Consiglio Nazionale delle Ricerche

Software per Microprocessori

Libreria MP2650

D. Franchi - P. Mancini

146

CNUCE

A cura di: D. Franchi
P. Mancini

C.N.U.C.E Istituto del C.N.R.

SOFTWARE PER MICROPROCESSORI

Libreria MP2650

D.Franchi

P.Mancini

INDICE

Indirizzo degli autori:
P.Mancini, Scuola Normale Superiore, Pisa.

INTRODUZIONE

Questo manuale descrive le funzioni e la modalita' di utilizzo della libreria MP2650 che e' stata implementata dagli autori adattando il software gentilmente messo a disposizione dalla Philips Elcoma (Milano) ed integrandolo con procedure e programmi di utilita'.

Con questa libreria e' possibile scrivere, assemblare, simulare e trasferire un programma per il microprocessore Signetics 2650 usando tutte le facilities del CMS/370.

Per consentire a tutti gli utenti interessati l'utilizzo della libreria MP2650, gli autori hanno predisposto il nastro magnetico dei text label F080 abilitato alla sola lettura e giacente presso il CNUCE.
Chi disponga di una macchina virtuale puo' trasferire sul proprio disco detto nastro magnetico; altrimenti la libreria MP2650 puo' essere richiamata, con le dovute modalita', dall'OS.
Per le specifiche dettagliate dei linguaggi di programmazione riferirsi ai seguenti manuali:

- Microprocessor 2650 Signetics, Philips Elcoma 1977.
- IBM Virtual Machine Facility/370: Command Language Guide for General Users, Order No. GC20-1804-3, 1975

DIRETTIVE DELLA LIBRERIA MP2650

HELP2650	spiegazioni sulla liberia MP2650
ASM fname	compilazione programmi (dove fname e' il nome del file sorgente)
GO control fname1...fnameN	esecuzione simulata (dove control e' il nome del file di comandi fname1...fnameN sonc i nomi dei moduli oggetto)
RHEX fname1...fnameN	caricamento di moduli oggetto (dove fname1...fnameN sono i nomi dei moduli oggetto)
D hexloc1 hexlocN	dump esadecimale dei moduli caricati (dove hexloc1 hexlocN sono gli indirizzi esadecimali di inizio e fine blocco)

ELEMENTI DEL LINGUAGGIO

Un programma in Assembler 2650 e' costituito da linee di cui l'ultima deve contenere la direttiva END ; dette linee sono fornite al CMS come normali linee di stampa di massimo 80 caratteri in un file di tipo SOURCE. Formalmente le linee sono definite come segue, usando parzialmente la forma normale di Backus integrata, per motivi di semplicita' ed intellegibilita' quando non sia equivoco, con riferimenti in linguaggio naturale e commenti esplicativi posti tra [ ].

```
Linea ::= {Commenti}|{Istruzioni}|{Direttive}
{Commenti} ::= *{COMMENTO}
{Istruzioni} ::= {ISTCODE}|{LABL} {ISTCODE}|{ISTCODE}
                 {COMMENTO}|{LABL} {ISTCODE} {COMMENTO}
{Direttive} ::= {DIRCODE}|{DIRCODE} {COMMENTO} [notare i
                blanks nelle istruzioni e direttive]
{LABL} ::= etichetta alfanumerica di non piu' di 4 caratteri
           contigui e iniziante con una lettera in colonna 1
{COMMENTO} ::= espressione alfanumerica
{DIRCODE} ::= una delle seguenti espressioni

  {LABL} ORG {EXPR}      [set origin, {LABL} opzionale]
  {LABL} EQU {EXPR}      [equivalence symbol]
  {LABL} ACON {EXPR}     [define address, {LABL} opzionale]
  {LABL} DATA {EXPR}     [define data, {LABL} opzionale]
  {LABL} RES {EXPR}      [reserve storage, {LABL} opzionale]
         END {EXPR}      [end assembly]
         EJE             [eject listing page]
         PRT {ONOFF}     [printer control]
         SPC {EXPR}      [space control]
         TITL {EXPR}     [title]
         PCH {ONOFF}     [punch control]

{EXPR} ::= {ESPRESSIONE ARITMETICA}|{EXPR},{ESPRESSIONE
           ARITMETICA}
{ONOFF} ::= ON|OFF
{ESPRESSIONE ARITMETICA} ::= espressione aritmetica composta
                             da {VAR},{COST} e operatori +,-.
{VAR} ::= {LABL}|$
{COST} ::= {NUMERO DECIMALE}|H'{NUMERO ESADECIMALE}'|O'{NUMERO}
           OTTALE}'|B'{NUMERO BINARIO}'|A'{CARATTERE ASCII}'|
           E'{CARATTERE EBCDIC}'
{SIMBOL} ::= {VAR}|>{VAR}|<{VAR}|{ESPRESSIONE ARITMETICA}
```

{ISTCODE} ::= una delle seguenti espressioni [dove op1,...op9, che sono i codici iniziali di ISTCODE, sono di seguito tabulati in corrispondenza verticale, le parentesi ( ) racchiudono gli elementi opzionali].

```
op1,n1 (*) {SIMBOL} (,n2) (,a)
     op2,n1 (*) {SIMBOL}
          op3,n1 {SIMBOL}
               op4 n1
                    op5,n1
                         op6
                              op7 {SIMBOL}
                                   op8 (*) {SIMBOL}
                                        op9 (*) {SIMBOL},3
```

op1	op2	op3	op4	op5	op6	op7	op8	op9	
ADDA	ADDR	ADDI	ADDZ						[addition]
ANDA	ANDR	ANDI	ANDZ						[and]
	BCFA								[branch false absolute]
	BCFR								[branch false relative]
	BCTA								[branch true absolute]
	BCTR								[branch true relative]
	BDRA								[branch decrement absolute]
	BDRR								[branch decrement relative]
	BIRA								[branch increment absolute]
	BIRR								[branch increment relative]
	BRNA								[branch non-zero absolute]
	BRNR								[branch non-zero relative]
	BSFA								[branch subr. false absolute]
	BSFR								[branch subr. false relative]
	BSNA								[branch subr. non-zero abs.]
	BSNR								[branch subr. non-zero rel.]
	BSTA								[branch subr. true absolute]
	BSTR								[branch subr. true relative]
								BSXA	[branch subr. indexed abs.]
								BXA	[branch indexed absolute]

