modulo sorgente Assembler

Le convenzioni per strutturare un modulo sorgente Assembler riguardano quanto segue:

- uso dei registri;
- prologo;
- prelievo/conversione degli argomenti;
- elaborazione degli argomenti;
- conversione/ritorno degli argomenti;
- epilogo;
- aree di dati.

Uso dei registri

I registri generali 11,12,13,14 (in seguito riferiti mnemonicamente come R11,R12,R13,R14) assumono il seguente significato particolare per il sistema:

- R11: indirizzo della ZRM;
- R12: indirizzo della COMAREA;
- R13: indirizzo dell'AREA LOCALE;
- R14: indirizzo di rientro al programma BASIC.

Il loro contenuto non deve pertanto essere modificato dal modulo Assembler. Se comunque, per esigenze di programmazione, si rendesse indispensabile l' impiego di tali registri, si provveda a ripristinarne il contenuto prima di tornare al programma BASIC.

Prologo

Il prologo del modulo sorgente è diverso a seconda che si desideri o meno l'allocazione dell'area locale. Per "area locale" si intende una porzione di stack disponibile al programmatore per il deposito temporaneo di dati durante l'esecuzione del modulo Assembler.
Prologo con allocazione di area locale:

```
START
PROC
```

Prologo senza allocazione di area locale:

```
START
PROC PROLOG=NO
```

Prelievo degli argomenti

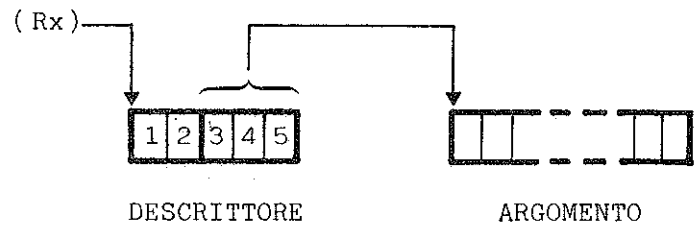
Gli argomenti, ovvero i dati numerici o alfanumerici che il programma BASIC fa elaborare attraverso il modulo Assembler (vedi Istruzione CALL, cap. 4), vengono prelevati dal modulo stesso attraverso la seguente istruzione:

```
GSA Rx,Ry
```

dove Rx e Ry indicano un qualunque registro generale. A seconda del valore numerico contenuto in Ry (valore che il programmatore deve previamente caricare in Ry), in Rx verrà automaticamente caricato l'indirizzo dell'argomento corrispondente:

- se l'argomento è numerico, l'indirizzo in Rx corrisponde realmente all'indirizzo di allocazione dell'argomento.

- se l'argomento è alfanumerico, l'indirizzo in Rx corrisponde all'indirizzo di un descrittore di 5 byte, i cui 3 ultimi byte contengono l' indirizzo di allocazione dell'argomento:



Per prelevare l'indirizzo di allocazione di un argomento alfanumerico è pertanto necessario incrementare di 2 il contenuto di Rx.

La corrispondenza tra valore di Ry e numero d' ordine dell'argomento si ricava dalla formula seguente:

$$Ry = N - i + 1$$

dove:

N = numero totale degli argomenti;
i = numero d'ordine dell'argomento da prelevare.

Se, ad esempio, si desidera prelevare il quarto (i=4) di 15 argomenti disponibili (N=15), in Ry si deve caricare il seguente valore:

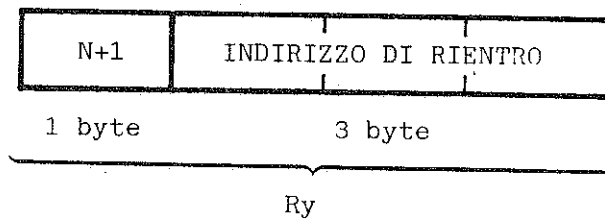
$$Ry = 15 - 4 + 1 = 12$$

Oltre agli argomenti che compaiono nell' istruzione CALL (capitolo 4),il programmatore può prelevare anche i seguenti parametri:

- indirizzo del nome del modulo Assembler, caricando il valore N+1 in Ry;

- numero totale degli argomenti (compreso il nome del modulo, quindi N+1) e indirizzo di rientro, caricando il valore Ø in Ry.

Il numero totale degli argomenti è contenuto nel byte più pesante; l'indirizzo di rientro nei restanti tre byte:



- argomenti diversi da quelli forniti dall'istruzione CALL. Per tale tipo di prelievo, si faccia riferimento alla Appendice C.

Conversione degli argomenti numerici

Se gli argomenti prelevati sono di tipo numerico e si desidera elaborarli, ad esempio, aritmeticamente, devono essere convertiti da formato memoria in formato binario per poter essere elaborati.
La conversione da formato memoria a binario si attua in due fasi distinte attraverso le seguenti istruzioni:

ISTRUZIONE	CONVERSIONE
CMST	Memoria ⟶ ISO
CSBH	ISO ⟶ Binario

Esempio: conversione del dato numerico + 20 (in doppia precisione).

FORMATO	STRUTTURA
MEMORIA (1)	B 2 | Ø Ø | Ø Ø | Ø Ø | Ø Ø | Ø Ø | Ø Ø | 8 1
	CMST ↓
ISO (2)	2 B | 3 2 | 3 Ø
	CSBH ↓
BINARIO (2)	Ø Ø 1 4

(1) vedi Appendice B
(2) vedi P6066 Linguaggio Assembler (GR code 3978440Z).

Elaborazione degli argomenti

Gli argomenti (eventualmente convertiti in formato binario, se numerici), vengono elaborati dalle istruzioni Assembler che costituiscono il modulo.

Conversione/ritorno degli argomenti al programma BASIC

Affinchè il programma BASIC possa riutilizzare gli argomenti elaborati dal modulo Assembler, può essere necessario (nel caso si tratti di argomenti numerici) che questi vengano convertiti in formato memoria.
La conversione da formato binario a formato memoria si attua in tre fasi distinte attraverso le seguenti istruzioni:

ISTRUZIONE	CONVERSIONE
CBS	Binario ⟶ ISO
CSD	ISO ⟶ DATA
CDM	DATA ⟶ Memoria

Esempio: conversione del dato numerico + 19 (in singo la precisione).

FORMATO	CONVERSIONE
BINARIO (2)	Ø Ø 1 3
	CBS
ISO (2)	2 B 3 1 3 9
	CSD
DATA (2)	Ø 2 8 Ø 8 Ø B 1 9 x
	CDM
MEMORIA (1)	1 9 Ø Ø Ø Ø C 1

(1) vedi Appendice B
(2) vedi P6066 Linguaggio Assembler (GR code 3978440Z).

Gli argomenti alfanumerici (stringhe) assumono automa ticamente il formato descritto nell'Appendice B.

Epilogo

L'epilogo contiene le istruzioni necessarie a rientra re nel programma BASIC, ovvero:

```
PRRET
ORG     *-2
BR      R14
```

Si noti il salto al registro R14 (BR R14) dove è con- tenuto l'indirizzo di rientro al programma BASIC (ve- di paragrafo Uso dei registri).
L'indirizzo di rientro di ogni chiamata a modulo As sembler viene caricato nel registro R14 all'atto del- la chiamata stessa.

Aree di dati

Come già premesso nel paragrafo "Prologo" è possibile allocare nello stack un'area locale, disponibile al programmatore per il deposito temporaneo di dati durante l'esecuzione del modulo Assembler.
L'area locale, in quanto non facente parte del programma, viene definita tramite una dummy section (vedi manuale P6066 Linguaggio Assembler - GR Code 3978440Z).

SEZIONI MODULO	ISTRUZIONI MODULO	
PROLOGO		START PROC PROLOG=NO
PRELIEVO E CONVERSIONE ARGOMENTI		BALR 3,0 USING *,3 LA 4,1 GSA 4,4 LA 5,2 GSA 5,5 LA 6,3 GSA 6,6 LA 0,6 MVI CAMPO,X'00' MVC CAMPO+1(6),CAMPO CMST CAMPO,0(6) CSBH HALF,CAMPO+1(6)
ELABORAZIONE ARGOMENTI		BC 4,ERROR TM HALF+1,X'01' BC 8,PARI LA 7,4 LH 6,HALF DM 7,6,RISUL B FINE
	PARI	MVC RESTO,X'0000'
		LH 9,HALF MLH 9,DUE STH 9,RISUL
	FINE	MVI CAMPO,X'00' MVC CAMPO+1(6),CAMPO
CONVERSIONE E RITORNO ARGOMENTI		CBS CAMPO+1,RISUL LA 0,6 CSD CAMPO2,CAMPO+1 CDM 0(5),CAMPO2 MVI CAMPO,X'00' MVC CAMPO+1(6),CAMPO CBS CAMPO+1,RESTO LA 0,6 CSD CAMPO2,CAMPO+1 CDM 0(4),CAMPO2
EPILOGO	ERROR	PRRET ORG *-2 BR R14
	DUE	DC H'2'
	R14	EQU 14
	CAMPO	DS CL7
	HALF	DS H
	RISUL	DS H
	RESTO	DS H
	CAMPO2	DS CL6

Fig. 2-1 Struttura di un modulo Assembler senza allocazione di area locale.

SEZIONI MODULO	ISTRUZIONI MODULO		
		START	
		PROC	
PRELIEVO E CONVERSIONE ARGOMENTI		BALR 3,0	
		USING *,3	
		USING LOCAL,13	
		LA 4,1	
		GSA 4,4	
		LA 5,2	
		GSA 5,5	
		LA 6,3	
		GSA 6,6	
		LA 0,6	
		MVI CAMPO,X'00'	
		MVC CAMPO+1(6),CAMPO	
		CMST CAMPO,0(6)	
		CSBH HALF,CAMPO+1(6)	
ELABORAZIONE ARGOMENTI		BC 4,ERROR	
		TM HALF+1,X'01'	
		BC 8,PARI	
		LA 7,4	
		LH 6,HALF	
		DM 7,6,RISUL	
		B FINE	
	RI	MVC RESTO,X'0000'	
		LH 9,HALF	
		MLH 9,DUE	
		STH 9,RISUL	
	NE	MVI CAMPO,X'00'	
		MVC CAMPO+1(6),CAMPO	
CONVERSIONE E RITORNO ARGOMENTI		CBS CAMPO+1,RISUL	
		LA 0,6	
		CSD CAMPO2,CAMPO+1	
		CDM 0(5),CAMPO2	
		MVI CAMPO,X'00'	
		MVC CAMPO+1(6),CAMPO	
		CBS CAMPO+1,RESTO	
		LA 0,6	
		CSD CAMPO2,CAMPO+1	
		CDM 0(4),CAMPO2	
EPILOGO	ERROR	PRRET	
		ORG *-2	
		BR R14	
	DUE	DC H'2'	
	R14	EQU 14	
DEFINIZIONE AREA LOCALE	LOCAL	DSECT DUMMY SECTION	
	OLD13	DS F	
	CAMPO	DS CL7	
	HALF	DS H	
	RISUL	DS H	
	RESTO	DS H	
	CAMPO2	DS CL6	
	ENDLOCAL	DS 0F	

Fig. 2-2 Struttura di un modulo Assembler con allocazione di area locale

3. CREAZIONE DI UN MODULO ASSEMBLER

La creazione di un modulo Assembler si articola in due fasi distinte:

- preparazione del modulo sorgente;
- preparazione del modulo oggetto.

Preparazione del modulo sorgente

Il modulo sorgente deve essere un file di tipo testo scritto in linguaggio Assembler.
Il modulo sorgente può anche risiedere su più file testo; in tal caso, esso conterrà solamente i riferimenti ai file testo che lo compongono e che non devono comunque superare il numero di 16.
Esempio:

```
1Ø   COPY  FILEØ   ┐
2Ø   COPY  FILE1   │
      :            │ max. 16 file
      :            │
      :            │
1ØØ  COPY  FILEF   ┘
```

dove FILEØ, FILE1 ... FILEF sono i nomi dei file testo che contengono le istruzioni Assembler del modulo sorgente.
La procedura per la preparazione del modulo sorgente è la seguente:

- accendere il sistema;
- inserire il floppy disk Assembler o BASIC ed attendere che il messaggio READY compaia sul display;
- introdurre il comando TEXT, digitando in tastiera le lettere TEX e premendo successivamente il tasto EOL;
- introdurre il modulo sorgente, frase per frase, rispettando il formato della frase stessa (vedi Struttura di una frase Assembler, capitolo 2);

- salvare eventualmente il modulo sorgente (si veda la sintassi del comando SAVE, capitolo 6).

Note

1. Nel caso di modulo sorgente composto da più file, i file componenti non devono mai contenere l'istruzione END di fine assemblaggio. Opzionalmente, la istruzione END può essere contenuta nell'ultimo file. L'opzionalità dell'istruzione END è valida anche per moduli sorgenti composti da un unico file.

2. La preparazione del file testo che sono componenti di un modulo sorgente richiede il salvataggio di tutti i file componenti, onde poterli richiamare durante l'assemblaggio.

3. Nel modulo sorgente non devono comparire richiami a simboli esterni (vedi manuale P6066 Linguaggio Assembler - GR code 3978440Z); è quindi vietato lo impiego dell'istruzione CALEXT.

4. Se si desidera il listing del modulo sorgente durante la sua preparazione, premere il tasto di console PRINT ALL che si illumina.

```
FILE     BIN

0010             START
0020             PROC
0030             BALR 3,0
0040             USING *,3
0050             USING LOCAL,13
0060             LA 4,1
0070             GSA 4,4
0080             LA 5,2
0090             GSA 5,5
0100             MVC ISO+1(3),2(5)
0110             MVI ISO,X'00'
0120             L 5,ISO
0130             ALRI 5,2
0140             LA 7,0
0150             LA 6,8
0160 LOOP        EQU *
0170             SLL  7,1
0180             CLI 0(5),X'30'
0190             BE AVANTI
0200             ALRI 7,1
0210 AVANTI      ALRI 5,1
0220             BCT 6,LOOP
0230             STH 7,HALF
0240             LA 0,0
0250             CBS  ISO(3),HALF
0260             CSD DATA,ISO
0270             CDM 0(4),DATA
0280             PRRET
0290             ORG *-2
0300             BR 14
0310 LOCAL       DSECT
0320 OLDR13      DS F
0330 HALF        DS  H
0340 ISO         DS CL4
0350 DATA        DS CL8
0360 ENDLOCAL    DS 0F

END OF LISTING
```

Fig. 3-1 Esempio di listing di un modulo sorgente.

Preparazione del modulo oggetto

Il modulo sorgente preparato secondo la procedura precedentemente descritta, viene trasformato in modulo oggetto rilocabile tramite il programma di utilità ASM.
Questo programma attiva l'assemblatore il quale, per svolgere le proprie funzioni, necessita temporaneamente di spazio su floppy disk.
Tale spazio, detto scratch file, è allocato, utilizzato e deallocato automaticamente dall'assemblatore:viene allocato su floppy disk sistema attribuendogli il nome @WORKF e viene deallocato al termine dell'assemblaggio.
Se, per qualche caso accidentale (errore fisico,power off), lo scratch file rimane su floppy disk, è bene operare in uno dei seguenti modi:

1) eseguire un successivo passo di assemblaggio che cancella automaticamente lo scratch file preesistente;

2) copiare la libreria, a cui lo scratch file appartiene, su di un altro floppy disk; in questa operazione lo scratch file spurio non viene trasferito e si evita, nelle successive operazioni, di trovarsi in presenza di spazio insufficiente su FDU.

Se non esiste spazio sufficiente ad allocare lo scratch file, il sistema segnala errore (vedi Appendice A) e l'utente deve indicare un'altra unità o creare spazio sul floppy disk in questione; lo spazio minimo richiesto per lo scratch file è di due settori.

Programma di utilità ASM

Il programma di utilità ASM traduce un modulo sorgente scritto in linguaggio Assembler in modulo oggetto rilocabile.

