Texas Instruments Home Computer

Mini Memory +

Line-by-line Assembler

Manuale in Italiano

(Unofficial)

SOLID STATE SOFTWARE™ **COMMAND MODULE**

Un modulo SSS che espande il vostro TI-99/4A incrementando la capacità di memoria e potenziandolo con le seguenti funzionalità:

- *Aggiunge 4K bytes di Memoria ad Accesso Casuale (RAM) per programmi e memorizzazione dati, 4K bytes di Memoria di Sola Lettura (ROM), e 6K bytes di Memoria Grafica di Sola Lettura (GROM).*
- *Contiene sottoprogrammi e routines aggiuntivi in TI BASIC che ti permetteranno di caricare e collegare i programmi BASIC a sottoprogrammi in liguaggio assembly, accedere in modo diretto alle risorse del computer oltre che alle espansioni di memoria esterne (vendute separatamente), da un programma TI BASIC.*
- *Incorpora una batteria atta a preservare dati o programmi salvati nella memoria, anche quando lo stesso verrà rimosso dal computer o se il computer verrà spento.*
- *Consente di risolvere i problemi con i tuoi programmi in linguaggio assembly mediante lo strumento EASY BUG, il programma di debug incluso.*
- *Viene fornito con una cassetta contenente un assemblatore Line-by-Line che ti permetterà di sviluppare tuoi programmi in linguaggio assembly.*

TRADUZIONE DEL MANUALE MINIMEMORY PER HOME COMPUTER TEXAS TI99/4A

INTRODUZIONE

Il modulo di comando SSS MiniMemory incrementa la versatilita' del calcolatore fornendo memoria aggiuntiva per il sistema e importanti strumenti per lo sviluppo dei programmi. In piu' il modulo contiene una batteria interna che permette di memorizzare permanentemente dati e programmi anche quando la console e' spenta e anche se il modulo viene estratto dalla console.

Le possibilita' del modulo includono:

- Una memoria totale di 14K bytes. Questa memoria consta di 6K bytes di Graphic Read Only Memory(GROM)(memoria grafica a sola lettura, 4K bytes di Read Only Memory(ROM) e 4K bytes di RAM. I programmi residenti in GROM e in ROM forniscono strumenti aggiuntivi importanti per lo sviluppo dei programmi. La memoria RAM fornisce spazio di memoria aggiuntiva per memorizzare dati e programmi.

- Una batteria interna nel modulo per conservare i dati o i programmi memorizzati nella memoria RAM.

- Files aggiuntivi. Oltre al file RAM di 4K-bytes nel modulo stesso, un segmento di 24K-bytes dell'unita' Espansione di Memoria, se collegata, puo' essere usata dai programmi TI BASIC.

- Possibilita' di linguaggio Assembly. Con il modulo i programmi oggetto in linguaggio assembly possono essere caricati nel modulo o nell'Espansione di Memoria, se collegata.

- Programmi aggiuntivi TI BASIC. Con il modulo diversi sottoprogrammi aggiuntivi possono essere chiamati con le istruzioni TI BASIC. Questi sottoprogrammi comprendono l'abilitazione delle istruzioni PEEK e POKE.

- Programmi aggiuntivi di utilita'. Il modulo comprende diversi programmi che permettono di inserirsi nelle risorse del calcolatore; per esempio, interfacciamento tra programmi utente e programmi residenti nelle ROM e/o GROM, interfacciamento tra programmi in linguaggio assembly e interprete TI BASIC e accesso alla RAM del Video Display Processor(VDP).

- Un programma residente di debug(di correzione). Il programma EASY BUG e' uno strumento utile nello sviluppo dei programmi con cui si puo' accedere alle risorse interne del computer e manomettere i programmi.

APPLICAZIONI

Si puo' usare la RAM nel modulo MM per immagazzinare sia dati

che programmi. Questa memoria e' "una memoria CPU"cioe' significa che e' una memoria ad accesso rapido. Si puo' trarre vantaggio da cio' in programmi che fanno spesso immagazzinamento e recupero dati rapido o per immagazzinare programmi in linguaggio assembly che fanno conti rapidi.
Un programma TI BASIC che si usa frequentemente puo' essere posto nel modulo MM piuttosto che su cassetta o dischetto per un caricamento piu' veloce.
Generalmente la RAM (come la memoria "utente" nella console del computer)perde il contenuto quando la console e' spenta. Il modulo MM, avendo una propria batteria, invece mantiene il contenuto quando la console e' spenta. I programmi o i dati possono essere immagazzinati in una RAM con batteria incorporata, la console puo' essere spenta e il modulo rimosso dalla console. Quando poi si reinserisce il modulo e si accende la console i dati o il programma e' pronto all'uso.

AVVERTIMENTO

> Quando si toglie o si inserisce il modulo MM, la console deve essere spenta per evitare la possibilita' di alterare i dati o i programmi immagazzinati nella RAM.

Un'importante possibilita' del modulo MM e' la sua capacita' di implementare programmi in linguaggio assembly. Il modulo permette di caricare programmi in linguaggio assembly ad accesso diretto verso i coimponenti programmabili del computer(come il microprocessore TMS9900 o il TM9918 Video Display Processor). I programmi in linguaggio assembly possono accedere direttamente ai device esterni come i joysticks o il registratore a cassetta tramite le porte d'interfaccia in console.
In piu' il modulo rende possibile ai sottoprogrammi in linguaggio assembly di essere chiamati da programmi TI BASIC. Questi sottoprogrammi in linguaggio assembly possono fornire funzioni che in TI BASIC potrebbero essere inefficienti o impossibili. Le routines di programma residenti nelle ROM e GROM del modulo MM forniscono un'interfaccia utile tra i programmi TI BASIC e i programmi in linguaggio assembly. Inoltre e' incluso nel modulo il programma di correzione (debugging) EASY BUG con cui si puo' accedere alla memoria e ai componenti programmabili del computer. EASY BUG include inoltre comandi per caricare e immagazzinare dati a immagine della memoria su registratore a cassetta.

INFORMAZIONE GENERALE SULLA MEMORIA

NOTA: In questo manuale il segno di maggiore(>)indica che il numero che segue e' un numero esadecimale(a base 16).
Il modulo MM contiene untotale di 14K bytes di memoria consistente di 4K bytes di RAM con batteria interna, 4Kbytes di ROM e 6K bytes di GROM. In ROM e GROM sono residenti molte routines che aggiungono altri programmi chimabili da TI BASIC e che sono utili per interfacciare programmi in linguaggio assembly con programmi TI BASIC.
L'appendice A mostra l'organizzazione della memoria per l'intero spazio di memoria. I 4K bytes di ROM nel modulo MM occupano gli indirizzi di memoria da >6000 a >6FFF(o da 24576 a 28671). L'appendice B illustra i dettagli dell'organizzazione della ROM.
I 4K bytes di RAM nel modulo occupano gli indirizzi di memoria

da >7000 a >7FFF(o da 28672 a 32767). L'appendice C mostra i dettagli dell'organizzazione della RAM quando e' usata per i files TI BASIC. L'appendice D descrive come la RAM e' organizzata quando e' usata per immagazzinare il linguaggio assembly. Le GROM occupano spazio di memoria che non e' direttamente mappata nello spazio di indirizzamento della memoria CPU. L'appendici E e F contengono informazioni sulla RAM nel VDP.

ULTERIORI INFORMAZIONI

Ulteriori informazioni si trovano su:

- User's Reference Guide

- TI Extended BASIC owner's manual

- Editor/Assembler owner's manual

COME SI USA QUESTO MANUALE

Non importante.

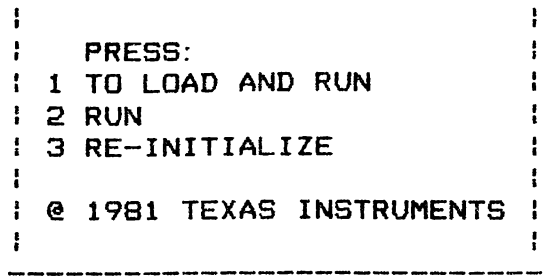
USO DEL MODULO

Prima di inserire o rimuovere il modulo MM, e' buona pratica spegnere la console del computer. Spegnendo la console si evitano 'rimbalzi di corrente' tra il modulo e i contatti della console che potrebbero causare la perdita o l'alterazione del contenuto della RAM del modulo.
Nota: occorre assicurarsi che il modulo sia scevro da cariche elettrostatiche prima di inserirlo nel computer.

1. Inserire il modulo nella fessura della console. Quando si accende il computer occorre attendere che appaia sullo schermo il titolo principale(Master Title Screen).
2. Premere un tasto qualsiasi per far apparire la lista sulla quale compariranno due opzioni : EASY BUG e MINIMEMORY. Se si sceglie EASY BUG il programma EASY BUG di correzione programmi proporra' una selezione di comandi. Se si sceglie MINIMEMORY si puo' scegliere tra caricare files, eseguire programmi o inizializzare la RAM del modulo MM. Si prema quindi il tasto corrispondente al numero posto vicino all'opzione desiderata.

Nota: Per rimuovere il modulo prima si ritorni il computer alla schermata iniziale principale premendo QUIT (FCTN 7); poi spegnere la console e rimuovere il modulo dalla fessura.

Se si sceglie MINIMEMORY dalla lista iniziale apparira' la seguente selezione:



TO LOAD AND RUN carica programmi assemblati in formato oggetto tagged o compresso(su dischetto) in memoria e li esegue.
RUN esegue i programmi precedentemente caricati in memoria.
RE-INITIALIZE reinizializza il modulo MM e lo prepara per caricare nuovi programmi(V. sotto)

LOAD AND RUN

L'opzione LOAD AND RUN permette di caricare ed eseguire programmi in liguaggio assembly sviluppati con l'Editor/Assembler e posti su dischetto. Quando si preme 1 per selezionare l'opzione LOAD AND RUN appare la segnalazione "FILE NAME?". Questo file deve contenere un programma in linguaggio assembly in formato oggetto.
Si scriva il nome del file e poi si prema ENTER; per es. scrivendo

DSK1.DEMO

e premendo ENTER si carica un file di nome DEMO da un dischetto sul Drive 1. Dopo che il file e' caricato il nome del file e' cancellato dallo schermo. Il computer e' ora pronto per accettare un altro nome di file. Si puo' caricare quanti file si vuole fino a riempire la memoria. Quando tutti i file sono stati caricati si prema ENTER(senza introdurre un nome di file)per continuare.
Quindi apparira' la segnalazione "PROGRAM NAME?". Il nome del programma e' un punto d'ingresso nel programma che e' contrassegnato da un'etichetta DEFinita in una lista DEF del programma. Premendo ENTER senza introdurre un nome di programma si crea una condizione di errore.

RUN

Se si e' precedentemente caricato un programma oggetto in linguaggio assembly, si scelga l'opzione RUN per eseguire il programma. Occorre ricordarsi che un programma caricato nel modulo MM e' conservato anche se la console e' spenta. Percio' si puo' eseguire questo programma senza ricaricarlo. SE si preme 2 quando la lista di selezione compare sullo schermo appare la segnalazione di "PROGRAM NAME?". Quindi si introduca il nome del programma. Il nome del programma e' un punto d'ingresso nel programma che appare nella tabella interna REF/DEF. Se si preme ENTER senza introdurre un nome di programma il computer localizza ed esegue il programma eseguito per ultimo.

RE-INITIALIZE

Se si preme 3 per selezionare l'opzione RE-INITIALIZE, si inizia-

lizza la RAM del modulo MM che ora accetta nuovi files. Qualsiasi programma o dato immagazzinato in RAM viene perso.
Quando si sceglie questa otzione lo schermo si ripulisce momentaneamente e poi la selezione riappare.
Se il messaggio MEMORY ALREADY INITIALIZED, HIT "PROC'D"TO CONFIRM viene visualizzato premi PROC'D se si vuol reinizializzare la memoria. La reinizializzazione cancella tutti i riferimenti ai programmi esistenti dalla memoria e la prepara per caricare nuovi programmi. Nota: premi PROC'D ONLY se si vuol caricare un nuovo programma e la memoria rimanente e' troppo piccola per aggiungere il nuovo programma.
Se non si vuol reinizializzare si prema un altro tasto per ritornare alla lista di selezione.

CARICARE E SALVARE FILES DATI TI BASIC

Probabilmente l'applicazione piu' comune per il modulo MM e' l'immagazzinamento di dati veloce e temporaneo in programmi TI BASIC. Poiche' i dati si mantengono anche a console spenta, il modulo MM e' utile per conservare piccole quantita' di dati. Si puo' definire un file di lunghezza fino a 4K bytes in MM. Se l'unita' Memory Expansion e' collegata alla console del computer il modulo MM permette anche di accedere ad un altro file, EXPMEM2, localizzato nell'unita' ME. Questo file puo' avere una lunghezza fino a 24K bytes.

FILES AGGIUNTIVI INTRODOTTI NEL SISTEMA

Il modulo MM introduce due nuovi files nel sistema.

1. MINIMEM-Il segmento di memoria di 4K di lettura e scrittura localizzato nel modulo stesso MM.

2. EXPMEM2-Un segmento di 24K bytes localizzato nell'unita' ME.

L'ultimo file e' reso possibile solo se l'unita' ME e' collegata al sistema e accesa.

ACCESSO AL FILE

La memoria nel modulo MM e l'espansione ME possono essere usate per immagazzinare files di dati da TI BASIC. Inoltre, se si vuol usare questi files per immagazzinare dati insieme a programmi in linguaggio assembly si deve prendere certe precauzioni per evitare di distruggere i dati o i programmi stessi. Si veda il paragrafo relativo.

ATTENZIONE

Se i dati sono immagazzinati nel modulo MM (nel file MINIMEM) le possibilita' del linguaggio assembly non possono essere usate.

SPECIFICHE FILE DATI

Le seguenti specifiche definiscono files di dati per il modulo MM

- Organizzazione del file: SEQUENTIAL e RELATIVE
- Tipi di files: DISPLAY e INTERNAL
- Lunghezza del record: VARIABLE e FIXED
- Modi operativi: INPUT, OUTPUT, UPDATE e APPEND
- Funzione BASIC: EOF

Le seguenti restrizioni devono essere applicate alle specifiche su citate:

- Il tipo record a lunghezza VARIABLE puo' essere usato solo con files SEQUENTIAL.
- Per un file con records a lunghezza VARIABLE un termine dati di lunghezza zero nel primo record sara' memorizzato in modo scorretto. Per assicurarsi sulla correttezza occorre che il primo nel file non sia una stringa nulla.

PREPARAZIONE DI UN FILE MINIMEMORY PER IMMAGAZZINAMENTO DATI

Si puo' pensare ai files introdotti nel sistema dal modulo MM come files ad alta velocita' come files su cassetta o su disco. Le istruzioni TI BASIC usate per inizializzare e accedere ai files nel modulo MM sono quelle descritte nel manuale utente. Per accedere a un file, lo si deve aprire con una istruzione OPEN listando le specifiche del file . Per es.

```
OPEN #3: "MINIMEM"RELATIVE, FIXED, UPDATE, DISPLAY
```

I dati possono essere scritti nel file con una PRINT e letti dal file con una INPUT. L'istruzione RESTORE riposiziona il file al suo record iniziale.

```
PRINT #3: A, B, C, D
RESTORE #3
INPUT #3: A, B, C, D
```

Si deve chiudere il file quando non e' piu' necessario accedervi o se si vuole re-OPEN per stabilire specifiche differenti(come cambiarlo da file di output a file di input)con l'istruzione:

```
CLOSE #3
```

INIZIALIZZAZIONE DI UN FILE PER L'ESPANSIONE DI MEMORIA PER L'IMMAGAZZINAMENTO DATI

N. T.

LEGGERE E SCRIVERE UN FILE DATI

I programmi seguenti illustrano come scrivere dati in MINIMEM e EXPMEM e poi rileggerli.

