Anno Accademico 2013/2014

Calcolatori Elettronici

Parte X: l'Assemblatore as88

Prof. Riccardo Torlone

Università Roma Tre

L'assemblatore as88

Disponibile presso:

- CD-ROM allegato al libro di testo del corso
- ftp://ftp.cs.vu.nl/pub/evert/
- Sito Web del corso

Il tool comprende:

- Programma assemblatore (as88)
  - Utilizzo Generale: as88 Nomeprogetto(.s)
- Emulatore-Interprete dell'architettura 8088 (s88)
  - Utilizzo Generale: s88 Nomeprogetto
- Programma tracer per il debugging (t88)
  - Utilizzo Generale: t88 Nomeprogetto(.$)

Direttive del compilatore

- Ogni programma assembly e strutturato in 3 sezioni:
  1. sezione di TESTO (direttiva: .SECT .TEXT): contiene le istruzioni del programma
  2. sezione DATI (direttiva: .SECT .DATA): alloca spazio nel segmento DATI per i dati (inizializzati)
  3. sezione BSS (direttiva: .SECT .BSS): alloca spazio nel segmento DATI per i dati (non inizializzati)

- E' possibile definire etichette di due tipi:
  - globali: identificatori alfanumerici seguiti dal simbolo ":" (possono occupare una intera riga)
  - locali: utilizzabili solo nel segmento TESTO, costituite da una sola cifra seguita dal simbolo ":".