op1	op2	op3	op4	op5	op6	op7	op8	op9	
COMA	COMR	COMI	COMZ						[compare]
						CPSL			[clear program status lower]
						CPSU			[clear program status upper]
				DAR					[decimal adjust register]
EORA	EORR	EORI	EORZ						[exlusive or]
					HALT				[halt]
IORA	IORR	IORI	IORZ						[inclusive or]
					LPSL				[load program status lower]
					LPSU				[load program status upper]
LODA	LODR	LODI	LODZ						[load]
					NOP				[no operation]
						PPSL			[preset program status lower]
						PPSU			[preset program status upper]
				REDC					[read control]
				REDD					[read data]
		REDE							[read extended]
				RRL					[rotate register left]
				RRR					[rotate register rigth]
				RETC					[return]
				RETE					[return enable]
					SPSL				[store program status lower]
					SPSU				[store program status upper]
STRA	STRR		STRZ						[store]
SUBA	SUBR	SUBI	SUBZ						[subtraction]
		TMI							[test mask immediate]
						TPSL			[test program status lower]
						TPSU			[test program status upper]
				WRTC					[write control]
				WRTD					[write data]
		WRTE							[write extended]
							ZBRR		[zero branch relative]
							ZBSR		[zero branch subr. relative]

dove
- n1 = 0,1,2,3 [registro o condizione]
- n2 = 1,2,3 [registro]
- a = +,- [auto-increment o decrement]
- * [indiretto]

COMANDI DEL SIMULATORE

COMANDI	PARAMETRI	
DUMP.	LOC,FWA-LWA(;...;LOC,FWA-LWA)	stampa l'area di memoria FWA-LWA quando eseguita l'istruzione all'indirizzo LOC.
FEND	none	termina l'intera esecuzione
INPUT	VALUE(;...;VALUE)	read dati per l'istruzione simulata di I/O
INSTR.	LOC(;...;LOC)	stampa il contenuto dei registri quando e' eseguita l'istruzione all'indirizzo LOC
LIMIT	NO	specifica il numero totale di istruzioni da eseguire
PATCH	LOC,VALUE((,...;LOC,VALUE)	inizializza ogni cella di indirizzo LOC al valore VALUE
REFER.	LOC(;...;LOC)	stampa il contenuto dei registri quando un'istruzione fa riferimento all'indirizzo LOC
SETP.	LOC(,PSL=VALUE) (,PSU=VALUE)	quando l'istruzione all'indirizzo LOC e' eseguita setta il Program Status Word
SETR.	LOC(,R0=VALUE)...(R6=VALUE)	quando l'istruzione all'indirizzo LOC e' eseguita setta i registri a VALUE
SROM	FWA-LWA	specifica gli estremi della ROM
START	LOC	fa partire il programma simulato dall'indirizzo LOC
STAT	none	stampa le statistiche
STOP.	LOC(;...;LOC)	termina l'esecuzione del programma eseguita l'istruzione all'indirizzo LOC
TEND	none	terminata l'ultima esecuzione si prepara ad eseguire la successiva

dove

LOC = indirizzo esadecimale della istruzione
NO = numero decimale

FWA = primo indirizzo esadecimale dell'area di memoria
LWA = ultimo indirizzo esadecimale dell'area di memoria
VALUE = valore a cui settare una cella di memoria
R0,...R6 = registri generali 0-6
PSL = program status lower
PSU = program status upper

blank = separatore tra comandi e parametri
( ) = racchiudono parametri opzionali
; = separatore tra insiemi di parametri
, = separatore traparametri in un insieme
;...; = indica la possibilita' di scrivere piu' parametri
o insiemi di parametri

Le linee di commento devono essere precedute da **

La memoria di lavoro del simulatore e' lunga 2048 celle di memoria di 4 bytes.

TRASFERIMENTO DATI

Nell'intento di consentire un agevole trasferimento di dati esadecimali dal CMS/370 a piccoli sistemi a microprocessore, tramite terminale, sono state previste le procedure

RHEX modulo oggetto

e la procedura

Dump hexloc1 hexlocN

che generano al terminale una sequenza di dati con il sequente formato

```
hexloc1=                DATO1 DATO2 .. .. .. .. .. .. ..
.......= .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..
hexlocX= .. .. .. DATON
```

dove hexloc1 e' l'indirizzo esadecimale di inizio del blocco di memoria, hexlocX e' l'indirizzo esadecimale di fine blocco di memoria con la prima cifra arrotondata a zero e DATO1 DATO2 ..... DATON sono i valori esadecimali dei bytes compresi tra i due indirizzi specificati. (E' possibile fare il Dump anche di un singolo byte specificandone il solo indirizzo)

Nel seguito e' riportato un esempio di utilizzo delle procedure per il trasferimento di dati.

ESEMPIO APPLICATIVO

Per facilitare la comprensione dei precedenti capitoli viene qui' riportato un esempio di utilizzo di tutte le procedure.

I commenti all'uso delle procedure sono scritti in caratteri minuscoli. Le linee seguenti inizianti con il carattere '.' (che in una seduta reale viene stampato dall'elaboratore come segnale di pronto), sono scritte dall'utente; tutte le altre sono la risposta dell'elaboratore.

Appena attaccata la macchina virtuale viene stampata la scritta

```
PER SPIEGAZIONI USO SIMULATORE FARE 'HELP2650'
```

converra' quindi per prima cosa acquisire informazioni sulle procedure utilizzate scrivendo il comando HELP2650 senza parametri (in ogni istante dell'elaborazione e' possibile chiedere aiuto a questo programma).

```
.HELP2650
```

La risposta del programma di spiegazioni e'

```
DI QUALE PARTE DEL SIMULATORE SIGNETICS VUOI INFORMAZIONI?

* INPUT E/O EDITING DI FILES (SOURCE)
* ASSEMBLAGGIO DI FILES      (ASMFILE)
* SIMULAZIONE DI FILES       (GOFILE)
* DUMP DI FILES OBJECT       (DUMFILE)
* CARATTERE DI TABULAZIONE   (TABCHAR)
* PRINT DI FILES             (PRFILE)
```

Cominciamo a chiedere spiegazioni sulla creazione di nuovi files, e quindi scriviamo:

```
.SOURCE
```

```
PER SCRIVERE UN FILE NUOVO  O CORREGGERNE UNO GIA' ESISTENTE
DARE IL COMANDO

EDIT NOMEFILE SOURCE
```

USARE I COMANDI DI EDITING DEL CMS/370 ED AL TERMINE IL COMANDO 'FILE'. IL FILE 'NOMEFILE SOURCE' VERRA' MEMORIZZATO SUL DISCO DELLA MACCHINA VIRTUALE.