Esempio per MINIMEM

```
100 OPEN #5: "MINIMEM", SEQUENTIAL, FIXED, OUTPUT, INTERNAL
110 INPUT X
120 INPUT Y
130 INPUT Z
140 PRINT #5: X, Y, Z
150 CLOSE #5
```

Questo segmento apre il file MINIMEM come un file di output nell'istruzione al numero 100. Le linee da 110 a 130 accettano valori introdotti da tastiera. La linea 140 scrive questi valori in MINIMEM e la linea 150 chiude il file MINIMEM.
A questo punto la console puo' essere spenta e il modulo rimosso dalla console. I dati sono conservati come se fossero su cassetta o su dischetto.
Il seguente segmento legge i dati su MINIMEM e li visualizza sullo schermo.

```
200 OPEN #5: "MINIMEM", SEQUENTIAL, FIXED, INPUT, INTERNAL
210 INPUT #5: P, Q, R
220 PRINT P, Q, R
230 CLOSE #5
```

ESEMPIO EXPMEM2

N. T.

CARICARE E SALVARE PROGRAMMI TI BASIC

In piu' all'immagazzinamento di dati il modulo MM e' anche utile per salvare corti programmi TI BASIC e programmi in linguaggio assembly. I programmi in linguaggio assembly su dischetto sono caricati con l'opzione LOAD AND RUN della lista di selezione del modulo MM mentre i programmi TI BASIC possono essere salvati e caricati usando i comandi SAVE e LOAD rispettivamente.
Il modulo MM puo' salvare fino a 4K bytes di programma(esattamente 4088 bytes)nella sua RAM. Per usare file di programmi in linguaggio assembly e files TI BASIC insieme si deve prendere qualche precauzione come descritto nel paragrafo successivo.

CARICARE E SALVARE UN PROGRAMMA

La seguente procedura mostra come creare un programma test di una sola istruzione, salvarlo nel modulo MM e poi ricaricarlo nella memoria della console.
Dopo aver selezionato TI BASIC si introduca il programma:

```
100 PRINT "THIS IS A TEST"
```

Si salvi introducendo il comando

SAVE MINIMEM

Dopo che il programma e' stato memorizzato la console puo' essere spenta e il modulo rimosso dalla console. Come test se il computer non e' stato spento introdurre il comando

NEW

per cancellare il programma in memoria. Per caricare il programma dal modulo MM introdurre il comando

OLD MINIMEM

Ora, introdurre il comando LIST per vedere che il programma venga ricaricato in memoria correttamente.

MESCOLARE PROGRAMMI IN LINGUAGGIO ASSEMBLY E FILES TI BASIC

I programmi in linguaggio assembly e i files TI BASIC non possono essere immagazzinati simultaneamente nel modulo MM. Se il modulo MM e l'unita' ME sono entrambe utilizzabili, allora si puo' mescolare programmi in linguaggio assembly e files TI BASIC con le seguenti restrizioni:

- Il modulo MM deve essere usato solo per il salvataggio dei programmi in linguaggio assembly.

- Si puo' anche salvare i programmi in linguaggio assembly in un segmento di 8K bytes nell'unita' ME.

- Il segmento di 24K bytes dell'unita' ME deve essere usato per i files TI BASIC.

ATTENZIONE

Se i dati sono salvati sul modulo MM(nel file MINIMEM) l'unita' ME non puo' essere usata per salvare programmi in linguaggio assembly.

Quando si hanno assieme il modulo MM e l'unita' ME e si vogliono mescolare programmi in linguaggio assembly e files TI BASIC si usino i seguenti passi per evitare di distruggere dati e/o programmi.

1. Inizializzare il modulo MM seguendo una delle due procedure. Una procedura e di selezionare l'opzione RE-INITIALIZE della MM. La seconda e' di selezionare il TI BASIC e usare il comando CALL INIT.

2. Dal TI BASIC si usi l'istruzione OPEN per riservare il file EXPMEM2 per il salvataggio dei dati.

3. CARICARE (LOAD) i programmi in linguaggio assembly che si vogliono usare. (v. paragrafo particolare).

SOTTOPROGRAMMI AGGIUNTIVI TI BASIC

Diversi sottoprogrammi inclusi nel modulo forniscono un'interfaccia tra programmi in linguaggio assembly e TI BASIC. Questi sottoprogrammi sono INIT, LOAD, LINK, PEEK, PEEKV, POKEV e CHARPAT. Ogni sottoprogramam e' discusso in questo capitolo. In queste discussioni il termine "CPU memory"si riferisce a tutta la memoria direttamente accessibile dall'UNITA' CENTRALE DI PROCESSO (Central Processin Unit - CPU). Cio' comprende la memoria nel modulo stesso, l'Estensione di memoria, se collegata, e la memoria in console (detta scratchpad).

Sottoprogramma INIT

Format: CALL INIT

L'istruzione di chiamata del sottoprogramma INIT non ha argomenti. Si raccomanda che venga generalmente utilizzata la CALL INIT in modo comando o in quello normale per evitare inavvertitamente di perdere programmi o dati posti in memoria. Percio' se l'istruzione CALL INIT viene usata in un programma, deve apparire primadei sottoprogrammi LOAD e LINK.

Il sottoprogramma INIT inizializza la memoria CPU per sottoprogrammi in linguaggio assembly e per reinizializzare le tabelle interne nel modulo MiniMemory. Quando questo sottoprogramma viene eseguito, controlla anche se la Espansione di Memoria e' collegata. Se si, posiziona i corrispondenti valori della tabella nel modulo in modo da permettere l'accesso sia al modulo che all'Espansione di Memoria.

ATTENZIONE

CALL INIT cancella tutti i programmi e dati nel modulo MM. Si usi quindi solo per cancellare la memoria per il caricamento di nuovi programmi o sottoprogrammi. Inoltre se l'Espansione di Memoria non e' collegata bene o se non e' stata accesa quando si esegue CALL INIT, il sottoprogramma INIT non la riconosce. Se cio' capita l'Espansione non puo' essere usata per caricare programmi.

Poiche' la MM contiene al proprio interno una batteria, il modulo non deve essere inizializzato ogni volta che la console viene accesa. Solo se si vuol reinizializzare la memoria del modulo occorre usare il sottoprogramma INIT.

PRECAUZIONE

La MM mantiene solo i dati contenuti nel modulo stesso. Tutti i dati nell'Espansione di Memoria vengono persi se il sistema viene spento.

Sottoprogramma LOAD

Il sottoprogramma LOAD e' utilizzabile in due modi:

- a caricare files oggetto in linguaggio assembly

- a caricare dati nella memoria CPU

La sintassi per l'istruzione CALL LOAD ha due forme, dipendenti dal modo d'uso dell'istruzione.

Caricamento di files oggetto

Format CALL LOAD(obj-nomefile[,obj-nomefile,...])

Questo formato dell'istruzione CALL LOAD carica un file oggetto di linguaggio assembly o introduce direttamente dati in memoria per essere eseguiti piu' avanti con l'istruzione CALL LINK. Il nome del file oggetto(obj-nomefile) puo' essere qualsiasi stringa di caratteri valida e specifica il file che deve essere aperto e letto dal sottoprogramma LOAD. Il codice oggetto rilocabile e' caricato al primo indirizzo consentito che dipende dalla configurazione del sistema; spazio e' poi riservato per i programmi in linguaggio assembly in accordo con la lunghezza specificata nel campo "0-tag" nel file oggetto. (Per una descrizione dei campi tag del programma oggetto si veda il manuale EDITOR/ASSEMBLER). Il codice oggetto assoluto e' caricato all'indirizzo assoluto specificato nel codice oggetto.
Per esempio l'istruzione

CALL LOAD("DSK1.DEMO")

carica il file DEMO dal dischetto nel Disk Drive 1.

AVVERTENZA

Il codice assoluto e' caricato all'indirizzo specificato nel codice oggetto. Nessun spazio e' riservato se la lunghezza non e' specificata nel campo "0-tag". Caricando dati in memoria usata dall'interprete TI BASIC puo' causare il "crash" del sistema. Se si sta usando solo il modulo MM senza l'Espansione di Memoria collegata alla console e accesa, il primo programma in linguaggio assembly e' caricato partendo da >7118, il piu' basso indirizzo consentito nella RAM del modulo. Se l'Espansione di memoria e' collegata e accesa, il primo programma in linguaggio assembly e' caricato partendo da >A000, l'indirizzo di partenza del segmento della memoria piu' alta nell'Espansione di memoria. I programmi seguenti sono caricati in sequenza cominciando dall'indirizzo piu' basso della memoria alta.
(si veda "Loading Assembly Language Programs" per ulteriori informazioni)

Caricamento di Files dati

Format: CALL LOAD(indirizzo,valore[,...,"",indirizzo,valore,...])

Quando si usa il sottoprogramma LOAD per caricare dati nella memoria CPU, occorre specificare una lista di interi(chiamati poke list). La poke list dovrebbe partire tra l'indirizzo 0 (>0000) e 32767 (>7FFF) o tra l'indirizzo -1(>FFFF) e -32768(>8000), seguito da una lista di interi utilizzati come valore dati di un solo byte(one-byte). Questi valori interi sono caricati in locazioni consecutive partendo ad un dato indirizzo. Una stringa vuota ("")separa l'ultimo byte di una poke list e l'indirizzo di partenza della prossima. L'indirizzo di una poke list e' assoluto e

il dato non e' rilocabile.
Per esempio l'istruzione

CALL LOAD(-32000,255,21,"",8197,85)

carica il valore >FF15 all'indirizzo di memoria >8300(indirizzi del byte >8300 e >8301)e il valore >55 all'indirizzo di memoria del byte >2005.
Se un programma in codice oggetto e' caricato direttamente con una poke list deve essere caricato anche il nome del punto d'ingresso in modo che il programma possa essere raggiunto da una istruzione di CALL LINK(descritta qui sotto).
Il nome del programma e l'indirizzo sono aggiunti alla tabella REF/DEF nella memoria del modulo nel seguente modo.
Dapprima il First Free Address (primo indirizzo libero)del modulo(FFAM)e il Last Free Address(ultimo indirizzo libero)del modulo(LFAM) devono essere letti dalla memoria per mezzo del comando PEEK(descritto qui sotto). Gli indirizzi di queste due variabili sono >701C e 701E rispettivamente. Dopo aver controllato che c'e' ancora abbastanza spazio (8 bytes)per aggiungere un'altra etichetta alla tabella REF/DEF si sottragga 8 dal vecchio LFAM e si introduca il nuovo valore LFAM in >701E usando l'istruzione CALL LOAD. Si carichi il nome del programma(2 bytes)nello spazio di 8 bytes aggiunto nella tabella REF/DEF.
Per esempio se LFAM e' >8000, lo si cambi in >7FF8 e si carichi il nome e poi l'indirizzo del programma.

Sottoprogramma LINK

Format: CALL LINK(nomeprogramma[,listaparametri,"",...])

Il sottoprogramma LINK passa il controllo e, opzionalmente, una lista di parametri da un programma TI BASIC a un programma in linguaggio assembly.
Il nome del programma e' una stringa che consta da uno a sei caratteri e deve essere un punto d'ingresso nella tabella REF/DEF. Questo nome deve essere definito in un programma che sia stato caricato precedentemente. Oppure, se il programma e' stato caricato byte dopo byte con una poke list in un'istruzione CALL LOAD, il nome del programma deve essere stato introdotto esplicitamente nella tabella REF/DEF. Vedi la sezione "Sottoprogramma LOAD" per ulteriori informazioni.
La lista dei parametri e' opzionale. Questa lista e' usata quando occorre passare dei parametri tra un programma in linguaggio assembly e uno TI BASIC. Si possono passare stringhe o variabili numeriche o espressioni.

Come si passano i parametri

In dipendenza che un parametro sia una variabile o un'espressione il parametro e' passato col nome o col suo valore. Le variabili sono passate col nome e le espressioni col valore.
Se una variabile e' passata ad un programma in linguaggio assembly, il suo valore puo' cambiare anche nel programma in assembly, oltre che cambiando il valore della variabile nel programma principale. Se una variabile in una lista di parametri non e' apparsa in precedenti istruzioni TI BASIC, l'interprete crea un punto d'ingresso nella Symbol Table per la variabile.
Le espressioni sono passate con il loro valore, a meno che' non siano direttamente associate con una variabile. Il valore di

un'espressione non puo' essere passato all'indietro al programma chiamante. Quando un elemento di una tabella, come per es. A(9), e' presente in una lista di parametri, e' passata come una variabile. Una tabella intera puo' essere passata facendo seguire il nome del parametro con le parentesi. Se la tabella ha piu' di una dimensione, una virgola deve essere posta all'interno delle parentesi per ogni dimensione aggiuntiva. Per esempio A() indica una tabella monodimensionale chiamata A. EXT$(,,) rappresenta una tabella di stringhe tridimensionale chiamata EXT$. Per specificare che certe variabili sono state usate solo per passare un valore, ma non per avere come ritorno un risultato, la variabile puo' essere racchiusa fra parentesi. Per esempio (SUMI)si riferisce al valore corrente della variabile numerica SUMI. (A$(5)) si riferisce al valore dell'elemento della tabella stringa A$(5). Si noti che una tabella completa non puo' essere passata tramite il valore ma deve essere passata tramite il nome; percio' (A()) potrebbe essere illegale.
Al massimo si possono listare 15 argomenti nella lista dei parametri.

Operazioni

Il sottoprogramma LINK permette le seguenti azioni.

- Valutare il nome del programma in linguaggio assembly e la sua lunghezza(da 1 a 6 caratteri) e porre questa informazione come valore nello stack.

- costruire la lista degli argomenti consistente di identificatori per ogni argomento nella lista dei parametri e costruire un punto d'ingresso nello stack per ogni argomento.

- muovere il nome del programma verso l'area dove il sottoprogramma d'utilita' puo' accedervi e trasferire il controllo al programma d'utilita'.

- dopo il ritorno, saltare al sottoprogramma d'errore se un errore e' stato rilevato. Altrimenti cancellere il punto d'ingresso usato durante l'esecuzione di un LINK e ritornare al programma BASIC chimante.

Il sottoprogramma LINK passa l'informazione sugli argomenti tramite la lista degli identificatori degli argomenti in CPU RAM e il valore nello stack in VDP RAM.
Gli identificatori degli argomenti, vecchio valore del puntatore dello stack, e il numero di argomenti nella lista sono localizzati nelle seguenti locazioni CPU RAM:

Indirizzo	Contenuto
>7002->7011	Ident. argom., un byte per ogni argomento.
>8310	Vecchio valore del puntatore dello stack dell'interprete BASIC.
>8312	Numero di argomenti nella lista dei parametri.

Gli identificatori degli argomenti sono i seguenti:

```
 0    Espressione numerica
 1        "       stringa
 2    Variabile numerica
 3        "     stringa
 4    Tabella numerica
 5       "     stringa
```

Ulteriori informazioni su ogni argomento somo memorizzate nel valore dello stack a otto bytes nella memoria VDP. La struttura di un valore individuale dello stack dipende dal tipo di argomento come descritto qui sotto.

