IL DEBUG

Il DEBUG è un'utility di uso frequente dell'MS-DOS e serve per la ricerca e l' eliminazione degli errori nei programmi;
Esso ci consente di visualizzare passo-passo tutte le istruzioni che vengono eseguite tramite il comando T; ci permette di scrivere dei programmi in assembler ed eseguirli; inoltre si possono leggere i registri e vedere i valori delle celle di memoria.

Per iniziare a scrivere con il DEBUG andare sul sistema operativo **MS-DOS.**
Al prompt dell'MS-DOS scrivere DEBUG senza estensione:

c:>DEBUG (invio)
-
(Nota: Il debug risponde con un trattino il quale indica che ora non bisogna inserire comandi del DOS ma comandi conosciuti dal DEBUG ; in altre parole indica che sta aspettando un comando dall'utente)
Se dopo il trattino inseriamo il simbolo ? cioè

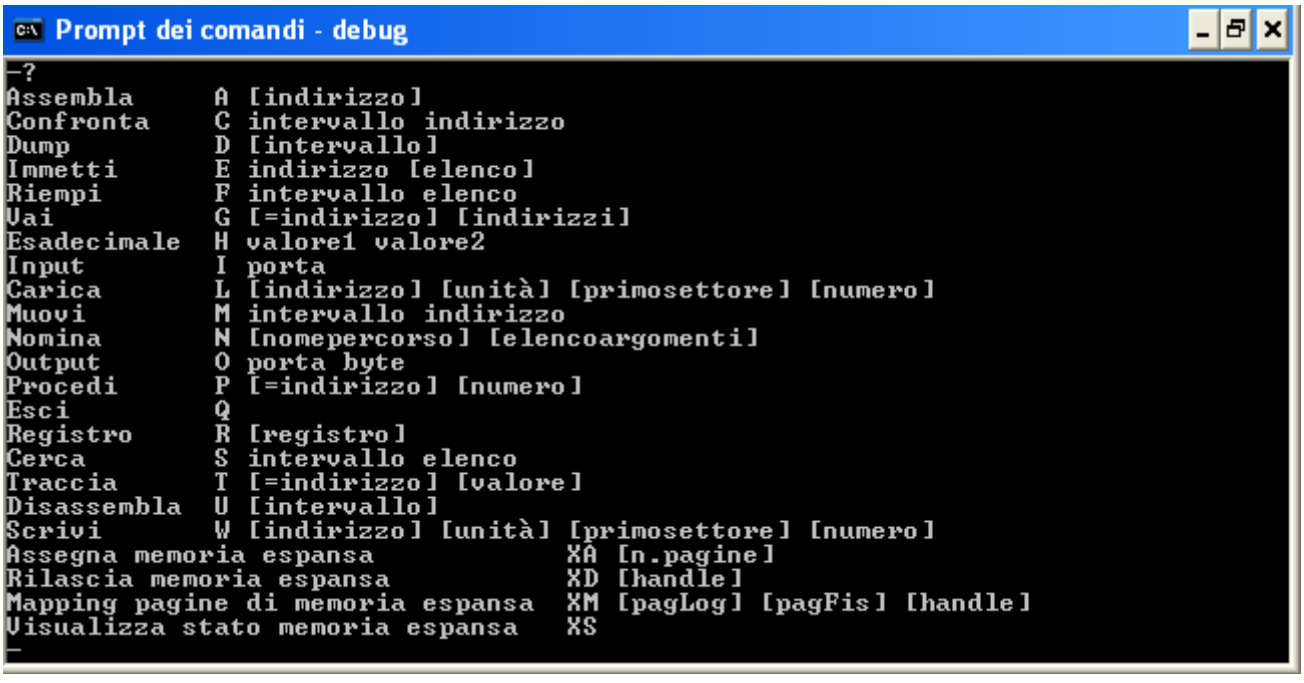
-? (invio)

il sistema ci permette di vedere tutti i comandi del DEBUG

PROMPT DEI COMANDI DEL DEBUG

Dopo essere entrati in debug inserire ? per visualizzare i comandi del debug e la loro sintassi:

-? (invio)



Autore Prof.ssa Malizia Maria Rosa

Comandi del debug:

Assembla	**A** [indirizzo]
Confronta	**C** intervallo indirizzo
Dump	**D** [intervallo]
Immetti	**E** indirizzo [elenco]
Riempi	**F** intervallo elenco
Vai	**G** [=indirizzo] [indirizzi]
Esadecimale	**H** valore1 valore2
Input	**I** porta
Carica	**L** [indirizzo] [unità] [primosettore] [numero]
Muovi	**M** intervallo indirizzo
Nomina	**N** [nomepercorso] [elencoargomenti]
Output	**O** porta byte
Procedi	**P** [=indirizzo] [numero]
Esci	**Q**
Registro	**R** [registro]
Cerca	**S** intervallo elenco
Traccia	**T** [=indirizzo] [valore]
Disassembla	**U** [intervallo]
Scrivi	**W** [indirizzo] [unità] [primosettore] [numero]
Assegna memoria espansa	**XA** [n.pagine]
Rilascia memoria espansa	**XD** [handle]
Mapping pagine di memoria espansa	**XM** [pagLog] [pagFis] [handle]
Visualizza stato memoria espansa	**XS**

-

Visualizzazione dei registri e sua modifica.

IL COMANDO R:

Dopo essere entrati nel debug se dopo il trattino inseriamo il simbolo R cioè

-R **(invio)** il sistema ci permette di vedere i registri con i loro contenuti.

I registri sono piccole aree di memoria; si trovano nella CPU, ma non devono essere confusi con la memoria. Infatti i registri, come la memoria, servono a memorizzare un'informazione, ma sono anche, a differenza della memoria, piccole aree di lavoro privilegiate sulle quali è possibile effettuare operazioni di tipo specifico. Infatti tra tutte le istruzioni che ogni microprocessore è in grado di fare sul contenuto dei registri, non di tutte le istruzioni ne esiste la corrispondente sulla memoria. Da questo segue che per poter eseguire tali istruzioni è necessario prima effettuare il trasferimento dei dati dalla memoria ai registri. Da tutto questo discorso si evince che **le operazioni sui registri sono più veloci di quelle sulla memoria.** Infatti lavorando con i registri le informazioni non devono essere più indirizzate in nessun posto e non devono arrivare all'Unità Centrale attraverso il BUS DATI. A differenza della memoria, in ogni microprocessore il numero dei registri è fisso.
Nel caso **dell'8086/8088 i registri sono 14 ed ognuno ha la lunghezza di una parola cioè di 16 bit**.
Nel DEBUG si usa solo il sistema ESADECIMALE, non c'è quindi la necessità di inserire la lettera H finale, ma se si vuole mettere è indifferente.

Il comando R del DEBUG non si limita a visualizzare i registri, ma se aggiungiamo il nome

del registro, il comando indicherà al DEBUG che desideriamo visualizzare il registro e poi modificarlo.

Esempio con il comando R:

-R CX (invio) permette di vedere i contenuti del registro CX;
mentre
-R AX (invio)
AX 0000 ;risposta del DEBUG
: 3A7 (invio) ; se dopo i due punti scriviamo 3A7 vuol dire che vogliamo modificare AX e che al posto del valore che prima aveva cioè 0000 vogliamo inserire 3A7.
A questo punto per vedere se il DEBUG ha effettuato il cambiamento che noi volevamo, basta digita re di nuovo il comando che ci permette di vedere i registri, cioè R
-R (invio)
avremo
AX 03A7 BX =..... eccetera.

Il comando D

- D (invio) permette di leggere un blocco di memoria; compaiono 8 righe organizzate in tre colonne. Nella colonna di sinistra vi sono due numeri esadecimali separati da **:** che non è un segno di divisione; E' un modo contorto per scrivere gli indirizzi di memoria, le cui motivazioni sono ormai prevalentemente storiche ma che è di uso universale e quindi intoccabile.