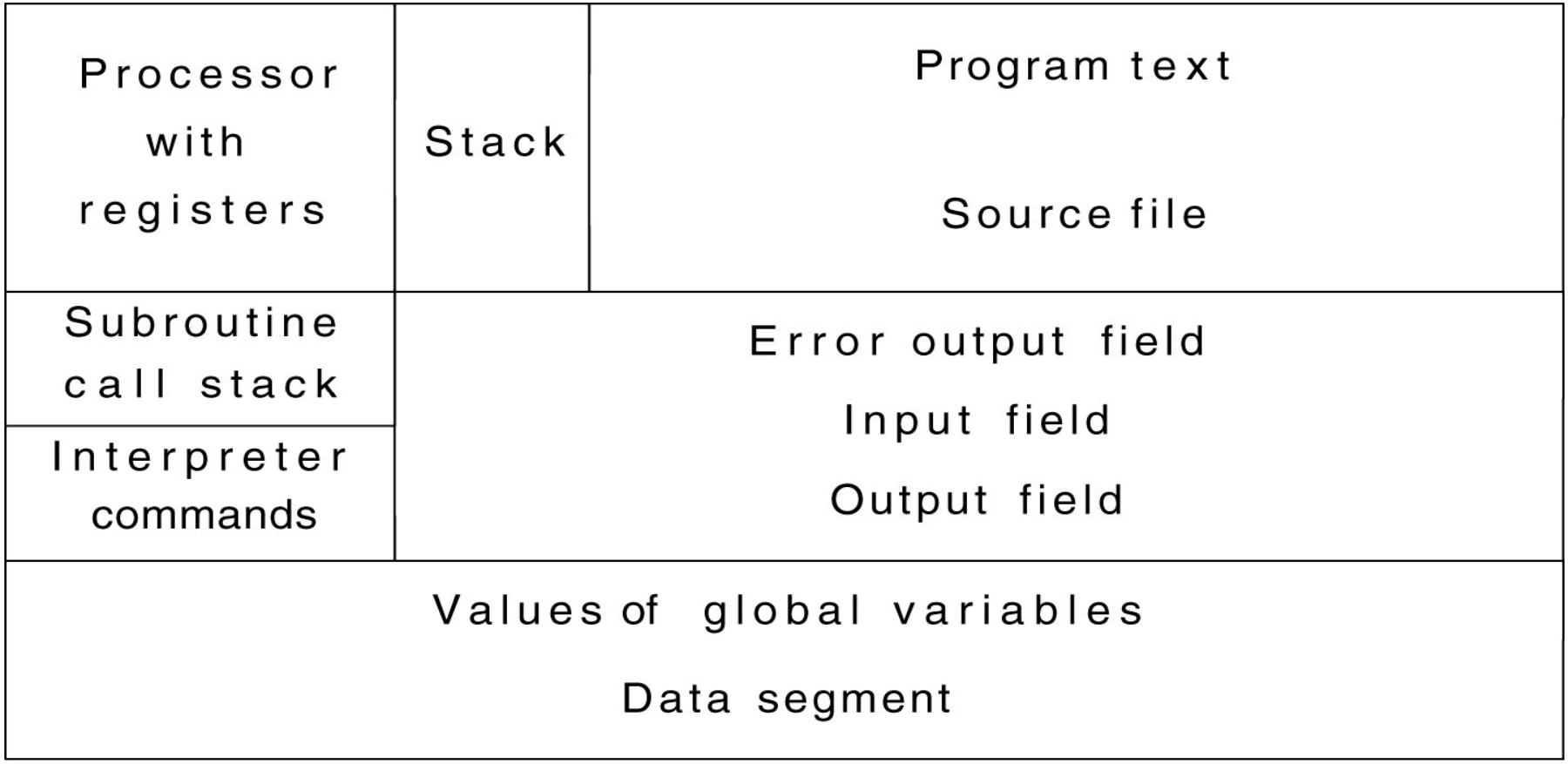
Vincoli sulle etichette

- Le etichette globali DEVONO essere univoche
  - Es: .SECT .DATA hw: .ASCII "Hello"
- Le etichette locali possono occorrere più volte
  - Es. JMP 1f
  - Salto verso la prossima etichetta denominata "1"
- E' possibile attribuire nomi simbolici alle costanti mediante la sintassi: identificatore=espressione
  - Es. BLOCKSIZE=1024
- I valori numerici possono essere:
  - ottali (cominciano per zero),
  - decimali,
  - esadecimali (cominciano per 0x)
- I commenti iniziano con il carattere "!"

Direttive del compilatore

Instruction	Description
.SECT .TEXT	Assemble the following lines in the TEXT section
.SECT .DATA	Assemble the following lines in the DATA section
.SECT .BSS	Assemble the following lines in the BSS section
.BYTE	Assemble the arguments as a sequence of bytes
.WORD	Assemble the arguments as a sequence of words
.LONG	Assemble the arguments as a sequence of longs
.ASCII "str"	Store str as ascii an string without a trailing zero byte
.ASCIZ "str"	Store str as ascii an string with a trailing zero byte
.SPACE n	Advance the location counter n positions
.ALIGN n	Advance the location counter up to an n-byte boundary
.EXTERN	Identifier is an external name

The Tracer (debugger)



Il tracer consente di effettuare l'esecuzione step-by-step del programma e di monitorare lo stato di registri/memoria

Uso dei registri con il tracer

```
start:                          ! 3
      MOV    AX,258             ! 4
      ADDB   AH,AL              ! 5
      MOV    CX,(times)         ! 6
      MOV    BX,muldat          ! 7
      MOV    AX,(BX)            ! 8
llp:  MUL    2(BX)              ! 9
      LOOP   llp                ! 10
.SECT .DATA                     ! 11
times:  .WORD  10               ! 12
muldat :.WORD  625,2            ! 13
```

(a)

```
CS: 00   DS=SS=ES002
AH:03 AL:02  AX:    770
BH:00 BL:02  BX:      2
CH:00 CL:0a  CX:     10
DH:00 DL:00  DX:      0
SP: 7fe0 SF  O D S Z C
BP: 0000 CC  - > p - -
SI: 0000   IP:0009:PC
DI: 0000   start + 4
```

(b)

```
CS: 00   DS=SS=ES002
AH:38 AL:80  AX: 14464
BH:00 BL:02  BX:      2
CH:00 CL:04  CX:      4
DH:00 DL:01  DX:      1
SP: 7fe0 SF  O D S Z C
BP: 0000 CC  v > p - c
SI: 0000   IP:0011:PC
DI: 0000   start + 7
```

(c)

(a) Parte del programma

(b) I registri dopo l'esecuzione di 7 righe

(c) I registri dopo l'esecuzione di 6 iterazioni del ciclo

The ACK-Based Assembler, as88

Escape symbol	Description
\n	New line (line feed)
\t	Tab
\\	Backslash
\b	Back space
\f	Form feed
\r	Carriage return
\"	Double quote