Espressione numerica

Lo stack contiene il valore dell'espressione numerica. Il valore e' espresso in notazione radice 100. Il primo byte e l'esponente di 100. Se l'eponente e' positivo, e' maggiore di 64. Un esponente negativo e' espresso come un valore minore di 64 nel primo byte. Il valore assoluto dell'esponente e' la differenza tra questo valore e 64. Gli altri sette bytes contengono un numero da 0 a 99 con digit a radice 100. Se il numero e' negativo, la prima parola (di due bytes) e' il complemento a 2 del numero. Per esempio:

```
>3F, >22, >00, >00, >00, >00, >00, >00      e' uguale a 0.34
>BE, >FB, >00, >00, >00, >00, >00, >00      e' uguale a -500
```

Espressione stringa

Un punto d'ingresso alla stringa consiste della seguente informazione.

Bytes 0-1	>001C
Byte 2	>65(Il tag stringa usato dall'interprete BASIC).
Bytes 4-5	Il puntatore al valore della stringa nella memoria VDP.
Bytes 6-7	La lunghezza della stringa. Il byte 6 dovrebbe essere sempre 0 poiche' la massima lunghezza della stringa e' di 255 caratteri.

Variabile numerica

Questo termine e' sia una variabile numerica che un elemento di una tabella(array). Lo stack contiene la seguente informazione:

Bytes 0-1	Il puntatore al punto d'ingresso della Symbol Table nella memoria VDP.
Byte 2	Zero
Bytes 4-5	Il puntatore al valore dell'ottavo byte della variabile nella memoria VDP.

Variabile stringa

Questo termine e' sia una variabile stringa che un elemento di una tabella stringa. Il punto d'ingresso allo stack contiene la seguente informazione:

Bytes 0-1	Il puntatore al punto d'ingresso della variabile nella Symbol Table nella VDP RAM.
Byte 2	>65(Il tag di stringa usato dall'interprete TI BASIC).
Bytes 4-5	Il puntatore al valore di stringa nella memoria VDP
Bytes 6-7	La lunghezza della stringa

Tabella numerica

Questo punto d'ingresso risulta dall'argomento del vettore A(), A(,),ecc. E' usato per permettere ad un sottoprogramma di manipolare un'intera tabella. Il punto d'ingresso di stringa contiene la seguente informazione:

Bytes 0-1	Il puntatore al punto d'ingresso della tabella nella Symbol Table
Byte 2	Zero
Bytes 4-5	Il puntatore al valore dello spazio della tabella in VDP RAM. Il valore dello spazio per una tabella numerica ha due bytes per ogni dimensione che indica l'indice massimo per quella dimensione. I valori del resto degli elementi sono memorizzati in ordine sequenziale.

Tabella stringa

Questo punto d'ingresso e' simile a quello per atbella numerica, eccetto che per il byte 2 che contiene >65. Il valore di spazio per una tabella stringa contiene due bytes per ogni dimensione che indicano l'indice massimo seguito da un puntatore ad ogni valore dell'elemento in tabella(valore stringa) nella VDP RAM. Si noti che con una tabella numerica ogni elemento della tabella e' memorizzato consecutivamente nella stessa area di memoria mentre gli elementi di una tabella stringa sono posti in aree di memoria non contigue.

Il sottoprogramma NAME LINK ROUTINE

Quando un sottoprogramma in linguaggio assembly e' chiamato dal TI BASIC da un'istruzione LINK il controllo passa al sottoprogramma tramite il programma NAME LINK che risiede fra i programmi di utilita'. Il programma NAME LINK trova il nome del programma nella tabella REF/DEF posta all'estremita' alta della memoria del modulo MM. Quando un programma in linguaggio assembly e' caricato ,il Loader(caricatore) aggiunge un punto d'ingresso a 8 byte alla tabella REF/DEF quando vede un'etichetta REFerenziata o DEFinita. Questa tabella REF/DEF parte da >7FFF e va verso >7118 che e' il primo indirizzo libero(FFAM) nel modulo MM.
La tabella REF/DEF viene esplorata dall'indirizzo piu' basso in su. Percio' se due routines sono caricate con lo stesso nome, viene usata quella che e' stata caricata per seconda. Se il nome che le e' stata data e' piu' grande di sei caratteri o se la routine NAME LINK non riesce a trovare il nome nella tabella, sara' segnalato un errore. La routine NAME LINK trasferisce il controllo al programma in linguaggio assembly con un'istruzione del tipo (BL) 9900 branch-and-link. Quando viene chiamato il programma in linguaggio assembler dalla routine di link (collegamento), l'area

di lavoro (workspace) e' postaa >70B8 e l'indirizzo di ritorno e' in R11di quel workspace.Prima di ritornare il programma dovrebbe cancellare il byte a >837C;altrimenti viene visualizzato un messaggio di errore anche se il programma non ha generato nessun errore.
Il programma in linguaggio assembly puo' assegnare nuovi valori a variabili numeriche o a stringhe o a elementi di tabelle numeriche o di stringa con funzioni d'utilita' fornite dal sistema. Queste funzioni d'utilita' sono descritte nella sezione "System Utility Routines".
I punti d'ingresso sul valore dello stack che risultano dai parametri passati dall'istruzione CALL LINK sono automaticamente cancellati dal sottoprogramma LINK.Se si manipola direttamente il valore dello stack,comunque,si deve riportare lo stack al suo stato originale prima di ritornare il controllo al sottoprogramma LINK.

SOTTOPROGRAMMA PEEK

Format: CALL PEEK(indirizzo,var[,var,...,"",indirizzo,var,...])

Il sottoprogramma PEEK e' usato per leggere bytes dalla CPU RAM direttamente nelle variabili TI BASIC.
Il parametro 'indirizzo' deve essere o un'espressione numerica o una variabile numerica.L'indirizzo e' un valore decimale da -32768 a 32767 e rappresenta un valore intero a due bytes.Gli indirizzi maggiori di >7FFF sono scritti come numeri negativi trattando il valore come un intero a complemento due.(Per esempio per accedere all'indirizzo maggiore di 32767 occorre sottrarre 65536 da lui).
La lista variabili(parametri var)deve constare di variabili numeriche.Ogni byte consecutivo letto dalla memoria e' assegnato ad ogni variabile nell'ordine indicato nella lista delle variabili.Una stringa nulla("")separa una sequenza PEEK dalla seguente in modo da poter ripetutamente usare PEEK per diverse locazioni di memoria in una singola istruzione.
Per esempio l'istruzione

```
CALL PEEK(8192,A,B,C(8),"".24576,X)
```

legge tre bytes dall'indirizzo >2000 in avanti;assegna i valori alle variabili A,B,C(8) consecutivamente;legge un byte dalla locazione >A000 e memorizza il valore nella variabile X.
Il valore di ritorno e' un valore di un byte ed e' sempre nel campo da 0 a 255.

SOTTOPROGRAMMA PEEKV

Format: CALL PEEKV(indirizzo,var[,var,...,"",indirizzo,var,...])

Il sottoprogramma PEEKV e' usato per leggere bytes dalla VDP RAM. Lavora esattamente come PEEK eccetto che PEEKV accede alla VDP RAM anziche' alla CPU RAM.
L'indirizzo e' un valore decimale da 0 a 16383 e la lista delle variabili(parametri var) e' una lista di variabili numeriche che

devono contenere i valori letti. Si noti che la VDP ha 16K di RAM e il tentativo di accedere ad un indirizzo di memoria piu' alto di 16383 puo' mandare in crash il sistema. Inoltre si veda "Sotto-programma PEEK" per ulteriori informazioni.

SOTTOPROGRAMMA POKEV

Format: CALL POKEV(indirizzo,var[,var,...,"",indirizzo,var,...])

Il sottoprogramma POKEV permette di modificare il valore nella VDP RAM. Lavora nello stesso modo di come lavora LOAD quando LOAD e' usato per modificare la memoria CPU.
L'indirizzo e' un valore decimale da 0 a 16383 e var e' un'espressione numerica o una variabile numerica che contiene un valore da porre nella memoria VDP all'indirizzo specificato. Ogni valore specificato e' memorizzato consecutivamente iniziando dall'indirizzo dato. Per esempio l'istruzione

```
CALL POKEV(784,30,30,30,"",2,V)
```

cambia il colore 16 con il colore 18(da >310 a >312 nella VDP RAM)col risultato di un foreground nero e un background grigio. Se il valore di V e' 161 il carattere A apparira' nell'angolo alto a sinistra dello schermo.

SOTTOPROGRAMMA CHARPAT

Format: CALL CHARPAT(car-cod,str-var[,car-cod,str-var,...])

Il sottoprogramma CHARPAT ritorna un insieme di 16 caratteri che specifica la configurazione del codice carattere.
Il codice carattere (car-cod) e' un qualsiasi numero compreso tra 32 e 159. I codici carattere da 32 a 95(fino a 127 sul TI-99/4A) sono normalmente riservati per i caratteri ASCII e sono inizialmente definiti dall'interprete TI BASIC. L'espressione stringa (definizione)del codice carattere e' letto nella str-var(variabile stringa). Questa espressione consta di 16 caratteri di cifre esadecimali che rappresentano il carattere. Ci si riferisca al sottoprogramma CHAR nel manuale TI BASIC per ulteriori dettagli sulla definizione del carattere.

COME CARICARE PROGRAMMI IN LINGUAGGIO ASSEMBLY

Il modulo MM e l'Espansione di Memoria sono un insieme potente. Pero' quando vengono usati insieme occorre esercitare qualche avvertenza per assicurarsi che i files oggetto rilocabili siano caricati nella propria area di memoria. Se vengono usati insieme i programmi rilocabili sono caricati nello spazio di memoria nella seguente sequenza:

1. Il segmento di memoria piu' alto dell'Espansione di memoria (l'area di 24 Kbytes che parte da >A000)

2. Il segmento di memoria piu' basso dell'Espansione di memoria (L'area di memoria di 8Kbytes che parte da >2000)
3. La memoria del modulo MM(l'area di 8Kbytes che parte da >7118)

Il primo indirizzo libero nella memoria alta e' inizializzato a >A000 dal sottoprogramma INIT e il codice rilocabile e' riallocato all'indirizzo di partenza di caricamento. Se un "0-tag" viene incontrato l'indirizzo di partenza di caricamento viene aggiornato dal primo indirizzo libero nella memoria alta e la lunghezza del programma e' aggiunta a questo indizzo. I programmi seguenti sono caricati sequenzialmente iniziando dall'indirizzo piu' basso nell'area di memoria alta. (Si veda l'Appendice A per una mappa di memoria CPU quando entrambi i moduli MM e EM sono utilizzati).
Se si sta usando solo il modulo MM senza L'EA collegata alla console e accesa, il programma e' caricato nella RAM del modulo. Il primo programma in linguaggio assembly e'caricato a >7118 , indirizzo piu' basso utilizzabile nella RAM del modulo MM. A volte si vorrebbe caricare un programma direttamente nel modulo MM quando l'EA e' collegata, oltrepassando la sequenza normale di caricamento. Per far cio', e' necessario rendere l'EA temporaneamente 'invisibile' al sistema cancellando i valori nelle locazioni di memoria da >7022 a >7029 (v. Tabella 2 qui sotto). Questi sono i valori dei puntatori che indicano la presenza dell'EA(v. Tabella 1 per questi valori).
La via piu' facile di completare questa attivita' e' di usare un piccolo programma TI BASIC che include le due versioni del sottoprogramma LOAD una con una poke list e una che carica il programma in linguaggio assembly nel modulo MM.

```
CALL INIT
100 CALL LOAD(28706,0,0,0,0
0,0,0,0)
110 CALL LOAD("DSK1.DEMO")
120 CALL LINK("LINES")
```

Il comando CALL INIT inizializza il sistema, cancellando ogni dato o puntatore di programma precedentemente caricato. La prima linea del programma azzera i riferimanti all'EA partendo dalla locazione di memoria 28706(>7022). La linea 110 carica un programma denominato DEMO dal dischetto nel Disk Drive 1 e la linea 120 esegue DEMO partendo dal punto etichettato con LINES.
Se si vuol informare di nuovo il sistema dell'esistenza della EA si puo' di nuovo usare una CALL LOAD con una poke list di appropriati valori decimali(v. Tabella 1).

Tabella 1. Variabili MM con EA collegata e accesa

Locazione	Val. Esadecimale	Val. decimale
>7022	>A0	160
>7023	>00	0
>7024	>FF	255
>7025	>E0	224
>7026	>20	32
>7027	>00	0
>7028	>3F	63
>7029	>FF	255

Tabella 2. Variabili MM con EA scollegata, spenta o 'invisibile'.
Tutti i valori precedenti sono azzerati a >00 e 0

Si puo' anche usare il comando M(Modify)dell'EASY BUG per restaurare i valori di tabella in modo che il sistema riconosca di nuovo la presenza dell'EA.

Nota: Quando si crea un programma in linguaggio assembly e' inportante conoscere come usare le direttive giuste del linguaggio assembly per essere sicuri sui progarmmi e che i dati associati siano caricati correttamente. Occorre riferirsi al manuale E/A come guida.

SOTTOPROGRAMMI DI SISTEMA DI UTILITA'

(System utility routines)

Le utility routines(UR)residenti nel modulo MM possono essere chiamate da un programma in linguaggio assembly per accedere alle risorse della macchina e come interfaccia all'interprete TI BASIC. L'uso di queste routines richiede una conoscenza delle routines stesse e l'organizzazione dei dati usati dalle routines. Ulteriori informazioni su questi argomenti sono incluse nel manuale E/A.
Due tipi di UR sono fornite nel modulo MM:

- Il primo programma contiene una collezione di utilita' per il sistema standard con cui collegarsi alle routines ROM/GROM realizzare una scansione della tastiera, accedere alla VDP ecc.

- Il secondo programma contiene utilita' d'interfaccia con cui un programma in linguaggio assembly puo' accedere alle variabili passate tramite un'istruzione CALL LINK in un programma TI BASIC. Questo programma contiene anche un'utilita' di manipolazione degli errori per ritornare eccezioni ad un programma TI BASIC.

UTILITA' STANDARD

Tutte le routines di utilita' usano UTILWS(indirizzo >7092)per l'area di lavoro d'utilita' per i registri(utility workspace registers) e tutti i parametri sono passati tramite i registri di chiamata del programma. Per convenienza USRWSP(indirizzo >70B8) e' riservato per il posizionamento dei registri del programma. Comunque ogni area di registro che si fornisce puo' essere usata per passare i parametri.
Le seguenti sezioni descrivono le convenzioni per passare i dati e la sintassi di chiamata per ciascuna routine.

VDP Single Byte Write - VSBW

Format: BLWP @VSBW Fa corrispondere VSBW a >6024

Questa routine scrive un valore di un singolo byte ad uno specificato indirizzo in VDP RAM.

```
RO    L'indirizzo VDP RAM
R1    Valore di un byte singolo nel byte piu' significativo del Registro 1
```

```
Esempio     LI    RO,>0200    Indirizzo >0200 in VDP RAM
            LI    R1,>4100    Codice carattere di A
            BLWP  @VSBW       Visualizza il carattere A
```

Questo programma visualizza il carattere A sullo schermo alla locazione >0200

VDP Multiple Byte Write - VMBW

Format: BLWP @VMBW Fa corrispondere VMBW a >6028

Questa routine scrive bytes multipli da CPU RAM A VDP RAM

```
RO    Indirizzo VDP RAM
R1    Indirizzo di partenza del buffer CPU RAM
R2    Numero di bytes da scrivere
```

Esempio

```
            LI    RO,>018E    Indirizzo VDP RAM >018E
            LI    R1,HI       Indirizzo del testo
            LI    R2,5        Numero di bytes da scrivere
            BLWP  @VMBW       Visualizza i caratteri

            ...
     HI     TEXT  'HELLO'     Testo da visualizzare
```

Questo programma visualizza la parola HELLO nel mezzo dello schermo(indirizzi VDP RAM >018E).