Occorrera' richiedere spiegazioni anche sul carattere di tabulazione. Basta scrivere:

```
.HELP2650 TABCHAR
```

```
IL CARATTERE DI TABULAZIONE DEFAULT E' ':'. PER POSIZIONARSI
A COLONNA 8 DARE  (IN EDIT) IL COMANDO CMS 'TABS  1 8'.  PER
```

CAMBIARE IL CARATTERE DI TABULAZIONE SCRIVI 'TAB X'-DOVE X E' IL NUOVO CARATTERE DI TABULAZIONE; ALTRIMENTI SCRIVI 'OK'.

Se si desidera usare un'altro carattere di tabulazione, per esempio il carattere '>', basta scrivere:

```
.TAB >
```

Fino ad una successiva definizione il carattere di tabulazione resta quello definito (soltanto per l'attuale seduta).

Cominciamo allora a scrivere il nostro file che chiameremo SOMMA. Il tipo deve essere SOURCE

```
.EDIT SOMMA SOURCE

NEW FILE:
EDIT:
.TABS 1 8
```

Con questo comando si tabula a colonna 8 con il carattere '>'

```
.INPUT
INPUT:
```

Possiamo ora scrivere il programma:

```
.*PROGRAMMA PER LA SOMMA ARITMETICA IN COMPLEMENTO A 2
.*DI DATI DI UN BYTE CONTENUTI NELL'AREA DI MEMORIA VET.
.*LENG = LUNGHEZZA DELL'AREA VET
.*VET = INDIRIZZO DELL'AREA DOVE SONO I DATI
.*SOMM = SOMMA FINALE (2 BYTES)
.*BENE = INDICA (SE =0) CHE LA SOMMA E' CORRETTA, (SE =1) OVF
.*LA SOMMA E' FATTA CON IL CARRY E AD OGNI PASSO SI CONTROLLA
.*SE C'E' STATO UN OVERFLOW.
.*
.*
.>ORG H'100' IL PROGRAMMA COMINCIA DALL'INDIRIZZO H'100'
.INIZ>LODI,0 VPSL   CARICA IN REG0 IL BYTE PER SETTARE LA PSL
.>LPSL   CARICA LA PSL CON IL CONTENUTO DI REG0
.>CPSU H'FF'   AZZERA LA PSU
.* INIZIALIZZAZIONE REGISTRI
.>LADA,1 LENG   REG1=PUNTATORE AREA VET
.>LODI,2 0   REG2,REG3 CONTENGONO LA SOMMA PARZIALE. (REG2
.>LODI,3 0   I BITS PIU' SIGNIFICATIVI, REG3 QUELLI MENO)
.>STRA,3 BENE   SI AZZERA LA CELLA BENE
.* CICLO DI SOMMA
.LOP>LODA,0 VET,1,-   LOAD IN REG0 DI (VET+(REG1-1)).
.*                          LA SCANZIONE COMINCIA DAL FONDO
.>ADDZ 3 SI SOMMA AL DATO LA SOMMA PARZIALE( BITS - SIGN.)
.>ADDI,2 0   SI SOMMA IL CARRY AL REG2 (CIOE' 1 SE C'E' STATO
.*          RIPORTO
.>CPSL,1   IN OGNI CASO SI RIAZZERA IL CARRY
.>TPSL H'04'   SI TESTA IL BIT DI OVERFLOW NELLA PSL
.>BCTR,0 SOVF   SALTA A SOVF SE CONDITION CODE=00, CIOE'
```

```
.*          SE C'E' STATO UN OVERFLOW
.>STRZ 3    SI CARICA IN REG3 LA PARTE MENO SIGNIFICATIVA
.*          DELLA SOMMA PARZIALE
.>BRNR,1 LOOP    SALTA A LOOP FINCHE' REG1 DIVERSO DA 0
.MEM>STRA,2 SOMM    SI MEMORIZZA REG2 E REG3 NELL'AREA DI
.>STRA,3 SOMM+1    MEMORIA INDIRIZZATA DA SOMM
.>HALT
.SOVF>LODI,0 H'FF'    SI SETTA BENE A H'FF' PER INDICARE L'OVF
.>STRA,0 BENE
.>LODI,2 0    SI AZZERA LA SOMMA FINALE
.>LODI,3 0
.>BCTR,3 MEM    SALTO INCONDIZIONATO A MEM
.*  AREA  R A M
.>ORG H'200'    INIZIO AREA DATI DALLA CELLA H'200'
.BENE>RES 1    SI RISERVA 1 BYTE ALLA CELLA BENE
.SOMM>RES 2    SI RISERVANO DUE BYTES PER LA SOMMA FINALE
.LENG>DATA B'00011101'    LUNGHEZZA AREA VET (29 DATI)
.VET>DATA H'1,2,3,4,5,6,7,8,9,A,B,C,D,E,F'
.>DATA D'16,17,18,9,,1,22'
.>DATA C'31,32,33,4,5,36,37'
.VPSL>EQU B'00001010' (COM=1,WC=1)SET INIZ PROG STATUS LOWER
.>END INIZ
.(pigiare tasto RETURN)
EDIT:
```

In questo ambiente e' possibile usare i subcommands EDIT/370 per poter ottenere stampe parziali, fare correzioni ecc... Al termine dare il comando:

```
.FILE
B;
```

Il file SOMMA SOURCE e' memorizzato sul disco della macchina virtuale.

Per assemblare il programma chiediamo aiuto al programma HELP2650:

```
.HELP2650 ASMFILE

PER ASSEMBLARE UN FILE DARE IL COMANDO

ASM NOMEFILE

L'ASSEMBLATORE PRODUCE DUE FILE SUL DISCO DELLA MACCHINA
VIRTUALE
NOMEFILE LISTING =LISTING TIPO SIGNETICS CON DIAGNOSTICA DI
                  ERRORI;
NOMEFILE OBJECT  =MODULO   OGGETTO    FORMATO   SIGNETICS
                  UTILIZZABILE SIA PER IL SIMULATORE CHE PER
                  L'ESECUZIONE SUL 2650.
Inviamo allora il comando:

.ASM SOMMA
EXECUTION BEGINS...
16 0105 00 00 00      0  LADA,1 LENG   REG1=PUNTATORE AREA VET
26 0115 75 00         RA       CPSL,1   IN OGNI CASO SI RIAZZ
32 011C 59 00         U  BRNR,1 LOOP   SALTA A LOOP FINCHE' RE
```

```
 TOTAL ASSEMBLER ERRORS =     4
R;
```

Ci sono 4 errori che sono deducibili dalle linee che sono state stampate. La lettera prima del codice mnemonico indica il tipo di errore. Se tale diagnostica non e' sufficiente per individuare gli errori commessi, potremo stampare l'intero file SOMMA LISTING per avere una visione totale del programma. Chiediamo aiuto al programma HELP2650 per avere informazioni sul print di files.

```
.HELP2650 PRFILE
```

PER AVERE LA STAMPA DI UN FILE DI QUALSIASI TIPO (SOURCE,LISTING,OBJECT, RESULT) DARE IL COMANDO CMS

```
PRINT NOMEFILE TIPO N1 N2
```

PER STAMPARE SULLA STAMPANTE DEL 370; OPPURE IL COMANDO CMS

```
TYPE NOMEFILE TIPO N1 N2
```

PER STAMPARE SULLA TELESCRIVENTE. N1 E N2 SONO RISPETTIVAMENTE IL NUMERO DELLA PRIMA E DELL'ULTIMA RIGA DI CUI SI VUOLE LA STAMPA. SE OMESSI E' STAMPATO L'INTERO FILE.