CALCOLIAMO L'INDIRIZZO:

ADDR = registro*10_H+OFFSET

L'**indirizzo si calcola così:** si prende il primo numero del registro, lo si moltiplica per 16 decimale (cioè si aggiunge uno zero in fondo a destra: 10_H è esadecimale e corrisponde al numero 16_{10} (shift a sinistra)) e si somma il suo OFFSET.

Esempio:
Calcoliamo l'indirizzo corrispondente a 0E23:100

0E23 è il registro, lo moltiplichiamo per 10_H facendo uno shift verso sinistra gli aggiungiamo uno zero e diventa E230 e a questo gli aggiungiamo l'offset 0100 ottenendo E330
0E23:0100
significa **E230+100=E330**.

Questo è l'indirizzo della prima posizione di RAM considerata.
Nella colonna centrale appaiono, su ogni riga, 16 coppie di cifre esadecimali. Ciascuna coppia corrisponde a un byte e descrive l'attuale contenuto di un particolare byte in memoria. Quello più a sinistra è il contenuto del byte il cui indirizzo è dato, nel modo indicato nella colonna di sinistra; quello successivo è il contenuto del byte successivo (quindi nell'esempio fatto, di indirizzo E331), e così via fino all'ultimo, corrispondente all'indirizzo E33F.
Alla riga successiva, all'estremità sinistra, troviamo l'indirizzo E33F.
Alla riga successiva, all'estremità sinistra, troviamo l'indirizzo E340, e il discorso ricomincia.

Il comando **D** **ci permette quindi di esaminare il contenuto di 8*16=128 byte per volta.**
La colonna di destra contiene in ciascuna 16 caratteri alfanumerici, che costituiscono una rappresentazione figurativa dei corrispondenti byte della colonna centrale, secondo la codifica, detta ASCII, di uso pressoché universale. Se i byte in memoria rappresentano un testo scritto questa codifica ci permette di "leggere" facilmente il testo; se rappresenta un programma o dati numerici codificati in altra forma, essa non serve assolutamente a nulla.
Avendo dato soltanto il comando D il programma DEBUG ha scelto di sua iniziativa un'area di RAM disponibile per scrivere nuovi programmi, ma è possibile fargli visualizzare qualunque area della memoria compresa entro i primi 1048576 byte facendo seguire il **D** dall'indirizzo iniziale (scritto sempre nel solito modo contorto che gli piace tanto).
Provate ad esempio a dare il comando

-D FFFF:0
così facendo leggete un pezzo di ROM. Il suo contenuto è generalmente incomprensibile (è scritto in linguaggio macchina), tuttavia nella prima riga, colonna di destra **leggete una data**: essa è la data in cui è stata scritta la ROM, che serve per identificazione e che è leggibile in codice ASCII.

Diamo adesso il comando
-D 0100 (invio)
permette di leggere un blocco di memoria a partire dall'indirizzo 0100 indicato.

Il comando A

-A (invio) ;converte le istruzioni in codice macchina (A = assembla)

-A 0100 (invio) ;assembla il programma dall'indirizzo 0100

A questo punto si può iniziare a scrivere un programma :

-A 0100 (invio)
MOV AL,10 (invio)
..... ; altre istruzioni
INT 3 ; questa istruzione indica che il programma è terminato interrupt di tipo 3;

L'istruzione INT 3 è un'interruzione di tipo tre e ci permette di tornare al DOS.
Alla fine di ogni programma scritto in assembler bisogna mettere tale interruzione.

Il comando U

-U (invio) **;Il comando U** disassembla le istruzioni, ci permette di vedere l'ultimo indirizzo e possiamo quindi stabilire il numero di celle occupate dal programma.
Nell'esempio precedente supponiamo di avere:
110 INT 3
allora per vedere quanto è lungo il nostro programma dobbiamo fare la differenza tra l'ultima cella occupata dall'istruzione INT 3 e la prima cella del nostro programma che nel nostro esempio iniziava con 0100 cioè
(0110-0100)+1 sono le celle occupate dal nostro programma.

Autore Prof.ssa Malizia Maria Rosa

Per uscire dal DEBUG

-Q (invio) ; indica quit cioè comando per uscire dal DEBUG

STRUTTURA DI UN PROGRAMMA

Quando si scrive un programma con il DEBUG si può iniziare con il comando

-A 0100
……. istruzioni

-int 3

Il comando int 3 (cioè interruzione di tipo 3) deve essere inserito alla fine del nostro programma e consente di ritornare al sistema operativo. Per disassemblare il programma si utilizza

-U (invio) che disassembla le istruzioni e ci permette di vedere l'ultimo indirizzo.

Per stabilire il numero di celle occupate dal nostro programma bisogna vedere l'indirizzo in cui abbiamo inserito il valore di int 3. Ad esempio se il programma è arrivato all'indirizzo

110 int 3

vuol dire che le celle occupate sono

(110-100)+1=11 ove 100 è l'indirizzo di inizio, 110 è l'indirizzo finale, +1 perché è quello attuale.

Per salvare il programma da noi scritto in un floppy-disk si usa la seguente procedura:

N a:\nome.com	**(invio)**	comando per salvare il file nome.com in assembler
R BX	**(invio)**	registro da cui si inizia a salvare il file
0	**(invio)**	comando per azzerare il contenuto di BX
R CX	**(invio)**	registro in cui si termina il salvataggio
11	**(invio)**	numero di celle occupate dal programma nell' esempio 11
w	**(invio)**	significa write e consente di salvare il programma

Per recuperare il programma sul disco:
uscire con il comando:

-Q (invio)

c:\DEBUG a:\nome.com (invio)
-u 100 (invio) ;per visualizzare il programma salvato dalla posizione 100
-r (invio) ;per vedere la prima riga del comando
-t (invio) ; per vedere passo-passo il programma

Autore Prof.ssa Malizia Maria Rosa

Il comando G

G = 100 (invio) G sta per go,100 sta per indirizzo da cui partire

Cioè G=100 è un comando che ci permette di eseguire le istruzioni successive a partire dall'indirizzo dato (100) e le esegue contemporaneamente.

Il comando T

Invece il comando

T=100 (invio) T è un comando per eseguire le istruzioni una per volta a partire dall'indirizzo voluto, ad esempio 100 fino a quando il programma non incontra int 3

LE ISTRUZIONI DELL'8086

Per quando riguarda la tipologia del set di istruzioni dell'8086, esse possono essere funzionalmente suddivise in sei gruppi:

- Istruzioni di **trasferimento**
- Istruzioni **aritmetiche**
- Istruzioni **logiche**
- Istruzioni **di salto**
- Istruzioni **di manipolazioni stringhe**

ISTRUZIONI DI TRASFERIMENTO

Il comando MOV

La sintassi del comando MOV è la seguente:

MOV destinazione, sorgente

questa istruzione indica al programma di prendere il dato rappresentato da "sorgente" e di copiarlo in "destinazione"; destinazione può essere un registro (a 16 o ad 8 bit) o una locazione di memoria; sorgente può essere un registro, una locazione o un dato immediato. Se destinazione è un registrodi segmento allora sorgente non può essere un dato immediato; il registro CS non può mai essere usato come destinazione. Non è possibile spostare il contenuto di un registro di segmento in un altro registro di segmento; se si desidera effettuare questa operazione bisogna utilizzare uno dei registri di uso generale, per esempio AX con due successive istruzioni MOV:

MOV AX, CS (invio) ;copia CS in AX
MOV DS, AX (invio) ;trasferisci il contenuto di AX in DS
Proviamo a fare adesso un piccolo programma che utilizza il comando MOV.