Formato

```
EXE [C] ASM,IN=filename1[,(,{FDU1|FDU2})],OUT={/|filename2[,(,{FDU1|FDU2})]},[LIST],
[XREF][,WSP=(,{FDU1|FDU2})]
```

dove:

IN=filename$_1$ — indica il file testo contenente il modulo sorgente da assemblare, oppure il file testo contenente i riferimenti ai file che contengono consecutivamente il modulo sorgente;

OUT=filename$_2$ — indica il file, di tipo oggetto rilocabile, su cui viene trasferito l'output dell'assemblatore;

OUT=/ — indica che l'output dell' assemblatore non viene registrato su floppy disk;

LIST — indica la richiesta del listing del modulo oggetto;

XREF — indica la richiesta delle cross references in output (vedi paragrafo Esempio di output dell'assemblatore);

WSP — indica la richiesta dello scratch file. Se non specificato, il file viene ugualmente riservato dall' assemblatore su floppy disk sistema;

FDU1/FDU2 — indica l'unità su cui risiede l'oggetto del relativo parametro (filename$_1$, filename$_2$, scratch file). Se non specificato, si opera su floppy disk sistema.

Gli operandi sopra descritti sono posizionali.

Azione

Il programma di utilità traduce un modulo scritto in linguaggio Assembler sorgente, residente quale file di tipo testo di nome filename1 su floppy disk, in un modulo oggetto rilocabile; l'output del programma di utilità è un file di tipo oggetto rilocabile di nome filename2.
Se, durante l'esecuzione del programma di utilità, si verificano situazioni di errore, il sistema segnala il messaggio relativo (vedi Appendice A) e passa nello stato comandi. Se l'assemblatore rileva nel programma sorgente degli errori, emette delle segnalazioni diagnostiche (vedi Appendice A).

Note

1. Nei seguenti casi:

   - se filename1 non esiste oppure non è di tipo testo;

   - se le direttive contenute in filename1 fanno riferimento ad un file che non esiste oppure che non è di tipo testo;

   - se il numero dei file riferiti nelle direttive di filename1 è maggiore di 16;

   il sistema interrompe l'esecuzione del programma di utilità ASM e segnala l'errore relativo.

2. Se filename2 esiste già, il sistema controlla se è di tipo oggetto rilocabile; in caso affermativo, dato che si presume essere un riassemblaggio dello stesso programma, il file viene rimpiazzato; in caso contrario viene segnalato errore (vedi Appendice A).

3. Il file di tipo oggetto prodotto dall'assemblatore deve essere allocato su un solo extent. Per controllare che questa esigenza venga rispettata, si utilizzi il comando CATALOG (capitolo 6) dopo l'assemblaggio.
   Nel caso risulti che il modulo oggetto risieda su più di un extent, procedere in uno dei seguenti modi:

   - eseguire il programma di utilità EXEC LIBCOPY (in ambiente BASIC), oppure:

- riassemblare il modulo sorgente su di un nuovo floppy disk.

4. L'operando WSP si utilizza per forzare l'allocazione dello scratch file su un floppy disk desiderato.

5. La fase di assemblaggio può essere interrotta in qualunque istante premendo il tasto BREAK .

Esempio di output dello assemblatore

L'esempio seguente illustra come viene documentata dall'assemblatore l'esecuzione del programma di utilità ASM. Il modulo sorgente riferito nell'esempio è il file testo di fig. 3-3.
L'assemblatore documenta sempre in stampa le seguenti informazioni:

Opzioni

```
                                                    PAGE 1

                *** ASSEMBLY OPTIONS ***        DATE : 79/12/14

  LIST =YES
  XREF =YES
  OUT  = BIN1    ( FDU1 )
```

dove:

LIST indica se è stato richiesto (YES) o no (NO) il listing del modulo oggetto;

XREF indica se è stato richiesto (YES) o no (NO) il listing delle cross references;

OUT indica il nome del modulo oggetto e l'unità (FDU1/FDU2) destinata a riceverlo.

```
                                                        PAGE 2

                    *** ASSEMBLY OPTIONS ***        DATE : 79/12/14

SOURCE FILE NAME            FROM STM    TO STM

    BIN                         1          36
```

dove:

SOURCE FILE NAME — indica il nome del modulo sorgente o i nomi dei file testo che compongono il modulo sorgente (vedi Preparazione del modulo sorgente, Capitolo 3);

FROM STM — indica il numero d'ordine della 1a istruzione del modulo sorgente;

TO STM — indica il numero dell'ordine dell'ultima istruzione del modulo sorgente.

```
                    ASSEMBLER  STATISTICS

                                                        PAGE 6

                                                DATE : 79/12/14

PROGRAM NAME =
NUMBER OF STATEMENT FLAGGED = 0
HIGHEST SEVERITY WAS  0
LINK OBJECT LENGTH = 00000078
OBJECT MODUL LENGTH = 006A
```

dove:

PROGRAM NAME — indica l'eventuale nome che accompagna la prima frase (START) del modulo sorgente; ad esempio BIN START;

NUMBER OF STATEMENT FLAGGED — indica il numero di frasi Assembler del modulo sorgente che l'assemblatore ha riscontrato errate;

HIGHEST SEVERITY WAS — indica il massimo grado di gravità di errore (severity code, vedi Segnalazioni diagnostiche durante l'assemblaggio, Appendice A) riscontrato nel modulo sorgente;

LINK OBJECT LENGTH — indica in esadecimale l' occupazione di memoria (in byte) del modulo oggetto completo di testata (vedi Appendice D);

OBJECT MODUL LENGTH — indica in esadecimale l' occupazione complessiva di memoria (in byte) delle istruzioni che compongono il modulo oggetto.
Il numero espresso da OBJECT MODUL LENGTH è sempre inferiore di 14 byte al numero espresso da LINK OBJECT LENGTH (vedi Appendice D).

Stampe opzionali

L'assemblatore può inoltre documentare in stampa opzionalmente (ovvero, soltanto se richiesto con lo specifico parametro LIST o XREF) sia il modulo oggetto che le relative cross references.

Modulo oggetto

```
                                                                      PAGE 4

                          EXTERNAL SYMBOL DICTIONARY

                                NO SYMBOL DEFINED

                                                                      PAGE 5
LINE  LOC      OBJECT CODE      STMT  SOURCE STATEMENT          DATE : 79/12/14

0010 000000                       1          START
0020 000000                       2          PROC
             180D
             41D0 0014
             23DD
             5000 D000
0030 00000C 0530                  3          BALR 3,0
0040                              4          USING *,3
0050                              5          USING LOCAL,13
0060 00000E 4140 0001             6          LA 4,1
0070 000012 2444                  7          GSA 4,4
0080 000014 4150 0002             8          LA 5,2
0090 000018 2455                  9          GSA 5,5
0100 00001A D202 D007 5002        10         MVC ISO+1(3),2(5)
0110 000020 9200 D006             11         MVI ISO,X'00'
0120 000024 5850 D006             12         L 5,ISO
0130 000028 0115                  13         ALRI 5,2
0140 00002A 4170 0000             14         LA 7,0
0150 00002E 4160 0008             15         LA 6,8
0160                              16   LOOP  EQU *
0170 000032 0407                  17         SLL  7,1
0180 000034 9530 5000             18         CLI 0(5),X'30'
0190 000038 4780 3030             19         BE AVANTI
0200 00003C 0107                  20         ALRI 7,1
0210 00003E 0105                  21   AVANTI ALRI 5,1
0220 000040 4660 3024             22         BCT 6,LOOP
0230 000044 4070 D004             23         STH 7,HALF
0240 000048 4100 0000             24         LA 0,0
0250 00004C DB02 D006 D004        25         CBS  ISO(3),HALF
0260 000052 D900 D00A D006        26         CSD DATA,ISO
0270 000058 DD00 4000 D00A        27         COM 0(4),DATA
0280 00005E                       28         PRRET
             58D0 D000
             4100 0014
             2200
             2800
0290 00006A                       29         ORG *-2
0300 000068 07FE                  30         BR 14
0310 000000                       31   LOCAL DSECT
0320 000000                       32   OLDR13 DS F
0330 000004                       33   HALF  DS  H
0340 000006                       34   ISO   DS CL4
0350 00000A                       35   DATA  DS CL8
0360 000014                       36   ENDLOCAL DS 0F
```

dove:

EXTERNAL SYMBOL DICTIONARY NO SYMBOL DEFINED
indica che nel modulo Assembler non vi sono riferimenti a simboli esterni;

LINE — numero di riga delle istruzioni del modulo sorgente;

LOC — valore dello scostamento in byte delle istruzioni assemblate (tale valore è contenuto nel location counter) rispetto all'inizio del modulo oggetto;

OBJECT CODE — codici esadecimali delle istruzioni e dei loro argomenti, organizzati a gruppi di 4 digit. I primi due digit dello object code rappresentano sempre il codice operativo dell'istruzione. Gli altri digit assumono significato diverso a seconda del formato dell'istruzione;

STMT — numero d'ordine progressivo delle istruzioni del modulo sorgente;

SOURCE STATEMENT
istruzioni del modulo sorgente.

```
                                CROSS REFERENCES                    PAGE  3

SYMBOL      LEN    VALUE    DEFN  REFERENCES                  DATE : 79/12/14

AVANTI        2    00003E     21     19
DATA          8    00000A     35     26     27
ENDLOCAL      4    000014     36      2     28
HALF          2    000004     33     23     25
ISO           4    000006     34     10     11     12     25     26
LOCAL         1    000000     31      2      5     28
LOOP          1    000032     16     22
OLDR13        4    000000     32
```

dove:

SYMBOL	nome simbolico;
LEN	occupazione di memoria (in byte) del nome simbolico;
VALUE	valore del location counter relativo al nome simbolico;
DEFN	numero d'ordine dell'istruzione ove è stato definito il nome simbolico;
REFERENCES	numeri d'ordine delle istruzioni che fanno riferimento al nome simbolico.

4. RICHIAMO DI MODULI ASSEMBLER DA PROGRAMMA BASIC

Caratteristiche del programma

Un programma in grado di utilizzare uno o più moduli Assembler si differenzia da un normale programma BASIC solo per l'introduzione di una nuova istruzione (CALL), che permette di richiamare tali moduli (uno alla volta) ovunque occorra.

Istruzione CALL

Funzione

Carica in memoria principale un modulo Assembler e ne inizia l'esecuzione.

Formato

$$\text{CALL string-exp1}\left[,\left\{\begin{matrix}\text{num-exp}\\ \text{string-exp}\end{matrix}\right\}\right]\ldots\left[;\text{DEB}\right]$$

dove:

string-exp$_1$ è un'espressione stringa il cui valore indica il nome di un modulo Assembler presente in una delle librerie aperte;

num-exp è un'espressione numerica il cui valore è passato come argomento al modulo Assembler richiamato ;

string-exp è un'espressione stringa il cui valore è passato come argomento al modulo Assembler richiamato ;

DEB è la stringa omonima e specifica al sistema di caricare il modulo di sistema che permette di eseguire il debugging del modulo Assembler caricato in memoria principale.

Azione

L'esecuzione del programma è trasferita all'insieme di istruzioni Assembler che costituiscono il modulo oggetto il cui nome è specificato da string-exp1. Il nome del modulo viene ricercato su tutte le librerie aperte. Se la ricerca ha esito positivo, viene compilata una tabella con l'indirizzo di disco del modulo stesso. In tal modo, si evitano ulteriori ricerche in caso di chiamata allo stesso modulo. La tabella consente il caricamento degli indirizzi di 16 moduli distinti.
Ogni ulteriore modulo richiamato sostituisce un indirizzo già presente in tabella, a partire dai primi caricati.
Esempio:

MODULO RICHIAMATO	INDIRIZZO CORRISPONDENTE
17°	1°
18°	2°
19°	3°
⋮	⋮

Se il modulo è reperito sulle librerie aperte, viene caricato nell'area di overlay ove permane sino ad una nuova occupazione di tale area, o da parte dell'utente o del sistema.

Note

1. Gli argomenti specificati nell'operando

$$\left[,\left\{\begin{array}{l}\text{num-exp}\\ \text{string-exp}\end{array}\right\}\right]\ldots$$

si considerano ordinati in senso crescente da sinistra verso destra, partendo da 1 con incremento di 1.
Esempio:

```
4Ø CALL MODUL,A$,C(3,4),B$(2)
      A$      = 1° argomento
      C(3,4)  = 2° argomento
      B$(2)   = 3° argomento
```

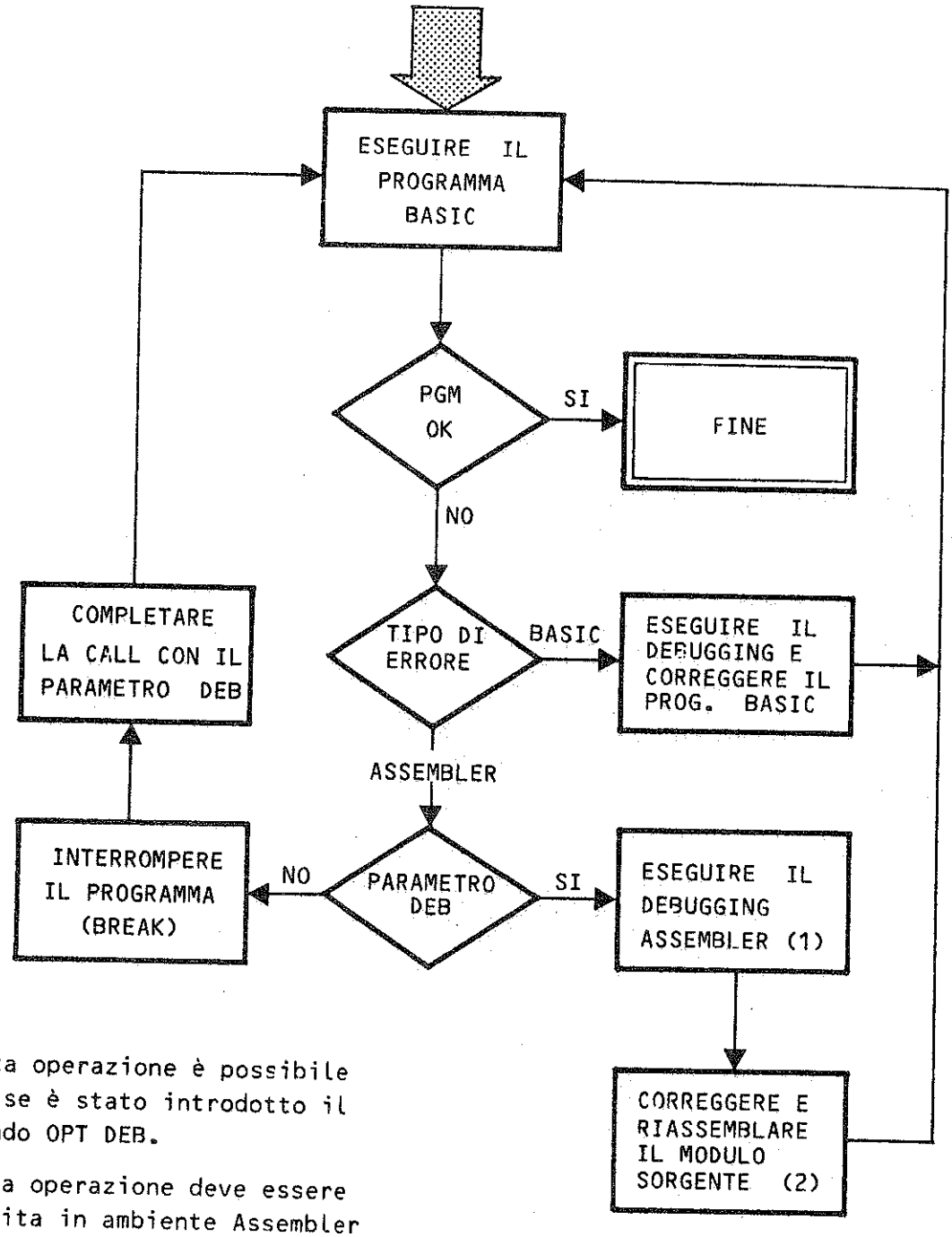
Gli argomenti passati al modulo Assembler sono caricati nell'area di stack da cui potranno essere prelevati (vedi Prelievo degli argomenti, capitolo 2).

