Struttura di un programma Assembly scritto per TASM

```
.MODEL ......

    ; indica al compilatore il modello di memoria da
usare
.STACK ......
    ; dimensiona lo Stack (ad esempio 100h)
.DATA
    ; inizio del segmento dati
.DATA?
    ; inizio del segmento dati
.CONST

.CODE
Etichetta_di_Inizio:  ; inizio del segmento di codice
    ........

    MOV AH,4Ch
    INT 21h       ; fine del programma

END Etichetta_di_Inizio ; fine del programma
```

PSEUDO - ISTRUZIONI

- Le parole in blu sono delle Pseudo-Istruzioni cioè sono comandi per l'assemblatore.

; definisce un commento

.MODEL

Definisce il tipo dei segmenti di memoria da utilizzare nel programma[1]. Le principali scelte possibili sono:

- **SMALL**: è il modello più comune, un segmento per il codice e uno solo per i dati e lo stack. Tutti i segmenti non superano i 64Kb.

- **TINY**: tutto il codice e i dati sono in un unico segmento (stanno in 64Kb).

- **MEDIUM**, **COMPACT**, **LARGE**, **HUGE** …

[1] **Approfondimento: http://www.enricomilano.it/download/guide/c/11c.htm**

.STACK 100h

Dice al compilatore quanto spazio deve riservare per lo stack. Se viene omesso (senza **100h**) il compilatore usa per default 400h (1Kb)

.DATA, .DATA? e .CONST.

Indicano la sezione dei dati suddivisa in 3 categorie:

.DATA

Inizializza il segmento dati. Dopo questa direttiva si dichiarano **le variabili da usare nel programma e le si inizializzano.** (vedi come si dichiarano le variabili)

.DATA?

Questa sezione contiene **i dati NON inizializzati** del vostro programma. A volte capita di voler impegnare una parte di memoria (variabili) senza inizializzarla.

.CONST

Questa sezione contiene **le costanti** usate dal vostro programma. Le costanti in questa sezione non potranno mai essere modificate dal vostro programma. (vedi come si dichiarano le costanti)

Non e' necessario utilizzare tutte e tre le sezioni nel vostro programma. Dichiarate solo la/e sezione/i che volete usare.

.CODE

Indica **l'inizio del segmento di codice del programma**. E' qui dove vengono messe le vostre istruzioni.

Inizia con

```
Etichetta_di_Inizio:
```

e finisce con

```
END Etichetta_di_Inizio
```

Etichetta_di_Inizio e' una qualsiasi etichetta arbitraria esempi START, INIZIO; sono obbligatorie e sono usate per specificare l'estensione del vostro programma. Entrambe le etichette devono essere identiche. Tutto il vostro codice deve risiedere tra **Etichetta_di_Inizio:** e **END Etichetta_di_Inizio.**

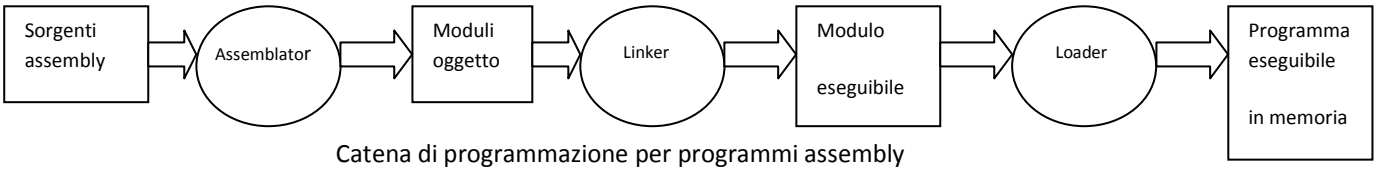
Il **frammento**:

```
MOV AH,4Ch
INT 21h
```

Serve a far terminare correttamente il programma. Ritorno al Sistema Operativo.