Allora invieremo il comando:

```
.TYPE SOMMA LISTING
```

ed otterremo la stampa dell'intero file con questo formato:

```
1 PIP ASSEMBLER VERSION 4 LEVEL 1                                    PAGE    1

 LINE ADDR B1 B2 B3 B4 ERR SOURCE

    1                          *PROGRAMMA PER LA SOMMA ARITMETICA IN COMPLEM
    2                          *DI DATI DI UN BYTE CONTENUTI NELL'AREA DI ME
    3                          *LENG = LUNGHEZZA DELL'AREA VET
    4                          *VET = INDIRIZZO DELL'AREA DOVE SONO I DATI
    5                          *SOMM = SOMMA FINELE (2 BYTES)
    6                          *BENE = INDICA (SE =0) CHE LA SOMMA E' CORRETT
    7                          *LA SOMMA E' FATTA CON IL CARRY E AD OGNI PASS
    8                          *SE C'E' STATO UN OVERFLOW.
    9                          *
   10                          *
   11                                 ORG H'100' IL PROGRAMMA COMINCIA DALL'I
   12 0100 04 0A              INIZ    LODI,0 VPSL   CARICA IN REGO IL BYTE PE
   13 0102 93                         LPSL   CARICA LA PSL CON IL CONTENUTO D
   14 0103 74 FF                      CPSU H'FF'   AZZERA LA PSU
   15                         * INIZIALIZZAZIONE REGISTRI
   16 0105 00 00 00      0            LADA,1 LENG   REG1=PUNTATORE AREA VET
   17 0108 06 00                      LODI,2 0    REG2,REG3 CONTENGONO LA SOMM
   18 010A 07 00                      LODI,3 0    PIU' SIGNIFICATIVI, REG3 QUE
   19 010C CF 02 00                   STRA,3 BENE   SI AZZERA LA CELLA BENE
   20                         * CICLO DI SOMMA
   21 010F 0D 42 04           LOP    LODA,0 VET,1,-   LOAD IN REGO DI (VET+(R
   22                         *                          LA SCANZIONE COMINCI
```

```
   23 0112 83                          ADDZ 3   SI SOMMA AL DATO LA SOMMA PARZ
   24 0113 86 00                       ADDI,2 0   SI SOMMA IL CARRY AL REG2 (C
   25                            *        RIPORTO
   26 0115 75 00              RA       CPSL,1   IN OGNI CASO SI RIAZZERA IL CA
   27 0117 B5 04                       TPSL H'04'   SI TESTA IL BIT DI OVERFLO
   28 0119 18 0A                       BCTR,0 SOVF   SALTA A SOVF SE CONDITION
   29                            *         SE C'E' STATO UN OVERFLOW
   30 011B C3                          STRZ 3   SI CARICA IN REG3 LA PARTE MEN
   31                            *         DELLA SOMMA PARZIALE
   32 011C 59 00               U       BRNR,1 LOOP   SALTA A LOOP FINCHE' REG1
   33 011E CE 02 01            MEM     STRA,2 SOMM   SI MEMORIZZA REG2 E REG3
   34 0121 CF 02 02                    STRA,3 SOMM+1   MEMORIA INDIRIZZATA DA
   35 0124 40                          HALT
   36 0125 04 FF               SOVF    LODI,0 H'FF'   SI SETTA BENE A H'FF' PE
   37 0127 CC 02 00                    STRA,0 BENE
   38 012A 06 00                       LODI,2 0   SI AZZERA LA SOMMA FINALE
   39 012C 07 00                       LODI,3 0
   40 012E 1B 6E                       BCTR,3 MEM   SALTO INCONDIZIONATO A MEM
   41                            *' AREA  R A M
   42                                  ORG H'200'   INIZIO AREA DATI DALLA CEL
   43 0200                     BENE    RES 1   SI RISERVA 1 BYTE ALLA CELLA BE
   44 0201                     SOMM    RES 2   SI RISERVANO DUE BYTES PER LA S
   45 0203 1D                  LENG    DATA B'00011101'   LUNGHEZZA AREA VET (
   46 0204 01 02 03 04         VET     DATA H'1,2,3,4,5,6,7,8,9,A,B,C,D,E,F'
            05 06 07 08
            09 0A 0B 0C
            0D 0E 0F
   47 0213 10 11 12 09                 DATA D'16,17,18,9,,1,22'
            00 01 16
   48 021A 19 1A 1B 04                 DATA O'31,32,33,4,5,36,37'
            05 1E 1F
1 PIP ASSEMBLER VERSION 4 LEVEL 1                                    PAGE    2

 LINE ADDR B1 B2 B3 B4 ERR SOURCE

   49 000A                     VPSL    EQU B'00001010' (CCM=1,WC=1) SET INIZIA
   50                                  END INIZ


 TOTAL ASSEMBLER ERRORS =    4
```

Gli errori si deducono dalla colonna ERR del LISTING; se c'e' una lettera significa che nella linea c'e' un errore ed il tipo della lettera indica il tipo di errore commesso. Alla linea 16 c'e' un errore del tipo O (Op-code error): infatti abbiamo scritto LADA al posto di LODA. Alla linea 26 ci sono due errori: di tipo R (Register field error) e di tipo A (Argument error). Entrambi sono stati causati dal carattere ',' dopo il codice operativo (deve esserci uno spazio). Alla linea 32 e' segnalato l'errore del tipo U (Undefined symbol); l'istruzione e' un branch all'istruzione con label LOOP e non esiste tale istruzione. Infatti l'errore e' stato commesso alla linea 21 dove la label e' stata scritta LOP al posto di LOOP.