VDP Single Byte Read - VSBR

Format: BLWP @VSBR Fa corrispondere VSBR a >602C

Questa routine legge un singolo byte da uno specificato indirizzo in VDP RAM.

```
RO    Indirizzo VDP RAM
R1    Valore letto da VDP RAM nel byte piu' significativo
```

Esempio

```
        LI    R0,>0380   Indirizzo VDP RAM >0380
        BLWP  @VSBR      Legge un byte
```

Questo programma legge un byte dalla tabella dei colori(>0380)nel byte piu' significativo del Registro 1.

VDP Multiple Byte Read - VMBR

Format: BLWP @VMBR Fa corrispondere VMBR a >6030

Questa routine legge bytes multipli da VDP RAM in CPU RAM.

R0	Indirizzo VDP RAM da cui leggere
R1	Indirizzo di partenza del buffer CPU RAM
R2	Numero di bytes da leggere

Esempio

```
        LI    R0,>0300   Indirizzo VDP RAM >0300
        LI    R1,BUFF    Area del buffer
        LI    R2,>0080   Carica il numero di bytes da leggere
        BLWP  @VMBR      Legge i bytes

        ...
BUFF    BSS   >0080      Inizializzazione del buffer con i
                         bytes letti
```

Questo programma copia i >0080 bytes dei dati della lista degli attributi degli sprite da VDP RAM >0300 nell'area buffer chiamata BUFF.

VDP Write to Register - VWTR

Format: BLWP @VWTR Fa corrispondere VWTR a >6034

Questa routine scrive un valore di un singolo byte ad un qualsiasi registro VDP RAM.

R0	Il byte meno significativo contiene il valore che deve essere scritto; il byte piu' significativo contiene il numero del registro VDP (da 0 a 7)

Keyboard Scan - KSCAN

Format: BLWP @KSCAN Fa corrispondere KSCAN a >6020

Questa routine controlla la tastiera e ritorna un codice di tasto e stato. Le seguenti locazioni di memoria sono usate per comunicazioni tra il programma utente e la routine.

>8374	Numero del supporto tastiera. Questo numero di un byte deve essere specificato dal programma. Il

significato di questo byte e' lo stesso di quello del key-unit nel sottoprogramma TI BASIC KEY.

>8375	Valore ASCII del tasto premuto(un byte).
>8376	Posizione Y del joystick(un byte)
>8377	Posizione X del joystick(un byte)
>837C	Registro di stato GPL(un byte)

Il byte di stato GPL puo' essere controllato in ritorno prima che il codice del tasto sia letto. Si puo' fare cio' con un'istruzione Compare Ones Corresponding(COC). (V. manuale E/A). Il bit 5 del byte di stato GPL e' posto a 1 se un tasto e' stato premuto dopo l'ultima chiamata a KSCAN. I bits di stato GPL sono assegnati come segue:

H	GT	COND	CARRY	OVF	0	0	0
7	6	5	4	3	2	1	0

Si veda la sezione "Extended Utilities"per una piu' completa descrizione dei bits di altro stato.

EXTENDED UTILITIES

Le Extended Utilities (EU) sono fornite per accedere alle routines GROM E ROM in console. Queste EU sono :
- GPLLNK (collegamento alle routines GPL in GROM)
- XMLLNK (collegamento alle routines in ROM)
- DSRLNK (collegamento alle Device Service Routines).

Poiche' le EU accedono alle routine in console estrema precauzione deve essere posta quando vengono usate. Occorre essere sicuri che i registri di lavoro GPL non siano cambiati, lo spazio di memoria usato dalla routine di console sia predisposta in modo opportuno e la routine ritorni correttamente al programma utente.

Collegamento tra GROM e Routines Residenti -GPLLNK

Format:	BLWP @GPLLNK	Fa corrispondere GPLLNK a >6018
	DATA c-r-i	Fornisce l'indirizzo della routine GPL che deve essere eseguita. (c=console, r=routine, i=indirizzo)

La routine GPLLNK attiva un riferimento interno per indicare che un programma GPL e' stato chiamato da un programma in linguaggio assembly, carica lo spazio di lavoro GPL(indirizzo >83E0), salta al codice GROM ed esegue la routine GPL specificata dalla direttiva DATA.
La routine GPL deve ritornare con un'istruzione RTN allo scopo di ritrasferirsi a ritroso alla chiamata iniziale. Quando l'istruzione RTN viene incontrata nella routine GPL l'istruzione ritorna alla routine di sistema. La routine di sistema controlla il riferimento interno e, trovandolo attivato, ritorna indietro alla routine in linguaggio assembly.
Alcuni degli indirizzi delle routines GPL e le loro chiuamate e convenzioni di ritorno sono date qui sotto. I nomi FAC, STACK, e STATUS sono usate nelle descrizioni seguenti. FAC e' fatto

corrispondere a >834A, STACK a >836E e STATUS a >837C.
STATUS e' il byte di stato GPL ed e' organizzato come segue:

	High	Greater	Condition	Carry	Overflow	Unused
Bit	7	6	5	4	3	2,1,0

Bit 7 Bit alto. Controllato durante l'esecuzione dell'interprete GPL.
Bit 6 Bit di maggiore che. Controllato dall'interprete GPL durante l'esecuzione del programma GPL.
Bit 5 Bit di condizione. Controllato dall'interprete GPL. La routine di controllo tasto accende questo bit quando un nuovo tasto viene spinto. Inoltre la routine DSR accende questo bit per indicare che un file non esiste.
Bit 4 Carry bit. Controllato dall'interprete GPL.
Bit 3 Overflow bit. Controllato dall'interprete GPL.

La direttiva DATA specifica l'indirizzo della routine GPL che deve essere eseguita. Ogni routine e' descritta qui sotto.

DATA >0016 Load Standard Character Set-Carica l'insieme dei caratteri standard nella VDP RAM.
Input: FAC-Puntatore all'indirizzo di partenza nella VDP RAM dove i caratteri sono caricati.
Output: La VDP RAM all'indirizzo specificato in FAC contiene l'insieme dei caratteri standard.

DATA >0018 Load Small Character Set-Carica l'insieme dei caratteri minuscoli(per il modo TEXT) nella VDP RAM.
Input: V. sopra
Output: V. sopra

DATA >0020 Execute Power-Up Routine-Alimenta e inizializza il sistema.
Input: Nulla
Output: Il sistema e' alimentato e inizializzato. Il suono e i circuiti VDP sono cancellati; i valori iniziali per i registri VDP, l'insieme dei caratteri, la tabella dei colori, e lo status block sono caricati. La dimensione utile VDP RAM e' memorizzata in >8370.

DATA >0034 Accept Tone-Emette un suono di accettazione in input. Nessuna inizializzazione e' richiesta prima di chiamare questa routine.

DATA >0036 Bad Response Tone-Emette un suono di avvertimento per risposta errata. Nessuna inizializzaziuone e' richiesta prima di chiamare questa routine.

DATA >0038 Get String Space Routine-Alloca uno spazio di memoria nella VDP RAM con uno specificato numero di bytes. Questa routine non dovrebbe essere usata al di fuori dell'ambiente TI BASIC. Se non c'e' abbastanza spazio la routine fa una "garbage collection"per eliminare temporaneamente le stringhe e poi tenta di nuovo. Se

non c'e' ancora abbastanza spazio la routine emette il messaggio d'errore MEMORY FULL(memoria piena).
Input: Gli indirizzi >830C e >830D dovrebbero contenere il numero di bytes che devono essere allocati.
Output: L'indirizzo >831C punta allo spazio allocato e l'indirizzo >831A punta al primo indirizzo libero nella VDP RAM. I quattro bytes agli indirizzi da >8356 a >8359 sono usati da questa routine. L'area FAC puo' essere distrutta se viene fatta una 'garbage collection' (raccolta d'immondizia).
Nota: Sebbene questa routine sia progettata per allocare uno spazio stringa nella VDP RAM, e' anche utile per assegnare spazio per il Peripheral Access Block(PAB) e il buffer dati richiesto da un DSR. Si veda il manuale E/A per il PAB.

DATA >003B Bit Reversal Routine-Fornisce un'immagine speculare di un bytes d'informazione. E' molto comunemente usato per formare un'immagine speculare della definizione di un carattere.
Input: FAC-Indirizzo dei dati nella VDP RAM.
FAC+2 Numero di bytes da rovesciare
Output: Il numero specificato di bytes nella VDP RAM sono rovesciati bit per bit; cioe' vengono scambiati i bits 0 e 7, poi i bits 1 e 6, i bits 2 e 5 e i bits 3 e 4.
Effetto collaterale: La CPU RAM da >8300 a >8340 viene cancellata.

DATA >003D Cassette Device Service Routine-Accede alla routine DSR del registratore a cassette.
Input: Il PAB e il buffer dati devono essere inizializzati nella VDP RAM prima della chiamata. L'offset dello schermo e' >60 per l'ambiente TI BASIC e >00 fouri dall'ambiente TI BASIC. L'indirizzo di partenza dello schermo deve essere >00 per l'indicatore (prompt) proveniente dalla routine DSR del registratore a cassetta. FAC e' il nome del supporto esterno (per es. "CS1"). L'indirizzo >8356 punta al primo carattere dopo il nome in PAB. Gli indirizzi >8354 e >8355 contengono la lunghezza del nome (per es. >0003 per "CS1"). La parola all'indirizzo >83D0 dovrebbe essere posta a >0000. L'indirizzo >836D deve essere posto a >08 per indicare una chiamata DSR. Il byte STATUS deve essere >00.
Output: La routine DSR di registratore a cassetta indica le operazioni necessarie.

DATA >004A Load Lower Case Character Set-Carica l'insieme di caratteri minuscoli in VDP RAM. Input e output sono gli stessi di quelli che si hanno caricando altri insiemi di caratteri.
Nota: questa routine si applica solo alla console del TI-99/4A.

Uno degi usi per le routines di collegamento GPL e' di chiamare le routines in virgola mobile scritte in GPL da un programma in linguaggio assembly. Quando queste routines in virgola mobile sono chiamate il contenuto delle locazioni della CPU RAM da >834A a >836F possono essere da loro utilizzate e le locazioni VDP RAM da >03C0 a >03DF sono utilizzate come area buffer.
Il byte di status GPL riflette la condizione di calcolo. Tutti i valori dei dati di input e output sono in virgola mobile(floating point format).
Quando accadono errori durante l'esecuzione di routines in virgola mobile questi sono indicati nella locazione >8354 della CPU RAM. I codici di errore sono dati qui sotto.

Codice	Descrizione dell'errore
01	Errore di Overflow
02	Errore di Sintassi
03	Overflow su un numero intero in conversione
04	Radice quadrata di un numero negativo
05	Numero negativo elevato ad una potenza non intera
06	Logaritmo di un numero negativo o di zero
07	Argomento non valido in una funzione trigonometrica

Le routines in virgola mobile sono descritte qui sotto.

DATA >0014 Convert Number to String(CNS)-Converte un numero in virgola mobile in una stringa ASCII.
Input: FAC-Valore in virgola mobile a 8 bytes
FAC+11-Se posto a zero, la stringa in output e' in formato BASIC. Altimenti l'output e' in FIX mode (virgola fissa) che richiede dati in FAC+12 e FAC+13.
FAC+12-Se uno, significa overflow fuori dal campo di +/-EE...E. L'underflow e' indicato come zero.
FAC+13-Il numero di digit da porre a destra del punto decimale. Un valore negativo disabilita il FIX mode.
Output: FAC-Modificato
FAC+11-Il byte meno significativo dell'indirizzo dove la sringa risultante e' localizzata. Si deve aggiungere il valore >8300 per ottenere l'indirizzo reale.
FAC+12-La lunghezza della stringa in bytes.

DATA >0022 Greatest Integer Function(INT)-Calcola l'intero piu' grande contenuto nel valore.
Input: FAC-Il valore in virgola mobile
Output: FAC-Il risultato. Per numeri positivi, l'intero e' il valore troncato. Per numeri negativi l'intero e' il valore troncato piu' uno.
Status-E' posto in funzione del risultato.

DATA >0024 Involution Routine(PWR)-Eleva un numero ad una potenza data.
Input: FAC-Il valore dell'esponente
STACK-Il puntatore allo stack in VDP RAM che contiene il valore a 8 bytes.
Output: FAC-Il risultato in formato a virgola mobile. E' calcolato come EXP(valore dell'esponente)*

LOG(ABS(base)).
STATUS-E' posto in funzione del risultato.
Condizioni d'errore: Numero negativo elevato a un numero non intero e zero elevato ad una potenza negativa.
Effetto collaterale: Le locazioni >8375 e >8376 sono distrutte e il contenuto della parola (one-word) a >836E e' decrementato di 8. Inoltre gli indirizzi da FAC+12 a FAC+19 sono distrutti.

DATA >0026 Square Root Routine(SQR)-Calcola la radice quadrata di un numero.
Input: FAC-Il valore di input
Output: FAC-la radice quadrata del valore di input.
STATUS-E' posto in funzione del risultato.
Effetti collaterali: Gli indirizzi >8375 e >8376 sono distrutti.

DATA >0028 Exponent Routine(EXP)-Calcola l'inverso del logaritmo naturale del valore di input.
Input: v. s.
Output: v. s.
Effetti collaterali: v. s.

DATA >002A Natural Logarithm Routine(LOG)-Calcola il logaritmo naturale di un numero.
Input: v. s.
Output: v. s.
Effetti collaterali: v. s.

DATA >002C Cosine Routine(COS)-Calcola il coseno di un numero.
Input: v. s.
Output: v. s.
Effetti collaterali: v. s.

DATA >002E Sine Routine(SIN)-Calcola il seno di un numero.
Input: v. s.
Output: v. s.
Effetti collaterali: v. s.

DATA >0030 Tangent Routine(TAN)-Calcola la tangente di un numero.
Input: v. s.
Output: v. s.
Effetti collaterali: v. s.

DATA >0032 Arctangent Routine(ATN)-Calcola l'arcotangente di un numero.
Input: v. s.
Output: v. s.
Effetti collaterali: v. s.

Prima di chiamare una routine GPL occorre controllare se la memoria usata dal programma e' accessibile e modificabile dalla routine(v. Effetti collaterali su descritti). Poiche' la CPU RAM e' usata da molti programmi di sistema e' facile non accorgersi dell'informazioni immagazzinate li'. Inoltre qualcuna di queste rou-

tines usa fino a 26 bytes dell'interprete BASIC iniziando alla locazione >03C0 in VDP RAM. Percio' occorre porre attenzione quando si usa una routine GPL.

COLLEGAMENTO CON LE ROUTINES RESIDENTI IN ROM - XMLLNK

```
Format:  BLWP @XMLLNK    Fa corrispondere XMLLNK A >601C
         DATA c-r-c      (console-routine-code)Definisce
                         il codice ID della routine in
                         ROM che deve essere eseguita.
               o

         DATA c-r-a      (console-routine-address)Fornisce
                         L'indirizzo della routine in ROM che
                         deve essere eseguita.
```

Si puo' accedere alle routines nella ROM della console tramite la routine XMLLNK. Si puo' accedere ad una routine nella rom di console in uno o due modi.
Un modo e' di specificare il codicew della routine in un'istruzione DATA. Il byte basso dell'istruzione DATA dovrebbe essere posto a zero. Per es.:

```
BLWP @XMLLNK
DATA >1200
```

salta alla routine CFI (che converte un numero in virgola mobile in un numero intero) della console.
Una lista di routines XML che possono essere chiamate da un programma in liguaggio assembly sono date nella seguente tabella.