Esempio con l'istruzione MOV

Programma scambia1.com

```
A 100
MOV AL, 01
MOV AH, 02
MOV BL, AL
MOV BH, AH
INT 3
```

Per salvare questo programma di nome scambia1.com in un floppy disk digitare dal programma DEBUG

```
N a:\scambia1.com (invio)
R BX (invio)
BX 0000 (invio)
:0 (invio)
R CX (invio)
: 10 (invio);  In questo modo diciamo quanti byte vogliamo salvare del programma
W (invio); in questo modo il nostro programma è già salvato nel dischetto.
```

Per richiamarlo digitare:

```
c:>debug a:\scambia1.com
```

Esempio con l'istruzione MOV

Programma scambia2.com

Questo esempio serve a far vedere come vengono scambiat i dati da un registro all'altro per cominciare a prendere familiarità con i primi comandi. Proviamo a scambiare i dati dal registro AX al registro BX. Scriviamo:

```
c:>DEBUG (invio)
-A 100
MOV AX, 123              ;mette 0123 in AX
MOV BX, 0                ;mette 0000 in BX
MOV BL, AL               ;mette 23 che si trova in AL e lo mette in BL
MOV BH, AH               ;mette 01 che si trova in AH e lo mette in BH
MOV AX, 0                ;mette 0000 in AX
MOV BX, AX               ; adesso in AX c'è zero quindi mette 0000 in BX
INT 3
```

Per salvare questo programma di nome scambia2.com in un floppy disk digitare dal programma DEBUG

```
N a:\scambia2.com (invio)
R BX (invio)
BX 0000 (invio)
:0 (invio)
R CX (invio)
: 10 (invio);  In questo modo diciamo quanti byte vogliamo salvare del programma
W (invio); in questo modo il nostro programma è già salvato nel dischetto.
```

Per richiamarlo digitare:

c:>debug a:\scambia2.com

Se vogliamo salvarlo in una cartella di nome prova che si trova su c:> prima bisogna creare la cartella prova su c:> con il comando MD
C:>MD prova (invio)
Poi scrivere il programma precedente scambia2.com con il debug e successivamente digitare:
N c:\prova\scambia2.com (invio)
R BX (invio)
BX 0000 (invio)
:0 (invio)
R CX (invio)
: 10 (invio); In questo modo diciamo quanti byte vogliamo salvare del programma
W (invio) ; in questo modo il nostro programma verrà salvato in c: nella cartella prova.

ISTRUZIONI ARITMETICHE

1) L'istruzione somma.

Per eseguire la somma con il DEBUG si usa il simbolo ADD che è l' istruzione somma
Sintassi:

ADD destinazione, sorgente

Questa istruzione fa in modo che venga eseguita la:
somma tra sorgente a destinazione e mette il risultato in destinazione.

destinazione <---- destinazione + sorgente

Esempio:

ADD AL, 10H ; AL←10_H

questa espressione fa in modo che il dato 10_H venga sommato al contenuto del registro AL ed infine riporta il risultato finale della somma nel registro AL.
Destinatario e sorgente devono avere la stessa grandezza, cioè lo stesso numero di bit.

ADD AL,BL ; AL← AL + BL
ADD AX,BX ; AX ← AX + BX
La somma si può eseguire solo fra due registri della stessa capacità.

2) L'istruzione differenza.

Per eseguire la differenza si utilizza il comando SUB che è istruzione della differenza
Sintassi:

SUB destinazione, sorgente

questa istruzione fa in modo che venga eseguita la sottrazione tra il valore contenuto nel registro destinazione e il valore contenuto nel registro sorgente e mette il risultato della differenza nel registro destinazione:

destinazione <--- destinazione - sorgente

Autore Prof.ssa Malizia Maria Rosa

Programma somma

Supponiamo adesso di volere eseguire la somma con il DEBUG della seguente formula matematica:

[8 - (4+1)]

```
A 100 (invio)
MOV Al, 8 (invio)          ;mette 8 in AL
MOV BL, 4 (invio)          ;mette 4 in BL
MOV BH, 1 (invio)          ;mette 1 in BH
ADD BH, BL (invio)         ;somma BL con BH e mette il risultato in BH
SUB AL, BH (invio)         ; sottrai AL con BH e metti il risultato in AL
INT 3 (invio)              ;interruzione software di tipo 3
```

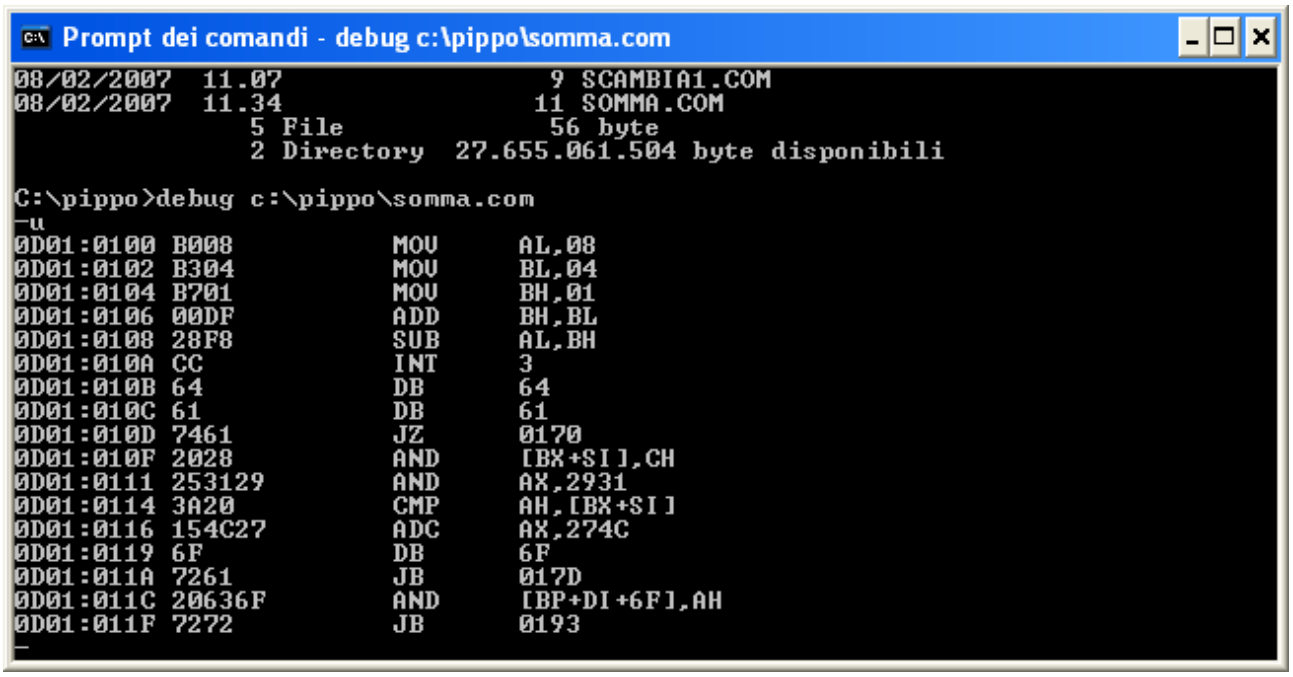
Per salvare il programma con il nome somma.com scriviamo:

```
-N a:\nome. com   ; nel nostro esempio possiamo chiamarlo a:\somma.com
R BX
BX 0000
: 0               ;cioè cancelliamo il valore di BX
R CX
CX 0000
: B               ;numero di celle occupate fino ad INT 3 cioè (A+1)
W                 ; la sigla W sta per Write
```

In questo modo il debug ci dice quanti byte vengono salvati nel nostro dischetto.