Valori di escape consentiti nell'*as88.*

Comandi del tracer (1)

Address	Command	Example	Description
			Execute one instruction
#	, ! , X	24	Execute # instructions
/T+#	g , ! ,	/start+5g	Run until line # after label T
/T+#	b	/start+5b	Put breakpoint on line # after label T
/T+#	c	/start+5c	Remove breakpoint on line # after label T
#	g	108g	Execute program until line #
	g	g	Execute program until current line again
	b	b	Put breakpoint on current line
	c	c	Remove breakpoint on current line

E' possibile interagire con il tracer:

- in modalità batch (fornendo in input un file con i comandi del tracer)
- in modalità interattiva (inserendo comandi da tastiera seguiti dal tasto INVIO)

Comandi del tracer (2)

Address	Command	Example	Description
	n	n	Execute program until next line
	r	r	Execute until breakpoint or end
	=	=	Run program until same subroutine level
	-	-	Run until subroutine level minus 1
	+	+	Run until subroutine level plus 1
/D+#		/buf+6	Display data segment on label+#
/D+#	d , !	/buf+6d	Display data segment on label+#
	R , CTRL L	R	Refresh windows
	q	q	Stop tracing, back to command shell

Chiamate di sistema

- Le chiamate di sistema consentono di utilizzare le procedure fornite dal sistema operativo.

- Le routine di sistema possono essere attivate con la sequenza di chiamata standard:
  - Si impilano gli argomenti sullo stack
  - Si impila il numero di chiamata
  - Si esegue l'istruzione SYS

- I risultati sono restituiti nel registro AX o nella combinazione di registri AX:DX (se il risultato e di tipo long)
- Gli argomenti sullo stack devono essere rimossi dalla funzione chiamante

Chiamate di sistema in as88 (1)

L'interprete 8088 supporta 12 chiamate di sistema.

- **_OPEN**: Apre il file *name* in lettura-scrittura

  Identificativo chiamata: **5**

  Argomenti: *name, 0=lettura/1=scrittura/2=lettura-scrittura;

  Valore Ritorno: un descrittore di file (fd)

- **_CREAT**: Crea un nuovo file di nome *name*

  Identificativo chiamata: **8**

  Argomenti: *name, *mode = permessi UNIX;

  Valore Ritorno: un descrittore di file (fd)

- **_READ**: Legge *n* byte da un file con descrittore *fd* trasferendoli nel buffer *buf*

  Identificativo chiamata: **3**

  Argomenti: fd, buf, n;

  Valore Ritorno: numero di byte letti correttamente

Chiamate di sistema in as88 (2)

- **_WRITE**: Scrive *n* byte sul file con descrittore *fd* prelevandoli dal buffer *buf*

  Identificativo chiamata: **4**

  Argomenti: fd, buf, n;

  Valore Ritorno: numero di byte scritti correttamente
- **_CLOSE**: Chiude un file precedentemente aperto

  Identificativo chiamata: **6**

  Argomenti: fd (descrittore di file)

  Valore Ritorno: 0 se l'operazione ha successo
- **_LSEEK**: Sposta il puntatore del file con descrittore *fd* di *offset* bytes

  Identificativo chiamata: **19**

  Argomenti: fd, offset, 0/1/2;

  Valore Ritorno: nuova posizione all'interno del file
- **_EXIT**: Interrompe un processo

  Identificativo chiamata: **1;**

  Argomenti: 0=successo/1=errore;

Chiamate di sistema in as88 (3)

- **_GETCHAR**: Legge un carattere dallo standard input

  Identificativo chiamata: **117**

  Valore ritorno: il carattere letto e posto in AL
- **_PUTCHAR**: Scrive un carattere sullo standard output

  Identificativo chiamata: **122**

  Argomenti: carattere da scrivere
- **_PRINTF**: Stampa una stringa formattata sullo standard output

  Identificativo chiamata: **127**

  Argomenti: stringa di formato, argomenti
- **_SSCANF**: Legge gli argomenti dal buffer *buf*

  Identificativo chiamata: **125**

  Argomenti: buf, stringa di formato, argomenti
- **_SPRINTF**: Stampa una stringa formattata sul buffer *buf*

