

LAVORIAMO CON IL DEBUG

Problema :

Risolvere la seguente funzione utilizzando i registri ad 8 bit:

$$8 - (1 + 4)$$

A 100	Assembla a partire dall'indirizzo 100
MOV AL, 08	AL<- 08
MOV BL, 04	BL<- 04
MOV BH, 01	BH<- 01
ADD BH, BL	BH<- BH +BL
SUB AL, BH	AL <- AL -BH
INT 3	

Per salvarlo nella directory pippo col nome somma.com bisogna eseguire la seguente procedura:
Dopo aver scritto il programma come da figura e aver digitato INT 3 premere invio; spunterà una linea. A questo punto scrivere:

- n c:\pippo\somma.com (invio)

- R BX (invio)

-0

R CX (invio)

- B (invio) Il valore inserito B dipende dalla lunghezza del nostro programma nel nostro caso dalla figura si vede che da 0100 l'istruzione dopo INT 3 è arrivata alla cella 010B da cui facendo la differenza tra (010B – 0100) = B

-W questa istruzione significa scrivi il programma

Dopo invio si avrà la risposta : sono stati scritti B byte, cioè il nostro programma è stato salvato

```

C:\pippo>debug c:\pippo\somma.com
-u
0D01:0100 B008      MOU     AL,08
0D01:0102 B304      MOU     BL,04
0D01:0104 B701      MOU     BH,01
0D01:0106 00DF      ADD     BH,BL
0D01:0108 28F8      SUB     AL,BH
0D01:010A CC          INT     3
0D01:010B 64          DB      64
0D01:010C 61          DB      61
0D01:010D 7461      JZ      0170
0D01:010F 2028      AND     [BX+SI],CH
0D01:0111 253129    AND     AX,2931
0D01:0114 3A20      CMP     AH,[BX+SI]
0D01:0116 154C27    ADC     AX,274C
0D01:0119 6F          DB      6F
0D01:011A 7261      JB      017D
0D01:011C 20636F    AND     [BP+DI+6F],AH
0D01:011F 7272      JB      0193
  
```

Per richiamare il file appena salvato bisogna eseguire la seguente procedura.

Appunti di sistemi

Dal prompt di c:> scrivere

```
c:> debug c:\pippo\somma.com      (invio)
```

```
-u                                (cioè u = assembla il programma appena salvato)
```

```
-r                                (cioè r = fai vedere i registri )
```

```
-t                                ( t = step to step cioè passo passo)
```

```
(continuare a dare invio e poi scrivere t fin quando si arriva
```

```
all'istruzione INT 3).
```

Vediamo cosa scrive il computer:

```
-r
```

```
AX=0000 BX=0000 CX=000B DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000  
DS=0D01 ES=0D01 SS=0D01 CS=0D01 IP=0100 NV UP EI PL NZ NA PE NC  
0D01:0100 B008      MOV   AL,08
```

```
-t
```

```
AX=0008 BX=0000 CX=000B DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000  
DS=0D01 ES=0D01 SS=0D01 CS=0D01 IP=0102 NV UP EI PL NZ NA PE NC  
0D01:0102 B304      MOV   BL,04
```

```
-t
```

```
AX=0008 BX=0004 CX=000B DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000  
DS=0D01 ES=0D01 SS=0D01 CS=0D01 IP=0104 NV UP EI PL NZ NA PE NC  
0D01:0104 B701      MOV   BH,01
```

```
-t
```

```
AX=0008 BX=0104 CX=000B DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000  
DS=0D01 ES=0D01 SS=0D01 CS=0D01 IP=0106 NV UP EI PL NZ NA PE NC  
0D01:0106 00DF      ADD   BH,BL
```

```
-t
```

```
AX=0008 BX=0504 CX=000B DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000  
DS=0D01 ES=0D01 SS=0D01 CS=0D01 IP=0108 NV UP EI PL NZ NA PE NC  
0D01:0108 28F8      SUB   AL,BH
```

```
-t
```

```
AX=0003 BX=0504 CX=000B DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000  
DS=0D01 ES=0D01 SS=0D01 CS=0D01 IP=010A NV UP EI PL NZ NA PE NC  
0D01:010A CC       INT   3
```

```
-
```

Esempio di moltiplicazione con registri a 8 bit:

scrivere un programma che risolva la seguente funzione:

$$5 * (8 - 3)$$

```

C:\> Prompt dei comandi - debug c:\pippo\prodot.com
0CAB:0119 A10534      MOV     AX,[3405]
0CAB:011C 3400             XOR     AL,00
0CAB:011E 9A0C10B505      CALL   05B5:100C
-q

C:\>debug c:\pippo\prodot.com
-u 100
0D01:0100 B008      MOV     AL,08
0D01:0102 B303      MOV     BL,03
0D01:0104 28D8      SUB     AL,BL
0D01:0106 B305      MOV     BL,05
0D01:0108 F6E3      MUL     BL
0D01:010A CC        INT     3
0D01:010B 64        DB     64
0D01:010C 61        DB     61
0D01:010D 7461      JZ     0170
0D01:010F 2028      AND     [BX+SI],CH
0D01:0111 253129    AND     AX,2931
0D01:0114 3A20      CMP     AH,[BX+SI]
0D01:0116 154C27    ADC     AX,274C
0D01:0119 6F        DB     6F
0D01:011A 7261      JB     017D
0D01:011C 20636F    AND     [BP+DI+6F],AH
0D01:011F 7272      JB     0193
    
```

Esempio moltiplicazione con registri a 8 bit.