Con questa diagnostica di errore andiamo a correggere il

file SOMMA SOURCE.

```
.EDIT SOMMA SOURCE
EDIT:
.NEXT 16
        LADA,1 LENG    REG1=PUNTATORE AREA VET
.C /A/O/
        LODA,1 LENG    REG1=PUNTATORE AREA VET
.L /LOP
LOP    LODA,0 VET,1,-   LOAD IN REG0 DI (VET+(REG1-1)).
.C /LOP/LOOP/
LOOP    LODA,0 VET,1,-   LOAD IN REG0 DI (VET+(REG1-1)).
.L /CPSL,
        CPSL,1    IN OGNI CASO SI RIAZZERA IL CARRY
.C /,1/ 1
        CPSL 1    IN OGNI CASO SI RIAZZERA IL CARRY
.FILE
R;
```

Il file SOMMA SOURCE e' corretto e possiamo riassemblarlo:

```
.ASM SOMMA
EXECUTION BEGINS...
 TOTAL ASSEMBLER ERRORS =     0
R;
```

Finalmente il file e' corretto e possiamo richiedere la stampa del file SOMMA SOURCE e dei files SOMMA LISTING e SOMMA OBJECT prodotti dall'assemblatore.

```
.TYPE SOMMA SOURCE

*PROGRAMMA PER LA SOMMA ARITMETICA IN COMPLEMENTO A 2
*DI DATI DI UN BYTE CONTENUTI NELL'AREA DI MEMORIA VET.
*LENG = LUNGHEZZA DELL'AREA VET
*VET = INDIRIZZO DELL'AREA DOVE SONO I DATI
*SOMM = SOMMA FINALE (2 BYTES)
*BENE = INDICA (SE =0) CHE LA SOMMA E' CORRETTA, (SE =1) OVF
*LA SOMMA E' FATTA CON IL CARRY E AD OGNI PASSO SI CONTROLLA
*SE C'E' STATO UN OVERFLOW.
*
*
        ORG H'100' IL PROGRAMMA COMINCIA DALL'INDIRIZZO H'100'
INIZ    LODI,0 VPSL    CARICA IN REG0 IL BYTE PER SETTARE LA PSL
        LPSL    CARICA LA PSL CON IL CONTENUTO DI REG0
        CPSU H'FF'    AZZERA LA PSU
* INIZIALIZZAZIONE REGISTRI
        LODA,1 LENG    REG1=PUNTATORE AREA VET
        LODI,2 0    REG2,REG3 CONTENGONO LA SOMMA PARZIALE. (REG2
        LODI,3 0    I BITS PIU' SIGNIFICATIVI, REG3 QUELLI MENO)
        STRA,3 BENE    SI AZZERA LA CELLA BENE
* CICLO DI SOMMA
LOOP    LODA,0 VET,1,-    LOAD IN REG0 DI (VET+(REG1-1)).
*                          LA SCANZIONE COMINCIA DAL FONDO
        ADDZ 3    SI SOMMA AL DATO LA SOMMA PARZIALE( BITS - SIGN.)
        ADDI,2 0    SI SOMMA IL CARRY AL REG2 (CIOE' 1 SE C'E' STATO
*          RIPORTO
        CPSL 1    IN OGNI CASO SI RIAZZERA IL CARRY
```

```
        TPSL H'04'    SI TESTA IL BIT DI OVERFLOW NELLA PSL
        BCTR,0 SOVF   SALTA A SOVF SE CONDITION CODE=00, CIOE'
*           SE C'E' STATO UN OVERFLOW
        STRZ 3    SI CARICA IN REG3 LA PARTE MENO SIGNIFICATIVA
*           DELLA SOMMA PARZIALE
        BRNR,1 LOOP   SALTA A LOOP FINCHE' REG1 DIVERSO DA 0
MEM     STRA,2 SOMM   SI MEMORIZZA REG2 E REG3 NELL'AREA DI
        STRA,3 SOMM+1   MEMORIA INDIRIZZATA DA SOMM
        HALT
SOVF    LODI,0 H'FF'    SI SETTA BENE A H'FF' PER INDICARE I'OVF
        STRA,0 BENE
        LODI,2 0    SI AZZERA LA SOMMA FINALE
        LODI,3 0
        BCTR,3 MEM    SALTO INCONDIZIONATO A MEM
*  AREA  R A M
        ORG H'200'    INIZIO AREA DATI DALLA CELLA H'200'
BENE    RES 1    SI RISERVA 1 BYTE ALLA CELLA BENE
SOMM    RES 2    SI RISERVANO DUE BYTES PER LA SOMMA FINALE
LENG    DATA B'00011101'    LUNGHEZZA AREA VET (29 DATI)
VET     DATA H'1,2,3,4,5,6,7,8,9,A,B,C,D,E,F'
        DATA D'16,17,18,0,,1,22'
        DATA O'31,32,33,4,5,36,37'
VPSL    EQU B'00001010' (COM=1,WC=1) SET INIZ PROG STATUS LOWER
        END INIZ
```

```
.TYPE SOMMA LISTING

1 PIP ASSEMBLER VERSION 4 LEVEL 1                                PAGE    1

 LINE ADDR B1 B2 B3 B4 ERR SOURCE

    1                       *PROGRAMMA PER LA SOMMA ARITMETICA IN COMPLEM
    2                       *DI DATI DI UN BYTE CONTENUTI NELL'AREA DI ME
    3                       *LENG = LUNGHEZZA DELL'AREA VET
    4                       *VET = INDIRIZZO DELL'AREA DOVE SONO I DATI
    5                       *SOMM = SOMMA FINELE (2 BYTES)
    6                       *BENE = INDICA (SE =0) CHE LA SOMMA E' CORRETT
    7                       *LA SOMMA E' FATTA CON IL CARRY E AD OGNI PASS
    8                       *SE C'E' STATO UN OVERFLOW.
    9                       *
   10                       *
   11                              ORG H'100' IL PROGRAMMA COMINCIA DALL'I
   12 0100 04 0A            INIZ   LODI,0 VPSL   CARICA IN REG0 IL BYTE PE
   13 0102 93                      LPSL   CARICA LA PSL CON IL CONTENUTO D
   14 0103 74 FF                   CPSU H'FF'   AZZERA LA PSU
   15                       * INIZIALIZZAZIONE REGISTRI
   16 0105 0D 02 03                LODA,1 LENG   REG1=PUNTATORE AREA VET
   17 0108 06 00                   LODI,2 0   REG2,REG3 CONTENGONO LA SOMM
   18 010A 07 00                   LODI,3 0   PIU' SIGNIFICATIVI, REG3 QUE
   19 010C CF 02 00                STRA,3 BENE   SI AZZERA LA CELLA BENE
   20                       * CICLO DI SOMMA
   21 010F 0D 42 04         LOOP   LODA,0 VET,1,-   LOAD IN REG0 DI (VET+(
   22                       *                          LA SCANZIONE COMINCI
   23 0112 83                      ADDZ 3   SI SOMMA AL DATO LA SOMMA PARZ
   24 0113 86 00                   ADDI,2 0   SI SOMMA IL CARRY AL REG2 (C
```

```
   25                                *          RIPORTO
   26 0115 75 01                              CPSL 1   IN OGNI CASO SI RIAZZERA IL CA
   27 0117 B5 04                              TPSL H'04'   SI TESTA IL BIT DI OVERFLO
   28 0119 18 0A                              BCTR,0 SOVF   SALTA A SOVF SE CONDITION
   29                                *          SE C'E' STATO UN OVERFLOW
   30 011B C3                                 STRZ 3   SI CARICA IN REG3 LA PARTE MEN
   31                                *          DELLA SOMMA PARZIALE
   32 011C 59 71                              BRNR,1 LOOP   SALTA A LOOP FINCHE' REG1
   33 011E CE 02 01                  MEM      STRA,2 SOMM   SI MEMORIZZA REG2 E REG3
   34 0121 CF 02 02                           STRA,3 SOMM+1   MEMORIA INDIRIZZATA DA
   35 0124 40                                 HALT
   36 0125 04 FF                     SOVF     LODI,0 H'FF'   SI SETTA BENE A H'FF' PE
   37 0127 CC 02 00                           STRA,0 BENE
   38 012A 06 00                              LODI,2 0   SI AZZERA LA SOMMA FINALE
   39 012C 07 00                              LODI,3 0
   40 012E 1B 6E                              BCTR,3 MEM   SALTO INCONDIZIONATO A MEM
   41                                *  AREA  R A M
   42                                         ORG H'200'   INIZIO AREA DATI DALLA CEL
   43 0200                           BENE     RES 1   SI RISERVA 1 BYTE ALLA CELLA BE
   44 0201                           SOMM     RES 2   SI RISERVANO DUE BYTES PER LA S
   45 0203 1D                        LENG     DATA B'00011101'   LUNGHEZZA AREA VET (
   46 0204 01 02 03 04               VET      DATA H'1,2,3,4,5,6,7,8,9,A,B,C,D,E,F'
            05 06 07 08
            09 0A 0B 0C
            0D 0E 0F
   47 0213 10 11 12 09                        DATA D'16,17,18,9,,1,22'
            00 01 16
   48 021A 19 1A 1B 04                        DATA O'31,32,33,4,5,36,37'
            05 1E 1F
1 PIP ASSEMBLER VERSION 4 LEVEL 1                                        PAGE    2

 LINE ADDR B1 B2 B3 B4 ERR SOURCE

   49 000A                           VPSL     EQU B'00001010' (COM=1,WC=1) SET INIZIA
   50                                         END INIZ



 TOTAL ASSEMBLER ERRORS =     0
```

```
.TYPE SOMMA OBJECT

:01001E34040A9374FF0D020306000700CF02000D4204838600750 1B504180AC35971E6
:011E1254CE0201CF0202400 4FFCC020006000700 1B6E18
:02031E201D0102030405060 708090A0B0C0D0E0F10111209000116191A1B04051E1FA4
:0100000 8
```