2. Il numero di argomenti che si possono passare al modulo Assembler specificato è limitato solamente dal numero di caratteri che possono costituire una istruzione BASIC (80 caratteri).

3. Al termine della esecuzione del modulo Assembler , viene eseguita l'istruzione esecutiva successiva all'istruzione CALL.

4. Se non si specifica l'opzione DEB, è impossibile eseguire operazioni di debugging sul modulo Assembler.

5. DEBUGGING DEL PROGRAMMA BASIC COMPLETO DI MODULO ASSEMBLER



(1) Questa operazione è possibile solo se è stato introdotto il comando OPT DEB.

(2) Questa operazione deve essere eseguita in ambiente Assembler (fig. 1-1).

Fig. 5-1 Procedura di debugging di modulo Assembler

Note

1. Il tipo di errore (BASIC/ASSEMBLER) viene evidenziato da dicitura specifica: per gli errori di tipo BASIC, ci si riferisca al Manuale Generale P6066; per gli errori di tipo Assembler, ci si riferisca all'appendice A del presente manuale.

2. Eseguita l'operazione di debugging ed apportate le opportune correzioni al modulo Assembler, per poterlo rieseguire è necessario ritrasformarlo in oggetto attraverso le seguenti fasi operative:

   - spegnere la macchina;
   - sostituire un floppy disk Assembler al floppy disk BASIC;
   - riaccendere la macchina;
   - riassemblare (vedi Preparazione del modulo oggetto, capitolo 3);
   - ritornare in ambiente BASIC (fig. 1-2).

Come accedere allo stato debugging

Il sistema può eseguire programmi sotto il controllo del debugging solo se l'opzione OPT DEB è caricata in memoria.
Le routine di debugging sono caricate in memoria, ma non sono attivate; vengono attivate quando:

- si specifica il parametro DEB nella CALL;
- durante l'esecuzione del programma si preme il tasto STEP ;
- a fronte di un errore.

Quando un programma viene eseguito sotto il controllo del modulo di debugging, il suo tempo d'esecuzione viene aumentato di circa un 20%.

Entrata in stato debugging: il sistema commuta nello stato debugging quando:

- si esegue una CALL col parametro DEB;
- si rileva un errore durante l'esecuzione di un programma;
- si incontra un alt indirizzato;
- si preme il tasto di console STEP

Quando il sistema è nello stato di debugging, il tasto di console STEP è illuminato.

1. Quando si esegue una CALL con il parametro DEB il sistema commuta nello stato di debugging. Il tasto di console STEP si illumina.

2. Se, durante l'esecuzione di un modulo Assembler , viene rilevato un errore, l'esecuzione del programma è interrotta, il sistema commuta nello stato di debugging e viene visualizzato un appropriato messaggio di errore sul display.

3. Quando si preme il tasto di console STEP , l'esecuzione di un programma viene interrotta. Il sistema commuta nello stato di debugging e sul display appare il messaggio. Se si preme di nuovo il tasto STEP , viene eseguita l' istruzione visualizzata e il sistema commuta di nuovo nello stato di debugging.

4. Per quanto accade al riconoscimento di un ALT INDIRIZZATO si vedano i comandi da tastiera del presente capitolo.

Dopo la segnalazione d'errore su display, si può premere il tasto STEP o il tasto BREAK che manda in terminazione il programma e pone il sistema nello stato comandi.
Quando si entra in stato debugging, a fronte di un errore del programma utente, vengono visualizzate su display le seguenti informazioni:

- il valore del program counter
- l'istruzione corrente (quella che deve essere eseguita)
- il condition code

nel seguente formato:

iiiiii oooo oooo oooo CC=x

dove:

iiiiii = program counter
oooo = codice oggetto in esadecimale
x = valore del condition code.

Se si verifica un errore durante l'esecuzione di un programma utente e non è stato lanciato il comando OPT DEB, il sistema segnala l'errore relativo su display ed il programma si ferma; l'unico modo per sbloccare il sistema è di premere il tasto BREAK che manda in terminazione il programma.

Strumenti dello stato debugging

Quando il programma è nello stato di debugging, si possono utilizzare i seguenti strumenti per ricercare la causa degli errori di un programma presente in memoria principale:

1. Comandi da tastiera;

2. Tasti di console.

Comandi da tastiera

Nello stato debugging è possibile introdurre da tastiera dei comandi che visualizzano o modificano aree di memoria e il contenuto dei registri. I comandi vengono introdotti da tastiera, carattere per carattere, e ogni linea di comando viene chiusa da tasto (END OF LINE). La lunghezza della linea di comando non deve superare i 32 crt; in caso contrario verrà rifiutata con segnalazione acustica. Nel caso in cui il comando introdotto risulti sintatticamente scorretto viene data segnalazione di comando errato; premendo il tasto (RECALL) è possibile riottenere la linea in modo da poterla correggere e reintrodurre.
I comandi che si possono introdurre in stato debug - ging sono i seguenti:

- display istruzione corrente
- display registro
- modifica registro
- stampa registri
- somma un immediato ad un registro
- display memoria
- modifica memoria
- stampa memoria
- avanzamento in display
- modifica program counter
- alt indirizzato.

Errori in stato debugging: introducendo i comandi da tastiera si può incorrere in errori di introduzione che possono essere così sintetizzati:

- comando errato, il comando appena introdotto non è sintatticamente corretto;

- modifica errata: la modifica non è stata eseguita correttamente, più precisamente:
  . è stato modificato il campo indirizzo o il campo indicante il registro

- i campi sono stati introdotti non nello stesso formato di output
- sono stati introdotti caratteri non esadecimali.

A fronte di questi errori il sistema visualizza que - sta segnalazione:

INCORRECT FORMAT

e si mette in attesa dell'intervento dell'operatore ; questi, premendo il tasto (RECALL) , può riottenere il comando errato in modo da poterlo correggere e rein - trodurre.
A fronte di un comando specificante un indirizzo fuori della memoria, il sistema visualizza questa segnalazione:

*MEMORY ADDRESSES EXCEEDED"

e si mette in attesa dell'intervento dell'operatore ; questi, premendo il tasto (RECALL) , può riottenere il comando errato in modo da poterlo correggere e reintrodurre.
Nel seguito si dà la descrizione funzionale dei comandi da tastiera.

Display istruzione corrente:

Funzione : visualizza l'istruzione corrente; l'istruzione corrente viene pure visualizzata ad ogni entrata in stato debugging.

Formato : (EOL)

Output : *iiiiii* oooo oooo oooo CC=x

dove:

iiiiii = program counter
oooo = codice oggetto in esadecimale
x = valore del condition code
CC=** = caratteri alfanumerici fissi

Messaggi di errore: *INCORRECT FORMAT*.

Display registro:

Funzione : visualizza, in esadecimale, il contenuto del registro indicato.

Formato : Ry[y] (EOL)

Output : *Reg.yy* xxxx xxxx

dove:

yy = numero del registro richiesto
xxxx = contenuto, in esadecimale, del registro
Reg.= caratteri alfanumerici fissi

Messaggi di errore: *INCORRECT FORMAT*.

Modifica registro:

Funzione : viene visualizzato, in esadecimale, il contenuto del registro indicato, con il pointer posizionato in ultima colonna. In questa situazione è possibile editare soltanto sul campo che indica il contenuto del registro.
Per rinunciare alla modifica è necessario digitare il tasto CLEAR

Formato : Ry[y]M (EOL)

Output : *Reg.yy* xxxx xxxx

dove:

yy = numero del registro richiesto
xxxx = contenuto, in esadecimale, del registro
Reg.= caratteri alfanumerici fissi

Messaggi di errore: *INCORRECT FORMAT*.

Stampa registri:

Funzione : stampa, in esadecimale, il contenuto di tutti i registri.

Formato : R (EOL)

Output : *Reg.00* xxxx xxxx ..... *Reg.12* xxxx xxxx
Reg.01 xxxx xxxx ..... *Reg.13* xxxx xxxx
Reg.02 xxxx xxxx ..... *Reg.14* xxxx xxxx
Reg.03 xxxx xxxx ..... *Reg.15* xxxx xxxx

dove:

xxxx = contenuto, in esadecimale, del registro
Reg.nn = caratteri alfanumerici fissi
nn = numero del registro

Messaggi di errore: *INCORRECT FORMAT*.

Somma immediato a registro:

Funzione : esegue la somma binaria fra il contenuto di un registro (24 bits meno significativi) e una costante esadecimale, visualizzando su display il risultato. Il contenuto del registro non viene modificato.

Formato : Ry[y] + xxxxxx (EOL)

Output : xxxxxx

dove:

yy = numero del registro richiesto
xxxxxx = costante esadecimale (nel formato) e risultato dell'operazione (nell' output).

Messaggi di errore: *INCORRECT FORMAT*.

Display memoria:

Funzione : visualizza 8 bytes contigui di memoria

Formato : iiiiii (EOL)

Output : *iiiiii* xxxx xxxx xxxx xxxx

dove:

iiiiii = indirizzo del primo byte
xxxx = contenuto, in esadecimale, della memoria
** = caratteri alfanumerici fissi.

Messaggi di errore: *INCORRECT FORMAT*
MEMORY ADDRESSES EXCEEDED

Modifica memoria:

Funzione : visualizza 8 bytes contigui di memoria. Il pointer sarà in ultima posizione. Dopo tale comando sarà possibile editare sul contenuto della memoria ma non sul campo in dirizzo.
Per rinunciare alla modifica è necessario digitare il tasto CLEAR .

Formato: : iiiiii M (EOL)

Output: : *iiiiii* xxxx xxxx xxxx xxxx

dove:

iiiiii = indirizzo del primo byte
xxxx = contenuto, in esadecimale, della memoria
** = caratteri alfanumerici fissi

Messaggi di errore: *INCORRECT FORMAT*
MEMORY ADDRESSES EXCEEDED

Stampa memoria:

Funzione : stampa, in esadecimale, il contenuto della memoria per tanti bytes quanti sono quelli indicati nel comando, con arrotondamento a 16. Se tale parametro è stato omesso, viene assunto il valore 16.

Formato : iiiiii L [yyy] (EOL)

Output : *iiiiii* xxxx xxxx xxxx xxxx ----- xxxx

dove:

iiiiii = indirizzo del primo byte
yyy = numero di bytes da stampare (in decimale)
xxxx = contenuto, in esadecimale, della memoria
** = caratteri alfanumerici fissi

Messaggi di errore: *INCORRECT FORMAT*
MEMORY ADDRESSES EXCEEDED

Avanzamento in display: (memoria o registro)

Funzione : permette di visualizzare il registro o i byte successivi a quelli appena visti. Questo comando ha effetto solo dopo una visualizzazione o un visualizzazione con modifica.

Formato : N (EOL)

Output : Esempio: se utilizzato dopo la visualizzazione di un registro (es. R2), provoca la visualizzazione del registro successivo (es. R3). Il registro successivo a R15 viene considerato il registro RØ. Se utilizzato dopo la visualizzazione di memoria, provoca la visualizzazione degli 8 byte successivi di memoria.

Messaggi di errore: *INCORRECT FORMAT*
MEMORY ADDRESSES EXCEEDED

Modifica program counter:

Funzione : modifica il valore del program counter. In pratica opera come un branch incondizionato a un' altra parte del programma.

Formato : iiiiii J (EOL)

Output : *iiiiii* oooo oooo oooo CC=x

dove:

iiiiii = program counter modificato
oooo = codice oggetto in esadecimale
x = valore del condition code
CC=** = caratteri alfanumerici fissi

Messaggi di errore: *INCORRECT FORMAT*
MEMORY ADDRESSES EXCEEDED.

Alt indirizzato:

Funzione : predispone un particolare indirizzo di BREAK-POINT all'interno del programma utente, al riconoscimento del quale la debugging viene invocata e si entra in stato debugging. Una nuova impostazione rimpiazza il BREAK-POINT precedente. Per annullare tale BREAK - POINT basta impostare il comando OS(EOL). L'indirizzo predisposto dovrà essere puntato sul codice operativo dell'istruzione, diversamente verrà ignorato. La ricerca del BREAK-POINT avviene dalla istruzione successiva a quella corrente.

Formato : iiiiii S (EOL)

Output : *iiiiii* oooo oooo oooo CC=x

dove:

iiiiii = program counter
oooo = codice oggetto in esadecimale
x = valore del condition code
CC=** = caratteri alfanumerici fissi

Messaggi di errore: *INCORRECT FORMAT*.

Comandi da console

Tre tasti di console sono particolarmente utili come strumenti di verifica dei programmi: CONTINUE , STEP e TRACE .
Nel seguito è spiegato il loro impiego e sono dati alcuni suggerimenti che permettono di utilizzare, nello stato di debugging, le prestazioni offerte dai tasti di console BREAK e NO PRINT .

Se, mentre il sistema è nello stato di debugging, è premuto il tasto di console CONTINUE , riprende l'esecuzione del programma che è in memoria principale. Il tasto CONTINUE si illumina quando è premuto mentre il sistema è nello stato di debugging. La funzione del tasto è attiva solamente quando il sistema si trova nello stato di debugging.

Se, mentre il sistema è nello stato di debugging, è premuto il tasto di console STEP , si può eseguire passo a passo (ossia istruzione per istruzione) il programma presente in memoria principale. Ogni volta che una istruzione del programma è eseguita,il sistema visualizza sul display l'istruzione logicamente successiva, che specifica quale istruzione del programma sara' eseguita alla successiva pressione del tasto STEP . Dopo di che, ogni volta che si preme il tasto STEP , viene eseguita la successiva istruzione (Si ricordi che il tasto di console STEP si illumina ogni volta che il sistema è nello stato di debugging).

Il tasto di console TRACE permette di visualizzare il codice oggetto di ogni istruzione eseguita, nell'ordine con cui essa è eseguita. (Il tasto TRACE è acceso quando la funzione omonima è attiva). Per utilizzare la funzione TRACE nello stato di debugging:

1. Premere il tasto TRACE
2. Premere il tasto. CONTINUE

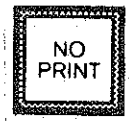
Come conseguenza, si riprende l'esecuzione del programma e vengono visualizzati i codici oggetto delle istruzioni eseguite. Si noti che non vengono stampati i codici delle istruzioni non esecutive, come * o DC, e l'esecuzione del programma è realizzata come quando la funzione TRACE non è attiva (la funzione TRACE può essere attivata anche a run-time.
Il codice oggetto di ogni istruzione eseguita viene visualizzato nel seguente formato:

iiiiii oooo oooo oooo CC=x

dove:

iiiiii = program counter
oooo = codice oggetto in esadecimale
x = valore del condition code
CC=** = caratteri alfanumerici fissi

Se si vuole tenere traccia delle visualizzazioni si prema il tasto PRINT ALL che fa stampare tutte le informazioni visualizzate.



Quando la funzione del tasto NO PRINT è attiva, la stampante integrata è inibita. Questa prestazione può essere utile per risparmiare tempo di stampa e carta. Quando la funzione NO PRINT è attiva, il tasto NO PRINT è illuminato e in questo caso viene disabilitata la eventuale funzione di PRINT ALL.

Il tasto BREAK, premuto nello stato di debugging, permette di terminare l' esecuzione di un programma e commuta il sistema nello stato comandi. Nello stato comandi, si può modificare il programma presente in memoria principale oppure introdurre un qualsiasi comando di sistema. Quando la funzione BREAK è attiva, il tasto BREAK è illuminato.