Sintassi dell'istruzione:

MUL sorg

Questa istruzione moltiplica sorgente per il contenuto di AL e mette il risultato in AX.
Cioè:

$$AX \leftarrow AL * sorg$$

ESEMPIO:

Scrivere un programma che risolva la seguente funzione:

$$10 * (F - A)$$

ricordiamo che

10 H= 16 dec

F H =15 dec

A H= 10 dec

Da cui il risultato di $10 * (F-A) = 50$ H cioè è uguale a 80 dec

Risolviamolo in debug:

```

C:\>debug c:\pippo\prodot2.com
0D01:011A 7261      JB      017D
0D01:011C 20636F      AND     [BP+DI+6F],AH
0D01:011F 7272      JB      0193
-q
C:\>debug c:\pippo\prodot2.com
-u 100
0D01:0100 B00F      MOV     AL,0F
0D01:0102 B30A      MOV     BL,0A
0D01:0104 28D8      SUB     AL,BL
0D01:0106 B310      MOV     BL,10
0D01:0108 F6E3      MUL     BL
0D01:010A CC        INT     3
0D01:010B 64        DB      64
0D01:010C 61        DB      61
0D01:010D 7461      JZ      0170
0D01:010F 2028      AND     [BX+SI],CH
0D01:0111 253129      AND     AX,2931
0D01:0114 3A20      CMP     AH,[BX+SI]
0D01:0116 154C27      ADC     AX,274C
0D01:0119 6F        DB      6F
0D01:011A 7261      JB      017D
0D01:011C 20636F      AND     [BP+DI+6F],AH
0D01:011F 7272      JB      0193

```

Programma somma con registri a 16 bit:

Sintassi dell'istruzione:

MUL sorg

Questa istruzione moltiplica AX * sorg e mette il risultato in DX : AX
Cioè:

DX:AX ← AX * sorg

ESEMPIO:

Risolviamo la funzione:

123 * (0F12 +111)

in esadecimale si ha:

123H =291 dec da cui il valore 123H deve essere posto in sorgente nel nostro caso in BX
0F12 H +111 H = 1023 H questo risultato deve essere posto in AX
il risultato del prodotto andrà nel registro DX:AX cioè:
DX:AX ← 0012 : 57C9

```

C:\> Prompt dei comandi - debug c:\pippo\multipl3.com
0D01:0116 154C27      ADC     AX,274C
0D01:0119 6F              DB     6F
0D01:011A 7261           JB     017D
0D01:011C 20636F        AND    [BP+DI+6F],AH
0D01:011F 7272           JB     0193
-q
C:\> debug c:\pippo\multipl3.com
-u 100
0D01:0100 B8120F      MOV    AX,0F12
0D01:0103 BB1101      MOV    BX,0111
0D01:0106 01D8       ADD    AX,BX
0D01:0108 BB2301      MOV    BX,0123
0D01:010B F7E3       MUL    BX
0D01:010D CC          INT    3
0D01:010E 61         DB     61
0D01:010F 2028       AND    [BX+SI],CH
0D01:0111 253129     AND    AX,2931
0D01:0114 3A20       CMP    AH,[BX+SI]
0D01:0116 154C27     ADC    AX,274C
0D01:0119 6F         DB     6F
0D01:011A 7261       JB     017D
0D01:011C 20636F    AND    [BP+DI+6F],AH
0D01:011F 7272       JB     0193

```

```

-u 100
0D01:0100 B8120F      MOV    AX,0F12
0D01:0103 BB1101      MOV    BX,0111
0D01:0106 01D8       ADD    AX,BX
0D01:0108 BB2301      MOV    BX,0123
0D01:010B F7E3       MUL    BX
0D01:010D CC          INT    3

```

DIVISIONI

Divisioni ad 8 bit:

Sintassi dell'istruzione:

DIV SORG

Dividendo : Divisore = Quoziente + Resto

Ax Sorg AL AH

Esempio: 11 : 2 = 8 + 1

Infatti :

11 H = 17 dec

17 dec : 2 dec = 8 dec + 1 dec

Scriviamo il programma :

Appunti di sistemi

-a 100

```
0D01:0100 B81100    MOV  AX,0011
0D01:0103 B302     MOV  BL,02
0D01:0105 F6F3     DIV  BL
0D01:0107 CC       INT  3
```

richiamiamo il programma salvato con L 100 (cioè carica il programma salvato dall'indirizzo 100) e lo visualizziamo passo passo:

-l 100

-r

```
AX=0000 BX=0000 CX=0011 DX=0000 SP=FFFE BP=FFF6 SI=0000 DI=0000
DS=0D01 ES=0D01 SS=0D01 CS=0D01 IP=0100 NV UP DI PL NZ AC PO NC
0D01:0100 B81100    MOV  AX,0011
```

-t

```
AX=0011 BX=0000 CX=0011 DX=0000 SP=FFFE BP=FFF6 SI=0000 DI=0000
DS=0D01 ES=0D01 SS=0D01 CS=0D01 IP=0103 NV UP DI PL NZ AC PO NC
0D01:0103 B302     MOV  BL,02
```

-t

```
AX=0011 BX=0002 CX=0011 DX=0000 SP=FFFE BP=FFF6 SI=0000 DI=0000
DS=0D01 ES=0D01 SS=0D01 CS=0D01 IP=0105 NV UP DI PL NZ AC PO NC
0D01:0105 F6F3     DIV  BL
```

-t

```
AX=0108 BX=0002 CX=0011 DX=0000 SP=FFFE BP=FFF6 SI=0000 DI=0000
DS=0D01 ES=0D01 SS=0D01 CS=0D01 IP=0107 NV UP DI PL NZ AC PO NC
0D01:0107 CC       INT  3
```

-

come si vede nel registro AX abbiamo in AL=08 cioè il quoziente e in AH=01 cioè il resto.