Codice	Nome	Funzione
06	FADD	Addizione in virgola mobile
07	FSUB	Sottrazione " "
08	FMULT	Moltiplicazione " "
09	FDIV	Divisione " "
0A	FCOMP	Confronto " "
0B	SADD	Addizione in stack in virgola mobile
0C	SSUB	Sottrazione " " "
0D	SMULT	Moltiplicazione " " "
0E	SDIV	Divisione " " "
0F	SCOMP	Confronto " " "
10	CSN	Converte una stringa in un numero
12	CFI	Converte un numero in virgola mobile in un intero
17	VPUSHG	Introduce un valore nello stack
18	VPOP	Estrae mun valore dallo stack
23	CIF	Converte un numero intero in un numero in virgola mobile.

Il codice XML ,che e' di un solo byte, e' suddiviso in un nibble alto contenente l'indirizzo della tabella XML e in un nibble basso contenente l'indice nella tabella. Ci sono 16 indici nella tabella definiti nello spazio di indirizzamento della CPU. Il nibble alto specifica da quale delle 16 tabelle prendere l'indirizzo di salto(branch address) e quello basso determina quale dei 16 indirizzi nella tabella deve essere usato. Ogni tabella puo' contenere fino a 16 indirizzi con punto d'ingresso a due bytes.
Un altro modo di accedere alla routine nella ROM di console e' di specificare i suoi indirizzi nell'istruzione DATA. Si noti che

il bit alto della parola in DATA deve essere acceso cosicche' il programma di sistema possa riconoscere quel dato come un indirizzo invece di un codice XML. Per es.

```
BLWP @XMLLNK
DATA >8D3A
```

salta all'indirizzo ROM di console >0D3A che e' la routine di confronto in virgola mobile.

AVVERTENZA

Usando direttamente gli indirizzi di memoria delle routines ROM di console si fa in modo che il programma in liguaggio assembly che chiama la routine diventa completamente dipendente dalla macchina. Poiche' gli indirizzi di memoria delle routines ROM di console possono cambiare con modifiche future, l'uso di questo metodo di accesso dovrebbe essere ristretto a casi dove non c'e' nessun altro modo ragionevole di arrivare al risultato richiesto.

Il FAC (il Floating Point Accumulator) parte all'indirizzo >834A, ARG(che contiene gli argomenti)parte all'indirizzo >835C e STACK e' all'indirizzo >836E. Il byte STATUS e' all'indirizzo >837C. Tutti gli errori overflow, eccetto che in CFI, ritornano >01 all'indirizzo >8354.

DATA >0600 Floating Point Addition(FADD)-Addiziona due valori
- Input: FAC-Primo valore
  ARG-Secondo valore
- Output: FAC-Risultato dell'addizione

DATA >0700 Floating Point Subtraction(FSUB)-Sottrae due valori
- Input: FAC-Valore che deve essere sottratto
  ARG-Valore da cui FAC e' sottratto
- Output: FAC-Risultato della sottrazione

DATA >0800 Floating Point Multiplication(FMULT)-Moltiplica due valori. v. s.

DATA >0900 Floating Point Division(FDIV)-Divide due valori.
- Input: FAC-Divisore
  ARG-Dividendo
- Output: FAC-Risultato

DATA >0A00 Floating Point Compare(FCOM)-Confronta due numeri in virgola mobile.
- Input: FAC-Primo argomento
  ARG-Secondo argomento
- Output: STATUS-A seconda del risultato. Il bit alto e' acceso se ARG e' logicamente piu' alto di FAC. Il bit 'maggiore di' e' acceso se ARG e' aritmeticamente maggiore di FAC. Il bit 'equal' e' acceso se ARG e FAC sono uguali.

DATA >0B00 Value Stack Addition(SADD)-Addiziona usando uno stack in VDP RAM.
- Input: STACK-Indirizzo in VDP RAM dove il termine a sinistra e' posto

FAC-Valore a destra
Output: FAC-Risultato

DATA >0C00 Value Stack Subtraction(SSUB)-Sottrae usando uno stack in VDP RAM.
Input: STACK-L'indirizzo in VDP RAM che contiene il termine a sinistra
FAC-Valore che deve essere sottratto
Output: FAC-Risultato

DATA >0D00 Value Stack Multiplication(SMULT)-Moltiplica usando uno stack in VDP RAM.
Input: STACK-Indirizzo VDP RAM che contiene il moltiplicando.
FAC-Moltiplicatore
Output: FAC-Risultato

DATA >0E00 Value Stack Division(SDIV)-Divide usando uno stack in VDP RAM.
Input: STACK-Indirizzo VDP RAM che contiene il dividendo.
FAC-Divisore
Output: Risultato

DATA >0F00 Value Stack Compare(SCOMP)-Confronta un valore nello stack in VDP RAM con il valore in FAC.
Input: STACK-Indirizzo in VDP RAM che contiene il valore da confrontare.
FAC-L'altro valore posto a confronto
Output: STATUS-Posto secondo del risultato. Il bit alto e' acceso se il valore puntato dallo STACK e' logicamente piu' alto di FAC. Il bit 'maggiore di' e' acceso se il valore puntato da STACK e' aritmeticamente maggiore di FAC. Il bit 'eguale a'e' acceso se i valori puntati da STACK e FAC sono uguali.

DATA >1000 Convert String to Number(CSN)-Converte una stringa ASCII in un numero in virgola mobile.
Input: FAC+12-Indirizzo della stringa in VDP RAM.
Output: FAC-Risultato della conversione in formato in virgola mobile.

DATA >1200 Convert Floating Point to Integer(CFI)-Converte un numero in virgola mobile in un intero.
Input: FAC-Il numero da convertire
Output: FAC-Il valore intero di una parola. Il valore massimo e' >FFFF. Se capita un overflow, FAC+10 (>8354)e' posto al codice di errore overflow tipo >03.

DATA >1700 Push Value onto Value Stack(VPUSHG)-Introduce un valore da FAC dentro il valore dello stack.

DATA >1800 Pop Value from Value Stack(VPOP)-Estrae un valore dallo stack e lo pone in FAC.

DATA >2300 Convert Integer to Floating Point(CIF)-Converte un intero in un numero in virgola mobile.
Input: FAC-Il valore intero da convertire

Output: FAC-Il risultato in virgola mobile

COLLEGAMENTO ALLE ROUTINES DI SERVIZIO DEI DEVICES-DSRLNK

Format: BLWP @DSRLNK Fa corrispondere DSRLNK a >6038
DATA c-r-c (console-routine-code)Definisce il codice della routine che deve essere eseguita.

DSRLNK collega un programma in linguaggio assembly ad ogni Device Service Routine(DSR) o sottoprogramma in ROM. Il dato da fornire e' >8 per il collegamento ad una DSR routine e >10 per il collegamento ad un sottoprogramma. Prima che la routine sia chiamata un Peripheral Access Block(PAB) deve essere inizializzato in VDP RAM. Un PAB e' un blocco di memoria che contiene informazioni su un file che deve essere esplorato. In piu', gli indirizzi CPU RAM >8356 e >8357 devono contenere un puntatore al device o al nome del sottoprogramma in PAB.
Dopo che la routine e' stata eseguita, l'informazione e' passata a ritroso al programma in linguaggio assembly nell'area UTLTAB. Per es. si supponga che le seguenti istruzioni siano state eseguite.

```
     REF   DSRLNK


     ...
     BLWP  @DSRLNK
     DATA  >8
```

Se non capita nessun errore il bit di 'equal' nello Status Register viene spento al ritorno da DSRLNK. Se un errore di I/O capita, il bit di 'equal' e' acceso e il codice di errore e' memorizzato nel byte piu' significativo del Registro 0 del workspace del programma chiamante.
Se chiamando la RS232 DSR routine il programma deve conservare e poi reimmagazzinare i valori posti nella GROM-Read e gli indirizzi GROM-Write. Il seguente segmento di programma mostra come mettere a posto questi valori.

```
        REF  GRMRA
        REF  GRMWA


        ...
SAVEG   BSS  2


        ...
        MOVB @GRMRA, @SAVEG
        MOVB @GRMRA, @SAVEG+1


        ...
        BLWP @DRSLNK
        DATA >8


        ...
        MOVB @SAVEG, @GRMWA
```

```
MOVB @SAVEG+1,@GRMWA
```

Nota: Poiche' la routine DSR per il reg.a cassetta e' in GROM, deve essere raggiunta tramite GPLLNK piuttosto che con DSRLNK. Per accedere ad una cassetta si usi BLWP @GPLLNK con DATA >003D.

UTILITA' D'INTERFACCIA CON IL TI BASIC

Queste utilita' permettono ad un programma in linguaggio assembly di leggere o ssegnare valori a variabili passate in una lista di parametri da un'istruzione CALL LINK in un programma TI BASIC. Queste routines d'utilita' includono utilita' con argomento passante e un'utilita' di error-reporting(rapporto sugli errori). Tutte le routines ad argomento passante usano la propria area di workspace localizzata a >7092. Comunque, tutti i parametri sono passati attraverso la workspace del programma chiamante.
Le seguenti sezioni descrivono le convenzioni di passaggio dati e la sintassi dell'istruzione chiamante per ciascuna routine.

ASSEGNAZIONE NUMERICA - NUMASG

Format: BLWP @NUMASG Fa corrispondere NUMASG a >6040

Questa routine assegna un valore numerico ad una variabile numerica passata come argomento.

- R0 Zero se si usa una variabile semplice numerica o il numero di elementi di una tabella. La routine d'utilita' dell'assegnazione controlla le codizioni di legalita'. Con un OPTION BASE 0, il numero di elementi deve stare entro il campo da 0 a (massimo numero di elementi-1). Con OPTION BASE 1 il numero degli elementi deve stare nel campo da 1 al massimo numero di elementi.
- R1 Numero di argomenti(un'intera parola)come appare nella lista degli argomenti della CALL LINK
- >834A Area FAC. Contiene un valore a 8 bytes in virgola mobile che deve essere assegnato alla variabile.

Se l'argomento richiesto non e' una variabile numerica o un elemento di una tabella numerica, viene emesso un segnale d'errore.

ASSEGNAZIONE DI STRINGA - STRASG

Format: BLWP @STRASG Fa corrispondere STRARG a >6048

Questa routine assegna una stringa a una variabile stringa passata come un argomento al programma in linguaggio assembly.
L'utilita' fa le seguenti azioni:

- Alloca spazio per la stringa in VDP RAM
- Copia la stringa nella VDP RAM allocata
- Assegna la stringa alla variabile scelta
- Modifica il punto d'ingresso allo stack per puntare alla nuova stringa. La stringa che deve essere assegnata deve essere creata in RAM dal programma in linguaggio assembly. Il primo byte della stringa contiene la lunghezza della stringa.

I registri sono assegnati con i seguenti valori:

R0 Zero se una stringa e' assegnata ad una variabile stringa semplice o il numero di elementi di una tabella se assegnato ad un elemento di una tabella. Con OPTION BASE 0 il numero di elementi deve stare nel campo da 0 a (il max numero di elementi -1). Con OPTION BASE 1 il numero di elementi deve stare nel campo tra 1 e il massimo numero di elementi.

R1 Numero di argomenti come appare nella lista degli argomenti di un'istruzione CALL LINK. (una parola intera)

R2 Indirizzo di una stringa che deve essere assegnata. La stringa deve essere in RAM.

Se l'argomento specificato non e' una variabile stringa o un elemento di una tabella stringa viene emesso in messaggio d'errore.

RECUPERO DI UN PARAMETRO NUMERICO - NUMREF

Format: BLWP @NUMREF Fa corrispondere NUMREF a >6044

Questa utilita' recupera il valore di un parametro numerico.

R0 Numero di un elemento di tabella se l'argomento e' una tabella numerica; altrimenti zero.

R1 Numero del parametro come appare nella lista degli argomenti nell'istruzione CALL LINK.

>834A Area FAC. L'indirizzo iniziale del valore a 8 bytes di un parametro numerico ritornato dalla routine d'utilita'.

RECUPERO DI UN PARAMETRO DI STRINGA - STRREF

Format: BLWP @STRREF Fa corrispondere STRREF a >604C

Questa routine recupera il valore di un parametro di stringa. Il programma deve allocare spazio nella memoria RAM prima di chiamare questa routine e il primo byte di questo buffer allocato deve contenere la massima lunghezza richiesta. Se la stringa non risulta adattabile al file viene segnalato un errore di condizione.

R0 Numero dell'elemento della tabella se l'argomento e' una tabella stringa; altrimenti zero.

R1 Numero di parametri come appare nella lista degli argomenti dell'istruzione CALL LINK.

R2 Indirizzo del buffer

Se la stringa e' posta nel buffer la stringa e' copiata dentro al buffer secondo il byte lunghezza e il byte di lunghezza e' modificato per riflettere l'attuale lunghezza della stringa.

RAPPORTO SUGLI ERRORI - ERR

Format: BLWP @ERR Fa corrispondere ERR a >6050

Questa routine trasferisce il controllo alla routine di rapporto sugli errori nell'interprete TI BASIC. Il programma in lin-

guaggio assembly puo' riferire su ogni errore TI BASIC esistente o su messaggi d'avvertimento dopo il ritorno al TI BASIC.

RO Codice d'errore nel byte piu' significativo

I messaggi d'errore che possono essere emessi dal programma sono listati nella tabella seguente.

ATTENZIONE

I codici d'errore piu' piccoli di >10 sono riservati alla MiniMemory. Percio'usando questi codici in un programma si puo' causare effetti collaterali imprevedibili.

Codice	Messaggio d'errore
00	Errore DSR - Nome errato
01	" Scrittura protetta
02	" Attributo errato
03	" Operazione illegale
04	" Buffer pieno
05	" Lettura dopo un EOF
06	" Errore su device
07	" Errore in un file
08	Memoria piena(chiude il file)
09	Istruzione scorretta(N/A)
0A	Tag errato
0B	Errore durante un controllo
0C	Definizione duplicata
0D	Referenze non definite
0E	Istruzione scorretta(N/A)
0F	Programma non trovato
10	Istruzione scorretta
11	Nome errato
12	Non si puo' continuare
13	Valore errato
14	Numero troppo grande
15	Errore su stringa
16	Argomento errato
17	Indice errato
18	Conflitto di nomi
19	Non si puo' fare questo
1A	Numero di linea errato
1B	Errore in un FOR-NEXT
1C	Errore in un I/O(assume L'indirizzo PAB in >8304)
1D	Errore in un file
1E	Errore in input
1F	Errore in un dato
20	Linea troppo lunga
21	Memoria piena(non chiude il file)
22	Errore di sintassi
23	Overflow numerico
24	Carattere irriconoscibile
25	Stringa troncata
26-FF	Errore sconosciuto

TRADUZIONE DEL MANUALE EASY BUG DEBUGGER DEL TI-99/4A

L'EASY BUG e' un utile e potente strumento di sviluppo programmi con cui si puo' correggere programmi in linguaggio assembly e accedere alle porte di input e/o output (I/O) dlla memoria del computer. Con EASY BUG si puo':

-Ispezionare e ,opzionalmente, modificare il contenuto della CPU e della VDP RAM.
-Visualizzare il contenuto della GROM.
-Eseguire programmi in linguaggio assembly da EASY BUG.
-Accedere direttamente ai devices periferici che sono collegati al computer via la porta seriale del microprocessore TMS9900, cioe' la Communications Register Unit(CRU).
-Salvare e caricare programmi su cassetta.