Per recuperare il programma salvato sul disco:
uscire con il comando:

```
-Q  (invio)

c:\DEBUG a:\nome.com   (invio)
-u 100 (invio)    ;per visualizzare il programma salvato dalla posizione 100
-r (invio)        ;per vedere la prima riga del comando
-t (invio)        ; per vedere passo-passo il programma
```



Per richiamare il file appena salvato bisogna eseguire la seguente procedura.
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=00**08** 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=00**04** 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=**01**04 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
-
```

3) Istruzione moltiplicazione

Istruzione MUL

L'istruzione MUL esegue la moltiplicazione per sorgente del moltiplicando che si trova in AL se la moltiplicazione è ad 8 bit, oppure esegue la moltiplicazione di sorgente per il moltiplicando che si trova in AX se la moltiplicazione è a 16 bit.
Il risultato del prodotto di sorgente per il moltiplicando (AL o AX) viene trasferito in AX se il moltiplicando è a 8 bit mentre viene trasferito in DX:AX se il moltiplicando è a 16 bit (Word alta in DX e word bassa in AX).
I flag C ed O vengono impostati ad 1 se la metà più significativa del risultato è diversa da zero.
Sintassi dell'istruzione MUL:

MUL sorgente

Moltiplica senza segno interi a 8 bit o 16 bit con il risultato in doppia precisione a 16 o a 32 bit.

ESEMPIO a 8 bit:

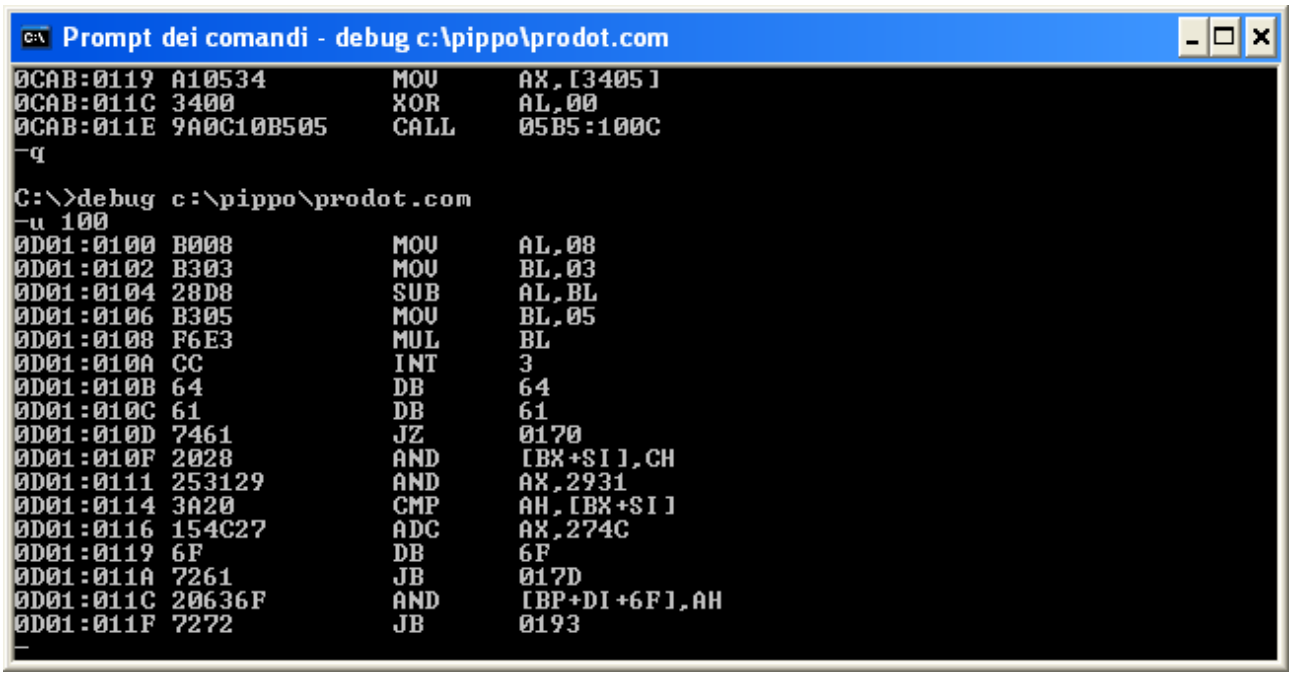
```
MOV AL, 2        ;sposta il numero 02 in AL
MOV BL,3         ;sposta il numero 03 in BL
MUL B3           ; AX ← AL * BL cioè sposta in AX il numero dato dal prodotto di AL per BL
                 ; nel nostro caso   AX = 0006
```

ESEMPIO a 16 bit:

```
MOV AX ,10    ; sposta in AX il numero 0010H
MOV BX, 2     ; sposta in BX il numero 0002H
MUL BX        ;sposta in DX: AX il risultato di AX per BX nel nostro caso DX:AX 0000:0020
```

Esempio di moltiplicazione con registri a 8 bit:
scrivere un programma che risolva la seguente funzione:

5 * (8 - 3)



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:

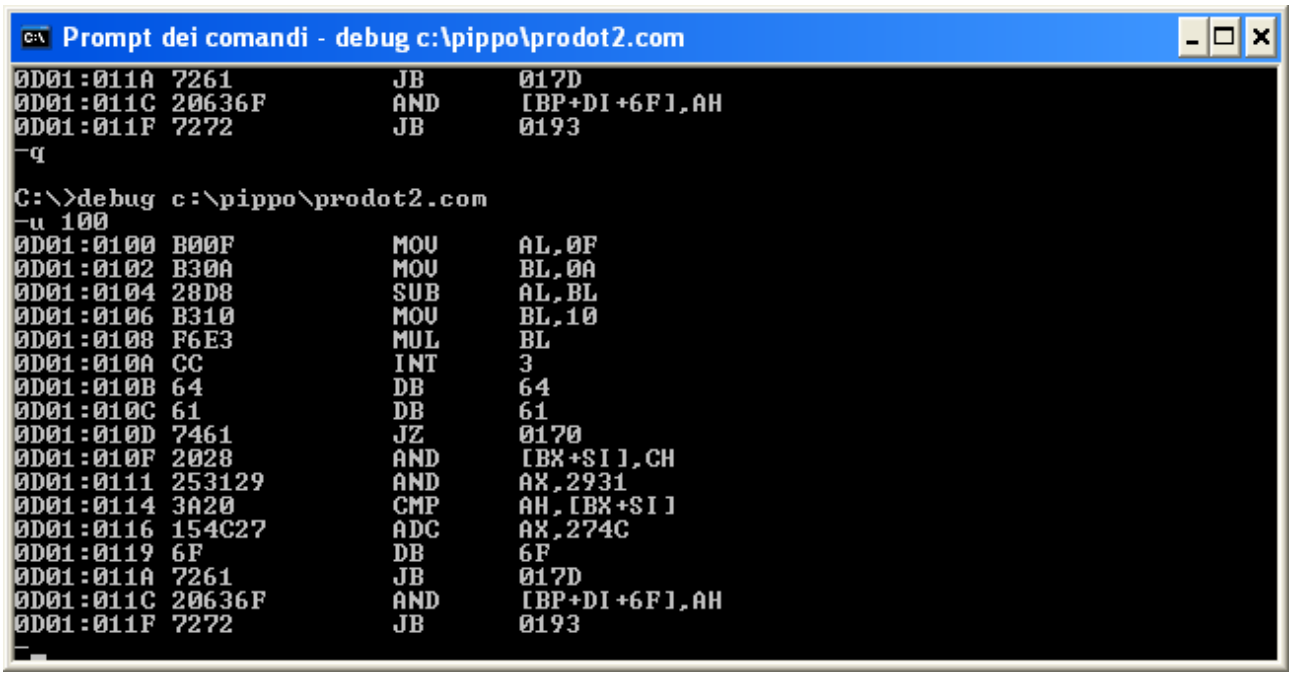
AX ← AL * sorg

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:



Programma con somma e moltiplicazione con registri a 16 bit:

Sintassi dell'istruzione moltiplicazione:

MUL sorg

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

DX :AX ← AX * sorg

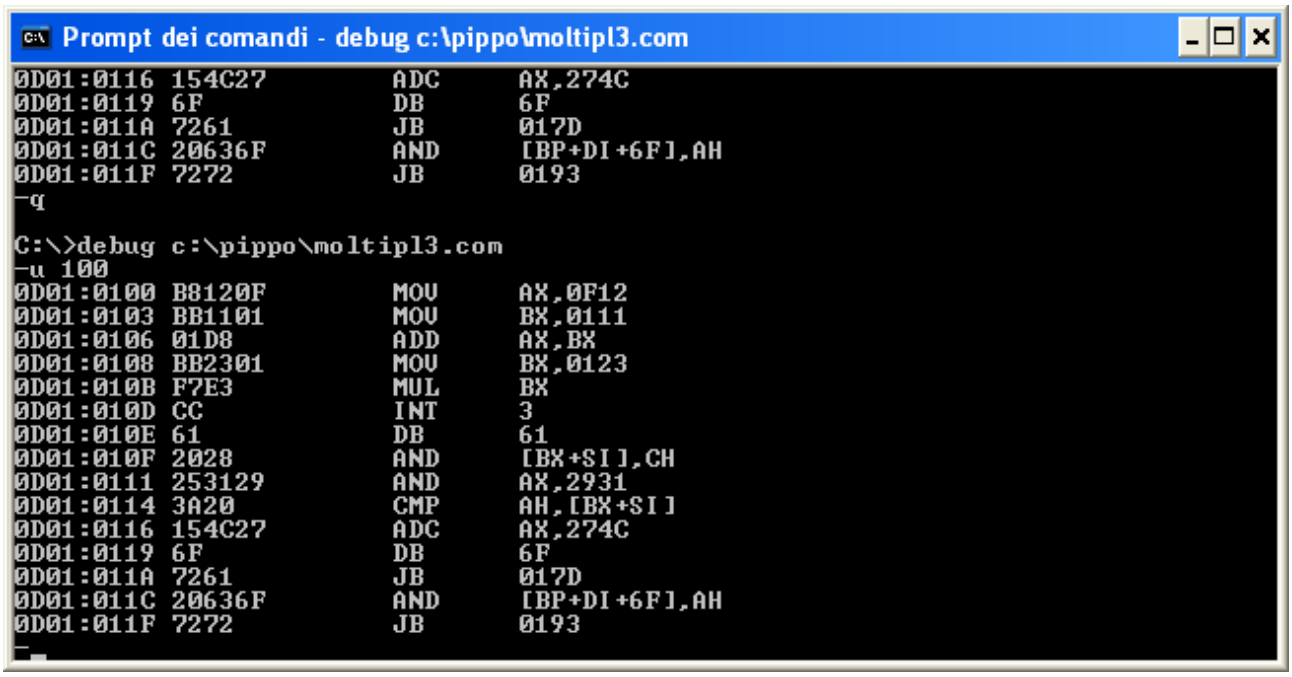
ESEMPIO:

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



-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

4) Istruzione divisione

Istruzione DIV

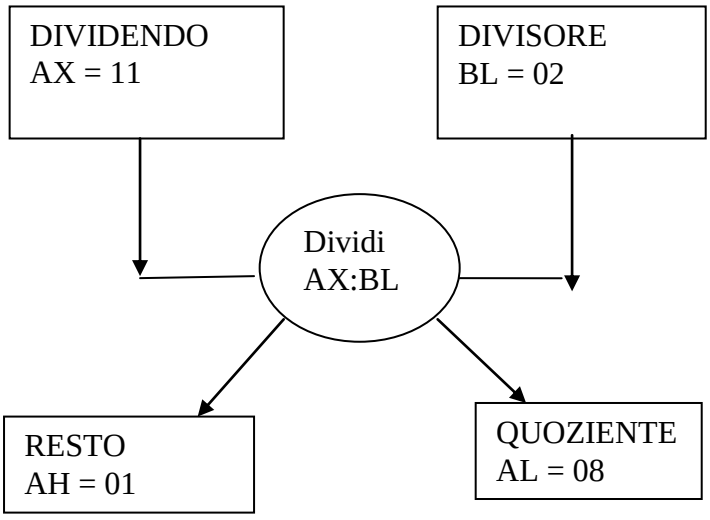
Per eseguire la divisione si segue la seguente sintassi:

DIV sorgente

L'istruzione DIV esegue la divisione senza segno;
il divisore a 8 o a 16 bit è in sorgente; il dividendo di precisione doppia , è in AX o nella coppia di registri DX:AX;
il quoziente va in AL (8bit) o in AX (16 bit) e il resto nei due casi, in AX o in DX.

Esempio di divisione ad 8 bit:

```
MOV AX,11          ; sposta in AX il numero esadecimale 0010H (dividendo)
MOV BL, 2          ; sposta in BL il numero esadecimale 02H (divisore o sorgente)
DIV BL             ; dividi AX con BL e sposta il quoziente in AL e il resto in AH
INT 3
```



DIVISIONI

Divisioni ad 8 bit:

Sintassi dell'istruzione:

DIV SORG

Dividendo : Divisore = RESTO + Quoziente
Ax Sorg AH AL

Esempio: 11 : 2 = 8 + 1

Infatti :
11 H = 17 dec
17 dec : 2 dec = 8 dec + 1 dec

Scriviamo il programma in debug :

```
-a 100
0D01:0100 B81100      MOV    AX,0011
0D01:0103 B302        MOV    BL,02
0D01:0105 F6F3        DIV    BL
0D01:0107 CC          INT    3
```

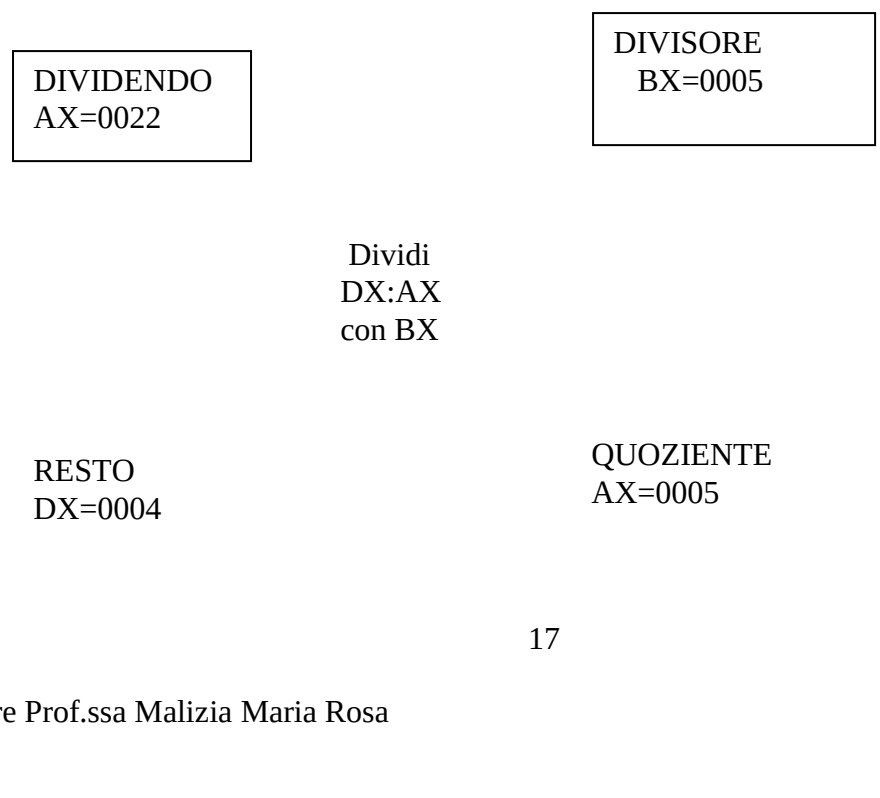
richiamiamo il programma salvato con L 100 (cioè carica il programma salvato dall'indirizzo 100)
e lo visualizziamo passo passo:

```
-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.
La stessa cosa si può fare a 16 bit;

Esempio di divisione a 16 bit:

```
MOV AX,22          ;sposta in AX il numero esadecimale 0022H DX deve essere nullo
                   ; 22H = 2*16+ 2= 34 decimale
MOV BX, 5          ; sposta in BX il numero esadecimale 0005H
DIV BX             ; divide AX=22H con BX=5H il quoziente è AX=6 ed il resto è DX=4
INT 3
```



Infatti

$22_H = 2*16+2 = 34_{10}$
34_{10} diviso 5 fa 6 con il resto di 4.

