La codifica Assembly

Docente: Prof. Mario Perna **Email:** prof.perna.mario@darzo.net **Revisione:** 04.01.2025

Il Linguaggio Assembly: sintassi

```
.MODEL SMALL
.STACK 100H
.DATA
    N DB 7          ;NUMERO ELEMENTI DI UN VETTORE
    VAL DB 4        ;VALORE DI OP1 DA CONFRONTARE CON TUTTI GLI ELEMENTI DI VECT
    VECT DB 1,2,3,4,5,6,7 ;I VALORI DI VECT
.CODE
.STARTUP
        MOV AX, @DATA
        MOV DS, AX



        MOV DX,0 ;RISULTATO CHE VA MESSO IN R8
        MOV CL, N ; CONTATORE

        MOV BH, 0 ; DETECTOR
        MOV BL, VAL ; VALORE DI OP1
        MOV SI, OFFSET VECT           ;MOV DX, OFFSET VECT

CICLO:  MOV AL,[SI]             ;INDIRIZZAMENTO INDIRETTO DI REGISTRO
        CMP BL,AL
        JE TROV
        ADD SI,1
        SUB CL,1
        CMP CL,BH ; CONFRONTO COL DETECTOR
        JE NOTROV  ;SCANSIONATO TUTTO NON TROVATO
        JMP CICLO;ALTRIMENTI

TROV:  MOV DH,1 ; METTO IN R8 1
NOTROV: NOP

        MOV AH, 4CH
        INT 21H         ;RESTITUISCO IL CONTROLLO AL DOS
END                     ;TERMINO IL PROGRAMMA
```

Il Linguaggio Assembly: sintassi

TITLE

Direttiva che definisce il nome del programma.

Il nome del file non deve superare gli 8 caratteri di lunghezza!

Il Linguaggio Assembly: sintassi

.MODEL

Direttiva che definisce il tipo di segmento di memoria da utilizzare nel programma.

- **TYNY**: tutto il codice e i dati sono in un unico segmento (stanno in 64Kb);
- **SMALL**: modello più comune, suddiviso in: un segmento per il codice, uno per i dati e lo stack tutti non oltre i 64Kb;
- **MEDIUM**: il codice usa più segmenti, può quindi superare la barriera dei 64Kb. I dati e lo stack sono organizzati come nel modello **small**.

Il Linguaggio Assembly: sintassi

.MODEL

Direttiva che definisce il tipo di segmento di memoria da utilizzare nel programma.

- **COMPACT**: identico a **small** ma per accedere ai dati si utilizzano i puntatori di tipo **FAR**, quest'ultimi possono infatti superare i 64Kb.
- **LARGE**: è come il **compact** ma con il codice in più segmenti; sia il codice che dati superano i 64Kb.

Il Linguaggio Assembly: sintassi

.STACK 200h

Direttiva che comunica al compilatore quanto spazio deve riservare per lo stack. Punta al registro **SS (Stack Segment)**

In caso di mancanza del valore 200h, il compilatore userà di default 200h (1Kb)

Il Linguaggio Assembly: sintassi

.CONST

Questa sezione contiene le costanti usate dal programma.

Come da definizione classica, le costanti sono tipi di dati che non mutano durante l'esecuzione del programma

In caso di omissione del valore 200h, il compilatore usa di default 200h (1Kb)

Il Linguaggio Assembly: sintassi

.DATA

Direttiva che inizializza il segmento dati. Punta al registro **DS (Data Segment)**

Questa direttiva serve per dichiarare le variabili da usare nel programma

Il Linguaggio Assembly: sintassi

.CODE

Direttiva che indica la sezione dove verranno scritte le istruzioni dell'algoritmo.

Il Linguaggio Assembly: sintassi

.STARTUP

Direttiva che determina i valori per **DS (Data Segment)** e **ES (Extra Segment)**. Indica il punto di partenza delle istruzioni del programma.

Il Linguaggio Assembly: sintassi

END

Direttiva che determina la fine del programma.

Prima di invocare questa direttiva **bisogna richiamare 2 istruzioni fondamentali**:

```
MOV AH, 4CH
INT 21H
```

Restituiamo al PC il controllo del programma, dunque il programma termina.

Il Linguaggio Assembly: BIOS / DOS interrupt call

Le interruzioni messe a disposizione dal BIOS e dal DOS sono moltissime e ognuna di queste comprende moltissime funzioni.

Si possono leggere e stampare caratteri, si può accedere ai file sull'hard-disk, si può gestire la stampante, lo scanner, la penna ottica, il mouse, si può leggere e modificare l'orologio di sistema, emettere suoni, realizzare grafici e disegni sul monitor.