OPERAZIONI

Quando si sceglie l'opzione EASY BUG dalla lista di selezione principale sullo schermo si visualizzano i seguenti comandi:

===COMMAND TYPES ARE===		TRADUZIONE
MXXXX	MODIFY CPU MEMORY	Modifica la memoria CPU
GXXXX	DISPLAY GROM MEMORY	Visualizza la memoria GROM
VXXXX	MODIFY VDP MEMORY	Modifica la memoria VDP
EXXXX	EXEC. ASSEMBLY PROGRAM	Esegue un prog. assembly
SXXXX	SAVE CPU MEMORY TO CS1 (STARTING AT XXXX)	Salva la mem. CPU su CS1 (partendo da xxxx)
L	LOAD STORAGE FROM CS1	Carica la memoria da CS1
===SPECIAL FUNCTION KEYS ARE===		Tasti funzione speciali
'AID'	DISPLAY THIS SCREEN	Visualizza questo schermo
PERIOD	ABORT A COMMAND	Cancella un comando
ENTER	ENTER COMMAND/DATA	Introduci un comando/dato
MINUS	DISPLAY LAST MEMORY (CURRENT UNCHANGED)	Visualizza l'ultima mem.
SPACE	DISPLAY NEXT MEMORY (CURRENT UNCHANGED)	Visualizza la prossima mem
NOTE	CPU RAM 8370-83FF IS RESERVED FOR EASY BUG	Le locaz. 8370-83FF sono riservate all'EASY BUG.

Questa schermata riassume i comandi e i tasti funzione speciali usati con L'EASY BUG. Le X seguenti le varie lettere comando indicano un indirizzo esadecimale da introdurre.
Premere un tasto qualsiasi eccetto QUIT per cancellare lo schermo e ricevere il segnalino (prompt) espresso con il punto interrogativo per chiedere l'uso di un comando.

COMANDI E FUNZIONI SPECIALI

Un comando espresso con una sola lettera e' usato per eseguire ogni routine di EASY BUG. Ogni comando(con eccezione del comando Load Storage)deve essere seguito da una a quattro cifre esadeci-

mali indicanti un indirizzo. Se sono introdotte piu' di quattro cifre vengono usate solo le ultime quattro. Se si introducono meno di quattro cifre vengono trattate come le ultime cifre di un valore a quattro cifre, essendo le prime uguali a zero.
Dopo aver scritto un comando e un indirizzo occorre premere ENTER per eseguire il comando.

M (Modifica la memoria CPU)	Permette di ispezionare e, anche, cambiare il contenuto della CPU.
G (Visualizza la mem. GROM)	Permette di visualizzare il contenuto della GROM.
V (Modifica la mem. VDP)	Permette di ispezionare e anche di cambiare il contenuto della memoria VDP
E (Esegue un prog. assembly)	Permette di eseguire un programma assembly in CPU RAM.
C (CRU Single-bit I/O)	Permette di ispezionare e anche di cambiare bits singoli di I/O
S (Salva la mem. CPU)	Permette di trasferire il contenuto della CPU in una cassetta.
L (Carica da mem. esterna)	Permette di caricare un progr. assembly da cassetta nella CPU.

Per fermare un'operazione di comando occorre premere il tasto PUNTO (.). Il punto interrogativo riapparira'.
I tasti funzione ENTER, MENO, BARRA SPAZIATRICE sono usati con i comandi M, G, V e C. Le funzioni di questi tasti sono inclusi nella descrizione di questi comandi.
Occorre premere AID per tornare alla schermata dell'EASY BUG dopo che lo schermo e' stato cancellato. Questo tasto opera solo quando viene introdotto immediatamente dopo un punto interrogativo dell'EASY BUG.
Ogni comando dell'EASY BUG e' descritto nelle sezioni seguenti.

MODIFY CPU MEMORY - M

Format: Mxxxx (dove xxxx e' un valore esadecimale)

Questo comando visualizza il contenuto di una locazione di memoria CPU scelta e da' la possibilita' di cambiare il dato nella locazione. Se la locazione di memoria non e' specificata, viene presa come >0000. Dopo aver scritto il comando e l'indirizzo e premuto ENTER l'indirizzo di memoria specificato e il suo contenuto vengono visualizzati. Per cambiare il contenuto all'indirizzo visualizzato, si scriva un valore esadecimale a due cifre e si prema ENTER. Le ultime cifre scritte sono il valore usato; percio' se si fa un errore nell'introduzione di un valore, semplicemente basta riscrivere le ultime due cifre corrette. Si noti che i tasti con le frecce a sinistra e a destra non lavorano con l'EASY BUG.
Dopo che si e' visualizzato una locazione di memoria e il suo contenuto si puo' premere la barra per avanzare di una locazione o il tasto del segno MENO per retrocedere.
Si noti che se si scrive un valore seguito da uno spazio o un meno il contenuto della locazione di memoria non viene modificato. Solo quando si preme ENTER direttamente dopo aver scritto un valore il contenuto si cambia. Scrivendo un PUNTO (.) termina il comando e viene visualizzato il punto interrogativo.
La CPU RAM risiede in console, nel modulo MM e nell'EA se colle-

gate. E' direttamente indirizzabile da un programma in linguaggio assembly.
Il seguente esempio inspeziona il contenuto delle locazioni di memoria >8300, >8301, e >8302; cambia il contenuto di >8302 a >F7; cambia il contenuto di >8303 a >12; e rivisualizza il contenuto di >8302 e >8303. Finalmente il contenuto di >8304 e' ispezionato ma non e' cambiato poiche' il valore introdotto (>3C) non e' stato seguito da ENTER. Scrivendo un PUNTO termina il commando e ritorna il punto interrogativo.

Visualizzazione	Introduzioni
?	M8300 <ENTER>
M8300 =00->	<ENTER>
M8301 =00->	<SPACE>
M8302 =00->	F7<ENTER>
M8303 =00->	8A12<ENTER>
M8304 =00->	<MINUS>
M8303 =12->	<MINUS>
M8302 =F7->	<SPACE>
M8303 =12->	<SPACE>
M8304 =00->	3C<SPACE>
M8305 =00->	<PERIOD>
?	

ATTENZIONE

Occorre non modificare il contenuto della memoria CPU negli indirizzi da >8370 a 83FF poiche' questa area di memoria e' usata da EASY BUG.

MODIFY VDP MEMORY - V

Format: Vxxxx

Questo comando visualizza il contenuto di un indirizzo scelto nella memoria VDP e fornisce la possibilita' di cambiare i dati a quell'indirizzo. Se la locazione di memoria non e' specificata viene usata >0000.
Nota: Poiche' la VDP RAM non si estende al di la' di 3FFF questo e' il piu' grande indirizzo che si puo' introdurre per un comando di modifica. Se si sceglie un indirizzo maggiore il valore e' visualizzato ma questa locazione "fantasma" non puo' essere variata. Per il resto, questo comando opera come il comando M.
La VDP RAM consiste di 16K bytes di memoria posta tra gli indirizzi >0000 e >3FFF. Contiene di solito informazioni correlate allo schermo e usate dal Video Display Processor come immagini sullo schermo, sprite, tabelle di colori e tavole di caratteri. E' anche usata in generale come spazio di immagazzinamento di programmi applicativi. In particolare la memoria piu' alta e' usata dalle routines DSR per passare informazioni I/O. I programmi applicativi usano inoltre parte della VDP RAM come buffer per le DSR e come PAB per passare informazioni su un file ad una appropriata DSR. Si veda l'appendice E per maggiori informazioni. Quando il linguaggioTI BASIC viene usato, la VDP RAM contiene il programma BASIC, la Symbol Table del programma.

il valore di stack, lo spazio delle stringhe ecc.
Occorre non alterare la VDP RAM senza avere sufficienti conoscenze dell'interprete BASIC poiche' l'interprete usa la VDP RAM in un modo speciale. Una dettagliata configurazione della VDP RAM mentre si usa il TI BASIC e' mostrata in Appendice F.
Poiche' la VDP RAM non e' direttamente indirizzabile dalla CPU il codice del linguaggio assembly del TMS9900(istruzioni e workspace inclusi)non possono essere eseguiti in VDP RAM.

DISPLAY GROM MEMORY - G

Format: Gxxxx (dove xxxx e' un valore esadecimale)

Questa comando e' usato per visualizzare il contenuto di locazioni di memoria GROM. Se la locazione di memoria non e' specificata con il comando , viene usata la >0000. Poiche' la GROM e' una memoria di sola lettura non e' possibile alterare il contenuto di queste locazioni. Per il resto questo comando lavora come il comando M. Il computer puo' indirizzare fino a 8 GROM di cui tre GROM in console e fino a cinque in un modulo SSS. Il numero di GROM in un modulo SSS dipende dalla lunghezza del programma nel modulo stesso.
Gli indirizzi GROM vanno da >0000 a >F7FF. Ogni GROM ha 6Kbytes di memoria che parte da un indirizzo con un numero pari del primo digit. Per es. la GROM 0 parte dall'indirizzo >0000 e occupa fina a 17FF; la GROM 1 parte da >2000 e occupa spazio fino a 37FF.
Di seguito c'e' la mappa dello spazio di memoria delle GROM.

GROM 0	Locazioni da	>0000	a >17FF	¦
GROM 1		2000	37FF	¦ Contenute nella console
GROM 2		4000	57FF	¦
GROM 3		6000	77FF	-
GROM 4		8000	97FF	-
GROM 5		A000	B7FF	- Contenute nel modulo SSS
GROM 6		C000	D7FF	-
GROM 7		E000	F7FF	-

EXECUTE ASSEMBLY PROGRAM - E

Format: Exxxx (dove xxxx e' un numero esadecimale)

Questo comando e' usato per eseguire un programma in linguaggio assembly. Il controllo del programma viene passato alla locazione specificata. Questo indirizzo deve essere un punto d'ingresso di un programma in linguaggio assembly. Se non viene specificata la locazione di memoria con un comando, viene usata la >0000.

CRU SINGLE-BIT I/O - C

Format: Cxxxx (dove xxxx e' un numero esadecimale)

Questo comando e' usato per visualizzare e, opzionalmente, cambiare il bit CRU alla locazione specificata. Se la locazione non e' specificata con un comando, viene presa la >0000.
Dopo aver scritto il comando e l'indirizzo e premuto ENTER, e' visualizzato l'indirizzo specificato con vicino lo stato del bit (0 o 1). Lo stato del bit e' indicato dalla cifra meno significativa di un valore a due cifre. La cifra a sinistra e' 0.
Per esempio la seguente visualizzazione di:

C0201 =00 ->

indica che il bit all'indirizzo >0201 e' a zero(cifra meno significativa di un valore a due cifre); invece

C0202 =01 ->

indica che il bit all'indirizzo >0202 e' 1.
Per cambiare lo stato del bit introduci 0 o 1.

SAVE CPU MEMORY TO CS1 - S

Format: Sxxxx (dove xxxx e' un numero esadecimale)

Questo comando copia il contenuto della memoria CPU sulla cassetta 1 partendo dall'indirizzo specificato. Questo comando e' usato per salvare il contenuto di un programma o di dati su una cassetta per poter essere ricaricato. Se non e' specificato l'indirizzo la partenza e' a >0000. Dopo aver scritto l'indirizzo di partenza e premuto ENTER viene visualizzato l'indicatore

TO ?

Occorre ora introdurre l'ultima locazione di memoria che si vuol copiare su cassetta. Dopo aver introdotto l'indirizzo e premuto ENTER il campo di memoria scelto sara' copiato sulla cassetta dell'unita 1 (CS1).
Nota: per salvare il contenuto del modulo MM incluse riferimenti e puntatori occorre introdurre l'indirizzo di partenza >7000 e l'indirizzo di fine >7FFF.

LOAD STORAGE FROM CS1 - L

Format: L

Questo comando carica un programma da cassetta. Il programma e' caricato nello stesso spazio di memoria occupato durante il salvataggio con il comando S.

Quando compare il punto interrogativo (?) sullo schermo occorre premere L per caricare il programma da cassetta. La procedura e' la stessa del caricamento di programmi da TI BASIC.

LINE-BY-LINE ASSEMBLER DELL'HOME COMPUTER TEXAS 99/4A

Introduzione

La cassetta compresa nella Minimemory contiene un assembler simbolico del tipo line-by-line (una linea alla volta) e un programma dimostrativo grafico chiamato LINES(LINEE). L'assembler permette di introdurre il codice sorgente del linguaggio assembly del TMS9900, una linea alla volta, direttamente da tastiera. Il programma dimostrativo LINES disegna automaticamente linee colorate sullo schermo.

Quando il programma Assembler e' caricato nel modulo MiniMemory, ogni istruzione sorgente e' immediatamente assemblata nel codice oggetto e memorizzata in locazioni di memoria specificate dal codice sorgente.

Percio', appena e' completata l'introduzione del programma e memorizzato il suo nome e indirizzo nella tavola REF/DEF, e' pronto per essere eseguito.

Come si carica l'Assembler Line-by-line

L'Assembler Line-by-line e il programma grafico LINES sono caricati contemporaneamente dalla cassetta con il comando L (LOAD) dell'opzione del debugger (correttore) EASY BUG. Di seguito e' descritta la procedura:

1. Quando la lista di selezione compare sullo schermo, si seleziona l'opzione MINIMEMORY, poi si preme 3 per riinizializzare (REINITIALIZE) la memoria e caricare un nuovo programma. Si preme QUIT per ritornare alla lista principale e poi si seguono i comandi soliti dell'EASY BUG.

2. Si preme QUIT, si sceglie l'opzione MINIMEMORY e poi l'opzione 2 (RUN); alla richiesta del nome del programma si puo' scegliere NEW (per un nuovo programma) o OLD (per continuare uno gia' iniziato).

La sintassi dell'ASSEMBLER

Ogni linea (line o record)del programma sorgente consta di quattro sezioni chiamate 'campi'. Questi campi, se presenti(qualcuno e' opzionale), devono comparire nell'ordine e nel formato(sintattico)richiesto dall'Assembler. In questo manuale, si usano le seguenti convenzioni nelle definizioni sintattiche per le istruzioni e per le direttive di macchina.

- I termini in lettere MAIUSCOLE, inclusi i caratteri speciali, devono essere introdotti esattamente come mostrato.
- I termini entro parentesi quadre ([]) sono opzionali.
- I temini entro le parentesi angolate (<>)sono campi richiesti.
- Un b minuscolo indica una spaziatura.
- Un b minuscolo seguito da tre puntini(b...)indica una o piu' spaziature.