5) Istruzione incremento

Incrementa di una unità il registro considerato o una variabile di memoria.
Sintassi

INC sorgente

ESEMPIO:

```
A  100
MOV AL,26          ;mette 26 in AL
INC AL             ;incrementa AL, adesso in AL viene posto 26+1=27
ADD AL,76          ;aggiunge 76 a 27 e lo mette in AL
MOV AH, AL         ;sposta il contenuto di AL e lo mette in AH
ADD AL, AH         ;aggiunge AL + AH e il risultato va in AL
INT 3
……
……....
```

6) Istruzione decremento

Decrementa di una unità il registro considerato.
Sintassi

DEC sorgente

ESEMPIO:

```
A 100
MOV AH,10          sposta 10 in  AH
MOV AL, AH         ;sposta AH in  AL
DEC  AL            ;decrementa AL; mette 0F in AL
MOV AX,0
MOV DS, AX
MOV BX,9
MOV AX, [BX]
INC AX             ; incrementa AX
INT 3
……
```

ESERCIZI CON INC, DEC E JNZ

ESERCIZIO:
Realizzare un ciclo che incrementi il registro AL da 0 a 9

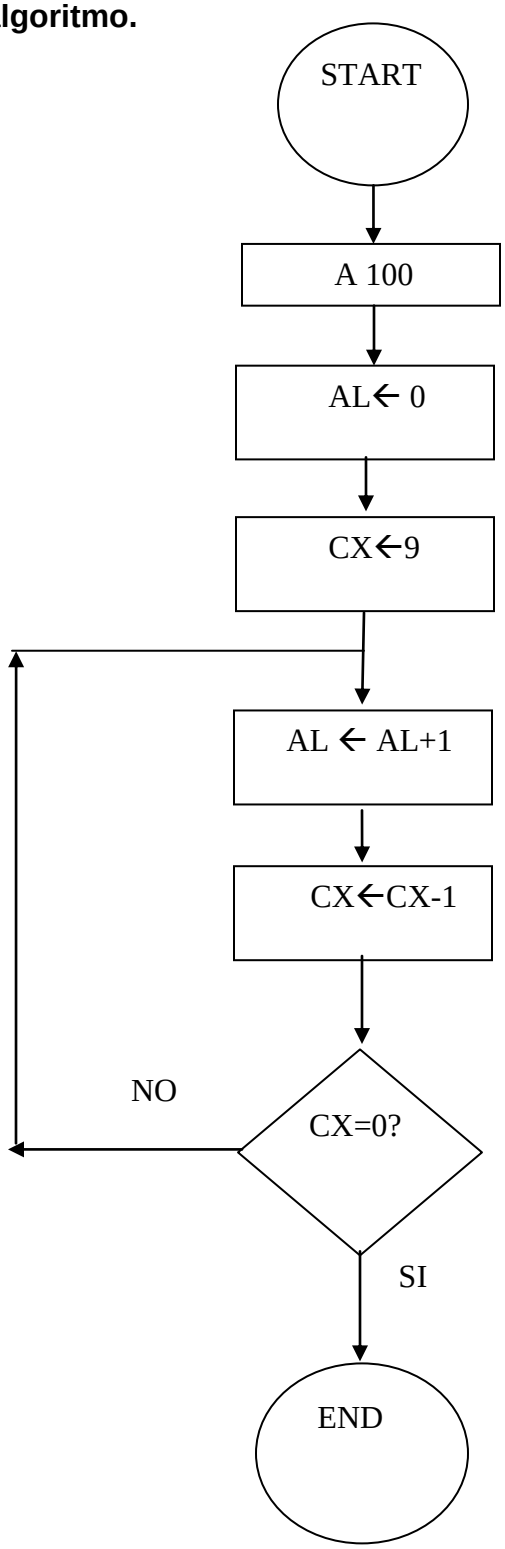
```
C:>DEBUG
                    A 100
       151D:0100   MOV AL,0
       151D:0102   MOV CX,9
(*)    151D:0105    INC AL
       151D:0107   DEC CX,
       151D:0108   JNZ  105     (*)
       151D:010A  INT 3
```

ESERCIZIO:
Realizzare un ciclo che decrementi il registro AL da 9 a 0

```
C:>DEBUG
                    A 100
       151D:0100   MOV AL,9
       151D:0102   MOV CX,9
(*)    151D:0105   DEC AL
       151D:0107   DEC CX,
       151D:0108   JNZ  105     (*)
       151D:010A  INT 3
```

ESERCIZIO:

Disegnare il flow-chart di un ciclo che incrementi il registro AL da 0 a 9 e scrivere il suo algoritmo.



ALGORITMO:
Inizio
Vai all'indirizzo 100
Azzera AX
Inserisci 9 in CX
(*) Incrementa AL
Decrementa CX
Se CX >0
Incrementa AL
Decrementa CX
Altrimenti
vai all'indirizzo 105 (*)
Fine

Esercizio:
Realizzare un **ciclo infinito** che **incrementi** il registro AL con un numero da 0 a 9 .

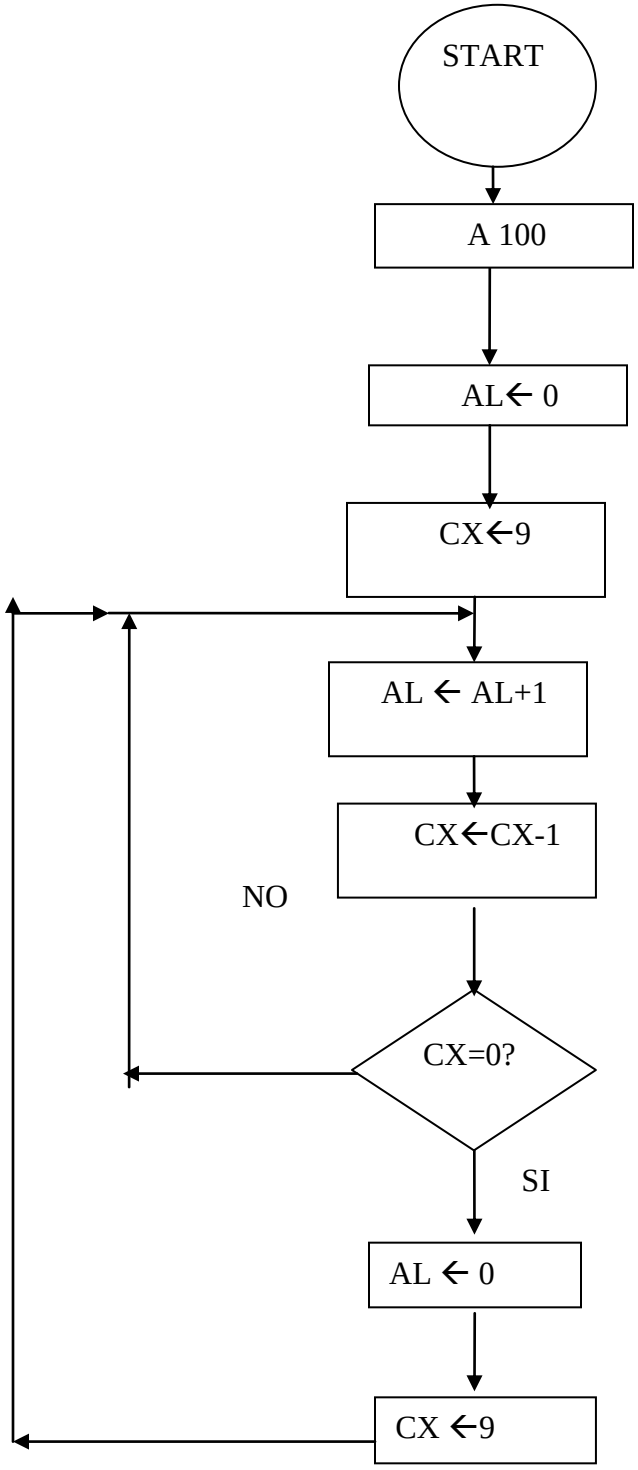
```
   C:>DEBUG
                    A 100
        151D:0100   MOV AL,0
        151D:0102   MOV CX,9
(*)     151D:0105    INC AL
        151D:0107   DEC CX,
        151D:0108   JNZ  105     (*)
        151D:010A   MOV AL,0
        151D:010C  MOV CX,9
        151D:010F   JNZ     (*)
        151D:0111   INT 3
```