Prima di procedere alla simulazione chiediamo informazioni
al programma HELP2650:

```
.HELP2650 GOFILE
```

PER SIMULARE L'ESECUZIONE DI UN FILE DARE IL COMANDO

GO CONTROL NOMEFILE1....NOMEFILEN

DOVE CONTROL E' IL NOME DEL FILE DI TIPO COMMANDS IN CUI DEVONO ESSERE SCRITTI I COMANDI NECESSARI PER LA SIMULAZIONE, NOMEFILE1...NOMEFILEN SONO I NOMI DEI FILES MODULO OGGETTO (DI TIPO OBJECT) DI CUI SI DESIDERA LA SIMULAZIONE. I RISULTATI DELLA SIMULAZIONE SARANNO DISPONIBILI NEL FILE "CONTROL RESULT" PRODOTTO DAI SIMULATORE.

Entriamo in ambiente EDIT per scrivere il nostro file di comandi che chiameremo COMSOM COMMANDS. I commenti riportati nel file descrivono le fasi della simulazione. Si faccia riferimento al file SOMMA LISTING per gli indirizzi.

```
.EDIT COMSOM COMMANDS
NEW FILE:
EDIT:
.INPUT
INPUT:
.**  PRIMA SIMULAZIONE
.**
.LIMIT 10000
.** ESECUZIONE MASSIMO 10000 ISTRUZIONI
.START 100
.** IL PROGRAMMA PARTE DALL'INDIRIZZO 100
.INSTR. 11C;125
.**STAMPA CONTENUTO REGISTRI OGNI VOLTA CHE ESEGUE L'ISTRUZIONE
.**ALL'INDIRIZZO 11C E 125. SI PUO' COSI' CONTROLLARE LA SOMMA
.** PARZIALE DATA DAI REGISTRI 2 E 3 BANCA 0.
.SROM 100-150
.** L'AREA DI MEMORIA ROM E' DEFINITA DALL'INDIRIZZO 100-150
.DUMP. 124,200-202
.**ESEGUITA L'ISTRUZIONE DI HALT (INDIRIZZO 124) STAMPA L'AREA
.** DI MEMORIA 200-202, CIOE' IL VALORE DI BENE (SE=0 SOMMA GIUSTA),
.** ED IL VALORE DELLA SOMMA FINALE (201-202)
.TEND
.**FINE PRIMA SIMULAZIONE
.**
.**  SECONDA SIMULAZIONE
.**
.START 100
.PATCH 203,0A
.** SI METTE NELLA CELLA LENG (INDIRIZZO 203) IL VALORE 10 DECIMALE
.** CIOE' SI SOMMANO SOLO I PRIMI 10 NUMERI DELL'AREA VET
.STAT
.** RICHIESTA DI STATISTICHE
.DUMP. 124,200-202
.**AL TERMINE DELLA SOMMA STAMPA IL VALORE FINALE DELLA SOMMA
.** ED IL BYTE DI CONTROLLO (BENE)
.TEND
.**
.**  TERZA SIMULAZIONE
.**
.LIMIT 10000
.START 100
```

```
.SETR. 10A R2=7F
.** SI CARICA IN FASE DI INIZIALIZZAZIONE (INDIRIZZO 10A)
.** IL REG2 (BITS PIU' SIGNIF. SOMMA PARZIALE) CON H'7F'
.PATCH 203,1D
.** PROVA PER GENERARE OVERFLOW
.INSTR. 125
.** QUANDO SI VERIFICA OVERFLOW SI STAMPANO I REGISTRI PER
.**CONTROLLARE QUAL'E' IL VALORE DELLA SOMMMA PARZIALE(REG2,REG3)
.DUMP. 100,200-203
.DUMP. 124,200-202
.STAT
.FEND
.** FINE SIMULAZIONE
.(piqiare tasto RETURN)
EDIT:
.FILE
R;
```