La sintassi generale per una direttiva Assembler e' la seguente:

[etichetta]b...<opcode>b[operando][,operando][b...commento]

Il campo etichetta (label) richiede una spaziatura(quando non c'e' etichetta)o uno o due caratteri. Il primo carattere deve essere alfabetico;il secondo,se presente,puo' essere alfanumerico. L'etichetta e' seguita da uno o piu' spaziature. se non si mette un'etichetta,premendo la barra spaziatrice si muove il cursore automaticamente fino all'inizio del campo opcode(codice operazione).
Il campo opcode contiene il codice operazione dell'azione che deve essere eseguita dall'istruzione sorgente. Il campo consta da uno a quattro caratteri alfabetici,come A per ADD(aggiungi) o AORG per la direttiva Origine Assoluta,seguita da una sola spaziatura.
Il campo operando consta di uno o due operandi,come richiesto dalla particolare istruzione. Si noti che il campo operando non ha all'interno nessuna spaziatura e gli operandi multipli sono separati da virgole. Il campo operando si conclude premendo la barra(il cursore avanza al campo commento)o ENTER(che significa la fine della linea). Se un'istruzione non ha operando,il campo operando e' omesso.
Il campo commento,se usato,puo' includere qualsiasi carattere e continua finche' non si preme ENTER per finire la linea.

Esempi

```
XY      MOV R1,@VP    Salva R1 in VP
Z       S R1, R2      Calcola la differenza.
```

L'Assembler Line-by-line predefinisce certi simboli. Quando un operando include il simbolo dollaro ($) come carattere iniziale, si considera che venga riferito al contenuto del contatore di locazione. Per esempio,alla locazione >7D00, le istruzioni

```
                JMP $+8
```

e

```
                JMP >7D08
```

devono essere considerate equivalenti. Quando si specificano gli operandi di registro,si puo' usare il simbolo R,seguito da un numero decimale. Percio',

```
                MOV R1,R15
```

e

```
                MOV 1,15
```

sono equivalenti.

Nota: il sistema numerico di default (iniziale)per l'Assembler e' il decimale;i numeri esadecimali sono indicati con il prefisso 'maggiore' (>).

Direttive Assembler

Questa sezione discute le sette direttive riconosciute dall'Assembler Line-by-line. Una direttiva Assembler non deve essere confusa con un'istruzione di linguaggio assembler che dice al microprocessore di eseguire solo una singola funzione,cosi' come ADD

o MOVE. Le direttive sono comandi ad aiuto programmato che guidano l'Assembler ad eseguire certe operazioni in assembly e L'Assembler puo' eseguire molte istruzioni per soddisfare una direttiva.
Le direttive descritte qui sono

AORG	Origine Assoluta
BSS	Blocco di partenza con simbolo
DATA	Parola di Inizializzazione
END	Fine programma
EQU	Costante di definizione in assembly
SYM	Visualizzazione tavola dei simboli
TEXT	Costante d'inizializzazione di stringa

AORG-Absolut Origin(Origine assoluta)

Format: [etichetta] AORG <indirizzo>

Questa direttiva puo' essere usata per posizionare il contatore di locazione ad uno specifico valore(sempre un indirizzo dispari) durante un'operazione Assembler. Generalmente, viene usata come primo ingresso al programma per posizionare la locazione di partenza del codice assemblato; comunque, puo'essere usato in ogni momento.

Esempio: AORG >7D80

BSS-Block Starting with Symbol(Blocco di partenza con simbolo)

Format: [etichetta] BSS <numero di bytes che devono essere riservati>

La direttiva BSS riserva un blocco di memoria(per magazzino variabili o registri di lavoro)senza inizializzare lo spazio. Partendo dall'indirizzo specificato dall'etichetta, l'Assembler incrementa il contatore di locazione del numero di bytes specificato nella direttiva.
Il numero di bytes deve essere zero o positivo. Il valore risultante in un contatore di locazioni e' arrotondato per difetto al numero pari, se necessario. In altre parole, il bit meno significativo dell'indirizzo e' troncato se il valore risultante e' dispari.

Esempio: WS BSS 32 Assumendo che WS si riferisca alla locazione >7D00, incrementa il contatore di locazioni a 7D20, riservando un blocco di 32 bytes come area di lavoro.

DATA-Word Initialization(Inizializzazione di parola).

Format: [etichetta] DATA <valore>

La direttiva DATA permette di inizializzare una parola o alcune parole di memoria ad un valore particolare. La direttiva e' particolarmente utilizzata per inserire una tabella di dati come parte del programma. In ogni punto nell'assembly del programma, si

puo' inserire una direttiva DATA nel campo opcode, seguita da una costante o un simbolo come un operando.
L'operando per una direttiva DATA puo constare di un riferimento dapprima indefinito, una costante numerica, un simbolo definito o una stringa di costanti numeriche e simboli definiti uniti col segno piu'(+) e meno(-).
Se l'operando e' il secondo(una stringadi somme e differenze), nessun controllo e' fatto per il riporto o l'overflow.

Esempi:

DATA	>1234	Impone che la locazione venga inizializzata con >1234
DATA	AX	Se AX = >3456 impone la sua inizializ. con >3456
DATA	GH	Se GH e' un riferimento prima indefinito, la locazione sara' inizializzata con il valore corrispondente a GH quando GH viene definito.
DATA	1+5-3	Inizializza la locazione a 3(equivalente a DATA 3)

La direttiva DATA accetta anche una sequenza di costanti separate da virgole.

DATA [costante(definita o indefinita), cost, cost, . . . , cost]

Nota che una costante indefinita e' accettabile solo come primo operando dell'istruzione.

Le direttive BSS e DATA hanno funzione simile; comunque, la direttiva BSS semplicemente posiziona a parte spazio di memoria senza inizializzarla, mentre la direttiva DATA e posiziona a parte spazio di memoria e la inizializza ad un valore specifico(o a valori specifici).

END-End Program(Fine programma)

Format: END

L'Assembler puo' essere sempre fatto uscire introducendo la direttiva END nel campo opcode. Quando la direttiva END e' introdotta, l'Assembler visualizza il numero di riferimenti indefiniti, se ci sono. Se esistono riferimenti indefiniti, occorre tornare all'Assembler e definirli prima di uscire dall'Assembler.
La direttiva SYM(descritta qui sotto)e' di aiuto nell'identificare riferimenti indefiniti prima di terminare con END il programma. Quando tutti i riferimenti sono stati definiti, l'Assembler visualizza il report:

0000 UNRESOLVED REFERENCES

Premendo ENTER a questo punto si esce dall'Assembler e si ritorna alla lista di selezione della Minimemory. Premendo un altro tasto qualsiasi eccetto che ENTER si impone all'Assembler di attendere la prossima istruzione.

EQU-Assembly-Time Constant Definition(costante di definizione di tempo)

Format: <etichetta> EQU <costante definita>

La direttiva EQU(equate) e' usata per definire un valore per una costante simbolica e per assegnare il valore di un simbolo definito ad un altro simbolo.

Nota: Nessuna direttiva e' fornita per cambiare il valore di un simbolo una volta che e' stato definito.

Esempi:

```
CD      EQU   >A55A   Pone in CD il valore >A55A
FG      EQU   15      Pone in FG il valore 15
A       EQU   FG      Pone a uguale a FG
```

SYM-Symbol Table Display(visualizzazione della tavola dei simboli)

Format: SYM

La direttiva SYM permette di rivedere, sempre durante l'introduzione di un programma, i simboli di riferimento e i loro valori associati che sono usati nel programma in quel punto.

La tavola dei simboli e' visualizzata in tre categorie:

RIFERIMENTI DEFINITI	Ogni etichetta che e' stata definita con un valore.
RIFERIMENTI NON DEFINITI(PAROLE)	Ogni etichetta che e' stata definita in una istruzione eccetto che in una istruzione di salto.
RIFERIMENTI NON DEFINITI(SALTI)	Ogni etichetta che e' stata definita in una istruzione di salto.

Se una categoria non ha simboli associati con SYM, non viene visualizzata. Se un simbolo e' non definito(e' stato menzionato ma non definito), anche l'indirizzo del simbolo viene visualizzato. Se la tavola dei simboli e' vuota, la direttiva SYM e' cancellata, e l'Assembler attende per un'altra istruzione.

Esempi:

```
Locazione          Istruzione      Commenti

                   AORG >7D00      Posiziona l'indirizzo di par-
                                   tenza del programma
7D00 0000  WS      BSS 32          Si riserva lo spazio di lavoro
7D20 0201          LWPI WS         Carica lo spazio di lavoro
7D22 7D00
7D24 0201          LI R1,W1        Carica il Registro 1 con un
                                   dato indefinito
7D26R0000
7D28R10FF          JMP J1          Salta ad un indirizzo indef.
```

7D2A XXXX SYM Visualizza la Symbol Table

RESOLVED REFERENCES
WS-7D00

UNRESOLVED REFERENCES (WORD)
W1-7D26

UNRESOLVED REFERENCES (JUMP)
J1-7D28

7D2A XXXX (XXXX e' un dato esistente in memoria)
L'Assembler attende per un'altra istruzione.

Se un simbolo non definito e' menzionato in piu' di una locazione, l'indirizzo e il simbolo di ogni riferimento, fino a un max di 32 riferimenti, viene visualizzato.

TEXT-String Constant Initialization(costante d'inizializzazione di stringa)

Format: [etichetta] TEXT <stringa di caratteri>

La direttiva TEXT permette di introdurre una stringa di caratteri e averli tradotti in codice ASCII e memorizzati come parte di programma. Ogni carattere visualizzato, eccetto il carattere apice ('), puo' essere introdotto come parte di un'istruzione TEXT, e il codice ASCII per ogni carattere e' memorizzato in memoria come se si introducesse il carattere. Occorre notare anche che i tasti di controllo e funzioni speciali(AID, REDO, ecc.)generano codici ASCII validi(nel campo da >0 a >F)che sono memorizzati ma non visualizzati sullo schermo.
La stringa TEXT puo' essere lunga quanto desiderato, e deve essere preceduta e terminata da un carattere apice ('). Se si introduce un numero dispari di caratteri ASCII, viene aggiunto un byte nullo (>00)alla stringa per forzare il contatore di locazione ad un altro legame.

Esempio: TEXT 'ABCD' Immagazzina i valori >4142 e >4344 nelle corrispondenti locazioni di memoria.

Nota: La funzione ERASE(cancella) non funziona sui caratteri in memoria che sono gia' stati introdotti come parte di una stringa in TEXT.

La SYMBOL TABLE(tavola dei simboli)

L'Assembler permette parole e riferimenti di salto non definiti; cioe' riferimenti a simboli che vengono definiti piu' tardi nel programma. L'Assembler mantiene traccia di tutti i simboli definiti o menzionati in un programma e memorizza questa informazione in una tabella di simboli(Symbol Table).
La Symbol Table e' normalmente una combinazione di tre tavole: riferimenti di simboli definiti, riferimenti di parole non defini-

te ma solo menzionate, e riferimenti di salti non definiti ma solo menzionati. Anche il numero di introduzioni nella tabella e' memorizzato; poiche' ogni introduzione e' memorizzata con quattro bytes, la lunghezza fisica della tabella e' di quattro volte il numero di introduzioni.
La Symbol Table parte alla locazione di memoria >7CD8. Poiche' ogni introduzione alla Symbol Table e' lunga quattro bytes, occorre assicurarsi che l'indirizzo di partenza del codice oggetto permetta uno spazio adeguato per il numero di introduzioni alla Symbol Table richiesti per il programma. Altrimenti, quando il programma viene assemblato, la Symbol Table puo' ricoprire l'inizio del programma.

DEFINED SYMBOL REFERENCES

Se una introduzione e' un'etichetta definita o un simbolo definito, la parola etichetta memorizzata nella Symbol Table e' semplicemente l'equivalente ASCII del simbolo. Un simbolo di un solo carattere e' memorizzato nella sua forma ASCII preceduto dal codice ASCII per 1; per esempio, il simbolo A e' memorizzato come >3141 . La seconda parola(la parola indirizzo)contiene l'indirizzo, costante equivalente, o locazione di memoria corrispondente all'etichetta o al simbolo. Per esempio, se l'etichetta AC(ASCII >4143)e' definito da >8375, l'introduzione alla Symbol Table diventa:

4143	Punto d'ingresso del simbolo definito
8375	Valore

UNRESOLVED WORD REFERENCES

Il punto d'ingresso alla Symbol Table per il riferimento di una parola non definita e' simile a quella su descritta, eccetto che il bit piu' significativo della parola etichetta e' posizionato, e la parola indirizzo punta all'ultima locazione in cui questa etichetta fu usata precedentemente come un riferimento. Per esempio, se l'etichetta AC e' usata come un riferimento precedente non definito nella locazione >7E00, e non e' stata fatto nessun altro ulteriore riferimento a questa etichetta nel programma, la Symbol Table e'

C143	Punto d'ingresso del rif. della parola non definita
7E00	Indirizzo

UNRESOLVED JUMP REFERENCES

Il punto d'ingrasso alla Symbol Table per un riferimento di salto non definito, a meno che il bit piu' significativo del 'byte meno significativo' della parola etichetta sia posizionato e la parola indirizzo punti alla locazione dell'ultima istruzione di salto che usa questa etichetta non definita.
Per esempio, se la locazione >7D00 ha un riferimento di salto non definito all'etichetta AC e non e' stato fatto nessun altro riferimento a questa etichetta, il punto d'ingresso alla Symbol Table e'

```
41C3    Punto d'ingresso del rif.di salto non definito
7D00    Indirizzo
```

Il byte meno significativo dell'istruzione di salto non definito indica la distanza al riferimento di salto non definito piu' recentemente usato alla stessa etichetta. Se non c'e' riferimento precedente al byte e' assegnato il valore -1(>FF).

NUMERO MASSIMO DI RIFERIMENTI NON DEFINITI VISUALIZZATI

La prima volta che l'Assembler pone un simbolo non definito nella Symbol Table, i caratteri del simbolo sono posti nella tavola seguiti dall'indirizzo dove il simbolo e' stato menzionato. Questo indirizzo e' chiamato puntatore(pointer).
Il contenuto di questo indirizzo e' poi posto a zero, indicando con cio' che questo e' il primo riferimento al simbolo non definito.
Per esempio, si consideri il seguente programma come e' stato assemblato.

```
7D00 02E0              LWPI WS
7D02R0000
7D04 C820              MOV @WS,@DG
7D06R7D02
7D08R0000
                       SYM
UNRESOLVED REFERENCES (WORD)
WS-7D06 WS-7D02 DG-7D08
```

In questo esempio il contenuto del primo riferimento non definito a WS(alla locazione >7D02) e' posizionato a zero per indicare il primo riferimento al quel simbolo.
Il contenuto del prossimo riferimento non definito a WS(alla locazione >7D06) e' posto a >7D02 (indirizzo del precedente riferimento a WS).
Come risultato di un errore di battitura durante un'introduzione di programma un riferimento non definito puo' apparire come se avesse un numero illimitato di puntatori.
Il seguente programma ne da' un esempio.

```
7D00            W1     EQU >1234
7D00 0201              L1 R1,WS
7D02R000
7D04                   AORG >7D00
7D00 0201              L1 R1,W1
7D02 1234
```

In questo segmento WSappare nella Symbol Table come un riferimento di parola non definito con un puntatore a 7D02.
La direttiva susseguente AORG impone alla locazione >7D02 di contenere >1234. Percio' c'e' un numero indefinito di puntatori a WS, fino a quando l'Assembler considera il valore >1234 nella locazione come se fosse il puntatore al riferimento precedente e cosi via.
Per prevenire la possibilita' di una visualizzazione indefinita di riferimenti non definiti la direttiva SYM visualizza un massimo di 32 riferimenti al simbolo non definito.