Esercizio: Realizzare un **ciclo infinito** che **decrementi** il registro al da 9 a 0.

```
   C:>DEBUG
                    A 100
        151D:0100   MOV AL,9
        151D:0102   MOV CX,9
(*)     151D:0105   DEC  AL
        151D:0107   DEC CX,
        151D:0108   JNZ 105  (*)
        151D:010A   MOV AL,9
        151D:010C  MOV CX,9
        151D:010F   JNZ     (*)
        151D:0111   INT 3
```


Autore Prof.ssa Malizia Maria Rosa

Esercizio:
Disegnare il flow-chart di un **ciclo infinito** che **incrementi** il registro AL con un numero da 0 a 9 e scrivere il suo algoritmo.



ALGORITMO CON LOOP INFINITO
Inizio
Vai all'indirizzo 100
Azzera AX
Inserisci 9 in CX
(*) Incrementa AL
Decrementa CX
Se CX >0
Incrementa AL
Decrementa CX
Altrimenti
vai all'indirizzo 105 (*)
Azzera AL
Inserisci 9 in CX
Vai all'indirizzo 105 (*)

7) Istruzione di confronto

CMP sorg1,sorg2

CMP vuol dire compara; esegue la sottrazione tra sorg1-sorg2 senza modificare gli operandi, ma alterando i flag, il cui valore può essere controllato dalle istruzioni di salto condizionato

UTILIZZIAMO GLI INDIRIZZAMENTI CON REGISTRI:

Modi di indirizzamento dell'8086

Il software dell'8086 prevede diversi modi per indirizzare gli operandi;
L' **indirizzamento diretto** con un offset a 16 bit, oppure l' **indirizzamento indiretto** con base (tramite i registri BX o BP); l'**indirizzamento con indice** (tramite SI o DI) più uno spiazzamento costante opzionale a 8 0 16 bit. questo spiazzamento costante può essere il nome di una variabile o direttamente un numero.

Esempio di indirizzamento:

Scriviamo adesso il semplice programma che chiameremo **a:\indir.com con comando DB**

PROGRAMMA A:\INDIR.COM

```
A 500              ; partiamo dall'indirizzo 500 per inserire i dati
DB 11, 22, 0       ; DB = define byte cioè definiamo i byte per inserire i byte in indirizzi
                   ; successivi
                   ; ad esempio nell'indirizzo 500  mettiamo  11     (F9)
                   ;                  "             501    "        22       (D4)
                   ;                  "             502    "        0        (02)
                   ; cioè F9+D4=1CD ove  CD va in AL mentre 1 è il carry CY
A 100              ; a partire dall'indirizzo 100 inseriamo le istruzioni
MOV AL, [500]      ; indirizzamento diretto: il contenuto dell'indirizzo 500
                   ; nel nostro caso contiene il
                   ;valore 11 il quale viene copiato nel registro AL
ADD AL, [501]      ;aggiunge il valore 22 che si trova nell'indirizo 501 e lo copia in AL cioè
                   ;AL <= 11+22=33, cioè in AL si avrà il valore 33
INT 3              ;fine delle istruzioni del programma.
```

Adesso proviamo a salvarlo nel dischetto:

senza uscire dal debug dopo INT 3 (invio) digitiamo:

```
N a:\nome. com   ; nel nostro esempio possiamo chiamarlo a:\indir.com
R BX
BX 0000
```

: 0 ;cioè cancelliamo il valore di BX
R CX
CX 0000
: B ;numero di celle occupate fino ad INT 3 cioè (A+1)
W ; la sigla W sta per Write

In questo modo il debug ci dice quanti byte vengono salvati nel nostro dischetto.

Per richiamare il file salvato sul dischetto dal prompt di c:> digitare:

C:>debug a:\indir.com
-g 100 ; questo comando ci permette di eseguire le istruzioni a partire dall'indirizzo
; dato cioè dall'indirizzo 100 e le esegue contemporaneamente
; il debug a questo comando risponde così:
AX=0000 BX=00000 CX=010B DX= 0000.....
.....NC.....
MOV AL,[0500] DS:0500=**F9**
-t ; per seguire passo passo il programma digitamo t ed il debug
;ci risponde così
AX=00**F9** BX=0000 CX=010B DX=0000.....
.....NC.....
ADD AL,[0501] DS:0501=**D4**
-t
AX=00**CD** BX=0000 CX=010B DX=0000.....
.....**CY**.....
MOV [0502],AL DS:0502=**CD**
-t
AX=00**CD** BX=0000 CX=010B DX=0000.....
.....CY.....
INT 3

-Q ;per uscire dal debug

Nota:
Se guardiamo il contenuto dell'indirizzo DS:0500=F9 e lo sommiamo con il contenuto dell'indirizzo DS:0501=D4 e spostiamo la somma nel registro AL allora il valore del registro AL sarà la somma dei due valori cioè AL←(F9+D4)= CD con carry ausiliario cioè CY (carry si); questo valore lo spostiamo all'indirizzo DS:0502=CD

ESEMPIO DI INDIRIZZAMENTO:

```
A 100
MOV AX, 01
MOV DS, AX
MOV BX, 09
MOV AX, [BX]          ;indirizzamento indiretto a registro (muove il contenuto del registro BX
MOV AL, BL            ;e lo mette nel registro AX
ADD AX, 2
SUB AX, BX
MOV BX, AX
INT 3
```

ESEMPIO:

Spostiamo a partire dall'indirizzo 500 i byte 11 e 22; sommiamo i due byte spostiamo la loro somma all'indirizzo 502

```
A 500
DB 11,22,0      ; definisce i byte  11,22,0 a partire dall'indirizzo 500
                ; all'indirizzo 500 inserisce il valore 11
                ; all'indirizzo 501 inserisce il valore 22
                ; all'indirizzo 502 inserisce il valore 0
                ; iniziamo il programma
A 100
MOV AL, [500]   ; sposta il contenuto dell'indirizzo 500 cioè il valore 11 in AL
ADD AL, [501]   ; somma il contenuto dell'indirizzo 501 cioè il valore 22 con AL che valeva 11
                ; e trasferisci il risultato della somma cioè 33 in AL
MOV [502], AL   ; sposta all'indirizzo 502 il risultato della somma
INT 3           ; interruzione di tipo 3
```

ISTRUZIONI DI SALTO e di chiamata

CALL label: salva nello stack l'indirizzo di rientro e salta alla procedura (subroutine) di etichetta (nome) label.
L'etichetta può riferirsi ad una subroutine allocata nello stesso segmento codice del programma chiamante o in un altro; nel primo caso (procedura **NEAR)** CALL salva sullo stack solamente il registro IP, nel secondo caso (procedura **FAR**) CALL salva nello stack prima CS e poi IP.