Uscita dallo stato debugging

Si può uscire in seguito ad una delle seguenti operazioni:

- premendo il tasto STEP ;in questo caso viene eseguita un'istruzione del programma utente, dopo di che si ritorna in stato debugging

- premendo il tasto CONTINUE ; in questo caso si ri - prende l'esecuzione del programma

- premendo il tasto BREAK ; in questo caso si passa direttamente alla terminazione del programma.

Output del debugging

I messaggi emessi, durante lo stato di debugging, vengono visualizzati su display a meno di quelli emessi a fronte di comandi di stampa da tastiera (stampa -registro e memoria).
Nel caso che l'utente voglia tenere traccia dei mes - saggi emessi, può premere il tasto PRINT ALL che comanda al sistema di mandare su stampa i messaggi che compaiono sul display.
Se il sistema ha nella sua configurazione una stampante IPSO oppure un video, la stampa dei messaggi viene effettuata sulla stampante integrata o IPSO a seconda di come si è configurato il sistema (vedi comando CONFIGURE).

Luci di console

Le lampadine, in stato debugging hanno il seguente significato:

- luce running:
  . fissa,quando il debugging è in attesa di un comando
  . pulsante,mentre il debugging sta elaborando un comando.

- luce overflow:
  . accesa,quando si cerca di introdurre da tastiera più di 80 crt.

- de-grad:
  . sempre spenta.

- TRACE/NO PRINT/PRINT ALL:
  . accesa,quando la predisposizione è in atto.

Si dà di seguito una tabella sintetizzante la situa - zione delle lampadine nei vari stati del sistema:

LUCE	STATO OFF	ESECUZIONE PROGRAMMA	DEBUGGING
RUNNING	0	*	1
STEP	0	0	1
CONTINUE	0	1	0
BREAK	0	0	0
CALC.MODE	0	0	0

dove:

0 = luce spenta
1 = luce accesa
* = luce pulsante

6. COMANDI DI SISTEMA

Comandi utilizzabili nel sistema Assembler

Nel sistema Assembler sono accettati i seguenti comandi del repertorio BASIC:

AUTO #
CATALOG
CONFIGURE
DATA
DCHANGE
DELETE LINE
EXEC
FETCH
LDKEYS
LIST
OLD
PURGE
REPLACE
RESEQUENCE
SAVE
SHIFT
SPACE
STKEYS
TEXT

Comandi identici in entrambi i sistemi

I seguenti comandi hanno sintassi e comportamento identici a quelli del sistema BASIC.

AUTO #
DATA
DELETE LINE
EXEC
FETCH
LDKEYS
LIST
RESEQUENCE
SHIFT
STKEYS
TEXT

Comandi con diverso comportamento nei due sistemi

Appartengono a questo gruppo i seguenti comandi: OLD, PURGE, REPLACE, SPAGE.

COMANDO	COMPORTAMENTO ASSEMBLER	COMPORTAMENTO BASIC
OLD PURGE REPLACE	Il file cui i comandi di riferisco no viene ricercato prima su floppy disk sistema e quindi su floppy disk utente.	Il file cui i comandi si riferiscono viene ricercato su tutte le librerie aperte.
SPACE	Viene stampato lo spazio (in byte) disponibile sui floppy disk sistema ed utente.	Viene stampato lo spazio (in byte) disponibile sulla prima libreria aperta.

Comandi con diversa sintassi nei due sistemi.

Appartengono a questo gruppo i seguenti comandi:

CATALOG, DCHANGE, SAVE, CONFIGURE.

La differenza saliente della nuova sintassi di tali comandi è che, operando in ambiente FDU, non si specificano le librerie ma si fa riferimento al supporto (FDU1/FDU2).
Se non viene specificato il supporto, si opera sulla libreria di sistema.
Di seguito sono illustrati i nuovi formati.

Comando CATALOG

```
CAT[ALOG] [filename]
          [  *    ] , [(,{FDU1}) ] , [T] [,F]
          [  +    ]     {FDU2}       [O]
          [  :    ]
```

Comando DCHANGE

```
DCH[ANGE] {FDU1}
          {FDU2}
```

Comando SAVE

SAV[E] filename[,(,{FDU1 / FDU2})]

Comando CONFIGURE

CON[FIGURE][EVD][,EP=n_1][,MS=n_2]

Note

1. Nel comando CATALOG, è possibile chiedere la selezione specifica soltanto per i file testo (T) oppure oggetto (O).

2. Nel comando CONFIGURE, è possibile chiedere la selezione specifica soltanto per il video esterno alfanumerico.

A. MESSAGGI DI ERRORE

Questi messaggi identificano eventuali errori che si verificano durante la fase di assemblaggio (programma di utilità ASM) oppure durante l'esecuzione di un modulo Assembler (istruzione CALL).
Gli errori di esecuzione di un modulo possono essere corretti durante la fase di esecuzione di un programma. Quando si verifica un errore, viene interrotta l'esecuzione del programma ed il sistema commuta nello stato di debugging se nella memoria sono presenti le routine di debugging (comando OPT DEB). Nello stato di debugging si può modificare opportunamente il codice oggetto e riprendere l'esecuzione del programma (vedi capitolo 5). Se le routine di debugging non sono presenti in memoria l'unico modo per sbloccare il sistema dalla situazione di errore è di premere il tasto BREAK : il sistema ritorna nello stato comandi e quindi si possono effettuare le necessarie correzioni sul programma sorgente. Se una segnalazione di errore persiste dopo diversi tentativi, ci si rivolga al più vicino servizio Olivetti evidenziando il messaggio visualizzato dal sistema.

Errori bloccandi durante l'assemblaggio

Questo paragrafo descrive i messaggi di errore bloccanti che si possono verificare durante l'esecuzione dell'utility ASM; vengono segnalati dall'utility.

ERRORE	DESCRIZIONE
INSUFFICIENT SPACE ON UNIT {FDU1 / FDU2} FOR WORK FILE	Gli extent liberi di disco assegnati allo scratch file non sono sufficienti a contenere la stringa intermedia e l'eventuale reference table: l'utente deve creare più spazio sul floppy disk indicato.
LOCATION COUNTER OVERFLOW	Il location counter assegnato ad ognuna delle sezioni di controllo del programma sorgente supera $2^{31}-1$:

ERRORE	DESCRIZIONE
	l'utente deve modificare in modo opportuno il proprio programma sorgente.
TOO MANY DEFINED NAMES	L'utente ha definito nomi in eccesso rispetto al numero consentito dalle dimensioni della memoria: l'utente deve o diminuire il numero dei nomi nel proprio programma sorgente o passare ad una macchina di capacità superiore..
TOO MANY DSECT	Il numero di dummy section definite supera 127: l' utente deve modificare opportunamente il proprio programma sorgente.
TOO MANY EXTERNAL NAMES	L'utente ha definito troppi nomi esterni, nel programma sorgente, rispetto alla capacità assegnata al dizionario: l'utente deve o diminuire il numero dei nomi esterni nel proprio programma sorgente o passare ad una macchina di capacità superiore.
TOO MANY LITERALS	L'utente ha definito literal tali che il codice oggetto generato, a fronte dei literal, supera la dimensione della tabella dei literal; l'utente deve o diminuire il numero dei literal nel proprio programma sorgente o passare ad una macchina di capacità superiore.

Errori non bloccanti durante l'assemblaggio

Questo paragrafo descrive i messaggi di errore non bloccanti che si possono verificare durante l'esecuzione dell'utility ASM; vengono segnalati dall'utility.

ERRORE	DESCRIZIONE
INSUFFICIENT SPACE ON UNIT {FDU1 / FDU2} FOR OBJECT FILE	Sul supporto non ci sono extent sufficienti per allocare il codice oggetto: l'utente deve creare più spazio sul floppy disk o fare scelte più opportune, a livello argomenti di lan -

ERRORE	DESCRIZIONE
	cio, nell'indicare l'unità di residenza del flusso di lavoro e del codice oggetto.
REP OVERFLOW	Overflow nella cross reference. Tabella stampata in modo incompleto.
OBJECT LENGTH Ø - FILE OBJECT NOT ALLOCATED	Codice oggetto non allocato perchè, a fronte del programma sorgente, non è stato generato alcun codice oggetto.
SOURCE LITERAL TABLE OVERFLOW	Tabella literal stampata in modo incompleto.

Segnalazioni diagnostiche durante l'assemblaggio

Questo paragrafo descrive l'elenco delle segnalazioni diagnostiche emesse dall'Assemblatore, in fase di assemblaggio di un programma sorgente, a fronte di errori nel sorgente. L'apposizione del segno (*), in coda al testo dell'errore, indica che il messaggio è accompagnato dall'indicazione della colonna, rispetto a inizio operandi, presso la quale si è rilevato l'errore. L'indicazione della colonna, in certi casi, per ragioni inerenti alla logica del sistema, perde di significato.
Ogni segnalazione diagnostica è corredata di un codice di due cifre (severity code) che indica la gravità dell'errore del sorgente:

SEVERITY CODE	SVILUPPO
Ø4	Errore ripristinabile che consente l'esecuzione di ogni eventuale passo successivo. L'oggetto generato dev'essere corretto prima della sua esecuzione.
Ø8	Errore non ripristinabile sull'output generato. L'esecuzione di ogni passo successivo dà risultati imprevedibili.
12	Errore non ripristinabile. Non viene creato oggetto.

ERRORE	DESCRIZIONE
AØØ1 ILLEGAL NAME FIELD	Spiegazione Nome assente in frasi in cui è obbligatorio indicare il nome frase DSECT e frase EQU, oppure nome presente in frasi in cui il nome non è ammesso: frasi EXT/USING/DROP/END/ORG/frasi di controllo listing. Severity code Ø8 Sviluppo La frase in cui è rilevato questo errore è comunque analizzata e tradotta; la frase DSECT crea comunque una sezione fittizia; la frase EQU senza nome è praticamente inoperante; i nomi non ammessi non sono posti nella tavola dei nomi.
AØØ2 INVALID NAME	Spiegazione Il campo nome contiene un nome scorretto; contiene più di 8 crt, oppure non inizia con un carattere alfabetico, oppure contiene dei caratteri speciali. Severity code Ø8 Sviluppo Il nome scorretto non viene posto nella tavola dei nomi; la frase in cui è rilevato questo errore è comunque analizzata e tradotta.
AØØ3 INVALID OR MISSING OPERATION CODE	Spiegazione Nel campo operazione della frase non è stato indicato un codice operativo, oppure il codice operativo non è correttamente isolabile, perchè costituito da più di 6 caratteri.

	Severity code 12 Sviluppo La frase non viene presa in considerazione dall'Assembler; l'eventuale nome presente non è posto nella tavola dei nomi.
AØØ4 UNDEFINED OPERATION	Spiegazione La stringa di caratteri indicata nel campo operazione della frase sorgente non corrisponde al codice mnemonico di una istruzione macchina, al codice di una istruzione direttrice o di una macro-istruzione. Severity code 12 Sviluppo La frase non viene presa in considerazione dall'Assembler; l'eventuale nome è posto nella tavola dei nomi.
AØØ5 ERROR IN LOGICAL SEQUENCE	Spiegazione Frase fuori sequenza logica: frase PROC non segue immediatamente la frase START/CSECT di inizio Control Section; frasi, diverse dalle frasi commento, o comunque trattate come commento, diverse dalle frasi di controllo listing e dalle frasi EQU, situate prima dello inizio della sezione di programma. Severity code 12 Sviluppo La frase fuori sequenza non è analizzata e tradotta dall'Assembler.

AØØ6 MISSING OPERAND FIELD	Spiegazione Campo operandi assente in frase in cui è obbligatorio. Severity code Ø8 Sviluppo Se si tratta di istruzione macchina, viene sviluppato l'oggetto a zero binario; l'eventuale nome viene posto nella tavola dei nomi. Se si tratta di frase direttiva all'Assembler, essa non viene eseguita; in particolare nel caso di frasi di definizione dati con campo operandi assente non viene sviluppato alcun oggetto, e all'eventuale nome viene attribuita lunghezza 1.
AØØ7 ILLEGAL CONTINUATION	Spiegazione Errore di continuazione: più di due linee di continuazione; i primi 15 caratteri delle linee di continuazione non a Ø; il campo operandi sulle linee di continuazione non inizia alla colonna 1 di continuazione del campo commento; l'ultima linea sorgente con l'indicativo di continuazione. Severity code Ø8 Sviluppo L'assembler isola il campo operandi fino a che è stato rilevato l'errore di continuazione. Si precisa che l'assembler considera facenti parte di una stessa frase sorgente tutte le linee di continuazione indicate nel campo continuazione, anche oltre le due linee permesse in base alle specifiche del linguaggio: il campo operandi si considera comunque concluso nelle prime due linee di continuazione.

AØØ8 ILLEGAL CHARACTER OR INVALID DELIMITER (*)	Spiegazione Carattere illegale alla posizione indicata: l'Assembler si aspetta un termine, un operatore o uno dei due delimitatori che separano i sottocampi degli operandi e gli operandi stessi (virgola, parentesi aperta o chiusa, apice di apertura della stringa caratteri nelle pseudo di definizione dati) e non lo trova. Severity code Ø8 Sviluppo L'analisi e traduzione della frase si interrompe.Nel caso di istruzione macchina, l'oggetto viene generato a Ø binario; nel caso di pseudo di definizione dati viene generata la corrispondente stringa oggetto o a Ø binario o viene allocata la corrispondente area solo se è stato possibile valutarne la lunghezza.
AØØ9 INCORRECT SPECIFICATION OF REGISTER OR MASK (*)	Spiegazione L'espressione usata per indicare un registro o una maschera non ha valore assoluto, o comunque un valore assoluto compreso tra Ø e 15. Severity code Ø8 Sviluppo L'Assembler prosegue l'analisi sintattica e semantica della frase; la stringa oggetto corrispondente è comunque posta a Ø binario.

AØ1Ø SYNTAX ERROR IN OPERAND FIELD (*)	Spiegazione La sintassi della frase prevede la fine del campo operandi e invece quest'ultimo continua scorrettamente. Severity code Ø8 Sviluppo L'Assembler interrompe l'analisi della frase, la cui corrispondente eventuale stringa oggetto è posta a Ø binario.
AØ11 INCORRECT SPECIFICATION OF IMMEDIATE (*)	Spiegazione L'espressione usata per indicare un valore immediato non è assoluta, o non ha valore compreso tra Ø e 255, oppure tra 1 e 16. Severity code Ø8 Sviluppo L'Assembler prosegue l'analisi sintattica e semantica della frase; la stringa oggetto corrispondente è comunque posta a Ø binario.
AØ12 PREMATURE END OF THE OPERAND FIELD	Spiegazione Fine prematura del campo operandi: sulla frase sorgente non è stato indicato il numero di operandi previsto, oppure la specificazione di un operando è incompleta. Severity code Ø8

	Sviluppo L'analisi della frase termina; l'oggetto relativo è posto a Ø binario; nel caso di pseudo definizione dati viene generata la corrispondente stringa oggetto a Ø binario o viene allocata la corrispondente area solo se è stato possibile valutarne la lunghezza.
AØ13 ALIGNMENT ERROR	Spiegazione Errore di allineamento: la frase richiede che l'operando sia allineato alla parola o alla semi-parola; questo errore viene segnalato solo quando l'indirizzo è espresso in modo implicito e tale indirizzo è dispari; in particolare viene segnalato nel caso in cui l'operando è espresso tramite literal il cui indirizzo risulti appunto dispari. Severity code Ø4 Sviluppo L'Assembler continua l'analisi e la traduzione della frase.
AØ14 TOO MANY OPERANDS (*)	Spiegazione Nella frase sorgente è stato specificato un numero di operandi eccessivo: • più di 16 registri specificati in una frase USING o DROP • più di 31 parametri specificati in una frase CALL o CALEXT o CALEXS • più di 31 nomi esterni indicati in una frase EXT • troppi operandi specificati in una qualsiasi istruzione-macchina; questo è il caso ad esempio in cui la posizione di inizio del literal eventualmente indicato non coincida con la posizione

	di fine dell'operando precedente. Severity code Ø8 Sviluppo Si interrompe l'analisi e la traduzione della frase: la stringa oggetto è generata a zeri binari.
AØ15 INVALID LENGTH IN MACHINE INSTRUCTION (*)	Spiegazione La lunghezza implicita o esplicita indicata in una istruzione-macchina non ha valore compreso fra 1 e 256 oppure tra 1 e 16. La lunghezza esplicita è stata espressa tramite una espressione non assoluta. Severity code Ø8 Sviluppo L'analisi della frase sorgente prosegue; la stringa oggetto viene comunque generata a zeri binari.
AØ16 INVALID DISPLACEMENT SPECIFICATION (*)	Spiegazione In una istruzione-macchina il campo scostamento è stato indicato tramite una espressione non assoluta, o comunque tramite una espressione il cui valore non è compreso tra Ø e 4Ø95. Severity code Ø8 Sviluppo L'analisi della frase sorgente prosegue; la stringa oggetto è comunque generata a zeri binari.