  Identificativo chiamata: **121**

  Argomenti: buf, stringa di formato, argomenti

Primo esempio

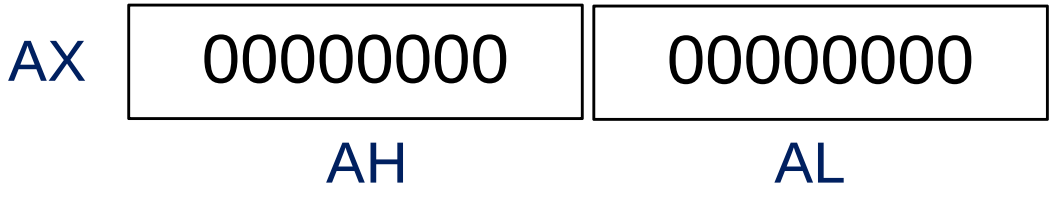
```
!calcolo di (a+3)*b
        _EXIT = 1
.SECT .TEXT
start:
        MOV   AX,(a)
        ADD   AX,3
        MUL     (b)
        PUSH  0
        PUSH  _EXIT
        SYS
.SECT .DATA
a:      .WORD   5
b:      .WORD   3
```

Esempio con ciclo e vari metodi di indirizzamento

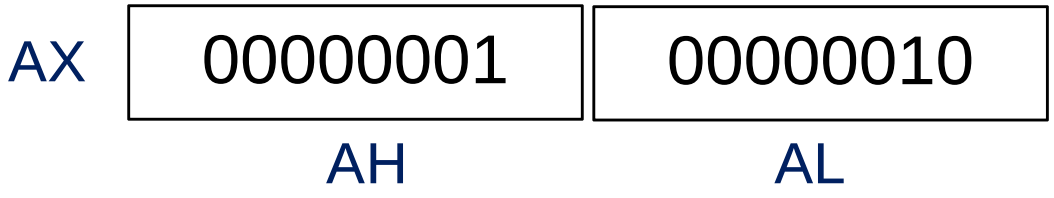
```
! Calcola moltiplicazioni ripetute per 2
        _EXIT  = 1
.SECT .TEXT
start:
        MOV AX,258 !test sui registri (inutile ai fini del programma)
        ADDB AH,AL !test sui registri (inutile ai fini del programma)
        MOV CX, (times)
        MOV BX, muldat
        MOV AX, (BX)
1:      MUL 2(BX)
        LOOP 1b
        PUSH  0
        PUSH  _EXIT
        SYS
.SECT .DATA
        times:   .WORD 5
        muldat: .WORD 1,2
```

Registri a 8 e a 16 bit

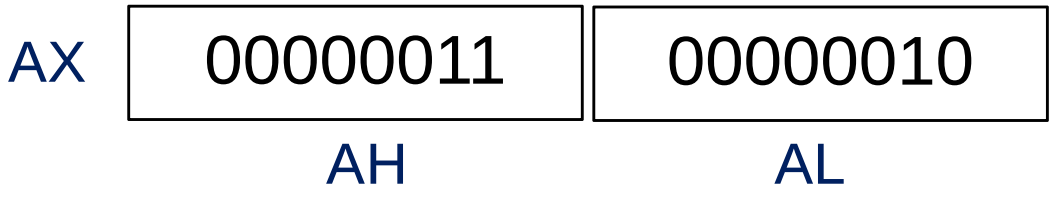
Tutti i registri possono essere visti come coppie di registri di 8 bit accessibili autonomamente (esempio: AX=AH:AL)



MOVE AX,258



ADD AH,AL



AX=770

Esempio: Hello world

```
! Simple "hello world" program
          _EXIT     = 1                 !  1
          _WRITE    = 4                 !  2
          _STDOUT   = 1                 !  3
.SECT .TEXT                             !  4
start:                                  !  5
          MOV       CX,de-hw            !  6
          PUSH      CX                  !  7
          PUSH      hw                  !  8
          PUSH      _STDOUT             !  9
          PUSH      _WRITE              ! 10
          SYS                           ! 11
          ADD       SP,8                ! 12
          SUB       CX,AX               ! 13
          PUSH      CX                  ! 14
          PUSH      _EXIT               ! 15
          SYS                           ! 16
.SECT .DATA                             ! 17
hw:                                     ! 18
.ASCII    "Hello World\n"               ! 19
de:       .BYTE     0                   ! 20
```