Scritto il file di comandi possiamo procedere alla simulazione

```
.GO COMSOM SOMMA
EXECUTION BEGINS...
R;
```

Terminata la simulazione possiamo vedere quali sono stati i risultati facendoci stampare il file "COMSOM RESULT"

```
.TYPE COMSOM RESULT

 2650 SM 1100  "PIPSIM" VERSION 1.2

 **  PRIMA SIMULAZIONE
 **
 LIMIT 10000
 ** ESECUZIONE MASSIMO 10000 ISTRUZIONI
 START 100
 ** IL PROGRAMMA PARTE DALL'INDIRIZZO 100
 INSTR. 11C;125
 **STAMPA CONTENUTO REGISTRI OGNI VOLTA CHE ESEGUE L'ISTRUZIONE
 **ALL'INDIRIZZO 11C E 125. SI PUO' COSI' CONTROLLARE LA SOMMA
 ** PARZIALE DATA DAI REGISTRI 2 E 3 BANCA 0.
 SROM 100-150
 ** L'AREA DI MEMORIA ROM E' DEFINITA DALL'INDIRIZZO 100-150
 DUMP. 124,200-202
 **ESEGUITA L'ISTRUZIONE DI HALT (INDIRIZZO 124) STAMPA L'AREA
 ** DI MEMORIA 200-202, CIOE' IL VALORE DI BENE (SE=0 SOMMA GIUSTA),
 ** ED IL VALORE DELLA SOMMA FINALE (201-202)
 TEND


 INSTR.
 IAR      INST              EADDR  (EADDR) PSU  PSL    R0  R1  R2  R3  R4  R5  R6
 011C BRNR,1 0071           010F    000D    00   4A    1F  1C  00  1F  00  00  00
 011C BRNR,1 0071           010F    000D    00   4A    3D  1B  00  3D  00  00  00
 011C BRNR,1 0071           010F    000D    00   4A    42  1A  00  42  00  00  00
 011C BRNR,1 0071           010F    000D    00   4A    46  19  00  46  00  00  00
 011C BRNR,1 0071           010F    000D    00   4A    61  18  00  61  00  00  00
```

```
011C BRNR,1 0071          010F   000D   00  4A   7B 17 00 7B 00 00 00
011C BRNR,1 0071          010F   000D   00  8A   94 16 00 94 00 00 00
011C BRNR,1 0071          010F   000D   00  8A   AA 15 00 AA 00 00 00
011C BRNR,1 0071          010F   000D   00  8A   AB 14 00 AB 00 00 00
011C BRNR,1 0071          010F   000D   00  8A   AB 13 00 AB 00 00 00
011C BRNR,1 0071          010F   000D   00  8A   B4 12 00 B4 00 00 00
011C BRNR,1 0071          010F   000D   00  8A   C6 11 00 C6 00 00 00
011C BRNR,1 0071          010F   000D   00  8A   D7 10 00 D7 00 00 00
011C BRNR,1 0071          010F   000D   00  8A   E7 0F 00 E7 00 00 00
011C BRNR,1 0071          010F   000D   00  8A   F6 0E 00 F6 00 00 00
011C BRNR,1 0071          010F   000D   00  4A   04 0D 01 04 00 00 00
011C BRNR,1 0071          010F   000D   00  4A   11 0C 01 11 00 00 00
011C BRNR,1 0071          010F   000D   00  4A   1D 0B 01 1D 00 00 00
011C BRNR,1 0071          010F   000D   00  4A   28 0A 01 28 00 00 00
011C BRNR,1 0071          010F   000D   00  4A   32 09 01 32 00 00 00
011C BRNR,1 0071          010F   000D   00  4A   3B 08 01 3B 00 00 00
011C BRNR,1 0071          010F   000D   00  4A   43 07 01 43 00 00 00
011C BRNR,1 0071          010F   000D   00  4A   4A 06 01 4A 00 00 00
011C BRNR,1 0071          010F   000D   00  4A   50 05 01 50 00 00 00
011C BRNR,1 0071          010F   000D   00  4A   55 04 01 55 00 00 00
011C BRNR,1 0071          010F   000D   00  4A   59 03 01 59 00 00 00
011C BRNR,1 0071          010F   000D   00  4A   5C 02 01 5C 00 00 00
011C BRNR,1 0071          010F   000D   00  4A   5E 01 01 5E 00 00 00
011C BRNR,1 0071          010F   000D   00  4A   5F 00 01 5F 00 00 00
COMMAND DUMP, IAR=0124
0200    00 01 5F 1D 01 02 03 04 05 06 07 08 09 0A 0B 0C
HALT.
IAR     INST              EADDR (EADDR) PSU PSL   R0 R1 R2 R3 R4 R5 R6
0124 HALT                               00  4A   5F 00 01 5F 00 00 00

NO. OF MACHINE CYCLES EXECUTED =        667
NO. OF INSTRUCTIONS   EXECUTED =        242

1
2650 SM 1100  "PIPSIM" VERSION 1.2

**FINE PRIMA SIMULAZIONE
**
**  SECONDA SIMULAZIONE
**
START 100
PATCH 203,0A
** SI METTE NELLA CELLA LENG (INDIRIZZO 203) IL VALORE 10 DECIMALE
** CIOE' SI SOMMANO SOLO I PRIMI 10 NUMERI DELL'AREA VET
STAT
** RICHIESTA DI STATISTICHE
DUMP. 124,200-202
**AL TERMINE DELLA SOMMA STAMPA IL VALORE FINALE DELLA SOMMA
** ED IL BYTE DI CONTROLLO (BENE)
TEND


COMMAND DUMP, IAR=0124
0200    00 00 37 0A 01 02 03 04 05 06 07 08 09 0A 0B 0C
HALT.
IAR     INST              EADDR (EADDR) PSU PSL   R0 R1 R2 R3 R4 R5 R6
0124 HALT                               00  4A   37 00 00 37 00 00 00
```

NO. OF MACHINE CYCLES EXECUTED = 249
NO. OF INSTRUCTIONS EXECUTED = 90

LODZ	0
LODI	3
LODR	0
LODA	11
ADDZ	10
ADDI	10
ADDR	0
ADDA	0
SUBZ	0
SUBI	0
SUBR	0
SUBA	0
ANDZ	0
ANDI	0
ANDR	0
ANDA	0
IORZ	0
IORI	0
IORR	0
IORA	0
EORZ	0
EORI	0
EORR	0
EORA	0
COMZ	0
COMI	0
COMR	0
COMA	0
STRZ	10
STRR	0