AØ17 REGISTER NOT PREVIOUSLY USED (*)	Spiegazione Un registro specificato in una frase DROP non è stato precedentemente definito come registro base in una opportuna frase USING. Severity code Ø4 Sviluppo L'Assembler ignora il registro specificato e prosegue nell'analisi della frase DROP.
AØ18 ADDRESSABILITY ERROR (*)	Spiegazione L'Assembler non riesce a risolvere l'indirizzo implicito specificato nella frase sorgente, poichè nessun registro base definito con una precedente frase USING è disponibile, cioè nessun registro base ha lo stesso attributo di rilocabilità dell'espressione che esprime l'indirizzo implicito o comunque quest'ultima ha un valore tale da dar luogo a uno scostamento 4Ø95. Severity code Ø8 Sviluppo L'analisi della frase sorgente prosegue; la stringa oggetto è generata a zeri binari.
AØ2Ø TOO MANY TERMS IN EXPRESSION (*)	Spiegazione Espressione con più di 16 termini. Severity code Ø8 Sviluppo L'analisi della frase termina.

AØ21 MORE THAN 5 LEVELS OF PARENTHESES (*)	Spiegazione Espressione con più di 5 livelli di parentesi. Severity code Ø8 . Sviluppo L'analisi della frase termina.
AØ22 RIGHT PARENTHESES OMITTED	Spiegazione Parentesi di chiusura omesse in una espressione. Severity code Ø8 Sviluppo L'analisi della frase sorgente si interrompe.
AØ23 UNDEFINED OR NOT PREVIOUSLY DEFINED SYMBOL (*)	Spiegazione Nella frase sorgente è indicato un simbolo non definito oppure un simbolo non precedentemente definito quando la previa definizione è richiesta (ad esempio frasi EQU, ORG .....). Severity code Ø8 Sviluppo L'espressione non può essere valorizzata; l' analisi della frase prosegue sugli operandi successivi.

AØ24 INVALID SELF-DEFINING TERM (*)	Spiegazione Termine autodefinito scorretto: i caratteri indicati nella stringa non sono conformi al tipo del termine stesso. Severity code Ø8 Sviluppo L'espressione non può essere valorizzata; l'analisi della frase comunque prosegue.
AØ25 SYNTAX ERROR IN SELF-DEFINING TERM (*)	Spiegazione Termine autodefinito non isolabile perchè non trova l'apice di chiusura della stringa in posizione corretta. Severity code Ø8 Sviluppo Si interrompe l'analisi sintattica della frase sorgente.
AØ26 VALUE OF SELF-DEFINING TERM TOO LARGE (*)	Spiegazione Termine autodefinito decimale con valore superiore a 16.777.215. Severity code Ø8 Sviluppo L'espressione non può essere valorizzata; l'analisi della frase comunque prosegue.

AØ27 PREMATURE END OF EXPRESSION (*)	Spiegazione Espressione troncata; ad esempio espressione che termina con un operatore aritmetico. Severity code Ø8 Sviluppo L'analisi della frase è interrotta.
AØ28 ARITHMETIC OVERFLOW IN EXPRESSION EVALUATION (*)	Spiegazione Overflow nel calcolo dell'espressione: il risultato intermedio non è compreso fra $2^{31} - 1$ e 2^{31}. Severity code Ø8 Sviluppo L'analisi della frase prosegue.
AØ29 FINAL RESULT OF EXPRESSION OUT OF RANGE (*)	Spiegazione Risultato dell'espressione negativo o comunque superiore a $2^{24}-1$. Severity code Ø8 Sviluppo L'analisi sintattica e semantica della frase sorgente prosegue.

AØ3Ø RELOCATABILITY ERROR (*)	Spiegazione Non è possibile valutare l'attributo di rilocabilità dell'espressione perchè più di un termine rilocabile risulta spaiato oppure il termine spaiato è preceduto dall'operatore. - . Severity code Ø8 Sviluppo L'espressione non è valorizzata; l'analisi della frase prosegue.
AØ31 FIRST OPERAND OF DIV./MULT. IS RELOCATABLE (*)	Spiegazione Espressione in cui compare un termine rilocabile come primo operando di una divisione o di una moltiplicazione. Severity code Ø8 Sviluppo L'espressione non è valorizzata; l'analisi della frase prosegue.
AØ32 SECOND OPERAND OF DIV./MULT. IS RELOCATABLE (*)	Spiegazione Espressione in cui il secondo operando di una divisione o moltiplicazione è rilocabile. Severity code Ø8 Sviluppo L'espressione non è valorizzata; comunque l'analisi della frase prosegue.

AØ33 BOTH THE OPERANDS OF DIV./MULT. ARE RELOCATABLE (*)	Spiegazione Espressione in cui compare una divisione o moltiplicazione fra termini rilocabili. Severity code Ø8 Sviluppo Non è possibile calcolare il valore dell'espressione; l'analisi della frase prosegue.
AØ34 EXPRESSION START WITH OPERATOR (*)	Spiegazione Il primo termine dell'espressione è preceduto dall'operatore aritmetico. Severity code Ø8 Sviluppo L'analisi dell'espressione e della frase termina.
AØ35 DIVISION BY ZERO IN EXPRESSION (*)	Spiegazione Nell'espressione compare una divisione con valore Ø a divisore. Severity code Ø4 Sviluppo Il risultato della divisione di un termine per un altro termine con valore Ø è posto a Ø.

AØ36 UNKNOWN TYPE (*)	Spiegazione Scorretta indicazione del tipo di una frase DC/DS o in un literal. Severity code 12 Sviluppo L'analisi della frase viene interrotta; non viene generata alcuna stringa oggetto.
AØ37 DUPLICATION FACTOR ERROR (*)	Spiegazione Scorretta indicazione del fattore di duplicazione di una frase DC/DS o in un literal: il fattore di duplicazione è stato indicato con una espressione non assoluta, o comunque ha valore superiore a 65.535; in un literal è stato indicato un fattore di duplicazione uguale a zero. Severity code 12 Sviluppo Prosegue l'analisi della frase: non viene generata alcuna stringa oggetto.
AØ38 DUPLICATION FACTOR NOT ALLOWED (*)	Spiegazione Errata indicazione del fattore di duplicazione in frasi DC o DS o in literal di tipo indirizzo. Severity code 12

	Sviluppo L'analisi della frase termina; non viene generata alcuna stringa oggetto.
AØ39 S-TYPE CONSTANT IN LITERAL (*)	Spiegazione Costante indirizzo di tipo S indicata in un literal. Severity code 12 Sviluppo L'analisi della definizione del literal si interrompe; non viene generata alcuna stringa oggetto.
AØ4Ø INVALID LENGTH MODIFIER (*)	Spiegazione Errata indicazione del modificatore di lunghezza in una frase DC o DS o in un literal: il modificatore di lunghezza è stato espresso con una espressione non assoluta, o comunque ha valore fuori range rispetto al tipo di costante. Severity code 12 Sviluppo L'analisi della frase prosegue; non viene generata alcuna stringa oggetto.
AØ41 PREVIOUSLY DEFINED NAME	Spiegazione Nella frase è definito un nome già definito in una precedente frase sorgente.

	Severity code Ø8 Sviluppo Rimane valida la prima definizione del nome.
AØ42 INVALID OPERAND IN ORG STATEMENT	Spiegazione Nella frase ORG l'operando è espresso tramite una espressione non rilocabile, o una espressione il cui attributo di rilocabilità è diverso rispetto alla sezione in cui compare la frase stessa. Severity code Ø8 Sviluppo Il location counter rimane immutato; la frase è quindi inoperante.
AØ43 CONSTANT TRUNCATED-HIGH ORDER DIGITS LOST	Spiegazione I bit di ordine superiore di una costante o literal di tipo H/F sono persi poichè il campo definito è troppo piccolo per contenere l'intero valore delle costanti. Severity code Ø4 Sviluppo Viene generata una stringa oggetto per la lunghezza implicita o esplicita indicata con il valore troncato.

AØ44 INVALID CONSTANT(*)	Spiegazione Errata indicazione del valore della costante in frasi DC e DS o in literal: i caratteri indicati non sono compatibili con il tipo di costante. Severity code Ø8 Sviluppo La stringa oggetto viene generata a zeri binari se la lunghezza della costante è indicata esplicitamente altrimenti nessuna stringa viene generata.
AØ45 QUOTES NOT PAIRED IN DC/DS/LITERAL (*)	Spiegazione L'indicazione del valore costante in frasi di definizione dati o in un literal non ha l'apice di chiusura; apici non pari nelle costanti tipo C. Severity code Ø8 Sviluppo La stringa oggetto viene generata a zeri binari se la lunghezza della costante è indicata esplicitamente altrimenti nessuna stringa è generata.
AØ46 INCORRECT OR DUPLICATE KEYWORD OPERAND (*)	Spiegazione Operando a parola-chiave compare più di una volta nell'ambito di una macro-istruzione oppure parola-chiave scorretta alla posizione indicata. Severity code 12

	Sviluppo L'Assembler prosegue nell'analisi sintattica della macro-istruzione; non viene però generata alcuna espansione.
AØ47 INVALID OPERAND IN MACRO-INSTRUCTION (*)	Spiegazione Operando invalido in una macro-istruzione perchè non è riconosciuto nè come simbolo nè come termine autodefinito decimale. Severity code 12 Sviluppo L'Assembler prosegue l'analisi sintattica della frase; non viene generata alcuna espansione.
AØ48 INVALID USING STATEMENT	Spiegazione Nella frase USING è stato indicato come registro base il registro generale Ø non in prima posizione. Severity code Ø8 Sviluppo L'analisi e l'esecuzione della frase si interrompe.

Errori durante l'esecuzione del modulo Assembler

Questo paragrafo descrive le condizioni di Abort di programma che si possono verificare durante l'esecuzione di un modulo Assembler lanciata da una CALL.

CODICE DI ERRORE	DESCRIZIONE
ABEND 1	Overflow di stack.
ABEND 2	Execute di una execute.
ABEND 4	Indirizzamento fuori memoria.
ABEND 5	Formato istruzione errato degli operandi. Istruzione RLSEM di segmento non attivo.
ABEND 8	Errore imputabile a hardware.
ABEND 9	Underflow di stack.
ABEND 11	Opzione inesistente.
ABEND 12	Il floppy disk di sistema è danneggiato.
ABEND 13	Errore di allineamento.
ABEND 14	Codice operativo inesistente.

Il formato di questi messaggi di errore è il seguente:

ABEND XX IN module-name

dove:

module-name indica il nome del modulo Assembler.

B. FORMATO DEI DATI IN MEMORIA

I dati trattati dalla macchina si suddividono in due gruppi:

- dati numerici;
- dati alfanumerici (stringa).

Formato numerico

I dati numerici sono espressi in virgola mobile (floating point), a singola o doppia precisicne, da un numero della forma:

$$N*10^{E}$$

dove:

$$N = \begin{cases} \left\{\begin{matrix}+\\-\\ \emptyset\end{matrix}\right\} \text{ X.YYYYY} & \text{in S.P.} \\ \left\{\begin{matrix}+\\-\\ \emptyset\end{matrix}\right\} \text{ X.YYYYYYYYYYY} & \text{in D.P.} \end{cases}$$

con 1 X 9 e Ø Y 9;

$$E = \begin{cases} \text{intero compreso tra -63 e +63} & \text{in S.P.} \\ \text{intero compreso tra -99 e +99} & \text{in D.P.} \end{cases}$$

Esempi:

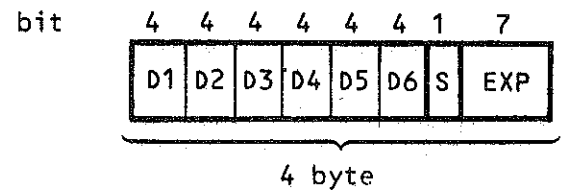
$123(SP) = 1.23000*10^{2}$

$-30,4795(DP) = -3.047950000000*10^{1}$

$0,00009437(SP) = 9.43700*10^{-5}$

Il formato dei dati numerici in memoria dipende dalla precisione dei dati stessi.

Singola precisione



dove:

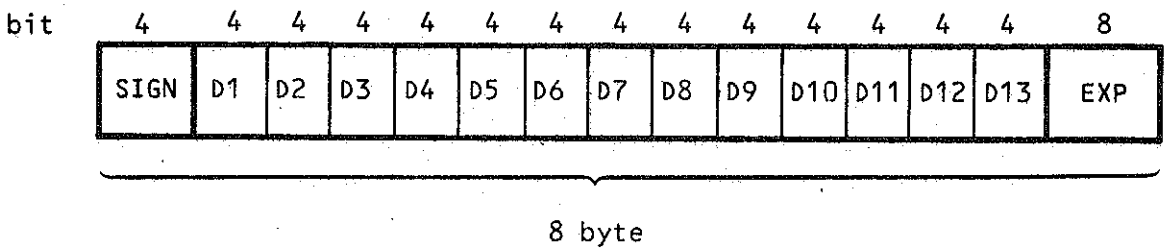
D1 ÷ D6 = cifre del dato numerico (Ø ÷ 9);

S = segno (+ = 1; - = Ø);

EXP = valore dell'esponente (espresso in esadecimale):

decimale	esadecimale
+63	7F
+62	7E
.	.
.	.
+ 1	41
Ø	4Ø
- 1	3F
.	.
.	.
-62	Ø2
-63	Ø1

Doppia precisione



dove:

SIGN = segno in esadecimale (+ = B; - = D);

D1 ÷ D13 = cifre del dato numerico (Ø ÷ 9);

EXP = valore dell'esponente (espresso in esadecimale):

decimale	esadecimale
+99	E3
+98	E2
⋮	⋮
+ 1	81
Ø	8Ø
- 1	7F
⋮	⋮
-98	1E
-99	1D

Note

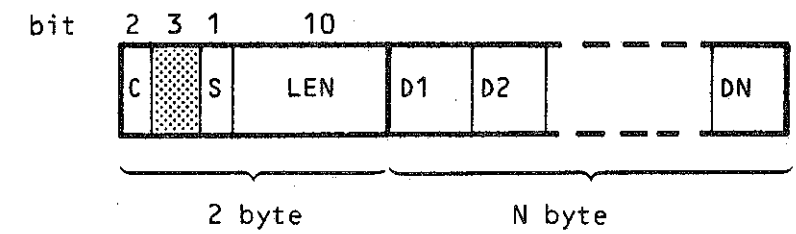
1) Se il dato numerico non è definito, il valore dello esponente è Ø.