Hello world con il tracer

```
    _EXIT = 1                    ! 1
    _WRITE  =4                   ! 2
    _STDOUT  =1                  ! 3
.SECT .TEXT                      ! 4
start:                           ! 5
    MOV    CX,de-hw              ! 6
    PUSH   CX                    ! 7
    PUSH   hw                    ! 8
    PUSH   _STDOUT               ! 9
    PUSH   _WRITE                !10
    SYS                          !11
    ADD    SP, 8                 !12
    SUB    CX,AX                 !13
    PUSH   CX                    !14
    PUSH   _EXIT                 !15
    SYS                          !16
.SECT .DATA                      !17
hw:                              !18
.ASCII "Hello World\n"           ! 19
de: .BYTE 0                      !20
```

(a)

```
CS: 00   DS=SS=ES: 002
AH:00 AL:0c   AX:     12
BH:00 BL:00   BX:      0
CH:00 CL:0c   CX:     12
DH:00 DL:00   DX:      0
SP: 7fd8  SF  O D S Z C  =>0004
BP: 0000  CC  - > p - -    0001
SI: 0000   IP:000c:PC      0000
DI: 0000   start + 7       000c

        MOV   CX,de-hw     !  6
        PUSH  CX           !  7
        PUSH  HW           !  8
        PUSH  _STDOUT      !  9
        PUSH  _WRITE       ! 10
        SYS                ! 11
=>      ADD   SP,8         ! 12
        SUB   CX,AX        ! 13
        PUSH  CX           ! 14

E
I

hw
■

> Hello World\n

hw + 0 = 0000: 48 65 6c 6c 6f 20 57 6f  Hello World  25928
```

(b)

(a) HlloWrld.s.

(b) La finestra corrispondente del tracer

Esempio: prodotto scalare di due vettori (1) [vp.s]

```
_EXIT      = 1                  ! 1   define the value of _EXIT
_PRINTF    = 127                ! 2   define the value of _PRINTF
.SECT .TEXT                     ! 3   start the TEXT segment
inpstart:                       ! 4   define label inpstart
        MOV  BP,SP              ! 5   save SP in BP
        PUSH vec2               ! 6   push address of vec2
        PUSH vec1               ! 7   push address of vec1
        MOV CX,vec2-vec1        ! 8   CX = number of bytes in vector
        SHR  CX,1               ! 9   CX = number of words in vector
        PUSH CX                 ! 10   push word count
        CALL vecmul             ! 11   call vecmul
        MOV  (inprod),AX        ! 12   move AX
        PUSH AX                 ! 13   push result to be printed
        PUSH pfmt               ! 14   psuh address of format string
        PUSH _PRINTF            ! 15   push function code for PRINTF
        SYS                     ! 16   call the PRINTF function
        ADD  SP,12              ! 17   clean up the stack
        PUSH 0                  ! 18   push status code
        PUSH _EXIT              ! 19   push function code for EXIT
        SYS                     ! 20   call the EXIT function
```

Esempio: prodotto scalare di due vettori (2)

```
vecmul:
        PUSH BP
        MOV  BP,SP
        MOV  CX,4(BP)
        MOV  SI,6(BP)
        MOV  DI,8(BP)
        PUSH 0

1:      LODS
        MUL  (DI)
        ADD  -2(BP),AX
        ADD  DI,2
        LOOP 1b
        POP  AX
        POP  BP
        RET
```