1. Catena di programmazione: Assembler, linker, loader

Il linguaggio macchina è costituito da un set di istruzioni codificate in binario.
Per rendere più semplice la programmazione si utilizza uno speciale linguaggio simbolico, denominato **assembly**, le cui istruzioni hanno una codifica più comprensibile per l'essere umano (ad es: MOV per trasferire dati, ADD per sommare, MUL per moltiplicare, ecc.).
Ogni istruzione assembly corrisponde ad una istruzione macchina e viceversa. Poiché ogni tipo di processore ha un proprio linguaggio macchina, esisterà anche un linguaggio assembly per ogni processore. Pertanto si avrà ad esempio un assembly per l'Intel 8086, un assembly per il Motorola, ecc.
I linguaggi assembly sono detti anche **linguaggi a basso livello**, in quanto sono più vicini al linguaggio della macchina. Viceversa i linguaggi più vicini all'utente umano vengono detti **linguaggi ad alto livello**.
Ogni programma scritto in assembly può essere eseguito dal processore solo dopo essere stato tradotto in linguaggio macchina e successivamente caricato in memoria centrale. La traduzione viene effettuata da un apposito programma detto **assemblatore** (assembler). Tale programma traduce i codici mnemonici utilizzati nell'assembly nei corrispondenti codici macchina, traduce i nomi simbolici di variabili in indirizzi di memoria e genera il cosiddetto **codice oggetto**.
Successivamente un altro programma detto **collegatore** (linker) collega tra di loro i diversi moduli oggetto che costituiscono il programma e collega gli indirizzi di memoria dei diversi moduli. Anche quando il programma è costituito da un solo modulo è comunque necessaria l'operazione di collegamento. Il risultato ottenuto è il programma eseguibile vero e proprio (in DOS e Windows si tratta di file caratterizzati dall'estensione ".exe"). Tale programma viene infine caricato in memoria da un ulteriore modulo di sistema detto **caricatore** (loader).



Catena di programmazione per programmi assembly

2. Istruzioni Assembly

La maggior parte delle istruzioni sono identificate da un codice mnemonico di tre caratteri e sono seguite da uno o più operandi separati da virgole.

Esempio: MOV destinazione, sorgente

Queste istruzioni sono tradotte in linguaggio macchina

- Istruzione MOV

L'istruzione **MOV** copia un dato da una posizione ad un'altra e la sua sintassi è:

MOV <destinazione>, <sorgente>

In pratica, questa istruzione copia il valore sorgente in destinazione.

- L'operando **<destinazione>** può essere un registro o una locazione di memoria,
- l'operando **<sorgente>** può essere un registro, una locazione di memoria o un valore immediato;
- i due operandi devono avere la stessa dimensione.
- i due operandi non possono essere entrambi locazioni di memoria.

DICHIARAZIONE DI VARIABILI

```
;programma che inserisce un valore presente nel resgistro BX nel registro AX
;per visualizzare il risultato viene utilizzato il turbo debug TD
    .MODEL SMALL
    .STACK
    .DATA
          num1 DB 5H ; dichiarazione variabile
    .CODE
BEGIN:

          MOV AX,@data  ; @data è l'indirizzo del segmento dati
          MOV DS,AX       ; DS:ax

         MOV AX,4C00H   ;funzione DOS Exit
         INT 21H
END BEGIN
```

```
.DATA

    Nome_variabile DB valore
```

Il frammento:

```
MOV AX, @data
MOV DS,AX
```

Serve per "caricare" la variabile nel segmento dati DS.

I tipi che una variabile può assumere sono quattro e ne definiscono la massima grandezza. Essi sono:
- DB (define byte): la variabile è grande 1 byte;
- DW (define word): la variabile è grande 2 byte;
- DD (define double): la variabile è grande 4 byte;

- DQ (define quad): la variabile è grande 8 byte
- DT (define ten): la variabile è grande 10 byte

INT 21h E LE FUNZIONI MS-DOS

Il Sistema Operativo MS-DOS offre al programmatore assembly un insieme di *funzioni* che permettono di eseguire le più comuni operazioni di gestione del sistema, ad esempio: operazioni di I/O (da tastiera, schermo), allocazione e deallocazione di aree di memoria.

Inoltre, per garantire la portabilità dei programmi su tutte le macchine dotate di MS-DOS prescindendo dall'hardware usato, ogni costruttore di hardware fornisce uno strato di software di interfaccia hw-sw. Questo è detto BIOS *(Basic Input Output System)* e permette la gestione a basso livello di: video (modalità grafiche, colori, palette, accensione pixels,...); tastiera, mouse (codice tasti, stato degli shift,...); stampante; dischi (cilindri, tracce, motore,...); porte seriali e parallele.

Accesso alle funzioni DOS e BIOS

Le funzioni **DOS** sono richiamabili mediante l'istruzione **INT 21h**, e si distinguono tra loro dal valore presente nel registro AH al momento della chiamata.

L'insieme delle funzioni disponibili varia in funzione della versione del DOS usata.

Le funzioni **BIOS** sono richiamabili mediante l'invocazione di un particolare Interrupt con l'istruzione **INT**; molte di esse possono fornire diversi servizi che si distinguono dal valore presente nel registro AH al momento della chiamata.
Esempio:

```
MOV    AH, 07h          ;Funzione DOS 'Read Keyboard
INT    21h              ; Without Echo'
MOV    AH, 0Ah          ;Servizio BIOS 'Write Char
INT    10h              ; at Cursor'
```