2) Se il dato numerico è una costante = Ø, il formato relativo è il seguente:

Singola precisione	Doppia precisione
D1 = Ø S = 1 (+) EXP = Ø1 (-63)	D1 = Ø SIGN = B (+) EXP = 1D (-99)

Formato stringa

Il formato dei dati alfanumerici in memoria è il seguente:



dove:

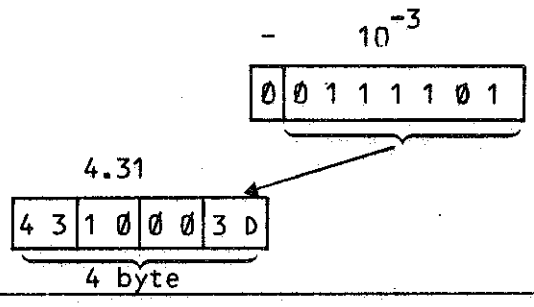
C = 11;

S = definitezza (Ø_ = stringa non assegnata; 1 = stringa assegnata);

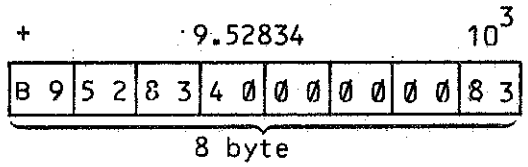
LEN = lunghezza attuale della stringa in byte (ovvero, numero di caratteri validi);

D1 ÷ DN = elementi della stringa (ovvero codici ISO dei caratteri) ognuno di 1 byte.

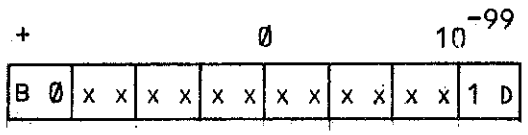
Esempi



- 0,00431 SP

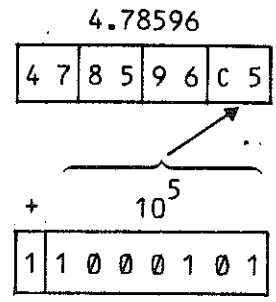


9528,34 DP

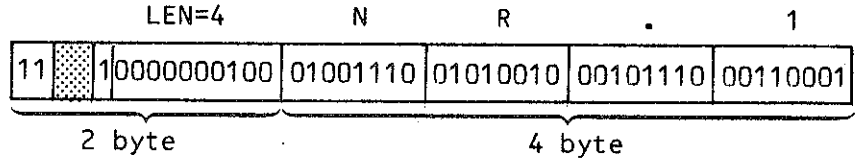


cost=Ø DP

+478596 SP



Stringa assegnata: NR.1



C. ACCESSO DIRETTO AI DATI IN MEMORIA DA MODULO ASSEMBLER

Dati accessibili al modulo Assembler

Il modulo Assembler può accedere direttamente ai dati contenuti nella memoria di sistema.
Le classi di dati cui il modulo può accedere sono le seguenti:

- variabili numeriche semplici;
- variabili numeriche multiple;
- variabili stringa semplici;
- variabili stringa multiple.

I nomi e le caratteristiche (ovvero, i descrittori)di ciascun elemento di ogni classe di dati sono organizzati nelle seguenti tabelle di memoria:

NOME DELLA TABELLA	CONTENUTO DELLA TABELLA
Numeric Variable Symbol Table (NVST)	Nomi delle var. num. semplici
Numeric Array Symbol Table (NAST)	Nomi delle var. num. mult. ed indirizzi del 1° elem. di ogni var.
String Variable Symbol (SVST)	Nomi delle var. stringa semplici e indirizzi assoluti di ogni var.
String Array Symbol Table (SAST)	Nomi delle var. stringa mult. ed indirizzi del 1° elem. di ogni var.

Gli indirizzi d'inizio di queste tabelle sono contenuti parte in ZRM e parte in COMAREA:

INDIRIZZO	ZONA DI MEMORIA	NOTA
NVSTA	COMAREA	A = Address (indirizzo)
NASTA SVSTA SASTA	ZRM	

Per accedere a tali indirizzi è quindi necessario che le relative zone di memoria (ZRM, COMAREA) siano definite all'assemblatore. Tale esigenza è soddisfatta con le istruzioni:

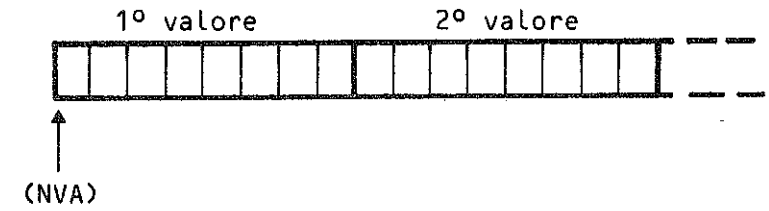
```
COPY ZRM
COPY COMARE.
```

Durante la ricerca di un elemento di una tabella, non si deve sconfinare dalla tabella stessa. Per rispettare questa esigenza, è necessario conoscere il numero di elementi che compongono ogni tabella.
Questi numeri sono contenuti in COMAREA:

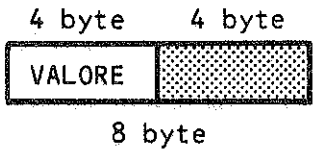
NUMERO DI ELEMENTI	NOTA
NVSTUI NASTUI SVSTUI SASTUI	UI = Used Items (elementi utilizzati)

La tabella dei descrittori delle variabili numeriche semplici, a differenza delle altre tabelle, fornisce soltanto il nome delle variabili, non ne fornisce lo indirizzo di allocazione.
I valori delle variabili numeriche semplici sono organizzati in memoria a partire da un indirizzo contenuto nel campo NVA (Numeric Variable Address) della ZRM.

Ogni valore occupa 8 byte, indipendentemente dal tipo di precisione (singola o doppia):

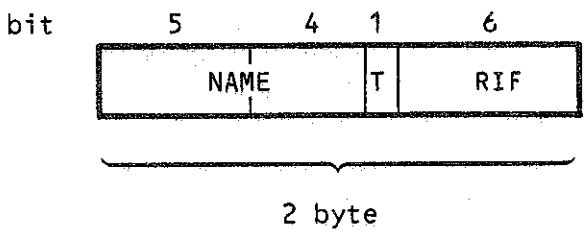


Se il valore è in singola precisione, è rappresentato (vedi Appendice B) sui primi 4 byte; i rimanenti 4 byte non sono significativi:



Variabili numeriche semplici (A.....Z, AØ.....Z9)

La tabella NVST è formata da tanti campi di 2 byte ciascuno, quante sono le variabili numeriche semplici (cioè NVSTUI). Ogni campo è organizzato come segue:



dove:

NAME = nome della variabile (lettera =I 5 bit meno sign. del corrispondente codice ISO; cifra =: 4 bit meno sign. del corrispondente codice ISO);

Esempi:

		NAME
lettera F =	0 1 0	0 0 1 1 0
lettera X =	0 1 0	1 0 1 1 1
lettera N =	0 1 0	0 1 1 1 1

Parte di codice comune a tutte le lettere ISO

		NAME
cifra 1 =	0 0 1 1	0 0 0 1
cifra 4 =	0 0 1 1	0 1 0 0
cifra 9 =	0 0 1 1	1 0 0 1

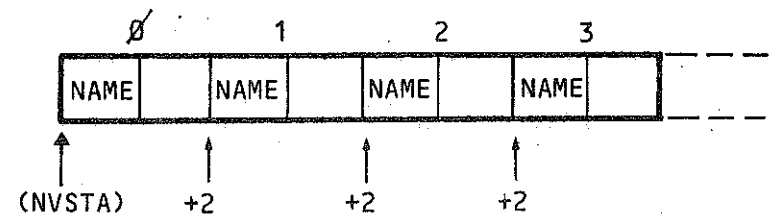
Parte di codice comune a tutte le cifre ISO.

T = precisione (singola = Ø; doppia = 1);

RIF = contatore dei riferimenti (quante volte la variabile viene riferita nel programma).

La ricerca di una variabile numerica semplice di nome noto si articola nelle seguenti fasi:

1) In NVSTA è contenuto l'indirizzo del nome della prima variabile semplice; i nomi delle variabili successive si ottengono incrementando ogni volta di 2 l'indirizzo precedente:



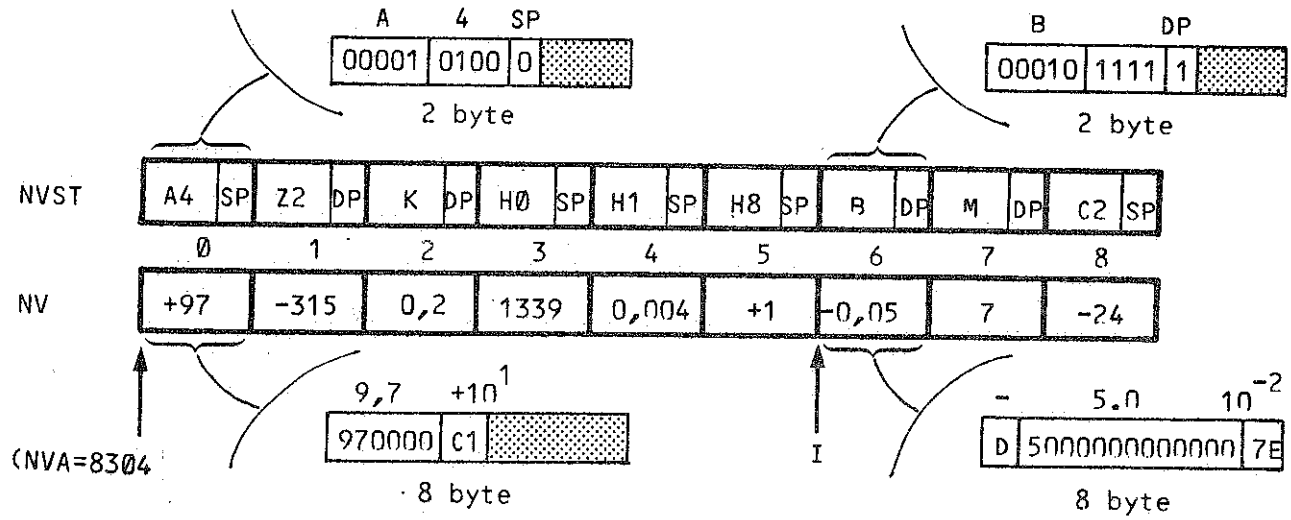
2) Trovata la variabile desiderata, se ne ottiene di conseguenza il numero d'ordine K nell'ambito della relativa tabella (NVST); si noti che le variabili sono numerate a partire da Ø;

3) L'indirizzo di allocazione del valore della variabile si ottiene sommando il seguente scostamento d all'indirizzo contenuto in NVA:

$$d = K * 8$$

Esempio

Data la seguente situazione schematica di memoria:



reperire il valore della variabile B:

- numero d'ordine di B (K = 6): si ricava attraverso opportune istruzioni di ricerca in tabella (vedi P6066 Linguaggio Assembler - GR code 3978440 Z);
- scostamento: d = 6 * 8;
- indirizzo di allocazione del valore di B:

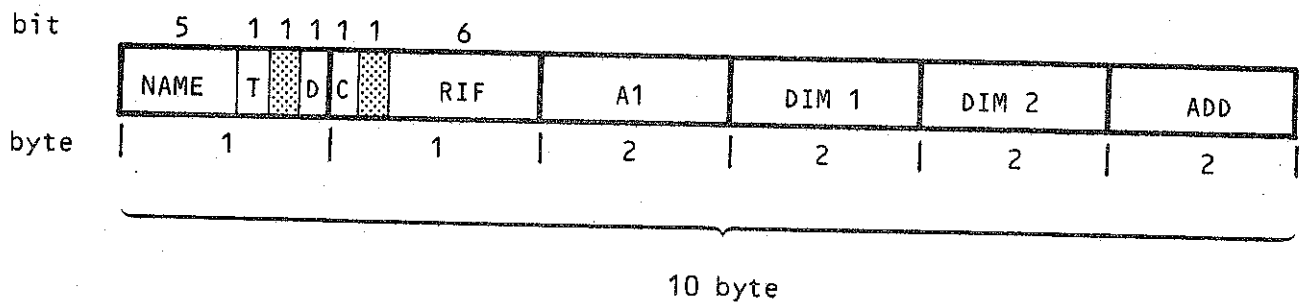
$$I = (NVA)+d.$$

Nota

Per il formato dei valori delle variabili, si veda la Appendice B.

Variabili numeriche multiple (A.....Z)

La tabella NAST è formata da tanti campi di 10 byte ciascuno, quante sono le variabili numeriche multiple (cioè NASTUI).
Ogni campo è organizzato nel modo seguente:



dove:

NAME = nome della variabile (i 5 bit meno sign. del corrispondente codice ISO);

T = precisione (singola = Ø; doppia = 1);

D = dimensioni (1 dimensione = Ø; 2 dimensioni = 1);

C = se la variabile è comune è C=1; altrimenti è C=Ø;

RIF = contatore dei riferimenti;

A1 = numero di elementi meno uno della variabile

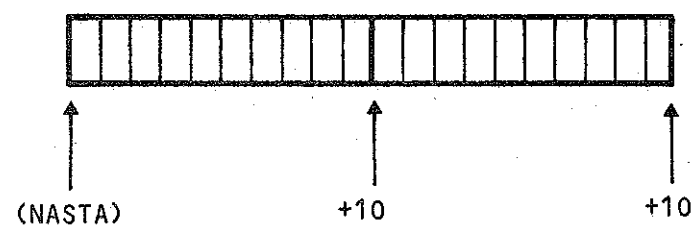
DIM1 = numero di righe meno una della variabile (se è a 2 dimensioni) oppure zero (se è ad 1 dimensione);

DIM2 = numero di colonne meno una della variabile (se è a 2 dimensioni) oppure numero di elementi della variabile (se è ad 1 dimensione);

ADD = indirizzo assoluto di parola del primo elemento della variabile.

La ricerca dell'elemento (i, j) di una variabile numerica multipla di nome noto si articola nelle seguenti fasi:

1) In NASTA è contenuto l'indirizzo del nome della prima variabile numerica multipla; i nomi delle variabili successive si ottengono incrementando ogni volta di 10 l'indirizzo precedente:



2) Trovata la variabile desiderata, se ne preleva lo indirizzo assoluto di parola ADD, incrementando di 8 l'indirizzo del nome della variabile;

3) L'indirizzo di allocazione del primo elemento della variabile si ricava moltiplicando per 4 l'indirizzo ADD;

4) L'indirizzo di allocazione dell'elemento (i,j) desiderato si ricava sommando all'indirizzo del primo elemento il seguente scostamento d:

$$d = \left[(j-1)*(DIM1+1)+i-1\right] * \begin{cases} 4 \text{ in S.P.} \\ 8 \text{ in D.P.} \end{cases}$$

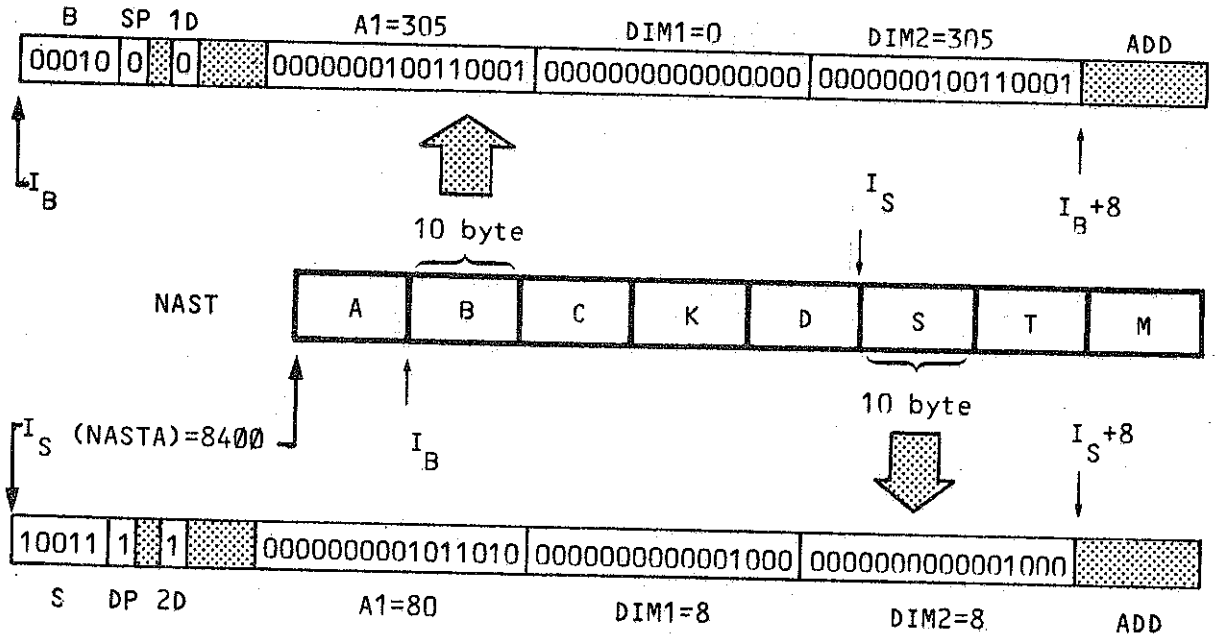
dove:

i = indice di riga;

DIM1+1 = numero di righe (= 1 nel caso di variabile ad 1 dimensione);

j = indice di colonna (= indice degli elementi nel caso di variabile ad 1 dimensione).