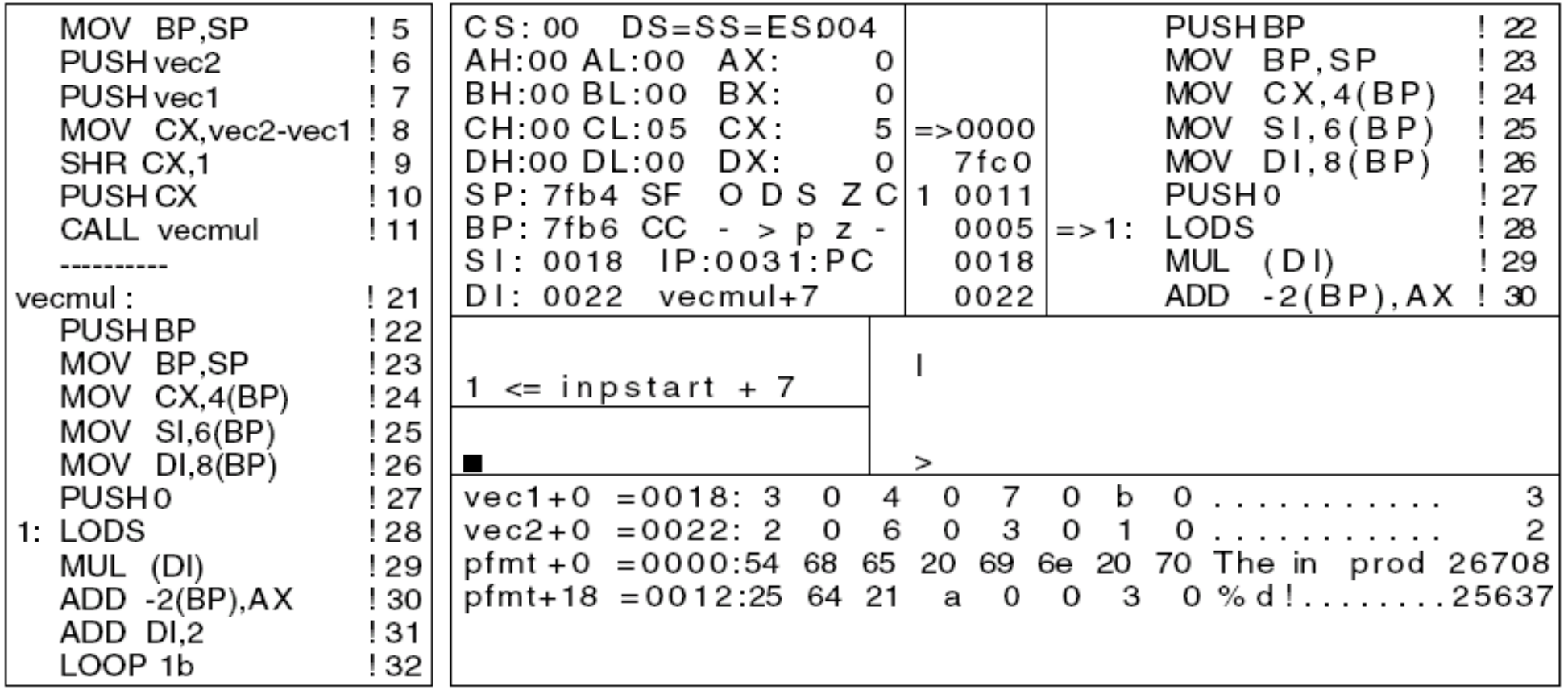
```
! 21   start of vecmul(count, vec1, vec2)
! 22   save BP on stack
! 23   copy SP into BP to access arguments
! 24   put count in CX to control loop
! 25   SI = vec1
! 26   DI = vec2
! 27   push 0 onto stack

! 28   move (SI) to AX
! 29   multiply AX by (DI)
! 30   Add AX to accumulated value in memory
! 31   Increment DI to point to next element
! 32   if CX > 0, go back to label 1b
! 33   Pop top of stack to AX
! 34   Restore BP
! 35   Return from subroutine
```

Esempio: prodotto scalare di due vettori (3)

```
.SECT .DATA                          ! 36  start DATA segment
pfmt: .ASCIZ "Inner product is: %d\n" ! 37   define string
.ALIGN 2                             ! 38   force address even
vec1:.WORD 3,4,7,11,3                ! 39   vector 1
vec2:.WORD 2,6,3,1,0                 ! 40   vector 2
.SECT .BSS                           ! 41   start BSS segment
inprod:      .SPACE 2                ! 42   allocate space for inprod
```