RET: è l'istruzione di ritorno da una subroutine, che provvede a richiamare IP dallo stack (e CS se la chiamata era FAR).
E' anche possibile scrivere
RET n con n intero: dopo il ripristino di IP ed eventualmente CS lo stack pointer SP viene incrementato di 1 esattamente n volte; ciò permette di rimuovere gli eventuali parametri inseriti nello stack dalla procedura chiamante (per il passaggio dei parametri).

JMP label: salto incondizionato a label; anche qui è possibile il salto FAR a un altro segmento e salti con indirizzamento indiretto.

Salti condizionati per numeri con segno

- L = Less ossia minore
- G = Greater cioè maggiore
- E = Equal o uguale

JL o JNGE "minore" o "non maggiore o uguale"
JLE o JNG "minore o uguale" o "non maggiore"
JNL o JGE "non minore" o "maggiore uguale"
JNLE o JG "non minore o uguale" o "maggiore"

Salti condizionati non specifici del tipo di numerazione adottata

JE o JZ	"uguale" o "zero"	Z=1
JNE o JNZ	"non uguale" o non zero	Z=0
JC	"riporto"	C=1
JNC	"non riporto"	C=0
JS	"negativo"	S=1
JNS	"non negativo"	S=0
JO	"overflow"	O=1
JNO	"non overflow"	O=0
JP o JPE	"parità pari"	P=1
JNO o JPO	"parità dispari"	P=0
JCXZ	"CX uguale a zero	CX=0

UTILIZZIAMO GLI INDIRIZZAMENTI CON INDICE:

Esercizio:

Inserire i byte 1, 2,3,4,5,6, all'indirizzo 400 , moltiplicarli per 2 e trasferire il prodotto all'indirizzo 500.
Visualizzare i risultati con i comandi d 400 e d 500

Svolgimento:

```
a 400
db 1,2,3,4,5,6
a 500
db 0,0,0,0,0,0

-a 100
0D00:0100 BE0004      MOV    SI,0400
0D00:0103 BF0005      MOV    DI,0500
0D00:0106 B302        MOV    BL,02
0D00:0108 B90600      MOV    CX,0006
0D00:010B 8A04        MOV    AL,[SI]
0D00:010D F6E3        MUL    BL
0D00:010F 8905        MOV    [DI],AX
0D00:0111 46          INC    SI
0D00:0112 47          INC    DI
0D00:0113 49          DEC    CX
0D00:0114 75F5        JNZ    010B
0D00:0116 CC          INT    3
-r
-t
t……
per visualizzare il risultato utilizzare il comando dump cioè  D    indirizzo
-d 400
0D00:0400  01 02 03 04 05 06 20 67-69 85 20 65 73 69 73 74   ..... gi. esist
0D00:0410  65 6E 74 65 0D 0A 09 25-31 20 62 79 74 65 0D 0A   ente...%1 byte..
0D00:0420  1B 54 6F 74 61 6C 65 20-64 65 69 20 66 69 6C 65   .Totale dei file
0D00:0430  20 65 6C 65 6E 63 61 74-69 3A 0D 0A 38 28 53 69    elencati:..8(Si
0D00:0440  20 8A 20 76 65 72 69 66-69 63 61 74 6F 20 75 6E    . verificato un
0D00:0450  20 65 72 72 6F 72 65 20-6E 65 6C 6C 61 20 76 61    errore nella va
0D00:0460  72 69 61 62 69 6C 65 20-64 27 61 6D 62 69 65 6E   riabile d'ambien
0D00:0470  74 65 29 0D 0A 0D 28 63-6F 6E 74 69 6E 75 61 20   te)...(continua
-d 500
0D00:0500  02 04 06 08 0A 0C 00 6F-6D 65 20 64 69 20 66 69   .......ome di fi
0D00:0510  6C 65 20 6E 6F 6E 20 76-61 6C 69 64 6F 0D 0A 3F   le non valido..?
0D00:0520  49 6D 70 6F 73 73 69 62-69 6C 65 20 61 70 72 69   Impossibile apri
```

```
0D00:0530  72 65 20 69 6C 20 66 69-6C 65 20 64 69 20 69 6E   re il file di in
0D00:0540  66 6F 72 6D 61 7A 69 6F-6E 69 20 64 65 6C 20 50   formazioni del P
0D00:0550  61 65 73 65 20 69 6E 64-69 63 61 74 6F 0D 0A 00   aese indicato...
0D00:0560  95 41 74 74 69 76 61 20-6F 20 64 69 73 61 74 74   .Attiva o disatt
0D00:0570  69 76 61 20 75 6E 20 63-6F 6E 74 72 6F 6C 6C 6F   iva un controllo
-
```

ESERCIZIO DI TRASFERIMENTO DI BYTE:

Dati i seguenti byte
0,1,2,3,4,5,6,7,8,9 all'indirizzo 400h
2,3,4,5,6,7,8,9,a,b all'indirizzo 410h
eseguire la somma e trasferirli all'indirizzo 420h

Svolgimento:

```
-a 400
0D01:0400 db 0,1,2,3,4,5,6,7,8,9
0D01:040A
-a 410
0D01:0410 db 2,3,4,5,6,7,8,9,a,b
0D01:041A
-a 420
0D01:0420 db 0,0,0,0,0,0,0,0,0,0

-A 100
0D01:0100 BE0004      MOV     SI,0400
0D01:0103 BD1004      MOV     BP,0410
0D01:0106 BF2004      MOV     DI,0420
0D01:0109 B90A00      MOV     CX,000A
0D01:010C 8A04        MOV     AL,[SI]
0D01:010E 8A6600      MOV     AH,[BP+00]
0D01:0111 00E0        ADD     AL,AH
0D01:0113 8805        MOV     [DI],AL
0D01:0115 46          INC     SI
0D01:0116 45          INC     BP
0D01:0117 47          INC     DI
0D01:0118 49          DEC     CX
0D01:0119 75F1        JNZ     010C
0D01:011B CC          INT     3
```

alla fine dell'esecuzione del programma con il commando d 400 possiamo visualizzare il trasferimento e la somma dei byte all'indirizzo 420h.

```
- d 400
```

```
0D01:0400  00 01 02 03 04 05 06 07-08 09 62 79 74 65 0D 0A   ..........byte..
0D01:0410  02 03 04 05 06 07 08 09-0A 0B 69 20 66 69 6C 65   ..........i file
0D01:0420  02 04 06 08 0A 0C 0E 10-12 14 0D 0A 38 28 53 69   ............8(Si
0D01:0430  20 8A 20 76 65 72 69 66-69 63 61 74 6F 20 75 6E    . verificato un
0D01:0440  20 65 72 72 6F 72 65 20-6E 65 6C 6C 61 20 76 61    errore nella va
0D01:0450  72 69 61 62 69 6C 65 20-64 27 61 6D 62 69 65 6E   riabile d'ambien
0D01:0460  74 65 29 0D 0A 0D 28 63-6F 6E 74 69 6E 75 61 20   te)...(continua
0D01:0470  25 31 29 0E 52 65 76 69-73 69 6F 6E 65 20 25 31   %1).Revisione %
```