Esempio.
Data la seguente situazione schematica di memoria:



reperire l'elemento (12) della variabile ad una dimensione B (di 306 elementi) e l'elemento (4, 3) della variabile a due dimensioni S (di 9 x 9 elementi).
Variabile B:

- indirizzo del descrittore di B: si ricava attraverso ricerca per incrementi successivi di 10 dell'indirizzo contenuto in NASTA; nel caso specifico è:

  I_B = (NASTA)+10;

- prelievo indirizzo di parola del primo elemento di B:

  (ADD) = I_B + 8;

- indirizzo di allocazione del 1° elemento di B:

  I_{B1} = (ADD)*4;

- indirizzo di allocazione dell'elemento (12) di B:

  $I_{B12} = I_{B1} + d = I_{B1} + \left[(j-1)*(DIM1+1)+i-1\right]*4 =$

  $= I_{B1} + \left[(12-1)*(0+1)+1-1\right]*4 = I_{B1}+44$

Variabile S:

- indirizzo del descrittore di S: si ricava attraverso ricerca per incrementi successivi di 10 dell'indirizzo contenuto in NASTA; nel caso specifico è:

  I_S = (NASTA)+10+10+10+10+10 = (NASTA)+50;

- prelievo indirizzo di parola del primo elemento di S:

  (ADD) = I_S+8;

- indirizzo di allocazione del 1° elemento di S:

  $I_{S1,1}$ = (ADD)*4

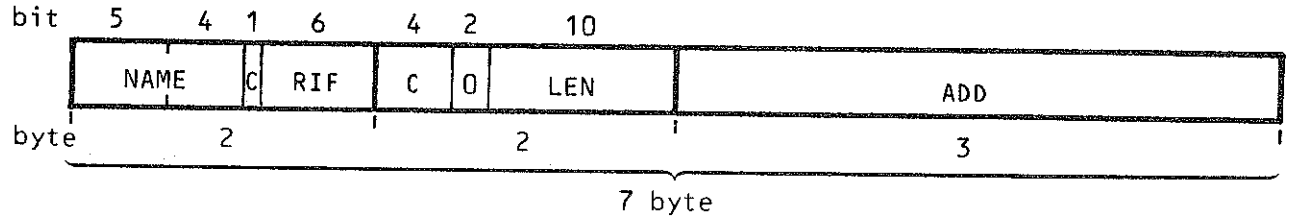
- indirizzo di allocazione dell'elemento (4,3) di S:

  $I_{S4,3} = I_{S1,1} + d = I_{S1,1} + \left[(j-1)*(DIM1+\ )+1-1\right]*8 =$

  $= I_{S1,1} + \left[(3-1)*(8+1)+4-1\right]*8 = I_{S1,1}+168.$

Variabili stringa semplici (A$...Z$, AØ$...Z9$)

La tabella SVST è formata da tanti campi di 7 byte ciascuno, quante sono le variabili stringa semplici (cioè SVSTUI).
Ogni campo è organizzato come segue:



dove:

NAME = nome della variabile (lettera = i 5 bit meno sign. del corrispondente codice ISO; cifra = i 4 bit meno sign. del corrispondente codice ISO);

C = se la variabile è comune è C=1, altrimenti è C=Ø;

RIF = contatore dei riferimenti;

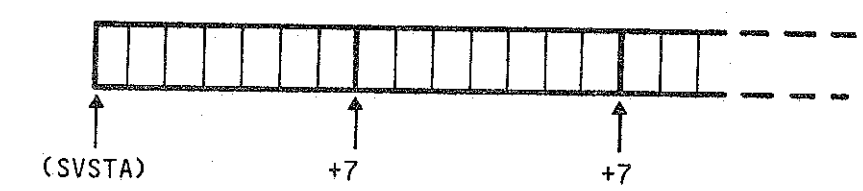
CØ = 11ØØØØ

LEN = lunghezza di allocazione della variabile (1 ÷ 1023), definita da DIM;

ADD = indirizzo assoluto della variabile.

La ricerca di una variabile stringa semplice di nome noto si articola nelle seguenti fasi:

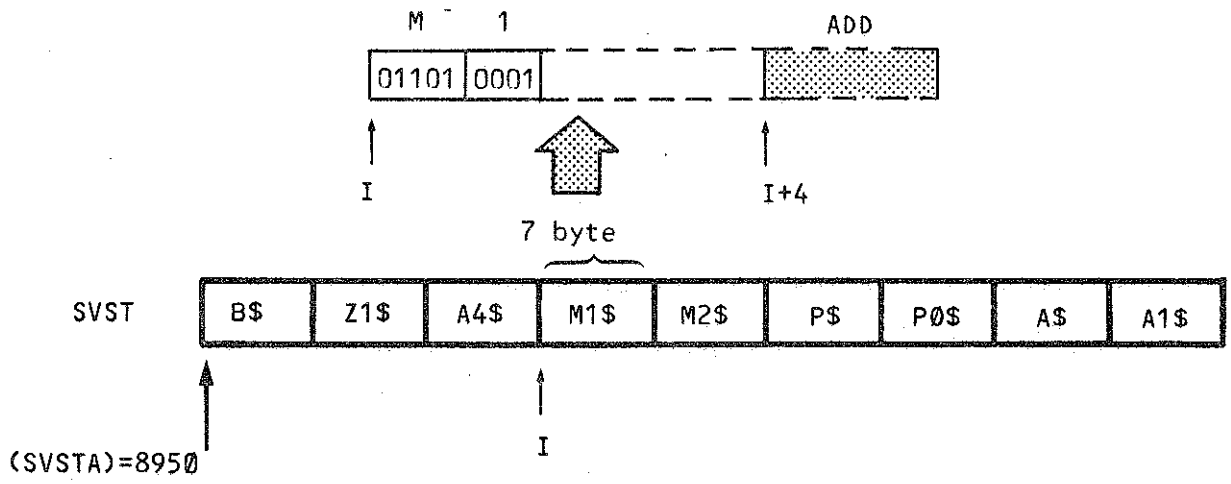
1) In SVSTA è contenuto l'indirizzo del nome della prima variabile stringa semplice; i nomi delle variabili successive si ottengono incrementando ogni volta di 7 l'indirizzo precedente:



2) Trovata la variabile desiderata, se ne preleva l'indirizzo assoluto ADD incrementando di 4 l'indirizzo del nome della variabile;

3) L'indirizzo di allocazione della variabile si ricava incrementando di 2 l'indirizzo assoluto ADD.

Esempio.
Data la seguente situazione schematica di memoria:



reperire il contenuto della variabile M1$:

- indirizzo del descrittore di M1$: si ricava attraverso ricerca per incrementi successivi di 8 dello indirizzo contenuto in SVSTA; nel caso specifico è:

  I = (SVSTA)+7+7+7 = (SVSTA)+21;

- prelievo dell'indirizzo assoluto di M1$:

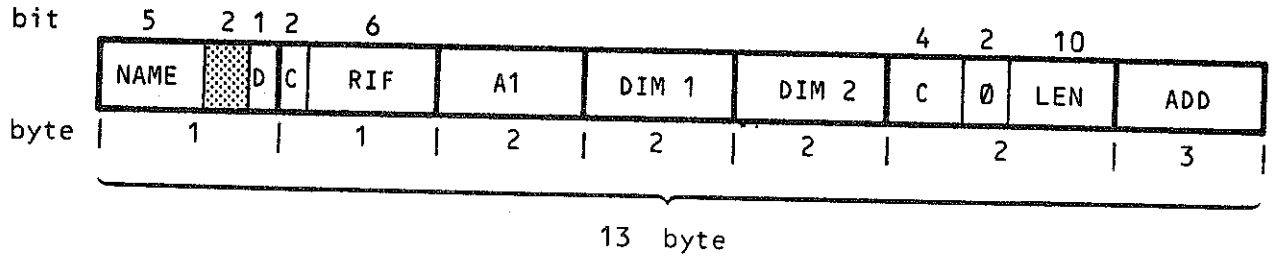
  (ADD) = I+4;

- indirizzo di allocazione del valore di M1$:

  $I_{M1\$}$ = (ADD) +2.

Variabili stringa multiple (A$....Z$)

La tabella SAST è formata da tanti campi di 13 byte ciascuno, quante sono le variabili stringa multiple (cioè SASTUI).
Ogni campo è organizzato come segue:



dove:

NAME = nome della variabile (i 5 bit meno sign. del corrispondente codice ISO);

D = dimensioni (1 dimensione = Ø; 2 dimensioni = 1);

C = se la variabile è comune è C=1, altrimenti è C=Ø;

RIF = contatore dei riferimenti;

A1 = numero di elementi meno uno della variabile;

DIM1 = numero di righe meno una della variabile (se è a 2 dimensioni) oppure zero (se è ad una dimensione);

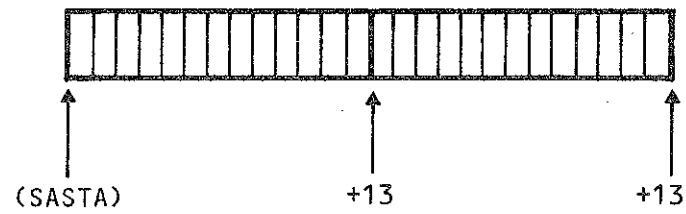
DIM2 = numero di colonne meno una della variabile (se è a 2 dimensioni) oppure numero di elementi della variabile (se è ad una dimensione);

LEN = lunghezza di allocazione dell'elemento (definita da DCL);

ADD = indirizzo assoluto del primo elemento della variabile.

La ricerca dell'elemento (i,j) di una variabile stringa multipla di nome noto si articola nelle seguenti fasi:

1) In SASTA è contenuto l'indirizzo del nome della prima variabile stringa multipla; i nomi delle variabili successive si ottengono incrementando ogni volta di 13 l'indirizzo precedente:



2) Trovata la variabile desiderata, si preleva l'indirizzo assoluto ADD del suo primo elemento, incrementando di 10 l'indirizzo del nome della variabile;

3) L'indirizzo di allocazione dell'elemento (i,j) desiderato si ricava sommando all'indirizzo del primo elemento il seguente scostamento d:

$$d = \left[(j-1)*(DIM1+1)+i-1\right]*(2+LEN)$$

dove:

i = indice di riga;

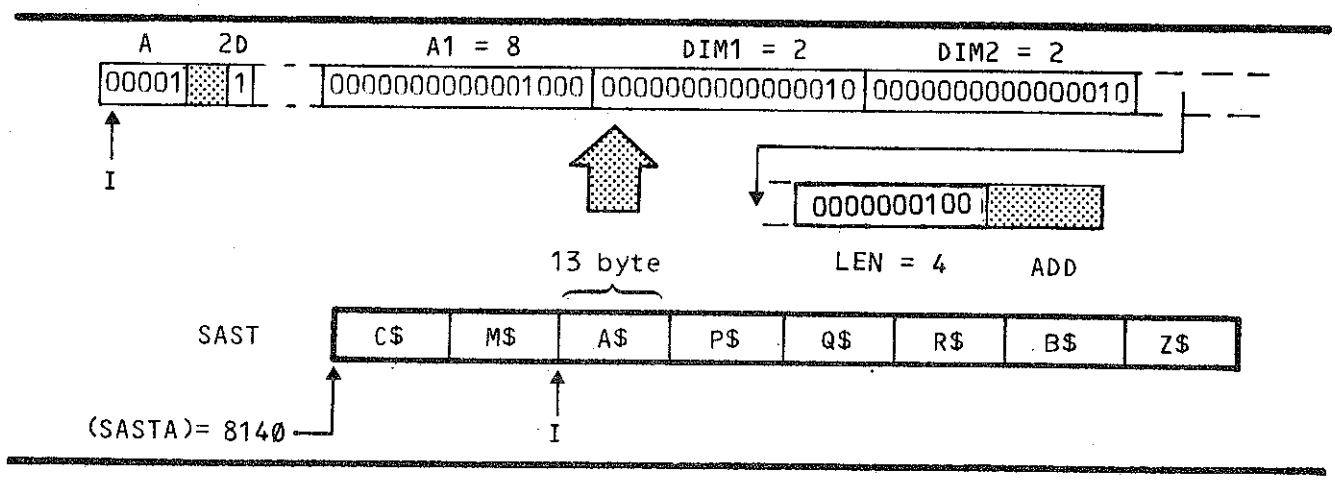
DIM1+1 = numero di righe (= 1 nel caso di variabile ad 1 dimensione);

j = indice di colonna (= indice degli elementi nel caso di variabile ad 1 dimensione);

LEN = lunghezza di allocazione dell'elemento (definita da DCL).

Esempio.

Data la seguente situazione schematica di memoria:



reperire l'elemento (2, 3) della variabile A$ (di 3x3 elementi di 4 byte ciascuno):

- indirizzo del descrittore di A$: si ricava attraverso ricerca per incrementi successivi di 13 dell'indirizzo contenuto in SASTA; nel caso specifico è:

  I = (SASTA)+ 13 + 13 = (SASTA)+ 26;

- prelievo dell'indirizzo assoluto del 1° elemento di A$:

  (ADD) = I + 10;

- indirizzo di allocazione dell'elemento (2, 3) di A$:

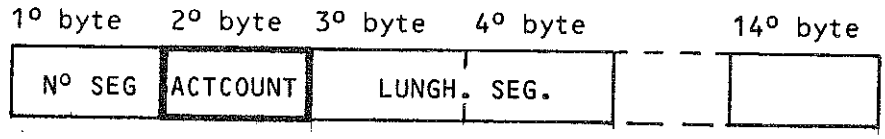
$$I_{A\$} = (ADD)+\left[(j-1)*(DIM1+1)+i-1\right]* (2+LEN) =$$

$$= (ADD)+\left[(3-1)*(2+1)+2-1\right]* (2+4) =$$

$$= (ADD)+ 42$$

D. COME RENDERE RESIDENTE UN MODULO ASSEMBLER

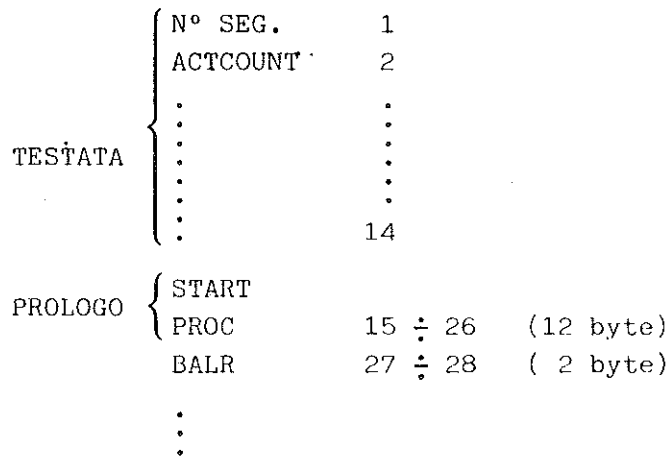
Ogni modulo Assembler chiamato da una istruzione CALL viene attivato all'atto della chiamata e viene disattivato al termine del modulo, quando il controllo ritorna al programma.
Ogni modulo Assembler inizia con una testata di 14 byte; il secondo byte della testata (ACTCOUNT) è utilizzato per indicare se il modulo è attivo ($\neq \emptyset$) o disattivo ($= \emptyset$):



- l'attivazione del modulo è frutto dell' incremento dell'ACTCOUNT di una unità;

- la disattivazione del modulo è frutto del decremento dell'ACTCOUNT di una unità.