Prodotto scalare di due vettori con il tracer



Esempio: stampa di un array di interi [c1.s]

```
#include "../syscalnr.h"          ! 1

.SECT .TEXT                       ! 2
vecpstrt:                         ! 3
   MOV  BP,SP                     ! 4
   PUSH vec1                      ! 5
   MOV  CX,frmatstr-vec1          ! 6
   SHR  CX                        ! 7
   PUSH CX                        ! 8
   CALL vecprint                  ! 9
   MOV  SP,BP                     ! 10
   PUSH 0                         ! 11
   PUSH _EXIT                     ! 12
   SYS                            ! 13

.SECT .DATA                       ! 14
vec1:   .WORD 3,4,7,11,3          ! 15
frmatstr: .ASCIZ "%s"             ! 16

frmatkop:                         ! 17
.ASCIZ "The array contains "      ! 18
frmatint: .ASCIZ " %d"            ! 19
```

```
.SECT .TEXT                       ! 20
vecprint:                         ! 21
   PUSH BP                        ! 22
   MOV  BP,SP                     ! 23
   MOV  CX,4(BP)                  ! 24
   MOV  BX,6(BP)                  ! 25
   MOV  SI,0                      ! 26
   PUSH frmatkop                  ! 27
   PUSH frmatstr                  ! 28
   PUSH _PRINTF                   ! 29
   SYS                            ! 30
   MOV  -4(BP),frmatint           ! 31
1: MOV  DI,(BX)(SI)               ! 32
   MOV  -2(BP),DI                 ! 33
   SYS                            ! 34
   INC  SI                        ! 35
   LOOP 1b                        ! 36
   PUSH '\n'                      ! 37
   PUSH _PUTCHAR                  ! 38
   SYS                            ! 39
   MOV  SP,BP                     ! 40
   RET                            ! 41
```

E' presente un errore: con il tracer possiamo correggerlo

Esempio: copia di stringhe [cp1.s, cp2.s]

```
.SECT .TEXT
stcstart:                          ! 1
   PUSH mesg1                      ! 2
   PUSH mesg2                      ! 3
   CALL strngcpy                   ! 4
   ADD  SP,4                       ! 5
   PUSH 0                          ! 6
   PUSH 1                          ! 7
   SYS                             ! 8
strngcpy:                          ! 9
   PUSH CX                         ! 10
   PUSH SI                         ! 11
   PUSH DI                         ! 12
   PUSH BP                         ! 13
   MOV  BP,SP                      ! 14
   MOV  AX,0                       ! 15
   MOV  DI,10(BP)                  ! 16
   MOV  CX,–1                      ! 17
   REPNZ SCASB                     ! 18
   NEG  CX                         ! 19
   DEC CX                          ! 20
   MOV  SI,10(BP)                  ! 21
   MOV  DI,12(BP)                  ! 22
   PUSH DI                         ! 23
   REP  MOVSB                      ! 24
   CALL stringpr                   ! 25
   MOV  SP,BP                      ! 26
   POP  BP                         ! 27
   POP  DI                         ! 28
   POP  SI                         ! 29
   POP  CX                         ! 30
   RET                             ! 31
.SECT .DATA                        ! 32
mesg1: .ASCIZ "Have a look\n"      ! 33
mesg2: .ASCIZ "qrst\n"             ! 34
.SECT .BSS
```

Esempio: stampa di una stringa in ordine inverso [r.s]

```
#include "../syscalnr.h"          !  1

start: MOV DI,str                 !  2
    PUSH AX                       !  3
    MOV BP,SP                     !  4
    PUSH _PUTCHAR                 !  5
    MOVB AL,'\n'                  !  6
    MOV CX,-1                     !  7
    REPNZ SCASB                   !  8
    NEG CX                        !  9
    STD                           ! 10
    DEC CX                        ! 11
    SUB DI,2                      ! 12
MOV SI,DI                         ! 13
    1: LODSB                      ! 14
    MOV (BP),AX                   ! 15
    SYS                           ! 16
    LOOP  1b                      ! 17
    MOVB  (BP),'\n'               ! 18
    SYS                           ! 19
    PUSH 0                        ! 20
    PUSH _EXIT                    ! 21
SYS                               ! 22
.SECT .DATA                       ! 23
str: .ASCIZ "reverse\n"           ! 24
```