TECNICHE DI EDITING

Come menzionato precedentemente l'Assembler mantiene alcuni codici sorgente nel buffer a nove pagine permettendo di rivedere linee di programma introdotte precedentemente.
Quando si raggiunge la fine del buffer, il titolo dell'Assembler Line-by-line 'scrolla' all'indietro sullo schermo per avvertire che il buffer e' pieno e si torna all'inizio.
Ogni nuovo codice sorgente da introdurre ricoprira' quello precedente. Percio' e' una buona idea rivedere il codice sorgente a quel punto usando i tasti con le frecce su e giu' per 'scrollare' i dati sullo schermo mentre il vecchio codice sorgente risiede nel buffer.
Se si trova un errore nel codice sorgente occorre fare una nota all'indirizzo della linea sbagliata.
Poi si usa la direttiva AORG per tornare a quell'indirizzo e ribattere la linea correttamente.
Si puo' anche correggere errori di battitura mentre si batte una linea premendo ERASE o usando il metodo che viene discusso qui sotto.
Una etichetta, o in campo etichetta o in campo operando, puo' essere corretta con semplicita' continuando a battere i simboli corretti dopo aver premuto la barra spaziatrice(SPACE BAR) per uscire dal campo. Per esempio, se si batte VF invece di CD come etichetta, allora si batta CD dopo aver premuto la barra per andare al campo successivo.
L'Assembler accetta che gli ultimi due caratteri siano introdotti nel campo come etichetta corretta. Se occorre correggere una etichetta di un solo carattere, si prema 1 per indicare che verra' usata un'etichetta a un solo carattere e poi si batta il carattere alfabetico corretto dopo aver premuto la barra.
Un metodo simile puo' essere usato per correggere una introduzione di un numero esadecimale nel campo operando. Per esempio, se si batte >1234 ma si voleva battere >2234, semplicemente si continua a battere il numero corretto. In altre parole si e' introdotto il numero >12342234 ma l'Assembler considera solo gli ultimi quattro caratteri come introduzione corretta. Le correzioni all'introduzione di numeri decimali e' alquanto piu' difficile poiche' l'Assembler considera le ultime 16 cifre introdotte come introduzione corretta. Se il numero introdotto e' lungo meno di 16 cifre occorre aggiungergli degli zero fino alle 16 cifre.
Per aver un miglio effetto e' probabilmente meglio cancellare la linea con ERASE e poi ribattere il numero in modo corretto.

CONDIZIONI D'ERRORE

Durante l'introduzione e l'assemblaggio di un programma ci sono tre condizioni che determinano sullo schermo un messaggio d'errore.

	CONDIZIONE	MESSAGGIO VISUALIZZATO
1.	Tentativo di ridefinire un'etichetta precedentemente definita.	*ERROR*
2.	Introduzione di un opcode o una direttiva non definita.	*ERROR*

3. Andare fuori limite con un'istruzione di salto. *R-ERROR*

Ogni errore visualizzato e' accompagnato da un suono di errore (bad-response)ed e' stampato sulla stessa linea dell'istruzione in cui c'e' l'errore.
Occorre premere un tasto qualsiasi per cancellare la linea(il contatore di locazione rimane invariato).
Se si tenta di saltare ad una etichetta indefinita e poi si trova che la definizione dell'etichetta provoca che le istruzioni di salto precedenti che gli si riferiscono vadano fuori campo l'indirizzo visualizzato a sinistra del messaggio *R-ERROR* e' l'indirizzo dell'istruzione di salto fuori campo.
Occorre continuare a premere ENTER per vedere altri indirizzi di salto fuori campo che si riferiscono allo stesso livello.
Se si prevede che un'istruzione di salto ad un'etichetta non ancora definita potrebbe essere fuori campo, e' bene far seguire all'istruzione di salto due istruzioni NOP(no operation) per permettere di correggere il programma se capita un errore.
Il seguente programma illustra questa procedura.

```
7D00 XXXX          JNE J1
7D00R16FF
7D02 1000          NOP
7D04 1000          NOP
7D06 C081          MOV R1,R2


7E10 XXXX   J1     EQU $
7D00 *R-ERROR*                  (Premi ENTER per vedere altri
                                rif.a salti fuori campo a J1;poi
                                premi un tasto qualsiasi per can-
                                cellare la condizione di errore)
7E10 XXXX          AORG >7000
7D00 1302          JEQ $+6      Si noti che la logica dell'istru-
                                zione di salto indiretto e' inver-
                                so a quello originale.
7D02 0460          B @J1
7D04 7E10
```

COME MANDARE IN ESECUZIONE IL PROGRAMMA(RUNNING YOUR PROGRAM)

Dopo che il programma e' assemblato il nome e l'indirizzo del programma devono essere aggiunti alla tavola REF/DEF in modo che il modulo MiniMemory possa trovare il programma ed eseguirlo.
Un modo di aggiungere il nome e l'indirizzo e' di usare il sottoprogramma LOAD del TI BASIC (v. 'Additional TI BASIC Subprograms'nel manuale utente).
Un altro modo di aggiungere il nome e l'indirizzo del programma e' di usare le direttive Assembler. Prima si deve determinare se c'e' abbastanza spazio in memoria per aggiungere il nome del programma alla tavola REF/DEF. Dopoche' si e' introdotto l'ultima linea del programma si usi la direttiva AORG per leggere dalla memoria il primo indirizzo libero(First Free Address in the Module = FFAM) e l'ultimo indirizzo libero(Last Free Address in the

Module = LFAM). Gli indirizzi di queste due variabili sono rispettivamente >701C e >701E. Si sottragga il valore >701C dal valore >701E; se la differenza e' maggiore di 7 bytes allora si ha abbastanza spazio per memorizzare il nome del programma.
Dopo essersi assicurati che c'e' abbastanza spazio(8bytes) per aggiungere il nome del programma alla tabella REF/DEF, si sottragga 8 dal vecchio LFAM e si inserisca il nuovo valore LFAM a >701E usando una direttiva DATA. Cio' riserva spazio nella tabella REF/DEF per il nome e l'indirizzo del programma.
Un nome di programma puo' essere di lunghezza da uno a sei caratteri; comunque il nome del programma nella tabella REF/DEF deve essere esattamente lungo sei caratteri.
Se il nome del programma e' minore di sei caratteri si deve modificare il nome con spaziature quando lo introduci nella tabella.
Si usi la direttiva AORG di nuovo per fare un nuovo punto d'ingresso nella tabella e poi introdurre il nome del programma per mezzo della direttiva TEXT. Quando si e' introdotto il nome del programma il contatore di locazioni avanza fino alla prossima parola di legame dove l'indirizzo del programma (due bytes) e' stato memorizzato. Si usi una direttiva DATA di nuovo per introdurre l'indirizzo di partenza del programma.

Esempio : DETERMINAZIONE DELLO SPAZIO DI MEMORIA RIMANENTE

Il seguente esempio assume che sia stato introdotto il 'Sample Program'(programma esemplificativo)dato nel manuale utente della MiniMemory e non si sia usciti dall'Assembler.

SCHERMO	INTRODUZIONE	COMMENTI
7F04 XXXX	AORG >701C	>7F04 rappresenta l'indirizzo del contatore di locazioni corrente e XXXX rappresenta un dato qualsiasi a quell'indirizzo.
701C XXXX		XXXX rappresenta l'indirizzo del vecchio FFAM
701C 7F04	DATA >7F04	L'istruzione DATA fornisce il nuovo FFAM, che e' il primo indirizzo immediatamente seguente il programma.
701E 7FE8		>7FE8 rappresenta l'indirizzo corrente di LFAM. Si sottragga FFAM da questo valore; se il risultato e' 7 bytes o maggiore, c'e' abbastanza spazio per il nome del programma.
701E 7FE0	DATA >7FE0	Si sottragga 8 bytes dal vecchio LFAM, e si memorizzi il risultato come nuovo LFAM per mezzo della direttiva DATA (>7FE0 rappresenta il nuovo LFAM)
7020 XXXX		Il contatore avanza alla prossima locazione e visualizza ogni dato assemblato.

ESEMPIO : AGGIUNGERE NOME E INDIRIZZO DEL PROGRAMMA

Il seguente esempio partendo dalla fine dell'esempio sopra descritto mostra come aggiungere il nome e l'indirizzo del programma DISP$(v. 'Sample Program'nel manuale utente MiniMemory)nella tabella REF/DEF.

SCHERMO	INTRODUZIONE	COMMENTO
7020 XXXX	AORG >7FE0	Mette il nuovo punto d'ingresso in tabella.
7FE0 XXXX		XXXX rappresenta un qualsiasi valore correntemente memorizzato alla locazione >7FE0.
7FE0 4449 7FE2 5350 7FE4 2420	TEXT 'DISP$ '	Si introduce il nome del programma DISP$. (Si noti che si e' aggiunto una spaziatura per modificare il nome a sei caratteri). I caratteri del nome, inclusa la spaziatura, sono memorizzati in sei bytes che iniziano alla locazione >7FE0.
7FE6 XXXX		Il contatore avanza alla prossima locazione dove il punto d'ingresso del programma sara'memorizzato e visualizza ogni valore correntemente memorizzato qui.
7FE6 7E20	DATA DS	L'etichetta DS e' resa uguale all'indirizzo che e' il punto d'ingresso nel programma.

Si e' ora pronti a uscire dall'Assembler ed eseguire il programma.

MEMORIZZARE IL PROGRAMMA SU REGISTRATORE A CASSETTE

Per memorizzare un programma su cassetta prima premere QUIT per avere l'opzione MiniMemory; poi selezionare EASY BUG e usare il comando S. Per far meglio introdurre l'indirizzo di partenza >7000 e l'indirizzo d'arrivo >7FFF per essere sicuri che la tabella REF/DEF e i puntatori nella memoria low siano salvati. Altrimenti e' necessario introdurre il nome del programma nella tabella REF/DEF ogni volta che si vuole caricare il programma. Per altri dettagli vedere la sezione 'EASY BUG Debugger'del manuale utente MiniMemory.

APPENDIX A

CPU Memory Map

>0000	Console ROM	0000
>1FFF		8191
>2000	Memory Expansion — 8K-byte segment (Low Memory)	8192
>3FFF		16383
>4000	Peripheral ROMs (mapped) for device service routines	16384
>5FFF		24575
>6000	Mini Memory — 4K-byte ROM segment	24576
>6FFF		28671
>7000	Mini Memory — 4K-byte RAM segment (Medium Memory)	28672
>7FFF		32767
>8000	Memory Mapped Devices for VDP, GROM, Sound and Speech CPU RAM at >8300 – >83FF	– 32768
>9FFF		– 24577
>A000	Memory Expansion — 24K-byte segment (High Memory)	– 24576
>FFFF		– 1

APPENDIX B

Mini Memory ROM Organization

>6000		Standard ROM/GROM Header
>6010	XML >70	NAMLNK — Name Link Routine
>6012	XML >71	TGOBLD — Tagged Object Loader
>6014	XML >72	CIF — Convert Integer to Floating
>6016		Unused
>6018	BLWP @GPLLNK	Link to GROM Routine
>601C	BLWP @XMLLNK	Link to ROM Routine
>6020	BLWP @KSCAN	Keyboard Scan
>6024	BLWP @VSBW	VDP Single Byte Write
>6028	BLWP @VMBW	VDP Multiple Byte Write
>602C	BLWP @VSBR	VDP Single Byte Read
>6030	BLWP @VMBR	VDP Multiple Byte Read
>6034	BLWP @VWTR	VDP Write to Register
>6038	BLWP @DSRLNK	Link to Device Service Routine
>603C	BLWP @LOADER	Tagged Object Loader
>6040	BLWP @NUMASG	Numeric Assignment Routine
>6044	BLWP @NUMREF	Get Numeric Parameter
>6048	BLWP @STRASG	String Assignment Routine
>604C	BLWP @STRREF	Get String Parameter
>6050	BLWP @ERR	Error Reporting Routine
>6054		Start of ROM program Area
	...	
>6F38		Start of Pre-Defined REF/DEF Table
	...	
>6FFF		End of Pre-Defined REF/DEF Table

APPENDIX C

RAM Organization—TI BASIC Files

MINIMEM	28672	>7000	ID Word >5AA5
(the 4K-byte	28674	>7002	File Information—Status
segment in the			Information
Mini Memory	28675	>7003	Logical Record Length
module)	28676	>7004	End-of-File Pointer
	28678	>7006	Current File Entry Point
	28680	>7008	Start of File Space
		...	
	32767	>7FFF	End of File Space
EXPMEM2	-24576	>A000	ID Word >5AA5
(the 24K-byte	-24574	>A002	File Information—Status
segment in the			Information
Memory Expansion	-24573	>A003	Logical Record Length
unit)	-24572	>A004	End-of-File Pointer
	-24570	>A006	Current File Entry Point
	-24568	>A008	Start of File Space
		...	
	-1	>FFFF	End of File Space

APPENDIX D

Mini Memory RAM Organization—Assembly Language Storage

28672	>7000	ID Word >A55A
28674	>7002	Identifiers for Arguments
	>7012	
28700	>701C	First Free Address in Medium Memory (>7000->7FFF)
28702	>701E	Last Free Address in Medium Memory
28704	>7020	Default Entry Address (>0000)
28706	>7022	First Free Address in High Memory (>A000->FFE0)
28708	>7024	Last Free Address in High Memory
28710	>7026	First Free Address in Low Memory (>2000->3FFF)
28712	>7028	Last Free Address in Low Memory
28714	>702A	Checksum Value
28716	>702C	Pointer to Flag Byte in PAB
28718	>702E	GPL Return Address
28720	>7030	CRU Address of Peripheral
28724	>7034	Device Name Length
28726	>7036	Pointer to Device Name in PAB
28728	>7038	Version Number of DSR
		...
28730	>703A	80-byte Record Buffer for Loader
		...
28810	>708A	NAME Buffer
		...
28818	>7092	UTILWS Utility Workspace
		...
28824	>7098	DSR Link Routine Workspace (Overlaps with UTILWS)
		...
28856	>70B8	USRWSP User Program Workspace Registers
		...
28888	>70D8	Linking Loader Workspace Registers
		...
28920	>70F8	Internal Data Storage
		...
28952	>7118	Free Space
		...
32767	>7FFF	Start of User Defined REF/DEF Tables

Used by Tagged Object Loader

APPENDIX E

VDP RAM Memory Map

>0000		0000
	Pattern Name Table (>0300 bytes)	
>02FF		767
>0300		768
	Sprite Attribute List	
>037F		895
>0380		896
	Pattern Color Table (>0380 – >3FFF) and Free Space	
>03FF		1023
>0400		1024
	Sprite Descriptor Blocks	
>077F		1919
>0780		1920
	Sprite Velocity Table	
>07FF		2047
>0800		2048
	Pattern Generator Area Default Characters >0900 – >0AFF Also used for PAB Area	
>0FFF		4095
>1000		4096
	Free Memory Space Used also for PABs and Buffers	
>137F		4991
>1380		4992
	Used as Buffer for Program File Load	
>34FF		13567
>3500		13568
	Blocks Reserved for Disk DSR	
>3FFF		16383

APPENDIX F

VDP RAM with BASIC Interpreter

>0000	Screen	0
>02FF		767
>0300	Color and Sprite Table	768
>031F		799
>0320	Crunch Buffer	800
>03BD		957
>03BE	BASIC Temporaries and Interpreter Roll-Out Area	958
>03FF		1023
>0400	Character Tables	1024
>05FF		1535
>0600	Value Stack	1536
	String Space	
	Dynamic Symbol Table and PABs	
	Static Symbol Table	
	Line Number Table	
	Crunched Program	
>37FF		16383

Texas Instruments invented the integrated circuit,
the microprocessor, and the microcomputer.
Being first is our tradition.

TEXAS INSTRUMENTS
INCORPORATED