1

STRA	3
LAR	0
RRR	0
RRL	0
BCTR	10
BCTA	0
BCFR	0
BCFA	0
BRNR	10
BRNA	0
BIRR	0
BIRA	0
BDRR	0
BDRA	0
ZBRR	0
BXA	0
BSTR	0
BSTA	0
BSFR	0
BSFA	0
BSNR	0
BSNA	0
ZBSR	0
BSXA	0

```
     RETC          0
     RETE          0
     WRTD          0
     REDD          0
     WRTC          0
     REDC          0
     WRTE          0
     REDE          0
     HALT          1
     NOP           0
     TMI           0
     LPSU          0
     LPSL          1
     SPSU          0
     SPSL          0
     CPSU          1
     CPSL         10
     PPSU          0
     PPSL          0
     TPSU          0
     TPSL         10
1
2650 SM 1100  "PIPSIM" VERSION 1.2

**
**  TERZA SIMULAZIONE
**
LIMIT 10000
START 100
SETR. 10A R2=7F
** SI CARICA IN FASE DI INIZIALIZZAZIONE (INDIRIZZO 10A)
** IL REG2 (BITS PIU' SIGNIF. SOMMA PARZIALE) CON H'7F'
PATCH 203,1D
** PROVA PER GENERARE OVERFLOW
INSTR. 125
** QUANDO SI VERIFICA OVERFLOW SI STAMPANO I REGISTRI PER
**CONTROLLARE QUAL'E' IL VALORE DELLA SOMMMA PARZIALE(REG2,REG3)
DUMP. 100,200-203
DUMP. 124,200-202
STAT
FEND


COMMAND DUMP, IAR=0100
0200    40 40 40 1D 01 02 03 04 05 06 07 08 09 0A 0B 0C
INSTR.
IAR     INST            EADDR (EADDR)PSU PSL   R0 R1 R2 R3 R4 R5 R6
0125 LODI,0 FF          0126   00FF   00  2E   04 0D 80 F6 00 00 00
COMMAND DUMP, IAR=0124
0200    FF 00 00 1D 01 02 03 04 05 06 07 08 09 0A 0B 0C
HALT.
IAR     INST            EADDR (EADDR)PSU PSL   R0 R1 R2 R3 R4 R5 R6
0124 HALT                              00  2E   FF 0D 00 00 00 00 00

NO. OF MACHINE CYCLES EXECUTED =        389
NO. OF INSTRUCTIONS   EXECUTED =        141

     LODZ          0
```

IODI	6
IODR	0
IODA	17
ADDZ	16
ADDI	16
ADDR	0
ADDA	0
SUBZ	0
SUBI	0
SUBR	0
SUBA	0
ANDZ	0
ANDI	0
ANDR	0
ANDA	0
IORZ	0
IORI	0
IORR	0
IORA	0
EORZ	0
EORI	0
EORR	0
EORA	0
CCMZ	0

1

CCMI	0
CCMR	0
CCMA	0
STRZ	15
STRR	0
STRA	4
LAR	0
RRR	0
RRL	0
BCTR	17
BCTA	0
BCFR	0
BCFA	0
BRNR	15
BRNA	0
BIRR	0
BIRA	0
BDRR	0
BDRA	0
ZBRR	0
BXA	0
BSTR	0
BSTA	0
BSFR	0
BSFA	0
BSNR	0
BSNA	0
ZBSR	0
BSXA	0
RETC	0
RETE	0
WRTD	0
REDD	0

```
WRTC          0
REDC          0
WRTE          0
REDE          0
HALT          1
NOP           0
TMI           0
LPSU          0
LPSL          1
SPSU          0
SPSL          0
CPSU          1
CPSL         16
PPSU          0
PPSL          0
TPSU          0
TPSL         16
```

Una volta che il programma SOMMA e' stato assemblato,simulato e verificato, possiamo procedere al suo trasferimento ad un sistema a microprocessore. Per questo chiamiamo il programma HELP2650:

```
.HELP2650 DUMFILE
```

PER FARE IL DUMP DI UN'AREA DI MEMORIA CON FORMATO ANALOGO A QUELLO OTTENIBILE CON IL SIGNETICS TWIN 2650, OCCORRE PER PRIMA DEFINIRE I FILES OBJECTS DI CUI SI DESIDERA IL DUMP CON IL COMANDO

```
RHEX NOMEFILE1....NOMEFILEN
```

DOVE NOMEFILE1...NOMEFILEN SONO I NOMI DEI MODULI OGGETTO (DI TIPO OBJECT) E SUCCESSIVAMENTE DARE IL COMANDO DI DUMP

```
DUMP HEXLOC1 HEXLOC2
```

DOVE HEXLOC1 ED HEXLOC2 SONO GLI INDIRIZZI ESADECIMALI DI INIZIO E FINE.

Se ci interessa trasferire l'area di memoria corrispondente all'area di ROM del programma SOMMA dovremo inviare il comando:

```
.RHEX SOMMA
EXECUTION BEGINS...
```

e definiamo il file SOMMA OBJECT di cui si desidera fare il trasferimento, ed il comando:

```
.D 100 12F
```

i cui parametri corrispondono esattamente agli indirizzi esadecimali di inizio e fine area di memoria in cui e' contenuto il programma. La risposta del calcolatore allora sara':

```
 0100=04 0A 93 74 FF 0D 02 03 06 00 07 00 CF 02 00 0D
 0110=42 04 83 86 00 75 01 B5 04 18 0A C3 59 71 CE 02
 0120=01 CF 02 02 40 04 FF CC 02 00 06 00 07 00 1B 6E


 >


R;
```

E' possibile fare il Dump di un numero qualsiasi di bytes compresi nell'area di memoria definita dai moduli oggetto caricati (e' possibile ottenere il Dump anche di un solo byte), ma il risultato del Dump puo' essere imprevedibile e generare errori se richiediamo il Dump di aree di memoria non definite esplicitamente. Comandi di Dump consecutivi devono essere sempre preceduti dalla definizione del modulo oggetto con il comando RHEX. Diamo alcuni esempi di possibili Dump:

```
.RHEX SOMMA
EXECUTION BEGINS...
.D 10A 115
 010A=                                  07 00 CF 02 00 0D
 0110=42 04 83 86 00 75


 >


R;



.RHEX SOMMA
EXECUTION BEGINS...
.D 12E
 012E=                                                 1B


 >


R;



.RHEX SOMMA
EXECUTION BEGINS...
.D 35A 371
**DUMP ERROR
R;


.RHEX SOMMA
EXECUTION BEGINS...
D 120 13F
 0120=01 CF 02 02 40 04 FF CC 02 00 06 00 07 00 1B 6E
```

0130= (possibili dati non significativi)

(possibile messaggio di errore dal CMS/370)