Per rendere residente in memoria un modulo Assembler è pertanto sufficiente incrementare ulteriormente di 1 il contenuto dell'ACTCOUNT, mentre per rilasciare un modulo residente è sufficiente decrementare ulteriormente di 1 il contenuto dell'ACTCOUNT.
Per accedere all'ACTCOUNT è necessario conoscere l'indirizzo di inizio modulo, caricato nel registro base. Il registro base viene caricato dopo il prologo attraverso l'istruzione BALR.
Consideriamo pertanto l'inizio di un modulo Assembler e la relativa occupazione di memoria (in byte):



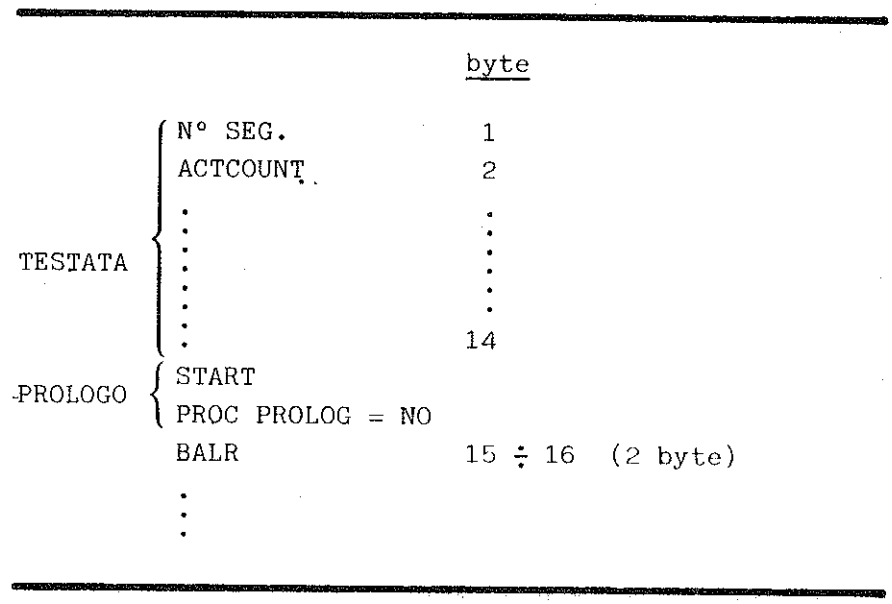
Quando viene caricato il registro base, lo scostamento rispetto all'inizio del modulo è di 28 byte; per individuare l' ACTCOUNT è pertanto sufficiente sottrarre 27 al contenuto del registro base.

Note

1) Il contenuto del registro base deve rimanere costante per tutta l'esecuzione del modulo in quanto tutti gli indirizzi sono calcolati rispetto a tale registro.
   Per eseguire la sottrazione, si consiglia pertanto di utilizzare un altro registro.

2) La disattivazione di un modulo residente è indispensabile quando si desidera utilizzare una routine diversa: poichè l'area di overlay può contenere una sola routine attiva alla volta, se il modulo residente non venisse disattivato, ogni chiamata a modulo diverso avrebbe come risultato l' esecuzione del modulo residente.

3) Nel caso non si desideri l'allocazione dell'area locale (vedi paragrafo Prologo, capitolo 2), quando viene caricato il registro base lo scostamento è di 16 byte anzichè 28; per individuare l'ACTCOUNT, si deve in tale caso sottrarre 15 anzichè 27:


byte
TESTATA
N° SEG.
1
ACTCOUNT
2
14
PROLOGO
START
PROC PROLOG = NO
BALR
15 ÷ 16 (2 byte)

E. ESEMPIO APPLICATIVO DI PROGRAMMA BASIC COMPLETO DI MODULO ASSEMBLER

Il seguente programma consente di trattare una stringa non superiore a 256 crt in uno dei seguenti modi:

. FUNZIONE Ø - Trasformare tutti i crt della stringa in crt NUL.

. FUNZIONE 1 - Sondare se un bit selezionato della stringa è uguale a 1.

. FUNZIONE 2 - Settare un bit selezionato della stringa.

. FUNZIONE 3 - Resettare un bit selezionato della stringa.

. FUNZIONE 4 - Cancellare (DELETE) tutti i crt della stringa.

Il programma impiega un modulo Assembler di nome "ASMOBJ" per elaborare le funzioni suddette.
I listing che corredano questo esempio sono autocommentati.

```
0010 DCL 256(A$,B$)
0020 DCL 25(C$)
0030 LET K=0
0040 LET C$="FUNZIONE SELEZIONATA ( "
0050 LET A$=""
0060 DISP "SELEZIONA FUNZIONE(I) E BIT(J)",,
0070 INPUT I,J
0080 DISP "INTRODUCI STRINGA",,
0090 INPUT A$
0100 LET B$=A$
0110 CALL "ASMOBJ",I,J,K,A$
0120 ON I+1 GOTO 130,170,260,350,440
0130 PRINT C$;I;" ): STRINGA NUL"
0140 GOSUB 500
0150 PRINT
0160 GOTO 460
0170 PRINT C$;I;" ): TEST BIT";J;"= 1"
0180 GOSUB 500
0190 ON K+1 GOTO 200,230
0200 PRINT "IL BIT SELEZIONATO (";J;" ) NON E' A 1"
0210 PRINT
0220 GOTO 460
0230 PRINT "IL BIT SELEZIONATO (";J;" ) E' A 1"
0240 PRINT
0250 GOTO 460
0260 PRINT C$;I;" ): SET BIT";J
0270 GOSUB 500
0280 ON K+1 GOTO 290,320
0290 PRINT "IL BIT";J;" E' STATO SETTATO"
0300 PRINT
0310 GOTO 460
0320 PRINT "IL BIT";J;" ERA GIA' SETTATO"
0330 PRINT
0340 GOTO 460
0350 PRINT C$;I;" ):RESET BIT";J
0360 GOSUB 500
0370 ON K+1 GOTO 380,410
0380 PRINT "IL BIT";J;" ERA GIA' RESETTATO"
0390 PRINT
0400 GOTO 460
0410 PRINT "IL BIT";J;" E' STATO RESETTATO"
0420 PRINT
0430 GOTO 460
0440 PRINT C$;I;" ): STRINGA DELETE"
0450 GOSUB 500
0455 PRINT
0460 DISP "NUOVA FUNZIONE? SI=0",,
0470 INPUT F
0480 IF F=0 THEN 60
0490 GOTO 530
0500 PRINT "STRINGA INTRODOTTA:";B$
0510 PRINT "STRINGA RITORNATA :";A$
0520 RETURN
0530 END
```

```
0010 ASMOD1    START                    * MODULO ASSEMBLER PER FUNZ. STRINGA
0020           PROC
0030 R0        EQU 0
0040 R1        EQU 1
0050 R2        EQU 2
0060 R3        EQU 3
0070 R4        EQU 4
0080 R5        EQU 5
0090 R6        EQU 6
0100 R7        EQU 7
0110 R8        EQU 8
0120 R9        EQU 9
0130           BALR R3,0
0140           USING *,R3
0150           USING LOCAL,13
0155 *                        ** PRELIEVO NUMERO ARGOMENTI **
0160           LA R7,0
0170           GSA R4,R7
0175           SRL R4,16
0176           SRL R4,8
0177 *                     ** CONTROLLO NUM. ARGOMENTI CORRETTO **
0180           CH R4,=H'5'
0190           BNE RIENTRO
0195 *                         ** PRELIEVO INDIRIZZO DI I **
0200           LA R4,4
0210           GSA R4,R4
0215 *                         ** PRELIEVO INDIRIZZO DI J **
0220           LA R5,3
0230           GSA R5,R5
0235 *                         ** PRELIEVO INDIRIZZO DI K **
0240           LA R6,2
0250           GSA R6,R6
0255 *                ** PRELIEVO INDIRIZZO DESCRITTORE STRINGA A$ **
0260           LA R7,1
0270           GSA R7,R7
0275 *                        ** MEMORIZZAZIONE NUM. CRT. DI A$ **
0280           MVC HALFX,0(R7)
0290           NI HALFX,X'03'
0300           MVC IND(3),2(R7)
0310           L R7,FULL                * R7 PUNTA ALLA STRINGA A$
0315 *           ** CONVERSIONE CODICE FUNZIONE (I) E BIT DA OPERARE (J) **
0320           CMST CAMPO,0(R4)
0330           CSBH HALFI,CAMPO+1(6)
0340           CMST CAMPO,0(R5)
0350           CSBH HALFJ,CAMPO+1(6)
0360           MVC 0(8,R6),ZERO         * AZZERAMENTO DI K
0365 *                           ** CONTROLLO FUNZIONE 0 **
0370           CLI HALFI+1,X'00'
0380           BNE ALTRFUN
0390           LA R9,BZERI              * R9 PUNTA UN BYTE CON TUTTI I BIT=0
0400 TAG1      LH R8,HALFX              * R8 CONTIENE IL NUM. DI CRT DI A$
0405 *                        ** LOOP SOSTITUZIONE CRT STRINGA **
0410 LOOP      MVC 2(1,R7),0(R9)
0420           ALRI R7,1
0430           BCT R8,LOOP
0440           B RIENTRO
0450 ALTRFUN   LA R9,BUNI               * R9 PUNTA UN BYTE CON TUTTI I BIT=1
0455 *                           ** CONTROLLO FUNZIONE 4 **
0460           CLI HALFI+1,X'04'
0470           BE TAG1
0480           LH R8,HALFJ
0490           SLRI R8,1
0500           LR R9,R8
```

(segue)

```
0510           SRL R8,3
0520           AR R7,R8
0530           LA R8,7
0540           NR R9,R8
0570           LA R8,BTAB
0580           IC R8,0(R8,R9)
0590           EX R8,TESTBIT
0600           BC 8,GOON1
0610           MVC 0(8,R6),UNO
0620 GOON1     EQU *
0625 *                          ** CONTROLLO FUNZIONE 1 **
0630           CLI HALFI+1,X'01'
0640           BE RIENTRO
0645 *                          ** CONTROLLO FUNZIONE 2 **
0650           CLI HALFI+1,X'02'
0660           BE GOON2
0665 *                          ** CONTROLLO FUNZIONE 3 **
0670           CLI HALFI+1,X'03'
0680           BNE ERRORE2
0690           LA R8,ANDTAB
0700           IC R8,0(R8,R9)
0710           EX R8,RESET
0720           B RIENTRO
0730 GOON2     EX R8,SET
0740           B RIENTRO
0750 ERRORE2   MVC 0(8,R6),DUE
0760 RIENTRO   EQU *
0770           PRRET
0780           ORG *-2
0790           BR 14
0800 BZERI     DC X'00'                    * BYTE CON TUTTI I BIT=0
0810 BUNI      DC X'FF'                    * BYTE CON TUTTI I BIT=1
0820 ZERO      DC X'B000000000000001D'     * 0 IN DOPPIA PRECISIONE
0830 UNO       DC X'B100000000000080'      * 1 IN DOPPIA PRECISIONE
0840 DUE       DC X'B200000000000080'      * 2 IN DOPPIA PRECISIONE
0850 BTAB      DC X'8040201008040201'
0860 ANDTAB    DC X'7FBFDFEFF7FBFDFE'
0870           DS 0H
0880 TESTBIT   TM   2(R7),X'00'
0890 RESET     NI   2(R7),X'00'
0900 SET       OI   2(R7),X'00'
0905 *                       ** DEFINIZIONE AREA LOCALE **
0910 LOCAL     DSECT
0920 OLD13     DS F
0930 FULL      DS 0F
0940           DS CL1
0950 IND       DS CL3
0960 HALFI     DS H
0970 HALFJ     DS H
0980 HALFX     DS H
0990 CAMPO     DS CL7
1000 ENDLOCAL  DS 0F
```

```
RUN
**** FORMALLY CORRECT PROGRAM ****
SELEZIONA FUNZIONE(I) E BIT(J)
0,0
INTRODUCI STRINGA
1234567890QWERTYUIOPASDFGHJKL\ZXCVBNM
FUNZIONE SELEZIONATA ( 0  ): STRINGA NUL
STRINGA INTRODOTTA:1234567890QWERTYUIOPASDFGHJKL\ZXCVBNM
STRINGA RITORNATA :■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■

NUOVA FUNZIONE? SI=0
0
SELEZIONA FUNZIONE(I) E BIT(J)
1,23
INTRODUCI STRINGA
SANTO
FUNZIONE SELEZIONATA ( 1  ): TEST BIT 23 = 1
STRINGA INTRODOTTA:SANTO
STRINGA RITORNATA :SANTO
IL BIT SELEZIONATO ( 23  ) E' A 1

NUOVA FUNZIONE? SI=0
0
SELEZIONA FUNZIONE(I) E BIT(J)
1,11
INTRODUCI STRINGA
SANTO
FUNZIONE SELEZIONATA ( 1  ): TEST BIT 11 = 1
STRINGA INTRODOTTA:SANTO
STRINGA RITORNATA :SANTO
IL BIT SELEZIONATO ( 11  ) NON E' A 1

NUOVA FUNZIONE? SI=0
0
SELEZIONA FUNZIONE(I) E BIT(J)
2,14
INTRODUCI STRINGA
SANTO
FUNZIONE SELEZIONATA ( 2  ): SET BIT 14
STRINGA INTRODOTTA:SANTO
STRINGA RITORNATA :SENTO
IL BIT 14  E' STATO SETTATO

NUOVA FUNZIONE? SI=0
0
SELEZIONA FUNZIONE(I) E BIT(J)
2,30
INTRODUCI STRINGA
SANTO
FUNZIONE SELEZIONATA ( 2  ): SET BIT 30
STRINGA INTRODOTTA:SANTO
STRINGA RITORNATA :SANTO
IL BIT 30  ERA GIA' SETTATO
```

```
NUOVA FUNZIONE? SI=0
0
SELEZIONA FUNZIONE(I) E BIT(J)
3,4
INTRODUCI STRINGA
SANTO
FUNZIONE SELEZIONATA (  3  ):RESET BIT 4
STRINGA INTRODOTTA:SANTO
STRINGA RITORNATA :CANTO
IL BIT 4  E' STATO RESETTATO

NUOVA FUNZIONE? SI=0
0
SELEZIONA FUNZIONE(I) E BIT(J)
3,19
INTRODUCI STRINGA
SANTO
FUNZIONE SELEZIONATA (  3  ):RESET BIT 19
STRINGA INTRODOTTA:SANTO
STRINGA RITORNATA :SANTO
IL BIT 19  ERA GIA' RESETTATO

NUOVA FUNZIONE? SI=0
0
SELEZIONA FUNZIONE(I) E BIT(J)
4,0
INTRODUCI STRINGA
1234567890QWERTYUIOPASDFGHJKL\ZXCVBNM
FUNZIONE SELEZIONATA (  4  ): STRINGA DELETE
STRINGA INTRODOTTA:1234567890QWERTYUIOPASDFGHJKL\ZXCVBNM
STRINGA RITORNATA :
```