Il Linguaggio Assembly: BIOS / DOS interrupt call

Per avere l'elenco completo delle interruzioni fate riferimento a qualche manuale di Assembler. Qui di seguito ne elenco solo alcune del DOS:

20h - Program terminate

21h - Function request

22h -Terminate address

23h - Ctrl-break exit address

24h - Critical error handler vector

25h - Absolute disk read

26h - Absolute disk write

27h - Terminate but stay resident

2Fh - Printer

Il Linguaggio Assembly: tipi di variabili

Tipi di variabili:

- **DB**: (Define Byte) pesa **8 bit**, ovvero **1 byte**
- **DW**: (Define Word) pesa **16 bit**, ovvero **2 byte**
- **DD**: (Define Double Word) pesa **32 bit**, ovvero **4 byte**
- **DQ**: (Define Quad) pesa **64 bit**, ovvero **8 byte**
- **DT**: (Define Ten) pesa **80 bit**, ovvero **10 byte**

Il Linguaggio Assembly: valori consentiti per tipologia di variabili

Tipi di variabili:

- **DB**: range 0-255
- **DW**: range 0 - 65.535
- **DD**: range 0 - 4.294.697.295
- **DQ**: range 0 - 18.446.744.073.709.551.615
- **DT**: range 0 - 1.2089258e+24

Il Linguaggio Assembly: sintassi per dichiarare una variabile

- La sintassi di dichiarazione di una variabile scalare è la seguente:

<identificatore> <tipo> <valore_iniziale>

Dove **valore_iniziale** può assumere:

- Un valore numerico;
- Una stringa di caratteri definita **tra apici singoli**
- Il carattere **?** (indica un valore nullo)
- Una espressione (operatori aritmetici, logici, relazionali: +, -, AND, OR, …)

Il Linguaggio Assembly: sintassi per dichiarare una variabile (esempi)

```
VALORE   DW ? ;var word non inizializzata
NUMERO1 DB 6 ;var byte inizializzata a 6
NUMERO2 DW 3 ;var word inizializzata a 3
F DB 110      ;var byte inizializzata a 110
MAX DW FFFFh ;var word inizializzata a 65535
CONFERMA DB 'Y' ;var byte inizializzata a 089
ANNULLA DB 'N' ; var byte inizializzata a 078
TA DB '@'     ;var byte inizializzata a 064
```

Le variabili possono essere scritte in qualsiasi punto del programma, ma <u>si preferisce farlo all'inizio</u> o alla fine dello stesso.

Il Linguaggio Assembly: le costanti

Le costanti sono nomi assegnati a valori, non hanno indirizzo e di conseguenza non compaiono nel codice oggetto ed eseguibile. Il valore di una costante non può essere modificato.

Per la dichiarazione di una costante si usa la direttiva EQU

Una costante, in Assembly 8086, può essere inizializzata con:

- Un valore numerico;
- Una stringa di caratteri tra apici
- Una espressione (operatori aritmetici, logici, relazionali: +, -, AND, OR, …)

Il Linguaggio Assembly: operazioni

MOV [DESTINAZIONE] [SORGENTE]

serve per copiare dati da un'operazione sorgente a una destinazione.

DESTINAZIONE E SORGENTE possono essere:
Registri, Valori immediati, Variabili ed indirizzi di memoria

Il Linguaggio Assembly: operazioni

ADD [DESTINAZIONE] [SORGENTE]

serve per sommare due valori e memorizzare il risultato nella destinazione.

DESTINAZIONE E SORGENTE possono essere:
Registri, Valori immediati, Variabili ed indirizzi di memoria

Il Linguaggio Assembly: operazioni

SUB [DESTINAZIONE] [SORGENTE]

serve per sottrarre due valori e memorizzare il risultato nella destinazione.

DESTINAZIONE E SORGENTE possono essere:
Registri, Valori immediati, Variabili ed indirizzi di memoria

Il Linguaggio Assembly: operazioni

MUL <OPERANDO>

Serve per moltiplicare due valori e memorizzare il risultato nella destinazione. Inoltre:

- assume implicitamente che il primo operando sia nel registro **AX o AL**.
- il secondo operando può essere un registro o una locazione di memoria.

```
Moltiplicazione a 8 bit (AL × operando)
mov AL, 5

mov BL, 10

MUL BL        ; AX = AL * BL = 5 * 10 = 50
```

Il Linguaggio Assembly: operazioni

DIV <OPERANDO>

serve per dividere due valori e memorizzare il risultato nella destinazione.

Per operandi a 8 bit (divisore 8 bit)

- Il dividendo è memorizzato nei registri **AX**.
  - **Parte bassa (AL)**: contiene il byte meno significativo del dividendo.
  - **Parte alta (AH)**: contiene il byte più significativo del dividendo.

```
mov AX, 30        ; Dividendo (AX = 30)
mov BL, 5         ; Divisore (BL = 5)
DIV BL            ; AX / BL → AL = quoziente, AH = resto
```